

OBSTACLE AVOIDANCE FRAMEWORK BASED ON REACH SETS

by

Alojz Gomola

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Departamento de Matemática
University of Porto

Dedication

This work is a commutation of a five-year-long journey. Many sacrifices were made along the way, at least now you may hold the summary of it. The book in your hands is applicable in many areas of the land/sea/air transportation. Please proceed with care and apply the knowledge to empower humankind.

I *dedicate* this work to anyone who is seeking knowledge. I would be glad if it can help you to find a missing piece in the jigsaw of science. You can expect a detailed cookbook with many useful ideas which needs to reach maturity.

The best is yet to come in the field of autonomous systems; the full autonomy is in our grasp. To get there its good to know what are the limits of your maneuverability in a given situation. This work offers you that.

Feel free to *reach* it!

Acknowledgements

This thesis was developed under *MarineUAS* - Innovative Training Network on Autonomous Unmanned Aerial Systems for Marine and Coastal Monitoring.

This project has received funding from the European Union's Horizon 2020 research and innovation programme, under the Marie Skłodowska-Curie grant agreement No. 642153.

Acknowledgements: The author acknowledges the support received from following parties and organizations/scientists:

- *Laboratório de Sistemas e Tecnologias Subaquáticas* (LSTS FEUP):
 - João Tasso de Figueiredo Borges de Sousa (supervisor)
- Departamento de Engenharia Eletrotécnica e de Computadores (DEEC FEUP):
 - Fernando Manuel Ferreira Lobo Pereira (co-supervisor)
 - António Pedro Rodrigues Aguiar (lecturer)
- *Unmanned Aerial Vehicles Laboratory* (UAV-Lab ITK NTNU):
 - Tor Arne Johansen (project manager)
 - Kristian Klausen (researcher)
- *Department of Electrical Engineering* (ISY LIU):
 - Martin Enqvist (professor)
 - Gustaf Hendeby (associated professor)
- *Honeywell International* (HISRO):
 - Tomáš Kábřt (Technical Manager)
 - Václav Mareš (Research & Development)
 - Milan Hrusecky (Supervisor)

Abstract

This work addresses an issue of *event-based/reactive obstacle avoidance* for *Unmanned Autonomous Systems* (UAS) operating in non-segregated airspace.

The *UAS* is controlled through *movement automaton*; this enables trajectory discretization and *control independence*. The movement automaton acts as an *interface* consuming movement *command chain* to control UAS or generate a reference trajectory for low-level control.

The *sensor readings* and *information sources* are fused through rating-based *data fusion*; this provides *sensor-platform independence*. The situational assessment is projected into operational space.

The UAS *operational space* is represented as a *planar grid*; this is separated into non-uniform cells. The *threats* are tracked for each cell, namely obstacles or intruders presence, geo-fencing or weather impact.

The *avoidance* or *navigation strategy* of UAS is represented as a *reach set* in operational space. The *reach set* is approximated as a tree where the root is initial system state; the nodes are expected states after movements application. The reach set is calculated for a range of initial states prior the flight, giving a low computational footprint, enabling approach implementation on embedded platforms.

The reach set approximation can include various *maneuvering properties*, like *high space coverage* or *trajectory smoothness*, for avoidance or navigation tasks. The *customization* is used to integrate UAS into *controlled airspace*, where *separation requirements* are included in *reach set*.

The basic services of *UAS Traffic Management* like position notification, airspace restriction, directives, and micromanagement are implemented to prove the operational feasibility of approach in controlled airspace.

The *verification of approach feasibility* was proven through *border-line case test scenarios* taken from general aviation practices and experience. The complete simulation environment with wide customization options is presented.

Resumo

Esse trabalho aborda o *problema de desviar de obstáculos baseado em eventos e de forma reactiva* para um sistema autónomo não tripulado que opera em *espaço aéreo não segregado*. O sistema é controlado através de um autómato de movimento, dessa forma é possível *discretizar a trajetória e controlar o veículo de forma independente*. O autómato actua como uma interface para controlar o sistema autónomo e gerar referências de trajetórias para um controlador de baixo nível.

As leituras do sensor e outras fontes de informações são combinadas através de uma técnica de fusão de dados baseado em escala, dessa forma o método é independente da plataforma. A avaliação situacional é projetada no espaço operacional.

O espaço operacional do sistema autónomo é representado em uma grelha planar, separada em células não uniformes. Os riscos são rastreados para cada célula, nomeadamente obstáculos e a presença de intrusos, geo-fencing ou distúrbios atmosféricos.

A *estratégia de evasão ou navegação* do sistema autônomo é representada como um conjunto alcançável no espaço operacional. O conjunto alcançável é aproximado como uma árvore na qual a raiz representa o estado inicial do sistema e os nós são os estados esperados após aplicar os movimentos. O conjunto alcançável é calculado para um conjunto de estados iniciais antes da execução da missão. Devido a baixa carga computacional, é possível implementar a estratégia em plataformas embarcadas.

A *aproximação do conjunto alcançável* inclui diversas propriedades de manobra, como grande cobertura de regiões ou suavidade de trajetórias, para tarefas de navegação e evasão. A customização é utilizada para integrar o sistema autónomo no espaço de controlo aéreo, onde os requisitos de separação estão incluídos no conjunto alcançável.

Os *serviços básicos do sistema de controlo de tráfego aéreo* como posição, como notificação da posição, restrição do espaço aéreo, diretivas e administração local são implementadas para provar a possibilidade da estratégia ser implementada no espaço de controlo aéreo.

A verificação da *estratégia foi comprovada através* de cenários chaves obtidos de exercícios de aviação e experiências. O ambiente de simulação completo com uma vasta gama de opções de customização é apresentado.

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 1 |
| 1.1 | Motivation | 1 |
| 1.2 | Goals | 4 |
| 1.3 | Overview | 5 |
| 1.4 | Contributions | 6 |
| 1.5 | List of Publications | 8 |
| 2 | Collision Avoidance | 13 |
| 2.1 | Overview | 14 |
| 2.2 | Airspace Classification | 16 |
| 2.3 | Aircraft Operational Rules | 21 |
| 2.3.1 | Visual Flight Rules | 22 |
| 2.3.2 | Instrumental Flight Rules | 23 |
| 2.4 | Separation from Air Traffic | 24 |
| 2.4.1 | Air Traffic Control | 26 |
| 2.4.2 | Airborne Collision Avoidance System X | 28 |
| 2.4.3 | Traffic Collision Avoidance System | 32 |
| 2.5 | UAS Traffic Management (UTM) | 32 |
| 2.5.1 | U-Space | 39 |
| 2.5.2 | NASA UTM | 45 |
| 2.6 | Event-Based Avoidance | 46 |
| 2.6.1 | Mid-Air Collision Prevention | 49 |
| 2.6.2 | Weather Impact | 50 |
| 3 | Background Theory | 53 |
| 3.1 | UAS System Model | 54 |
| 3.1.1 | Continuous-time Systems | 54 |
| 3.1.2 | Discrete-time Systems | 54 |
| 3.1.3 | Adversarial Behavior in Continuous-time Systems | 55 |
| 3.2 | Reach Sets | 55 |
| 3.2.1 | Definitions | 55 |
| 3.2.2 | Computation of Reach Sets | 56 |
| 3.3 | Hybrid Automaton | 57 |

| | | |
|----------|---|-----------|
| 3.4 | LiDAR | 58 |
| 4 | Problem Statement | 61 |
| 4.1 | Basic Definitions | 61 |
| 4.1.1 | World | 61 |
| 4.1.2 | Mission | 62 |
| 4.1.3 | Flight Constraints | 62 |
| 4.1.4 | Partition of Space | 63 |
| 4.1.5 | Information Source | 63 |
| 4.1.6 | Single Observation by Single Sensor Classification | 64 |
| 4.1.7 | Multiple Observations by Single Sensor Classification | 64 |
| 4.1.8 | Sensor Fusion | 65 |
| 4.1.9 | Data Fusion | 65 |
| 4.1.10 | Known World | 66 |
| 4.1.11 | Safety Margin | 66 |
| 4.2 | Basic Obstacle Avoidance Problem | 67 |
| 4.3 | Initial Assumptions | 68 |
| 4.4 | Incremental Problem Definition | 69 |
| 4.5 | Avoidance Requirements | 72 |
| 4.5.1 | Energy Efficiency | 73 |
| 4.5.2 | Trajectory Tracking | 74 |
| 4.5.3 | Safety | 77 |
| 4.6 | Navigation Requirements | 79 |
| 5 | State of the Art | 81 |
| 5.1 | Movement Control | 81 |
| 5.2 | Surveillance | 83 |
| 5.3 | Navigation Algorithms | 84 |
| 5.4 | Reach Set Estimation | 86 |
| 5.5 | Software Testing | 87 |
| 5.6 | UTM Services | 89 |
| 6 | Approach | 93 |
| 6.1 | Overview | 94 |
| 6.2 | UAS Model and Control | 97 |
| 6.2.1 | Movement Automaton Applications | 97 |
| 6.2.2 | UAS Model | 97 |
| 6.2.3 | UAS Movement Automaton | 99 |
| 6.3 | Space Discretization - Avoidance Grid | 103 |
| 6.4 | Reach Set Approximation | 109 |
| 6.4.1 | Trajectory Set Approximation | 110 |
| 6.4.2 | Distinctive Properties of the Trajectories | 112 |

| | | |
|----------|--|------------|
| 6.4.3 | Heuristic Trajectory Tree Building | 115 |
| 6.4.4 | Coverage-Maximizing Reach Set Approximation | 118 |
| 6.4.5 | Turn-Minimizing Reach Set Approximation | 122 |
| 6.4.6 | ACAS-X like Reach Set Approximation | 126 |
| 6.4.7 | Combined Reach Set Approximation - Tree Merge | 129 |
| 6.5 | Situation Representation in the Avoidance Grid | 135 |
| 6.5.1 | Obstacles | 135 |
| 6.5.2 | Intruders | 145 |
| 6.5.3 | Constraints | 149 |
| 6.5.4 | Data fusion | 153 |
| 6.6 | Avoidance Concept | 159 |
| 6.6.1 | Avoidance Grid Run | 160 |
| 6.6.2 | Mission Control Run | 166 |
| 6.6.3 | Computational Complexity | 180 |
| 6.7 | UTM Prototype Implementation | 183 |
| 6.7.1 | UTM Architecture | 184 |
| 6.7.2 | Handling Head-on Approach | 185 |
| 6.7.3 | Handling Converging Maneuver | 186 |
| 6.7.4 | Handling Overtake Maneuver | 188 |
| 6.7.5 | Position Notification Implementation | 190 |
| 6.7.6 | Collision Case Implementation | 193 |
| 6.8 | UTM Directives Framework Implementation on UAS | 201 |
| 6.8.1 | Rule Engine Architecture | 201 |
| 6.8.2 | Rule Engine Setup | 203 |
| 7 | Simulations | 205 |
| 7.1 | Test Plan | 205 |
| 7.1.1 | Testing approach | 205 |
| 7.1.2 | Test Cases Summary | 209 |
| 7.1.3 | Performance Evaluation | 209 |
| 7.2 | Testing Configuration | 213 |
| 7.3 | Non-cooperative test cases | 215 |
| 7.3.1 | Building avoidance | 215 |
| 7.3.2 | Slalom | 220 |
| 7.3.3 | Maze | 224 |
| 7.3.4 | Storm | 229 |
| 7.3.5 | Emergency Converging | 232 |
| 7.3.6 | Emergency Head-On | 238 |
| 7.3.7 | Emergency Mixed Head-On with Converging | 244 |
| 7.4 | Cooperative Test Cases | 251 |
| 7.4.1 | Rule-Based Converging | 251 |

| | | |
|----------|--|------------|
| 7.4.2 | Rule-Based Head-On | 257 |
| 7.4.3 | Rule-Based Mixed Head-On with Converging | 263 |
| 7.4.4 | Rule-Based Overtake | 270 |
| 7.5 | Test Cases Conclusion | 279 |
| 7.5.1 | Performance Evaluation | 279 |
| 7.5.2 | Adversary Behaviour Impact | 281 |
| 7.5.3 | Computation Footprint | 284 |
| 7.6 | Reduced Reach Sets Performance | 287 |
| 8 | Conclusion and Future Work | 293 |
| 8.1 | Summary | 293 |
| 8.1.1 | Hierarchical Decision Making | 294 |
| 8.1.2 | Use of Developed Reach Set Approximations | 295 |
| 8.2 | Comparison to Other Methods | 296 |
| 8.2.1 | Conservative and Adaptive Method Comparison | 296 |
| 8.2.2 | Scalability | 298 |
| 8.3 | Approach Reusability | 300 |
| 8.4 | Lessons Learned | 301 |
| 8.5 | Future Work | 305 |
| A | Complementary Definitions | 307 |
| B | Movement Automaton Theory | 313 |
| B.1 | Specialization of Hybrid Automaton | 313 |
| B.2 | Formal Movement Automaton Definition | 318 |
| B.3 | Segmented Movement Automaton | 319 |
| B.4 | Reference Trajectory Generator | 322 |
| C | Intruder Intersection Models | 325 |
| C.1 | Small-Body Direct-Movement Intruder Intersection | 325 |
| C.2 | Notable-Body Direct-Movement Intruder Intersection | 326 |
| C.3 | Maneuvering-Intruder-Intersection | 327 |
| D | Conflict Resolution Schemes | 335 |
| D.1 | Cooperative Conflict Resolution | 335 |
| D.2 | Non-Cooperative Conflict Resolution | 337 |
| E | Additional UTM functionality | 339 |
| E.1 | Weather Case Implementation | 339 |
| E.2 | Rule: Detect Collision Cases | 341 |
| E.3 | Rule: Resolve Collision Case | 344 |
| E.4 | Rule: Close Collision Cases | 345 |
| E.5 | Rule: Head on Approach | 346 |

| | |
|--|------------|
| E.6 Rule: Converging Maneuver | 348 |
| E.7 Rule: Overtake | 348 |
| E.8 Rule: Right Plane Heading | 350 |
| E.9 Rule: Enforce Safety Margin | 352 |
| F Comparison to the Previous Version of the Framework | 355 |
| G Framework - Matlab Implementaiton | 357 |
| G.1 Functionality Description | 357 |
| H Approach Guidelines | 365 |
| H.1 Guideline - Grid Size Calculation | 365 |
| H.2 Guideline - Safety Margin Calculation | 367 |
| Bibliography | 369 |

List of Tables

| | | |
|------|---|-------|
| 1 | List of Acronyms | xxiii |
| 2 | List of Organizations | xxiii |
| 3 | List of symbols | xxiv |
| 4 | Terminology | xxvi |
| 2.1 | Collision avoidance systems context overview [16]. | 13 |
| 2.2 | ICAO airspace summaries [3, 5]. | 19 |
| 2.3 | Small UAS Classes according to EASA. [47] | 40 |
| 2.4 | Proposed separation minima for UAS. [47] | 41 |
| 2.5 | Aspects of UAS flight rules.[47] | 42 |
| 2.6 | Geo-fencing in U-space. [47] | 44 |
| 4.1 | Controlled airspace margins violations incidents. | 77 |
| 4.2 | Non-controlled airspace margins violations incidents. | 79 |
| 4.3 | Navigation requirements evaluation metrics. | 79 |
| 6.1 | Input values for main axes movements. | 100 |
| 6.2 | Input values for diagonal axes movements. | 100 |
| 6.3 | Changing ratings from fuzzy to Boolean parameters. | 156 |
| 6.4 | Time-stamped <i>position notification</i> structure. | 192 |
| 6.5 | Collision case structure attendant data. | 198 |
| 6.6 | Collision case structure for given decision time-frame. | 200 |
| 7.1 | Test Cases Summary. | 209 |
| 7.2 | Movement orientations. | 214 |
| 7.3 | <i>UAS</i> parameters. | 214 |
| 7.4 | <i>Navigation Space</i> parameters. | 214 |
| 7.5 | <i>Avoidance Space</i> parameters. | 214 |
| 7.6 | <i>UAS</i> coloring. | 214 |
| 7.7 | Mission setup for <i>Building avoidance</i> scenario. | 215 |
| 7.8 | <i>Obstacle set</i> for <i>Building avoidance</i> scenario. | 216 |
| 7.9 | Distance to Body/Safety Margin Peaks for <i>Building avoidance scenario</i> . . | 218 |
| 7.10 | Path tracking for properties <i>Building avoidance</i> | 219 |
| 7.11 | Mission setup for <i>Slalom</i> scenario. | 220 |

| | | |
|------|---|-----|
| 7.12 | <i>Obstacle set</i> for <i>Slalom</i> scenario. | 220 |
| 7.13 | Distance to body/safety margin peaks for <i>Slalom scenario</i> | 222 |
| 7.14 | Path tracking properties for <i>Slalom</i> scenario. | 223 |
| 7.15 | Mission setup for <i>Maze</i> scenario. | 224 |
| 7.16 | <i>Obstacle set</i> for <i>Maze</i> scenario. | 225 |
| 7.17 | Distance to body/safety margin peaks for <i>Maze scenario</i> | 227 |
| 7.18 | Path tracking properties for <i>Maze</i> scenario. | 228 |
| 7.19 | Mission setup for <i>Storm</i> scenario. | 229 |
| 7.20 | <i>Constraint set</i> for <i>Storm</i> scenario. | 229 |
| 7.21 | Distance to body/safety margin peaks for <i>Storm scenario</i> | 231 |
| 7.22 | Path tracking properties for <i>Storm</i> scenario. | 232 |
| 7.23 | Mission setup for <i>Emergency converging</i> scenario. | 233 |
| 7.24 | Avoidance parameters for <i>Emergency converging</i> scenario. | 234 |
| 7.25 | Distance to safety margin peaks for the <i>emergency converging scenario</i> . . | 236 |
| 7.26 | Path tracking properties for the <i>Emergency converging</i> scenario. | 237 |
| 7.27 | Mission setup for <i>Emergency head-on</i> scenario. | 238 |
| 7.28 | Avoidance parameters for <i>Emergency head on</i> scenario. | 239 |
| 7.29 | Distance to safety margin peaks for <i>Emergency head-on scenario</i> | 242 |
| 7.30 | Path tracking properties for <i>Emergency head-on</i> scenario. | 243 |
| 7.31 | Mission setup for the <i>Emergency mixed</i> scenario. | 244 |
| 7.32 | Avoidance parameters for <i>Emergency mixed</i> scenario. | 245 |
| 7.33 | Distance to safety margin peaks for the <i>emergency mixed scenario</i> | 247 |
| 7.34 | Path tracking properties for the <i>Emergency mixed</i> scenario. | 248 |
| 7.35 | Mission setup for <i>Rule based converging</i> scenario. | 252 |
| 7.36 | Collision case for <i>Rule-based converging</i> scenario. | 254 |
| 7.37 | Distance to safety margin peaks for the <i>rule-based converging scenario</i> . . | 255 |
| 7.38 | Path tracking properties for <i>Rule-based converging</i> scenario. | 256 |
| 7.39 | Mission setup for <i>Rule-based head-on</i> scenario. | 257 |
| 7.40 | Collision case for the <i>rule-based head-on</i> scenario. | 261 |
| 7.41 | <i>Rule-based head-on</i> safety margin distances. | 262 |
| 7.42 | Path tracking properties for <i>rule-based head-on</i> scenario. | 262 |
| 7.43 | Mission setup for <i>rule-based mixed</i> scenario. | 264 |
| 7.44 | Collision cases for <i>rule-based mixed</i> scenario. | 267 |
| 7.45 | Distance to safety margin peaks for <i>rule-based mixed scenario</i> | 268 |
| 7.46 | Path tracking properties for <i>rule-based mixed</i> scenario. | 270 |
| 7.47 | Mission setup for all <i>Rule based overtake</i> scenarios. | 271 |
| 7.48 | Collision case for <i>Rule-based Overtake</i> scenario 2x speed. | 274 |
| 7.49 | Convergence and divergence waypoints for various speed differences. . . . | 274 |
| 7.50 | Distance to safety margin peaks for various overtaking speed in <i>rule-based overtake scenario</i> | 276 |
| 7.51 | Path tracking properties for <i>rule overtake 2x speed</i> scenario. | 277 |

| | | |
|------|--|-----|
| 7.52 | <i>Test cases performance evaluation.</i> | 279 |
| 7.53 | <i>Computation load statistics</i> for all test cases. | 284 |
| 7.54 | <i>Reduced reach set</i> computation methods performance | 291 |
| 8.1 | Mission setup for <i>Performance test</i> scenario. | 296 |
| 8.2 | <i>Obstacle set</i> for <i>Building avoidance</i> scenario. | 296 |
| B.1 | Orientation input values for main axes movements. | 321 |
| B.2 | Orientation input values for diagonal axes movements. | 321 |
| E.1 | Static/Dynamic weather constraint for given decision time-frame. | 340 |
| E.2 | Detect collision cases rule definition. | 343 |
| E.3 | Resolve collision case rule definition. | 344 |
| E.4 | Close collision case rule definition. | 346 |
| E.5 | Head on Approach rule definition. | 347 |
| E.6 | Converging maneuver rule definition. | 348 |
| E.7 | Overtake rule definition. | 350 |
| E.8 | Right plane heading rule definition. | 352 |
| E.9 | Enforce safety margin rule definition. | 353 |

List of Figures

| | | |
|------|--|-----|
| 1.1 | Present and future <i>Air Traffic Management</i> (ATC) [2]. | 1 |
| 1.2 | Flight rules overview. [2]. | 2 |
| 1.3 | <i>Detect & Avoid</i> (DAA) principle [8]. | 3 |
| 2.1 | Airspace classification examples. | 17 |
| 2.2 | Example situation in Czech Republic airspace. | 20 |
| 2.3 | Well Clear Threshold [38, 39]. | 25 |
| 2.4 | Example of DAM flight rerouting to homogenize traffic density [41]. | 28 |
| 2.5 | ACAS-X concept scheme [42]. | 30 |
| 2.6 | ACAS-X incident overview [42]. | 32 |
| 2.7 | Localized Weather Model [59]. | 51 |
| 2.8 | Example of upper troposphere winds [61]. | 52 |
| 3.1 | Example: The hybrid automaton example for UAS maneuver [80]. | 58 |
| 3.2 | Six space classifications [83]. | 59 |
| 5.1 | Movement Automaton for Copter UAS [87]. | 83 |
| 5.2 | <i>Movement set primitives</i> for Lattice-Based Movement Planning. [118]. | 86 |
| 5.3 | Example of LSTS toolchain deployment in a real environment [129] | 87 |
| 5.4 | UTM Operation Modes [2]. | 90 |
| 6.1 | Avoidance levels based on reaction time. | 93 |
| 6.2 | Avoidance Framework Concept. | 94 |
| 6.3 | Implemented movement set example. | 101 |
| 6.4 | Example: The <i>LiDAR</i> reading portioning - cells. | 105 |
| 6.5 | <i>Rapid Exploration tree as a result of</i> Constrained trajectory expansion. | 115 |
| 6.6 | <i>Coverage-Maximizing</i> reach set approximation. | 121 |
| 6.7 | <i>Turn-minimizing</i> reach set approximation. | 125 |
| 6.8 | ACAS-X imitation reach set approximation for various separation modes. | 129 |
| 6.9 | <i>Combined</i> reach set approximation. | 133 |
| 6.10 | Different count of LiDAR hits with different distance from UAS. | 136 |
| 6.11 | Overshadowed map obstacle by detected obstacles. | 137 |
| 6.12 | Obstacle hindrance impact on visibility in <i>Avoidance Grid Slice</i> | 141 |
| 6.13 | Map obstacle states after <i>Data fusion</i> | 141 |

| | | |
|------|---|-----|
| 6.14 | Example of Extracted Map Obstacle [156]. | 142 |
| 6.15 | Intruder UAS intersection rate along the expected trajectory. | 145 |
| 6.16 | Significant steps of <i>Avoidance grid run</i> (inner loop). | 161 |
| 6.17 | Example: The situation to be evaluated by <i>Avoidance Run</i> | 163 |
| 6.18 | Example: The <i>Visibility</i> evaluation by <i>Avoidance Run</i> | 164 |
| 6.19 | Example: The <i>Reachability</i> evaluation by <i>Avoidance Run</i> | 165 |
| 6.20 | Definitions for <i>Mission Control Run</i> (outer loop). | 166 |
| 6.21 | Joining multiple <i>Avoidance Grid Runs</i> to achieve Navigation. | 167 |
| 6.22 | Mission control run activity diagram. | 168 |
| 6.23 | Mission control orchestration diagram. | 177 |
| 6.24 | UAS Traffic Management (UTM) architecture overview. | 184 |
| 6.25 | Head-on approach detection/resolution/Closing | 186 |
| 6.26 | Converging maneuver Detection/Resolution/Closing | 187 |
| 6.27 | Overtake maneuver Detection/Resolution/Closing | 189 |
| 6.28 | Rule engine components overview. | 202 |
| 6.29 | Rule engine initialization with Rules of the air. | 203 |
| 7.1 | Test scenario for <i>Building avoidance</i> (static ground obstacles). | 217 |
| 7.2 | Distance to body/safety margin evolution for <i>Building avoidance scenario</i> | 218 |
| 7.3 | <i>Building avoidance</i> path tracking. | 219 |
| 7.4 | Computation time for <i>Building avoidance</i> scenario. | 219 |
| 7.5 | Test scenario for <i>Slalom</i> with a <i>hidden waypoint</i> | 221 |
| 7.6 | Distance to body/safety margin evolution for <i>Slalom scenario</i> | 222 |
| 7.7 | <i>Slalom</i> path tracking. | 223 |
| 7.8 | Computation time for <i>Slalom</i> scenario. | 224 |
| 7.9 | Test scenario for <i>Maze</i> | 226 |
| 7.10 | Distance to body/safety margin evolution for <i>Maze scenario</i> | 227 |
| 7.11 | <i>Maze</i> path tracking. | 228 |
| 7.12 | Computation time for <i>Maze</i> scenario. | 229 |
| 7.13 | Test scenario for <i>Storm</i> (Dynamic hard constraint). | 230 |
| 7.14 | Distance to body/safety margin evolution for <i>Storm scenario</i> | 231 |
| 7.15 | <i>Storm</i> avoidance scenario path tracking. | 232 |
| 7.16 | Computation time for <i>Maze</i> scenario. | 232 |
| 7.17 | Test scenario for <i>Emergency converging</i> (Intruder avoidance). | 235 |
| 7.18 | Distance to safety margin evolution for <i>emergency converging scenario</i> | 236 |
| 7.19 | <i>Trajectory tracking</i> for <i>Emergency converging</i> test case. | 237 |
| 7.20 | Computation time for <i>Emergency converging</i> scenario. | 238 |
| 7.21 | Test scenario for <i>Emergency head-on approach</i> (Intruder avoidance). | 241 |
| 7.22 | Distance to safety margin evolution for <i>emergency head-on scenario</i> | 242 |
| 7.23 | <i>Trajectory tracking</i> for <i>Emergency head-on</i> test case. | 243 |
| 7.24 | Computation time for <i>Emergency head-on</i> scenario. | 243 |

| | | |
|------|--|-----|
| 7.25 | Test scenario for the <i>Emergency mixed</i> situation with the <i>self-separation mode</i> | 246 |
| 7.26 | Distance to safety margin evolution for the <i>emergency mixed scenario</i> | 246 |
| 7.27 | Trajectory tracking for the <i>Emergency mixed</i> situation test case. | 248 |
| 7.28 | Computation time for <i>Emergency multiple</i> scenario. | 249 |
| 7.29 | Test scenario for <i>Rule-based converging</i> | 254 |
| 7.30 | Distance to safety margin evolution for the <i>rule-based converging scenario</i> | 255 |
| 7.31 | <i>Trajectory tracking</i> for <i>Rule-based converging</i> test case. | 256 |
| 7.32 | Computation time for <i>Rule-based converging</i> scenario. | 257 |
| 7.33 | Test scenario for the <i>rule-based head-on approach</i> (virtual roundabout). | 260 |
| 7.34 | Distance to safety margin evolution for the <i>rule-based head-on scenario</i> | 261 |
| 7.35 | <i>Trajectory tracking</i> for <i>rule-based head-on</i> test case. | 262 |
| 7.36 | Computation time for <i>rule-based head-on</i> scenario. | 263 |
| 7.37 | Test scenario for <i>rule-based mixed</i> situation with the <i>self-separation mode</i> | 266 |
| 7.38 | Distance to safety margin evolution for <i>rule-based mixed scenario</i> | 268 |
| 7.39 | Trajectory tracking for <i>rule-based mixed</i> situation test case. | 269 |
| 7.40 | Computation time for <i>rule-based multiple</i> scenario. | 270 |
| 7.41 | Test scenario for <i>rule-based Overtake</i> (double speed of overtaking aircraft). | 273 |
| 7.42 | <i>Rule-based overtake</i> trajectories at a different speed. | 275 |
| 7.43 | Overtake speed-dependent distance to safety margin evolution for <i>rule-based overtake scenario</i> | 275 |
| 7.44 | Trajectory tracking for <i>rule-based overtake double speed</i> situation test case. | 277 |
| 7.45 | Computation time for <i>rule-based overtake</i> scenario. | 278 |
| 7.46 | Adversarial behaviour of <i>UAS 2</i> (magenta) to compliant <i>UAS 1</i> (blue) | 282 |
| 7.47 | Expected/Real Distance to body/safety margin evolution for <i>adversarial behavior</i> of UAS 2. | 283 |
| 7.48 | Coverage-maximizing reach set computation example. | 288 |
| 7.49 | Turn-minimizing reach set computation example. | 288 |
| 7.50 | ACAS-like reach set computation example. | 289 |
| 8.1 | A testing scenario for method performance comparison [183]. | 297 |
| 8.2 | Distance to body margin evolution [183]. | 298 |
| A.1 | Polar surface calculation notation and plot | 310 |
| B.1 | Example of input signal segmentation to movements. | 314 |
| C.1 | One rate position $[d_d, d_\theta, d_\varphi]$ (green). deviated from linear trajectory (blue line) at point $x(10)$.(blue) with initial position x_s (red) | 328 |
| C.2 | Probability of intruder i_k position in ellipsoid $E(x(t), v)$ | 329 |
| C.3 | Avoidance grid $\mathcal{A}(t_i)$ (blue) intersection with elliptic cone intruder $i_k(x, v, \theta, \varphi)$ (red) example. | 330 |
| C.4 | Time Of Arrival (TOA) for one ellipsoid $E_D(x(\tau), v)$ | 332 |

| | | |
|-----|---|-----|
| D.1 | Cooperative conflict resolution via UTM authority. | 335 |
| D.2 | Non-cooperative conflict resolution via UAS claims. | 338 |
| E.1 | Right plane heading rule evaluation for various angles of approach α . . . | 351 |
| E.2 | Enforce safety margin rule evaluation for various angles of approach α . . | 353 |
| F.1 | Obstacle avoidance based on Reach sets concept [9]. | 355 |

List of Algorithms

| | | |
|-----|--|-----|
| 6.1 | <i>Wave-front propagation of Rapid Exploration Tree to form Reach Set</i> | 118 |
| 6.2 | Expansion Constraint function for <i>Coverage-Maximizing Reach Set Approximation</i> | 120 |
| 6.3 | Expansion Constraint function for <i>Turn-Minimizing Reach Set Approximation</i> | 123 |
| 6.4 | Expansion Constraint function for <i>ACAS-like Reach Set Approximation</i> | 128 |
| 6.5 | Reach Set Merge Function and Combined Reach Set calculation | 132 |
| 6.6 | Find best <i>Path</i> in <i>Avoidance Grid</i> | 162 |

Nomenclature

This chapter summarize used symbols (tab. 3), acronyms (tab. 1), terminology (tab. 4) and, organizations (tab. 2) mentioned in work.

| Acronym | Meaning |
|---------------------|---|
| UAS | Unmanned Autonomous System(including naval vehicles) |
| RPAS | Remotely Piloted Aerial System(lesser degree of autonomy) |
| LOS | Line Of Sight |
| VLOS | Visual Line Of Sight |
| BLOS | Behind Line Of Sight |
| SAA | Sense And Avoid |
| DAA | Detect And Avoid |
| MAC | Mid-Air Collision |
| CM-RSA | Coverage-Maximizing Reach Set Approximation |
| TM-RSA | Turning-Minimizing Reach Set Approximation |
| TCAS | Traffic Alert and Collision Avoidance System |
| ACAS X | Airborne Collision Avoidance System X |
| ACAS X _U | Airborne Collision Avoidance System X for UAS |
| CD&R | Collision Detection and Resolution |
| GPS | Global Positioning System |
| IMU | Internal Measurement Unit |
| LiDAR | Light Detection and Ranging |
| ADS-B | Automatic Dependent Surveillance – Broadcast |
| GSE | Ground Support Equipment |
| ATC | Air Traffic Control |
| ATO | Air Traffic Organization |
| C2 | Control and Communications |
| MOPS | Minimum Operational Performance Standard |

Table 1: List of Acronyms

| Acronym | Organization name |
|---------|---|
| ICAO | International Civil Aviation Organization (UN) |
| EASA | European Aviation Safety Agency (EU) |
| JARUS | Joint Authorities for Regulation of Unmanned Systems (EU) |
| FAA | Federal Aviation Administration (USA) |
| LSTS | Laboratório de Sistemas e Tecnologia Subaquática (PT) |
| FEUP | Faculdade de Engenharia da Universidade do Porto (PT) |

Table 2: List of Organizations

| Symbol | Explanation |
|---|---|
| A, B, C, D, \dots | Capital letters are used for matrices |
| $A(\dots), B(\dots), \dots$ | Functional matrices, (\dots) denotes parameters |
| $f(\dots), g(\dots), \dots$ | Vector or scalar functions (\dots) denotes parameters |
| $\vec{f}(\dots), \vec{g}(\dots), \dots$ | Explicit vector functions, when equation contains both types of scalar and vector functions |
| t, x, y, z, \dots | Vectors or scalar coefficients |
| $\vec{x}, \vec{o}, \vec{g}, \dots$ | Explicit vectors, when function contains both types of scalar and vector parameters. |
| θ, φ | Greek letters denoting angles in radians |

Table 3: List of symbols

| Terminology | Definition |
|---------------------|--|
| Air Traffic Control | A service operated by appropriate authority to promote the safe, orderly, and expeditious flow of air traffic |
| Aircraft | A device that is used or intended to be used for flight in the air |
| Airspace | Any portion of the atmosphere sustaining aircraft flight and which has defined boundaries and specified dimensions. Airspace may be classified as to the specific types of flight allowed, rules of operation, and restrictions by International Civil Aviation Organization standards or State regulation |
| Civil Aircraft | Another than public aircraft. |
| Collision Avoidance | The Sense and Avoid system function where the UAS takes appropriate action to prevent an intruder from penetrating the collision volume. The action is expected to be initiated within a relatively short time horizon before the closest point of approach. The collision avoidance function engages when all other modes of separation fail. |
| Communication Link | The voice or data relay of instructions or information between the UAS pilot and the air traffic controller and other NAS users. |
| Control Station | The equipment used to maintain control, communicate with, guide, or otherwise pilot an unmanned aircraft. |
| Crewmember (UAS) | In addition to the crewmembers identified in 14 CFR Part 1, a UAS flightcrew member includes pilots, sensor/payload operators, and visual observers, but may include other persons as appropriate or required to ensure safe operation of the aircraft. |
| Data Link | A ground-to-air communications system which transmits information via digitally coded pulses. |

| Terminology | Definition |
|--------------------------|--|
| Detect and Avoid | A term used instead of Sense and Avoid in the Terms of Reference for RTCA Special Committee 228. This new term has not been defined by RTCA and may be considered to have the same definition as Sense and Avoid when used in this document. |
| ICAO | International Civil Aviation Organization is a specialized agency of the United Nations whose objective is to develop the principles and techniques of international air navigation and to foster the planning and development of international civil air transport. |
| Manned Aircraft | Aircraft piloted by a human onboard. |
| RTCA | RTCA, Inc. is a private, not-for-profit corporation that develops consensus-based recommendations regarding communications, navigation, surveillance, and air traffic management system issues. RTCA functions as a Federal Advisory Committee. The FAA uses its recommendations as the basis for policy, program, and regulatory decisions and by the private sector as the basis for development, investment and other business decisions (www.rtca.org) |
| See and Avoid | When weather conditions permit, pilots operating instrument flight rules or visual flight rules are required to observe and maneuver to avoid another aircraft. |
| Self-Separation | Sense and Avoid system function where the UAS maneuvers within a sufficient time-frame to remain well clear of other airborne traffic. |
| Sense and Avoid | The capability of a UAS to remain well clear from and avoid collisions with other airborne traffic. Sense and Avoid provides the functions of self-separation and collision avoidance to establish an analogous capability to “see and avoid” required by manned aircraft. |
| Unmanned Aircraft | <ol style="list-style-type: none"> 1. A device used or intended to be used for flight in the air that has no onboard pilot. This device excludes missiles, weapons, or exploding warheads, but includes all classes of airplanes, helicopters, airships, and powered-lift aircraft without an onboard pilot. 2. An aircraft that is operated without the possibility of direct human intervention from within or on the aircraft. |
| Unmanned Aircraft System | An unmanned aircraft and its associated elements related to safe operations, which may include control stations (ground, ship, or air-based), control links, support equipment, payloads, flight termination systems, and launch/recovery equipment. |

| Terminology | Definition |
|----------------------|---|
| | An unmanned aircraft and associated elements (including communications links and the components that control the unmanned aircraft) that are required for the pilot-in-command to operate safely and efficiently in the national airspace system. |
| Visual Line of Sight | Unaided (corrective lenses and/or sunglasses exempted) visual contact between a pilot-in-command or a visual observer and a UAS sufficient to maintain safe operational control of the aircraft, know its location, and be able to scan the airspace in which it is operating to see and avoid other air traffic or objects aloft or on the ground. |

Table 4: Terminology

Note. Acronyms (tab. 1) and *Terminology* (tab. 4) comply with *ICAO*, *FAA*, and, *EASA* definitions, refer to [1] for more detailed information.

Chapter 1

Introduction

This works present an approach based on *reach set approximation* to detect & avoid various sort of threats in a *controlled/non-controlled* airspace environment.

The *motivation* is summarized in (sec. 1.1). The work *goals* are given in (sec. 1.2). A *thesis organization* with notes is summarized in (sec. 1.3). Notable contributions of work are listed in (sec. 1.4). The listing of student publications/technical reports/open source contributions is given in (sec. 1.5).

1.1 Motivation

The commercial potential of *Unmanned Autonomous Systems* (UAS) is significant enough to initiate one of the most significant changes in *aviation* history [2]. The current *usage* of UAS is limited by *strict regulations* [3, 4, 5]. The goal is to enable *full integration* of UAS in *European airspace* by the end of the year 2035 [6].

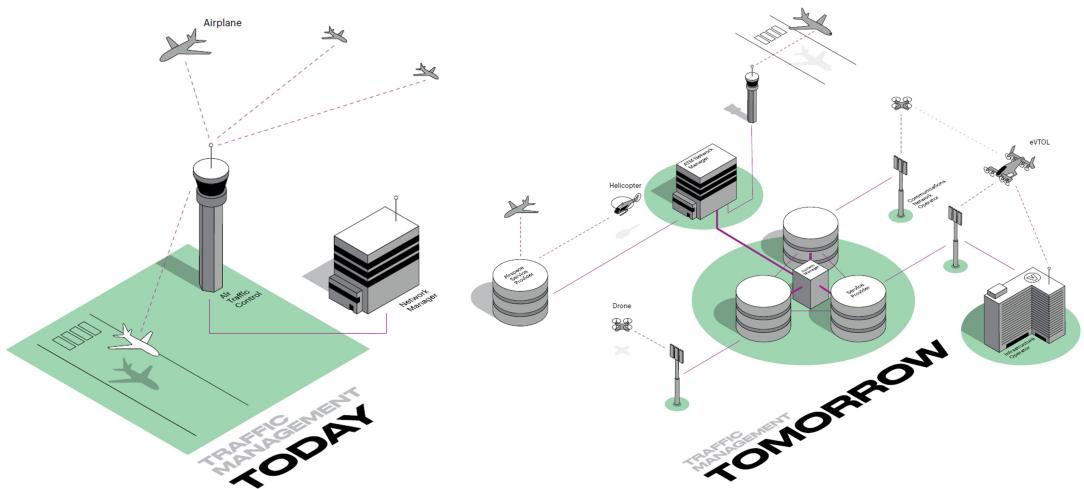


Figure 1.1: Present and future Air Traffic Management (ATC) [2].

UAS integration into Air Traffic: The major effort is focused on *Air Traffic Control* (ATC) changes. The *actual organization* (sec. 2.2) and management (sec. 2.4) of airspace is centralized and *human operated*.

The *ongoing changes* are shown in (fig. 1.1). The *UAS Traffic Management* (UTM) (sec. 2.5) complementing ATC is introduced to manage unmanned aviation. The additional traffic hubs, for UAS *delivery & transportation* services, are added. The greatest change is on previously *low-altitude uncontrolled* airspace; this space has new authority (UTM). The future UAS must implement mechanisms for *event-based* navigation and avoidance (sec. 2.6).

Since 2014, there is a visible strong political support for developing rules on drones, but regulations are harmonizing slowly. The European Aviation Safety Agency (EASA) has been tasked to develop a regulatory framework for drone operations and proposals for the regulation of "low-risk" UAV/UAS operations. In achieving this, EASA is working closely with the Joint Authorities for Regulation of Unmanned Systems (JARUS) [7].

The *operational rules* (sec. 2.3) with *rules of the air* [4] which are enforced now are simple. The *future flight rules* (fig. 1.2) will be more complicated including the precise waypoint system and more complex missions. The future flight rules will be micro-managed by UTM. The example of such micro-management is shown in (sec. 6.7, 6.8.1).

The *increase of traffic density* increases the *accident probability/severity*. Different threats are endangering UAS; the *obstacles* impose eminent destruction threat, intruders can cooperate in mutual avoidance, the weather avoidance becomes more critical, the international/national authority can impose additional operational constraints. These threats need to be well managed and prioritized to guarantee safe UAS operations.

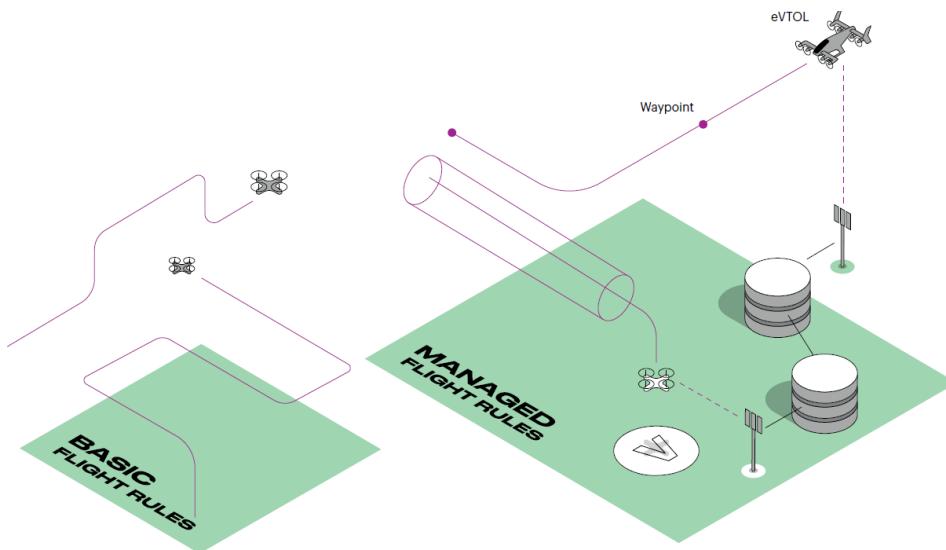


Figure 1.2: Flight rules overview. [2].

Detect & Avoid: The other approach is bottom-up, that means the development of basic, adaptable functionality to cover complex navigation/avoidance tasks in *controlled airspace*. The intuitive definition of *Detect & Avoid* (DAA) functionality is given in (fig. 1.3).



Figure 1.3: *Detect & Avoid* (DAA) principle [8].

The DAA can be applied in the short term (avoidance) and long term (navigation) tasks. It consists of three continuously repeating major steps:

1. *Sense* - the processing of intermediate sensor reading, the physical world approximation creation.
2. *Detect* - an addition of information from other sources to create a complex situation assessment scenario, future possibilities projection, and control strategy selection.
3. *Avoid* - execution of selected control strategy while checking possible changing events.

The *short term avoidance* (sec. 6.6.1) can be chained into *long term navigation* (sec. 6.6.2). This long term navigation can be used to solve *UTM/ATM* imposed constraints, under assumption that there is sufficient *data fusion* (sec. 6.5.4).

Challenge Motivation: Both *Air Traffic Control* and *Detect & Avoid* requires a tool to assess possible UAS maneuvering:

1. *Air Traffic Control* needs to know the UAS maneuvering capabilities in order to validate feasibility of *issued orders* and to predict a future *trajectory* for *collision prevention*.
2. *Detect & Avoid* needs to calculate feasible *system constraints feasible maneuvering strategy* in finite time to ensure own safety.

Reach Sets: These challenges are addressable by the employment of *reach sets* (sec. 3.2). The *reach set* is self-validating set of possible maneuvering strategies for some initial UAS state and some time-period.

To guarantee finite time evaluation, the discretization of *operation space* (sec. 6.3) and *movement discretization* (sec. 6.2.1) needs to be introduced. The discretization gives finite, countable sets of choices; therefore it also guarantees finite time evaluation process.

Our *implementation of reach set approximation* (sec. 6.4) enables to encode some sought behavioral patterns as natural properties (sec. 6.4.3). This implementation

enables to generate specific task-oriented *reach set approximation* for avoidance (sec. 6.4.4), navigation (sec. 6.4.5), UTM behavior predictions (sec. 6.7.6).

1.2 Goals

Situation: The *UAS* equipped with cooperative/non-cooperative surveillance sensors, with prior knowledge of operation space has to fly a mission represented ordered set of waypoints. The set of sensors can change depending on UAS construction. The minimal airworthiness for a given operation is assumed.

Problem: Given environment and artifact definitions (sec. 4.1) with *initial assumptions* (sec. 4.3) and *incremental problem definition* (sec. 4.4) develop *obstacle avoidance framework* which will satisfy *avoidance* (sec. 4.5) and *navigation* (sec. 4.6).

Expected Solution: Define an approach based on *reach sets* which are capable of:

1. *Static obstacle avoidance* - to avoid the ground, man-made structures in open terrain.
2. *Intruders avoidance* - to avoid flying objects which does not have the intention to harm our UAS, detected in sufficient distance.
3. *Geo-fencing support* - to avoid known zones/airspace portions, which have forbidden entry.
4. *Weather avoidance* - to avoid known zones of harmful weather conditions.
5. *Cooperative conflict resolution* - to communicate own position to authority and to follow authority orders.
6. *Treat prioritization* - to assess avoidance according to natural or man-made priorities. (Rather break geo-fence, than crashing into the ground).

Validation: Develop test-framework to showcase approach properties. Define *test scenarios* (sec. 7.1.1) to validate *Expected Solution Performance* (sec. tab. 7.1) concerning *avoidance capability* (sec. 7.5.1) and *computational feasibility* (sec. 7.5.3).

Application Requirements: There are following application requirements, based on similar applications for *manned aviation* and *industry expectations*:

1. *Low-performance requirements* - the computational footprint of the approach should be polynomial. The most of actual UAS systems have *embedded computer* with low computation power.

2. *Deterministic* - the *avoidance strategy* should be achieved in finite time frame. The mandatory requirement for *airborne operation support application*, the advice needs to be reproducible under the same conditions.
3. *Scalability* - the *avoidance framework* should be portable to the different platforms, and it should work with different sensor array. The interface requirement for *control* and *data fusion* coming from other *collision avoidance systems*.
4. *Adaptability* - the *avoidance process* should have tuning points where is possible to change behavior according to UAS context. The regulations are changing with location, time and circumstances, the part of calculation/control process needs to be implemented dynamically.

1.3 Overview

The thesis is organized like follows:

1. *Introduction* (ch. 1) - the introduction chapter giving an overview of work motivation, goals, contributions and *author's list of publication*.
2. *Collision Avoidance* (ch. 2) - This chapter gives aerospace-related background. The manned aviation is serving as the knowledge base for an assumption of future UAS Detect & Avoid functionality. The chapter gives overview of airspace classification, which told us where and what we can do or expect. Aircraft operational rules for general are reflected into *separation functionality*. The separation can be passively enforced by Air Traffic Management authority or active enforced by *ACAS-X/TCAS* systems. The *UAS Traffic Management* is parallel to manned aviation practices with additional layer of complexity. The *Event-Based avoidance* is introduced to give overview of concepts used later.
3. *Background Theory* (ch. 3) - this chapter outlines background necessary for approach understanding. The *control theory* system models are used as a base for the *reach set* calculation. The important concept of *hybrid automaton* (special case of hybrid automaton) is introduced. The LiDAR related theory and complements are presented at last.
4. *Problem Statement* (ch. 4) - this chapter states the problem solved in this work. The basic definition and terminology are established at beginning with initial problem and assumptions. Incremental problem is introduced with increasing complexity and relaxed conditions. Avoidance and Navigation functional and non-functional requirements are stated at last.
5. *State of Art* (ch. 5) - this chapter covers important results of other researcher works in topics of Movement Automaton, Sensor & Data Fusion, Navigation Algorithm,

Reach Sets, and Testing Approach. The *UAS Traffic Management* concept relevant to this work is introduced.

6. *Approach* (ch. 6) - this chapter describes the approach, it starts with an overview (sec. 6.1), outlining the block scheme of the system (fig. 6.2). The discretization of the space, trajectories, and system model are covered in (sec. 6.2 - 6.3), the following topics are covered:
 - a. *Reach Set Estimation* (sec. 6.4) - the discretization, performance evaluation, and generation algorithms.
 - b. *Encounter Modeling* (sec. 6.5.1 - 6.5.2) - static obstacles, intruders, static/moving constraints, and more.
 - c. *Collision Avoidance* (sec. 6.6) - the avoidance/navigation loop with global data fusion procedure, complexity and safety margin calculations.
 - d. *Further to Cooperative Operations* (sec. 6.7 - 6.8) - the approach to satisfy scalability and *UTM* requirements.
7. *Simulations* (ch. 7) the simulations cover aspects developed in approach, the test plan (tab. 7.1) summarizes test cases. The results are outlined in (tab. 7.52), the computation load statistics are summarized in (tab. 7.53). The tests are divided into the following categories:
 - a. *Non-cooperative Test Cases* (sec. 7.3) - various obstacles, weather constraints, and non-cooperative intruders test cases
 - b. *Cooperative Test Cases* (sec. 7.4) - maneuvers in controlled airspace under supervision of traffic management.
 - c. *Reach Set Estimation Performance and Properties* (sec. 7.6) - the comparison of various estimation methods, the impact on complexity.
8. *Conclusion and Future Work* (ch. 8) - work conclusion, summarizing achieved results, comparing other approaches, outlining reusable modular parts of approach, utilizing the future work on approach shortcomings.

1.4 Contributions

The *contributions* of this work can be divided into two categories:

Conceptual Contributions: The contributions are enhancing and enriching the conceptual level of *Detect & Avoid* systems, namely:

1. *Movement automaton control and prediction* - the necessity to abstract the control of the system from the *detect and avoid* systems, leads to customization of the

hybrid automaton (sec. 3.3) to *movement automaton* (def. B.2). The movement automaton can be adapted to nonlinear system (sec. 6.2.2) as an instance (sec. 6.2.3). The movement automaton can be also used as a predictor of the system (sec. B.4). The *initial state disparity issue* [9] has been addressed in (sec. B.3).

2. *Trajectory as a discrete command chain* - the *movement automaton* as a control interface consuming the discrete command chain enabled the *finite discrete representation* of trajectory (eq. B.33).
3. *Reach set discretization* - the *infinite reach set* (def. 1) can be represented as a tree of movements from system initial state (def. 13). This tree can have associated properties, like reachability of each trajectory segment (eq. 6.91). The advantage of having finite maneuver set, with little precision sacrifice, which can be calculated prior the flight have a huge impact on *computational complexity* (sec. 6.6.3). The interfaces enable to use the approach on different platforms with small computational power.
4. *Operational space assessment* - the operational space is separated by a grid into finite set of the cells. Each cell has properties to track like the occurrence of an obstacle, the presence of intruder or impact of constraint. The universal data fusion procedure (sec. 6.5.4) is enabling accumulation of treat information from various sources.
5. *Hierarchical threat assessment* - the *various threat* sources (obstacles, intruders, constraints) are categorized according to operational environment/rules, and their avoidance priority is handled according to that (fig. 6.22).
6. *UAS Traffic Management* - the architecture proposal for traffic management as the cooperative authority (sec. 6.7.1) covers some basic maneuvers (sec. 6.7.2, 6.7.3, 6.7.4), the more important is the adaptability of presented approach to cooperative (app. D.1) and non-cooperative (app. D.2) avoidance modes, showing adaptability.

Implementation Contributions: The concepts, which solve implementation issues of *Detect & Avoid* systems, namely:

1. *Operational space segmentation* - the *planar grid* (sec. 6.3) slice (fig. 6.4) have been selected, because it can be used for fast assessment of LiDAR scan data to estimate, obstacle (6.10) or visibility (6.11). The cell volume is increasing with distance from UAS, and this gives us decreased space status assessment complexity.
2. *Wave-front algorithm for avoidance estimation* - to estimate reach set the space exploration method has been developed. The *rapid exploration tree* (fig. 6.5) is employed as *wave-front* expansion (alg. 6.1). Various shapes and properties of *reach*

set estimation can be achieved employing the *constrained expansion functions* for *coverage-maximizing* (alg. 6.2), turn-minimizing (alg. 6.3), and, *ACAS-like* (alg. 6.4) reach set approximations.

3. *Encounter and constraints models* - the planar grid used in solution required the development of encounter models for static obstacles, constraints (sec. 6.5.1), intruders, moving constraints (sec. 6.5.2). The intersection algorithms can be reused in other approaches using an unusual grid.
4. *Avoidance process enhancement (Rule engine)* - the air traffic rules are changing based on time and geographic location, the UTM concept is under development. These reasons are calling to use flexible implementation architecture, the rule engine (sec. 6.8.1). The rule engine can be set to cover any kind of rules (fig. 6.29).

1.5 List of Publications

This *section* contains the list of published articles, proceedings, technical reports and module projects, relevant to the *thesis topic*.

Article: Alojz Gomola, Joao Borges de Sousa, Fernando Lobo Pereira, and Pavel Klang. Obstacle avoidance framework based on reach sets. In Iberian Robotics conference, pages 768–779. Springer, 2017. [9]¹.

Summary: This report on preliminary investigations concerning the development of a LiDAR-based detect and avoid (DAA) system with a low computational footprint for small Unmanned Air Systems. The focus is on the integration with nominal flight control systems and computational feasibility. The proposed system decomposes the SAA problem into the following components detection, space assessment, escape trajectory estimation and avoidance execution. The control logic is encoded with the help of a hybrid automaton. The properties of the system are studied with the help of approximations to time slices of the UAV reach set.

Article: Juraj Števek, Michal Kvasnica, Miroslav Fikar, and Alojz Gomola. A parametric programming approach to automated integrated circuit design. IEEE Transactions on Control Systems Technology, 26(4):1180–1191, 2018. [10]².

Summary: The proposal of an optimization-based slotting approach to automated *integrated circuit* design for generating a power transistor. The original slotting problem is formulated as a mixed-integer linear program. It is solved through parametric optimization with an advantage that usage of any commercial optimization solvers on user’s side is avoided with short evaluation time and simple implementation. The approach is

¹Draft available online: <https://goo.gl/kZujZE>

²IEEE copy online: <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7981386>

applied in specific very large scale integration production technology and demonstrated on an example.

Contributions: Point-location algorithm for MPC region selection.

Proceedings: Kristian Klausen, Jostein Borgen Moe, Jonathan Cornel van den Hoorn, Alojz Gomola, Thor I Fossen, and Tor Arne Johansen. Coordinated control concept for recovery of a fixed-wing uav on a ship using a net carried by multirotor UAVs. In Unmanned Aircraft Systems (ICUAS), 2016 International Conference on, pages 964–973. IEEE, 2016, [11]³.

Summary: Ship-based Unmanned Aerial Vehicle (UAV) operations represent an important field of research which enables a large variety of mission types. Most of these operations demand a high level of endurance which normally requires the use of a fixed-wing UAV. Traditionally, a net located on the ship deck is used for recovering the fixed-wing UAV. However, there are numerous challenges when attempting autonomous landings in such environments. Waves will induce heave motion, and turbulence near the ship will make approaches challenging. In this paper, we present a concept using multi-rotor UAVs to move the recovery operation off the ship deck. To recover the fixed-wing UAV, a net is suspended below two coordinated multirotor UAVs which can synchronize the movement with the fixed-wing UAV. The approach trajectory can be optimized with respect to the wind direction, and turbulence caused by the ship can be avoided. In addition, the multirotor UAVs can transport the net at a certain speed along the trajectory of the fixed-wing UAV, thus decreasing the relative velocity between the net and fixed-wing UAV to reduce the forces of impact. This paper proves the proposed concept through a simulation study and a preliminary control system architecture.

Contributions: Ground station maneuver implementation, messaging between ground station/UAVs. RTK-GPS for precise navigation integration, Net-release mechanism design, Mechanical parts for 3D printer modeling.

Proceedings: Alojz Gomola. An aspect-oriented solution for mutual exclusion in embedded systems. In Alena Kozakova, editor, ELITECH15: 17th Conference of doctoral students, pages 964–973, Bratislava, Slovak Republic, May 2015. Online publication, [12]⁴.

Summary: Embedded systems are developed for a wide range of applications. The best-known applications are industrial process control and banking solutions. Fault tolerance is a crucial requirement in long-term embedded systems. This paper presents solution for mutual exclusion in embedded systems. The usual mutual exclusion solution using semaphores is a crosscutting concern. Semaphores are difficult to maintain in code, and their failure rate is high. We propose new aspect-oriented solution for mutual exclusion. Our solution utilizes aspect-oriented approach, is usable in other systems and

³Public copy available online: http://folk.ntnu.no/torarnj/ICUAS2016_singlecolumn.pdf

⁴Draft available online: <https://github.com/logomo/Elitech15-paper>

designed to be robust against program changes, and it provides a solution to aspect fragility problem.

Technical report: Alojz Gomola, Pavel Klang, and Jan Ludvik. Probabilistic approach in data fusion for obstacle avoidance framework based on reach sets. In Internal publication collection, pages 1–93. Honeywell, 2017, [13]⁵.

Summary: The *sensor input* and *information sources* fusion procedure design to obtain rated space assessment for visibility, obstacle occupancy, and reachability evaluation. A unique statistical approach was proposed to couple partial ratings under reading and time uncertainty. The key contribution is scalable approach to evaluate *UAS action space* and *Feasible Trajectories* properties.

Technical report: Alojz Gomola. Model predictive control of unmanned air vehicles with obstacle avoidance capabilities, technical report, FEUP, 2017, [14]⁶.

Summary: Initial solution of *predictive control* problem for *UAS* navigation in constrained, non-controlled airspace. The key contribution is *Movement Automaton* (a special type of hybrid automaton) establishment in current form (sec. 6.2.1). The *stability*, *controllability* and *observability* properties have been proven. The *feasibility* of *Movement Automaton* as *control interface* and *future state predictor* has been shown through formal proof and excessive testing.

Technical report: Alojz Gomola. Optimal control of unmanned air vehicles with obstacle avoidance capabilities in partially known environment. Technical report, FEUP, 2017. [15]⁷.

Summary: The *optimization problem* for *obstacle avoidance* (eq. 4.34) to satisfy *avoidance requirements* (sec. 4.5)

Framework: Feature-based ACAS⁸ implementation to support claims of this work has been developed through the course of years 2016-2019. The framework provides:

1. *Simulation environment* - full support for single/multiple UAS simulation support in controlled/non-controlled airspace.
2. *Mission Control Support* - standard mission control support for *sparse ordered waypoint* mission type.
3. *UAS Traffic Management* - support for *collaborative* weather and collision avoidance with general authority in controlled airspace. The configurable *event-handling* through *rule-engine*.

⁵Public copy available online: <https://github.com/logomo/Data-Fusion-Report>

⁶Public copy available online: <https://github.com/logomo/Predictive-control---Final-report>

⁷Public copy available online: <https://github.com/logomo/Optimal-Control>

⁸Matlab prototype available online: <https://github.com/logomo/Feature-based-ACAS>

4. *Encounter Models* - the model for *static obstacles* supporting point-cloud generation of LiDAR sensor, various intruders model supporting the ADS-B like encounter model, static/dynamic polygon constraints with altitude range.
5. *Collision Avoidance* - the support for *cooperative* and *non-cooperative* behavior in controlled airspace, *non-collaborative*, *non-adversary* behavior in non-controlled airspace.
6. *Reach Set Estimation Methods* - four methods for various property focused *Reach Set Estimations*.

Chapter 2

Collision Avoidance

The context of Collision Avoidance is introduced in (tab. 2.1), the structure was taken from Gardi [16]. The *state of art* changes was incorporated into the table.

| <i>Function</i> | <i>Equipment/Task</i> |
|----------------------------|---|
| Communication | Telecommunication datalinks, Controller Pilot Data Link-Control (CPDLC), Voice Communication |
| Navigation | Navigation sensors including GNSS, INS, etc. providing 3D/4D navigation capabilities. |
| Surveillance | Cooperative Systems (TCAS, ACAS, etc.) Non-cooperative Sensors (LiDAR, Cameras, etc.) |
| Situation Awareness | Early Warning Systems, CDTI Display |
| Autonomous Decision Making | Strategic, Tactical, Emergency Flight Planning, Intelligent Collision Detection, Conflict Resolution and Prevention, Weather/Terrain/Constraints Avoidance |

Table 2.1: Collision avoidance systems context overview [16].

2.1 Overview

The *Detect and Avoid*, as a part of *Collision Avoidance*, impacts all collision avoidance aspects (tab. 2.1). This work focuses on the *Reach Sets* which gives us the following focus area:

1. *Communication* - it is assumed the command & control communication link is stable. This aspect is not affected by reach sets.
2. *Navigation* - minimal navigation framework needs to be implemented for full experimentation with the navigation capabilities of the *Reach Set* based trajectory generation.
3. *Surveillance* - the surveillance will be covered with necessary low-cost technologies, the simulated sensor inputs for following surveillance equipment is considered:
 - a. *Non-cooperative* - LiDAR Sensor.
 - b. *Cooperative* - ADS-B In/Out.
4. *Situation awareness* - the situation awareness focuses on *space segmentation* and *safety evaluation* to support proper safe trajectory selection from *reach set*.
5. *Autonomous decision making* - the *reach set* covers all possible avoidance strategies, to know how to select proper strategy is key in successfully avoidance maneuvering.

Communication: An overview elaboration on capability, reliability, security, architecture have been summarized by Johansen et al. in [17].

The current state of art *communication lines* and relay approaches are sufficient to provide necessary utilities. The use of an existing 4G/3G mobile network is the most probable candidate for low altitude UAS operations. The necessity to build a back-up network for communication is still an open topic.

Navigation: An overview is given by Nex [18] *Waypoint planning in a 3D environment* is elaborated in [19]. *Waypoint Tracking and Test Environments* are thoughtfully discussed in [20, 21, 22, 23].

All navigation methods are fairly similar. Consisting of the following steps in the loop:

1. Select goal waypoint.
2. Evaluate feasible navigation strategies (cost function).
3. Select navigation strategy and generate reference trajectory.
4. Follow the reference trajectory with UAS system.

The *evaluation process* and selection criteria need to be designed in the context of *reach sets*.

Surveillance: TCAS and ACAS systems cover the cooperative surveillance, an interesting aspect of these systems are *Resolution Advisories* [24] for TCAS [25], for ACAS. These advisories are giving the suggestions for the pilot to avoid an occurring collision. The responsibility for following advisories and avoiding collision is on the pilot.

This mechanism needs to be changed to increase the determinism of UAS behavior. The voluntary approach of advisories needs to be replaced with a mandatory approach (directives).

Situation awareness: The aspect of the situation awareness of surroundings has been introduced in [26]. *LiDAR-based SAA* system has been introduced by Sabatini [27] further enhanced by Ramasay [28]. Other *Non-Cooperative* sensors and their feasibility have been outlined in Ramasay work [29].

The common ground of these works is an operational space discretization into various forms of finite discrete sets to enable deterministic decision making. The key issue is to find a good rate between space democratization and solution precision. The large cells in the grid usually hide many escape routes. The small cells in grid usually increase the computational complexity and diminish computation time optimal solution.

Examples of *situation awareness*: implementation can be found mainly in *human-centered* systems, *Early Warning System* has been proposed by Lee [30] and an adaptive version by Miller [31]. Effects of *CDTI Display* visualization and human decision impact have been examined by Thomas [32]. *Self Separation* aspect has been examined by Williams [33].

The important concept for *UAS* is internal data representation and autonomous situation resolution. The autonomous situation resolution (decision-making process) can be extracted from human pilot operation procedures.

Manned Aviation Concepts: The introduction of necessary concepts from manned aviation is organic in UAS concept understanding. Many of the concepts are taken directly from manned aviation. The main contribution is to change the *human decisions* into *autonomous system decisions*.

Airspace Classification: For integration of the UAS systems into non -segregated airspace it is necessary to know the classification of the *operational space*. Who is the authority, in which space, and when the authority is enforced. The general overview of airspace classes and concepts accepted by ICAO/FAA/EASA are outlined in (sec. 2.2). The common viewpoint is emphasized.

Aircraft Operational Rules: It is necessary to know the basic rules in controlled/uncontrolled airspace. What is expected to be done by the aircraft in various flight modes. What is minimal equipment's, what is airworthiness and so on. The basic regulations are outlined in (sec. 2.3). Visual Flight Rules (VFR) interesting parts can

be found in (sec. 2.3.1). Instrumental Flight Rules interesting parts can be found in (sec. 2.3.2).

Active/Passive Separation and Self-Separation: The *safe navigation* in *airspace* have multiple levels, going from least strict to very strict and keeping aircraft or UAS *well clear* of all threats. There is first protective barrel known as *well clear*; then there is a smaller protective barrel representing *near miss*, then the smallest protective barrel representing *crash zone*. The *Well clear* state of aircraft/UAS in airspace important parts are mentioned in (sec.2.4).

The important role of *Air Traffic Control* for manned aircraft is introduced in (sec. 2.4.1). The general aviation *routing* principles can be used on the various scale for *UAS routing*. The form of *ATC* commands and directives must persist in future UAS traffic management, for compatibility reasons.

The current Collision Avoidance Systems systems TCAS (2.4.3) and ACAS-X (2.4.2) which can be used as unmanned approach base are introduced.

UAS Traffic Management: The traffic management functionality is analyzed in (sec. 2.5), two major movements EU USPACE (2.5.1) and US NASA UTM (2.5.2) exists. The most notable information from operation specification is extracted there.

Event-Based Avoidance (sec. 2.6) defines basic event-based control invoked by *UTM*; two major categories are analyzed in *Mid-Air Collision Prevention* (sec. 2.6.1) and *Weather Impact* (sec. 2.6.2).

2.2 Airspace Classification

Motivation: The *Airspace Classification*, last changed by ICAO in 1990, is described in [5]. The purpose of airspace classification from *collision avoidance perspective* are the following:

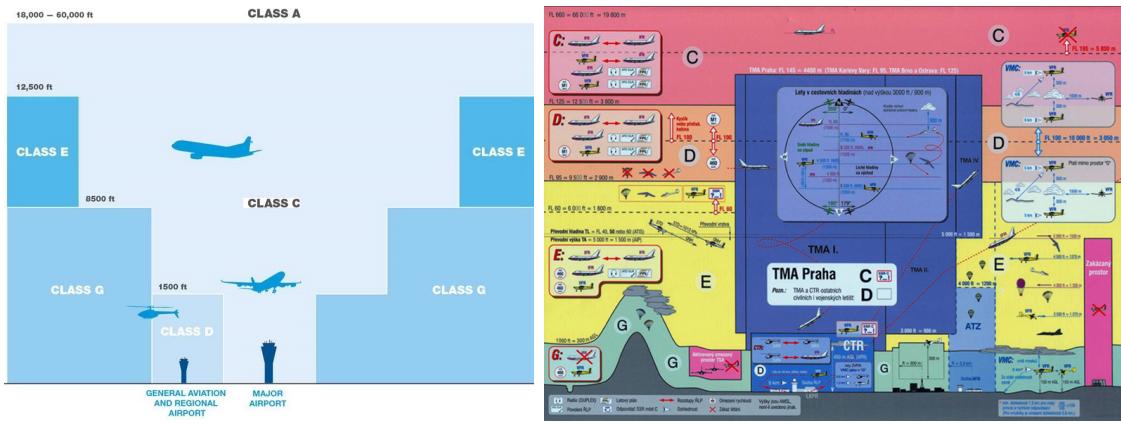
1. *Separation* - Maintaining a minimum distance between an aircraft and another aircraft or terrain to avoid collisions. There are following separation types:
 - a. *Vertical separation* - to ensure sufficient altitude differences from threats separate those airspace attendants.
 - b. *Horizontal separation* - to endure that *airspace attendants* have sufficient horizontal distance from threats
2. Clearance - permission award process by Air Traffic Control (ATC) for an airspace attendant to proceed with flight plan execution/change.
3. *Organization* - ensure that *airspace attendant* can expect a minimal level of separation and airspace organization depending on the type.

Note. This work focuses on *separation* in both controlled/non-controlled airspace.

Airspace Categorization: The airspace is segmented depending on the *altitude*. The altitude boundaries can be given in one of two ways:

1. *Altitude above Mean Sea Level* (AMSL) - the barometric altitude is used as boundary reference.
2. *Altitude above Ground Level* (AGL) - the boundary copies terrain.

There is an example of airspace classification for *Australia*¹ and *Czech Republic*²:



(a) Australian airspace classification.

(b) Czech Republic airspace classification.

Figure 2.1: Airspace classification examples.

The *Airspace classes* are given like follow:

Class A (Upper) Operational Airspace (18 000 - 60 000 feet AMSL) (considered as part of national airport controlled airspace in EU) - the operational airspace where the most of the commercial/military flights are conducted

Class B International Airport Airspace (0 - 18 000 feet AMSL) (a special class of airports in the U. S., considered as C airspace in EU) - the airspace down to the ground, with stricter preventive measures against mid-air collision and ground collisions.

Class C National Airport Airspace (0 - 18 000 feet AMSL) - the standard national level airport airspace with preventive measurements against mid-air collision and ground collision situations.

Class D Regional Airport Airspace (0 - 1 500 feet AMSL) - the *regional airport authority* inactive for the most of the time.

Class E (Lower) Operational Airspace (1 500 - 12 500 feet AMSL) - the controlled airspace on lower flight levels, lower airworthiness requirements are applied.

¹Australian airspace classification: <http://www.airservicesaustralia.com/services/how-air-traffic-control-works/how-airspace-is-managed/>

²Czech Republic Airspace Classification:https://lis.rlp.cz/vfrmanual/actual/enr_1_en.html

Class F (Upper) Uncontrolled Airspace (500 - 1 500 feet AGL) - the upper portion of uncontrolled airspace (deprecated and merged into E class airspace in most countries).

Class G (Lower) Uncontrolled Airspace (0 - 500 feet AGL) - the lower portion of *airspace* which is *free to fly*.

Note. The *boundaries of the airspace classes* may vary between countries in all segments. The *ATC authority* is usually enforced from *FL-60* (6 000 feet AMSL).

Airspace Roles and Responsibilities: The *airspace characteristics* is given in (tab 2.2). The *characteristics* are following for each airspace class:

1. *Controlled Airspace* - indicates if *Air Traffic Control* has authority over the *airspace class*, in general, the airspace classes can be divided into:
 - a. *Uncontrolled Airspace* (classes F/G) - the *ATC* have an only advisory role, the responsibility for safe flight is only on the *pilot side*.
 - b. *Controlled Airspace* (classes A-E) - the *ATC* have full mandate to issue directives and validate or revoke clearance for pilots action. If the pilot is following ATC recommendation and order to given a *degree of precision*, it should remain safe.
2. *Instrumental Flight Rules* - indication if manned aviation with compliant with IFR requirements can enter into airspace.
3. *Special Visual Flight Rules* - indication if manned aviation with compliant with SVFR requirements can enter into airspace (the U.S. only).
4. *Visual Flight Rules* - indication if manned aviation with compliant with VFR requirements can enter into airspace.
5. *Flight Clearance* - the flight plan approval and flight plan changes are required to enter and operate in a given airspace. The *Flight Clearance* can be:
 - a. *Required* - full cooperation with ATC is required.
 - b. *Advisory Only* - the ATC provides only flight plan advisories to minimize collision risk, the responsibility for safety and surveillance is on pilot side.
6. *Separation* - the ATC is actively looking for conflict occurrence and changes the *traffic flow* to keep airspace attendants separated.
7. *Traffic Information* - the ATC is providing the *movement and intentions* of air traffic attendants in the given segment to others.

| Class | Controlled | IFR | SVFR | VFR | ATC Clearance | Separation | Traffic Information |
|-------|------------|-----|------|-----|-----------------------|--|--|
| A | Yes | Yes | No | No | Required | For all flights | N/A |
| B | Yes | Yes | Yes | Yes | Required | For all flights | N/A |
| C | Yes | Yes | Yes | Yes | Required | For all flights: IFR/SVFR to IFR/VFR | Provided: - all VFR |
| D | Yes | Yes | Yes | Yes | Required | Provided: IFR to IFR | Provided: - all VFR - all IFR |
| E | Yes | Yes | Yes | Yes | Required: IFR/SVFR | Provided: IFR to IFR | Provided: - all VFR - all IFR |
| F | No | Yes | No | Yes | Advisory only | Provided: IFR to IFR | Provided: - all VFR - all IFR |
| G | No | Yes | No | Yes | No | No | Provided: ³ - all VFR - all IFR |

Table 2.2: ICAO airspace summaries [3, 5].

Example of Airspace Segmentation: There is an example of an *airspace map* for the Czech Republic⁴. The example snapshot contains (fig .2.2) following elements giving the complex feeling of *National Airspace*:

1. *Active Airports* (red fill circles) - the active B, C, D class airports, with permanent or temporary ATC for IFR/VFR flights.
2. *Inactive Airports* (gray fill circles) - the inactive or without temporary ATC, enabling VFR/IFR operations after ATC clearance from the active airport.
3. *Flight Corridors* (orange boundary polygons) - the permanent/temporary flight corridor in defined flight levels. The flight corridors have time slot reservation and moving along them requires ATC clearance.
4. *Airport Corridors* (aquamarine boundary polygons) - the corridors where entering/leave requires ATC clearance, usually used for climb/descent maneuvers, the higher safety measurements are imposed.
5. *Restricted Airspace* (orange fill polygons) - temporary or permanently banned airspace portions. These areas are established by ATC in cases of emergency or military maneuvers or other special requests.

⁴The live map for Czech National Airspace can be viewed on: <http://aisview.rlp.cz/>

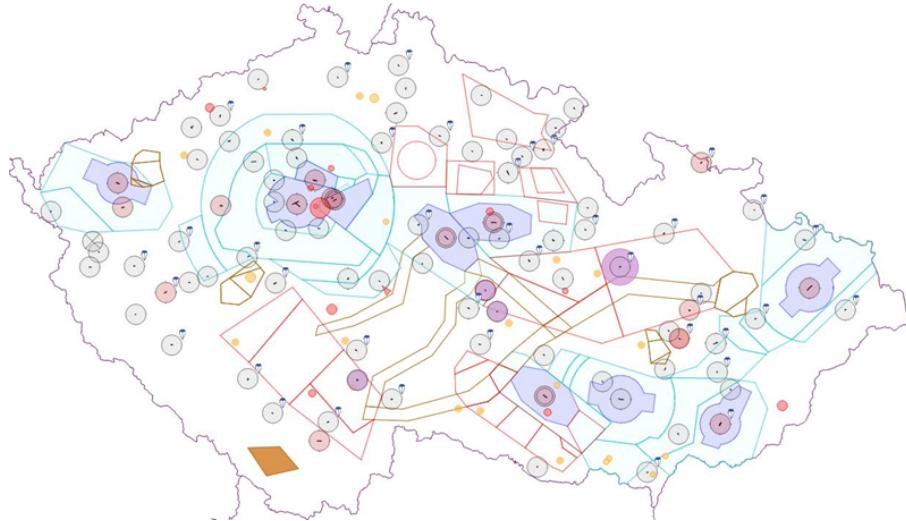


Figure 2.2: Example situation in Czech Republic airspace.

Note. Most of the marked zones are restricted for the RPAS/UAS systems. To integrate RPAS/UAS systems, it is necessary to make them compliant with manned aviation regulations and requirements. The impact of RPAS/UAS system failures on human body [34] and *civil airplanes* [35] outlines future integration requirements.

Aircraft Categorization: The aircraft categorization for *avoidance priority* is defined in ICAO Annex 2. [4] also known as *aviation classes for immediate avoidance*. This categorization is based on maneuverability:

1. *Manned aviation in distress* - any kind of manned aviation in distress has the highest priority in avoidance.
2. *Balloons* - having only limited vertical maneuverability, impaired by any wind impact significantly, the balloons are avoided by any capable manned aviation.
3. *Gliders* - absence of propulsion impairs vertical avoidance capabilities.
4. *Air-fueling and towing* - dependable load impairs maneuvering capability.
5. *Airships* - limited cruising speed turning angles impairs overall avoidance performance.
6. *Other Manned aviation with propulsion* - normal maneuverability, capable of horizontal and vertical separation

Note. This categorization is based on airspace attendant maneuverability

There is also supplemental categorization based on operational approach speed (for manned aviation only) [36] going like follow:

Class A *Small single engine* - cruising speed (100 - 110 knots), runway approach speed (70 - 110 knots).

Class B *Small multi-engine* - cruising speed (130 - 150 knots), runway approach speed (85 - 130 knots).

Class C *Airline jet*- cruising speed (160 - 240 knots), runway approach speed (115 - 160 knots).

Class D *Large jet/military jet*- cruising speed (185 - 265 knots), runway approach speed (130 - 185 knots).

Class E *Special military*- cruising speed (230 - 275 knots), runway approach speed (155 - 230 knots).

Note. The *differences* between cruising speed/runway approach speed are not significant (concerning ratio). The speed does not impact maneuverability as much as propulsion type and steering elements.

2.3 Aircraft Operational Rules

Motivation: The *aircraft operation rules* are ranging from personal, through technical, to standardization category. In this section, the *flight rules* will be outlined in necessary depth for *collision avoidance*. The goal of this section is to give an overview of airspace constraints.

Rules Origin: The *Rules of the Air* are provided by the following documents:

1. *SERA Regulation 923/2012* - laying down the common rules of the air and operational provisions regarding services and procedures in air navigation and amending Implementing Regulation (EU) No 1035/2011 and Regulations (EC) No 1265/2007, (EC) No 1794/2006, (EC) No 730/2006, (EC) No 1033/2006 and (EU) No 255/2010 [37] notable contributions:
 - a. *Table of cruising levels* - Appendix III.
 - b. *ATS airspace classes — services provided and flight requirements* - Appendix IV.
2. *SERA Regulation 2016/1185* - Commission Implementing Regulation (EU) 2016/1185 of 20 July 2016 amending Implementing Regulation (EU) No 923/2012 as regards the update and completion of the common rules of the air and operational provisions regarding services and procedures in air navigation (SERA Part C) and repealing Regulation (EC) No 730/2006.

3. *ICAO Annex II.* - the most accepted rules of the air document [4], providing general rules of the air (sec. 6.7.2, 6.7.3, 6.7.4).

Note. This section contains important parts from previously mentioned documents.

2.3.1 Visual Flight Rules

Motivation: A *Visual Flight Rules* (VFR) requires the pilot ability to see outside the cockpit to:

1. *Control an aircraft* - to check responses to control input (UAS self-diagnostic).
2. *Check altitude* - to check and asses an altitude based on the estimated ground distance (UAS - barometric altimeter, ranging sensors).
3. *Navigate* - to steer aircraft for reaching long term goal, including position estimation. (UAS Navigation Module, GPS Module)
4. *Avoid other obstacles and intruders* - see and avoid procedures, following rules of the air in case of intruder avoidance. (UAS Detect And Avoid system).

Note. Each of VFR task has an equivalent task in IFR or UAS implementation. The system impact on aircraft airworthiness is interchangeable up to some degree.

See And Avoid: The pilot has situations awareness of its surroundings and velocity. The *horizontal/vertical* avoidance maneuvers are executed if necessary.

Night VFR: Some countries (ex. U. S.) allows flights under VFR when the sun is after horizon (astronomical night). The separation minimums are same. There is a *clear sky requirement* (FAA) which disallows any clouds on higher flight levels.

Traffic Advisories: The *United States, Australia, and, Canada* ATC provides the service of *flight following*. A pilot can request the *flight following* outside the *B, C, D* class airspace, the ATC will communicate possible threats to pilot, the responsibility for safety is on the pilot.

Note. The *traffic advisories* are a weaker version of *directives*; they can be used for RPAS systems communication.

Weather Separation: VFR Weather Minimums – *Visual Meteorological Conditions* (VMC). Europe currently follows SERA (Standardised European Rules of the Air) rules, which are mostly the same as ICAO rules used throughout the world (local exceptions may apply). Current VFR Weather Minimums are:

1. Altitude: at and above 10000 feet (3000 m), in every class of airspace – flight visibility 8000 m; 1500m horizontally from clouds, 1000 feet (300 m) vertically from clouds.
2. Altitude: below 10000 feet (3000 m) and above 3000 feet (900 m) or above 1000 feet (300 m) above terrain (whichever is higher) in every class of airspace – flight visibility 5000 m, 1500m horizontally from clouds, 1000 feet (300m) vertically from clouds.
3. Altitude: at or below 3000 feet (900 m) or at or below 1000 feet (300m) above terrain in class A, B, C, D, E airspace (controlled) – flight visibility 5km and 1500 m horizontally from clouds 1000 feet (300 m) vertically from clouds.
4. Altitude: at or below 3000 feet (900 m) or at or below 1000 feet (300 m) above terrain in class F and G airspace (uncontrolled) – flight visibility 5000 m, clear of cloud and with a sight of the surface.

There are exceptions from the last rule. ICAO rules allow for flights (at or below 3000 feet or at or below 1000 feet above terrain in F and G uncontrolled airspace) when flight visibility is no less than 1500m:

1. at speeds that, in the prevailing visibility, will give adequate opportunity to observe other traffic or any obstacles in time to avoid a collision,
2. in circumstances in which the probability of encounters with other traffic would normally be low, e.g., in areas of low volume traffic and for aerial work at low levels

A similar exception (at or below 3000 feet or at or below 1000 feet above terrain) applies to helicopters, which can fly when flight visibility is less than 1500 m.

Refer home-country AIP⁵ (usually or AIP ENR 1.1) for local restrictions.

Note. The clouds are very dangerous for UAS because they impair sensors, causes freezing and damages the on-board electronic, the WMC can be used in *weather safety handling definitions*.

2.3.2 Instrumental Flight Rules

Idea: The key idea of *Instrument Flight Rules* (IFR) is to provide additional surveillance resulting in better air traffic knowledge, weather and situation awareness.

The situation is different in the case of UAS; single autonomous agent replaces the combination of human pilot decisions and surveillance provided information. The main challenge is to replicate the human pilot data fusion process leading to *situation awareness* and later decision-making process leading to *aircraft control*.

⁵Czech republic AIP 1.1 document: https://lis.rlp.cz/ais_data/aip/data/valid/e1-2.pdf

Instrument Flight Rules: By definition [4], *Implementation of Visual Flight Rules under the weather or other conditions.* The main goal of IFR is to keep aircraft separated and with clearance.

The *separation of aircraft* is the main responsibility of *Air Traffic Control* (ATC) which is providing IFR aircraft with guidance.

There is minimum equipment which needs to be carried by aircraft to be considered IFR airworthiness [3]. The minimal equipment to be carried for IFR flight in European Airspace (EuroControl) is given as follow:

1. *GPS* - mandatory for all flight levels in controlled airspace.
2. *Transmitter* - the way to communicate with ATC/Ground. There can be digital transmitter equipment to receive automatic warnings from TCAS/ACAS systems, *ATC directives*, and, notices to airman (NOTAMS).
3. *Transponder* - the broadcasting device is giving out aircraft position and additional mandatory information. The current plan is to make ADS-B In/Out mandatory for all air traffic attendants.
4. *Barometric Altimeter* - to measure precise AMSL altitude.

Note. Carried does not mean used. Most of the pilots in the UK believe that usage of GPS is illegal and they avoid it in small private flights in controlled airspace.

The other popular practice is to turn off the *transponder* because planes tend to hide their position and heading for safety reasons.

Required Navigation Performance: The required navigation performance is depending on *flight level* where the *Flight Plan* (Mission) is executed. The *planned trajectory deviation* is the key performance parameter [3].

2.4 Separation from Air Traffic

Remaining "Well Clear": The separation from *air traffic* is an activity when *our airplane* tries to stay away from other traffic in a safe manner.

Before the definition of what is safe, there is a need for some margin definitions around the aircraft. The margins are enclosing a space in the form of a barrel, where *airplane position* is center, the horizontal plane is base for a circular boundary, the vertical axis is base for *distance boundary* (fig. 2.3).

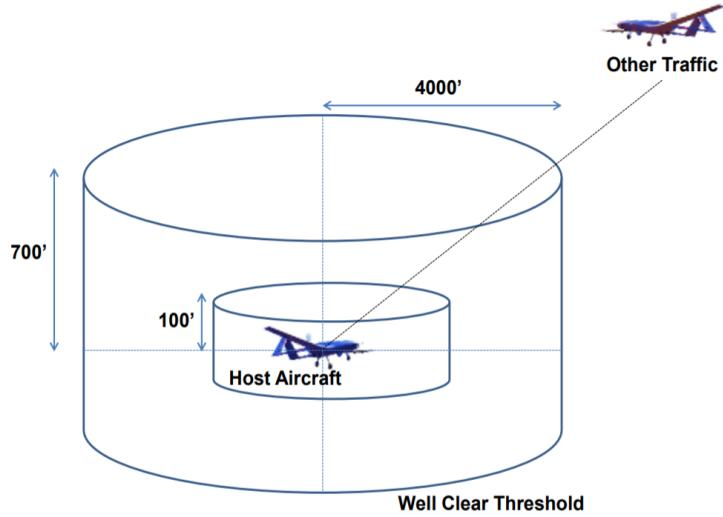


Figure 2.3: Well Clear Threshold [38, 39].

The boundaries and margins classification is taken from [38, 39] and goes like follow:

1. "*Alert*" - the distance to at least one surrounding aircraft is within the *alert margin*; the pilot is alerted about the possible threat, no action is required from the pilot side.
2. "*Well Clear*" - the intruder enters into *well clear range*, the intruder is threatening the *airplane* directly, the pilot is noticed about the security incident.
3. "*Near Miss*" - the intruder gets very close to the *airplane*, the body hit or *turbulence impact* is very probable.
4. "*Body Hit*" - the intruder stuck the *airplane body*.

These incidents are increasing their severity, the goal of separation is to keep every airspace attendant *well clear*, outside well clear threshold (fig. 2.3).

Air Traffic Control: The air traffic control has a passive role in separation, it manages the airspace and gives *clearance* for air space users actions. There is active support for VFR (sec. 2.3.1) and IFR (sec. 2.3.2). The role of *Air Traffic Control* is discussed in (sec. 2.4.1)

ACAS-X/TCAS: There is support systems fo prevent *Airborne Collision*, which is supporting the *active avoidance* for *IFR* flights (sec. 2.3.2). Their role is to provide the surveillance of *airplane* surroundings and give advisories to pilot. More about next-generation system family ACAS-X can be read in (sec. 2.4.2). The current generation *Airborne Collision* System TCAS is discussed in (sec. 2.4.3).

2.4.1 Air Traffic Control

Motivation: The modern *Air Traffic Control* (ATC) procedures are outlined in ICAO 4444 [3]. The ATC roles and responsibilities regarding *clearance*, *self-separation*, and, *provided traffic information* are summarized in (tab. 2.2).

The *main role* of *Air Traffic Control* (ATC) is to support the organization of the *airspace* concerning *preemptive detect and avoid*.

Note. The *Reactive* and *Event-Based* detect and avoid for manned aviation are covered by *ACAS-X/TCAS systems* (sec. 2.4.2, 2.4.3)

Commands issued by ATC: There are multiple levels of commands issued by ATC, their characteristics and the compulsory level is defined as follows:

1. *Notification* - the information notification, commonly known as *NoTice to AirMan* (NOTAM), depending on flight mode, can be transmitted as a voice message or information broadcast. They usually contain information about weather and traffic situation in the given sector.
2. *Warning* - directed message to specific general aviation which may require some direct action. The information is usually informative, but the action is not mandatory.
3. *Recommendation* - directed a message to specific general aviation which requires direct action. The order is usually a specific action, but the action is not mandatory to be executed by the pilot.
4. *Directive* - directed a message to specific general aviation which requires direct action. The order is specific action, and the order fulfillment is mandatory for the pilot.

Separation enforcement: The *separation* is the main feature of the ATC in controlled airspace of airports (B, C, D class). Its enforced by the management of *action clearance*. The ATC issues the clearance for take-off/landing sequence.

The clearance for climb/descent is given at the beginning when the flight plan is approved. The ATC issues the time slots for selected pathways. The continuous monitoring of air traffic is executed in periods. The airplane deviation from the cleared plan should be minimal.

If there is any incident the *ATC* can take the following actions:

1. *Heading change* - order *general aviation* to change heading in a given time frame (horizontal navigation). This command is usually issued to correct horizontal deviations in path tracking.
2. *Velocity change* - order *general aviation* to change velocity in a given time frame. This command is usually issued to correct time deviations in path tracking.

3. *Altitude change* (Flight Level) - order *general aviation* to climb or descent in a given time frame. This command is usually issued to correct vertical deviations in path tracking (wrong flight level).
4. *Divergence* - order *general aviation* to follow different waypoint in flight plan (goal change). This command is usually used to resolve incidents or to reroute traffic to another hub.
5. *Convergence* - order *general aviation* to return to following original waypoint (goal return). This command is usually used when incident has been resolved in a short time, and the original flight *path* can be re-established.
6. *Restrictions Enforcement* - order *general aviation* to avoid some point with defined distance.

Note. All ATC commands can be requested be *general aviation* for clearance to be granted. Meaning the airplane can ask ATC to perform any of the listed actions.

The *separation* can be divided into two distinct types to form *well clear barrel* (fig. 2.3). The separation types are the following:

1. *Horizontal separation* - keep clear of any intruders on the horizontal plane (flight level plane).
2. *Vertical separation* - keep clear of any intruders on given altitude (flight level) range.

Note. The *horizontal/vertical* separation is enforced independently, reducing 3D avoidance problem to 2D/1D avoidance problem.

Traffic Information: The air traffic information is delivered to general aviation depending on airspace type (tab. 2.2).

Note. The *D class* airports usually do not have radar or transponder; therefore, they can provide only visual guidance in altitude/horizontal range around "control tower".

Dynamic Airspace Management: A real-time *Airspace Management* approach has been presented in [40] following *Dynamic Airspace Management* [41].

The *airspace* is usually divided into the *clusters* where each cluster is managed by separate ATC. When the airplane is leaving one cluster, airplane hand-over is executed.

There is a problem when some airspace cluster is *congested* or overloaded by controlled airplanes. The example of such a situation is given in (fig. 2.4).

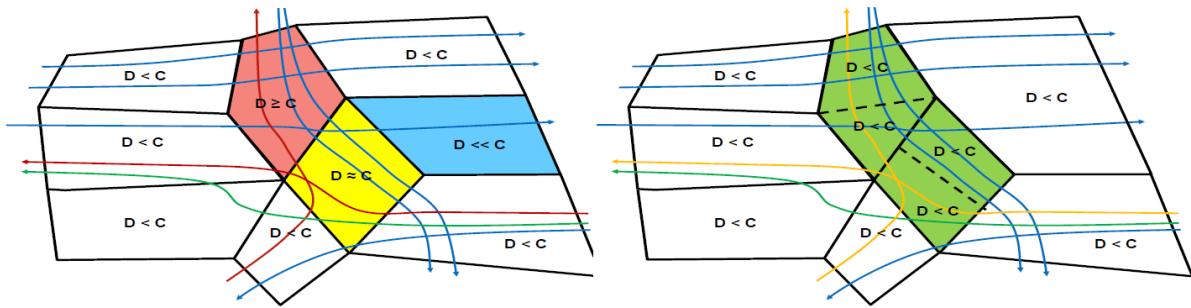


Figure 2.4: Example of DAM flight rerouting to homogenize traffic density [41].

Note. The airways cannot be changed, because the real-time change of airways is difficult. The change of cluster authority is possible because there are no changes for aircraft. The airspace clusters are divided into three categories (fig. 2.4):

1. *Under-fill* (blue) - there is fewer airplanes than its authority capacity.
2. *Saturated* (yellow) - there is enough airplanes to fill authority capacity.
3. *Over-fill* (red) - there are more airplanes than its authority capacity.

The algorithm [41] will swap some airspace portions between neighboring authorities to balance the load (all authorities should be saturated in ideal conditions) (green) (fig. 2.4).

2.4.2 Airborne Collision Avoidance System X

This section follows the summary leaflet [42]. The standard is still under development, the principles relevant for this work are outlined.

Overview: A development of *ACAS-X* system is the FAA funded research and development program of a new approach to airborne collision avoidance. It has been ongoing since early 2008. ACAS-X approach takes advantage of years of TCAS development. The main purpose of new system development is the rapid evolution of computational capabilities and the emergence of *Unmanned Autonomous Systems*.

The main purpose for manned aviation is to provide necessary advisories to pilot for *Mid-Air Collision* (MAC) avoidance. There are following improvements:

1. *Reduce Unnecessary Advisories* - The pilot/UAS is receiving avoidance advisories or warning.
2. *Extending collision avoidance to other classes of aircraft* - the current TCAS system is available mainly for bigger manned aviation, the future lies in the integration of UAS systems into non-segregated airspace.
3. *Improvement of Surveillance Environment* - There is development in ADS-B technology and modern non-cooperative sensors like LiDAR or millimeter radar, which are enhancing surveillance capabilities of current and future airplanes.

ACAS-X variants: The *ACAS-X* is not universal "one-size fits all" system. There are multiple variations for the different type of aviation:

1. *ACAS-X_P* - the general purpose ACAS-X that makes active interrogations to establish the range of intruders. The successor to TCAS II.
2. *ACAS-X_P* - a version of ACAS-X that relies solely on passive ADS-B to track intruders and does not make active interrogations. It is intended for general aviation (a class of aircraft not currently required to fit TCAS II).
3. *ACAS X_O* - a mode of operation of ACAS X designed for particular operations for which ACAS-X_A is unsuitable and might generate an unacceptable number of nuisance alerts (e.g., procedures with reduced separation, such as closely spaced parallel approaches).
4. *ACAS X_U* - designed for *Unmanned Aircraft Systems* (UAS).

Note. The *ACAS X_U* is meant also for *reduced separation* approach in tightly packed airspace (ex. air-taxi). The determinism and *false-positive* alerts occurrence minimization must be assured in order to enable UAS systems into non-segregated airspace.

ACAS-X Concept: The *ACAS-X* collision avoidance logic (fig. 2.5) is distinguished into two phases:

1. *Offline development phase* (Pre-calculation) similar to (sec. 6.4) - ACAS-X is based on a *probabilistic model* providing a statistical representation of the aircraft position in the future (position cone). It also takes into account the safety and operational objectives of the system (payload/weather/visibility/airspace/aircraft class). This enables the logic to be tailored to particular procedures or *airspace configurations*.

This is fed into an *optimization process* called dynamic programming to determine the best course of action to follow according to the context of the conflict. This takes account of a reward (safety) to the cost (fuel consumption). The concurrent optimization enables to explore multiple maneuvers to determine which will increase the *separation level* (safety) and which will decrease *fuel consumption* (cost).

Key metrics for *operational sustainability* and pilot acceptability include *minimizing* the frequency of resolution advisories (UAS/GA) and traffic alerts (GA). This results into a decrease of reversals/intentional intruder altitude crossing cases.

2. *Real-time operation* (Avoidance run) similar to (sec. 6.6.2) - the *look-up table* is used in real-time onboard the aircraft to resolve conflicts. An ACAS-X system collects *surveillance measurements* from an array of *information sources* and *sensors*. The *situation evaluation* is executed every second (decision time).

Various models are used (e. g. a probabilistic sensor model accounting for sensor error characteristics) to estimate a state distribution, which is a probability distribution over the current positions and velocities of aircraft and intruders. The *state distribution* determines where to look in the numeric look-up table to determine the best action to take. If deemed necessary *Resolution Advisory* is issued to pilots/UAS control module.

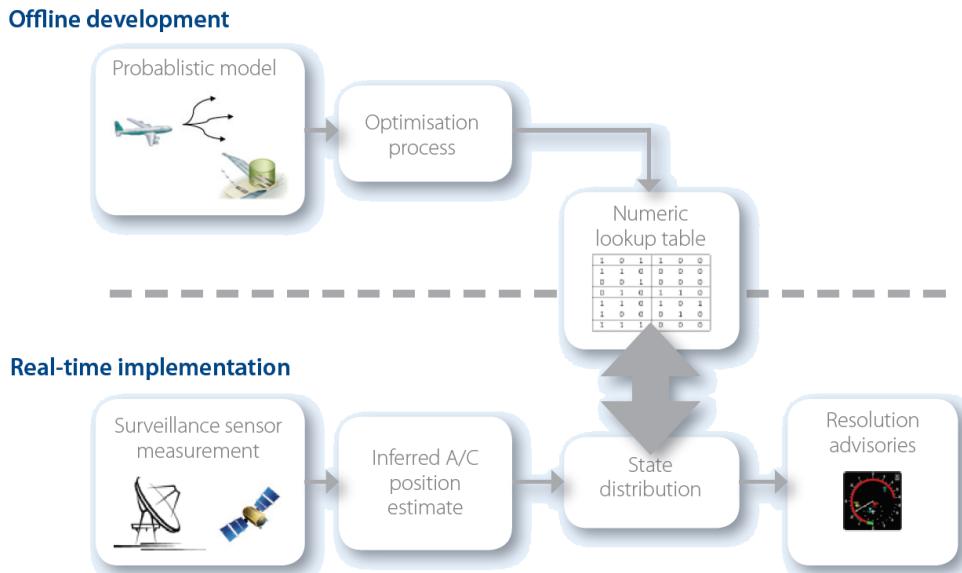


Figure 2.5: ACAS-X concept scheme [42].

Note. Two-phase calculation with offline development and real-time operation phase concept will be used differently in our method. (sec. 6.6)

Advisories: The *ACAS-X* is taking the advisories categorization from *ICAO Annex 10*. [43]. The advisories are recommended (directives for UAS) actions to take.

Airborne Collision Avoidance Systems (ACAS) equipment provides two types of advisories to pilots: Resolution Advisories (RAs) and Traffic Advisories (TAs). These are defined as follows:

1. *Resolution advisory* (RA) - an indication given to the flight crew recommending:
 - a. Manoeuvre intended to provide separation from all threats.
 - b. Manoeuvre restriction intended to maintain existing separation.
2. *Corrective resolution advisory* - a resolution advisory that advises the pilot to deviate from the current flight path.
3. *Preventive resolution advisory* - a resolution advisory that advises the pilot to avoid certain deviations from the current flight path but does not require any change in the current flight path.

4. *Traffic advisory* (TA) - An indication given to the flight crew that a certain intruder is a potential threat.

Note. The *UAS* system with full autonomy must handle the solution of the *directives* (advisories) from UTM and other systems. The example of configurable handling mechanism - *Rule engine* is given in (sec. 6.8.1).

Example Overview of the Incident: The *example of the incident* and a resolution is outlined in (fig. 2.6). The *example* is given for a better understanding of *ACAS-X* roles & responsibilities.

1. *Initial state* - the initial state is given like follow:

Green Airplane (A/C 1) is cruising on *flight level* FL-390 (39 000 feet)

Orange Airplane (A/C 2) is cruising on *flight level* FL-370 (37 000 feet)

2. *Orange aircraft transponder mishap* - the *orange airplane* appears on *flight level* FL-405 (45 000 feet). The *green aircraft* controller (pilot) wants to *descent* to *orange aircraft* real *flight level* (FL-370).
3. *Green airplane descent* - the *Green aircraft* starts to descend. The *Orange aircraft* keeps the *flight level* (FL-370).
4. *Green aircraft Active Surveillance* - the *green aircraft* uses *active surveillance* to detect *orange aircraft*.

The *ACAS-X* issues *Adjust Vertical Speed*, *Adjust - Resolution Advisory* (AVSA RA) to the *green aircraft* to *level off* and stop *descent* or at least slow it.

Then as *ACAS-X* issues *Climb Resolution Advisory* to the *green aircraft* mandating to get altitude.

Note. The main issue is that *pilot* is a responsible decision-maker; therefore pilot can refuse to follow *ACAS-X advisories*.

5. *Orange airplane starts descending* - the *orange aircraft* detects the *flight level disparity* due to the *active surveillance* detection of *green aircraft* on *FL-370*.

The *Green aircraft* is tail-gating *orange aircraft* from above. The *ACAS-X* evaluates situation and issues a *Descent Resolution Advisory* to avoid pursuit.

The *pilot* of *orange aircraft* follows the order of RA

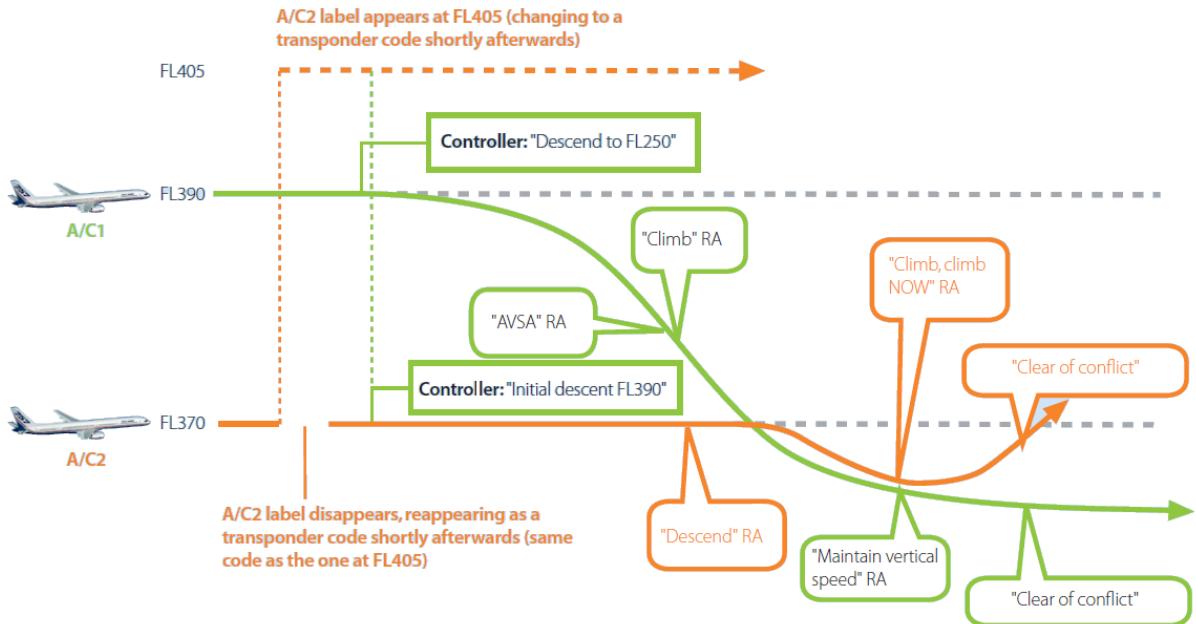


Figure 2.6: ACAS-X incident overview [42].

6. *Tailgating detection* - the *orange airplane* is in front of the *green airplane*, both airplanes are descending at the same rate.

Note. This situation is very dangerous because it is very close to *intentional pursuit scenario*.

The *orange airplane* is *pursued*, its *ACAS-X* issues the *Climb Now Resolution Advisory*. The pilot follows the advisory.

The *green airplane* is *pursuing*, and its goal is to *descend* to flight level FL-250. Its *ACAS-X* issues *maintain vertical speed resolution advisory*.

7. *Conflict resolution* - both airplanes follow *Resolution Advisories*. When the significant vertical distance is reached. The conflict is marked as resolved.

2.4.3 Traffic Collision Avoidance System

The *TCAS* functionality is equal to *ACAS-X_A* (sec. 2.4.2) functionality. The *TCAS II*. version 7.1 technical specification can be found in [44]. A *Resolution Advisory* (RA) detection algorithm is outlined in [45].

2.5 UAS Traffic Management (UTM)

Introduction: This section strongly follows [6], which outlined the basic concept of operations for Remotely Piloted Aerial Systems (RPAS)/Unmanned Autonomous Systems (UAS) Air Traffic Management (ATM) (later re-branded as UTM).

The *RPAS/UAS* integration into *non-segregated airspace* follows the general manned aviation procedure:

- For a selected type of Operations VLOS/BVLOS/VFR/IFR:
- For a selected class of Air traffic (class 1. - class 7):
 - For a selected class of airspace (class A - class G):
 - Deliver Operation Performance Standards (\geq General Aviation)

The prototype of regulation for *RPAS/UAS* standard from EASA can be found at [46]. The section will continue with an outline of important functionality.

Airspace Assessment: The future *UTM* must be capable of *airspace* assessment. In manned aviation, the *airspace assessment* is normally triggered by either rise of traffic, environmental issues, capacity issues, and safety concerns or adapting the design to meet forecasted demands.

Presently RPAS/UAS operations have not triggered an airspace assessment s most areas are indicated as *no RPAS/UAS zones*. The *restricted areas* are already known on aviation maps (airport, nuclear power station, etc.). However, there are similarities with RPAS/UAS operations below 500 ft (AGL), that can trigger this requirement for an airspace assessment like, but not exclusive:

1. *The increase of operations density* - UAS taxi can lead to an increase in traffic density in class C and F airspaces, the UAS delivery system can lead to increased traffic density in F class airspace.
2. *Introduction of BVLOS, autonomous VLOS/ELOS/BLOS operations* - current RPAS/ UAS operations are limited to VLOS which limits operation space. When this limitation is lifted, new business cases will open, leading to an *increase* of RPAS/UAS traffic.
3. *Safety Concerns* - there is not enough accidents or critical RPAS/UAS misuse cases, to increase safety concerns, especially G class airspace does not have many manned aviation parallels.
4. *Environmental Aspect* - the RPAS/UAS is not constructed from clean and safe materials, any accident can lead to serious habitat damage (ex. gasoline in water reservoir).

The *assessment* should develop a new type of airspace organization able to cater to the new demand of operations and ensure safety levels are met. The airspace assessment can take into considerations the following aspects:

1. *Airspace classification* - further airspace decomposition in F/G uncontrolled airspace to establish *flight routes* and controlled areas. Further flight levels in C class airspace segmentation to enforce manned/unmanned aviation separation.

2. *Traffic complexity and density* - the *congestion* of traffic is very common on the road. The capability to stop or stay still in the air is very costly to implement and maintain both in manned/unmanned aviation.
3. *Geographical situation* - flat-lands vs. mountains, urban vs. rural areas.
4. *Privacy* - in *very low altitude* operations the privacy is always a concern. The current restrictions to flew over private properties needs to be lifted in order to enable increased higher traffic density.
5. *Security* - the *UTM* and *RPAS/UAS* systems are creating a network of *autonomous agents*; this network is vulnerable to any kind of *cyber/physical* threat.

Types of RPAS/UAS operations: The *future UTM functionality* must cover a wide range of functionality. It is envisaged that RPAS/UAS will operate in a mixed environment adhering to the requirements of the specified airspace it is operating in. RPAS/UAS will be able to operate as follows:

1. *Very Low-Level (VLL) operations* ($< 500\text{feet(AGL)}$).
2. *IFR (Instrumental Flight Rules) or VFR (Visual Flight Rules)* ($500\text{feet} \leq \text{altitude} < 60000\text{feet(AMSL)}$) - following the same rules that apply to manned aircraft. These can be conducted in RLOS or B-RLOS conditions.
3. *Very High-Level operations (VHL suborbital IFR operations above FL600)* ($\geq 60000\text{ feet (AMSL)}$).

VLL operations (Below 500 feet AGL): Operations performed at altitudes below 500 feet are not new to manned aviation as many operators - police, armed forces, balloons, gliders, training crafts, fire-fighting, ultra-light aircraft are allowed to operate in this environment. The rule allows VFR traffic to operate, under specific conditions prescribed by the competent national authorities, conditions that can differ from State to State. RPAS/UAS operating in this volume of airspace do not however confirm either IFR or VFR as given in ICAO Annex 2. [4].

1. *VLOS (Visual Line Of Sight)* - RPAS/UAS operations within 500 meters range and max 500 feet altitude from the a pilot. One of the main responsibilities of the pilot is the safe execution of the flight through visual means. The distance can be increased by the use of one or more observers, sometimes referred to as Extended-VLOS (E-VLOS).
2. *BVLOS (Beyond Visual Line Of Sight)* - RPAS/UAS operations beyond 500 meters range but below 500ft. BVLOS does not require the operator to ensure the safety of the flight visually, and technical solutions such as DAA and C2 data link are required. RPAS/UTM does not adhere to VFR or IFR requirements; however, it is

foreseen that these flights could be conducted in IMC or VMC conditions. BVLOS operations are already being conducted in several States. Some examples are:

- a. Power-line control.
- b. Maritime surveillance.
- c. Pipeline control.
- d. Agriculture.

VLL Management System: In order to accommodate the expected growth of traffic in this airspace and ensure a sufficient level of safety, it is anticipated the necessity for a supporting UTM system. This VLL Traffic Management system will provide a series of localization and information services, aiming to the provision of information to the RPAS pilots and manned traffic. The VLL UTM system will not provide an active control service for RPAS in a normal ATC fashion, due to a large number of RPAS/UAS involved. Such a system could be based on existing technologies, such as the mobile phone network. Specific RPAS/UAS reporting systems, providing authorization and information capability, are already in use in several states.

The RPAS/UAS management system will have to cater to the following aspects:

1. RPAS/UAS Flight planning.
2. RPAS/UAS Flight authorization.
3. Real-time RPAS/UAS tracking capability.
4. Provision of actual weather and aeronautical information.

As previously mentioned, it is envisaged that the VLL management system will not support the active controlling of RPAS/UAS at lower altitudes. The large number of RPAS/UAS will not make this possible, notwithstanding any liability aspects. The system will be supporting operations and will be able to provide sufficient data to safely execute an RPAS/UAS flight, based on the information available to it. Data required could include, but are not limited to:

1. Planned flight plans.
2. Active RPAS/UAS flight plans/missions.
3. Airspace data.
4. NOTAM (NOtice To AirMan).
5. Weather.
6. Geo-fencing.

7. Manned operations below 500 feet (AGL).

The following assumptions have been made for future ATM/UTM systems:

1. A C2 service is provided.
2. The State has executed airspace, and assessment geo-fencing is in place.
3. RPAS/UAS have surveillance capability similar concerning performance and compatible to manned aircraft surveillance capability.
4. Specific RPAS/UAS traffic management system is in place.

Note. RPAS/UTM vehicle categorization is outlined in *U-SPACE section* (sec. 2.5.1).

The classification of traffic in this airspace segment goes like follow:

Class I.: Class I traffic is primarily reserved for RPAS Category A (buy and fly). In areas of low traffic density this class can operate from the ground up to 500ft and is a subject to the following requirements:

1. Mandatory declaration of operation.
2. RPAS must be capable to self-separate in 3D.
3. VLOS operations only.
4. Geo-fencing capability which ensures that this category remains separated from no-drone zones.

Class II.: Class II traffic operates in free flight due to the nature of their operations like Surveys, filming, search and rescue and other operations that have no fixed route structure. Class II can operate from the ground up to 500 feet (AGL) and is a subject to the following requirements:

1. Mandatory authorization for operation.
2. Surveillance capability (C2 4G chip or other means).
3. VLOS and BVLOS operations.
4. Free flight Capability.
5. RPAS/UAS must be capable to self-separate in 3D.
6. BVLOS will have barometric measurement equipage.

Class III.: Class III traffic only operates in BVLOS and is mainly used for transport purposes. It can operate as free flight or within a route structure pending on the requirements set by the airspace assessment.

1. Mandatory authorization for operation.
2. Has surveillance capability.
3. BVLOS operations only.
4. Free flight or route structure.
5. Shall have barometric measurement equipage.
6. Can operate from the ground up to 500 ft.

Class IV.: Class IV traffic can operate within the layer between the ground and 500 feet. This category is designed for highly specialized operations and as such not many of these types RPAS/UAS are expected. These can be civil, state or military operations and as such:

1. Require special authorization.
2. Should be addressed on a case by case basis.
3. VLOS and BVLOS.
4. Could require surveillance capability.

IFR/VFR Operations (between 500ft - FL 600): For RPAS/UAS to fly either IFR/VFR requires that they meet the airspace requirements as set for manned aviation. These operations include airports, TMA and Enroute. For IFR capable RPAS additional requirements can be set for flying in the volumes of airspace where normal transport aircraft operate. As such it is envisaged to have minimum performance standards for elements such as speed, climb/descent speed, turn performance and latency.

Operations of Small RPAS above 500 feet: In principle operations above 500 feet by small RPAS/UAS are not allowed unless they meet the IFR/VFR airspace requirements and have a solution to be visible to manned traffic. Other aspects like wake turbulence and separation standards would also have to be addressed. However, States can still on a case by case basis accommodate RPAS/UAS above 500ft if the risk assessment of the intended operation is acceptably low.

The classification of traffic in this airspace segment goes like follow:

Class V.: Class V is IFR/VFR operations outside the Network not flying SIDs and STARs. In this environment, RPAS/UAS not meeting Network performance requirements will be able to operate without negatively impacting manned aviation. Operations at airports will be accommodated through segregation of launch and recovery.

Ground operations can also be accommodated through either towing or wing walking.

Operations from uncontrolled airports or dedicated launch and recovery sites are to be conducted initially under VLOS/VFR until establishing radio contact with ATC.

No additional performance requirements will be set in this environment compared to manned aviation.

RPAS/UAS operating in the environment will file a flight plan including information such as:

1. Type of RPAS/UAS,
2. Mission plan,
3. Contingency procedure.
4. RPAS/UAS will meet CNS airspace requirements.
5. RPAS/UAS will be able to establish two-way communication with ATC/UTM if required.
6. RPAS/UTM will remain clear of manned aircraft.
7. RPAS operator must be able to contact ATC/UTM (if required) regarding special conditions such as data link loss, emergency, controlled termination of flight.
8. RPA/UTM DAA capability will be cooperative with existing ACAS systems.

Class VI.: Class VI. is IFR operations, including Network, TMA and Airport operations with RPAS/UAS capable of flying SIDs and STARs as designed for manned operations. These are either manned transport aircraft enabled to fly unmanned with similar capabilities or new types able to meet the set performance requirements for the Network, TMA and airports. General requirements RPAS/UAS operating in this environment will file a flight plan (mission) including:

1. Type of RPAS/UAS.
2. Contingency procedure.
3. Mission plan (navigation, route, level).
4. RPAS/UAS will meet CNS airspace requirements.
5. RPAS/UAS will be able to establish two-way communication with ATC/UTM.
6. RPAS/UAS operator must be able to contact ATC (if required) in regard regarding special conditions such as data link loss, emergency, controlled termination of flight.
7. RPAS/UAS DAA capability will have ACAS functionality.

Note. The class operation class V. – VI. is covered mostly in this work.

VHL operations (Above FL 600): Suborbital unmanned flights operating at altitudes above FL 600 are expected to grow fast in numbers. Apart from military HALE RPAS, several other vehicles (i.e., space rockets, Virgin Galactic, etc.) operate through or in this block of airspace. At this moment, no management of this traffic is foreseen in most parts of the world. Particular attention should be given to the entry and exit of this high altitude volume as they need to interact with the airspace below.

The classification of traffic in this airspace segment goes like follow:

Class VII.: Class VII consists solely of IFR operations above FL600 and transiting non-segregated airspace.

These types of RPAS/UAS are solely designed for operations at very high altitudes. The launch and recovery of fixed-wing RPAS/UAS can be from dedicated airports and outside congested airspace unless Class VI requirements are met. This airspace will be shared with many different RPAS/UAS. Although their operations will not directly impact the lower airspace, however, they will have to transit through either segregated or non-segregated airspace to enter or exit the airspace above FL 600.

For such cases, temporary segregated airspace should be considered. Transition performance in segregated or non-segregated airspace below FL600 will be very limited since they will be focusing on long missions (up to several months).

The airspace in which these types of operation take place is mostly seen as uncontrolled. This requires no management of this traffic; however due to the expected numbers - estimated to be around 18000 just for Google and Facebook - it will become necessary to manage this type of operation since the performance envelopes differ a lot. Speeds can vary from average wind speed at those altitudes (for Google balloons) up to above-mach.

Launch and recovery of unmanned balloons or aircraft, together with emergency situations, will also require a set of procedures and pre-arranged coordination capabilities to ensure the safety of traffic below this altitude.

2.5.1 U-Space

The *Concept Of OpeRations of U-Space* (CORUS) [47] has been released recently. This concept describes the difference between the standard ATM and proposed European UTM solution. This section will get through the interesting part of this pivotal document.

The *U-space* is separated into following functionality based phases:

U1 (year < 2020) - sets the scene with registration and geo-fencing.

U2 (year< 2025) - introduces tracking, flight planning and messages sent to the remote pilot during flight.

U3 (year< 2030) introduces collaborative detect and avoid and tactical conflict resolution.

U4 (year < 2035) brings safe interoperation with manned aviation.

The *Aspects* important for *Obstacle avoidance* will be outlined and discussed over this section. Our work focuses on *European Airspace* (EASA); therefore more focus will be on *U-space*

Small UAS Classification: Manned aviation is covered by existing rules, for example [4, 48]. Excluding some specific situations, manned aviation does not fly below VFR airspace, do not enter *very low level* (VLL) altitudes [49].

The certified airworthiness is mandatory for *airspace attendants* with *Maximum Take-Off Mass* (MTOM) over 150kg. The other *airspace attendants* need to fulfill only *Minimal Operation Performance Specification* (MOPS).

In [50] EASA proposed several classes for UAS below 150kg MTOM; see Appendix 1 of the annex to Opinion 1-2018 entitled "...on making available on the market of unmanned aircraft intended for use in the 'open' category" and on third-country UAS operators. In that text, the next smaller mass mentioned below 150kg is 25kg MTOM. A similar break is proposed in some national legislation, for example in the UK at 20kg. As a working definition, this little chart shows a possible breakdown by MTOM. Note that EASA classes depend on many factors, not only MTOM.

| EASA class | Maximum take-off mass | Remarks |
|------------|-----------------------|--|
| C0 | $\leq 250g$ | "Child's Toy" with very limited capabilities |
| C1 | $\leq 900g$ | "Adult's Toy" small flying camera |
| C2 | $\leq 4kg$ | "Small UAS" with the package |
| C3 | $\leq 25kg$ | "Standard UAS" attainable AGL altitude $\leq 120m$ |
| C4 | $\leq 25kg$ | "Standard UAS" no automatic control mode altitude $> 120m$ |
| - | $\leq 150kg$ | "Heavy UAS", not defined in [50] |
| - | $> 150kg$ | Regulated by EASA like the manned aircraft [49] |

Table 2.3: Small UAS Classes according to EASA. [47]

Note. The class C3 and C4 are different in operational restrictions. This work focuses mainly on UAS classes C2/C3/C4 because they have enabled *automatic control mode*

Separation Minima: The *Separation Minima* defines *minimal distances* between airspace attendants to ensure secure *operation*. The separation minima are taken from Corus [47].

All ideas for safe concurrent operation of UAS are based on the idea of keeping the UAS systems apart or physically distant from some risk source.

A geo-fence, for example, is simply a method of providing separation. There will need to be separation minima for UAS just as there are for manned aircraft and these will be needed by services such as Monitoring which seeks to warn about loss of separation and Tactical Conflict Resolution which may act to avoid the loss of separation.

Separation minima will be different from those for manned aircraft as small UAS systems are generally much smaller and often slower moving than manned aircraft. CORUS proposes the following as separation minima (tab. 2.4)

| Flight Type Interaction | Horizontal | Vertical— | Remark |
|-------------------------------------|------------------------------------|------------------------------------|--|
| Any UAS - Manned or person carrying | 2.5 NM | 500 ft | Half the current manned aircraft separation |
| VLOS - VLOS | Remain "Well Clear" | Remain "Well Clear" | The remote pilot is not expected to judge distance by sight from the remote piloting position. |
| VLOS - BVLOS | Remain "Well Clear" + 200 ft | Remain "Well Clear" + 200 ft | The remote pilot is not expected to judge distance by sight from the remote piloting position. |
| BVLOS – BVLOS | 200 ft | 150 ft | Figures come from a pessimistic estimate of satellite navigation performance. |

Table 2.4: Proposed separation minima for UAS. [47]

Note. The *BVLOS – BVLOS* separation minima are interesting because the *autonomous mode* is considered as BVLOS to BVLOS avoidance in case of autonomous UAS.

The *separation minima* for *UAS - Manned aviation* is unreasonably huge (2 nautical miles), and in current it should be considered as moving constraint.

Flight Rules: The *aspects* of UAS flight rules for U-SPACE concept is summarized in the table:

| Aspect | UAS Flight Rules |
|---|--|
| Flight plan required | Yes |
| Allowed flight type | VLOS, EVLOS, BVLOS |
| Provision of separation in U1, VLOS & EVLOS | Pilot |
| Provision of separation in U1, BVLOS | Geo-fence / Geo-cage |
| Provision of separation in U2 | 1.Strategic Conflict Resolution enabled by flight planning 2.Traffic Information Service enabled by position reporting 3.Pilot (visual) |
| Provision of separation in U3 & U4 | 1.Strategic Conflict Resolution enabled by flight planning 2.Traffic Information Service enabled by position reporting 3.Cooperative Tactical Conflict Resolution 4.Detect and Avoid 5.Pilot (visual) |
| Separation from manned aviation in U2, VLOS or EVLOS UAS flight | The pilot is responsible to get the UAS out of the way of the manned aircraft. |
| Separation from manned aviation in U2, BVLOS UAS flight | Flight plan required from both. Separation by planning. BVLOS pilot should use traffic information to avoid the manned aircraft (which is tracked). |

Table 2.5: Aspects of UAS flight rules.[47]

Following *detect and avoid* requirements can be outlined based on (tab. 2.5).

1. *Separation in U1* - only *identification services* are provided in this phase. The *Detect And Avoid* support can be provided only to UAS pilot in the form of visual or sound advisories.
2. *Separation in U2* - the *position notifications* are added, enabling, preemptive collision avoidance by flight planning (mission control). The *traffic information* can be

added to pilot software for better situation awareness.

3. *Separation in U3 & U4* - the *advanced avoidance concepts*, from our perspective following aspects are interesting:
 - a *Cooperative Tactical Conflict Resolution* - The UTM infrastructure and hierarchy for *cooperative conflict resolution* must be established. In form of UTM *directives* and UAS *fulfillment*.
 - b *Detect and Avoid* - reactive obstacle/intruder avoidance and situation awareness on a very high level.
4. *Separation from Manned Aviation* - the *well clear threshold* (fig. 2.3) for manned aviation (tab.2.4) are too big. The *effective application of reactive obstacle avoidance* is not reasonable, because the manned aviation will be out of range for most sensors (except ADS-B).

Note. Our work covers *cooperative conflict resolution* and *detect & avoid*.

Geo-fencing Modes: A Geo-fencing appears in U1, U2, and U3 and is successively refined. It is supported by aeronautical information for UAS systems. This table summarizes the different features by level:

| Capability | Level | Features |
|---|-------|---|
| Pre-Tactical Geo-Fencing | U1 | Information provided before flight. The user should have access to AIP and NOTAM defined geo-fences in a form that can be used when planning and that can be loaded onto the UAS if it has geo-fence fence features in its navigation system |
| On-board Geo-Fencing | U1 | The ability of the UAS to keep itself on the correct side of a geo-fence by having geo-fence definitions (location, time, height) within its navigation system |
| Tactical Geo-Fencing | U2 | This service delivers to the pilot and /or UAS operator updates to and new definitions of Geo-Fences occurring at any time, including during flight. The creation of geo-fences with immediate effect require that they are defined outside the AIP. |
| UAS Aeronautical Information Management | U2 | U2 include a non-AIP repository of Geo-Fences. The UAS Aeronautical Information Management service includes all information coming from such a source, combined with information from the AIP and NOTAMS together with any other UAS relevant sources. |
| Dynamic Geo-Fencing | U3 | This service delivers updates definitions of geo-fences directly into the UAS, even in flight. This service relies on capabilities of the UAS in U3 to receive communications from U-space and to deal with geo-fence updates. |

Table 2.6: Geo-fencing in U-space. [47]

The *impact of geo-fencing* on *Detect and Avoid* system is following:

1. *Pre tactical* - The *flight plan* (mission) is prepared to avoid all *known forbidden areas*. In this phase, the geo-fence covers static space constraints.
2. *Onboard* - The *flight plan* (mission) specification does not contain all static space constraints. These constraints are known prior the flight. If UAS approaches such constraints, it needs to avoid them. The concept of soft constraints - restricted, but breakable space constraints emerge.
3. *Tactical* - The *space constraints* are updated during the flight. This can also be used for notifying the weather situations, restricted airspace, and all sort of *static or moving constraints*.

4. *Dynamic* - The *space constraints* the updates are real time.

Note. The work covers dynamic and tactical *Geo-fencing*.

Actively maintaining separation during flight: The standard ATM functionality [3] and *Rules Of the Air* [4, 5] are covered and to be implemented for *U-Space*.

2.5.2 NASA UTM

The *NASA UTM*⁶ is UAS Traffic Concept developed by *National Aeronautics and Space Administration* (NASA) in cooperation *Federal Aviation Administration* (FAA). The *concept* is very similar to *EASA U-SPACE*.

Note. This work is focused on *European Airspace*; the *details* will be omitted.

Useful concepts: The *NASA UTM* concept has greater maturity level concerning *Detect & Avoid* concept than European *U-Space*. There is vast amount of publications which can be used in *U-Space* from these publications the following useful studies containing DAA concepts were taken into account:

1. The non-cooperative intruder avoidance concept [51] provides a general idea about the *topic*. The *vertical separation* and *vertical encounter model* is presented.
2. The *Detect and Avoid* performance evaluation is crucial for system performance assessment. The assessment framework [52] provides us with methodological guidelines. The used concepts are abstracting the multidimensional performance criteria into simple metrics:
 - a. *Crash Distance* - the distance to the obstacle/intruder margin.
 - b. *Safety Margin* - the virtual margin around obstacle/intruder.
3. To *Ensure* the compatibility between *UAS Detect And Avoid System* and *Manned Aviation Collision Avoidance* (ACAS/TCAS) systems the following approach was proposed [53].

⁶Related research and articles: <https://utm.arc.nasa.gov/documents.shtml>

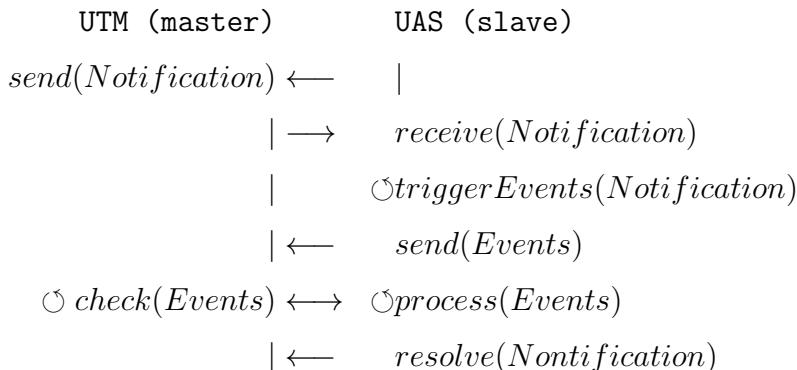
2.6 Event-Based Avoidance

Avoidance Urgency: The avoidance problem is most dependable on the *reaction time frame* or *opportunity time*. A *Reaction Time frame* can be derived from *manned aviation*:

1. *Preventive Detect & Avoid* - the parallel to a *flight plan* which is approved by *Air Traffic Control*. The purpose of *preventive avoidance* in UAS systems is to *mitigate collision risk*. The *collision mitigation* is ensured by planned route validation, airway reservations, etc. The *Preemptive Detect & Avoid* for UAS will be covered by the *UTM* mission plan acceptance procedure.
2. *Event-Based Detect & Avoid* - the parallel to *manned aviation Surveillance* and *ATC directives*. The *reaction time frame* is in minutes to a tenth of minutes. The practices can be taken from *Instrumental Flight Rules* (sec. 2.3.2). Following sources of conflicts are expected:
 - a. *UTM directive* - the directive to change heading or goal from authority, similar to *ATM directives* [3].
 - b. *UAS threat detection* - the *Surveillance* detection of a *direct impact*.
3. *Reactive Detect & Avoid* - the parallel to *manned aviation See & Avoid* or *Visual Flight Rules*. The reaction time frame is in seconds or tenths of seconds. The *avoidance maneuver* is usually to minimize the damaging impact on the environment or to preserve the *UAS* intact.

This work focus on *Event-Based* and *Reactive Detect & Avoid* level. The *preventive level* is the implementation of pre-flight procedures which are not outlined yet.

Notification Event Resolution: The *future UAS/UTM* network can be abstracted as a hierarchical agent network, where UTM is master for given *controlled airspace portion* and UAS systems are slaves. The communication principle is outlined by the following communication diagram:



The *UTM authority* (master) sends a notification about a dangerous situation or a directive for one or multiple *UAS*(slaves). The *UAS* receive *Notification/Directive*, this

can trigger multiple events on UAS side. The *triggered Events* are notified to *UTM* (master). Then *UAS* process events and *UTM* checks the outcome. Once *events* from notification are resolved, the *UAS* will send out *resolution notification*.

The presented communication schema requires to implement following mechanisms to ensure concept flexibility:

1. *UTM notification mechanism* - to notify some airspace changes it is necessary to implement a minimal detection mechanism and to send necessary data for *slave event handling* system. The *UTM* should also cover mechanisms for *fail-safe* directives fulfillment check.
2. *UAS rule engine* - The *UAS Navigation/Detect & Avoid* systems must have some level of flexibility. The processing of some events are changing with time and having a static structure of event handling is obsolete. The *rule engine* with well-defined process decision points is a modern approach to event handling.

Threat Hierarchy: The *Threat Hierarchy* depends on the *airspace type* and its decided by *rules and regulations*, sometimes by a common sense. The *hierarchy* goes like follow:

- *Controlled Airspace* - there should be no static obstacles; usually the *airspace* is controlled from flight levels where is no presence of *permanent structures*.
- *Air Traffic Control Restrictions* - there are restrictions of *airspace corridors* or *portions*. These restrictions are overall following and cannot be broken without causing a serious security incident.
- *Weather Restrictions (Critical Conditions)* - there are static or moving areas of bad weather, which impact can harm the UAS to the point of no recovery. These areas can be classified as *hard constraints*.
- *UAS Traffic Management Restrictions* - there are static or moving areas from entering by *UTM* authority. They are similar to *Air Traffic Control* restrictions. The *restricted space* is prohibited to enter by *UAS*, but it can contain or be entered by *manned aviation*.
- Note.* The pilot community statement to UAS integration is to have preferential treatment to manned aviation.
- *Non-cooperative Intruders* - the *intruding UAS* or *bird* is entering into the *controlled portion*. These need to be avoided without using *cooperative* capabilities.
- *Weather Restrictions (Breachable Conditions)* - there are weather impacted areas where the weather impact is not fatal to UAS (humidity resistance, wind resistance, improved exoskeleton). These impacted areas can be entered if its safe and cost-effective for the UAS. These type of restrictions are considered as *soft constraints* because they can be broken without significant drawback.

- *Non-Controlled Airspace* - there is an addition of *static obstacles* and *geo-fencing* threats to UAS, some of the *controlled airspace aspects* are relaxed.
 - *Static obstacles* - there is terrain and man-made structures which are considered static in UAS mission time frame. These need to be avoided with the highest priority. They are usually detected with *UAS Sensors*.
 - *Intruders* - there can be intruders who do not have an intention to harm the *UAS*. These intruders can be handled the same way as in term of *controlled airspace*.
 - *Geo-fencing* - there are important structures or natural formations which are protected against the entrance of a *UAS*. The protection zone can have different shapes and can impact different altitudes even in *Controlled Airspace*. These zones can be considered *hard constraints* or *soft constraints* depending on the situation.
 - *Weather Restrictions (Critical/Breachable Conditions)* - the weather have the same impact as in *controlled airspace*. The *weather* in non-controlled airspace can be considered as *hard constraints* or *soft constraints* depending on the situation.

Minimal Operational Data Set: The *operational equipment* should be at least on the *manned aviation* grade. The *minimal Instrumental Flight Rules (IFR)* equipment was outlined in (sec. 2.3.2). The minimal operational data set can be defined through mandatory equipment, the listing goes as follow:

1. *Precise positioning* - the precise and real-time position for UAS are mandatory for precise navigation and precise position notifications.
2. *Self Identification Service* - each UAS needs to provide own identity, sharing position information and intentions information. The unique identifier and registration are mandatory to provide UAS ownership and responsibility link.
3. *Barometric Altitude* - the precise measurement for barometric altitude is necessary to enter into *controlled airspace*. The reference barometric pressure is provided by *Air Traffic Services* for selected airport/national airspace.
4. *Terrain Sensor* - the ability to avoid static obstacles and terrain is necessary for *low altitude operations*.
5. *Transponder (Cooperative Intruders Sensor)* - a complement to *self-identification service* and *position notification*. All mentioned functionality is covered by single device ADS-B In/Out.
6. *Non-cooperative Intruders Sensor* - the identification and extraction of non-cooperative intruders (hobby UAVs, birds) is increasing overall safety.

2.6.1 Mid-Air Collision Prevention

Idea: The first fatal mid-air collision occurred in 1912. The occurrence rate increased with technological progress. The *European airspace management* is thoroughly analyzed in [54].

Mid-Air Collision Situations: The most common situations when *Mid Air Collision* occurs are:

1. *Approach* (airport landing sequence) - the traffic density is increasing with proximity to traffic hub (airport). The aircraft lower velocity in early approach phase to a critical level which significantly reduces maneuverability. The *final approach* phase is most dangerous because the aircraft is close to the ground prepare for the landing.
2. *Descent* (a decrease of altitude) - the pilot is heading plane down, the dead angle is much greater than in other situations.
3. *Cruise* (keeping the same altitude) - the pilot is keeping the altitude and heading, the awareness usually decreases significantly in this slight phase.
4. *Climb* (an increase of altitude) - the pilot is heading plane up, the dead angle is increased significantly.

Note. The *Mid-Air Collision* occurrence is strongly correlating with *traffic density*. Therefore the most of *near-miss /collision cases* happen in the vicinity of the airport. It is expected to elevate the risk of *Mid-Air Collision* by enabling *UAS* into *B, C, D, airports class* airspace.

Collision Situation Awareness: The surveillance capability of manned aviation is limited by the *pilot's field of vision* in case of VFR (sec. 2.3.1) and by *technical limitations* in case of IFR (sec. 2.3.2). The *surveillance and avoidance* support systems like TCAS (sec. 2.4.3) and ACAS (sec. 2.4.2) can be used as a base of future *DAA* system.

The *UAS* collision situation awareness system is taking the *mid-air* collision prevention to another level; additional functionality needs to be implemented:

1. *Intruder trajectory prediction* - anticipate future *intruder actions* based on gathered knowledge. Recognize dangerous actions or triggering situations. The pilot in manned aviation processes the triggering events of future dangerous situations.
2. *Intruder intersection model* - anticipate future maneuverability and physical properties (turbulence, body size) in intersection model of the intruder and try to estimate an impact area in future. This process is done by the pilot in manned aviation. Some supplementary information like *aircraft type* and some properties are provided by the surveillance system. The final decision and estimate need to be automatized in the form of impact probability or impact rating.

3. *Decision-making process* - the situation assessment gives an outline of the surrounding space properties, this process is well covered by *surveillance*. The decision-making process needs to be flexible and adaptable, but the limited computational resources need to be taken into account (discretization problem).

Note. The example of *enclosed operational space* which is necessary for *intruder intersection* detection is given in [55].

2.6.2 Weather Impact

Idea: The *climate* has stable properties throughout the year. There are observations that climate is shifting in Europe, the periods of sprint/autumn weather are shortening, the periods of summer and winter are prolonging.

Overall the *European Climate* is getting similar to *North America's* continental climate. This has a severe impact on many aspects of modern society including transportation, especially aviation, emerging UAS industry.

The key fact is that the occurrence of critical weather conditions, like storms or heavy winds is increasing. Along with the increasing intensity of these events, the magnitude increases to non-construction mitigable levels.

Transportation Impact: The *train transportation* is the most robust and very infrastructure dependant transportation type. On the other hand, the *aerial transportation* infrastructure is sparse, and most of the maneuvering is done in open airspace.

Weather Impact on transportation, in general, has been introduced by Koetse in the study [56]. The *bad weather* situations are well avoidable by *general aviation*. The *general aviation* avoidance capability comes from the *high organization* of controlled *airspace*, high grade of surveillance equipment (weather radar).

The situation with *UAS* systems is different; they usually have more delicate construction and significantly smaller takeoff weight. The *weather* could be even harsher on the lower altitudes. The *implementation of weather avoidance* is necessary for *safe UAS operations* in non-controlled airspace. The *UAS operations* can stick to existing weather avoidance approaches in controlled airspace.

The *challenge* to avoid weather situations is similar to the geo-fencing problem on low altitudes. The *serious weather case* can be encapsulated into a protected area with some *altitude limitations*.

Weather avoidance: *Weather-based* preemptive planning was introduced into manned aviation in 2015 [57]. There is a global approach to weather avoidance. Meaning there is one global *weather model*. Similar impact on every *controlled airspace* attendant. This impact model needs to be refined for various *aircraft classes*.

Note. The *UAS classes* are summarized in (tab. 2.3), the *separation minima* is specified in (tab. 2.4). This needs to be accounted in weather impact calculation.

The *separation minima* are also accounted for *weather separation*. The *UAS* must keep minimal distance to *dangerous weather condition*.

The radius and impact zone of *dangerous weather condition* is evaluated concerning *UAS* class; the 5 kg machine is more impacted than 150 kg machine or *autonomous personal transportation*.

Severe Weather Condition Detection Capabilities for the current level of standard aviation equipment have been reviewed in [58]. The capability is sufficient for medium scale *UAS* (25 kg). The *numeric models* for *local airspace cluster* can have precision up to one cubic meter of air. The precision of *numeric prediction* depends on *weather stations* density and equipment precision.

The *dynamic routing* of *aircraft* (manned aviation) has been outlined in Balaban et al. [59]. The operational space is separated into *altitude layers* where each layer is separated into homogeneous euclidean grid cells. Each cell (fig. 2.7) has an evaluation of *relative humidity* (fig. 2.7a), *temperature*, *wind velocity and heading* (fig. 2.7b). There is possible to make a prediction and route all aircraft in advance. The scaling challenge also remains for this approach.

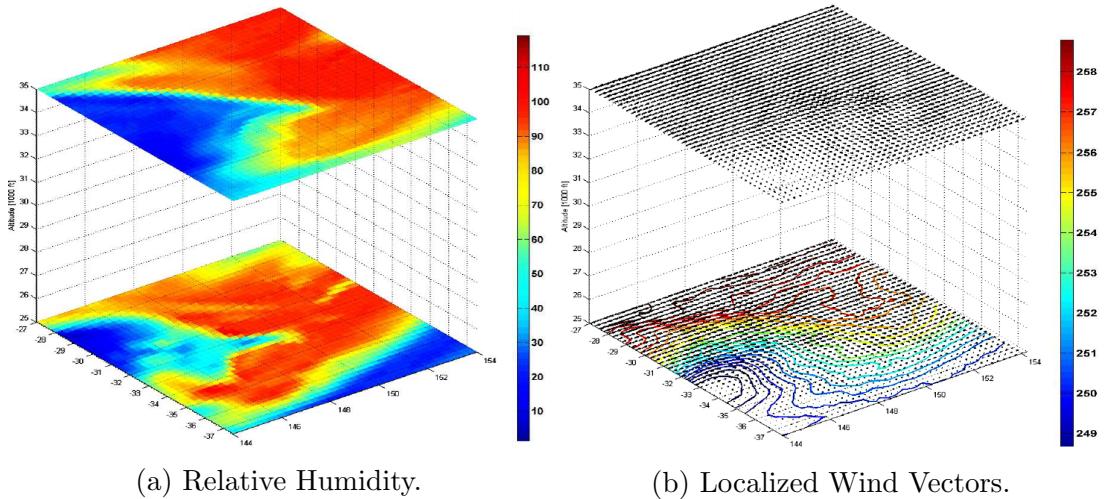


Figure 2.7: Localized Weather Model [59].

Localized Weather Impact Example: An *Icing Risk* in a localized environment have been predicted based on the numerical model [60]. It is shown that icing can be prevented by *planned* and *reactive* avoidance. The prevention can be done by placing hard or soft constraint into an environment.

Weather Models: *Weather Models* can be extracted from *Climate and weather model archive at the National Oceanic and Atmospheric Administration* [61]. There is an example of troposphere winds (0 - 60 000 feet AMSL) (fig. 2.8) which can be useful in fuel-efficient route planning.



Figure 2.8: Example of upper troposphere winds [61].

Chapter 3

Background Theory

Motivation: Cooperative and Non-Cooperative *Sense and Avoid* (SAA) systems are key enablers for the *Unmanned Aerial Systems* (UAS) to routinely access non-segregated airspace [62]. Both cooperative and non-cooperative SAA systems are being developed to address this integration requirement.

The *DAA capability* is defined as the automatic detection of possible conflicts by the UAS platform under consideration and performing avoidance maneuvers to prevent the identified collisions. An analysis of the available DAA candidate technologies and the associated sensors for both cooperative and non-cooperative SAA systems is presented in [63].

Non-cooperative *Collision Detection and Resolution* (CD&R) for UAS is considered as one of the major challenges that need to be addressed [64] for the insertion of UAVs in non-segregated air space. As a result, many non-cooperative sensors for the SAA system have been adopted. Light Detection and Ranging (LiDAR) is used for detecting, warning and avoiding obstacles for low-level flying [27].

An approach to the definition of encounter models and their applications to SAA strategies is presented in [65] for both cooperative and non-cooperative scenarios.

Since 2014, there is a visible strong political support for developing rules on drones, but regulations are harmonizing slowly. The European Aviation Safety Agency (EASA) has been tasked to develop a regulatory framework for drone operations and proposals for the regulation of "low-risk" UAS operations. In achieving this, EASA is working closely with the Joint Authorities for Regulation of Unmanned Systems (JARUS) [7].

Background Areas: Following Areas are introduced in this chapter:

1. *UAS System Model* (sec. 3.1) - continuous and discrete mathematical models.
2. *Reach Sets* (sec. 3.2) - introduction to representation and calculation methods.
3. *Hybrid Automaton* (sec. 3.3) - intuitive definition of the *hybrid automaton*.
4. *LiDAR* (sec. 3.4) - a summary of *LiDAR* technology and terminology introduction.

3.1 UAS System Model

This section strongly follows [66].

3.1.1 Continuous-time Systems

Consider a class of systems given by functions:

$$\begin{aligned} StateEvolution : & input(time) \rightarrow state(state_0, time) \\ & input(time) : [0, FinalTime] \rightarrow \mathbb{R}^p \\ & input(time) \in \mathbb{R}^p, state(time) \in \mathbb{R}^n \end{aligned} \quad (3.1)$$

Where $input(time)$ and $state(state_0, time)$ are sets of continuous-time signals. These are often called continuous-time systems because they operate on continuous-time signals.

Frequently, such systems can be defined by differential equations that relate the input signal to the output signal.

A prototypical description of a controlled (there is a control input signal) continuous-time system is:

$$\begin{aligned} \frac{d}{dt} state(time) = \\ f(time, state(time), input(time)), input(time) \in Inputs(time) \end{aligned} \quad (3.2)$$

Where $f : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^p \rightarrow \mathbb{R}^n$ satisfies the conditions for existence and uniqueness of the ordinary differential equation and u is our control [67].

3.1.2 Discrete-time Systems

Consider another class of systems given by functions

$$\begin{aligned} StateEvolution : & input(k) \rightarrow state(k), \\ & k \in \{0, t_s, 2.t_s, 3.t_s, \dots i.t_s\}, i \in \mathbb{N}^+ \\ & input(k) \in \mathbb{R}^p, state(k) \in \mathbb{R}^n \end{aligned} \quad (3.3)$$

Where $input(k)$, $state(k)$ is a set of discrete-time signals. They can be represented by a function f like $f : \{0, t_s, 2.t_s, 3.t_s, \dots i.t_s\} \rightarrow \mathbb{R}^n, i \in \mathbb{N}^+$ where t_s is sampling time and i is discrete step [68].

3.1.3 Adversarial Behavior in Continuous-time Systems

Consider a subclass of continuous time systems where are two sets of control signals $uas(\text{time})$ and $adversary(\text{time})$ which are accommodated in following system:

$$\begin{aligned} \frac{d}{dt} \text{state}(\text{time}) &= f(t, \text{state}(\text{time}), uas(\text{time}), adversary(\text{time})), \\ uasControl(\text{time}) &\in UASInputs(\text{time}) \subset \mathbb{R}^u, \\ adversaryControl(\text{time}) &\in AdversaryInputs(\text{time}) \subset \mathbb{R}^v \end{aligned} \quad (3.4)$$

This system representation is often used in definition of problem of pursuit/evasion problem. Krasovskii developed a solution approach to this problem in [69]. A complex example of can be found in article [70].

3.2 Reach Sets

Informally, the *Reach Set* of a system described by a differential equation is the *set of all states that can be reached from an initial state within a given time interval*. Similar definitions applies to the systems with different representation and control, such as *hybrid automaton*.

3.2.1 Definitions

For following definitions consider *nonlinear UAS system* described in (sec. 3.1).

Definition 1 (Reach set starting at a given point). *Suppose the initial position and time $(\text{state}_0, \text{time}_0)$ are given. The reach set $\text{ReachSet}[\tau, \text{time}_0, \text{state}_0]$ of nonlinear system at time $\tau \geq \text{time}_0$, starting at $(\text{state}_0, \text{time}_0)$ is given by:*

$$\text{ReachSet}[\tau, \text{time}_0, \text{state}_0] = \bigcup \{\text{state}(\tau) : \text{input}(s) \in \text{Inputs}(s), s \in (\text{time}_0, \tau]\} \quad (3.5)$$

Reach set starting at given set can be used to determine reach set in case of *hybrid system* input control switch and it is defined as follow:

Definition 2. *set starting at a given set] The reach set at time $\tau > t_0$ starting from set $States_0$ is defined as:*

$$\text{ReachSet}[\tau, \text{time}_0, States_0] = \bigcup \{\text{ReachSet}[\tau, \text{time}_0, \text{state}_0] : \text{state}_0 \in States_0\} \quad (3.6)$$

Reach set for adversarial behavior can be used to calculate possible escape routes from pursuer and it is defined as follow:

Definition 3 (Reach set under adversarial behavior). *Consider now the case of adversarial behavior([69, 70]). where $\text{input}(t)$ is our control and $\text{adversary}(t)$ is adversary control which is independent of $\text{input}(t)$, let $\text{differentialControl}(t) = \text{input}(t) - \sup_{\text{state} \in \text{state}(t)} \text{adversary}(t)$, which represents worst possible input change in given state and time, then reach set for system is represented as:*

$$\text{ReachSet} \left[\begin{array}{c} \tau, \\ \text{time}_0, \\ \text{state}_0 \end{array} \right] = \bigcup \left\{ \text{state}(\tau) : \begin{array}{l} \text{differentialControl}(s) \in \text{DifferentialControlSet}(s), s \in (\text{time}_0, \tau] \\ \text{differentialControl}(s) \in \text{DifferentialControlSet}(s), s \in (\text{time}_0, \tau] \end{array} \right\} \quad (3.7)$$

Reach set under state constraints are usable to define state constrained systems in terms of dynamics and technical capabilities.

Definition 4 (Reach set under state constraints). *Suppose the initial position and time $(\text{state}_0, \text{time}_0)$ and state constraints are given $\text{state}(t) \in \mathbb{A} \subset \mathbb{R}^n, \dot{x}(t) \in \mathbb{B} \subset \mathbb{R}^n$. The reach set $\text{ReachSet}[\tau, \text{time}_0, \text{state}_0]$ of nonlinear UAS system at time $\tau \geq \text{time}_0$, starting at position and time $(\text{state}_0, \text{time}_0)$ is given by:*

$$\text{ReachSet} \left[\begin{array}{c} \tau, \\ \text{time}_0, \\ \text{state}_0 \end{array} \right] = \bigcup \left\{ \text{state}(\tau) : \begin{array}{l} \forall s \in (\text{time}_0, \tau], \text{state}(s) \in \mathbb{A}, \\ \dot{\text{state}}(s) \in \mathbb{B}, \\ \exists \text{input}(s) \in \text{Inputs}(s) \end{array} \right\} \quad (3.8)$$

3.2.2 Computation of Reach Sets

Several techniques for reachability analysis of systems have been proposed. They can be (roughly) classified into two kinds:

1. Purely symbolic methods based on:
 - a. the existence of analytic solutions of the differential equations and
 - b. the representation of the state space in a decidable theory of the real numbers.
2. Methods that combine
 - a. numeric integration of the differential equations
 - b. symbolic representations of approximations of state space typically using (unions of) polyhedra or ellipsoids.

These techniques provide the algorithmic foundations for the tools that are available for computer-aided verification of hybrid systems ([71], [72], [73]).

The set-valued Lebesgue integral provides a conceptual tool for the direct computation of the reach set. In what follows we describe techniques from dynamic optimization which are used to compute reach sets for dynamic systems.

The relation between dynamic optimization and reachability was first observed in [74]. A typical problem of optimal control can be formulated as follows:

$$\max \left(\int_{initialTime}^{finalTime} cost(time, state(time), control(time)) dt + \dots \right) \\ \dots + FinalCost(state(finalTime)) \quad (3.9)$$

For nonlinear system:

$$\dot{state}(t) = f(t, state(t), control(t)), control(t) \in ControlSet(t) \subset \mathbb{R}^p \quad (3.10)$$

Where *cost* is given as a cost function of time, state and input and *FinalCost* represent cost functional.

There are two main techniques to solve this problem, the maximum principle, and dynamic programming.

The maximum principle gives necessary conditions of optimality. Dynamic programming may be used to derive sufficient conditions of optimality.

A good reference on the maximum principle is [75]. A less known reference with detailed geometric interpretations is [76]. A good reference on dynamic programming is given in [77].

3.3 Hybrid Automaton

First, the notion of *hybrid* automaton [71, 78, 79] needs to be introduced:

Definition 5. *Hybrid automaton (3.11) is given as structure:*

$$HybridAutomaton(AutomatonStates, SystemState, VectorField, \\ DiscreteTransition, ResetMap) \quad (3.11)$$

The automaton States is given as a set of discrete states, for every time time $\in Domain$ hybrid automaton stays in exactly one of the states.

System State is given in domain $x \in \mathbb{R}^n, n \in \mathbb{N}^+$, representing the trajectory evolution.

emphVector Field (3.12) is bounded to single AutomatonState and represents local System State evolution when given automaton State is Active.

$$VectorField : AutomatonState \times SystemState \rightarrow SystemState \quad (3.12)$$

DiscreteTransition (eq. 3.13) indicates changes of states in the automaton; the changes

are triggered by satisfying the specific condition given by Automaton State and System State.

$$\text{DiscreteTransition} : \text{AutomatonState} \times \text{SystemState} \rightarrow \text{AutomatonState} \quad (3.13)$$

ResetMap (eq. 3.14) defines changes of State to some default value, specific automaton State and System State triggers this change.

$$\text{ResetMap} : \text{State} \times \text{SystemState} \rightarrow \text{SystemState} \quad (3.14)$$

Hybrid Automaton Example: An example of a *hybrid automaton* is given in (fig. 3.1). The automaton is used to control *UAS system* to perform simple level up (increase altitude) maneuver [80]. The automaton has three discrete states representing *hover*, *transition*, and, *level* portion of the maneuver.

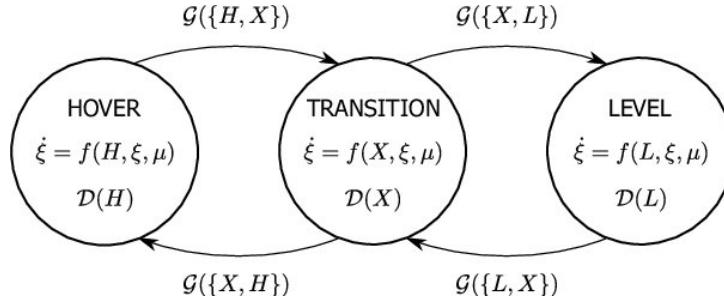


Figure 3.1: Example: The hybrid automaton example for UAS maneuver [80].

3.4 LiDAR

LiDAR(Light Detection And Ranging) is an active form of remote sensing: information is obtained from a signal which is sent from a transmitter and reflected by a target and detected by a receiver back at the source. Following types of information can be obtained:

1. *Range to target* - topographic LiDAR or laser altimeter.
2. *Chemical properties of target* - differential absorption LiDAR.
3. *The velocity of target* - Doppler LiDAR.

Chemical properties of the target are out of scope. The velocity of target seems as interesting property to investigate, but this type of LiDAR is usually used for meteorological measurements of wind currents [81]. Extended research in LiDAR as obstacle detection sensor has been executed by research group around Sabatini [27] and Ramasy [28].

A LiDAR output is represented as point cloud it is described by the definition:

Definition 6 (Scanned point and Point-cloud). Consider viewpoint as the origin of \mathbb{R}^3 space, Let $\text{point} \in \text{PolarCoordinates}$ be defined as:

$$\text{point} = [\text{distance}, \text{horizontal}^\circ, \text{vertical}^\circ, \text{time}]^T \quad (3.15)$$

Where horizontal° is the horizontal angle from the origin, vertical° is a vertical angle to the origin, and, time is a time of retrieval.

Point-cloud is set of points scanned in small enough time-frame, based on processing raw point data it can have following representations:

1. Local point-cloud - position of the sensor is used as the origin of space and points can be represented in orthogonal or planar representation.
2. Global point-cloud - global position of the sensor is used as a reference to calculate the global position of points.

Point-cloud is usually addressed as *raw point-cloud* in case if it is represented in Local planar coordinates. Other forms of point cloud require further processing, and they are not feasible for real-time obstacle detection and avoidance [82].

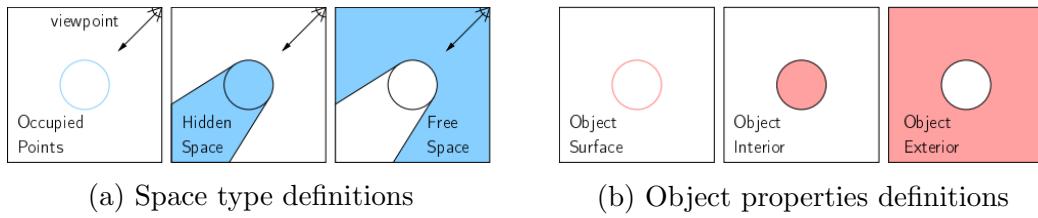


Figure 3.2: Six space classifications [83].

Because of real-time obstacle avoidance, it is necessary to introduce the following terminology:

1. *Occupied points* - points which have been detected by LiDAR (also addressed as visible points).
2. *Hidden space* - space which is hidden behind occupied points, from the viewpoint it is uncertain what is in that space.
3. *Free space* - space which is visible from viewpoint and it is not occupied by known objects.
4. *Object surface* - detected and undetected object surface
5. *Object interior* - occupied space by the object.
6. *Object exterior* - free space around known objects.

The existing method for space segregation [83] leads to the following definition:

Definition 7 (Accessible space). *Consider known space as space explored by sensor (it can have different viewpoint along previous 3D trajectory). The intersection between object exterior (Exterior) and free space Free gives us Accessible space (Accessible).*

$$\text{Accessible} = \text{Exterior}(\text{object}) \cap \text{Free}(\text{object}) \quad (3.16)$$

Accessible space S_A (def. 7) is our bordering limitation for reachable space of system $\text{ReachSet}[\tau, \text{time}_0, \text{state}_0]$ (def. 1.).

Chapter 4

Problem Statement

A *UAS* equipped with several types of sensors is tasked to fly several types of *Missions* in a 3-dimensional space. There is a *Terrain* map, an *Object map*, and a *Weather* forecast for target region that are known a priori. The map of the *Terrain* may not be up to date, and *Uncharted obstacles* may affect flight safety. The *UAS* has to comply with a set of *Flight Rules* specifying *Flight Constraints*. The performance of the *UAS*, including sensor performance, is affected by the *Weather*.

Several difficulties must be faced. First, the design space of is large and complex. Second, *Trajectory calculation* Third, *Navigation*.

4.1 Basic Definitions

A few definitions are needed.

4.1.1 World

The world of interest consists of:

Space has a Global Reference Frame with three axes X^+, Y^+, Z^+ .

$$Space \subseteq \mathbb{R}^3 : \text{main axes: } X^+, Y^+, Z^+, \text{ center } [x_0, y_0, z_0] \quad (4.1)$$

Object (4.2) is a generic subset of *Space* which has *Boundary*.

$$Object \subset Space : \forall point \in Object, point \text{ is solid} \quad (4.2)$$

A nonlinear first order state-space model gives the dynamic model of the UAS:

$$\dot{x} = f(t, x, u) \quad (4.3)$$

Where $x \in \mathbb{R}^{6+n}$, $n \in \mathbb{N}^{0+}$ is *system state* containing minimal information about UAS position $[x, y, z]$ and orientation $[\theta, \varphi, \rho]$ (roll, pitch yaw), and $u(t)$ is *control signal* belonging to R^k , $k \in \mathbb{N}^+$, bounded by control set $u(t) \in U$.

The map of the terrain is given by *TerrainMap* mapping (x, y) to the terrain elevation

$$\text{TerrainMap} : (x, y) \rightarrow z \quad (4.4)$$

Weather (4.5) can impact the performance of the *UAS*. *Wind* can decrease flying capabilities and maneuverability, *visibility* can impair the performance of sensors such as LiDAR and cameras, *humidity* can be a serious danger to electronic equipment, and in combination low *temperature* it can cause icing on the UAS body, and *air pressure* can impact ranging sensor and flight performance.

$$\text{Weather} : (\text{position} \in \text{Space}) \times \text{time} \rightarrow \begin{bmatrix} \text{wind } (w_x, w_y, w_z) \\ [m/s, m/s, m/s] \\ \text{visibility } v [m] \\ \text{humidity } h [0 - 100\%] \\ \text{air pressure } a [hPa] \\ \text{temperature: } \tau [C] \end{bmatrix} \quad (4.5)$$

4.1.2 Mission

Mission (4.6) which UAS should fly is given as a set of *ordered, feasible regarding UAS dynamic* (4.3), *waypoints* in a subspace of *Space* (4.1).

$$\text{Mission} = \left\{ \text{waypoint}_1, \text{waypoint}_2, \dots, \text{waypoint}_m : \begin{array}{l} \forall_{i=1 \dots m} \text{waypoint}_i \in \text{Space} \end{array} \right\}, \quad m \in \mathbb{N}^+, m \geq 2 \quad (4.6)$$

Waypoint Passing (4.7) function maps system (4.3) trajectory *projection in Space* and *Mission* waypoints (4.6) to a vector of *passing times*.

$$\text{WaypointPassing} : \text{TrajectoryProjection} \times \text{Mission} \rightarrow \text{Time}^m \quad (4.7)$$

Note. The *Mission* (4.6) is considered as completed if and only if \forall waypoints are reached and in the given order (check the output of 4.7).

4.1.3 Flight Constraints

Flight constraints arise from aviation rules and ATM/UTM.

There are H sets of hard flight constraints expressed as subsets of *Space*:

$$H\text{FlightConstraints} = \{H\text{FlightConstraint}_1, \dots, H\text{FlightConstraint}_H\} \quad (4.8)$$

The union of all $H\text{FlightConstraint}_i$ is the set *HFlightConstraint*. *Example:* turn-

ing radius of UAS, climb rate, etc.

There are S sets of soft flight constraints expressed as subsets of $Space$:

$$SFlightConstraints = \{SFlightConstraint_1, \dots, SFlightConstraint_S\} \quad (4.9)$$

The union of all $SFlightConstraint_i$ is the set $SFlightConstraint$. Example: sharp maneuvers, restricted flight areas etc.

The *Weather* may impose either soft or hard flight constraints.

4.1.4 Partition of Space

In what follows it is convenient at each time t to partition the $Space$ into four disjoint subsets $Free(t)$, $Restricted(t)$, $Occupied(t)$ and $Uncertain(t)$.

There is a *SpaceClassification* function (4.10)

$$SpaceClassification : y \in Space \mapsto s \in \{Free, Restricted, Occupied, Uncertain\} \quad (4.10)$$

Note. *SpaceClassification* (4.10) is a *total* function mapping each element of $Space$. Other followup *classifications* are considered as *total* functions.

There is set of *Space Hard constraints*, where one *HardConstraint* (4.11) is given as:

$$HardConstraint = \{point \in Space : SpaceClassification(point) = Occupied\} \quad (4.11)$$

There is set of *Space Soft constraints*, where one *SoftConstraint* (4.12) is given as:

$$SoftConstraint = \{point \in Space : SpaceClassification(point) = Restricted\} \quad (4.12)$$

4.1.5 Information Source

Definition 8. *Information source* (4.13) represents *a priori information* for each point in $Space$ at time *prior mission execution* mapping it into one of distinctive categories Free, Occupied, Restricted, and Uncertain.

$$InformationSource = SpaceClassification(PriorTime) \quad (4.13)$$

Definition 9. *Landmark* is a partition of $Space$ (4.1) which is notable in the context of society or law given properties. *Landmark* can be notable buildings, notable natural structures or crucial infrastructure. *Landmark* is part of terrain map, but it's special status can induce additional properties.

For obstacle avoidance problem following *Information sources* are available:

1. *Terrain map* - a map of terrain with notable landmarks.
2. *Object map* - a map of notable structures with *protection zones* considered as *occupied* or *restricted* space.
3. *Fly zones restriction map* - a map of restricted flight areas considered as *restricted* constraints.

4.1.6 Single Observation by Single Sensor Classification

The *UAS* is equipped with single *Sensor*, which returns *Space* classification for given *observation position* and observation time.

Observation (4.14) at given UAS *position, time*, and for given *sensor* sorts points from *sensor reading* into following distinguish sets (4.14):

1. $Free_O(position, sensor, time)$ - observable space by sensor and considered as *Free* by sensor reading,
2. $Occupied_O(position, sensor, time)$ - observable space by sensor and considered as *Occupied* by sensor reading,
3. $Uncertain_O(position, sensor, time)$ - all other points.

$$Observation(position, sensor, time) \rightarrow \begin{cases} Free_O(position, sensor, time) \\ Occupied_O(position, sensor, time) \\ Uncertain_O(position, sensor, time) \end{cases} \quad (4.14)$$

4.1.7 Multiple Observations by Single Sensor Classification

Let add pairs of *observation position* and *observation time* in manner $\{(position_1, t_1), (position_2, t_2), \dots (position_k, t_k)\}$, $k \in \mathbb{N}^+$ that point position is independent and time is ordered in fashion $t_1 < t_2 < \dots < t_k$.

Free space from multiple sensor *reading* over multiple *positions* is inclusive space because we are obtaining additional information regarding space reachability by sensor reading independent on sensor space and orientation limitation. Therefore the union of single instances of observations is used to represent *Combined free space* $Free_O(sensor)$ (4.15).

$$Free_O(sensor) = \bigcup_{i=\{1, \dots, k\}} Free_O(sensor, position_i, time_i) \quad (4.15)$$

Occupied space (4.16) from multiple sensor *reading* over multiple *positions* is inclusive space, because of increased information; therefore it has similar handling to *Free space*.

$$\text{Occupied}_O(\text{sensor}) = \bigcup_{i=\{1, \dots, k\}} \text{Occupied}_O(\text{sensor}, \text{position}_i, \text{time}_i) \quad (4.16)$$

Uncertain space (4.17) from multiple sensor *reading* over multiple *positions* is exclusive space, because of decrease in uncertainty.

$$\text{Uncertain}_O(\text{sensor}) = \bigcap_{i=\{1, \dots, k\}} \text{Uncertain}_O(\text{sensor}, \text{position}_i, \text{time}_i) \quad (4.17)$$

4.1.8 Sensor Fusion

The observation at fixed time t_{fix} can be made by multiple sensors $\text{sensor}_1, \text{sensor}_2, \dots, \text{sensor}_k$, $k \in \mathbb{N}^k$, for k sensors execute observations Observation_1 ($\text{sensor}_1, \text{position}_1, t_{fix}$), Observation_2 ($\text{sensor}_2, \text{position}_2, t_{fix}$), \dots , Observation_k ($\text{sensor}_k, \text{position}_k, t_{fix}$).

Sensor Fusion (4.18) function with *data fusion parameters* is introduced which combines *Observations* for each *sensor* and *Weather*, then uniquely maps each point into four distinguish sets: $\text{Free}(t_{fix})$, $\text{Occupied}(t_{fix})$, $\text{Restricted}(t_{fix})$, and $\text{Uncertain}(t_{fix})$ (special case of (4.10)).

$$\text{SensorFusion} : \left[\begin{array}{l} \text{Observation}_1(\text{sensor}_1, \text{position}_1, t_{fix}) \times \\ \text{Observation}_2(\text{sensor}_2, \text{position}_2, t_{fix}) \times \\ \times \dots \times \\ \text{Observation}_k(\text{sensor}_k, \text{position}_k, t_{fix}) \times \\ \text{Weather}(\dots) \\ \text{fixed time } t_{fix} \times \\ \text{sensorFusionParameters} \end{array} \right] \rightarrow \left\{ \begin{array}{l} \text{Free}(t_{fix}) \\ \text{Occupied}(t_{fix}) \\ \text{Restricted}(t_{fix}) \\ \text{Uncertain}(t_{fix}) \end{array} \right\} \quad (4.18)$$

4.1.9 Data Fusion

Multiple *Information Sources*: $\text{InformationSource}_1, \text{InformationSource}_2, \dots, \text{InformationSource}_l$, $l \in \mathbb{N}$, where l is the count of information sources needs to be fused with *Sensor Fusion* function (4.18) outcome at *fixed time* t_{fix} .

Data fusion function (4.19) combines the classification from various *Information Sources* with classification from *Sensor Fusion* function (4.18) for UAS position at *fixed time* t_{fix} , under given *Data Fusion Parameters*.

$$\begin{array}{c}
 DataFusion : \\
 \left[\begin{array}{l}
 InformationSource_1 \times \\
 InformationSource_2 \times \\
 \times \cdots \times \\
 InformationSource_l \times \\
 SensorFusion(\dots) \times \\
 Weather(\dots) \\
 position \times \\
 fixed\ time\ t_{fix} \times \\
 dataFusionParameters
 \end{array} \right] \rightarrow \left\{ \begin{array}{l}
 Free(t_{fix}) \\
 Occupied(t_{fix}) \\
 Restricted(t_{fix}) \\
 Uncertain(t_{fix})
 \end{array} \right\}
 \end{array} \quad (4.19)$$

Each *point* in *Space* is uniquely classified into one of sets $Free(t_{fix})$, $Occupied(t_{fix})$, $Restricted(t_{fix})$, $Uncertain(t_{fix})$ (special case (4.10)).

Note. Moreover *Data Fusion* function is covering the case of *Multiple information sources, combined with multiple sensor readings over multiple times*, including *Weather* as *sensor impact factor* and *information source*.

4.1.10 Known World

Known world (4.20) for some *fixed time* t_{fix} is given as the joint set of points which belongs to one of *Data Fusion* (4.19) output sets $Free(t_{fix})$ or $Occupied(t_{fix})$ or $Restricted(t_{fix})$. The *Known World* is compact set with existing boundary.

$$KnownWorld(t_{fix}) = Free(t_{fix}) \cup Occupied(t_{fix}) \cup Restricted(t_{fix}) \quad (4.20)$$

4.1.11 Safety Margin

Let say that *mission* was executed in *time* interval $t \in [missionStart, missionEnd]$ in *Known world* (4.20). For every *position*, extracted from *UAS model state* $x(t)$, keeps distance from any point in $Occupied(t)$ with greater or equal to *safetyMargin* (s_m) (4.21).

$$\forall t \in [missionStart, missionEnd] :$$

$$distance(x(t), Occupied(t), t) \geq safetyMargin \quad (4.21)$$

4.2 Basic Obstacle Avoidance Problem

Given:

1. Initial system state x_0 , for UAS model (4.3).
2. Mission (4.6) to be executed.
3. Space (4.1), with existing objects (4.2).
4. Sensor system $\{sensor_1, sensor_2, \dots, sensor_k\}$ with existing sensor fusion function (4.18).
5. Weather information (4.5) during the flight is available.
6. Information sources $informationSource_1, informationSource_2, \dots, informationSource_l$ containing hard (4.11) and soft space constraints (4.12), with existing data fusion function (4.19).
7. Hard (4.8) and Soft 4.9 flight constraints given by ATM and rules of the air.

Generate *Control command chain* to complete a mission (4.6) with the satisfaction of following conditions:

1. Waypoint passing function condition (4.7).
2. Set safety margin s_m to *Occupied* space in $KnownWorld(time)$ (4.20) is not breached at any time (4.21).

4.3 Initial Assumptions

Initial assumptions are the following:

Assumption 1. *Filtered sensor readings are available.*

SensorObservation (4.14) for a given position, time returns classification of Space which is corresponding with the real situation.

Assumption 2. *There are no moving obstacles.*

The initial Space Classification Function (4.10) is static for all observation times $t \in (-\infty, \infty)$. Moreover, there are no intruders or adversaries present.

Assumption 3. *The movement takes place in the unrestricted airspace.*

Assumption 4. *The mission consists of a set of reachable waypoints.*

For specific UAS system (4.3) and Mission (4.6), there exists a control which satisfies Waypoint passing (4.7) criterion and SafetyMargin (4.21) condition.

Assumption 5. *The UAS is moving with constant velocity.*

For given UAS system (4.3) there is a subset of state $velocity(t) \subset x(t)$ which contains velocity parameters. Then there exist transformation function $LinearVelocity(\circ)$ which maps $velocity(t)$ to linear velocity $\in \mathbb{R}^1$. For time t in missionStart and missionEnd in Mission (4.6) constraint (4.22) with some $constantVelocity \in \mathbb{R}^+$ holds.

$$\forall t \in \begin{bmatrix} missionStart, \\ missionEnd \end{bmatrix} : LinearVelocity(velocity(t)) = constantVelocity \quad (4.22)$$

Note. Initial assumptions 1., 2., 3., 4, and 5. will be relaxed in Incremental problem definition.

4.4 Incremental Problem Definition

This section contains *incremental problem definition* as increments of (sec. 4.2). Each problem contains definition and references to addressed issues.

Problem 1. *Basic Avoidance (sec. 4.2) is to navigate through KnownWorld under the assumption that every waypoint in Mission is reachable. The KnownWorld is fed through SensorFusion function which is joining LiDAR scanning into Free(t), Occupied(t), and, Unknown(t) sets in discrete scan times t.*

$$\begin{aligned}
 KnownWorld &:= SensorFusion(t) \forall point \in KnownWorld(t) \\
 &= Free(t) \cup Occupied(t) \cup Unknown(t) \\
 Mission &:= \forall waypoint \in Mission \text{ are reachable} \\
 Sensors &:= \{LiDAR\} \\
 SensorFusion &:= \{Clasificaiton function\} \\
 HFlightConstraints &:= \{vehicle dynamic\}
 \end{aligned} \tag{4.23}$$

Challenges for problem 1. :

1. Navigation Loop Implementation (sec. 6.6.2).
2. Avoidance Loop Implementation (sec. 6.6.1).

Problem 2. *Intruder Problem in addition to Known world evolution (pr.1) the ADS-B sensor is introduced into Sensors array, this imposes HardConstraint of Flight corridor for detected intruders impacting the evolution of Free(t), and Occupied(t) sets significantly.*

$$\begin{aligned}
 KnownWorld &:= SensorFusion(t) \forall point \in KnownWorld(t) \\
 &= Free(t) \cup Occupied(t) \cup Unknown(t) \\
 Mission &:= \forall waypoint \in Mission \text{ are reachable} \\
 Sensors &:= \{LiDAR, ADS - B\} \\
 SensorFusion &:= \{Advanced joint sets\} \\
 HFlightConstraints &:= \{vehicle dynamic\} \\
 HardConstraints &:= \{intruder corridors\}
 \end{aligned} \tag{4.24}$$

Challenges for problem 2. :

1. Intruder Intersection Models (*minimal operation requirements achieved*): Linear Intersection Model (app. C.1), Body-volume intersection (app. C.2), Maneuverability uncertainty intersection (app. C.3).

2. Flight Corridors (*sec. 6.5.3*).

Relaxed Assumption: 2., the UAS encountering cooperative and non-cooperative intruders.

Problem 3. *Static restrictions, in addition to the Intruder problem (pr. 2) the InformationSources are expanded by static restriction sources:*

1. ObstacleMap - a database containing notable landmarks, buildings, structures, with well-defined protection zones.
2. FlightRestrictions - a database containing ATM flight restrictions in non-segregated airspace for UAS relevant airspace categories.

This change impacts DataFusion by splitting Free(t) set into Free(t) and Restricted(t) disjoint sets. Also SoftConstraints are introduced which contain restricted areas from relevant information sources.

$$\begin{aligned}
 KnownWorld &:= DataFusion(t) \forall point \in KnownWorld(t) \\
 &= Free(t) \cup Occupied(t) \cup Unknown(t) \cup Restricted(t) \\
 Mission &:= \forall waypoint \in Mission \text{ are reachable} \\
 Sensors &:= \{LiDAR, ADS - B\} \\
 SensorFusion &:= \{\text{Advanced joint sets}\} \\
 InformationSources &:= \{TerrainMap, ObstacleMap, FlightRestriction\} \\
 DataFusion &:= \{\text{Advanced data fusion}\} \\
 HFlightConstraints &:= \{\text{vehicle dynamic}\} \\
 HardConstraints &:= \{\text{intruder corridors, terrain, obstacles}\} \\
 Softconstraints &:= \{\text{protection zones}\}
 \end{aligned} \tag{4.25}$$

Challenges for problem 3. :

1. Obstacle Map (*fig. 6.13*).
2. Visibility Rating Concept (*fig. 6.12*).
3. Static Constraints (*sec. 6.5.3*).

Relaxed Assumption: 3., the UAS is moving in restricted space now.

Problem 4. *Dynamic restrictions in addition to Static restrictions (pr. 3), the Weather as information source is introduced. Soft constraints are extended by medium level dangerous zones from weather map. Hard constraints are expanded by protection zones where the weather conditions are harsh. Overall Weather constraints are dynamic and changing position and shape over mission time. Modern weather systems can provide streamline overview of weather situation.*

$$\begin{aligned}
 KnownWorld &:= DataFusion(t) \forall point \in KnownWorld(t) \\
 &= Free(t) \cup Occupied(t) \cup Unknown(t) \cup Restricted(t) \\
 Mission &:= \forall waypoint \in Mission \text{ are reachable} \\
 Sensors &:= \{LiDAR, ADS - B\} \\
 SensorFusion &:= \{Advanced joint sets\} \\
 InformationSources &:= \{TerrainMap, ObstacleDatabase, \\
 &\quad FlightRestriction, Weather\} \\
 DataFusion &:= \{Advanced data fusion\} \\
 HFlightConstraints &:= \{vehicle dynamic\} \\
 HardConstraints &:= \{intruder corridors, terrain, obstacles, protection zones\} \\
 Softconstraints &:= \{protection zones\}
 \end{aligned} \tag{4.26}$$

Challenges for problem 4. :

1. Moving Constraints including Weather (def. 20).
2. Weather Avoidance Case (app. E.1).

Problem 5. Rules of the air, in addition to Dynamic restrictions (pr. 4), Rules of the air framework introduction, inducing new SFlightConstraints including air-spaces and rules of air impact on control mechanism.

$$\begin{aligned}
 KnownWorld &:= DataFusion(t) \forall point \in KnownWorld(t) \\
 &= Free(t) \cup Occupied(t) \cup Unknown(t) \cup Restricted(t) \\
 Mission &:= \forall waypoint \in Mission \text{ are reachable} \\
 Sensors &:= \{LiDAR, ADS - B\} \\
 SensorFusion &:= \{Advanced joint sets\} \\
 InformationSources &:= \{TerrainMap, ObstacleDatabase, \\
 &\quad FlightRestriction, Weather\} \\
 DataFusion &:= \{Advanced data fusion\} \\
 HFlightConstraints &:= \{vehicle dynamic\} \\
 SFlightConstraints &:= \{airspaces, rules of the air\} \\
 HardConstraints &:= \{intruder corridors, terrain, obstacles, protection zones\} \\
 Softconstraints &:= \{protection zones\}
 \end{aligned} \tag{4.27}$$

Challenges for problem 5. :

1. UTM Implementation (sec. 6.7).
2. Rule Engine for UAS (sec. 6.8.1).
3. Rule Implementation (sec. 6.8.2).

Relaxed Assumption: 5., the UAS is required to move with different velocity during Overtake maneuver.

Note. The assumptions 1. for filtered sensor output and 4. Reachable waypoints hold for all problem increments.

4.5 Avoidance Requirements

SAA systems have the following conflicting performance criteria:

1. *Energy efficiency* - minimize energy consumption and flight time.
2. *Trajectory tracking* - stick to the proclaimed trajectory in a mission plan.
3. *Safety* - avoid harm sources during the mission execution.

Note. Energy efficiency and Trajectory Tracking an optional criteria, while the safety is mandatory.

4.5.1 Energy Efficiency

Energy efficiency can be measured by *cost function* (eq. 4.28), consisting from the *cost of flown trajectory* (eq. 4.29) and the *expected reach cost* (eq. 4.30) portions. There are optimalizaiton techniques based on *Reach sets* [84]. The inputs for the *cost function* are:

1. *Time* - current mission time.
2. *Initial state* - UAS state at the beginning of a *mission*.
3. *Applied movements* - list of already executed movements.
4. *Future movements* - list of movements to be applied in future.
5. *Current state* - UAS state at *Time*, with current position and orientation included.
6. *Waypoint* - current goal waypoint.

$$Cost(t, \dots) = costTrajectoryFlown(t, \dots) + expectedReachCost(t, waypoint, \dots) \quad (4.28)$$

Cost of flown trajectory (eq. 4.29) from the *initial state* to the *current state* is calculated as a sum of energy consumed for each movement with the following components:

1. *Direct cost* - a cost of consumed energy to execute the movement.
2. *Horizontal cost* - a portion of the direct cost which was used for horizontal steering multiplied by *horizontal penalization*.
3. *Vertical cost* - a portion of the direct cost which was used for ascending/descending multiplied by *vertical penalization*.

$$costTrajectoryFlown \left(\begin{array}{c} time, \\ initialState, \\ appliedMovements \end{array} \right) = \sum_{movement \in appliedMovements} \left(\begin{array}{c} duration.directCost + \\ horizontal(cost, penalization) + \\ vertical(cost, penalization) \end{array} \right) \quad (4.29)$$

Expected reach cost (eq. 4.29) is calculated for a *planned trajectory* portion and a *direct waypoint distance* to the *latest future UAS position*. *Cost of the planned trajectory* is calculated by the same formula as a *cost of flown trajectory* (eq. 4.29), the initial state is replaced with a *current state*, and *executed movements* are replaced with *planned movements*.

$$\text{expectedReachCost} \begin{pmatrix} \text{time}, \\ \text{currentState}, \\ \text{futureMovements}, \\ \text{waypoint} \end{pmatrix} = \begin{pmatrix} \text{distance}(\text{futureState}, \text{waypoint}) + \\ \text{costTrajectoryFlown} \left(\begin{pmatrix} \text{currenState}, \\ \text{futureMovements} \end{pmatrix} \right) \end{pmatrix} \quad (4.30)$$

Note. The tuning parameters of cost function are *Horizontal penalization* $\in [0, \infty]$ and *Vertical penalization* $\in [0, \infty]$. Which are used to enhance the outcome of the *cost function*.

Following setup of tuning parameters are used in our simulations:

1. *horizontalPenalization* \leq *verticalPenalization* $< \infty$ - in *uncontrolled airspace*, all kind of maneuvers are allowed. Horizontal maneuvers are cheaper for a plane UAS.
2. *horizontalPenalizaiton* $<$ *verticalPenalizaiton* $= \infty$ - in *controlled airspace* any kind of horizontal maneuvering must be allowed by UTM.

The tuning parameters are set up *verticalPenalizaiton* \leq *horizontalPenalizaiton* $< \infty$ for *copter* UAS in an *uncontrolled airspace*

4.5.2 Trajectory Tracking

Trajectory Tracking is a crucial parameter for *controlled airspace* and is expected to be important in *upcoming UTM systems*. There is a *mission plan* which is compared with *real-time airspace situation* obtained from UAS *Position notifications*. The optimization based on *Reach Set* is given in [85].

Motivation: *Situation awareness* for modern DAA systems depends on planned trajectory tracking. The main conflict is between *navigation precision* and *situation evaluation*. If the planned trajectory is defined for the *continuous domain*, it takes much effort to calculate collision points.

The discrete domain of *Movement Automaton* (def. 33) can be used as a *tool for situation awareness*. The main idea is to use *Movement automaton as a predictor for trajectory intersection* [86, 87].

Movement Automaton trajectory tracking: There is a *reference trajectory* which is used as comparison by *aviation authority* (ATM,UTM) given as:

$$\text{ReferenceTrajectory} = \{(point_1, time_1), (point_2, time_2), \dots, \dots (point_n, time_n)\} \quad n \in \mathbb{N}^+, point_k \in \mathbb{R}^3 \quad (4.31)$$

Reference Trajectory (eq. 4.31) is given as set of *points* in *Global Coordinate System* for given *UAS*, *operational time-frame* and other authority depending properties.

The *movement* automaton is executing $Trajectory(initialState, buffer)$ where buffer is set of *Movements*. The buffer is changing according to following pattern during *mission* time frame:

Mission Start:

$$buffer = \{plannedMovements\}, planned = m$$

During Mission:

$$buffer = \{executedMovements, plannedMovements\},$$

$$executed = n, planned = o$$

(4.32)

Mission End:

$$buffer = \{executedMovements\}, executed = p;$$

Movement count constraints:

$$m, n, o, p \in \mathbb{N}^+, n + o = m, m \leq p$$

At the beginning of the mission (eq. 4.32) the buffer is filled with m movements, the *Trajectory* generated from this buffer and *initial state* is *predicted*.

During the *mission execution phase*, the buffer contains *executed movements* and *planned movements*. Trajectory created from the *initial state* and this buffer can be split into:

1. *Executed part* - trajectory portion generated and executed from *executed movements*
2. *Predicted part* - trajectory portion generated as a future reference from *planned movements*

After *mission execution*, there is only *executed movements* The trajectory generated from the *initial state* and buffer is *Executed Trajectory*

Note. The part of the trajectory bounded to the past, the part of the trajectory lies in the future. The strong point of *Movement Automaton* is its ability to work as *predictor* and *trajectory memory* at the same time.

By selecting proper time series $t_1 \dots t_n$ one can compare future or past segments of trajectory (eq. 4.32) with reference (eq. 4.31)

Reference Trajectory Deviation for reference trajectory given by (eq. 4.31) and *Trajectory segment* (Executed/Predicted) (4.32) with existing *State projection function* (eq. B.11) for *time series* is given as:

$$\text{Deviation} \left(\begin{matrix} \text{timeSeries}, \\ \text{Trajectory}, \\ \text{Reference} \end{matrix} \right) = \sum_{\substack{\text{time}_i \in \\ \text{timeSeries}}} \left(\frac{\text{StateProjection}(\text{Trajectory}, \text{time}_i)}{\text{Reference}(\text{time}_i)} - \right)^2 \quad (4.33)$$

Reference Trajectory Deviation (eq. 4.33) is designed as discrete *Mean Square Error* function, where the *timeSeries* is set of *times* from *reference trajectory* ($point_i, time_i$) pair. The *state projection*.

Trajectory tracking is defined as *dual minimization problem* where the *primary objective* is depending on the *airspace type*:

1. *Reference trajectory deviation* (eq. 4.33) in *Controlled Airspace*.
2. *Cost of Flown Trajectory* (eq. 4.28) in *Non-controlled Airspace*.

Trajectory tracking can be defined as an optimization problem (eq. 4.34).

$$\begin{aligned}
 \text{Minimize:} & \quad costOfTrajectoryFlown & (4.28) \\
 \text{Minimize:} & \quad referenceTrajectoryDeviation & (4.33) \\
 \text{Subject to:} & \\
 & \quad UAS \text{ Dynamics} & (B.13) \\
 & & (B.12) \\
 & \quad MovementAutomatonControl & : \\
 & & (B.21) \\
 & \quad Mission & (4.6) \\
 & \quad KnownWorld(t) & (4.20) \\
 & \quad SafetyMargin(t) & (4.21) \\
 & \quad HardFlightConstraints & (4.8) \\
 & \quad SoftFlightConstraints & (4.9) \\
 & \quad HardSpaceConstraints & (4.11) \\
 & \quad SoftSpaceConstraints & (4.12)
 \end{aligned} \tag{4.34}$$

The *reference trajectory* is given by *mission* set of *waypoints*. The *UAS* dynamics with specific *Movement Automaton* goal is to fly in *Known World* to keep *Safety Margin* from *Obstacle Space*. The *Obstacle space* is a result of *Data fusion* procedure (sec. 4.1.9) combining the *sensor reading*, information sources, and constraints.

Feasible Trajectory for *tracking problem* (eq. 4.34) is a trajectory which in addition to *basic obstacle problem* (sec. 4.2) keeps deviation from the *reference trajectory* under certain threshold:

$$Deviation(timeSeries, Trajectory, Reference) \leq performanceMargin \in \mathbb{R}^+ \tag{4.35}$$

Feasible trajectory condition (eq. 4.35) is used as *margin* for airworthiness, and *Deviation* is used as a performance indicator further in this work.

4.5.3 Safety

Safety is very broad term there are following incidents which can occur and will be discussed in (app. H.2) *Safety margin* is a broad term describing minimal distance to the center of intruder/adversary, a surface of the obstacle, a boundary of the protected area.

Controlled airspace safety Safety for *controlled airspace* in given *flight level* is given a list of incidents:

1. *Soft constrained zone breach* - UAS fly to *soft constraint body* or *hard constraint protection zone*, these incidents can happen, and have least avoidance priority.
2. *Hard constrained zone breach* - UAS fly to *hard constraint protection zone*, typical geo-fencing, restricted airspace breaches.
3. *Well-clear breach* - UAS fly to *well-clear barrel* without impacting other aircraft, via the wake turbulence or other induced physical phenomenon and vice-versa. This type of breaches are allowed in case of inevitable *near miss situations* or *Clash incidents*
4. *Near miss situation*- UAS fly to *near miss cone/barrel*, inducing wake turbulence, or other kinds of flight disturbance. These incidents are allowed at very low rate (near $1 : 10^6$).
5. *Clash incident* - UAS body impacts another aircraft hull/propulsion/steering systems and components. This kind of incidents are very severe, and they should never happen.

Note. It is assumed that flight level in controlled airspace is free of terrain, static ground obstacles, the climb/descent maneuvers are not covered in this work, and they are topic for multiple dissertation theses. For more information refer to ICAO document 4444.

The *relation* for breach of *safety margin* and *body margin* for each object is given in (tab. 4.1):

| Violation of: | Safety Margin | Body Margin |
|-----------------|---|------------------------------|
| Soft constraint | none | Soft constraint zone breach |
| Hard constraint | Soft constrained zone breach | Hard constrained zone breach |
| Intruder | Well clear breach, Near Miss situation | Clash incident |

Table 4.1: Controlled airspace margins violations incidents.

Uncontrolled airspace safety Safety for *uncontrolled airspace* is applied in *F/G* class of airspace, which is given as airspace between the *ground level* and *other airspace prevalence*.

Note. Clarification of controlled/uncontrolled airspace:

1. *Class F* airspace is given as space between the ground level or water surface, and it is constrained up to the 500 feet above ground level.
2. *Class C* airspace or *Controlled airspace* is considered starting at first flight level, which is given by Air traffic control zone starting at least at 300 feet from highest ATC zone ground point. It is measured based on Above Sea Level altitude. *This is not a problem in Portugal, because of terrain diversity, but its a huge problem in the Netherlands.*
3. *Class A* airspace starts at ground level and covers the majority of airport infrastructure - this is not a problem, because it's modeled as hard constraint, which is unbreakable in non controlled airspace.

Safety for *uncontrolled airspace* is given a list of incidents:

1. *Soft constrained zone breach* - UAS fly into the *soft constrained zone* or *hard constraint protection zone*, it is allowed to happen on very a low rate.
2. *Hard constrained zone breach* - UAS fly into the *hard constrained zone*, only airports and critical infrastructure are considered as hard constraints; it is not allowed to happen.
3. *Intruder near miss* - UAS fly into *other aircraft near miss zone*, it is allowed to happen on a very low rate in case of other intermediate threats with higher priority.
4. *Intruder clash* - UAS has contact with other man-made aircraft, it is not allowed to happen.
5. *Adversary clash* - UAS has contact with another flying object which did not intentionally avoided UAS. (*Bird strike, Differential games, etc..* is out of the scope of this thesis).
6. *Structure harm* - UAS fly close to structure, and its propulsion can damage/harm structure.
7. *Structure crash* - UAS fly into a natural/man-made ground structure (building, tree, human).
8. *Ground harm* - UAS fly close to the ground, and its propulsion reflection can impact Ground or UAS.
9. *Ground collision* - UAS collides with the ground.

The *relation* for breach of *safety margin* and *body margin* for each object is given in (tab. 4.2):

| Violation of: | Safety Margin | Body Margin |
|-----------------|------------------------------|------------------------------|
| Soft constraint | none | Soft constraint zone breach |
| Hard constraint | Soft constrained zone breach | Hard constrained zone breach |
| Intruder | Intruder near miss | Intruder clash |
| Adversary | none | Adversary clash |
| Structure | Structure harm | Structure crash |
| Ground | Ground harm | Ground crash |

Table 4.2: Non-controlled airspace margins violations incidents.

4.6 Navigation Requirements

Navigation requirements are not the main part of this work; they are used to show the variability of the approach for *DAA* requirements:

1. *Contextual behavior* - change navigation and decision behavior based on context:
 - a. *Airspace type* - Controlled/Uncontrolled,
 - b. *Navigation mode* - Cooperative/Emergency.
2. *Determinism* - same result for same dataset in finite time.
3. *Threat prioritization* - threats prioritization based on *context*, (tab. 4.1) and (tab. 4.2).
4. *Rule compliance* - compliance with a given set of rules based on context (focus on rules of the air).

| Requirement | Evaluation metrics |
|------------------------------|--|
| Constrained space navigation | Mission scenario does not have a direct path between waypoints, additional borderline cases. |
| Contextual behavior | Avoidance system changes behavior based on the mission and vehicle context. |
| Determinism | Multiple runs of same non-borderline scenario returns same avoidance paths. |
| Rule compliance | Rules applied comply with aviation standardization. |

Table 4.3: Navigation requirements evaluation metrics.

Chapter 5

State of the Art

This chapter is introducing notable works and concepts of *researchers* in the field of *control theory*, *software engineering* and other essential fields used in *Detect and Avoid* systems.

5.1 Movement Control

Idea: The key idea is to create *interface* between *controlled plant* (UAS) and *Avoidance Algorithm* to ensure *Concept Reusability* at maximum degree. The concept is the following:

1. *Interface consumes* discrete command chain and guides UAS along the *desired trajectory*.
2. *An interface* can be used to *predict trajectory* based on *initial state* and future command chaining.

Frazzoli Movement Automaton: The following paragraph strongly follows Frazzoli work [87] (sec 3.1-3.5). Frazzoli provided the concept of *Movement Automaton* (def. 33) a specialized type of *Hybrid Automaton* (eq. 3.11), the concept is taken from his works [86, 87]. Other aspects and similarities are discussed in this chapter.

The approach was proposed to reduce the *computational complexity problem of motion planning*. The quantization of the system dynamics is done through restriction of feasible nominal system trajectories to the *family of time parametrized curves*. These can be obtained by the interconnection of trajectory primitives.

Trajectory primitives are repeatable portions of trajectory (def. [87].3.1). The trajectory primitives are interconnected by *transitions* to create maneuvers (movements) (def. [87].3.3).

By combining movements as a set of trim trajectories the trajectory can be represented as a set of discrete time bounded commands. This is summarized in the definition (def. [87].3.4) based on *hybrid automaton definition* (sec. 3.3).

Definition 10 (Maneuver Automaton). *A maneuver automaton over a mechanical control system S , with symmetry group H is described by the following objects:*

1. A finite set of indices $Q = Q_T \cup Q_M \in \mathbb{N}$, where the subscript T relates to trim trajectories, and the subscript M relates to maneuvers;
2. A finite set of trim trajectory parameters $(\bar{g}, \bar{\xi}, \bar{u})_q$; with $q \in Q_T$.
3. A finite set of maneuver parameters, and state and control trajectories $(T, u, \phi)_q$, with $q \in Q_M$.
4. The maps $\text{Previous} : Q_M \rightarrow Q_T$, and, $\text{Next} : Q_M \rightarrow Q_T$ such that $\text{Previous}(q)$ and $\text{Next}(q)$ give, respectively, the index of the trim trajectories from which the maneuver q starts and ends.
5. A discrete state $q \in Q$.
6. A continuous state, denoting the position on the symmetry group, $h \in H$.
7. A clock state $\theta \in \mathbb{R}$, which evolves according to $\dot{\theta} = 1$, and which is reset after each switch on q .

Note. It is apparent that decisions can be made about the future evolution of the system only when the system is executing a trim trajectory (that is, the discrete state is in one of the nodes in the graph). While executing a maneuver the system is committed to it and must keep executing the maneuver until its completion. As a consequence, for motion planning and control design purposes, one can concentrate the study of the evolution of the system on and between nodes.

Architecture: The Movement Automaton can be seen as a consistent hierarchical abstraction of the continuous dynamics, in sense outlined in [88]: *Any sequence of movement primitives generated by the Movement Automaton results by construction in a trajectory which is executable by the full continuous system. We will give a deeper meaning to hierarchical consistency.*

Optimal Path Generation: If the maneuvers are instantaneous (i.e., the UAS can transition instantaneously between two different trim trajectories), Reduction of stronger results obtained by Dubins [89] and Reeds [90] concerning optimal paths for kinematic cars on the plane (see also [91]).

Controllability: The systems controlled by Movement Automaton (as in [92]), is controllable according to our definition, even though it is not small-time controllable [93].

Other Properties: The other properties of movement automaton, like *Stability*, *R robustness* and other important control properties are proven in [87].

Example: The *example* is given in (fig. 5.1). The *States* (Barrels) are connected by *Transitions* (green arrows).

Hover is the neutral and *initial state*, in this place the UAS stays on place and maintains altitude.

Forward flight is when *UAS* is flying in frontal direction with constant speed. The speed-up and slow-down are incorporated in *Transition* between *Hover* and *Forward flight* states, and it takes some time to execute. *Transitions* between Turning states and *Flight forward* state are almost instant.

Steady turn left/right is when *UAS* is flying in the frontal direction and starts steady turning left or right.

Note. UAS in (fig. 5.1) ignores the vertical maneuvering, and it is expected to fly on horizontal plane.



Figure 5.1: Movement Automaton for Copter UAS [87].

Note. The *Movement Automaton* is used as with modification (sec. B.2). The implementation is described in (sec. 6.2). The testing configuration was given in (tab. 7.2).

5.2 Surveillance

Idea: There is a *need for the abstract representation of operational space*, which is independent of used sensors, technologies, information sources. The universal obstacle avoidance system should be *portable* between various platforms. Our previous work *Obstacle avoidance framework based on reach sets* [9] has introduced a concept of the *control interface*. The concept of *control interface* increases portability of the solution.

The original concept used cell status interpretation (boolean values), which is hard-wired to LiDAR technology. The basic methods for *Statistical Sensor Fusion* were outlined in [94]. The result of the application of the method is *data fusion interface* (sec. 4.1.9) - interface to fuse sense data from various online, offline, cooperative, non-cooperative sources into single scalable space and trajectory evaluation procedure.

Related work: UAS specific sensor fusion has been proposed by Ramsay in [29]. *Next generation avoidance concept* [95] is introducing concept of higher-level sensor fusion called *data fusion*.

The uncertainty and properties in *Remotely Piloted Systems* have been discussed in [96]. The work provided the concept of various performance ratings like visibility and obstacle rating; more details have been given in [97]. These ratings were modeled only for operator decision making [98], results are usable for automated decision making and space assessment.

5.3 Navigation Algorithms

Idea: The basic idea is to provide hierarchical *navigation frame* with *some optimal path search capabilities*.

Space Assessment: *Probabilistic trajectory assessment* has been firstly proposed in [99] where trajectory was tracking and predicting *safety properties* along.

Game theory viewpoint is firstly used in [100]. Probabilistic path planning using safety zones similar to cell classification of this work has been used in [101].

Probabilistic path search similar to our reach set representation using rapidly exploring path trees have been used in [102, 103]. The relationship between classic grid search and probabilistic lattice search have been established in [104]. A probabilistic approach for trajectory estimation via reduced lattice search is known from 1986 from work of Gessel [105] lattice paths were enumerated via movement sequences and a similar technique is used in our reach set estimation method using movement automaton. Pruning methods comparison and complexity can be found in [106].

Overall concepts of probabilistic sets have been given by Hirota in [107]. Free flight safety rating similar to our reachability concept has been presented in [108].

Standard Navigation: The standard navigation is given as *expected cost optimization problem* for *future cost function* (eq. 4.30). The key concept of navigation algorithm was fully taken from [109]. The decision was made based on navigation survey [110]. The *descent* for landing is out of scope in this work, can be found in [111]. The navigation principle is roughly described in (sec. 6.6.2).

Maze Solving Capabilities: The *maze solving capability* is usable in *controlled airspace* where 2D maze solving algorithms are applicable. The notable implementation was for *micro mouse robot* based on right-hand rule [112]. Flood fill algorithm is partially usable for 3D environment [113]. The application of *maze solving* was given in case study [114].

Hybrid Automaton Path Planning: A hybrid automaton path planning based on *A** algorithm was given by Richards in [115]. The key idea was to use *hybrid automaton* (eq. 3.11) as a reference generator. This idea was taken and formulated as *Movement Automaton Predictor mode*.

The similar idea where *potential fields* were used as the *intruder model* and path was re-planned based on events is given in [116].

Mode Switch: The *Mode Switch Control* idea has been presented in [117]. There was the definition of behavioral switch between:

1. *Navigation Mode* - navigation control and behavior was used.
2. *Task-Specific Mode* - mode specific for tasks, authors were using modes for search and rescue.

This concept will be reused; the *task-specific mode* will be *Emergency Avoidance Mode* in Our case. The triggering events and switch conditions will be defined in (sec. 6.6.2).

Used Concepts: The *Following concepts* were used in navigation loop:

1. *Standard navigation* took from [109] minor implementation changes using offline optimization. The purpose of the navigation loop is to bring us closest to the waypoint if it is reachable. Navigation example (sec. 7.4.3).
2. *Maze solving capabilities* partially taken as secondary functionality based on [113]. The purpose is *looping prevention*. The example was given in (sec. 7.3.3).
3. *Mode switch* partially taken as the main feature from [117], the triggering events were identified and defined by author and can be found over (chapter 6).

Shortcomings:

1. *Hierarchical calculation* - there is a need to calculate the *avoidance trajectory* for incremental constraint applications. For example:
 - a. Calculate *Minimal escape path* covering physical obstacles and intruders.
 - b. Apply next level of constraints, like airspace restrictions and some virtual constraints. Then calculate path if exists, continue.
 - c. Apply nice to have constraints, like non-lethal weather, recalculate the path.
2. *Source Reliability Evaluation* - reliability evaluation is an empirical process usually done by hand. The result aggregation is not standardized. There can be multiple sources of the same rating, for example, visibility, which needs to be aggregated into one.
3. *Ambiguous rating definition* - There are multiple definitions especially for *Reachability rating* in works [102, 103, 105].

5.4 Reach Set Estimation

Idea: The basic idea for *Discrete Reach set Estimation method* is taken from [118]. The focus of their work is to generate paths that are kinematically feasible. Path following controllers in order to find techniques to stabilize the system around these paths [119, 120].

Lattice-based planners have been deployed with great success on several robotic platforms [121, 122, 123, 124, 125]. However, a problem with lattice-based approaches is the exponential complexity in the dimension of the state space which can limit the use for more complicated models.

The optimization problem was solved real-time by *Avocado solver* [126].

Example: The *example of movement lattice* is given in (fig. 5.2). Truck (black rectangle) is towing Trailer (red rectangle). The *state* has only one *reach set impacting variable - trailer displacement*. When trailer displacement is 0° (fig. 5.2a) the lattice representation of *Reach Set* is different than in case of small left/right tilt (fig. 5.2b).



Figure 5.2: *Movement set primitives* for Lattice-Based Movement Planning. [118].

Benefits: Presented method of *Lattice Search* is de-facto *Reduced Reach Set Approximation* in open space for *Truck-Trailer* system.

The idea of *Movement primitives* is very close to *Movement Automaton* (def. 33), which can be used as a *control interface* effectively.

The *Constraints* of obstacle set in *Known world* (sec. 4.1.10) are supported to some degree.

Shortcomings: The lattice search approach has the following shortcoming:

1. *Limited system dimension* - given method works in *real time* only if dimension of *system state space* does not exceed 4^{th} rank.
2. *Real-time optimization* - real-time optimization is main cause of *limited system dimension*. If the decision time can be discrete (which movement automaton enforces), then offline optimization can be used.

3. *Continuous space disparity* - the example (fig. 5.2) shows there are member variables of *State Space* which significantly impacts the shape of the lattice (reach set estimation). Space disparity is not a problem in real-time environment. The discretization of *time-domain* raises this as a shortcoming.
4. *Trajectory Tracking* - approach generates *Continuous Domain* reference signal. For *Discrete Domain*, it is necessary to address this issue.

5.5 Software Testing

Idea: *Reuse LSTS toolchain architecture* for DAA testing framework.

LSTS Toolchain: Software architecture used in modern unmanned aerial vehicles must be system independent and scalable. Writing own control software for unmanned aerial vehicle and ground station is unthinkable in the current state of the art. Most notable framework for unmanned aerial vehicle development is the LSTS toolchain from University of Porto [127]. This toolchain is widespread in other universities, and multiple independent applications are based on it [128].

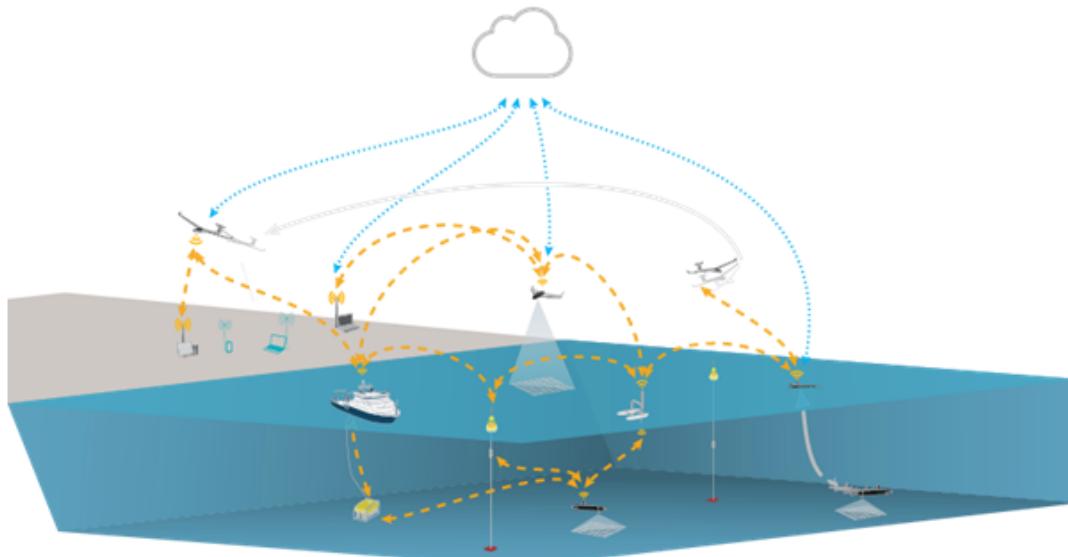


Figure 5.3: Example of LSTS toolchain deployment in a real environment [129]

Example of software architecture implementation is shown in figure 5.3 LSTS HUB (cloud iconography) is collecting all important data from currently executing missions. Data are transferred via REST API (dotted blue lines) to HUB. Commanded vehicles can be unmanned copters, planes, ships, submarines or floating sensors compounds. Each vehicle has installed DUNE which is responsible for vehicle command and ground control station communication. Deployment, a range of command and status messages can vary. The ground station can be implemented on a personal computer (any platform) in NEPTUS environment or mobile platform (Android OS) in ACCU environment. Ground

station environments are customizable and open source. The layout of the ground station can be customized to need of current mission via plugins or console configuration. Vehicles and ground stations are communicating via IMC protocol (orange dashed lines). The communication channel is platform independent.

Glued is a Debian-based open source operating system which was initially released in 2010. Over the years it has become fairly widespread and been provided continuous updates. Meanwhile, a notable development community has emerged, where advice and support can be received as more applications are investigated. Some of the merits of the operating system are its reliability and customizability. Operation system is developed for unmanned autonomous vehicles. It is widely used in air, sea and land vehicles. Customization allows the operating system to be tailored to the specific usage. For this purpose, this includes stripping off functionality which is not required, thus releasing resources to be focused on the essential tasks. Glued is the operating system favored by the Beaglebone development community, and thus there exist considerable amounts of helpful documentation for this set-up.

DUNE: Unified Navigational Environment is an onboard software solution for unmanned vehicles. Multiple applications already run on this software. Almost any extension can be added to this to this environment. The software solution provides a means for interacting with the connected components as well as control, navigation, supervision and plan execution. It is both CPU architecture independent and OS independent. It is written in C++ and developed by LSTS: Underwater Systems and Technology Laboratory.

Neptus [129, 130, 131] is a command and control software operated from a ground station. It is designed to operate well together with Dune and was also developed by LSTS. Neptus provides tools for remotely monitoring UAVs and assigning plans and commands in real-time missions, supporting multiple connections dynamically. Furthermore, it provides possibilities for both simulating missions and reviewing previously performed operations. This is presented in a customizable interface equipped with map layers and control panels. It is written in Java and available for both Windows and Linux systems.

The *Inter-Module Communication* [132] (IMC) protocol was developed by LSTS to provide reliable communication between the systems. The protocol is message-oriented, such that messages can be sent and received from a bus which connects independently run threads or systems. Thus it functions as a method of communication between tasks internally in Dune, and can also be passed to and from other vehicles or computers running Dune or Neptus. IMC is platform independent at multiple messages have been already developed and supported by both DUNE and Neptus (around 400 status/command messages).

HUB is communication hub for data dissemination and situation awareness. This module is responsible for complex mission execution when cooperation of multiple pilots/vehicles is required. This module can be imagined as an airport tower (traffic control) center, which is monitoring all flights in the airport (operation site) area.

Movement Automaton Control Marconi used *hybrid automaton* with forced *State Switch* via buffer [133]. The key concept is that *Automata* state switch is forced as *external source command*. Our *Movement Automaton* implementation knows only *forced state switch* like in [86].

Used concepts: Most of the architecture was re-used in our approach, the concept of *Rule engine* (sec. 6.8) was introduced to cover missing *UTM* related functionality. The implementation in Matlab was influenced by Alessandetti works [134, 135]. The other aircraft dynamic and control related concepts were taken from [136].

5.6 UTM Services

Idea: Take the Airbus UTM concept [2] combine it with *EUROCONTROL* concept [137] to obtain a legal framework. *Provide conflict resolution functionality for Controlled Airspace*:

1. *Collision Detection* - define minimal required functionality for collision detection.
2. *Collision Resolution* - implement *Rules of the Air* [138].

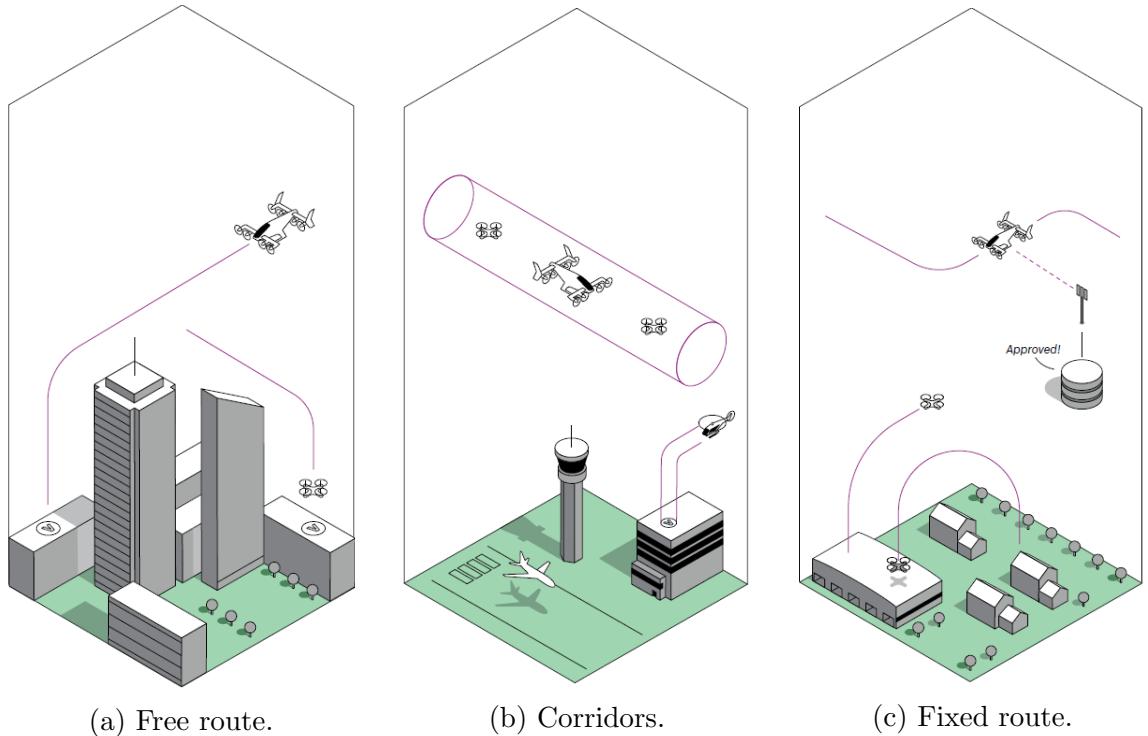


Figure 5.4: UTM Operation Modes [2].

UTM Operation Modes: *defined in [2]* are following:

1. *Free route* (fig. 5.4a) is when aircraft can fly any path, so long as their planned path is coordinated with and de-conflicted from the paths of other aircraft by a traffic manager and approved based on calculated risk. Free routing is being introduced worldwide, such as free route airspace. This allows commercial flights to freely plan their route through participating sectors during the cruise. There is less freedom for an aircraft in this situation than in basic flight, since its request may be rejected.
2. *Corridors* (fig. 5.4b) are defined volumes in space, useful for managing airspace in high demand or to manage traffic flow and separation. Coordination is necessary to ensure safety in this airspace. A corridor may take on many different shapes. Aircraft are often guided inside corridors using predetermined routes analogous to approach procedures used worldwide today.
3. *Fixed route* (fig. 5.4c) are used to ensure safety when there is high traffic density or in any location where the structure is required to ensure safe operations. This could include locations such as airports or warehouses. These routes could be constructed or modified dynamically based on calculated risk. The most restrictive version is a predetermined path, where the only variable is when an aircraft is at a specific point in the path.

Used Concepts: The implementation of our UTM services is focused on *Free route mode* (fig. 5.4a). The *Corridors* and *Fixed routes* are just additional *space/time constraints*.

Chapter 6

Approach

The levels of *Avoidance* depending on *reaction time* are summarized in (fig. 6.1).

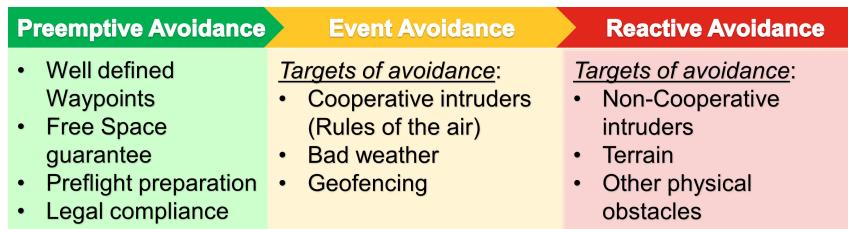


Figure 6.1: Avoidance levels based on reaction time.

This work will focus on handling *Event Avoidance*, and *Reactive Avoidance* and the *Avoidance Path* will be calculated using *Reach set Based Methods*.

The *Preemptive Avoidance* is trying to remove any possible threat before the flight. The risk mitigation is tedious and its done only when necessary. Even the best *preemptive* avoidance could fail.

Reactive Avoidance is solving most urgent situations with very short reaction opportunity. This work focus on physical obstacles and terrain. Non-cooperative intruders are partially considered. The adversary behavior was is not considered.

Event Avoidance has more opportunity to react. Some threats are known prior the flight (geo-fenced areas, etc.). The future UTM implementation is also considered as *Event Avoidance*, due to the time horizon and authority enforcement.

Basic Idea: Create deterministic finite-time *Reactive Avoidance* based on *Reach sets* to ensure *trajectory feasibility*. Enhance method with a set of the rules to enable handling more complex situations.

The *Discretization* is the key to ensure calculation in finite time. Finite *partition* of *operational space (Known World)* and finite representation of *Reach set* guarantees finite count of calculation steps. Aircraft conflict prediction mentioned in [139].

6.1 Overview

The *Overview* is based on *Existing Emergency avoidance framework* [9] (fig. F.1). To achieve goals defined in *Problem Definition* (sec. 4.2, 4.4) following *Avoidance Framework Concept* (fig. 6.2) is proposed:



Figure 6.2: Avoidance Framework Concept.

Structure of Avoidance Framework:

1. *Unmanned Aircraft System* (UAS) (Role: Controlled Plant) - the *UAS* is controlled via *interface* implemented as *Movement Automaton*. The model used is described in (sec. 6.2.2).
2. *Movement Automaton* (Role: Control Interface/Predictor) - consumes *Discrete Command Chain* to generate discrete *reference trajectory*, it can also be used as a predictor of *future UAS states* (sec. B.4). The movement Automaton used in this work is given in (sec. 6.2.3).
3. *Sensor Field* (Role: Surveillance Providers), the following sensors, were considered in this work:
 - a. *LiDAR* (Static obstacle detection) - detection of physical obstacles (eq. 6.50)
 - b. *ADS-B* (Intruder UAS/Plane detection) - detection of intruders who are broadcasting their position and sometimes heading with plans and additional parameters. The *intersection models* are given in (sec. 6.5.2, app. C.1, C.2, C.3).

4. *Information Sources* (Role: Known World Information Enhancers):
 - a. *Obstacle Map* (Static Restriction Source) - imposing static soft/hard constraints on *Known Word/Operational Space*. Static constraints are given in (sec. 6.5.3).
 - b. *Weather Information* (Static/Dynamic Restriction Source) - imposing static/moving soft/hard constraints on *Known World/Operational Space*. Moving constraints are given by (def. 20).
 - c. *Other Airspace Restrictions* - like restricted airspace, geo-fencing, and other future constraint sources, all of them are covered by *Static/Dynamic Constraints* for now.
5. *Data Fusion* (Role: Sensor Input Interface) - is the unifying interface to asses *Operational State Properties* mainly *Obstacle Rating*, *Visibility*, *Map Obstacle Rating*, *Intruder Rating* for a portion of the space. The partial *ratings* are proposed in related sections. The data fusion procedure with *defuzzification* and final assessment into space sets are outlined in (sec. 6.5.4)
6. *Reach Set Approximation* (Role: Reachability Estimator) - as *data fusion* is providing the situation assessment, the *Reach set* is providing maneuvering capability assessment. The introduction is given in (sec. 6.4), the properties are defined in (sec. 6.4.2), the approximation methods with constrained expansion are outlined in (sec. 6.4.4, 6.4.5, 6.4.7, 6.4.6). The reach set estimation is the main contribution of this work.
7. *Grids: Navigation/Avoidance* (Role: Operation Space Segmentation & Situation Evaluation) - space discretization in polar coordinates grid, different reach sets are used for different grid type, defined in (sec. 6.3).
8. *Avoidance loop* (Role: Short Term Decision Maker) - using data from *Sensor fusion* in *Avoidance/Navigation Grid* trimming *Reachable Space* approximated by *Reach Set* generating feasible *Avoidance Path*. *Avoidance Path* is fed to controlling *Movement Automaton*. The Goal is given by *Navigation Loop*. Avoidance loop is given in (sec. 6.6.1).
9. *Navigation loop* (Role: Long Term Decision Maker) - using data from *Avoidance Loop*, *Mission plan* and *UTM* directives defines the current long term navigation goal. Details are given in (sec. 6.6.2).
10. *Command and Control Communication Link* (C2 Link) (Role: Communication Link) - standard communication link with sufficient reliability.
11. *UAS Traffic Management* (UTM) (Controlled Airspace Authority) - checking possible collisions and enforces counter-measurements. Details are given in (sec. 6.7).

Communication in Avoidance Framework:

1. *UAS* \leftrightarrow *Movement Automaton* - sharing *actual system state*, commanding the UAS platform.
2. *Reach Set* \leftrightarrow *Movement Automaton* - predicting a set of feasible trajectories for the given situation.
3. *Reach Set* \leftrightarrow *Grids* - providing trajectory set depending on the active mode (Navigation/Emergency Avoidance).
4. *Avoidance Loop* \leftrightarrow *Data Fusion* - assessing the situation in *operational space* based on sensor readings/information sources.
5. *Avoidance Loop* \leftrightarrow *Navigation Loop* - determining long term goal based on situation assessment and UTM directives.
6. *Avoidance Loop* \rightarrow *Grids* - feeding assessment data and constraints into selected operational space Grid.
7. *Grids* \rightarrow *Avoidance Loop* - returning feasible and *cost-effective* avoidance path after situation assessment and *Reach set* pruning.
8. *Avoidance Loop* \rightarrow *Movement Automaton* - issuing and monitoring movement commands based on actual *avoidance strategy*.
9. *Navigation Loop* \leftrightarrow *C2 Link* \leftrightarrow *UTM* - communication to receive directives and send fulfillment.

6.2 UAS Model and Control

The key feature of *Movement Automaton* is to interface the *UAS system* as the *discrete command chain*. Following topics are introduced in this section:

1. *Movement Automaton Applications* (sec. 6.2.1) - the listing of related work and similar approaches to ours.
2. *UAS Model* (sec. 6.2.2) - a simple plane model used in this work as the *controlled plant*.
3. *UAS Movement Automaton* (sec. 6.2.3) - movement automaton for *UAS Nonlinear Model* constructed from scratch.

6.2.1 Movement Automaton Applications

Movement Automaton is a basic interface approach for discretization of *trajectory evolution* or *control input* for any *continuous or discrete system model*.

Main function of *Movement Automaton* is for system given by equation $\dot{\text{state}} = f(\text{time}, \text{state}, \text{input})$ with initial state state_0 to generate *reference trajectory* $\hat{\text{state}}(t)$ or *control signal* $\text{input}(t)$.

Using *Movement Automaton* as *Control Proxy* will provide us with *discrete command chain* interface. This will reduce the *non-deterministic* element from *Evasive trajectory* generation, by reducing infinite maneuver set to finite *movement set*.

Non-determinism of *Avoidance Maneuver* has been discussed as an issue in following works:

1. Newton gradient method for evasive car maneuvers [140].
2. Non-holistic methods for trajectory generation [141].
3. Stochastic approach to elliptic trajectories generation [142].

Examples of *Movement Automaton Implementation as Control Element* can be mentioned as follows:

1. Control of traffic flow [143].
2. Complex air traffic collision situation resolution system [86, 87].
3. SAA/DAA capable avoidance system [9].

6.2.2 UAS Model

Motivation: Simplified rigid body kinematic model will be used. This model has decoupled roll, yaw and pitch angles. The focus is on *reach set approximation methods*; therefore the *UAS model* is simplified.

State Vector (eq. 6.1) defined as a positional state in euclidean position in right-hand euclidean space, where x , y , z can be abstracted as latitude, longitude, altitude.

$$state = [x, y, z, roll, pitch, yaw]^T \quad (6.1)$$

Input Vector (eq. 6.2) is defined as the linear velocity of UAS v and angular speed of rigid body $\omega_{roll}, \omega_{pitch}, \omega_{yaw}$.

$$input = [v, \omega_{roll}, \omega_{pitch}, \omega_{yaw}]^T \quad (6.2)$$

Velocity vector function (eq. 6.3) is defined through the standard rotation matrix and linear velocity v , oriented velocity $[v_x, v_y, v_z]$ given by (eq. 6.4).

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} v \cos(pitch) \cos(yaw) \\ v \cos(pitch) \sin(yaw) \\ -v \sin(pitch) \end{bmatrix} \quad (6.3)$$

UAS Nonlinear Model (eq. 6.4) is given by *first order equations*:

$$\begin{aligned} \frac{dx}{dt} &= v \cos(pitch) \cos(yaw); & \frac{droll}{dt} &= \omega_{roll}; \\ \frac{dy}{dt} &= v \cos(pitch) \sin(yaw); & \frac{dpitch}{dt} &= \omega_{pitch}; \\ \frac{dz}{dt} &= -v \sin(pitch); & \frac{dyaw}{dt} &= \omega_{yaw}; \end{aligned} \quad (6.4)$$

Discretization for *fixed step k* we start with discretization of the model:

The *linear velocity* in text step is given:

$$v(k+1) = v(k) + \delta v(k) \quad (6.5)$$

The *roll, pitch, yaw* for next step are given

$$\begin{aligned} roll(k+1) &= roll(k) + \delta roll(k) \\ pitch(k+1) &= pitch(k) + \delta pitch(k) \\ yaw(k+1) &= yaw(k) + \delta yaw(k) \end{aligned} \quad (6.6)$$

The $\delta v(k)$ is *velocity change*, $\delta roll(k)$, $\delta pitch(k)$, $\delta yaw(k)$, are *orientation changes* for current discrete step k . If the duration of *transition* is $0s$ (as. 6) then 3D trajectory evolution in discrete time is given as:

$$\begin{aligned}
x(k+1) &= x(k) + v(k+1) \cos(pitch(k+1)) \cos(yaw(k+1)) &= \delta x(k) \\
y(k+1) &= y(k) + v(k+1) \cos(pitch(k+1)) \sin(yaw(k+1)) &= \delta y(k) \\
z(k+1) &= z(k) - v(k+1) \sin(pitch(k+1)) &= \delta z(k) \\
time(k+1) &= time(k) + 1 &= \delta time(k)
\end{aligned} \tag{6.7}$$

The $\delta x(k)$, $\delta y(k)$, $\delta z(k)$ are positional differences depending on *input vector* for given discrete time k :

$$input(k) = \begin{bmatrix} \delta x(k), \delta y(k), \delta z(k), \delta v(k), \\ \delta roll(k), \delta pitch(k), \delta yaw(k), \delta time(k) \end{bmatrix}^T \tag{6.8}$$

The *state vector* for discrete time is given:

$$state(k) = \begin{bmatrix} x(k), y(k), z(k), v(k), \\ roll(k), pitch(k), yaw(k), time(k) \end{bmatrix}^T \tag{6.9}$$

6.2.3 UAS Movement Automaton

Motivation: An *UAS Nonlinear Model* (eq. 6.4) can be modeled by *Movement Automaton* (def. 33).

Movement Primitives by (def. 28) are given as (eq. B.1). Each movement primitive will last for fixed duration 1s.

Assumption 6. Let assume that transition time of roll, pitch, yaw, and the linear velocity is 0s.

Under the assumption (as. 6) the *movement transitions* (def. 29) have zero duration. Therefore movement primitives can be considered as movements.

Note. The assumption (as. 6) can be relaxed under the condition that *path tracking controller exists*.

Movements satisfying (def. 30), for the nonlinear model (eq. 6.4) reduced to *discrete model* (eq. 6.10), are given by *apply movements* function (eq. 6.5, 6.6, 6.7).

$$state(k+1) = applyMovement(state(k), input(k)) \tag{6.10}$$

Movement Set for the discrete model (eq. 6.10) is defined as a set of unitary movements on main axes (tab. 6.1) and diagonal axes (tab. 6.2).

The maneuvering capability of several commercial small fixed-wing UAS was abstracted together. The turning rate on horizontal/vertical is defined as 15° .

The deltas are posed in *UAS body-fixed coordinate frame* (ap. A) for discrete time k .

| Parameter | Movement | | | | |
|---------------------------|----------|-------|------|------|-------|
| | Straight | Down | Up | Left | Right |
| $\delta x(k)[m]$ | 1.00 | 0.98 | 0.98 | 0.98 | 0.98 |
| $\delta y(k)[m]$ | 0 | 0 | 0 | 0.13 | -0.13 |
| $\delta z(k)[m]$ | 0 | -0.13 | 0.13 | 0 | 0 |
| $\delta roll(k)[^\circ]$ | 0 | 0 | 0 | 0 | 0 |
| $\delta pitch(k)[^\circ]$ | 0 | 15° | -15° | 0 | 0 |
| $\delta yaw(k)[^\circ]$ | 0 | 0 | 0 | 15° | -15° |

Table 6.1: Input values for main axes movements.

| Parameter | Movement | | | |
|---------------------------|-----------|------------|---------|----------|
| | Down-Left | Down-Right | Up-Left | Up-Right |
| $\delta x(k)[m]$ | 0.76 | 0.76 | 0.76 | 0.76 |
| $\delta y(k)[m]$ | -0.13 | 0.13 | 0.13 | -0.13 |
| $\delta z(k)[m]$ | -0.13 | -0.13 | 0.13 | 0.13 |
| $\delta roll(k)[^\circ]$ | 0 | 0 | 0 | 0 |
| $\delta pitch(k)[^\circ]$ | -15° | -15° | 15° | 15° |
| $\delta yaw(k)[^\circ]$ | 15° | -15° | 15° | -15° |

Table 6.2: Input values for diagonal axes movements.

Note. The *movement set* in shortened form is given as:

$$MovementSet = \left\{ \begin{array}{l} \text{Straight, Left, Right, Up, Down,} \\ \text{DownLeft, DownRight, UpLeft, UpRight} \end{array} \right\} \quad (6.11)$$

The *implemented movement set example* (fig. 6.3) shows the movement used as basic building blocs of the trajectory for fixed-wing UAS:

1. *Initial position* (red plane) - the initial position, before any movement execution.
2. *Straight movement application* (blue plane) - the *neutral movement application* brings plane forward.
3. *Main axes movements* (cyan planes) - the application of movements from (tab. 6.1) $\{\text{Up, Down, Left, Right}\}$.
4. *Diagonal axes movements* (magenta planes) - the application of movements from (tab. 6.2) $\{\text{DownLeft, DownRight, UpLeft, UpRight}\}$.

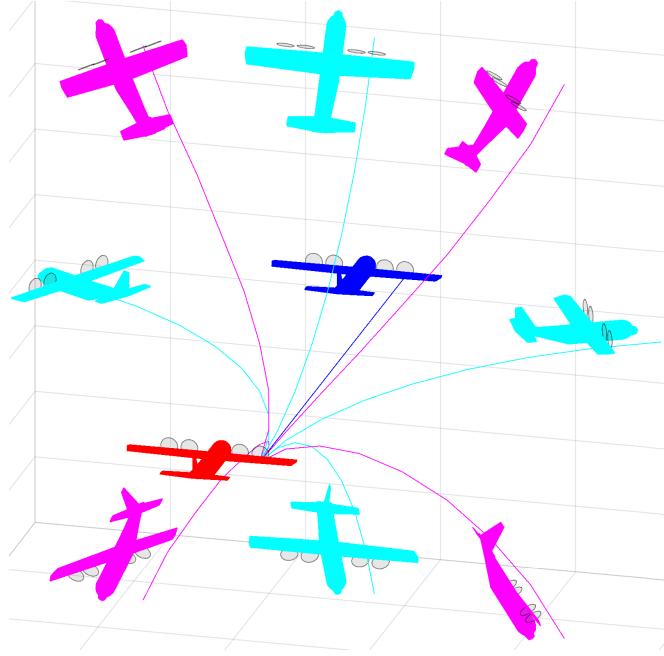


Figure 6.3: Implemented movement set example.

Trajectory by (def. 32) for initial time $time = 0$, initial state $state(0)$ and *Movement Buffer* (from def. 31):

$$Buffer = \left\{ movement(j) : \begin{array}{l} movement(j) \in MovementSet(\text{eq.6.11}), \\ j \in 1 \dots n, n \in N^+ \end{array} \right\} \quad (6.12)$$

Assumption 7. *The buffer is always non-empty, ordered, finite list of movements.*

Note. The buffer has finite count n of movements stored. The buffer is the planning instrument used by higher level navigation/avoidance algorithm to control UAS (Control/Command interface) (fig. 6.2).

The discrete trajectory (eq. 6.13) is ordered set of states bounded to discrete time $0 \dots n$, where n is movement count of *Buffer*. Trajectory set has $n + 1$ members defined like the following:

$$Trajectory(state(0), Buffer) = \left\{ \begin{array}{l} state(0) = state(0), \\ state(1) = applyMovement(state(0), movement(1)), \\ state(2) = applyMovement(state(1), movement(2)), \\ \vdots = \vdots \\ state(n - 1) = applyMovement(state(n - 2), movement(n - 1)), \\ state(n) = applyMovement(state(n - 1), movement(n)) \end{array} \right\} \quad (6.13)$$

The $movement(k)$ vector is selected from movement tables (tab. 6.1, 6.2).

Note. Parameter movement(\cdot) (eq. 6.13) is a movement order index in buffer (eq. 6.12).

6.3 Space Discretization - Avoidance Grid

Operation Space: The *Operation Space* is a space where UAS can effectively surveillance its surroundings, and it has the capability to act.

A Motivation for Discretization: The UAS surroundings needs to be represented in an *avoidance-friendly manner*, following principles matters:

1. *Discrete representation* - the space around UAS should be segmented into finite and exclusive portions which are considered as one point of the grid. This enables fast situation assessment.
2. *Threat proximity* - any form of threat gets more important with decreasing distance to UAS.
3. *LiDAR swipe density* - one LiDAR swipe scans many points; the grid needs to be customized to swipe characteristics.

The *Main Sensor* is *LiDAR* (problems 4.23.-5). The *effective occupancy computation* needs to be done for all problems; the inspiration is taken from [144]. The *effective occupancy computation* is done in *LiDAR* scan portioned into *polar coordinates grid*. The *operation space* is abstracted as a *grid* where *space portions* are representing the points in the grid.

Note. Each member of the grid is a cell, represented as a point with properties, like threat level, visibility.

The *Discrete Situation Evaluation* is executed for a *UAS* local coordinate frame in fixed *time*. The goal is to enable *fast discrete situation assessment*.

LiDAR Swipe: The *point* scanned by *LiDAR*, where the *UAS position* is center of the *local coordinate frame*, and *UAS heading is defining the main axes* is given as:

$$\text{point} = [\text{distance}, \text{horizontal}^\circ, \text{vertical}^\circ]. \quad (6.14)$$

Note. For polar/Euclidean transformations and local/global coordinate frames refer to background theory (app. A).

The *right side* of UAS $\text{horizontal}^\circ \in] -\pi, 0[$, the *left-side* of UAS $\text{horizontal}^\circ \in [0, \pi]$, the *down-side* of UAS $\text{vertical}^\circ \in] -\pi, 0[$, the *top side* of UAS $\text{vertical}^\circ \in [0, \pi]$

LiDAR Swipe Portioning: The *polar coordinate space* can be portioned into distinctive cells. Each cell then represents one point in the grid.

The *reason* for this swipe portioning is *LiDAR* scanning density¹, which is extremely dense. The *threat state* in the cell can be assessed with linear complexity.

¹Example rotary LiDAR Velodyne VL-16 specs: https://www.cadden.fr/wp-content/uploads/2017/02/Velodyne_VLP-16-Puck.pdf

The *polar → euclidean* coordinate frame transformation is not amenable for LiDAR swipe. The *threat assessment based on LiDAR swipe in planar space portions* has minimal complexity, and it is cost effective. [145].

Cell: To discretize operational space into a grid of points there is a need to define cell space, which bounds the portion of the *local planar coordinate frame*. The point (eq. 6.14) is defined by distance, horizontal $^\circ$ offset angle, and vertical $^\circ$ offset angle. The cell is a closed compact set of such points. The boundary can be defined like follow:

Definition 11. Cell

The cell bounds a portion of space in UAS local polar coordinate frame, defined by boundary ranges:

1. Distance Range - starts and ends: $distance_{start} < distance_{end}$ in \mathbb{R}^+ .
2. Horizontal Range - starts and ends: $horizontal_{start}^\circ < horizontal_{end}^\circ \in]-\pi, \pi]$.
3. Vertical Range - starts and ends: $vertical_{start}^\circ < vertical_{end}^\circ \in]-\pi, \pi]$.

The space portion belonging to the cell is given by function as:

`cell.spacePortion = ...`

$$\left\{ \begin{array}{l} \text{point} \in \mathbb{R}^3 \text{ where :} \\ \left(\begin{array}{lll} \text{cell.distance}_{start} < \text{point.distance} \leq \text{cell.distance}_{end}, \\ \text{cell.horizontal}_{start}^\circ < \text{point.horizontal}^\circ \leq \text{cell.horizontal}_{end}^\circ, \\ \text{cell.vertical}_{start}^\circ < \text{point.vertical}^\circ \leq \text{cell.vertical}_{end}^\circ \end{array} \right) \end{array} \right\} \quad (6.15)$$

To evaluate a static obstacle threat, it is necessary to know how many LiDAR hits landed in the cell space portion. For one LiDAR Scan the hits set is given a set of all points which lands into cell space portion:

$$cell.LiDARHits = \{\text{point} \in \text{LidarScan} : \text{point} \in \text{cell.spacePortion}\} \quad (6.16)$$

Note. The *cell* space portion volume is increasing with the distance. This satisfies the requirement for threat-distance importance.

Effective Operation Space: The goal is to determine which of the operation space is going to be considered in our avoidance grid. The effective operation space determination according to [146] is influenced by the following factors:

1. *Sensors ranges* - there is no reason to assess the situation over effective *sensor range*.

2. *Information sources impact* - there is no real impact on *effective space boundary*, the information search and intersection algorithms are only of the importance.
3. *UAS maneuverability* - the space where UAS can maneuver, bounded by space-time (reach set boundary).
4. *Computation power* - the situation evaluation and threat assessment capabilities of the onboard computer.
5. *Airworthiness requirements* - the *regulations* can impose some minimal requirements on *effective operation space boundary*.

Let show an example of an *effective operation space* for the UAS (fig. 6.4). The *full LiDAR Swipe* (cyan and red lines) of *UAS* (blue plane) has a *shape* of the conical cylinder.

Note. Under *ideal circumstances*, the *LiDAR swipe* would have a *ball shape*, but in real cases the *UAS body portion* where *LiDAR* is mounted is unused.

The *frontal portion* (red line) is a set of cells where *UAS* can make maneuvers. According to the *previous conditions*, there is no reason to consider a space portion out of this area.

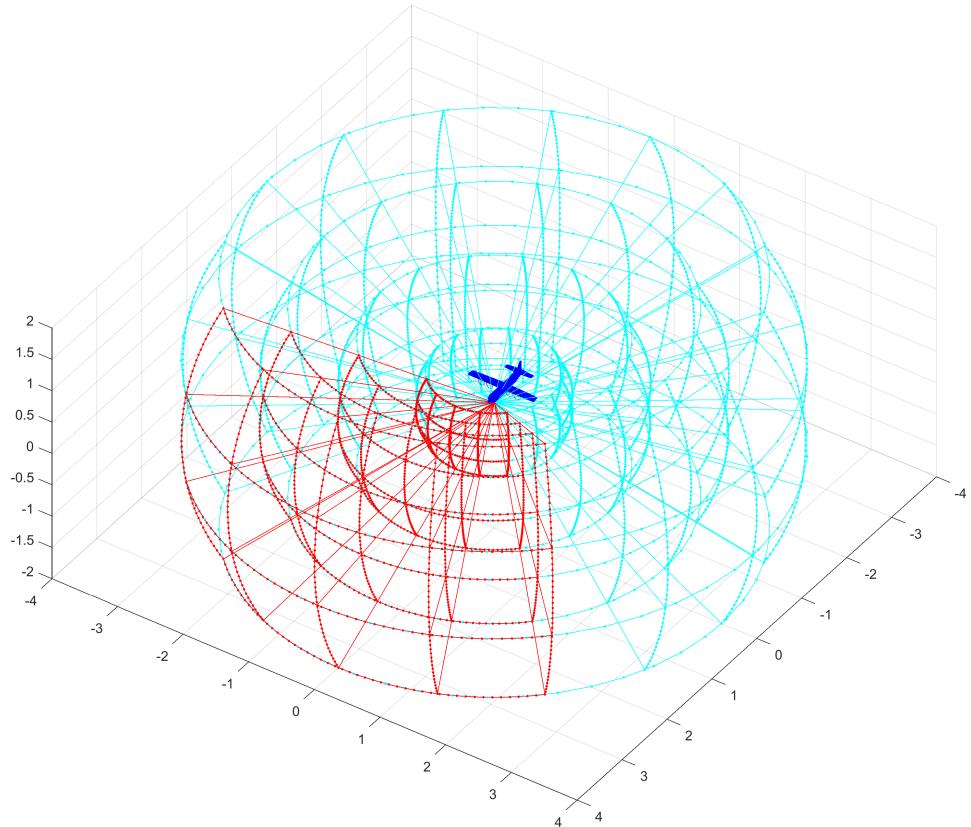


Figure 6.4: Example: The *LiDAR* reading portioning - cells.

Avoidance Grid Definition: The *effective operation space* is going to be portioned into cells.

Note. The avoidance grid is a set of cells from the full LiDAR swipe, that are reachable in limited number of movements by given control (sec. 6.2.3).

Definition 12. Avoidance Grid

The effective space portion (fig. 6.4 red lines) given by a portion of space in UAS local polar coordinate frame, bounded by:

1. Distance Range - in range $distance_{start} < distance_{end}$ in \mathbb{R}^+ .
2. Horizontal Range - in range by $horizontal_{start}^\circ < horizontal_{end}^\circ \in]-\pi, \pi]$.
3. Vertical Range - in range $vertical_{start}^\circ < vertical_{end}^\circ \in]-\pi, \pi]$.

The goal is to separate the effective operation space into cells (def. 11). The idea is to split distance range into multiple distinctive distance ranges with count $layerCount \in \mathbb{N}^+$. The ranges for distance layers are given as follow:

$$\begin{aligned} layer_{start}^i &= (i - 1) \times \frac{distance_{end} - distance_{start}}{layerCount} \\ layer_{end}^i &= i \times \frac{distance_{end} - distance_{start}}{layerCount} \quad ; \quad i \in 1 \dots layerCount \end{aligned} \quad (6.17)$$

The same separation Layer horizontal/vertical separations defined by $horizontalCount \in \mathbb{N}^+$ / $verticalCount \in \mathbb{N}^+$:

$$\begin{aligned} horizontal_{start}^j &= (j - 1) \times \frac{horizontal_{end}^\circ - horizontal_{start}^\circ}{horizontalCount} \\ horizontal_{end}^j &= j \times \frac{horizontal_{end}^\circ - horizontal_{start}^\circ}{horizontalCount} \quad ; \quad j \in 1 \dots horizontalCount \end{aligned} \quad (6.18)$$

$$\begin{aligned} vertical_{start}^k &= (k - 1) \times \frac{vertical_{end}^\circ - vertical_{start}^\circ}{verticalCount} \\ vertical_{end}^k &= k \times \frac{vertical_{end}^\circ - vertical_{start}^\circ}{verticalCount} \quad ; \quad k \in 1 \dots verticalCount \end{aligned} \quad (6.19)$$

Then $cell_{i,j,k}$ space portion by (def. 11) has the following ranges:

1. Cell Distance Range (eq. 6.17) depending on layer index i .
2. Cell Horizontal Angle Range (eq. 6.18) depending on horizontal angle index j .
3. Cell Vertical Angle Range (eq. 6.19) depending on vertical index k .

Note. The example of *Avoidance Grid Cells* is given in (fig. 6.4 red boundary).

The Avoidance Grid is the set of cells:

$$AvoidanceGrid = \left\{ \begin{array}{l} i \in 1 \dots layerCount \\ cell_{i,j,k} : j \in 1 \dots horizontalCount \\ k \in 1 \dots verticalCount \end{array} \right\} \quad (6.20)$$

Note. For any distinctive cells $cell_{i,j,k}$, $cell_{m,n,o}$ their *space portion intersection* is empty set:

$$\forall cell_{i,j,k}, cell_{m,n,o} : cell_{i,j,k} \cap cell_{m,n,o} = \emptyset, i \neq o \vee j \neq n \vee k \neq o \quad (6.21)$$

Grid Sizing Approach: The sizing approach used in this work is outlined in (app. H.1).

Cell in Avoidance Grid Properties: For each cell in the Avoidance Grid there are properties to be checked:

1. *Is there visibility to the cell?* - how good is an observation of the cell by Sensor Field.
2. *Is there threat present?* - how sure the data fusion is that there is eminent threat in the cell.
3. *Is the cell reachable?* - if there is any trajectory which can get UAS to that cell without too much threat along the way.

The answers to these questions are given later in *data fusion procedure* outline (tab. 6.3).

6.4 Reach Set Approximation

Summary: There is a need to have a tool with a finite count of trajectories, which has enough variability to support avoidance task. The reach set covers all possible trajectories, but it is not countable. Trajectories represented as tree originating in the same initial state can be considered as a skeleton of the reach set. There is a need to compare trajectories regarding avoidance capability. To achieve this, it is necessary to distinguish them based on measurable criteria. Different approximation based on the measurable criteria is introduced to cover different avoidance behaviors.

Motivation: *Reach set* is a strong tool for *Obstacle Avoidance* because it contains all possible *avoidance maneuvers*. The current implementations (sec. 3.2.2) have the following flaws:

1. *Realistic approximation - nonlinear systems or heavily constrained systems* cannot be approximated well by *linear continuous-time Reach Sets*.
2. *Finite count of possibilities - continuous-time Reach Set* contains infinite possibilities for *avoidance maneuvers*; the DAA system demands conflict resolution in finite-time.
3. *Computationally feasible data structures -* binding related properties seem problematic because *continuous- time reach sets* do not have unique identifier of maneuver, trajectory nor segment.

Proposed Solution Features: Our Reach set Estimation method will provide the following features:

1. *System Control Interface -* implemented via *Movement Automaton*, requiring only a *discrete command chain* to approximate system behavior.
2. *Finite count of possibilities -* finite number of elements in *Reach set* will enable *scalable calculation*.
3. *Computationally feasible data structures -* approximation of Reach set as a set of trajectories, each trajectory can be split into a finite number of segments. Each element will have a unique identifier enabling both-side property binding.
4. *Computationally feasible data-structures -* some specific behavior, like horizontal/vertical separation, or maneuver shape can be encoded into different types of reach set approximation algorithms.

6.4.1 Trajectory Set Approximation

Summary: To achieve finite representation, it is necessary to establish a general relationship between the reach sets and trajectory trees, built from a set of prototypical movements.

Discretization of Reach set: There is a need for a discrete finite *Reach Set approximation* to enable *Avoidance Strategy Evaluation* in finite-time. Replacing *Continuous Control Set Inputs*(t) by *Movement Automaton* is feasible:

Definition 13 (Reach set Approximation by Movement Automaton). A trajectory (*def. 32*) for system $\dot{\text{state}} = f(\text{time}, \text{state}, \text{input})$ under control of the movement automaton \mathcal{MA} is given as execution of movement buffer (*def. 31*) with an initial state of system state_0 . Therefore notation $\text{Trajectory}(\text{state}_0, \text{buffer})$ is used.

The Complete Reach Set (6.22) for system with initial state state_0 with existing control strategy $\text{control}(\text{time}) \in \text{Controls}(\text{time})$. for time $\tau > \text{time}_0$.

$$\text{ReachSet}(\tau, \text{time}_0, \text{state}_0) = \bigcup \{\text{state}(s) : \text{control}(s) \in \text{Controls}(s), s \in (\text{time}_0, \tau]\} \quad (6.22)$$

The Reach Set Approximation by Movement Automaton (6.23) of the system under the control of the movement automation \mathcal{MA} consist from the set of trajectories $\text{Trajectory}(\text{state}_0, \text{Buffer})$, which are executed in constrained time $\tau > \text{time}_0$.

$$\text{ReachSet}(\tau, \text{time}_0, \text{state}_0) = \left\{ \text{Trajectory}(\text{state}_0, \text{buffer}) : \begin{array}{c} \text{duration(buffer)} \\ \leq \\ (\text{time}_0 - \tau) \end{array} \right\} \quad (6.23)$$

Note. Reach Set Approximation (def. 13) is a subset of Full Reach Set (def. 2) in continuous space \mathbb{R}^n it inherits all important properties, like *Invariance* [147].

Discretization of Reach Set have been achieved leaving us with a *finite count* of Trajectories, instead of *Infinite subspace or \mathbb{R}^N*

Approximated Reach Set Containment: The *Approximated Reach Set* introduced in (def. 13) is constrained only by *future expansion time* τ . UAS makes space assessment in *Avoidance Grid*. There is no point to consider Trajectories outside of *Avoidance Grid*

Definition 14 (Contained Approximated Reach Set). For a pair $(\text{state}_0, \text{AvoidanceGrid}_0)$

at time $time_0$ and prediction horizon $\tau = \infty$ there is Contained Reduced Reach Set:

$$ReachSet \begin{pmatrix} time_0, \\ state_0, \\ AvoidanceGrid_0 \end{pmatrix} = \left\{ \begin{array}{l} Trajectory(\dots) \\ \in \\ ReachSet(6.23) \end{array} : \begin{array}{l} \forall segment \in AvoidanceGrid_0, \\ segment \in Trajectory(\dots) \end{array} \right\} \quad (6.24)$$

Properties: Container Approximated Reach Set *contains only trajectories where all segments belong to Avoidance Grid, there are following functions:*

1. *The membership function for any Trajectory in Constrained Reduced Reach set returns Ordered Set of Passing Cells.*
2. *The cost function for any Trajectory Portion in Constrained Reduced Reach Set return Cost of Execution*

Passing cell: Cell of Avoidance Grid which has some intersection with Trajectory.

Note. Contained Reduced Reach Set (eq. 6.24) which is contained in the Avoidance Grid and have a *Membership Function* enable Property transition between Reach set and *Avoidance grid*.

Example: Visibility from cells along Trajectory can be gathered to calculate Trajectory's feasibility.

Reach Set Pruning: There is a need to implement *Set Difference* between Reach Set and Constraint Set. Constraint Set can be an *Obstacle Set* from *Known World* (sec. 4.1.10) and other different constraints.

Reach Set Trajectory Tree: (6.25) Any Reach Set where *Control Strategy Constraint* is implemented as *Movement Automaton*, with defined *Movements* set and for single initial $state_0$. The Reach Set is given as discrete tree with root $Trajectory(state_0, \emptyset)$.

$$ReachSet(state_0, \dots) = \left\{ Trajectory(state_0, buffer) : \begin{array}{l} buffer \in Movements^i, \\ i \in \{1, \dots, k\} \end{array} \right\} \quad (6.25)$$

For each *Trajectory Segment*, there exists *intersection function* which evaluates as true if there exists at least one point in *Segment* which belongs to *Constraint Set*. Formally:

$$intersection(segment, Set) : \begin{cases} \exists point \in segment, & : true \\ point \in Set & \\ Otherwise & : false \end{cases} \quad (6.26)$$

Definition 15 (Pruned Reach Set). *For Reach set represented as Trajectory Tree (eq. 6.25) and some constraint set (Set) where exist intersection function (eq. 6.26). The Pruned Reach set is given as follows:*

$$\text{Prune}(\text{ReachSet}, \text{Set}) = \left\{ \text{Trajectory}(\dots) : \begin{array}{l} \forall \text{segment} \in \text{Trajectory}, \\ \neg \text{intersection}(\text{segment}, \text{Set}) \end{array} \right\} \quad (6.27)$$

Note. Pruning(def. 15) [148] is applied multiple times for various *Constraints Set*.

Example of *Approximated Reach set Calculation* (def. 13), *Reach Set Containment* (def. 14), and, *Pruning* is given in [9].

6.4.2 Distinctive Properties of the Trajectories

Summary: A characterization of trajectories is provided to support the selection of feasible representatives to build a skeleton of the reach set.

Motivation: The need to Make *Reach Set* scalable approach. This may be a problem due to the *Expansion rate*. *Reach set* represented as a *Trajectory Tree* (eq. 6.25) for Avoidance Grid with *layer – count* and Movement automaton with *movement – count*, the *Node count* is given as:

$$1 + \left(\sum_{i \in \{1 \dots \text{layerCount}\}} (\text{movementCount})^i \right) \quad (6.28)$$

This scaling is not feasible for *Avoidance Grid* with many layers (< 10) or *Movement Set* with many movements (< 9). There is a need for *Reduced Reach set calculation*.

Performance Criteria: The scaling factor (eq. 6.28) shows that there are going to be many trajectories. The main point is that not every trajectory in *Reach Set* is giving us *maneuverability advantage*. Our expectations lie in following *Performance Requirements*:

1. *Reach set* must *Cover* maximum of the *possible unique maneuvers* in *Avoidance Grid*.
2. *Trajectories* in *Reach Set* should be smoothest possible to prevent cargo damage / UAS wear.

Trajectory footprint: Discrete space of *Avoidance Grid* is organized in cells. The *cell* is a minimal space portion accessible by *property binding*. There is a need to know if two trajectories contribution to *Maneuverability* in this environment.

Each trajectory passes through space in *Avoidance Grid*. If there exists a method to extract unique identifier for each *trajectory passed cells*, we can compare two trajectories *Coverage* in *Avoidance Grid*.

Definition 16 (Trajectory footprint). *For Trajectory from Reach set (def. 14) defined for Avoidance Grid has membership function. Membership Function returns ordered set of passing cells:*

$$\text{footprint} \left(\begin{array}{c} \text{Trajectory}, \\ \text{AvoidanceGrid} \end{array} \right) = \left\{ \begin{array}{l} \text{cell} \in \text{AvoidanceGrid} : \\ \quad \text{isMember}(\text{trajectory}, \text{cell}) \end{array} \right\} \quad (6.29)$$

Then we can define equality function for Trajectory₁ and Trajectory₂, as the comparison of their footprints in common Avoidance Grid as follow:

$$\text{isEqual} \left(\begin{array}{c} \text{Trajectory}_1, \\ \text{Trajectory}_2, \\ \text{AvoidanceGrid} \end{array} \right) : \left\{ \begin{array}{ll} \left(\begin{array}{c} \text{footprint}(\text{Trajectory}_1, \dots) \\ = \\ \text{footprint}(\text{Trajectory}_2, \dots) \end{array} \right) & : \text{true} \\ \text{Otherwise} & : \text{false} \end{array} \right. \quad (6.30)$$

Note. Depending on Movement Automaton's movement set and Avoidance Grid parameters, there can be multiple trajectories which are equal.

Coverage set: Now it is possible to create a set of unique trajectory footprints due to footprint function (eq. 6.29). Similarly, there is a possibility to create Reach set skeleton containing unique trajectories, by using equality function (eq. 6.30). Coverage set is sufficient for now.

Definition 17 (Coverage Set). Coverage set (6.31) is defined for Avoidance Grid and Reach Set pair as a set of unique Trajectory footprints:

$$\text{CoverageSet} \left(\begin{array}{c} \text{AvoidanceGrid}, \\ \text{ReachSet} \end{array} \right) = \left\{ \text{footprint} \left(\begin{array}{c} \text{Trajectory}, \\ \text{AvoidanceGrid} \end{array} \right) : \forall \text{Trajectory} \in \text{ReachSet} \right\} \quad (6.31)$$

Coverage set properties: Trajectory footprint (eq. 6.29) is not a bijection, neither injection for ReachSet → CoverageSet. This implies the following properties:

1. Equal Reach Sets in same Avoidance Grid have equal Coverage Sets.
2. Equal Coverage Sets does not imply Reach Set equality.
3. For two Coverage Sets, there is a possibility to compare their member count to create coverage ratio.

The second Property gives us a proposition that there is a possibility of Reach Set Reduction without losing Coverage.

Definition 18 (Coverage Ratio). Coverage Ratio is a ratio of Coverage Set Member Count between two Reach Sets. Reach set with a lesser count of unique Trajectories is considered as Reduced Reach Set. Reach set with greater Count of unique Trajectories is considered as Reference Reach Set.

$$\begin{aligned} \text{referenceCoverage} &= |\text{CoverageSet}(\text{ReferenceReachSet}, \text{AvoidanceGrid})| \\ \text{reducedCoverage} &= |\text{CoverageSet}(\text{ReducedReachSet}, \text{AvoidanceGrid})| \\ \text{CoverageRatio} &= \frac{\text{reducedCoverage}}{\text{referenceCoverage}} \in [0, 1] \end{aligned} \quad (6.32)$$

Note. Reference Reach Set is usually Full Reach Set containing all possible trajectories in space contained by Avoidance Grid. In case Full Reach Set cannot be computed, Avoidance Grid is too large, most complex Reach Set is used as Reference Reach Set.

Trajectory smoothness: Trajectory other than straight line have some changes in UAS heading.

The goal is to minimize Maneuvering of UAS, because:

1. Every Heading Change needs to be reported to UTM.
2. Sharp Maneuvering can damage cargo/wear UAS.
3. Often course changes make Intruder prediction harder for other Civil General Aviation.

For this purpose, Smoothness Metric needs to be applied for Reach Set or Trajectory. In the case of Movement Automaton Control, two distinguish Movement Sets are introduced: Smooth and Chaotic movements set with the following properties:

$$\begin{aligned} \text{MovementSet} &= \text{SmoothMovements} \cup \text{ChaoticMovements} \\ \text{SmoothMovements} \cap \text{ChaoticMovements} &= \emptyset \\ |\text{SmoothMovements}| > 0, \quad |\text{ChaoticMovements}| > 0 \end{aligned} \quad (6.33)$$

Then Smoothnes clasifier for Trajectory(*initialState, buffer*) can be defined as *isSmooth* and Smooth Movement Counter function as *smoothCount* like follow:

$$\begin{aligned} \text{isSmooth}(\text{movement}) &= \begin{cases} \text{movement} \in \text{SmoothMovements} & : 1 \\ \text{movement} \in \text{ChaoticMovements} & : 0 \end{cases} \\ \text{smoothCount}(\text{Trajectory}(\dots, \text{buffer})) &= \sum_{\forall \text{movement} \in \text{Buffer}} \text{isSmooth}(\text{movement}), \end{aligned} \quad (6.34)$$

Definition 19 (Smoothness Rating for Trajectory). Smoothness for trajectory generated by Movement Automaton for some Initial State with some Movement Buffer, under the assumption of Smooth and Chaotic Movement Set split (eq. 6.33), with existing classification and counter functionals (eq. 6.34) is given as follows:

$$\text{Smoothness}(\text{Trajectory}(\dots, \text{buffer})) = \frac{\text{isSmooth}(\text{Trajectory})}{\text{movementCount}(\text{Trajectory})} \in [0, 1] \quad (6.35)$$

For Trajectory with $\text{buffer} = \emptyset$ Smoothness is given as 1.

6.4.3 Heuristic Trajectory Tree Building

Summary: There is a need for building tool to create trajectory set based reach set approximation iteratively. The iterative procedure is based on wave-front expansion algorithm. The node selection criteria for trajectory construction is used to build performance-based reach set approximations.

Motivation: Purpose of Navigation is to move forward to *Goal Waypoint* in *Mission*. Structure of *Avoidance Grid* is designed to enable *forward* and *turning* maneuvers. The *Avoidance Grid* is organized in *Layers* characteristic by the same distance from *Avoidance Grid Origin*.

Survey of motion planning algorithm was given in [149]. The ideal candidate for propagation algorithm is *Wave-front* algorithm propagating *Trajectory tree* through Layers. Due to the *Avoidance Grid* onion-like layers, there is a possibility to implement turn maneuver through layers iterative and effectively.

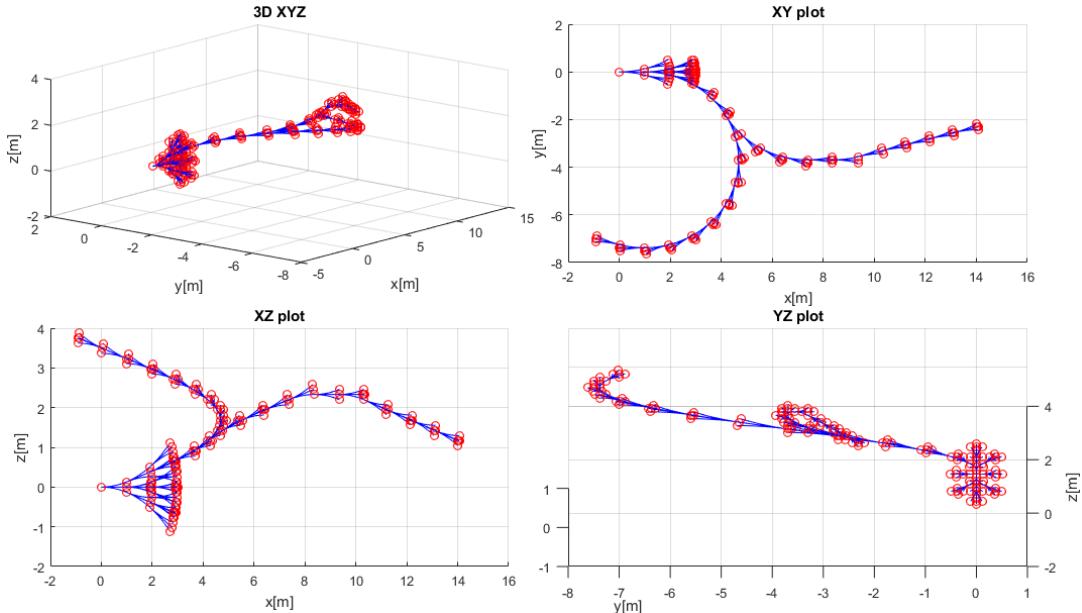


Figure 6.5: *Rapid Exploration tree as a result of Constrained trajectory expansion.*

Rapid Exploration Tree (fig. 6.5) was selected because it enables *Movement Automation Utilization* and *Property Binding*. A similar approach was used for space exploration [150].

The example (fig. 6.5) shows a *Rapid Exploration Tree* in *Free Space* containing *Waypoint Navigation Path* and *Turn Away Path*. Both paths are starting in same *Root Node* (red circle) which was expanded with simple *Movement Automaton* (a bunch of nodes originating from one node is showing the way of expansion). The connection (blue line) between two nodes (red circles) represents *Trajectory portion* for *Executed Movement*.

Rapid Exploration Tree Node will contain the following information:

1. *Initial state* - root entry point, used in state evolution calculation.
2. *Trajectory (state evolution)* - trajectory passing through *state space* in the local coordinate frame of *Avoidance Grid*.
3. *Buffer* (applied movements) - ordered list of *executed movements* applied on the *initial state* to obtain *state evolution*.
4. *Cost* - calculated for *state evolution* based on a *predefined cost function*.
5. *Footprint* - ordered set of *passing cells* in *Avoidance Grid*.
6. *Parent Node Reference* - tree reference for the parent node, not in case of the *root node*.
7. *Other Bounded Properties* - value list of other properties, depending on *Expansion Constraints* and *Reachability* evaluation algorithm.

Wave-front propagation of Rapid Exploration Tree is given in (alg. 6.1).

The *Avoidance Grid* have UAS with *position* \in *Initial State* at the *origin*. The *Grid Layer* is a column ordered set of cells with same *Mean distance* from the origin. *Grid Layers* are indexed from origin starting with 1; there is a maximum of $i \geq 1$ layers.

Step: Initialization contains base structure preparation like follows:

1. *Avoidance Grid* - space containing *Reach set* (def. 14).
2. *Movement Automaton* - Used as *Predictor*, consuming *buffer* containing *Movements* to generate *Trajectory(initialState, buffer)*.
3. *Reach Set* - tree consisting from *Wave-frontNodes* representing the endpoint of *Trajectory(initialState, buffer)* where each *Edge* represents *one Movement application*. The root is set as a node containing *Initial State*.

Function *initializeReachSet(root, stack, grid, automaton)* will take root and enforces *full waveform propagation* to *First Layer*.

Step: Wave-front Propagation is forced propagation of trajectories from layer i to layer $i + 1$. The process goes as follows:

1. *Selection of Feasible candidates* - function $[candidates, leftovers] = ExpansionConstraints.select(stack)$ for working layer, row and cell selects *feasible trajectory nodes* ordered by *Cost function*. The *Example of Cost Function* can be *Trajectory Smoothness* (def. 19).
2. *Expansion of Candidates* - for each *candidate* function *candidate.expandNode(automaton)* is invoked. This function will expand *Candidate Node structure* by appending *Full Trajectory Tree Evolution* until each *Leaf Trajectory* reaches *Next Layer*. Simply put *Parent Node Node(initialState, buffer, cost, footprint)* buffer is appended by movements until the next layer is reached.
3. *Leftovers purge* - function *reachSet.purge(leftovers)* removes unexpanded *Nodes* leading to cell, effectively removing trajectories which do not lead to the *next layer*.
4. *Append Reach Set* - function *reachSet.append(leafs)* puts newly created *Nodes* (*Trees*) into *Reach Set* structure. The *Wave-front Propagation* for one cell is finished.

Step: After Layer Propagation Purge is covered by function *reachSet.purgeSameFootprint()* which takes trajectories with the same footprint and keeps some of them based on *Selection criteria*, more in (sec. 6.4.4, 6.4.5). *Pruning methods* over *Large Decision Trees* are *fast and viable* [151].

Note. *Reach Set* is usually computed *Prior the Flight* for *some Initial State* in *Local Coordinate Frame* in *right had coordinate frame* with X^+ used as *main axis*.

Algorithm 6.1: *Wave-front propagation of Rapid Exploration Tree to form Reach Set.*

Input : Node(initialState,buffer=∅,cost=0,footprint=∅), AvoidanceGrid,
ExpansionConstraints, MovementAutomaton(movementSet)

Output: ReachSet(AvoidanceGrid)

```

# Initialization Sequence;
grid=AvoidanceGrid, automaton=MovementAutomaton, root = Node;
reachSet = initializeReachSet(root,stack,grid,automaton);

# Main Expansion through, layers (i), rows (j), cells(k);
for layer(1...i) in grid do
    for row(1...j) in layer do
        for cell(1...k) in row do
            # apply selection criteria ;
            [candidates,leftovers] = ExpansionConstraints.select(stack);

            # collect expansions ;
            leafs = [];

            for candidate in Candidates do
                | leafs= [leafs, candidate.expandNode(automaton)];
            end

            reachSet.purge(leftovers);
            reachSet.append(leafs);

        end
    end
    reachSet.purgeSameFootprint();
end

```

6.4.4 Coverage-Maximizing Reach Set Approximation

Summary: This procedure will produce trajectories maximizing aggressive maneuvering, ideal for avoidance tasks.

Motivation: Design of calculation method for *Reach Set Approximation* guarantying high *Maneuverability*.

Background: There is *Coverage Ratio* property of *Reach Set* (def. 18). It has been shown that creating *Reach Set* via *greedy approach* is not feasible due to the *Scaling Factor*. *Contracted Expansion* (sec. 6.4.3) is enabling to apply selection criteria while building *Reach Set* in given *Cell*.

The *Cell* $cell_{i,j,k}$ has a center and walls from UAS viewpoint: a front wall, back wall (for $layer > 1$), a top wall, left wall, right wall, bottom wall. It is expected that trajectory

leading close to one cell walls will continue to a different cell, increasing the chance to obtain more *Unique Footprints*.

Expansion Constraint Function Implementation (alg. 6.2) is based on the simple principle: *Select candidate Nodes which are closest to outer walls of Cell, with a unique footprint.*

Tuning Parameters: *Proximity to Cell outer wall* gives good chances to break into other rows or columns in the *Avoidance Grid*. *Unique footprint* guarantees future *Unique Footprint* after appending Trajectory by *Movement application*.

1. *Considered Footprint Length* - how much last cells in footprint should be considered in unique path track, minimal value 1, default value 3, maximal value ∞ . If there is a need to generate non-redundant trajectories use ∞ , it will consider full footprint.
2. *Spread Limit* - the upper limit of candidates which are going to be select for further expansion, minimal value 1, default value *Count of unique Moves in Movement set*, maximal value ∞ . If more than default values are selected, the algorithm will generate *redundant trajectories*. If less is selected, then some trajectories are omitted, and *Coverage Rate* decreases sharply.

Step: Initialization initialization of *candidate* array (return value), *leftovers* array (return Value). Node array *passing* is populated with *Nodes* which represents *end node of Trajectory*, and the tip of the *trajectory is constrained in the cell_{i,j,k}*.

Step: Evaluate best trajectories with unique Footprints following steps are executed:

1. *Best Performance Map* is created with a *footprint* as a key set element to ensure footprint uniqueness.
2. *Wall distance* for the *test node* is calculated as a closest trajectory portion distance to the *top, bottom, left, right* wall of the *cell_{i,j,k}*
3. The *Footprint* for the *test node* is created with the maximal length given by *Footprint Length* tuning parameter.
4. *Existence and Performance Test* is executed to ensure that the best performing node is selected. If there is no key entry in the *Best Performance Map*, then a new entry for *Test Node* is created. If there is a key entry, the performance of *Old Node* and *Test Node* is compared, and better is stored.

Step: Select candidates is executed on *Best Performance Map* records using *Wall distance* as pivot parameter, ordering by closest proximity and limited by *Search Limit* tuning parameter. The *Leftovers* are difference set between *Passing Nodes* and *Candidate Nodes*.

Algorithm 6.2: Expansion Constraint function for *Coverage-Maximizing Reach Set Approximation*

```

Input : Node[] stack, Cell celli,j,k
Tuning Parameters: int+ footprintLength, int+ spreadLimit
Output : Node[] candidates, Node[] leftovers

# Initialize structures;
Node[] candidates = [], Node[] leftovers= [];
Node[] passing = celli,j,k.getFinishingTrajectories(stack);

# Select best performing trajectories with unique footprint;
Map<Footprint,Node> bestPerformanceMap;
for Node test ∈ passing do
    wallDistance= test.minimalDistanceToWall(celli,j,k);
    footPrint = test.getFootprint(lastCells = footprintLength);
    if bestPerformanceMap.contains(footPrint) then
        old = bestPerformanceMap.getByKey(footprint);
        oldPerformance= old.minimalDistanceToWall(celli,j,k);
        if oldPerformance > wallDistance then
            | bestPerformanceMap.setByKey(footprint,test);
        end
    else
        | bestPerformanceMap.setByKey(footprint,test);
    end
end

# Select best performing nodes up to spreadLimit count;
candidates = bestPerformanceMap.select(count =
    spreadLimit).orderBy('wallDistance','Ascending');
leftovers = passing - candidates;
return [candidates,leftovers]

```

Example: for *Avoidance Grid* with *Distance 10 m*, *Layer count 10*, *Horizontal range* $[-45^\circ, +45^\circ]$, *Horizontal Cell Count 7*, *Vertical range* $[-30^\circ, +30^\circ]$, and *Vertical Cell Count 5*. Is given in (fig. 6.6). The UAS is at *Back-side* of *Figure* (initial state is at all *Trajectory Origins*). The *black dashed line* marks *Avoidance Grid space boundary*. Each trajectory has its own color and ends at *Front-side* of *Avoidance Grid Boundary*.

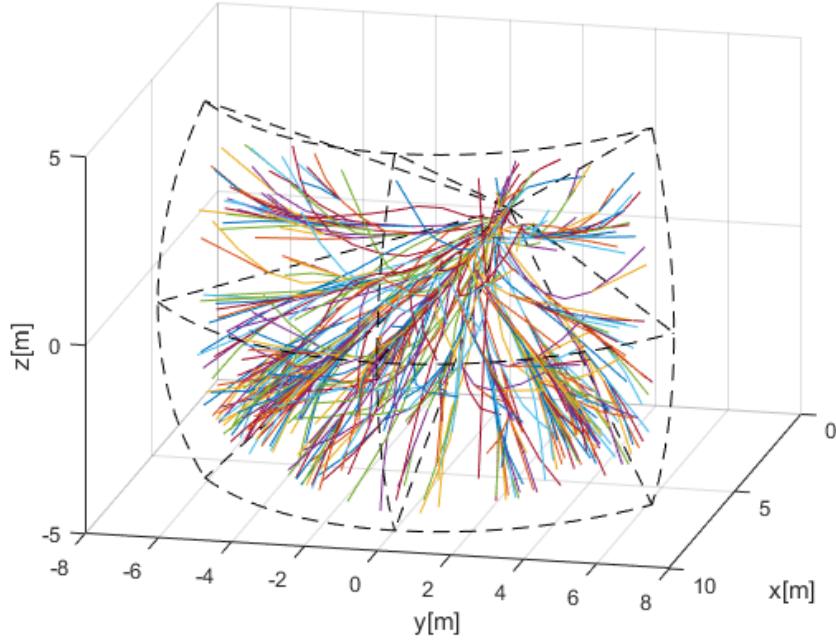


Figure 6.6: *Coverage-Maximizing* reach set approximation.

Pros and Cons: It can be seen from example (fig. 6.6) that *Coverage-Maximizing Reach Set Approximation Method* (alg. 6.2) generates much *turning* and *shaky trajectories*.

High Coverage Ratio (~ 0.9) is provided while keeping *medium node count*. The calculation complexity scales linearly with grid size. The *upper limit of trajectories* is given as follow:

$$\begin{aligned} \text{countTrajectories}(\text{ReachSet}) &\leq \text{layerCount} \times \text{spreadLimit} \\ &\quad \times \text{size}(\text{Movements}) \end{aligned} \quad (6.36)$$

The *upper limit of nodes* is given as follow:

$$\begin{aligned} \text{countNodes}(\text{ReachSet}) &\leq \text{layerCount} \times \text{layerCellCount} \\ &\quad \times \text{size}(\text{Movements}) \times \text{spreadLimit} \end{aligned} \quad (6.37)$$

The *absence of Smooth Trajectories* disqualifies *Coverage Maximizing -RSA* to be used for *Navigation*. This type of reach set is feasible for *Avoidance* because it contains a variety of maneuvers.

6.4.5 Turn-Minimizing Reach Set Approximation

Summary: This procedure will produce trajectories minimizing aggressive maneuvering, ideal for navigation tasks.

Motivation: Imagine having an *Avoidance Grid* like (fig. 6.4). There is a need of *Reach Set Approximation* which will have *Smooth Trajectories* (def. 19) going nearby *cell centers*.

Background: The *Smoothness Rating for Trajectory* (def. 19) uses two distinct sets *Smooth Movements* and *Chaotic Movements* (eq. 6.33) which are defined for our *Movement Automaton* (sec. 6.2) like the following:

$$\begin{aligned} \text{SmoothMovements} &= \{\text{Straight}\} \\ \text{ChaoticMovements} &= \text{Movements} - \text{SmoothMovements} \end{aligned} \tag{6.38}$$

Smooth Movements contains only *Straight* movement because others are considered as extreme turning movements. *Smooth Movements* should contain only direct flight movements or slight heading correction. *Chaotic Movements* set is a supplement of *Movement Automaton's Movement Set*.

The *Avoidance Grid* (fig. 6.4) cell centers for fixed indexes j_{fix} , k_{fix} are linearly aligned with the *initial state*. That means that cell centers of cells $cell_{1,j_{fix},k_{fix}}, \dots, cell_{i,j_{fix},k_{fix}}$, where i is a count of *layers* lie on one line. If the trajectory can achieve *cell center* on some *layer*, only minor trajectory corrections are required to stay on the given line. This type of trajectory gives us the following advantages:

1. *Minimal steering at the beginning* - the minimal steering is advantageous in *Controlled Airspace* because it diminishes the amount of communication to *UTM Service*.
2. *Additional safe space in the linear segment* - once the *center of the cell* is reached, *Trajectory* sticks to the line between cell centers. Each point on this line has the *maximal distance* to outer walls of the cell. This gives us extra space given as minimum of distance between *UAS position* and *Outer cell walls*.

Expansion Constraint Function Implementation (alg. 6.3) is based on the simple principle: *Select candidate Nodes which are closest to Cell center, with a unique footprint*.

Note. *Cell center* can be closely reached by *smooth movement* from a previous cell or *chaotic movement* from a neighboring cell from the current or previous layer. These trajectories are usually equivalent in *Smoothness*.

Algorithm 6.3: Expansion Constraint function for *Turn-Minimizing Reach Set Approximation*

Input : Node[] stack, Cell cell_{i,j,k}

Tuning Parameters: int⁺ spreadLimit

Output : Node[] candidates, Node[] leftovers

```

# Initialize structures;
Node[] candidates = [], Node[] leftovers= [];
Node[] passing = celli,j,k.getFinishingTrajectories(stack);

# Select unique smoothest trajectories;
Map<Buffer,Node> bestPerformanceMap;
for Node test ∈ passing do
    centerDistance= test.getPerformance(celli,j,k];
    footPrint = test.getFootprint();
    if bestPerformanceMap.contains(footPrint) then
        old = bestPerformanceMap.getByKey(footprint);
        oldPerformance= old.getPerformance(celli,j,k);
        if oldPerformance > centerDistance then
            | bestPerformanceMap.setByKey(footprint,test);
        end
    else
        | bestPerformanceMap.setByKey(footprint,test);
    end
end

# Select best performing nodes up to spreadLimit count;
candidates = bestPerformanceMap.select(count =
    spreadLimit).orderBy('cellCenterDistance','Ascending');
leftovers = passing - candidates;
return [candidates, leftovers]

```

Tuning Parameter: *Proximity to Cell Center* gives a good chance to keep trajectory smooth or *smooth after one correction maneuver*. It has been mentioned that *Cell Center* can be reached by various trajectories. In this method full footprint length is always considered; therefore only one tuning parameter can be offered:

1. *Spread Limit* - the upper limit of candidates which are going to be selected for further expansion, minimal value 1, default value *Count of unique Moves in Movement set*, the maximal value ∞ . If maximal value ∞ is selected, the algorithm will generate the skeleton of *Reach Set* with full Coverage and with the smoothest *Trajectories*.

Step: Initialization sets candidate *Nodes* as empty set, leftover *Nodes* as empty set, and selects all *Nodes* from *Stack* which represents *Finishing Trajectories* in working cell $cell_{i,j,k}$.

Step: Evaluate smoothest trajectories with unique Footprints is implemented as *multi-criteria filtration*.

The *first criterion* is the *distance to Cell Center* which is penalized by trajectory *smoothness rate* implemented in method $Node.getPerformance(Cell cell_{i,j,k})$ defined as follow.

$$getPerformance(Node, Cell) = \frac{distance(Node.Trajectory, Cell.Center)}{SmoothnessRate(Node.Trajectory)} \quad (6.39)$$

Distance of *Trajectory* is *enumerator* because its considered as the *base value* and is defined in the interval $[0, maximalWallDistance]$. The *Smoothness Rate* is the denominator, because it is a penalization coefficient defined in the interval $[0, 1]$.

The *second criterion* is *trajectory uniqueness*. This is provided by *Best Performance Map*, where best performing *Node* belongs to one unique *trajectory footprint*. The implementation is identical to *coverage-maximizing set expansion* (alg. 6.2).

Step: Select candidates is executed on *Best Performance Map* records using *Penalized Cell Center Distance* as pivot parameter, ordered in ascending order and limited by *Spread Limit* tuning parameter. The *Leftovers* are difference set between *Passing Nodes* and *Candidate Nodes*.

Example: for *Avoidance Grid* with *Distance 10 m*, *Layer count 10*, *Horizontal range* $[-45^\circ, +45^\circ]$, *Horizontal Cell Count 7*, *Vertical range* $[-30^\circ, +30^\circ]$, and *Vertical Cell Count 5*. Is given in (fig. 6.7). The UAS is at *Back-side* of *Figure* (the initial state is at all *Trajectory Origins*). The *black dashed line* marks *Avoidance Grid space boundary*. Each trajectory has its color and ends at *Front-side* of *Avoidance Grid Boundary*. The *Spread Limit*, in this case, was set to 9 which is *Size of the Movement Set*.

Note. Please note *Trajectories* are organized in bundles going around *Cell Centers smoothly*. Most of the steering maneuvers are executed at the *beginning* of the *Avoidance Grid*.

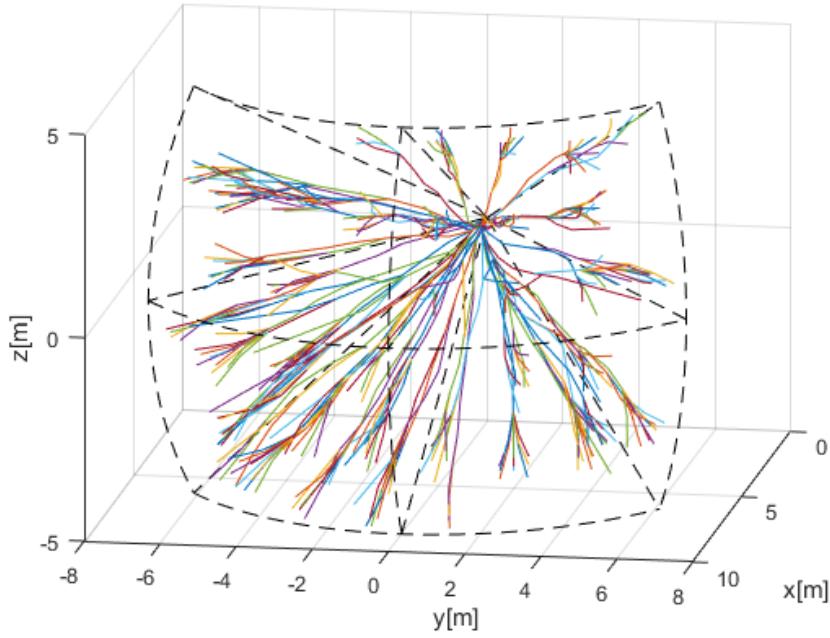


Figure 6.7: Turn-minimizing reach set approximation.

Pros and Cons: It can be seen from example (fig. 6.7) that *Turn-Minimizing Reach Set Approximation Method* (alg. 6.3) generates *smooth evenly spread trajectories*.

High smoothness ratio (≥ 0.9) is provided while keeping low node count for UAS systems. The calculation complexity scales linearly with grid size. The upper limit of trajectories is given as follow:

$$\begin{aligned} \text{countTrajectories}(\text{ReachSet}) &\leq \text{layerCellCount} \times \text{spreadLimit} \\ &\quad \times \text{size}(\text{Movements}) \end{aligned} \quad (6.40)$$

The *upper limit of nodes* is given as follow:

$$\text{countNodes}(\text{ReachSet}) \leq \text{layerCount} \times \text{layerCellCount} \times \text{spreadLimit} \quad (6.41)$$

The absence of *High Coverage Ratio* disqualifies *Turn-Minimizing Reach Set Approximation* to be used for *Emergency Avoidance*. This type of *Reach Set* is feasible for *Open Space Navigation* or *Controlled Airspace Navigation*. Its low turning rate in contained *Trajectories* are desired for such tasks.

6.4.6 ACAS-X like Reach Set Approximation

Summary: This procedure will produce trajectories encoding ACAS-X separation modes, ideal for controlled airspace navigation.

Motivation: The implementation of *ACAS-Xu* behavior in DAA system will be mandatory for *National Airspace System Integration* in United spaces [152].

Implementation of ACAS-Xu like behavior increase usability of approach, if it can be achieved without major concept changes.

Background: The *ACAS-Xu* system on the operational level has been described in [25]. The *Policy for Collision Avoidance* proposal has been given in [153].

Some behavioral patterns can be encoded into *Reach Set*. ACAS-Xu navigation part is basically *Look-up table of Maneuvers for Allowed Separations*.

The *Evasive Maneuver* selection process in ACAS-Xu is similar to our approach: *Select most energy efficient maneuver in compliance with space-time constraints*. ACAS-Xu intruder model is similar to our *Body Volume Intersection Model* (app. C.2). The *ACAS-Xu* defines following base separations:

1. *Horizontal* - movements on a *Horizontal Plane* in *Global Coordinate System*.
2. *Vertical* - movements on a *Vertical Plane* in *Global Coordinate System*.

There are allowed custom separations which can be used, for further experimentation:

1. *Slash* - movement on $+45^\circ$ *Tilted Plane to Horizontal Plane* in *Global Coordinate System*.
2. *Backslash* - movement on -45° *Tilted Plane to Horizontal Plane* in *Global Coordinate System*.

For given *Movement Automaton* implementation (sec. 6.2) the separations are given as follow:

$$\begin{aligned}
 \text{Horizontal} &= \{\text{Straight}, \text{Left}, \text{Right}\} \\
 \text{Vertical} &= \{\text{Straight}, \text{Up}, \text{Down}\} \\
 \text{Slash} &= \{\text{Straight}, \text{UpLeft}, \text{DownRight}\} \\
 \text{Backslash} &= \{\text{Straight}, \text{UpRight}, \text{DownLeft}\}
 \end{aligned} \tag{6.42}$$

For each *Node*(..., *buffer*) and each *separation* there is a evaluation function *isSeparation* which decides, if *Trajectory* defined by node buffer is made up only from *Separation* movements. The function *isSeparation*(...) is defined like:

$$isSeparation(buffer, separation) = \begin{cases} \forall movement \in buffer, & : true \\ \quad movement \in separation \\ otherwise & : false \end{cases} \quad (6.43)$$

Following *Separation Modes* can be defined with given *separations*:

1. *Horizontal* (ACAS-X defined mode) containing *horizontal* separation.
2. *Vertical* (ACAS-X defined mode) containing *vertical* separation.
3. *Horizontal-Vertical* (ACAS-X defined mode) containing *horizontal, vertical* separations.
4. *Full* (custom defined mode) containing all *Separation Modes*.

Note. Every separation modes generate 2D trajectories set on *Respective plane*. There is no need for *Tuning parameters* for further *Expansion Constraint*.

Expansion Constraint Function Implementation (alg. 6.4) is based on the simple principle: *Select only candidate Nodes which Trajectories have at least one desired Separation Mode*.

Step: Initialization sets candidate *Nodes* as the empty set, leftover *Nodes* as the empty set, and, select all nodes to form a *stack* which represents *Finishing Trajectories* in working $cell_{i,j,k}$,

Step: Candidate Selection Process is evaluated for each *test Node* from *passing Node Set*.

For each *applicable separation*, given as input parameter *separations*, The test function *isSeparation* (eq. 6.43) is applied:

1. If *test Node* trajectory belongs to at least one allowed separation it is added to candidates set.
2. Else is added to *Leftovers*.

Note. *Separation sets* (eq. 6.42) are not *exclusive sets* in *Movement Automaton* domain. One *Trajectory* contained by *Node* can belong to multiple *Separations*.

Algorithm 6.4: Expansion Constraint function for *ACAS-like Reach Set Approximation*

Input : Node[] stack, Cell $cell_{i,j,k}$, Separation[] separations

Tuning Parameters: $None : \emptyset$

Output : Node[] candidates, Node[] leftovers

```

# Initialize structures;
Node[] candidates = [], Node[] leftovers= [];
Node[] passing = celli,j,k.getFinishingTrajectories(stack);

# Select nodes containing trajectories with usable separations;
for Node test  $\in$  passing do
    for separation  $\in$  separations do
        # Get separations for Node;
        Separations[] nodeSeparations = test.getSeparations();
        # If trajectory given by buffer is on Separation plane;
        if isIn(isSeparation(test.buffer, separation)(6.43) then
            | candidates.append(test);
        end
    end
    # If there was no applicable separation, throw Node away;
    if test  $\notin$  candidates then
        | leftovers.append(test);
    end
end

# Return results;
return [candidates, leftovers]

```

Example: for *Avoidance Grid* with *Distance 10 m*, *Layer count 10*, *Horizontal range* $[-45^\circ, +45^\circ]$, *Horizontal Cell Count 7*, *Vertical range* $[-30^\circ, +30^\circ]$, and *Vertical Cell Count 5*. Is given in (fig. 6.8). The UAS is at *Back-side of Figure* (initial state is at all *Trajectory Origins*). The *black dashed line* marks *Avoidance Grid space boundary*. Each trajectory has its own color and ends at *Front-side of Avoidance Grid Boundary*.

Full separation mode is given in (fig. 6.8a). *Horizontal-Vertical separation mode*, used in original *ACAS-Xu* testing [25], given in (fig. 6.8b). *Horizontal separation mode* given in (fig. 6.8c) is usually used by planes. *Vertical separation mode* given in (fig. 6.8d) is usually used by copters.

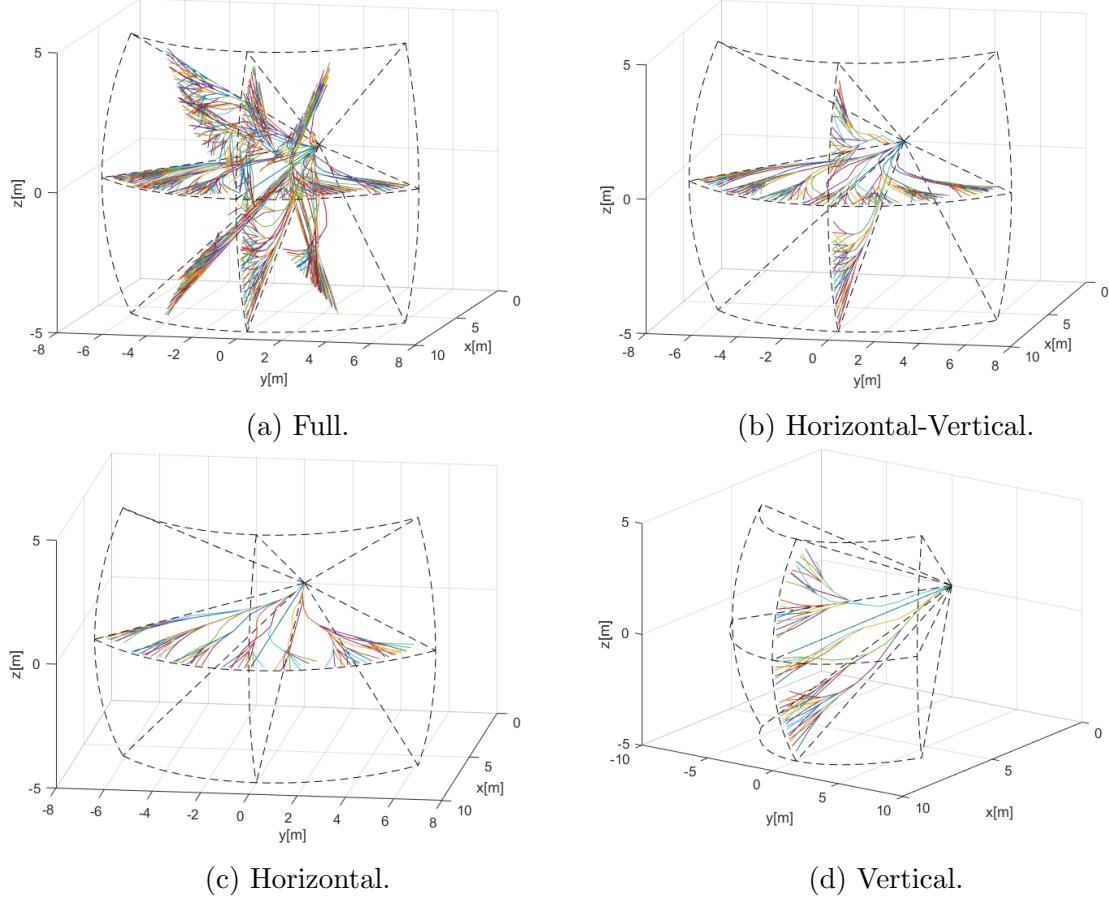


Figure 6.8: ACAS-X imitation *reach set* approximation for various *separation modes*.

Pros and Cons: It can be seen from examples (fig. 6.8) that *ACAS-like Reach Set Approximation Method* (alg. 6.4) generates a full reach set for 2D plane located in 3D space.

The *Reach Set* contains trajectories with *high coverage ratio* and *high smoothness rating* for selected 2D separation plane. Overall performance compared to full 3D reach sets (sec. 6.4.4, 6.4.5 6.4.7) is poor.

The *node* and *trajectory* count boundary was not implemented. It is common knowledge that *2D* avoidance sets do not require scaling [25]. Otherwise, trajectory footprint mechanism like in *Turn-Minimizing Reach Set Approximation* (alg. 6.3) can be introduced.

This reach set implements *Planar-Separation* as a native feature, it can be used for both *navigation* and *avoidance* tasks in *Controlled Airspace*. For *Non-controlled Airspace*, there are far more superior *Combined Reach Set* (sec. 6.4.7).

6.4.7 Combined Reach Set Approximation - Tree Merge

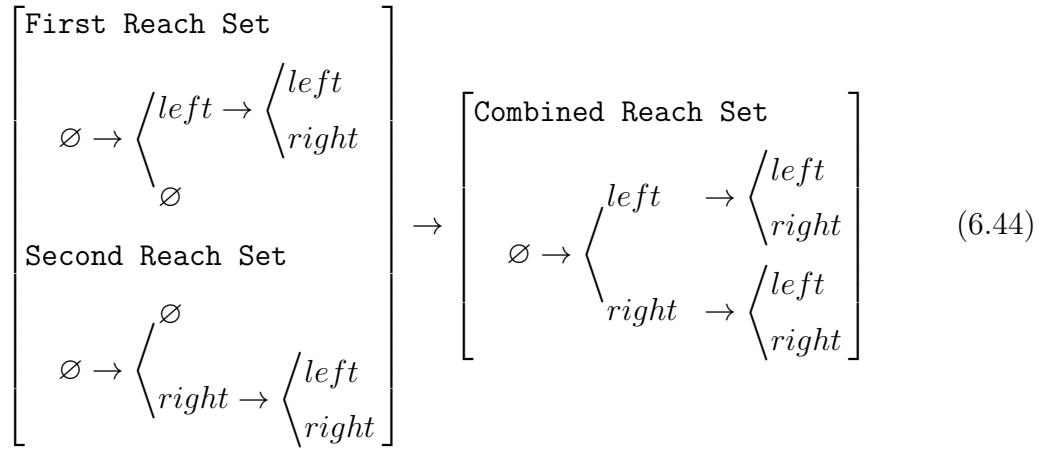
Motivation: Turn-Minimizing Reach Set Approximation (sec. 6.4.5) is *efficient* for *Navigation in Controlled Airspace*. Coverage-Maximizing Reach Set Approximation (sec. 6.4.4) is good for *Emergency avoidance*. The need for the differentiation between *Navigation* and *Emergency Avoidance* mode is necessary for *Controlled Airspace*, but not

for *Non-controlled Airspace*. The combination of *Turning-Minimizing* and *Coverage-Maximizing* reach set approximations is an obvious solution.

Automatic mode switch can be provided by a combination of *Navigation Reach Set* and *Avoidance Reach Set* with an elevated cost function. Overall having a method to merge multiple trees would be beneficial.

Background: If two *Reach Set Approximation* were calculated for the same *Avoidance Grid* and *Initial State*, using same *Movement Automaton* and *UAS model* are possible to merge.

The *Reach Set Approximation* is a tree with *Root Node* in *initial state* with movement buffer = \emptyset . The *movement buffer* in each node can be used as *route trace* during the merging procedure. The example two reach set merge can be given as follow, where only the *latest* applied movement is taken into account.



First Reach Set contains two trajectories given by buffers $\{left, left\}$ and $\{left, right\}$. *Second Reach Set* contains two trajectories given by buffers $\{right, left\}$ and $\{right, right\}$. The *Combined Reach Set* contains all four trajectories.

Note. The combined tree [154] does not need to have combined amount of original *Reach Sets* trajectories. There can be *Duplicity* which means that any bounded property like *Cost* must be *calculated* again.

Combined Reach Set Calculation Function (alg. 6.5) is implemented as function *NodecombinedReachSet(...)* which takes root Node with *initial State*, *Avoidance Grid* and respective parameters for each calculation method. *turn-minimizing spread* for *Turn-Minimizing Reach set calculation* and *coverage spread*, *Footprint Length* for *Coverage-Maximizing Reach Set Approximation*.

Separate Reach Sets are calculated using *Wave-front propagation* (alg. 6.1) using respective *Constrained Expansion* functions for *Turn-Minimizing* (alg. 6.3) and *Coverage-Maximizing* (alg. 6.2) reach sets.

Combined Reach Set is created using *Node mergeTree(...)* function, because different cost function or *Bounded Parameters Calculation* may be applied on *Original Reach Sets*.

Cost for each node needs to be recalculated due to original reach sets disparity. Function *combined.applyCostFunction()* will recalculate the new cost for each node.

The Goal is to have a penalization for *non-turn-minimizing behavior*, implementation of *Automatic Mode Switch* can be done like follows:

1. Calculate Normal Cost for Node $\text{Cost}(\text{Node})$ for the associated trajectory: $\text{Cost}(\text{Node.Trajectory})$.
2. Calculate Penalization for additional maneuvering, calculate Smoothness Rating for Trajectory (def. 19) in the interval $[0, 1]$, introduce penalization with base 100%.

The final $\text{Cost}(\text{Node})$ function is applied to each *Combined Reach Set Node* and look like follows:

$$\begin{aligned} \text{Cost}(\text{Node}) = & \text{Cost}(\text{Node.Trajectory}) \times \dots \\ & \dots \times (1 + (1 - \text{SmoothnessRate}(\text{Node.Trajectory}))) \end{aligned} \quad (6.45)$$

Tree Merge Function *mergeTree(…)* implements *Outer Join* operation on two trees. Example was given in (eq. 6.44). Function is applied on *root Node* iterating over *Movements in Movement Set*, because *Movement is pivot*.

Algorithm 6.5: Reach Set Merge Function and Combined Reach Set calculation

```

# Tree merge function;

Node mergeTree(Node firstNode, Node secondNode)
    # Try to copy reference node or return null;
    Node referenceNode = (firstNode?:(secondNode?: return null));
    Node merged = new Node(referenceNode);
    merged.leafs= [];

    # Try to fetch movement nodes if exist in any sub tree;
    for movement ∈ Movements do
        firstLeaf = firstNode.getLeafFor(movement);
        secondLeaf = secondNode.getLeafFor(movement);
        newLeaf = mergeTree(firstLeaf,secondLeaf);
        if newLeaf ~=: null then
            | merged.leafs.append(newLeaf);
        end
    end
    return merged

# Combined Reach Set calculation function;

Node combinedReachSet(Node root, AvoidanceGrid grid, int+ coverageSpread,
int+ turnSpread, int+ footprintLength)
    Node cmrsa = chaoticReachSet(root,grid, footprintLength,coverageSpread);
    Node tmsra = harmonicReachSet(root,grid, turnSpread);
    Node combined = mergeTree(cmrsa,tmsra);
    combined.applyCostFunction();
    return combined

```

Example: for *Avoidance Grid* with *Distance 10 m*, *Layer count 10*, *Horizontal range* $[-45^\circ, +45^\circ]$, *Horizontal Cell Count 7*, *Vertical range* $[-30^\circ, +30^\circ]$, and *Vertical Cell Count 5*. Is given in (fig. 6.9). The UAS is at *Back-side* of *Figure* (initial state is at all *Trajectory Origins*). The *black dashed line* marks *Avoidance Grid space boundary*. Each trajectory has its own color and ends at *Front-side* of *Avoidance Grid Boundary*. The *Coverage-Maximizing Spread* was set to 8, *Footprint Length* to 3 and *Turn-Minimizing Spread* to 1.

Note. Notice there are typical trajectories from both *Turn-Minimizing* (fig. 6.7) and *Coverage-Maximizing* (fig. 6.6) *Reach Set Approximations*.

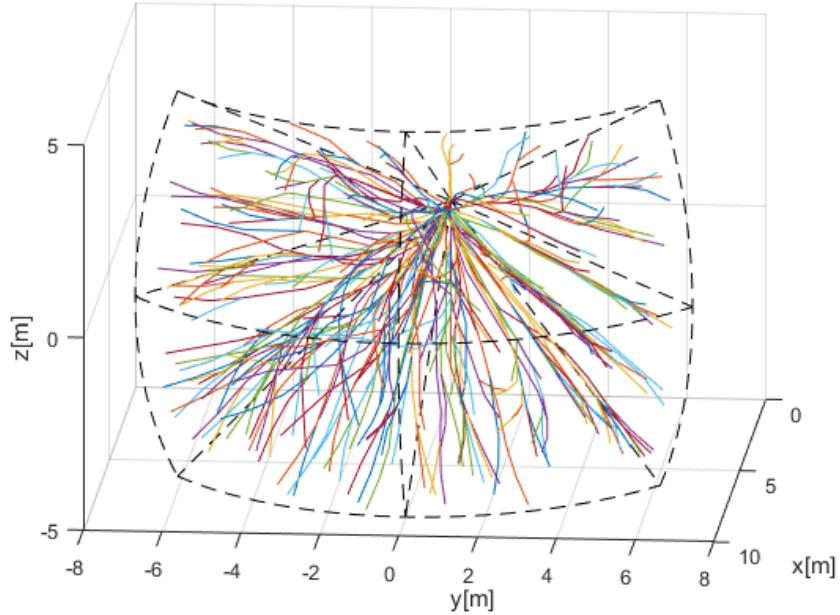


Figure 6.9: *Combined reach set approximation.*

Pros and Cons: It can be seen from example (fig. 6.9) that *Combined Reach Set Approximation* (alg. 6.5) contains both types of maneuvers. *Cheaper turn-minimizing* for navigation and *More Expensive Coverage-Maximizing* for *Emergency Avoidance*. The upper limit of trajectories is given as follow:

$$\begin{aligned} \text{countTrajectories}(\text{ReachSet}) &\leq \text{countTrajectories}(CM - RSA) \\ &+ \text{countTrajectories}(TM - RSA) \end{aligned} \quad (6.46)$$

The *upper limit of nodes* is given as follow:

$$\text{countNodes}(\text{ReachSet}) \leq \text{countNodes}(CM - RSA) + \text{countNodes}(TM - RSA) \quad (6.47)$$

Turn-Minimizing Reach Set is ideal for *Non-controlled Airspace* missions because it contains *Automatic Mode Switch* between *Navigation* and *Emergency Avoidance*.

6.5 Situation Representation in the Avoidance Grid

Summary: There is a need to have a safety assessment of the operational space in the form of the Avoidance Grid. Each type of threat coming from different sources (sensors, maps) like obstacles, intruders, and constraints is handled separately. The data fusion procedure provides a unified representation of sourced threats.

This section gives an overview how different types of threat are projected into *avoidance grid*:

1. *Obstacles* (sec. 6.5.1) - how static obstacles are represented, how to map data are processed and represented, how the concept of visibility impact certainty of an obstacle in space.
2. *Intruders* (sec. 6.5.2) - how intruders are projected into avoidance grid, how great is probability to encounter a specific intruder in given space and time.
3. *Constraints* (sec. 6.5.3) - how are constraints like geo-fencing or weather represented, how is their impact on space calculated.
4. *Data Fusion* (sec. 6.5.4) - how is the final threat in cell calculated, how this threat impacts the safety of passing trajectories, how we know which cells in the grid are safe.

6.5.1 Obstacles

Summary: There is a need to asses filtered LiDAR readings into detected obstacle rating associated to each cell. There are known obstacles in the form of a map which are verified taking into the account visibility constraints.

Introduction: The *static obstacles* were used in the original concept [13], the *Avoidance Grid* and *Movement Automaton* were repurposed to enable *finite time deterministic* avoidance. A *Constraint-based path search* and *obstacle modeling* is summarized in [155].

This section is handling basic problems of *static obstacle* detection and its focused on following real-world fixed position threats:

1. *Static Obstacles* - detected by LiDAR sensor or fused from *Obstacle Map* information source.
2. *Geo-fencing Areas* - defined by offline/online information source as permanent flight restriction zones. There is usually no physical obstacle. Space is considered as a *hard/soft constraint*.
3. *Long-term bad weather Areas* - the *weather* is often changing (hour period), there are *weather events* which last for *hours or days*.

Changing Scanning Density of LiDAR: A LiDAR sensor is scanning in conic section given by $distanceRange$, $horizontalRange$, $verticalRange$, where distance range is in the interval $[0, maxDistance]$, horizontal offset range is in $[-\pi, \pi]$, and vertical offset range is in $[\varphi_s, \varphi_e]$.

Let say that $d_{horizontal}^\circ$, $d_{vertical}^\circ$ is unitary angle offset in which one LiDAR send and return is executed. That means the *LiDAR* ray is sent every $d_{horizontal}^\circ$, $d_{vertical}^\circ$ offset movement. The *LiDAR* ray density is decreasing with *distance offset*. The same amount of *LiDAR* rays passes through $cell_{i,j,k}$ in Avoidance Grid.

A surface of the area given by some distance d and unitary offsets $d_{horizontal}^\circ$, $d_{vertical}^\circ$ is changing with *distance*. The minimal triggering area of the object surface is not changing. This fact has an impact on the count of the hits on the object surface.

The example is given in (fig. 6.10) where we have two identical objects (red circle) in distances 5 and 10 meters. The closer object consumes 5 LiDAR beam hits, and the farther object consumes only 3 LiDAR beam hits. The probability of obstacle encounter is remaining the same for the closer and farther object. The *detected obstacle rate* assessment should return the same detected obstacle collision rate for objects with the same scanned surface (with different LiDAR ray hit count).

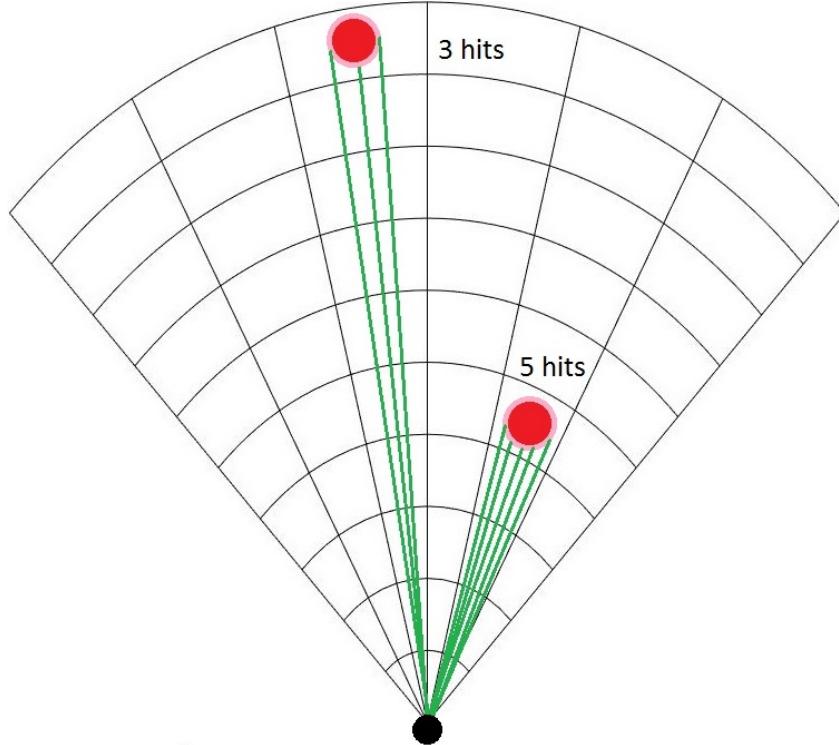


Figure 6.10: Different count of LiDAR hits with different distance from UAS.

Map and Detected Obstacles Fusion: The concept of *offline/online obstacle map* is mandatory in modern obstacle avoidance systems and increases the safety of navigation/avoidance path. The *older* concept was considering only LiDAR reading or *real-time sensor readings* in general [13].

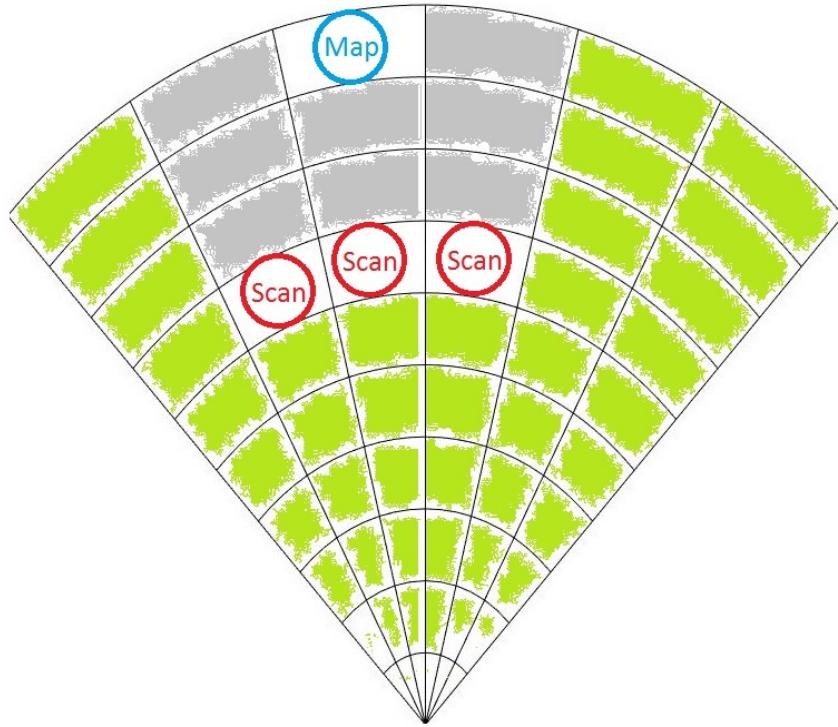


Figure 6.11: Overshadowed map obstacle by detected obstacles.

The fusion of real-time sensor readings and obstacle map (prior knowledge) is required. Data fusion of these two sources is strongly depending on visibility property because there are three basic scenarios:

1. *Dual detection* - the obstacle is marked on the map and detected by the sensory system at some point of the time (older concept works).
2. *Hindered vision* - the detected obstacles are hindering vision to map obstacle, therefore, map obstacle uncertainty arises (older concept fails).
3. *False-positive map* - map obstacle occupied space is visible by the sensory system, but negative detection is returned. Therefore the map is giving *false-positive* information.

The second case is given in fig. 6.11, where map obstacle (blue circle) is overshadowed by three scanned obstacles (red circle). The visible space is denoted by green fill; the invisible space is denoted by gray fill.

Detected Obstacles The *visibility* inside avoidance grid and *obstacle* probability are interconnected for most ranging sensors (ex. LiDAR). The goal of this section is to introduce *visibility hindrance* concept which includes space uncertainty assessment and detected obstacle processing.

Detected Obstacle Rating: The *detected obstacle rating* defines UAS chances to encounter detected obstacle in avoidance grid $cell_{i,j,k}$. Final *detected obstacle rating* is

merged information (eq. 6.86). The *sensor field* can contain *multiple static obstacle sensors*.

Detected Obstacle Rate for LiDAR: Let us have only one sensor set as homogeneous two-axis rotary LiDAR. For one $cell_{i,j,k}$ there exists a set of passing LiDAR beams:

$$lidarRays(cell_{i,j,k}) = \left\{ \begin{bmatrix} horizontal^\circ \in horizontalOffsets, \\ vertical^\circ \in verticalOffsets \end{bmatrix} \in \mathbb{R}^2 : \begin{array}{l} horizontal^\circ \in cell_{i,j,k}.horizontalRange, \\ vertical^\circ \in cell_{i,j,k}.verticalRange \end{array} \right\} \quad (6.48)$$

The horizontal and vertical offset of LiDAR ray is homogeneous. Meaning the horizontal/vertical distances between each two neighboring LiDAR beams are equal.

The set $lidarRays(cell_{i,j,k})$ (eq. 6.48) is finite countable and nonempty for any $c_{i,j,k}$, otherwise it will contradict the definition of avoidance grid (def. 12).

The hit function $lidarScan()$ returns a distance of single beam return for beam with dislocation $[horizontal^\circ, vertical^\circ] \in lidarRays(cell_{i,j,k})$ angle offsets. The set of LiDAR hits (eq. 6.49) in cell $cell_{i,j,k}$ is defined like follow:

$$lidarHits(cell_{i,j,k}) = \left\{ \begin{bmatrix} distance = lidarScan(), \\ horizontal^\circ \in horizontalOffsets, \\ vertical^\circ \in verticalOffsets \end{bmatrix} \in \mathbb{R}^2 : \begin{array}{l} distance \in cell_{i,j,k}.distanceRange, \\ horizontal^\circ \in cell_{i,j,k}.horizontalRange, \\ vertical^\circ \in cell_{i,j,k}.verticalRange \end{array} \right\} \quad (6.49)$$

The *naive* obstacle rate in case of LiDAR sensor defined as a ratio between landed hits and possible hits:

$$obstacle_{cell_{i,j,k}}^{LiDAR} = \frac{lidarHits(cell_{i,j,k})}{lidarRays(cell_{i,j,k})} \quad (6.50)$$

Note. The *naive obstacle rate* (eq. 6.50) ignores that *LiDAR rays* are getting farther apart from each other. The *cell surface* is increasing with cell distance from *UAS*.

The hindrance (eq. 6.51) rate is naturally defined as a supplement to naive obstacle rate. This definition is sufficient because it is reflecting the *remaining sensing capability* of LiDAR.

$$hindrance_{cell_{i,j,k}}^{LiDAR} = 1 - \frac{lidarHits(cell_{i,j,k})}{lidarRays(cell_{i,j,k})} \quad (6.51)$$

Cell Density Function: Let us start with the differential form of the cell surface (eq. A.22). The target object has several hits in *Avoidance Grid*. Target $cell_{i,j,k}$ has following properties which are used in surface calculation:

1. *Horizontal span* - defines the range of horizontal scanner partition.
2. *Vertical span* - defines the range of vertical scanner partition.

By rewriting (eq. A.22) and using horizontal range parameter and inverted vertical range parameter following surface integral is obtained (eq. 6.52).

$$\begin{aligned} \text{Area}(cell_{i,j,k}) = & \\ & \int_{horizontal^{\circ}_{start}}^{horizontal^{\circ}_{end}} \int_{vertical^{\circ}_{end}}^{vertical^{\circ}_{start}} radius^2 \cos(vertical^{\circ}) \ dvertical^{\circ} dhorizontal^{\circ} \quad (6.52) \end{aligned}$$

Note. The *radius* parameter is the *average* distance of hits landed in $cell_{i,j,k}$. This helps to reflect real *scanned surface*.

Numerically stable integration exist for boundaries *horizontal* $^{\circ}$ in $[-\pi, \pi]$, *vertical* $^{\circ}$ $\in [-\frac{\pi}{2}, \frac{\pi}{2}]$ given as follow:

$$\begin{aligned} \text{Area}(radius, horizontalRange, vertical^{\circ}_{start}, vertical^{\circ}_{end}) &= \dots \\ &= \begin{cases} vertical^{\circ}_{start} < 0, vertical^{\circ}_{end} \leq 0 : \\ \quad radius^2(\sin |vertical^{\circ}_{start}| - \sin |vertical^{\circ}_{end}|) \times horizontalRange \\ vertical^{\circ}_{start} < 0, vertical^{\circ}_{end} > 0 : \\ \quad r^2(\sin |vertical^{\circ}_{start}| + \sin |vertical^{\circ}_{end}|) \times horizontalRange \\ vertical^{\circ}_{start} \geq 0, vertical^{\circ}_{end} < 0 : \\ \quad r^2(\sin vertical^{\circ}_{end} - \sin vertical^{\circ}_{start}) \times horizontalRange \end{cases} \quad (6.53) \end{aligned}$$

An intersection surface for the cell is defined in (eq. 6.53). The Area covered by LiDAR hits (eq. 6.54) is defined as LiDAR hit rate (hits to passing rays ratio) multiplied by *Average* cell intersection surface (eq. 6.53).

$$lidarHitArea(cell_{i,j,k}) = \frac{lidarHits(cell_{i,j,k})}{lidarRays(cell_{i,j,k})} \times \text{Area} \left(\begin{matrix} radius, horizontalRange, \\ vertical^{\circ}_{start}, vertical^{\circ}_{end} \end{matrix} \right) \quad (6.54)$$

There is user-defined parameter for *LiDAR threshold area*, which represents a minimal considerable surface area for the obstacle to be a threat. The *detected obstacle rate* considering surface is defined in (eq. 6.55), and it removes the bias of naive approach (eq.6.50).

$$obstacle(LiDAR, cell_{i,j,k}) = \min \left\{ \frac{lidarHitArea(cell_{i,j,k})}{UAS.lidarThresholdArea}, 1 \right\} \quad (6.55)$$

Visibility Rate for LiDAR: For each $cell_{i,j,k}$ and each sensor in sensor field there exist hindrance rate, which defines how much vision is clouded in a single cell. Example of hindrance calculation for LiDAR has been given by (eq. 6.51). Let us consider cell row $cellRow(j_{fix}, k_{fix})$ with fixed horizontal index j_{fix} and vertical index k_{fix} is given as a series of cells (eq. 6.56).

$$cellRow(j_{fix}, k_{fix}) = \left\{ cell_{i,j,k} \in AvoidanceGrid : \begin{array}{l} i \in \{1, \dots, layersCount\}, \\ j = j_{fix}, k = k_{fix} \end{array} \right\} \quad (6.56)$$

For each $cell_{i,j,k}$ there exists a function which calculates final visibility hindrance rate. Then for ordered cell row:

$$cellRow(j_{fix}, k_{fix}) = \{cell_{1,j_{fix},k_{fix}}, cell_{2,j_{fix},k_{fix}}, \dots, cell_{layersCount,j_{fix},k_{fix}}\}$$

and for one selected $cell_{i,j,k}$ the visibility rate is naturally defined as a supplement to hindrance from previous cells. The visibility is defined in (eq. 6.57).

$$\begin{aligned} visibility(cell_{i_c,j_c,k_c}) &= \dots \\ \dots &= 1 - \sum_{\substack{index < i_c \\ index \in \mathbb{N}^+}} hindrance(cell_{a,j_c,k_c} : cell_{a,j_c,k_c} \in cellRow(j_c, k_c)) \end{aligned} \quad (6.57)$$

Example: Let be $cell_{4,j_{fix},k_{fix}}$ is selected for visibility rate assessment, then $cell_{1,j_{fix},k_{fix}}$, $cell_{2,j_{fix},k_{fix}}$, and $cell_{3,j_{fix},k_{fix}}$, are used as a base of cumulative hindrance rate.

The cumulative hindrance rate for any $cellRow(j_{fix}, k_{fix})$ is bounded:

$$0 \leq \sum_{cell \in cellRow(j_{fix}, k_{fix})} visibility(cell) \leq 1 \quad (6.58)$$

Note. A cumulative hindrance rate does not always reach 1 in case of LiDAR sensor, because some rays may pass or hit after leaving avoidance grid range.

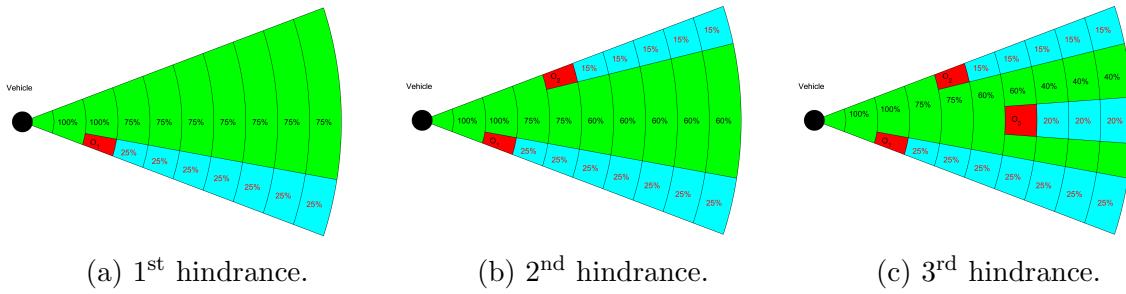


Figure 6.12: Obstacle hindrance impact on visibility in *Avoidance Grid Slice*.

For one cell row $cellRow(j_{fix}, k_{fix})$, where count of layers is equal to 10, and layers have equal spacing. There is LiDAR sensor

During consequent LiDAR scans $s(t_0)$, $s(t_1)$, $s(t_2)$, and $s(t_3)$ the obstacle sets $\mathcal{O}_1(t_1) = \{o_1\}$, $\mathcal{O}_2(t_2) = \{o_1, o_2\}$, and $\mathcal{O}_3(t_3) = \{o_1, o_2, o_3\}$ are discovered. Assigned hindrance rates are like follow:

1. *Time t₀* - there is no obstacle nor hindrance, all cells are fully visible.
2. *Time t₁* (fig. 6.12a) - $\mathcal{O}_1(t_1) = \{o_1\}$ was detected, the hindrance rate for $cell_{3,j_{fix},k_{fix}}$ is equal to 0.25. The visibility rate in cells $cells_{4-10,j_{fix},k_{fix}}$ is 0.75.
3. *Time t₂* (fig. 6.12b) - $\mathcal{O}_2(t_2) = \{o_1, o_2\}$ was detected, the additional hindrance rate for $cell_{5,j_{fix},k_{fix}}$ is 0.15. The visibility rate in $cells_{6-10,j_{fix},k_{fix}}$ is lowered by additional 0.15 and its set to 0.60 now.
4. *Time t₃* (fig. 6.12c) - $\mathcal{O}_3(t_3) = \{o_1, o_2, o_3\}$ was detected the additional hindrance rate for $cell_{7,j_{fix},k_{fix}}$ is 0.20. The visibility rate in $cells_{8-10,j_{fix},k_{fix}}$ is lowered by additional 0.20 and its set to 0.40 now.

Map Obstacles: Use *stored LiDAR readings* from the previous mission to build a compact obstacle map [156]. Then use *this map* as an additional information source.

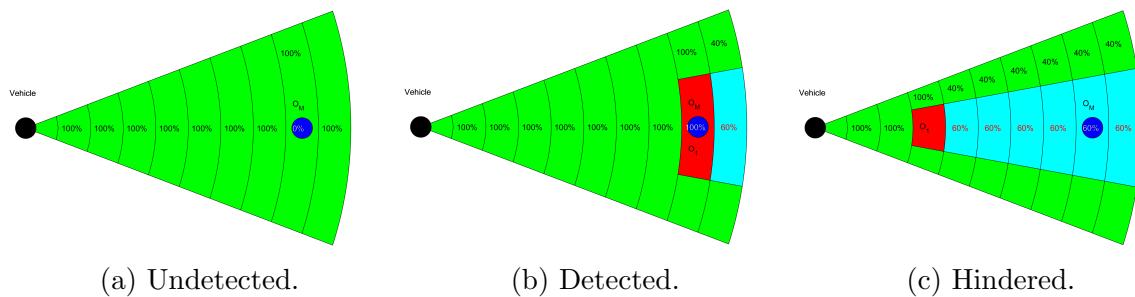


Figure 6.13: Map obstacle states after *Data fusion*.

Concept: A *map obstacle* state has very simple logic, there are three possible cases:

1. *Undetected* - Map obstacle O_M is charted on the map (fig. 6.13a) but is undetected by any sensor in the sensor field; therefore the probability of map obstacle occurrence is equal to 0.

2. *Detected* Map obstacle O_M is charted on the map and detected by any sensor in sensor field (fig. 6.13b). The map obstacle rate is equal to detected obstacle rate, usually its equal to 1.
3. *Hindered* Map obstacle O_M is hindered behind other detected obstacle O_1 (fig. 6.13c). The detected obstacle O_1 is in $cell_{i,j,k}$ and is reducing visibility in follow up $cellRow_{i_f > i,j,k}$ by 60 percent.

Implementation: The formulation of final map obstacle rate $map(cell_{i,j,k})$ was outlined in previous examples. These examples are showing the *desired behavior* and its solved by *data fusion* (sec. 6.5.4).

First, let us start with obstacle map definition. The obstacle map (eq. 6.59) defines a map obstacle set of information vectors with a position in the global coordinate frame, orientation bounded to a global coordinate reference frame, safety margin and additional parameters.

$$obstacleMap = \left\{ \begin{bmatrix} position, \\ orientation, \\ safetyMargin, \\ parameters \end{bmatrix} : \begin{array}{l} position \in \mathbb{R}^3(GCF), \\ orientation \in \mathbb{R}^3(GCF), \\ safetyMargin \in \mathbb{R}^+(m), \\ parameters \in \{\dots\} \end{array} \right\} \quad (6.59)$$

The *Map Obstacle* concept is taken from my *master student work* [156], implementing a *compact representation* of point-cloud obstacle map. Te example of *cuboid obstacles* with the *safe zone* is given in (fig. 6.14).

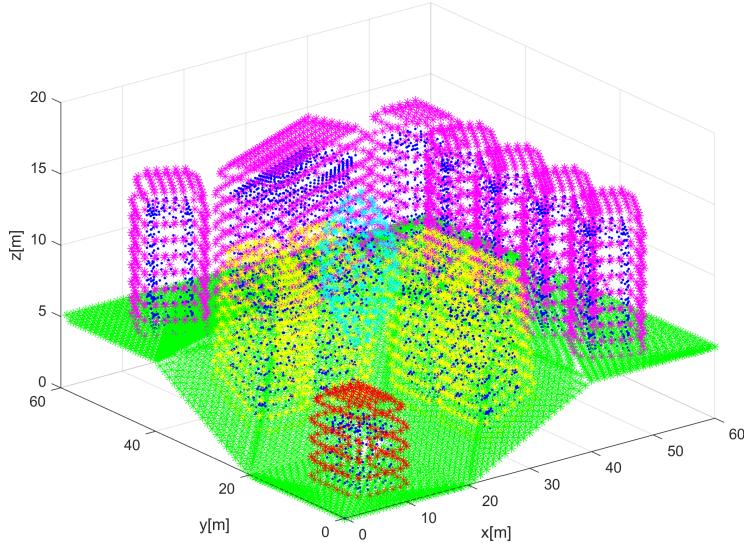


Figure 6.14: Example of Extracted Map Obstacle [156].

The space covered by any obstacle is non-empty by definition. There are following types of map charted obstacles which are implemented in the framework:

1. *Ball obstacle parameters* = \emptyset - simple ball with center at the *position*, with an offset safety margin.
2. *Line obstacle parameters* = $[length]$ - simple line bounded by length $\in]0, \infty[$ with center at the *position* and given orientation to the main axis in the global coordinate frame, with safety margin < 0 .
3. *Plane obstacle parameters* = $[length, width]$ - bounded rectangle plane partition defined by length $\in]0, \infty[,$ and width $w \in]0, \infty[$ with center at \vec{p} and given orientation \vec{o} concerning the main axis in the global coordinate frame, with a safety margin.
4. *Cuboid obstacle parameters* = $[length, width, depth]$ - bounded cuboid space partition defined by length $\in]0, \infty[,$ width $\in]0, \infty[,$ and depth $d \in]0, \infty[$ with center at the *position* and rotated in orientation concerning the main axis in the global coordinate frame, with a safety margin.

The *map obstacles* are stored in the clustered database. The *selection criterion* is given in (eq. 6.60).

$$\text{avoidanceGrid.radius} \geq \text{distance}(\text{UAS.position}, \text{mapObstacle}) - \text{totalMargin} \quad (6.60)$$

The *total margin* is a combination of *safety margin* and *body margin* (in case of a line/plane/cuboid obstacle). The *selection* was implemented as standard cluster select, selecting 26 surrounding clusters around UAS + own UAS cluster.

The *compact obstacle representation* is transformed into *homogeneous point-cloud representations*:

1. *Body Point-cloud* - representing obstacle body approximation by geometrical shape (eq. 6.61). This point cloud is considered as hard constraints.

$$\text{bodyPointCloud} = \{\text{point} \in \mathbb{R}^3(\text{GCF}) : \text{point} \in \text{mapObstacleBody}\} \quad (6.61)$$

2. *Safety Margin Point Cloud* - representing safety coating around mapped obstacle body approximation (eq. 6.62). This point cloud is considered as soft constraint.

$$\text{marginPointCloud} = \{\text{point} \in \mathbb{R}^3(\text{GCF}) : \text{point} \in \text{mapSafetyMargin}\} \quad (6.62)$$

Note. The *safety margin point cloud* is hollow in relationship to a *body point cloud*, therefore:

$$\text{bodyPointCloud} \cap \text{marginPointCloud} = \emptyset$$

The *map obstacle* discretization to point cloud leads to the problem how to calculate *impact rate*. The *theoretical impact rate* for an *obstacle* is given as:

$$\text{impactRate} = \frac{\text{volume}(\text{mapObstacle} \cap \text{cell}_{i,j,k})}{\text{volume}(\text{cell}_{i,j,k})} \in [0, 1]$$

The *map obstacle related point clouds* (eq. 6.61, 6.62) are homogeneous [156]. That means *each point* in point clouds covers a similar portion of object volume. There is *threshold volume* (eq. 6.63) which represents minimal object volume to be considered as an *obstacle*.

$$0 < \text{thresholdVolume} \leq \frac{\text{volume}(\text{pointCloud})}{|\text{pointCloud}|} \quad (6.63)$$

The *impact rate* of one point when intersecting a $\text{cell}_{i,j,k}$ is given as count of *threshold obstacle bodies* in *point cloud covered mass* multiplied by inverted point count (eq. 6.64).

$$\text{point.rate} = \frac{\text{pointCloudVolume}}{\text{thresholdVolume}} \times \frac{1}{|\text{pointCloud}|} \quad (6.64)$$

The *intersection set* between *point cloud* and $\text{cell}_{i,j,k}$ is defined in (eq. 6.64). The *cell intersection with points* is defined in (eq. 6.15).

$$\begin{aligned} \text{intersection}(\text{map}, \text{cell}_{i,j,k}) = \dots \\ \dots \{ \text{points} \in \mathbb{R}^3 : (\text{point} \rightarrow \text{AvoidanceGridFrame}) \in \text{cell}_{i,j,k} \} \end{aligned} \quad (6.65)$$

The *map obstacle rating* for $\text{cell}_{i,j,k}$ and obstacle for our *information source* is defined in (eq. 6.66).

$$\text{map}(\text{cell}_{i,j,k}, \text{obstacle}) = \max \left\{ \sum_{\forall \text{point} \in \text{intersection}(\text{map}, \text{cell}_{i,j,k})} \text{point.rate}, 1 \right\} \quad (6.66)$$

The *map obstacle rating* for $\text{cell}_{i,j,k}$, and *our information source* is given as maximum of all possible cumulative ratings from each obstacle in *active map obstacles* set (eq. 6.67).

$$\text{map}(\text{cell}_{i,j,k}) = \max \{ \text{map}(\text{cell}_{i,j,k}, \text{obstacle}) : \forall \text{obstacle} \in \text{ActiveMapObstacles} \} \quad (6.67)$$

Note. The *body point cloud* (eq. 6.61) never intersect, because they are created for inclusive obstacles. The *safety margin point cloud* (eq. 6.62) can intersects, because they represent protection zones around physical obstacles. Therefore the *maximum obstacle rating* (eq. 6.67) needs to be selected.

6.5.2 Intruders

Summary: The intruder information coming from ADS-B needs to be assessed in relationship to the Avoidance Grid. The final assessment consists of time encounter and space encounter ratings. The space encounter rating describes the probability of UAS meeting intruder in the same space. The time encounter rating is reflecting simulations time in the same cell.

Intruder behavior: *Adversarial behavior* of moving obstacle is trying to destroy avoiding our UAS. The *Intruder UAS* [157] is not trying to hurt our *UAS* actively. The *Adversarial behaviour* is neglected in this work. The non-cooperative avoidance is assumed, it can be relaxed to *cooperative avoidance* in *UTM controlled airspace*.

Intruder information: The *observable intruder information set* for any kind of intruder, obtained through the sensor/C2 line, is following:

1. *Position* - position of an intruder in the *local* or *global* coordinate frame, which can be transformed into *avoidance grid coordinate frame*.
2. *Heading and Velocity* - intruder heading and linear velocity in avoidance grid coordinate frame.
3. *Horizontal/Vertical Maneuver Uncertainty Spreads* - how much can an *intruder* deviate from the *original linear path* in a *horizontal/vertical* plane in *Global coordinate Frame*.

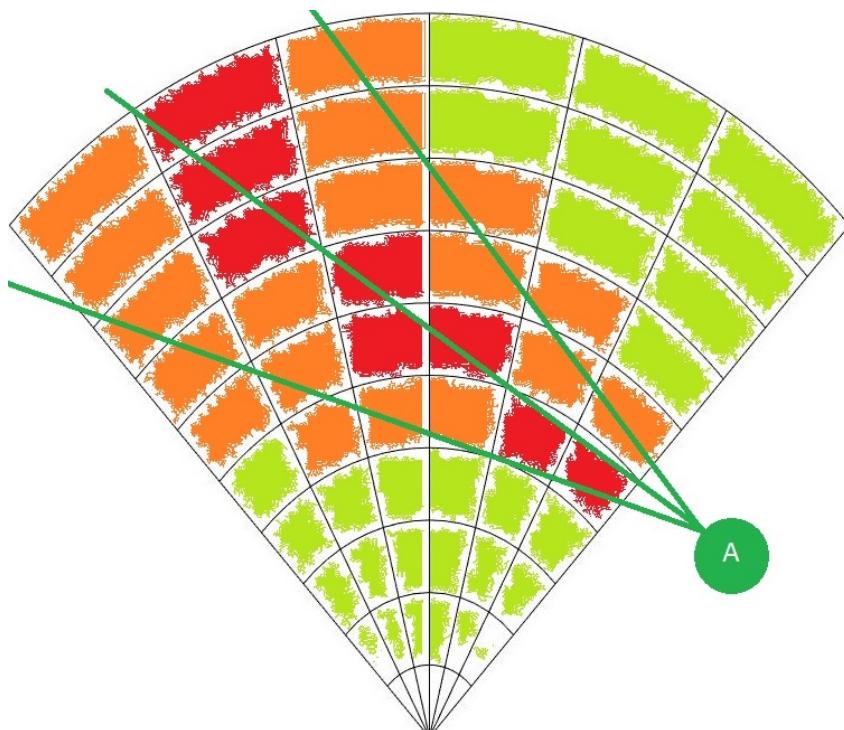


Figure 6.15: Intruder UAS intersection rate along the expected trajectory.

Example of Intruder Intersection: Let us neglect the *time-impact* aspect on the *intersection*. The *intruder* (black "I" circle) is intersecting one *avoidance grid horizontal slice* (fig. 6.15). The intruder is moving along linear path approximation based on velocity (middle green line). The *Horizontal Maneuver Uncertainty spread* is in *green line boundary area intruder intersection rating* is denoted as green-orange-red cell fill reflecting intersection severity: red is a high rate of the intersection, orange is the medium rate of the intersection and green is a low rate of the intersection.

Moving Threats: The *UAS* can encounter following threats during the *mission execution*:

1. *Non-cooperative Intruders* - the intruders who does not implement any approach to ensure mutual avoidance efficiency.
2. *Cooperative Intruders* - the intruders whom actively communicate or follow common agreed behavior pattern (ex. Rules of the Air).
3. *Moving Constraints* - the constrained portion of *free space* which is shifting its boundary over time (ex. Short term bad weather).

Note. Our approach considers only *UAS* intruders because *Data Fusion* considers data received through *ADS-B* messages. The *Intruders* extracted from *LiDAR* scan were not considered (ex. birds). The proposed *intruder intersection models* are reusable for other *intruder sources*.

Approach Overview: The *Avoidance Grid* (def. 12) is adapted to the *LiDAR* sensor. The *Euclidean grid intersections* are fairly simple. The *polar coordinates grid* is not. The need to keep *polar coordinates grid* is prevalent, because of fast *LiDAR* reading assessment. There are following commonly known methods to address this issue:

1. *Point-cloud Intersections* - the *threat impact area* is discredited into a sufficiently thick point cloud. This point-cloud have *point impact rate* and *intersection time* assigned to each point. The *point-cloud* is projected to *Avoidance Grid*. If the *impact point* hits $cell_{i,j,k}$ the cell's impact rate is increased by the amount of *point impact rate*. The final *threat impact rate* in $cell_{i,j,k}$ is given when *all* points from point cloud are consumed. Close point problem [158] was solved by the application of method [159].
2. *Polygon Intersections* - the *threat impact area* is modeled as a polygon, each $cell_{i,j,k}$ in *Avoidance Grid* is considered as a *polygon*. There is a possibility to calculate cell space geometrical inclusive intersection. The *impact rate* is then given as rate between *intersection volume* and $cell_{i,j,k}$ volume. The algorithm used for intersection selected based on:[160] the selected algorithm *Shamos-Hoey* [161].

Note. The *Intruder Intersection* models are based on *analytical geometry* for *cones* and *ellipsoids* taken from [162].

Intruder Behaviour Prediction: *Intruder Intersection Models* is about space-time intersection of *intruder body* with *avoidance Grid* and *Reach Set*:

1. The *UAS* reach set defines *time boundaries* to *enter/leave* the cell in avoidance grid.
2. The *Intruder* behavioral pattern defines the *rate of space intersection* with cell bounded space in avoidance grid.

The multiplication of *space intersection rate* and *time intersection rate* will give us *intruder intersection* rate for our *UAS* and intruder.

Intruder Dynamic Model: The definition of avoidance grid enforces most of these methods to be numeric. Let us introduce the intruder dynamic model:

$$\begin{aligned} position_x(t) &= position_x(0) + velocity_x \times t \\ \text{dposition/dtime} = velocity &\quad | \quad position_y(t) = position_y(0) + velocity_y \times t \quad (6.68) \\ position_z(t) &= position_z(0) + velocity_z \times t \end{aligned}$$

Position vector in euclidean coordinates $[x, y, z]$ is transformed into *Avoidance Grid* coordinate frame. Velocity vector for $[x, y, z]$ is *estimated and not changing*. The time is in interval $[entry, leave]$, where *entry* is intruder entry time into avoidance grid and *leave* is intruder leave time from avoidance grid.

Note. If the *intruder* is considered, time of entry is marked as $intruder_{entry,k}$ where k is intruder identification, time of leave is marked as $intruder_{leave,k}$ where k is intruder identification.

Cell Entry and Leave Times $UAS_{entry}(cell_{i,j,k})$ and $UAS_{leave}(cell_{i,j,k})$ are depending on intersecting *Trajectories* and *bounded cell space* (eq. 6.15). There is *Trajectory Intersection* function from (def. 14) which evaluates *Trajectory segment* entry and leave time.

The UAS *Cell Entry* time is given as the minimum of all *passing trajectory segments* entry times (eq. 6.69) if there is no *passing trajectories* the UAS *entry time* is set to 0.

$$UAS_{entry}(cell_{i,j,k}) = \min \left\{ 0, entry(Trajectory, cell_{i,j,k}) : Trajectory \in PassingTrajectories \right\} \quad (6.69)$$

The UAS *Cell Leave* time is given as the maximum of all *passing trajectory segments* entry times (eq. 6.70), if there are no *passing trajectories* the UAS *leave time* is set to 0.

$$UAS_{leave}(cell_{i,j,k}) = \max \left\{ 0, leave(Trajectory, cell_{i,j,k}) : \begin{array}{c} \\ Trajectory \in PassingTrajectories \end{array} \right\} \quad (6.70)$$

Time Intersection Rate: The key idea is to calculate how long the *UAS* and *Intruder* spend together in same space portion ($cell_{i,j,k}$). The *Intruder* can spend some time in $cell_{i,j,k}$ bounded by an interval of *intruder* entry/leave time.

The *UAS* can spend some time, depending on the *selected trajectory* from *Reach Set*. The time spent by UAS is bounded by entry (eq. 6.69) and leave (eq. 6.70).

The intersection duration of these two intervals creates *time intersection rate* numerator, the *maximal duration* of *UAS* stay gives us *denominator*. The *time intersection rate* is formally defined in (eq. 6.71).

$$time \begin{pmatrix} UAS, \\ Intruder, \\ cell_{i,j,k} = \circ \end{pmatrix} = \frac{\left| [intruder_{entry}(\circ), intruder_{leave}(\circ)] \cap [UAS_{entry}(\circ), UAS_{leave}(\circ)] \right|}{|[UAS_{entry}(\circ), UAS_{leave}(cell_{\circ})]|} \quad (6.71)$$

Intruder Intersection Rate: The *Intruder Intersection Rate* (eq. 6.72) is calculated as the *multiplication* of *space intersection rate* (defined later) and *time intersection rate* (eq. 6.71).

$$intruder \begin{pmatrix} UAS, \\ Intruder, \\ cell_{i,j,k} \end{pmatrix} = time \begin{pmatrix} UAS, \\ Intruder, \\ cell_{i,j,k} \end{pmatrix} \times space \begin{pmatrix} UAS, \\ Intruder, \\ cell_{i,j,k} \end{pmatrix} \quad (6.72)$$

Note. If there is no information to derive *Intruder* entry/leave time for cells the *time intersection rate* is considered 1.

The *Intruder cell reach time* (eq. 6.73) is bounded to a discrete point in the intersection model [158, 159]. The intruder *entry/leave time* is calculated similarly to *UAS cell entry* (eq. 6.69)/*leave* (eq. 6.70) time.

$$pointReachTime(Intruder, point) = \frac{distance(Intruder.initialPosition, point)}{|Intruder.velocity|} \quad (6.73)$$

Space Intersection Rate: The *Space Intersection Rate* reflects the probability of *Intruder* intersection with a portion of space bounded by $cell_{i,j,k}$, to be precise with intruder trajectory or vehicle body shifted along the trajectory. The principles for *space intersection rate* calculation are the following:

1. *Line trajectory* - intruder trajectory is given by linear approximation (eq. 6.68), depending on *intruder size* the intersection with avoidance grid can be:
 - a. *Simple line* - intersection is going along the trajectory line defined by intruder model (eq.6.68).
 - b. *Volume line* - intersection is going along the trajectory line defined by intruder model (eq. 6.68), and intruder's *body radius* is considered in the intersection.
2. *Elliptic cone* - initial position is considered as the top of a cone, the main cone axis is defined by the intruder linear trajectory (eq. 6.68) $time \in [0, \infty]$. The cone width is set by the horizontal and vertical spread.

6.5.3 Constraints

Summary: There are different constraints from various sources with different impacts. There is a need for constraint impact assessment on the cells in the Avoidance Grid.

Static Constraints: The *constraints* (ex. weather, airspace) usually covers a large portion of the *operation airspace*.

Converting constraints into valued *point-cloud* is not feasible, due to the *huge amount of created points* and low *intersection rate*. The *polygon intersection* or *circular boundary of a 2D polygon* is a simple and effective solution [55, 163].

The key idea is to create *constraint barrels* around dangerous areas. Each *constraint barrel* is defined by a circle on the *horizontal plane* and the *vertical limit range*.

Representation: The *minimal representation* is based on (sec. 2.4, 2.6.2) and geofencing principle. The *horizontal-vertical separation* is ensured by *projecting boundary* as 2D polygon oh horizontal plane and *vertical boundary* (barrel height) as *altitude limit*.

The *static constraint* (eq. 6.74) is defined as a structure vector including:

1. *Position* - the center position in the global coordinates *2D horizontal plane*.
2. *Boundary* - the ordered set of boundary points forming edges in the global coordinates *2D horizontal plane*.
3. *Altitude Range* - the *barometric altitude* range $[altitude_{start}, altitude_{end}]$.
4. *Safety Margin* - the *protection zone* (soft constraint) around constraint body (hard constraints) in meters.

$$constraint = \{position, boundary, altitude_{start}, altitude_{end}, safetyMargin\} \quad (6.74)$$

Active constrain selection: The *active constraints* are constraints which are impacting *UAS active avoidance range*.

The *active constraints set* (eq. 6.75) is defined as a set of *constraints* from all *reliable Information Sources* where the *distance* between UAS and constraint body (including safety margin) is lesser than the avoidance grid range. The *horizontal altitude range* of avoidance grid must also intersect with *constraint altitude range*.

ActiveConstraints = ...

$$\dots = \left\{ \begin{array}{l} \text{constraint} \in \text{InformationSource :} \\ \quad \text{distance}(\text{constraint}, \text{UAS}) \leq \text{AvoidanceGrid.distance,} \\ \quad \text{constraint.altitudeRange} \cap \text{UAS.altitudeRange} \neq \emptyset \end{array} \right\} \quad (6.75)$$

Cell Intersection: The *importance of constraints* is on their impact on *avoidance grid cells*. The *most of the constraints* (weather, ATC) are represented as 2D convex polygons. Even the *irregularly shaped constraints* are usually split into smaller convex 2D polygons.

The idea is to represent convex polygon boundary as a sufficiently large circle to cover polygon. The Welzl algorithm to find *minimal polygon cover circle* [55] is used.

First the *set of constraint edges* (eq. 6.76) is a enclosed set of 2D edges between neighboring points defined as follow:

$$\text{edges}(\text{constraint}) = \left\{ \begin{array}{l} \text{point} \in \text{boundary,} \\ \left[\text{point}_i, \text{point}_j \right] : i \in \{1, \dots, |\text{boundary}| \}, \\ j \in \{2, \dots, |\text{boundary}|, 1\} \end{array} \right\} \quad (6.76)$$

The *constraint circle boundary* with calculated center on the 2D horizontal plane and radius (representing body margin) is defined in (eq. 6.77).

$$\text{circle}(\text{constraint}) = \left[\begin{array}{l} \text{center} = \frac{\sum \text{boundary.point}}{|\text{boundary.point}|} + \text{correction} \\ \text{radius} = \text{smallestCircle}(\text{edges}(\text{constraints})) \end{array} \right] \quad (6.77)$$

The ($\text{cell}_{i,j,k}$ and *constraint* intersection (eq. 6.78) is classification function. The *classification* is necessary, because one *constraint* induce:

1. *Body Constraint* (hard constraint) - the distance between $\text{cell}_{i,j,k}$ closest border and *circular boundary* center is in interval $[0, \text{radius}]$.
2. *Protection Zone Constraint* (soft constraint) - the distance between $\text{cell}_{i,j,k}$ closest border and *circular boundary* center is in interval $[\text{radius}, \text{radius} + \text{safetyMargin}]$.

intersection, constraint) = ...

$$\dots = \begin{cases} \text{hard} & : \begin{bmatrix} \text{distance}(\text{cell}_{i,j,k}, \text{circle}(\text{constraint})) \leq \dots \\ \dots \leq \text{circle}(\text{constraint}).\text{radius}, \\ \text{constraint}.altitudeRange \cap \text{cell}_{i,j,k}.altitudeRange \neq \emptyset, \end{bmatrix} \\ \text{soft} & : \begin{bmatrix} \text{distance}(\text{cell}_{i,j,k}, \text{circle}(\text{constraint})) > \dots \\ \dots > \text{circle}(\text{constraint}).\text{radius}, \\ \text{distance}(\text{cell}_{i,j,k}, \text{circle}(\text{constraint})) \leq \dots \\ \dots \leq \text{circle}(\text{constraint}).\text{radius} + \text{safetyMargin}, \\ \text{constraint}.altitudeRange \cap \text{cell}_{i,j,k}.altitudeRange \neq \emptyset, \end{bmatrix} \\ \text{none} & : \text{otherwise} \end{cases} \quad (6.78)$$

The *intersection impact* of constraint is handled separately for *soft* and *hard* constraints. The *avoidance* of hard constraints is *mandatory*, the *avoidance* of soft constraints is *voluntary*.

The constraints which have a *soft intersection with the cell* are added to cells impacting constraints set:

$$\text{cell}_{i,j,k}.\text{softConstraints} = \left\{ \text{constraint} \in \text{ActiveConstraints} : \begin{array}{l} \text{intersection}(\text{cell}_{i,j,k}, \text{constraint}) = \text{soft} \end{array} \right\} \quad (6.79)$$

The constraints which have a *hard intersection with the cell* are added to cells impacting constraints set:

$$\text{cell}_{i,j,k}.\text{hardConstraints} = \left\{ \text{constraint} \in \text{ActiveConstraints} : \begin{array}{l} \text{intersection}(\text{cell}_{i,j,k}, \text{constraint}) = \text{hard} \end{array} \right\} \quad (6.80)$$

Note. The final *constraint rate value* (eq. 6.89) is determined based on *mission control run* feed to *avoidance grid* (fig. 6.22) defined in 7th to the 10th step.

Moving Constraints: The basic ideas is the same as in case *static constraints* (sec. 6.5.3). Horizontal constraint and altitude constraint is outlining the constrained space. The only additional concept is moving of *constraint* on the horizontal plane in a global coordinate system.

The constraint intersection with *avoidance grid* is done in *fixed decision Time*, for cell in *fixed cell leave time* (eq. 6.70), which means concept from static obstacles can be fully reused.

Definition: The *moving constraint definition* (eq. 6.81) covers minimal data scope for moving constraint, assuming linear constraint movement.

Definition 20. Moving Constraints The original definition (eq. 6.74) is enhanced with additional parameters to support constraint moving:

1. Velocity - velocity vector on 2D horizontal plane.
2. Detection time - the time when constraint was created/detected, this is the time when center and boundary points position were valid.

$$\begin{aligned} \text{constraint} = & \{\text{position}, \text{boundary}, \dots \\ & \dots, \text{velocity}, \text{detectionTime}, \dots \\ & \dots \text{altitude}_{\text{start}}, \text{altitude}_{\text{end}}, \text{safetyMargin}\} \quad (6.81) \end{aligned}$$

Cell Intersection: The *intersection algorithm* follows (eq. 6.78), only shift of the center and boundary points is required.

First let us introduce Δtime (eq. 6.82), which represents the difference between the constraint detection time and expected cell leave time (eq. 6.70).

$$\Delta\text{time} = UAS_{\text{leave}}(\text{cell}_{i,j,k}) - \text{detectionTime} \quad (6.82)$$

The constraint boundary is shifted to:

$$\begin{aligned} \text{shiftedBoundary}(\text{constraint}) = & \{\text{newPoint} = \text{point} + \text{velocity} \times \Delta\text{time} : \dots \\ & \dots \forall \text{point} \in \text{constraint.boundary}\} \quad (6.83) \end{aligned}$$

The constraint center is shifted to:

$$\text{shiftedCenter}(\text{constraint}) = \text{constraint.center} + \text{velocity} \quad (6.84)$$

Note. The Δtime is calculated separately for each $\text{cell}_{i,j,k}$, because UAS is also moving and reaching cells in different times. The *cell leave time* can be calculated in advance after reach set approximation.

Alternative Intersection Implementation: The alternative used for intersection selected based on polygon intersection algorithms review [160], the selected algorithm is *Shamos-Hoey* [161].

The implementation was tested on *Storm scenario* (sec. 7.3.4) and it yields the same results.

6.5.4 Data fusion

Summary: There is a need for the final threat assessment in the Avoidance Grid. The data fusion provides mechanisms to represent, process, and assess threat in the cell including the safety of trajectories in the RSA. The output of the data fusion procedure is used further in Avoidance run (sec. 6.6.1).

Introduction: The data fusion interfaces *Sensor Field* and *Information Sources* from *cell/trajectory properties*. The *Data Fusion Function* is outlined in (4.19).

First, there will be an outline of *Partial Rating* commutation. Then these ratings will be discredited into Boolean values as properties of *Avoidance Grid/Trajectory*. Then these Boolean values will be used for further classification of space into *Free(t)*, *Occupied(t)*, *Restricted(t)* and *Uncertain(t)*.

All mentioned ratings are the result of *Filtered Sensor Readings* from *Sensor Field* and *Information Sources* with prior processing. This section will focus on *final fuzzy value calculation* and *discretization*.

Note. All rating values are in the *range*: [0, 1], and they were introduced in previous sections.

Visibility: The *sensor reading* of *sensor* if *Sensor field* returns a value of *visibility* for cell space in time of decision t_i .

The *visibility* for the cell is given in (eq. 6.85) as minimal visibility calculated from all capable sensors in *Sensor Field*.

$$\text{visibility}(\text{cell}_{i,j,k}) = \min \left\{ \begin{array}{l} \text{visibility}(\text{cell}_{i,j,k}, \text{sensor}_i) : \\ \forall \text{sensor}_i \in \text{SensorField} \end{array} \right\} \quad (6.85)$$

The example of *visibility* calculation for *LiDAR* sensor is given in (fig. 6.13).

Note. Sensor reliability for *visibility* is already accounted for prior *data fusion*. If not *weighted average* should be used instead.

Detected Obstacle: Sensors detect the physical obstacles in *Sensor Field*. Each *sensor* returns *detected obstacle rating* in the range [0, 1] reflecting the probability of obstacle occurrence in a given cell.

The *maximal value* of *detected obstacle* rating is selected from readings multiplied by *visibility rating* to enforce *visibility bias*.

$$\begin{aligned} \text{obstacle}(\text{cell}_{i,j,k}) = \max & \left\{ \begin{array}{l} \text{obstacle}(\text{cell}_{i,j,k}, \text{sensor}_i) : \\ \forall \text{sensor}_i \in \text{SensorField} \end{array} \right\} \times \dots \\ & \dots \times \text{visibility}(\text{cell}_{i,j,k}) \end{aligned} \quad (6.86)$$

The example of *detected obstacle rating* calculation for *LiDAR* sensor is given in (eq. 6.50).

Map Obstacle: The *Information Sources* are feeding *Avoidance Grid* with partial information of *Map obstacle rating*. *Map Obstacle Rating* shows the certainty that *charted obstacle* is in a given cell. This property is bound to *Information Source*, and it has the range in $[0, 1]$.

The *Map Obstacle Rating* for a cell (eq. 6.87) is calculated as the product of maximal *Map Obstacle Rating* and *inverse visibility*. This gives *visibility biased* certainty of *Map Obstacle*.

$$\text{map}(\text{cell}_{i,j,k}) = \max \left\{ \begin{array}{l} \text{map}(\text{cell}_{i,j,k}, \text{source}_i) : \\ \forall \text{source}_i \in \text{InformationSources} \end{array} \right\} \times \dots \times (1 - \text{visibility}(\text{cell}_{i,j,k})) \quad (6.87)$$

The example of *Map Obstacle Rating* calculation is given in (fig. 6.13).

Intruder: There is a set of *Active Intruders*, each intruder is using its *parametric intersection model*. This parametric *intersection* model calculates *partial intersection ratings* representing *intersection certainty* ranging in $[0, 1]$. The more *partial intersection rating* is closer to 1 the higher is the probability of aerial collision with that intruder in that cell.

The *geometrical bias* is used for cumulative of multiple intruders; the *intruders are not cooperative*; therefore their occurrence cannot be addressed by the simple *maximum*. The proposed formula (eq. 6.88) is simply bypassing the intruder rating if there is one intruder. If there are more intruders, the geometrical bias is applied.

$$\text{intruder}(\text{cell}_{i,j,k}) = 1 - \prod_{\forall \text{intruder}_i \in \text{Intruders}} \left(1 - \text{intersection} \left(\text{cell}_{i,j,k}, \text{intruder}_i \right) \right) \quad (6.88)$$

The *intruder intersection models* are outlined in (app. C).

Constraint: The *constraints* are coming from various *Information Sources*, the *hierarchical constraint application* is resolved by higher level logic. All *constraints* in this context are considered as *hard*.

The *Constraints rating* (eq. 6.89) is in the range $[0, 1]$ reflecting certainty of constraint application in the cell (usually 1).

$$\text{constraint}(\text{cell}_{i,j,k}) = \max \left\{ \begin{array}{l} \text{constraint}(\text{cell}_{i,j,k}, \text{source}_i) : \\ \forall \text{source}_i \in \text{InformationSources} \end{array} \right\} \quad (6.89)$$

The *Constraint Rating* calculation example for *static* constraints is given in (sec. 6.5.3), the example for *moving* constraints is given by (def. 20).

Note. Weather is already considered in constraints; the weather is handled as soft/hard static/moving constraints.

Threat: The concept of threat is a *rating of expected harm* to receive in a given portion of space. The threat can be time-bound to *decision time* t_i (time sensitive *intruder intersection models*).

The *harm prioritization* is addressed by higher navigation logic (fig. 6.22). All *sources of harm* are considered as equal. The threat is formalized in the *following definition*:

Definition 21. *The Threat is considered as any source of harm. The threat is a maximal aggregation of various harm ratings. Our threat for a specific cell is defined by (eq. 6.90).*

$$\text{threat}(\text{cell}_{i,j,k}) = \max \left\{ \begin{array}{l} \text{obstacle}(\text{cell}_{i,j,k}), \text{map}(\text{cell}_{i,j,k}), \\ \text{intruder}(\text{cell}_{i,j,k}), \text{constraint}(\text{cell}_{i,j,k}) \end{array} \right\} \quad (6.90)$$

Reachability: The *Reachability* for trajectory reflects how safe is the *path along*. The *Threat* (def. 21) for each cell has been already assessed. The set of *Passing Cells* is defined in *Trajectory Footprint* (eq. 6.29).

The *Trajectory Reachability* is given as a product of *Threats* along the trajectory (eq. 6.91). The *Trajectory Reachability* can be calculated for each *trajectory segment* given as $\{\text{movement}_1, \dots, \text{movement}_i\} \subset \text{Buffer}$ originating from state_0 .

$$\text{reachability}(\text{Trajectory}) = \prod_{\substack{\forall \text{cell}_{i,j,k} \in \\ \text{PassingCells}}} (1 - \text{threat}(\text{cell}_{i,j,k})) \quad (6.91)$$

Note. The *Reachability* of *trajectory segment* gives the property of *safety* of route from the beginning, until the last point of the segment. There can be a very unsafe trajectory which is very safe from the beginning.

The *Reachability* of the *cell* is given by the best trajectory segment passing through the *given cell*. This is given by property, that every trajectory is originating from root state_0 , which means that one safe route is sufficient to reach space in the cell.

The *Trajectory segment* reachability is sufficient, because the overall performance is not interesting, the *local reachability* is sufficient. The cell reachability is formally defined in (eq. 6.92).

$$\begin{aligned} \text{reachability}(\text{cell}_{i,j,k}) = \max\{\text{Trajectory}.Segment(\text{cell}_{i,j,k}).\text{Reachability} : \\ \forall \text{Trajectory} \in \text{PassingTrajectories}(\text{cell}_i, j, k)\} \quad (6.92) \end{aligned}$$

Note. Function $\text{Trajectory}.Segment(\text{cell}_{i,j,k})$. Reachability gives same results for any segment in $\text{cell}_{i,j,k}$, because (eq. 6.91) accounts each cell *threat* only once.

Discretization: The *fault tolerant* implementation needs to implement sharp Boolean values of properties mentioned before. The *fuzzy values* are usually threshold to Boolean equivalent. The *operational standards* for *Manned Aviation* [3] demands the fail rate below 10^{-7} because there is no definition for *UAS* the *minimal fail rate* is expected to be at a similar level.

The *fuzzy values* $[0, 1]$ are projected to *Boolean* properties of *cell* and *Trajectory* in the following manner (tab. 6.3).

The high values of *Visibility* (eq. 6.85) and *Reachability* (eq. 6.92, 6.91) are expected. The low *threshold* for *threats* values is expected. The error margin is solved by *Sensor Fusion*, therefore, initial *false positive* cases have a low rate. The *Detected Obstacle Rate* (eq. 6.86), *Map Obstacle Rate* (eq. 6.87), *Intruder Rate* (eq. 6.88), and *Constraint Rate* (eq. 6.89) thresholds are considered low.

| Threshold = 10^{-7} | | |
|-----------------------|-------------------------------------|-------------------------------|
| Visible | $visibility(\text{cell}_{i,j,k})$ | $\geq (1 - \text{threshold})$ |
| Detected Obstacle | $obstacle(\text{cell}_{i,j,k})$ | $\geq \text{threshold}$ |
| Map Obstacle | $map(\text{cell}_{i,j,k})$ | $\geq \text{threshold}$ |
| Intruder | $intruder(\text{cell}_{i,j,k})$ | $\geq \text{threshold}$ |
| Constraint | $constraint(\text{cell}_{i,j,k})$ | $\geq \text{threshold}$ |
| Reachable Trajectory | $reachability(\text{trajectory})$ | $\geq (1 - \text{threshold})$ |
| Reachable Cell | $reachability(\text{cell}_{i,j,k})$ | $\geq (1 - \text{threshold})$ |

Table 6.3: Changing ratings from fuzzy to Boolean parameters.

Space Classification: The *Data Fusion Function* is outlined in (4.19). This classification is resulting in four distinct cell sets.

The *Uncertain* space for decision time t_i is a portion of *Avoidance Grid* which *UAS* cannot *read* with *Sensor Field*. The *cells* with a $\neg \text{Visible}$ property. The *Uncertain* space is given by (eq. 6.93).

$$\text{Uncertain}(t_i) = \{\text{cell}_{i,j,k} : \text{cell}_{i,j,k} \in \text{AvoidanceGrid}(t_i), \text{cell}_{i,j,k}. \neg \text{Visible}\} \quad (6.93)$$

The *Occupied* space for decision time t_i is the set of cells which are classified as *Detected*

Obstacles. The *Visibility* is not an issue, due to the initial damping in (eq. 6.86). The formal definition is the space portion where it is possible to detect *obstacle bodies* or their portions (eq. 6.94).

$$Occupied(t_i) = \left\{ cell_{i,j,k} : \begin{array}{l} cell_{i,j,k} \in AvoidanceGrid(t_i), \\ cell_{i,j,k}.DetectedObstacle \end{array} \right\} \quad (6.94)$$

The *Constrained* space for decision time t_i is *Visible* portion of *Avoidance Grid* where the *Intruder* or *Constraint* is present. The mathematical formulation is given in (eq. 6.95).

$$Constrained(t_i) = \left\{ cell_{i,j,k} : \begin{array}{l} cell_{i,j,k} \in AvoidanceGrid(t_i), \\ cell_{i,j,k}.Visible, \\ cell_{i,j,k}.Constraint \vee cell_{i,j,k}.Intruder \end{array} \right\} \quad (6.95)$$

The *Free* space is the space which is *Visible* and \neg *Obstacle*, \neg *Intruder*, and, \neg *Constrained*. The mathematical definition is simple set subtractions from *Avoidance Grid* (eq. 6.96).

$$\begin{aligned} Free(t_i) &= AvoidanceGrid(t_i) - \dots \\ &\dots - (Uncertain(t_i) \cup Occupied(t_i) \cup Constrained(t_i)) \end{aligned} \quad (6.96)$$

The *Reachable* space for time t_i , used in *Avoidance* because its free and there is a safe trajectory, is given as a set of cells from *Avoidance Grid* which are *Reachable*. The mathematical definition is given in (eq. 6.97).

$$Reachable(t_i) = \left\{ cell_{i,j,k} : \begin{array}{l} cell_{i,j,k} \in AvoidanceGrid(t_i), \\ cell_{i,j,k}.Reachable \end{array} \right\} \quad (6.97)$$

Note. The Reachable Space at decision time t_i : The *Reachable space* is a non-empty set and its a subset of $Free(t_i)$ space:

$$|Reachable(t_i)| > 0, \quad Reachable(t_i) \subset Free(t) \quad (6.98)$$

6.6 Avoidance Concept

Summary: There is a need for a functional orchestration of previous concepts to achieve avoidance and navigation capabilities. The avoidance grid threat assessment done in (sec. 6.5.4) needs to be applied on the RSA of choice to produce a safe trajectory for one fixed time. This procedure is described in the avoidance grid run. There is a need to join output multiple avoidances runs over the time to achieve the required avoidance/navigation capabilities. This procedure is described in the navigation run. There is a need to assess the computational complexity of the approach to show implementation feasibility.

This section introduces *Platform Independent Avoidance Concept* core functionality (fig. 6.2) modules responsible for *pathfinding* and *navigation*. The sections are organized like follow:

1. *Avoidance Grid Run* (sec.6.6.1) (inner avoidance run) - the *best pathfinding* in one *Avoidance Grid* with *situation assessment* done.
2. *Mission Control Run* (sec . 6.6.2) (outer navigation run) - main navigation and decision making an algorithm for *non-cooperative obstacle avoidance*.
3. *Computational Complexity* (sec. 6.6.3) - the *computational feasibility study* and *weak point identification* of our approach.

6.6.1 Avoidance Grid Run

Summary: Based on the provided threat assessment find the optimal trajectory in compliance with safety and given navigation goal.

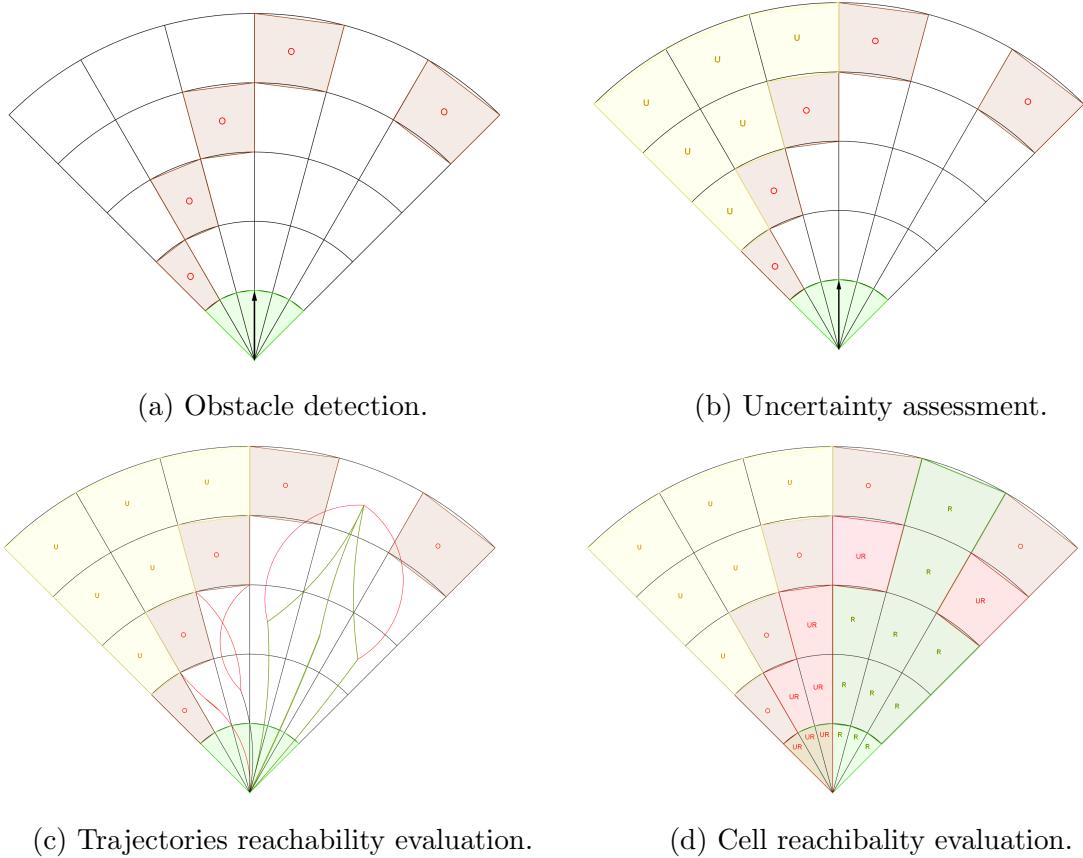
Main Goal: The main goal of this section is to introduce the trajectory selection process, based on a *situation assessment*, originating from *Data Fusion Procedure* (sec. 6.5.4).

Note. The *rating calculation* is outlined in (sec. 6.5.4). Low-cost sensor fusion example usable to feed our data fusion procedure is given in [164]. Semi-optimal concatenation trajectory search like ours can be found in [165].

Note. The *Sensor Fusion Procedure* is solving all the following steps (sec. 6.5.4). The *main purpose* of *Avoidance Run* is finding the best path under certain conditions.

Space Assessment Principle: The *Avoidance Grid* is fed through *Data Fusion* (sec. 6.5.4). The process of *rating assessment* (tab. 6.3) is given in (fig. 6.16):

1. *Obstacle detection* (fig. 6.16a) - assessment of *detected obstacles* (eq. 6.86). The red (O) *cells* have *Detected obstacle* set as *true*. The other threats: *map obstacles* (eq. 6.87), *intruders* (eq. 6.88), *constraints* (eq. 6.89) are *false*. The red (O) *cells* are representing $\text{Occupied}(t_i)$ (eq. 6.94) space in *Avoidance Grid* at decision time t_i .
2. *Uncertainty assessment* (fig. 6.16b) - the uncertain cells are cells which status cannot be *assessed*. The *Visibility* (eq. 6.85) is low. The *Uncertain* cells (yellow (U) mark) are equal to $\text{Uncertain}(t_i)$ (eq. 6.93) in *Avoidance Grid* in *decision time* t_i . The $\text{Constrained}(t_i)$ (eq. 6.94) space is equal to \emptyset in this example.
3. *Trajectory reachability evaluation* (fig. 6.16c) - the *Reach Set* given as *Trajectory Set* (eq. 6.25). is then projected through *Avoidance Grid* and pruned according to (def. 15). *Reachable Trajectories* (eq. 6.91) are only those contained in $\text{Free}(t_i)$ space (eq. 6.96). The *Reachable Trajectories* are denoted as *green lines*. The *Unreachable* trajectory segments are denoted as *red lines*.
4. *Cell reachability evaluation* (fig. 6.16d) - the evaluation of *cells* reachability is going according to (eq. 6.92). The *Reachable cells* are those which *contains* at least one *Reachable Trajectory Segment*.

Figure 6.16: Significant steps of *Avoidance grid run* (inner loop).

Finding Best Path: ² Each $cell_{i,j,k}$ in *Avoidance Grid* at *decision time* t_i has assessed ratings according to *data fusion procedure* (tab. 6.3). The following properties are known prior the *trajectory* selection:

1. *Reachability* for each $cell_{i,j,k}$ (eq. 6.92).
2. *Reachability* for each *Trajectory*(\circ) (eq. 6.91).
3. *Free Space* as non-empty set of *cells* in *Avoidance Grid* (eq. 6.96), with *Reachable Space* (eq. 6.97).
4. *Goal Waypoint* WP_G from *Mission Control Run* (sec. 6.6.2).

²Avoidance Run Function Implementation: RuleEngine/MissionControl/MissionControl.m::
findBestPath(avoidanceGrid)

Algorithm 6.6: Find best Path in Avoidance Grid

Input : Cell[] reachable (eq. 6.97), Waypoint goal, AvoidanceGrid(t_i) grid

Output: Trajectory avoidancePath, Error message

```

# Initialization & Reachability test;
avoidancePath = ∅;
if reachable == ∅ then
| return [avoidancePath, "No path available, empty Reach Set"]
end

avoidanceCell = GetRandomCell(reachable);

# Look for for goal cell;
if goal ∈ grid then
    # Goal is inside Avoidance Grid, Check if reachable;
    avoidanceCell = grid.selectCellXYZ(goal);
    if avoidanceCell.Reachable != true then
        | return [avoidancePath, "Waypoint not Reachable"]
    end
else
    # Goal is outside Avoidance Grid, look for closest reachable celli,j,k;
    minimalDistance = distance(avoidanceCell,goal);
    for celli,j,k ∈ reachable do
        if distance(celli,j,k,goal) < minimalDistance then
            | if isOuterCell(celli,j,k) then
                | | minimalDistance = distance(celli,j,k,goal);
                | | avoidanceCell = celli,j,k;
            | end
        end
    end
end

# Reachable cell was found, Look for cheapest reachable trajectory;
avoidancePath = GetRandomTrajectory(avoidanceCell);
for trajectory ∈ avoidance Cell && trajectory.Reachable == true do
    if trajectory.Cost < avoidancePath.cost then
        | avoidancePath = trajectory;
    end
end

message = ∅;
return [avoidancePath,message]

```

The *Algorithm* (alg. 6.6) is based on *shortest path* search. Navigation is trying to reach *goal waypoint*; therefore it tries to shorten the distance between *final trajectory cell*

and *goal waypoint*. If there is *reachable space* two situations can occur:

1. *Goal waypoint is inside the Avoidance Grid* - the avoidance cell is $\text{cell}_{i,j,k}$ containing *goal waypoint* if reachable.
2. *Goal waypoint is outside the Avoidance Grid* - the avoidance cell is the closest cell considered as an *outer cell* to *goal waypoint*.

The *Avoidance Path* selection is simple lowest cost selection of $\text{Trajectory} \in \text{cell}_{i,j,k}$.

Note. *Outer cell* is a $\text{cell}_{i,j,k}$ which has at least one *wall* directly neighbouring with *outer space* (*Universe – KnownWorld(t_i)*). The *outer cell* is selected to prevent navigation to the *trap*.

Space Assessment Example: For better understanding, there is the following example of *space assessment* and *Best Path Selection*.

The *UAS* (blue plane) is following a *mission plan* in open space. Then there is a detection of a *collision situation* (fig. 6.17). The *Obstacle* is detected in the *top-right* Avoidance Grid corner.

The *LiDAR hits* are denoted as red filled circles. The *Avoidance Grid* space is constrained by the black dashed line. The *Avoidance Grid* is separated into five layers going from top to *bottom*. The *Reach Set* is projected as a set of *Trajectories* with colorization.

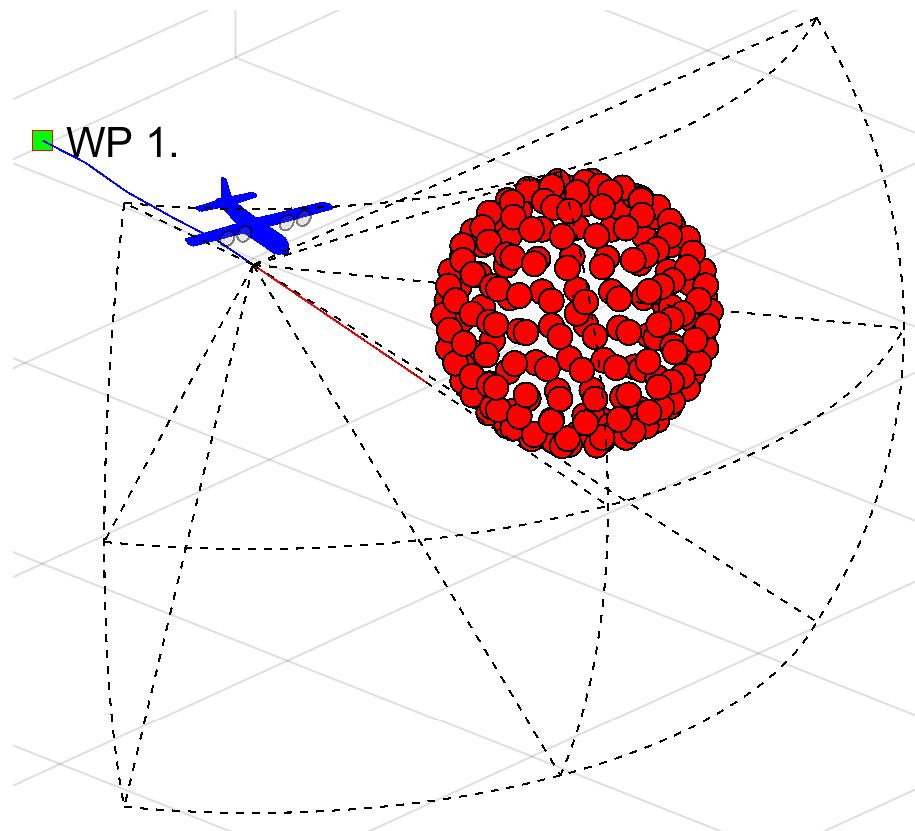


Figure 6.17: Example: The situation to be evaluated by *Avoidance Run*.

Visibility Assessment: The visibility assessment (fig. 6.18) divides the *Avoidance Grid* into two

1. *Visible space* (blue filled cells) is space *through* which *LiDAR* rays roamed freely until they hit an *Obstacle*.
2. *Uncertain space* (black filled cells) is space where no *LiDAR ray* passed nor hit. Therefore its status is uncertain.

Note. The *detected obstacle cells* are part of *visible space* because there is certainty about its containment.

The *Reach Set* trajectories are colored based on their visibility, blue for *uncertain* trajectories and green for visible trajectories.

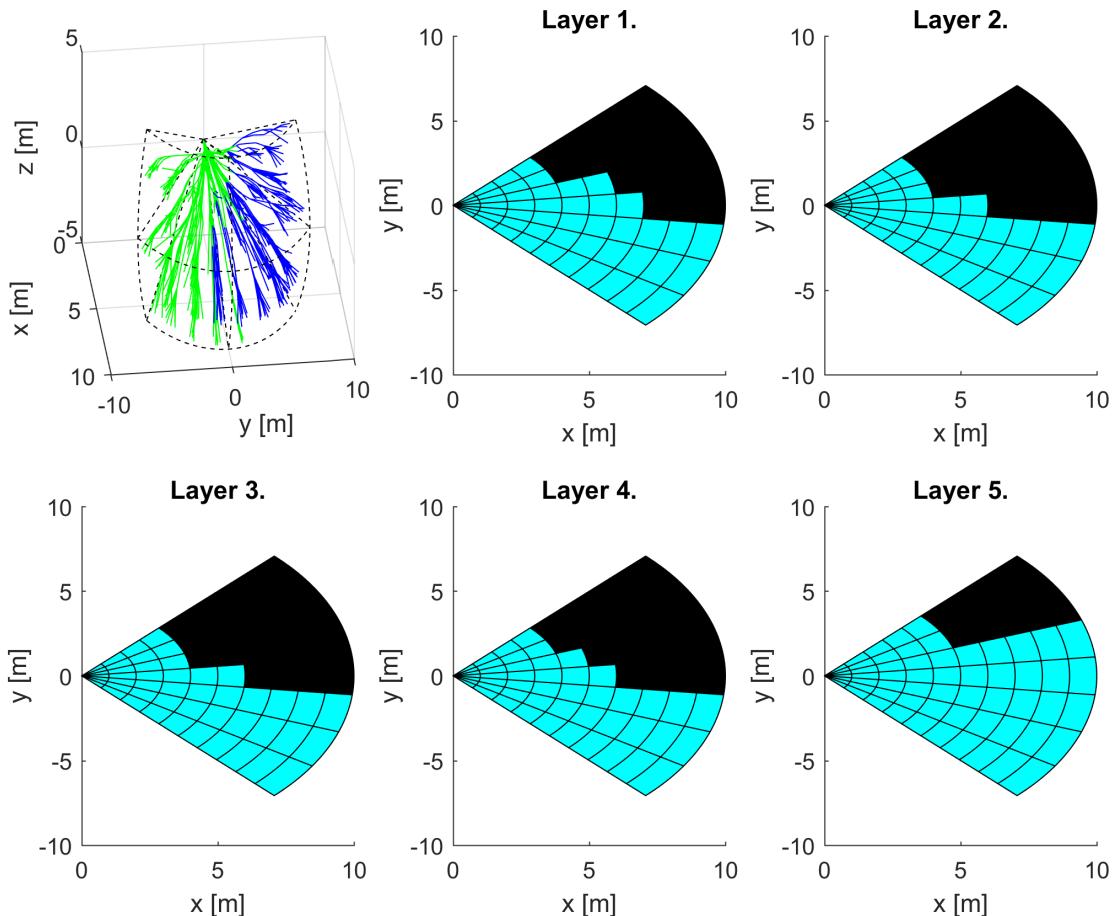


Figure 6.18: Example: The *Visibility* evaluation by *Avoidance Run*.

Reachability Assessment: For Each trajectory, the *Reachability* is assessed (fig. 6.19). The *Obstacle Space* and *Uncertain Space* are rendering *reachability*, effectively separating *trajectories* into two categories:

1. *Unreachable Trajectories* (red lines) - there is at least one trajectory segment leading through *Obstacle* or *Uncertain* space.
2. *Reachable Trajectories* (green lines) - all trajectory segments are lying in *Free space*.

Cells in Avoidance grid are divided in a similar matter, depending on the count of *reachable trajectories* passing through them:

1. *Unreachable Cells* (red fill) - there is no trajectory through *free space* or the *cell* is not in *free space*.
2. *Reachable cells* (green fill) - there is at least one *feasible trajectory* reaching *free cell*.

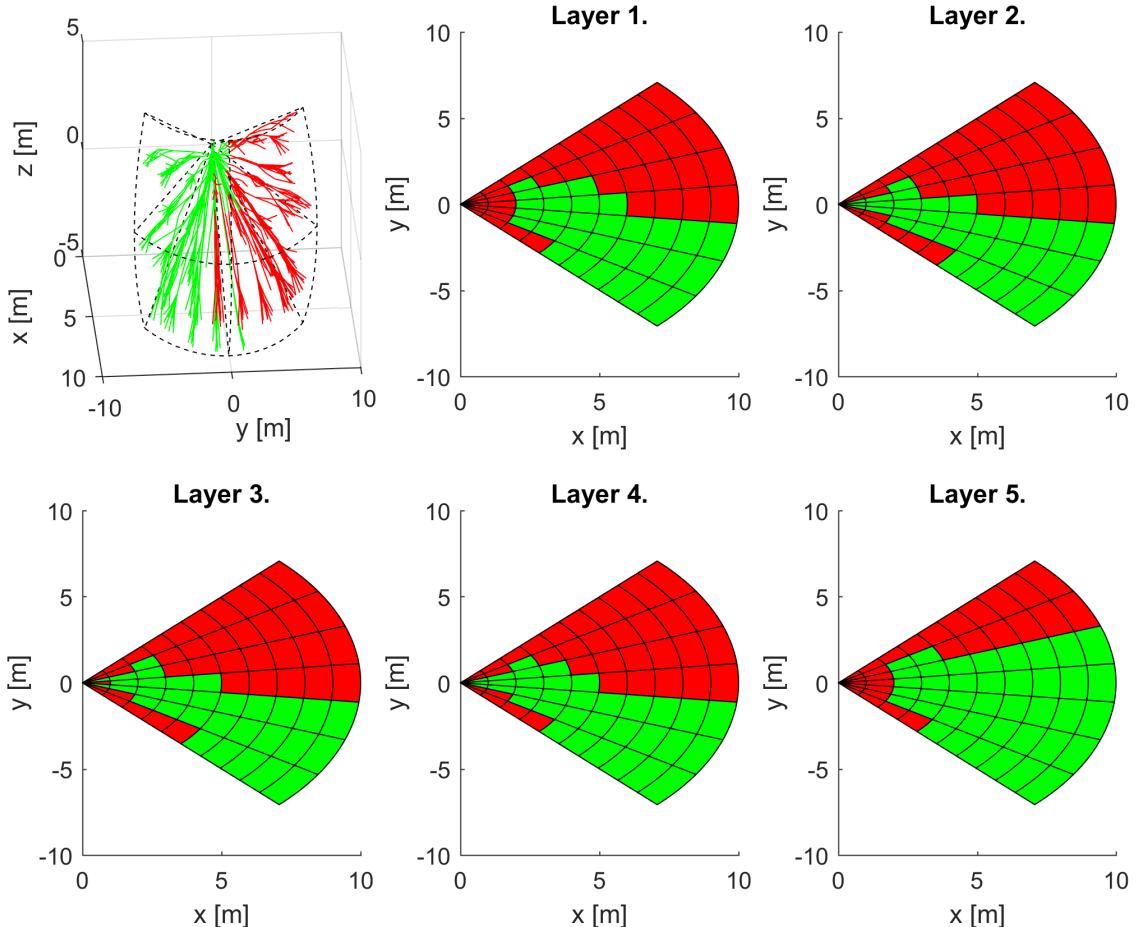


Figure 6.19: Example: The *Reachability* evaluation by *Avoidance Run*.

Note. The *best avoidance path* is selected form *reachable outer cells* (green fill in fig. 6.19), depending on *goal waypoint* according to (alg. 6.6).

6.6.2 Mission Control Run

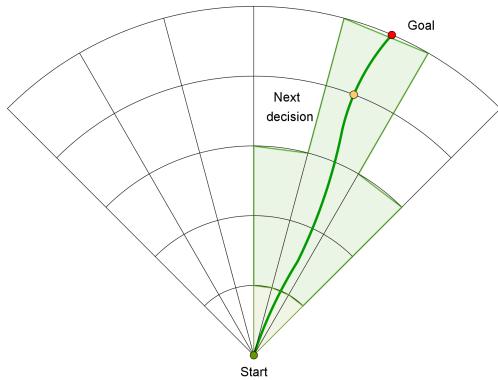
Summary: Event-based navigation algorithm connecting results of the multiple Avoidance Grid Runs over time to generate a trajectory satisfying the mission. The concept of discrete future events is introduced to support the processing of various threats and commands. The overview of process description and thread orchestration is provided.

Introduction and Motivation: This section will introduce *Navigation Concept* using *Reach Set Approximation*. The *Avoidance Framework Concept* (fig. 6.2) defines *Navigation Module* as a *sub-system* for long term *trajectory tracking*. The *Avoidance Grid Run* (sec. 6.6.1) is solving the *Path Search* problem inside operation space constrained by *Avoidance Grid* for time t_i .

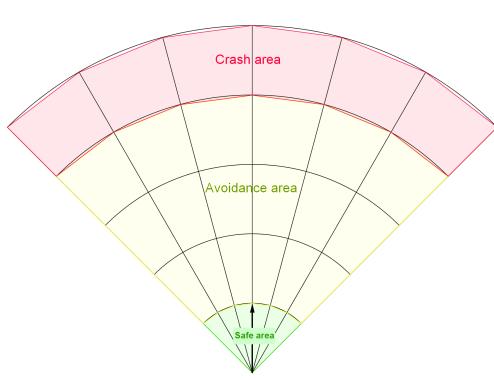
There is a need to build a trajectory between *Waypoints* which are further away than the *distance* of one *Avoidance Grid*. The *UAS* is controlled via *Movement Automaton*. The *Movements* which are in *Movement Buffer* can be replaced with other movements. This feature of *Movement Automaton* is called *Movement Chaining* (eq. B.8).

To join the multiple *Avoidance Grids* paths following terminology needs to be established (fig. 6.20a):

1. *Goal* (Selecting Goal of Navigation) - the point where UAS want to get in the global coordinate frame. The selection needs to be defined.
2. *Next Decision* - the point when the next *Avoidance Grid Run* is applied. The outline of events and triggers is required. The *decision* will be made in the *next decision time* t_{i+1} .



(a) Mission control run example.



(b) Grid Zones.

Figure 6.20: Definitions for *Mission Control Run* (outer loop).

The *Avoidance Grid* from *UAS* viewpoint can be separated into following zones (fig. 6.20b):

1. *Crash Area* (last layers) - there is no place for safe return and the *border* of *Avoidance Grid* is near. The *Decision Point* needs to lie before this zone.

2. *Avoidance Area* (middle layers) - the area of *Active Avoidance Maneuvering*. The *Reach Set Approximation* performance (sec. 6.4.2) is important in this area.
3. *Safe Zone* (first layers) - there is space for safe return or damage mitigation.

Joining *Avoidance Grid Runs* (fig. 6.21) example portrays *Avoidance Grid Runs* invoked on various *Decision Points* to achieve *Navigation* functionality. The UAS (blue plane) is flying Mission (green numbered waypoints). The *Avoidance Grid* boundary (black dashed line) for each *Decision Point* (UAS position at time t_i). Following the example of *Navigation* (fig. 6.22) run is shown:

1. *Mission Start* (fig. 6.21a) - UAS at the start of the mission have one *Avoidance Grid* at its position to determine the *Navigation Path* to *Waypoint 2* (goal waypoint). The planned path (red line) is leading directly to *Avoidance Grid* boundary (black dashed line).
2. *Mission End* (fig. 6.21b) - UAS have reached the *last waypoint*. All *Avoidance Grid* boundaries (black dashed line) for all *runs* are drawn along flown trajectory.
3. *Waypoint Reach* (fig. 6.21c) - the *waypoint* is inside *Avoidance Grid*, the navigation path (red line) leads directly to *goal waypoint*. (Excessive *Avoidance Grid* boundaries are removed.)
4. *Next Waypoint* (fig. 6.21d) - the new *Goal Waypoint* is selected, the UAS moves to new goal (invoking *Avoidance Grid Runs* when necessary).

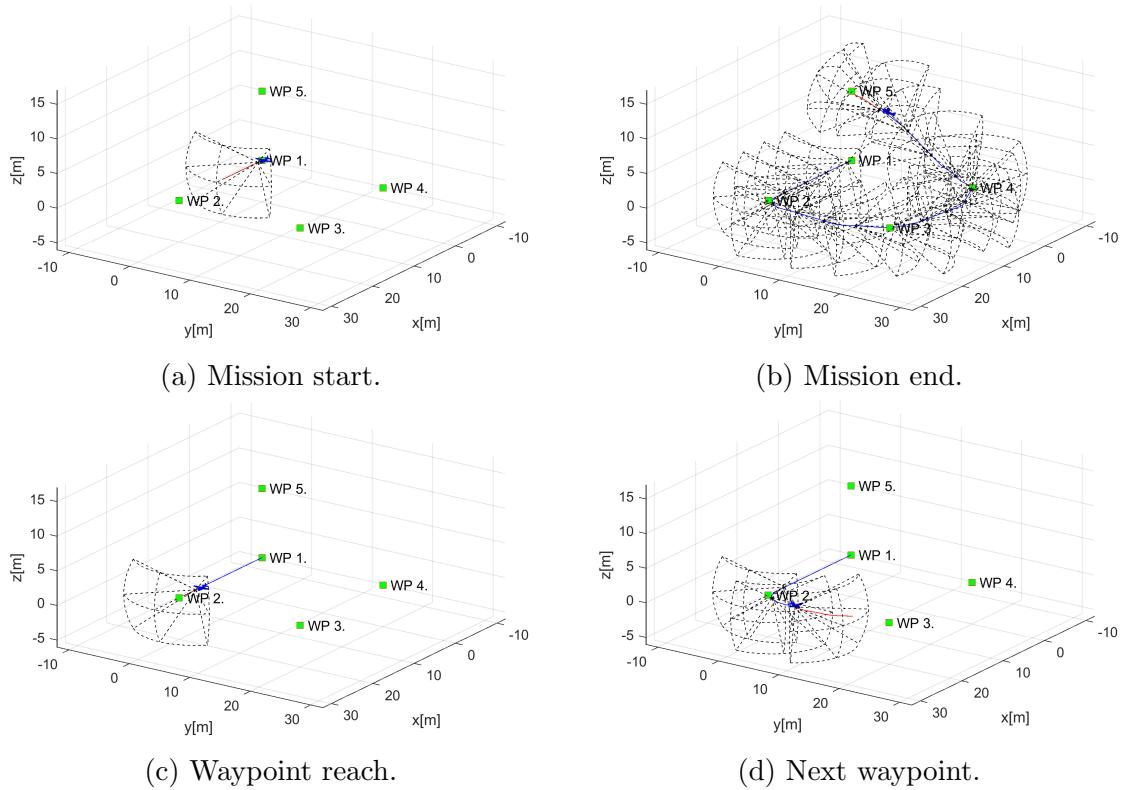


Figure 6.21: Joining multiple *Avoidance Grid Runs* to achieve *Navigation*.

General Concept:³ The *General Concept* is taken from [166, 167], consisting of following main modules:

1. *Navigation Loop* - module responsible for *Navigation* providing *Goal Waypoint*.
2. *Data Fusion* (background in sec. 6.5.4) - module responsible for *Surveillance Data Feed*.
3. *Situation Assessment* - module responsible for *UAS Safety Evaluation*.
4. *Avoidance Run* (background in sec. 6.6.1) responsible for *Avoidance Path* selection.

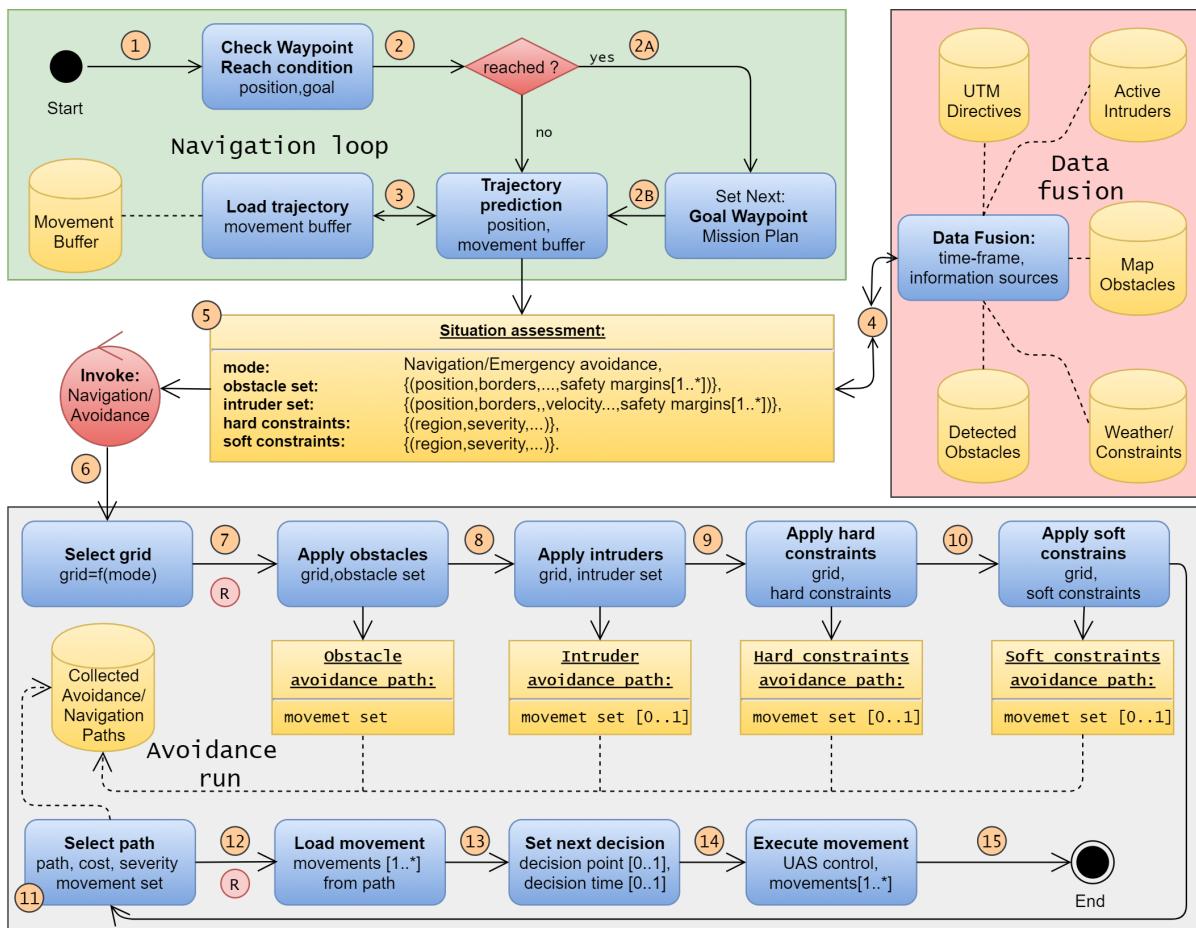


Figure 6.22: Mission control run activity diagram.

The main changes to *Navigation architecture* are given in *Mission Control Run* activity diagram (fig. 6.22):

1. *Situation Assessment* - added event-based mode switching control.
2. *Avoidance Run* - added hierarchical evaluation for *Avoidance Path* selection; This is responsible for prioritizing threat avoidance according to a type.

³Mission Control Run Function Implementation: RuleEngine/MissionControl/MissionControl.m::runOnce(.)

The *Operation Mode* is introduced, based on *Situation assessment* and *Triggering Events* one of the following modes are selected in *Avoidance Run*:

1. *Navigation Mode* - the *UAS* is navigating through *Airspace* following *cost-effective patterns* and obeying *Airspace Authority* (UTM). The *Navigation Grid* is an instance of *Avoidance Grid* (sec. 6.3) with initialized *Navigation Reach Set* (ex. *Turn-Minimizing Reach Set Approximation* (sec. 6.4.5)).
2. *Emergency Avoidance Mode* - the *UAS* is *threatened* by obstacle, intruder, hard constraint or *soft constraint*, the *UAS* is navigating through *Airspace* following *safe avoidance patterns* and *minimizing the impact* of possible damages. The *Avoidance Grid* is a term used for *Emergency Avoidance Mode*. The *Avoidance Reach Set Approximation* is initialized in *Avoidance Grid* (ex. *Coverage-Maximizing Reach Set Approximation* (sec. 6.4.4))

Note. Depending on *Operation Mode* the pair of *Avoidance Grid* and *Reach Set* is selected in *Avoidance Run* part.

The *Navigation Grid* and *Avoidance Grid* share the space portioning pattern; therefore the *Data Fusion* (sec. 6.5.4) needs to be evaluated only once for both grids.

Decision Time Frame ($[t_i, t_{i+1}]$): The *Mission Control Run* is executed for *Decision Time Frame* bounded to the *period* of the *UAS* *executed movement* (fig. 6.2).

The *UAS System* (sec. 6.2.2) controlled by *Movement Automaton Implementation* (sec. 6.2.3) *Planned Movements* can be changed at any time. The real impact on control is shown after the *actual movement* is executed.

Note. For our *Movement Automaton Implementation* movements, the average *movement duration* is *1/velocity second* (tab. 6.1, 6.2).

The *Decisions* are made based on *system state* in *current time-frame* started at t_i for the *next time frame* starting at t_{i+1} .

Note. Because the *Decision Delay* is crucial in *Avoidance System*, it is beneficial to have *short time movements*. On the other hands, the *length and duration of movements* are impacting *Reach Set Complexity*. The proper construction of movement automaton is greatly impacting overall *approach performance*.

Initialization: The *UAS* is going to solve a problem for *Rules of the Air* (eq. 4.27). Using control scheme (fig. 6.2) with given *Sensors*:

$$Sensors = \{LiDAR, ADS - B\} \quad (6.99)$$

The sensors obstacle assessment into avoidance grid is outlined for static obstacles in (sec. 6.5.1) and for moving obstacles in (sec. 6.5.2.)

The *Data Fusion Procedure* is given as follow:

$$DataFusion = \{RatingBasedDataFusion \quad (\text{sec.6.5.4})\} \quad (6.100)$$

Then the *UAS system* (sec. 6.2.2) with *Movement Automaton Implementation* (sec. 6.2.3) with empty movement buffer:

$$MovementBuffer = \{\} \quad (6.101)$$

The *Avoidance Grids* for both *Operation Modes* are created with *identical space segmentation*. The *Reach Set Approximations* are loaded based on initial *UAS State* at decision time 0. The *Reach Set Approximation* is always selected based on *UAS System State*. The initial *Operation Mode* is set up as *Navigation*. The initialization is summarized like follow:

$$\begin{aligned} AvoidanceGrid(0) &= \{UAS.position(0), AvoidanceReachSet(UAS.ReachSet)\} \\ NavigationGrid(0) &= \{UAS.position(0), NavigationReachSet(UAS.ReachSet)\} \\ OperationMode &= Navigation \end{aligned} \quad (6.102)$$

The *Mission* is set up as a set of *ordered waypoints*. The *initial goal waypoint* is *first waypoint*. The initialization is summarized like follow:

$$\begin{aligned} Mission &= \{Waypoint_1 \dots Waypoint_n\} \\ GoalWaypoint &= Mission.waypoint_1 \\ LastWaypoint &= Mission.waypoint_n \end{aligned} \quad (6.103)$$

The *actual threats* are set as empty sets for *decision time* $t_i = 0$:

$$obstacles = \{\}, intruders = \{\}, hardConstraints = \{\}, softConstraints = \{\} \quad (6.104)$$

Navigation Loop (1st-3rd step): The purpose of *Navigation Loop* is to select proper *Goal Waypoint* from *Mission* (sec. 4.1.2). If *last waypoint* have been reached the *Landing Procedure* will be initiated and *Mission Control Run* Ends.

First, start with the definition of *waypoint reach condition* (def. 22) and *Unreachable waypoint* (def. 23).

Definition 22. *Waypoint Reach Condition for current decision time t_i for UAS position and current Goal Waypoint is satisfied only if:*

$$\begin{aligned} & \text{distance}(UAS.\text{position}(t_i), \text{GoalWaypoint}(t_i)) \\ & \leq \\ & 2 \times \max \{ \text{length}(\text{movement}) : \forall \text{movement} \in \text{MovementSet} \} \end{aligned} \quad (6.105)$$

Note. The movements in our solution have a *uniform length* of 1 m (tab. 6.1, 6.2), therefore the waypoint reach condition is satisfied when the *distance to goal waypoint* is lesser than 2 m. The maximal movement length has an impact on *navigation/avoidance precision*.

Definition 23. *Unreachable Waypoint.* The Goal Waypoint evaluates as *unreachable* in decision time t_i when Avoidance Grid Run (alg. 6.6) cannot find the navigation/avoidance path leading to it.

Formally: The Avoidance/Navigation Grid has range defined as final layer distance. When the Goal Waypoint is in range of Grid:

$$\text{Grid}(t_i).\text{range} \geq \text{distance}(UAS.\text{position}(t_i), \text{GoalWaypoint}(t_i)) \quad (6.106)$$

and following condition is satisfied:

$$\begin{aligned} & \forall \text{cell}_{i,j,k} \in \text{Grid}(t_i) \ \exists \text{cell}_{i,j,k}.\text{Reachable} == \text{true} \wedge \dots \\ & \dots \wedge \text{distance}(\text{cell}_{i,j,k}, \text{GoalWaypoint}(t_i)) \leq \dots \\ & \dots \leq 2 \times \max \{ \text{length}(\text{movement}) : \forall \text{movement} \in \text{MovementSet} \} \end{aligned} \quad (6.107)$$

The Goal Waypoint is *unreachable*.

Then the *Navigation Loop* is invoked every *decision time t_i , Mission Control Run* (fig. 6.22), it is described as a sequence of the following steps:

1st Check Waypoint Reach Condition - the *UAS position* for given a *time frame* t_i is checked under condition (eq. 6.105). If the condition is met continue with 2nd step otherwise continue with 3rd step.

2nd Set Next Waypoint - until the following condition is met:

$$\text{GoalWaypoint} == \text{LastWaypoint}$$

Set next goal waypoint like follow:

$$\text{GoalWaypoint} = \text{Mission.getNextWaypoint}()$$

Otherwise, enforce *Landing sequence* (Out of Scope).

3rd Trajectory Prediction - the *Movement Buffer* is loaded with planned movements from *Movement Automaton*. The *future trajectory* is predicted according to (eq. B.33):

$$\begin{aligned} PredictedTrajectory = \\ Trajectory(state = UAS.state(t_i), buffer = futureMovements) \end{aligned}$$

The *Predicted Trajectory* is used in 5th step *Situation Assessment*.

Data Fusion (4th step) The *Data Fusion* (sec. 6.5.4) in this context is *Threat Sets* preparation for *Avoidance Run*. It depends on the values of *Boolean values* defined in (tab. 6.3) for *threat* classification.

Note. Avoidance Grid's Data fusion (sec. 6.5.4) is run in the 7th- 10th step (fig. 6.22).

The *static obstacles* source is from *LiDAR* scan received at least at the beginning of current *decision frame* t_i :

$$obstacles = LiDAR.scan(UAS.position(t_i))$$

The *intruder's* source are valid *active intruders notifications* received from ADS-B In positioned to *future expected positions* at *decision time* t_{i+1} :

$$intruders = ADS - B.getActiveIntruders(t_{i+1})$$

Note. The *Intruders* needs to be predicted for the next decision time-frame starting at time t_{i+1} Due to their mobility.

The *hard/soft constraints* are obtained from *Information Sources* and the area of next decision time t_{i+1} *Avoidance Frame* is used as space parameter in the search. The sets of hard and soft constraints are obtained in the following manner:

$$hardConstraints = InformationSources.fuse(AvoidanceGrid(t_{i+1}))$$

$$softConstraints = InformationSources.fuse(AvoidanceGrid(t_{i+1}))$$

The results of *Data Fusion* threats set preparation are used in the next step.

Invoke Navigation/Avoidance based on Situation Assessment (5th-6th step): The *deciding events* depending on *Trajectory Prediction* (3rd step) and *Data Fusion* (4th step) (fig. 6.22) are the following:

1. *General Events* are triggered regardless *Operation Mode*. They are considered after *specific mode events* are handled and *Navigation/Avoidance Grid* is selected:

- a. *Empty Movement Buffer* ($MovementBuffer = \emptyset$) - if there is no movement in *Movement buffer* to be executed (from 3rd step: Load Trajectory), the *Avoidance Run* is enforced to run with *Navigation/Avoidance Reach Set Approximation* to generate the new path.
 - b. *Waypoint Reached* (2nd step) - the *Navigation Loop* run is forced to set goal *Goal Waypoint*. If the *last waypoint* from *Mission* (sec. 4.1.2) the *Landing Procedure* is enforced.
 - c. *Waypoint Unreachable* - this type of event is very situations based. The *Waypoint Reachability* (assumption. 4) has not been relaxed; therefore this event is not properly handled in approach. The *implementation* considers *selecting next waypoint in the mission* as a goal waypoint of the *first waypoint* if *unreached/unreachable waypoints* are exhausted.
2. *Navigation Mode Events* are triggered if *Operation Mode* is set as *Navigation*:
- a. *Empty Navigation Grid* ($|threats| = 0$) - if *movement buffer* contains at least one *movement*, the *Avoidance Run* is omitted. The *Operation Mode* stays in *Navigation Mode*.
 - b. *Collision Case Resolution* ($|ActiveCollisionCases| > 0$) - there is new/active *Collision Case* (sec. 6.7.6), the *Navigation Reach Set Approximation* trajectories will be constrained according to active *Collision Case(s)* requirements. If there exists at least one *Reachable* avoidance path, the *Operation Mode* will remain *Navigation*. If there is no *Reachable* avoidance path, the *Operation Mode* switches to *Emergency Avoidance*.
 - c. *Static Obstacle Detection* ($LiDAR.Hits > threshold$) - if *static obstacle set* contains at least one *detected obstacle* (eq. 6.50) intersecting with *Navigation grid* the *Operation Mode* will be switched to *Emergency Avoidance Mode*.
 - d. *Intruder Detection* ($intruders > 0$) - if *active intruders set* contains at least one *intruder* which expected impact area (intersection models (app. C)) *Navigation grid* the *Operation Mode* will be switched to *Emergency Avoidance Mode*.
 - e. *Hard or Soft Constraint Occurrence* ($|hardConstraints| > 0 \vee |softConstraints| > 0$) - if *hard/soft constraint set* contains at least one *constraints* which intersects (static constraints (sec. 6.5.3), moving constraints (def. 20)) *Navigation grid* the *Operation Mode* will be switched to *Emergency Avoidance Mode*.
3. *Emergency Avoidance Events* are triggered if *Operation Mode* is set as *Emergency Avoidance*:
- a. *Empty Avoidance Grid* ($|threats| = 0$) - if there is no *detectable threat*, the remainder of *avoidance path* is removed from *Movement Buffer*. The *Operation Mode* is switched to *Navigation*, and new *navigation path* is selected.

- 5th Situation Assessment** - if there is any flag raised by *Event Triggers*, there is an *avoidance situation*.

The *Event Triggers* describe complex *Operation Mode* switching. The simplified principle is the following: *If UAS is in Emergency Avoidance Mode Always Invoke Avoidance Run. If UAS is in Navigation Mode Invoke Only if Necessary.*

If there was event trigger continue with 7th step, otherwise, wait for *next decision time* t_{i+1} , execute movement and continue with 1st step.

- 6th Invoke Navigation/Avoidance** depending on the *Operation Mode* the *Reach Set/Grid* pair is selected. The future $state(t_{i+1})$ in next decision frame t_{i+1} is necessary for Grid/Reach Set initialization. The *next decision frame initial state* is obtained by *prediction*:

$$state(t_{i+1}) = Trajectory(state(t_i), currentMovement)$$

The *Reach Set Approximation* is loaded based on *mode* and $state(t_{i+1})$. The *Grid* is initialized as $Free(t_{i+1})$ (eq. 6.96) for all cells.

Avoidance Run (7th-15th step): The *Avoidance Run* goal is to obtain *Path* represented as $Trajectory(state(t_{i+1}), MovementBuffer))$ (eq. B.33) from *Navigation/Avoidance Grid* and associated *Navigation/Avoidance Reach Set Approximation*.

If the *Operation Mode* is set as *Navigation Mode*, the algorithm continues with the 11th step. Otherwise, the *Avoidance Grid Space Assessment* is run multiple times to obtain $Reachable(t_{i+1})$ (eq. 6.97). The *Threat Data* obtained from the 4th step are used.

- 7th Apply Obstacles** - The *Space assessment* (tab. 6.3) for *Avoidance Grid* is calculated with following threat modification:

$$intruders = \emptyset, softConstraints = \emptyset, hardConstraints = \emptyset$$

The *Find Best Path* (alg. 6.6) is applied, the resulting *avoidance path* is labeled as *Obstacle Avoidance Path*.

- 8th Apply Intruders** - The *Space assessment* (tab. 6.3) for *Avoidance Grid* is calculated with following threat modification:

$$softConstraints = \emptyset, hardConstraints = \emptyset$$

The *Find Best Path* (alg. 6.6) is applied, the resulting *avoidance path* is labeled as *Intruders Avoidance Path*.

9th Apply Hard Constraints - The *Space assessment* (tab. 6.3) for *Avoidance Grid* is calculated with following threat modification:

$$\text{hardConstraints} = \emptyset$$

The *Find Best Path* (alg. 6.6) is applied, the resulting *avoidance path* is labeled as *Hard Constraint Avoidance Path*.

10th Apply Soft Constraints - The *Space assessment* (tab. 6.3) for *Avoidance Grid* is calculated without any modification.

The *Find Best Path* (alg. 6.6) is applied, the resulting *avoidance path* is labeled as *Soft Constraints Avoidance Path*.

Note. The 7th to 10th steps are code-optimized for efficient calculation.

11th Select Path - based on *Operation Mode* the *Navigation/Avoidance Path* is selected.

The *Navigation Path* for *Navigation Mode* is selected by a standard *Find Best Path* (alg. 6.6) procedure. The *Navigation Reach Set Approximation* can be constrained by *Rule Engine* (fig. 6.29).

The *Avoidance Path* for *Emergency Avoidance Mode* is selected from *Collected Avoidance Paths* with the following priority:

1. *Soft Constraints Avoidance Path* - if exists continue with 12th step, if does not exist try to select;
2. *Hard Constraints Avoidance Path* - if exists continue with 12th step, if does not exist try to select;
3. *Intruders Avoidance Path* - if exists continue with 12th step, if does not exist try to select;
4. *Obstacle Avoidance Path* - continue with the 12th step.

Note. The *Waypoint Reachability* (assumption 4) is weakened to the point that it is necessary for the waypoint to be *Reachable* only in static obstacle environment. The *Constrained* and *Occupied* spaces are shrunk in the following matter to increase UAS survival chances. There are following relaxations with their conditions:

1. *Soft Constraint Relaxation* - they are breakable by default. This kind of situation is allowed to happen under any circumstances.
2. *Hard Constraints Relaxation* - they can be broken in case of emergency (airspace constraints) or UAS robust build (Weather Constraints). This kind of situation is allowed under very specific conditions depending on *broken constraint* severity.

3. *Intruder Occupied Space Relaxation* - this can be broken if and only if there is guarantee the Intruder dynamic and navigation algorithm allows to avoid *Collision* with UAS. This relaxation should be used as *the last resort*.

12th Load Movements - the *Movement Buffer* is flushed for *future decision times* t_{i+1}, \dots, t_{i+k} . The *Navigation/Avoidance Path* movements are pushed into *Movement Buffer* instead. The *executed movement* for *decision time* t_i remains (because movement is executed at this time point).

13th Set Next Decision - the *next decision point* is set depending on circumstances:

1. *Navigation Mode* (no active collision cases) - *Decision Point* is set as the point before *UAS* enters into *Crash Zone* (fig. 6.20b) in *Navigation Grid*.
2. *Navigation Mode* (at least one active collision case) - *Decision Point* is set after *next movement execution*. Current decision point *UAS.Position*(t_i), next decision point *UAS.Position*(t_{i+1}).
3. *Emergency Avoidance Mode* (any circumstances) - *Decision Point* is set after the *next movement execution*. Current decision point *UAS.Position*(t_i), next decision point *UAS.Position*(t_{i+1}).

14th Execute Movement - the *First Movement* from *Movement Buffer* is loaded to be executed in decision time frame $[t_{i+1}, t_{i+2}]$.

15th Finish Avoidance Run - if the *UAS* is flying, continue with 1st step.

Decision Frame: The *mission control run* (fig. 6.22) describes the overall process in sequence. The *orchestration overview* is given in (fig. 6.23).

The key idea is to explain what happens in one *decision frame*. The *mission control run* is implemented as multi-thread application which sends the signals between threads. Each thread is the semi-independent process with forced synchronization on *decision frame switch*.

The notable threads and their roles & responsibilities are summarized like follow:

1. *Sensor Fusion* - responsible for processing real-time sensor array (sec. 4.1.8). The output is a partial known world assessment (sec. 4.1.10). *Obstacle detection* and *intruder detection* events can be risen by this thread.
2. *Data Fusion* - responsible for enhancing data from *sensor fusion* by mixing data originating from *information sources* (sec. 4.1.9). The information sources used in this work contains constraints originating from *geo-fencing*, *weather*, *airspace restrictions*. This thread is delayed by *sensor fusion*. A *data fusion procedure* strongly depends on the *operational space context* (controlled/non-controlled airspace). The output of *data fusion* is full *known world assessment* (sec. 4.1.10, 6.5.4). The *UTM-related* and *constraint related* events can arise from *data fusion*.

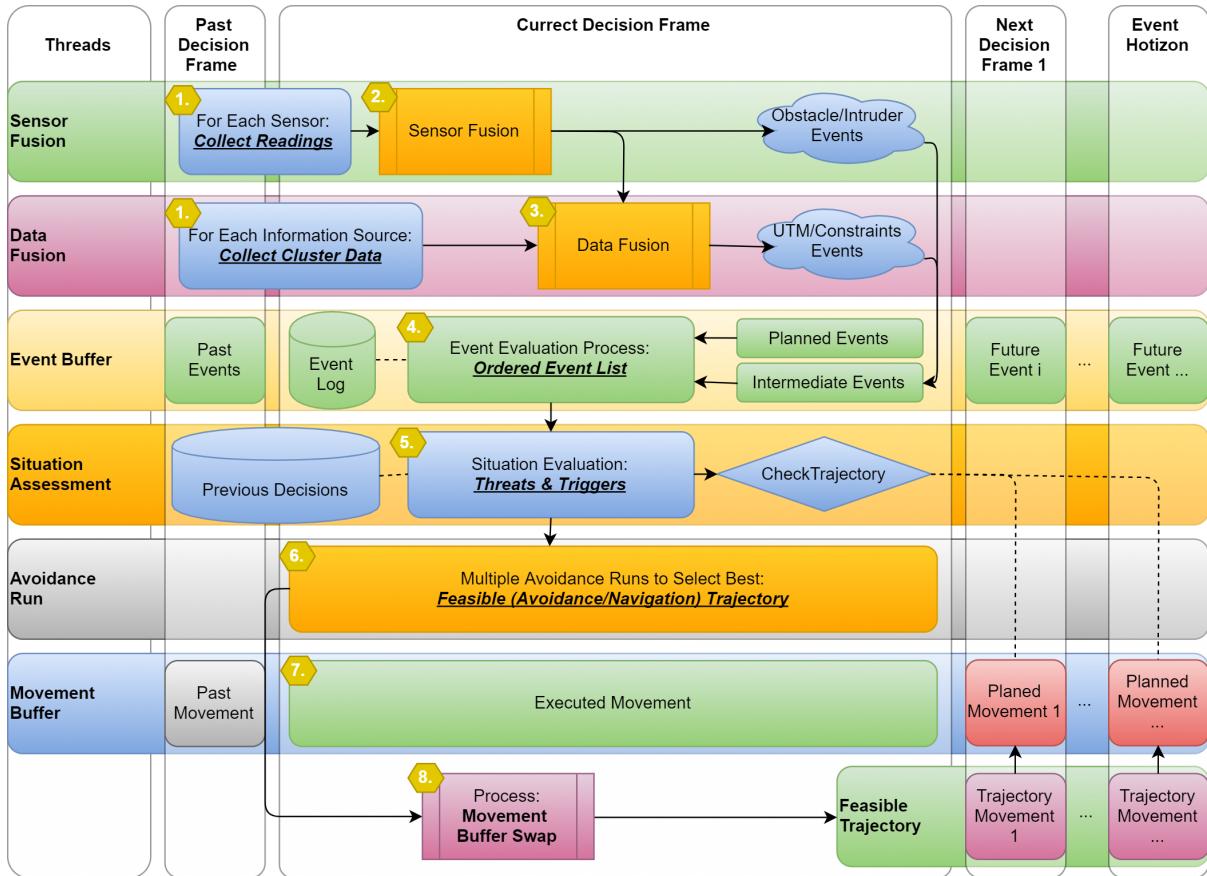


Figure 6.23: Mission control orchestration diagram.

3. *Event Buffer* - special data structure to store, raise, handle, prioritize events raised by other threads.

The *implemented events* are listed in the 5th-6th step of *mission control run*. The events can be categorized like follow:

- Planned events* - raised in previous decision frames to be executed in actual or future *decision frame*.
- Intermediate events* - raised in *actual decision frame* by other threads to be solved intermediate.

The event buffer thread executes following event-related activities:

- Storing* - the *events* are stored in the *event log*. The trace is useful for process and rules fine-tuning.
- Raising* - the combination of events (multiple avoidance events) (example sec. 7.4.3) can trigger additional avoidance behavior in the form of combined-event.
- Handling* - the events are handled by invoking the *situation assessment* or by rule engine invocation (sec. 6.8.1).
- Prioritizing* - the multiple events can rise during one *decision frame*. Some events cannot be merged and need to have proper prioritization before handling, like the *obstacle detection* events before *intruder detection event*.

4. *Situation Assessment* - invoked by *event buffer* to assess the situation, responsible for proper *avoidance run* (sec. 6.6.1) dataset preparation and invocation. The main responsibility is to check *planned trajectory feasibility* stored in *movement buffer* as *planned movements*.
5. *Avoidance Run* - invoked by *necessity to plan trajectory* originating from *event buffer* or *situation assessment* threads. The avoidance run produces one or multiple *avoidance/navigation* feasible trajectories according to the 7th-11th step of *mission control run*.
6. *Movement Buffer* - represents *movement automaton implementation* (sec. 6.2.3). The movement automaton consumes *movement automaton buffer* each decision frame contains exactly one *movement*. The movements can be viewed as:
 - a. *Past movements* - already executed movements in *past decision frames*.
 - b. *Executed movement* - actually executed movement in the current decision frame, this movement cannot be changed.
 - c. *Future movements* - future planned movements to be executed after *current decision frame* expires. These movements outline planned trajectory (predictor mode sec. B.4).
7. *Feasible Trajectory* - consists of *future planned movements* taking place directly after the *correct decision frame*. If its necessary, the planned trajectory in movement buffer is no longer feasible, the planned movements will throw away and replaced by *trajectory movements*.

The *roles & responsibilities* of each thread have been explained to outline their orchestration and roles in *mission control run* (fig. 6.22). The numbered steps in (fig. 6.23) shows the threads orchestration in the following manner:

1. *Sensor & Data fusion data set preparation/collection* - the sensor readings are collected through multiple past and over current *decision frame*. Each sensor reading is filtered and processed according to best practices.
The raw information from various data sources is loaded for relevant space clusters. The relevant space clusters are determined based on *UAS expected position*.
2. *Sensor fusion* - the readings from sensors are preprocessed according to (sec. 6.5.1, 6.5.2).
3. *Data fusion* - the information sources are preprocessed according to (sec. 6.5.1, 6.5.2).
4. *Event evaluation process* - the events are evaluated, if there is any triggering event (5th-6th mission control run steps) the situation evaluation process is called.

5. *Situation evaluation process* - the situation is evaluated according to 5th-6th mission control run steps.
6. *Feasible trajectory selection process* - from collected *navigation/avoidance trajectories* (7th-10th mission control run steps). If there are more feasible trajectories (increasing threat) the one compliant with most of the threats is selected.
7. *Movement execution* - the movement for the *current decision frame* is being executed.
8. *Movement buffer swap* - if there is a new *feasible trajectory* the future movements for next decision frames are flushed away. The movement buffer is then filled with *feasible trajectory movements*.

Note. This step impacts the duration of future *decision frames*.

6.6.3 Computational Complexity

Summary: Brief approach computational complexity analysis considering navigation/control/data fusion in support of real-time application feasibility.

Introduction: The *Computational Complexity* one mission control run assessment is necessary to identify the strong and weak points of approach. Let us get through modules to assess notable calculations/algorithms complexity on high abstraction level.

Navigation Loop: On the navigation loop, the *waypoint reach condition* (eq. 6.105) is checked, this is a unitary operation with worst complexity $\mathcal{O}(1)$. The selection process of the next *Goal Waypoint* can get through all waypoints in the mission if they are all unreachable the complexity is $\mathcal{O}(|\text{waypoints}|)$.

The *notable steps* complexity is following:

Reach Condition: $\mathcal{O}(1)$

Select Next Waypoint: $\mathcal{O}(|\text{waypoints}|)$

Data Fusion: The *data fusion* is all about *threat selection*.

If *UAS* is in *controlled airspace*, it needs to iterate over received *collision Cases* to select *active ones*. The complexity of this step is linear; therefore boundary is given as $\mathcal{O}(|\text{collisionCases}|)$.

Thresholding *Detected Obstacles* is done by simple comparison of *LiDAR ray hits* in given $\text{cell}_{i,j,k}$ of *Avoidance Grid*.

Any loading of *threats* from *information sources* depends on clustering. The *Airspace Clustering* is considered as static for our setup. Therefore the *count of active airspace clusters* has the main impact on complexity. The *count of information sources* is static and not changing over mission time. Information sources usually implement *Hash search function* with complexity $\mathcal{O} \ln |\text{searchedItemSet}|$.

The *computational complexity* boundaries for *Data fusion* in our setup are following:

Select Active Collision Cases: $\mathcal{O}(|\text{collisionCases}|)$

Threshold Detected Obstacles: $\mathcal{O}(|\text{cells}|)$

Load Map Obstacles: $\mathcal{O}(\ln |\text{activeClusters}| \times |\text{informationSources}|)$

Load Hard Constraints: $\mathcal{O}(\ln |\text{activeClusters}| \times |\text{informationSources}|)$

Load Soft Constraints: $\mathcal{O}(\ln |\text{activeClusters}| \times |\text{informationSources}|)$

Note. The *real-time clustering* is a *hard non-polynomial problem* [168]. Usually, all information sources and sensor have *polynomial complexity* of processing. The *controlled airspace clusters* are usually set for a very long period. Therefore *Obstacle Map*, *Airspace Constraints*, and, *Weather Constraints* can be considered as preprocessed

Situation Assessment: The *Situation Assessment* is evaluating triggering events. The *evaluation* is usually simple existence question without further calculations. The *complexity* of *event evaluation* for our case is $\mathcal{O}(1)$. There are *eight* triggers. The count of *triggers* needs to be accounted in complexity boundary:

$$\mathcal{O}(|triggers| \times eventEvaluationComplexity)$$

Note. The *trigger calculation complexity* needs to stay low because the *triggers* are verified every *Mission Control Run*. The *Avoidance Run* trigger frequency should be very low under normal conditions.

Avoidance Run: The *Avoidance run* is the most critical part of *Mission Control Run* because of *Avoidance Path* calculation. The *Navigation Path* calculation is less complex (Rule engine is not accounted); therefore *Emergency Avoidance Mode* is assumed.

The *threat insertion* is realized in 7th to the 10th step. The first is *Avoidance Grid* filled with *Static Obstacles*. The *Avoidance Grid* is designed to separate rotary *LiDAR* ray space into hit count even cells. Insertion of *LiDAR* scan into *Avoidance Grid* complexity depends on *total cell count*. The *upper boundary* for *insert obstacles* is given like follow:

$$\text{Insert Obstacles: } \mathcal{O}(|cells|)$$

The *intruders intersection model* type impact the insertion complexity. The *linear intersection* (app. C.1) is going through the maximum of *layers count* cells.

The *body volume intersection model* (app. C.2) can check the *simple intersection condition* overall *Avoidance Grid* in the worst case; therefore complexity for this check is bounded by a *count of cells*.

The *Maneuverability Uncertainty Intersection* (app. C.3) can hit all cells in *Avoidance Grid*. The calculation complexity boundary is exponential depending on the *horizontal/vertical spread* in [rad]. The *intersection* implementation was done *ad-hoc*. The impact of *intersection application* is visible only when there are more than *four* concurrency intruders (fig. 7.28).

The *complexity boundary for intruder insertion* is given like follow:

$$\text{Insert Intruders: } \mathcal{O} \left(\sum \begin{bmatrix} |linearIntersections| \times |layers| \\ |bodyvolumeIntersections| \times |cells| \\ |cells|^{horizontalSpread \times verticalSpread} \end{bmatrix} \right)$$

Note. The *intruder intersection* is critical in *non-controlled airspace*. The main complexity gain in *controlled airspace* is from *rule application*. Our *rule complexity* is in the worst case depending on *Reach Set node count*, and *Active Collision Cases count*.

$$\text{Apply Our Rules: } \mathcal{O}(|activeCollisionCases| \times |nodes|)$$

For *Hard/Soft Constraints* The algorithm used for intersection polygons was selected based on a study [160], the selected algorithm *Shamos-Hoey* [161]. The *calculation complexity* boundary is given like follow:

Hard Constraints Intersection:

$$\mathcal{O}(|cells| \times |hardConstraints| \times \max |constraintPoints|^2)$$

Soft Constraints Intersection:

$$\mathcal{O}(|cells| \times |softConstraints| \times \max |constraintPoints|^2)$$

Each *threat* category application in *Mission Control Run* is done after *each intersection* in 7th to the 10th step. All ratings (tab. 6.3) expect *Reachability*(cell_{ij,k}) and *Reachability(Trajectory)* are calculated. The *calculation complexity* boundary for one *reachability rating* is $\mathcal{O}(1)$. (eq. 6.91, 6.92). The *Recalculate Reachability* operation applied 4× have maximal *complexity* boundary given as follow:

$$\text{Recalculate Reachability: } \mathcal{O}(4 \times (|nodes| + |cells|))$$

Each time at the end of in 7th to the 10th step the *Avoidance Path is Selected*. The *Worst Case* (expected) scenario is to *select* four paths for each *threat* application. The algorithm for *best path selection* (alg. 6.6) iterates overall *cells* in avoidance grid and over all *trajectories* passing through that cell. The complexity boundary for *path selection* is given as follow:

$$\text{Select Path: } \mathcal{O}\left(4 \times \left(|cells| + \frac{|nodes|}{|cells|}\right)\right)$$

Conclusion: Overall approach complexity is *low*. If proper *Information Sources* with efficient clustering and *intersection models for intruders* are used, the approach will stay within *non-polynomial complexity*. The average load time for *testing scenarios* is summarized in (tab. 7.53).

Note. The calculation of *Reach Set* is eliminated by pre-calculation for *state range* [9].

6.7 UTM Prototype Implementation

Summary: The UAS system is already equipped to fend the treat itself. The practical applications require some degree of cooperation with authority (UTM). The requirements for UTM supervised operations are outlined in (sec 2.5). First, the interaction architecture is established. The notable maneuvers and situations are analyzed under VFR/IFR conditions. The position notification message and handling are proposed to support collision case calculations and life-cycle management.

Introduction: The *Traffic Management* for UAS is based on existing Air Traffic Management System for manned aviation [3]. The controlled airspace segments are *static* and have one *authority for one zone* principle. The dynamic zones have been proposed in [41]. However, it will be omitted for *simplification purpose*. The necessity for *UAS integration* into *National Airspace* has been outlined in [62].

The latest *Airbus blueprint* [2] outlines some functionality. The main purpose of this section is to show *Reach Set based Approach* capability to follow *Usual Air Traffic Management* commands.

The section is organized to introduce:

1. *UTM Architecture* (sec. 6.7.1) - centralized ATM-like authority over airspace cluster.
2. *Handling Standard Collision Situations* - head-on approach (sec. 6.7.2), converging situation (sec. 6.7.3), overtaking (sec. 6.7.4).
3. *Position Notification* (sec. 6.7.5) - position notification design.
4. *Collision Case* (sec. 6.7.6) - calculation and handling of *collision situations*.

The additional material can be found in:

1. *Cooperative Conflict Resolution* (app. D.1) - the model used for conflict resolution in *controlled airspace*.
2. *Non-Cooperative Conflict Resolution* (app. D.2) - the model used for conflict resolution in *non-controlled airspace* and *emergency avoidance*.
3. *Weather Case* (app. E.1) - definition and handling of *weather hazards*.

6.7.1 UTM Architecture

Summary: The UTM authority needs to communicate with the UAS attendants. The communication scheme is asynchronous notification(UAS)-directive(UTM).

UTM Concept is based on *asynchronous event-based control* [169]. *Event in controlled airspace* is handled in the form of *cases* [170]. There are following *event sources*:

1. *Weather Information Service* (from [171]) - used to create *weather case* (tab. E.1).
2. *Position Notification from UAS systems* (tab. 6.4) - used to create *collision cases* (new functionality) (tab. 6.6).

Decision Frame (eq. 6.108). The *UTM* is operating in discrete decision frames which are starting on current *decision time* and ending at next *decision time*:

$$\text{decisionFrame}_i = [\text{decisionTime}_i, \text{decisionTime}_{i+1}[, \quad i \in 1, \dots, k, k \in \mathbb{N}^+ \quad (6.108)$$

Event-based Airspace Control is collecting events in previous $\text{decisionFrame}_{i-1}$ and issuing commands in current decisionFrame_i . There are following phases during the *UTM frame* cycle:

1. *Planning* - the detection phase, when the hazardous situations are assessed.
2. *Fulfillment* - the monitoring phase, controlled UAS systems fulfill the state of affairs for directives and mandates.
3. *Acknowledgment* - the closing phase, when UTM assess and acknowledges the performance of controlled UAS systems.

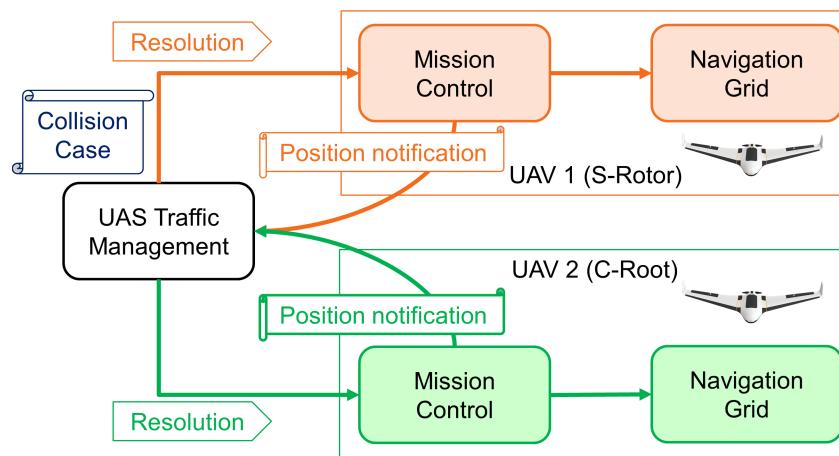


Figure 6.24: UAS Traffic Management (UTM) architecture overview.

Architecture (fig. 6.24). There are multiple UAS systems equipped with standard *Mission Control* and *Navigation* procedures.

Depending on the *airspace cluster* decision time frame they are sending *periodical position notifications* (tab. 6.4).

The *UAS Traffic Management* (UTM) collects the event data from *Weather Information Service* and *Position Notifications* calculating respective *cases*.

If there is an *active collision/weather case*, the *UTM* will send *resolutions* to respective airspace attendants.

6.7.2 Handling Head-on Approach

Summary: Two UAS are facing each other head-on. There is a need to define triggers for detection and resolution approach for autonomous UAS. Rules for VFR/IFR modes in manned aviation are the base for the autonomous collision resolution. The concept of the virtual roundabout is introduced.

Goal: Identify required parameters sufficient for automatic solution of *Head-on collision* situation.

VFR: The *Visual Flight Rules* (VFR) are specified in annex 2 [4], and there is a *Head-on* approach for two or more air crafts. The definition is rather vague: "The pilot should diverge from original heading to the right to create sufficient, safe space for avoidance."

IFR: The *Instrument Flight Rules* in annex 2. [4] and 11. [5] are defining the boundaries and events for success full *Head-on resolution* in larger detail.

The parameter values are useless due to the UAS scaling factor; the following parameters can be used in UTM:

1. The *angle of approach* $\geq 130^\circ$ - the minimal planar angle between aircraft positions and expected collision point is in the interval $[130^\circ, 180^\circ]$.
2. *Minimal detection range* - the minimal detection range of head-on collision is $2 \times \text{turningRadius} + \text{safetyMargin}$.
3. *Safety margin* - during avoidance all aircraft keeps mutual distance at least the value of safety margin.

Triggering Events: The *head-on approach* (fig. 6.25) *triggering events* are the following:

1. *Detection* (fig. 6.25a) - the *collision case* is open when *collision point* with the respective angle of approach is detected. This must happen until the *point of no return* is achieved.

2. *Resolution* (fig. 6.25b) - the *virtual roundabout* is enforced until the closing condition is met.
3. *Closing* (fig. 6.25b) - based on the condition that all vehicles are heading away from *collision point* and their mutual heading is neutral or opposite.

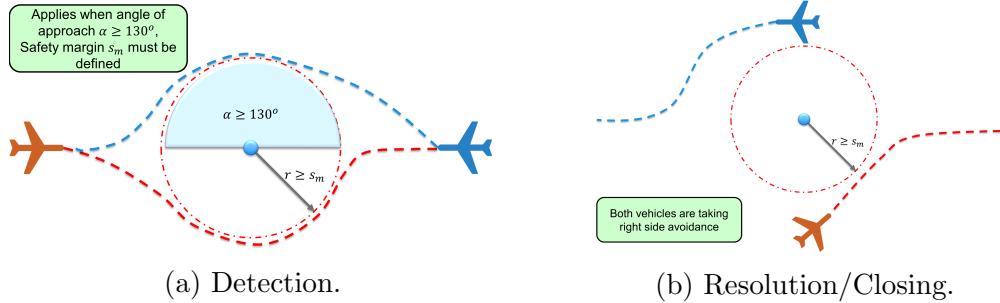


Figure 6.25: Head-on approach detection/resolution/Closing

Virtual roundabout: The *flight levels* can be abstracted as the *virtual 2D surface*. The *airspace attendants* are moving on virtual routes which can cross each other. The idea is to create virtual roundabout with enforced velocity to enable smooth collision avoidance.

1. *Center* - the center defined in *airspace cluster* local coordinate system (flight level defining the horizontal placement).
2. *Diameter* - the minimal distance to *center*, accounting the *wake turbulence* and other phenomena.
3. *Enforced velocity* - all attendants at *virtual roundabout* keeps the same velocity. It helps to keep constant mutual distances.

6.7.3 Handling Converging Maneuver

Summary: Two planned trajectories of the UAS are perpendicular, thus resulting in a potential collision. There is a need to define triggers for detection and resolution approach for autonomous UAS. Rules for VFR/IFR modes in manned aviation are the base for the autonomous collision resolution.

Goal: Identify *required parameters* sufficient for automatic solution of *Converging Maneuver*.

VFR: The *Visual Flight Rules* (VFR) are specified in annex 2 [4]. The rule is different from *Head-on Approach* (sec. 6.7.2) because multiple roles are depending on the relative aircraft position:

1. *Avoiding Aircraft* - there is an aircraft on the relative right side (blue).

2. *Right Of the Way (ROA) Aircraft* - there is an aircraft on the relative left side (red).

The *avoiding aircraft* should take the *right of the way aircraft* from behind, with sufficient *safety margin*, and return to original *heading* afterward. The *magnitude of avoidance curve* must consider *wake turbulence* and other impacts of *avionic properties*.

Note. This rule is applied only when both *aircraft* belong to the same *maneuverability class* [4].

IFR: The *Instrument Flight Rules* in annex 2. [4] and 11. [5] are defining *converging maneuver* in detail.

The *parameters* from a *head-on approach* can be reused:

1. $70^\circ \leq \text{the Angle of Approach} < 130^\circ$ - the minimal planar angle between aircraft position and expected collision point is in the interval $[70^\circ, 130^\circ]$.
2. *Minimal detection range* - given as *turningRadius + safetyMargin*, while *safety margin* is accounting all impact factors.
3. *Safety margin* - during avoidance all aircraft keeps mutual distance at least on the value of *Safety Margin*.

Note. The lesser *angle of approach* induces stronger wake turbulence impact on avoiding aircraft. This results in an increase of *safety margin*.

The *wake turbulence* is represented as a droplet at the back of the plane. *Wake turbulence range* can be calculated based on wake turbulence cone.

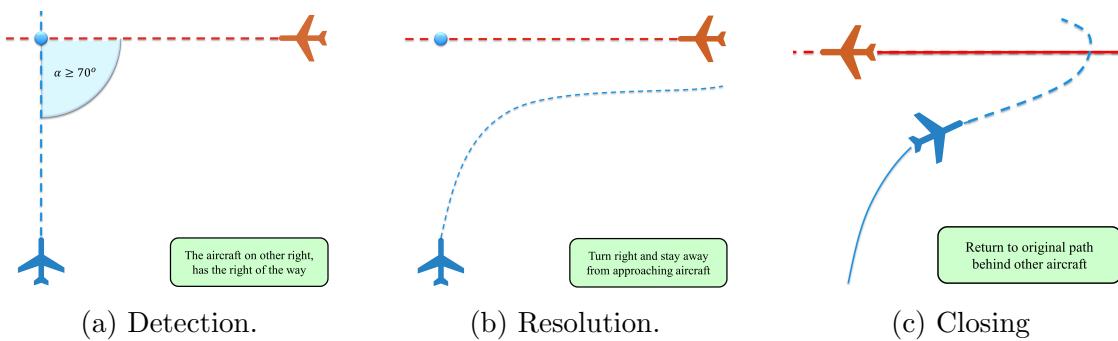


Figure 6.26: Converging maneuver Detection/Resolution/Closing

Triggering Events: The *converging maneuver* (fig. 6.26) *triggering events* are the following:

1. *Detection* (fig. 6.26a) - The *avoiding airplane* (blue) detects *collision point* (blue circle) which satisfy the *converging maneuver conditions*. The distance between *aircraft position* and *collision point* is lesser than the *detection range*.

2. *Resolution* (fig. 6.26b) - the *Right Of the Way* aircraft (red) stays at the original course. The *avoiding aircraft* (blue) follows the *parallel* to another *plane*. The distance of *avoiding plane* to *other plane trajectory* is greater or equal to *safety margin*.
3. *Closing* (fig. 6.26c) - when both planes have an opposite heading, and they miss each other the converging maneuver can be closed. The *avoiding airplane* will return to *original trajectory* while keeping the distance from *another plane* (red) at greater or equal to *safety margin*.

6.7.4 Handling Overtake Maneuver

Summary: Two UAS are on the same airway, flying in the same direction. The slower UAS is in front of the faster UAS. The slower UAS has the right of way, and the faster UAS needs to make an overtake. There is a need to define triggers for detection and resolution approach for autonomous UAS. Rules for VFR/IFR modes in manned aviation are the base for the autonomous collision resolution.

Goal: Identify *required parameters* sufficient for automatic solution of *Overtake Maneuver*

VFR: The *Visual Flight Rules* (VFR) are specified in annex 2 [4]. The rule states that faster air traffic attendant may overtake slower one, from right side keeping sufficient distance (*safety margin*). There are two forced roles:

1. *Overtaking* - faster aircraft with similar heading cruising in similar altitude than *overtaken* (blue). It is expected that *faster aircraft* has maneuvering capability to avoid slower aircraft.
2. *Overtaken* - slower aircraft which keeps the *Right of the way*

Note. This rule is applied only when both aircraft have the same maneuverability class [4]. The overtake is considered *borderline emergency maneuver* in controlled airspace because the aircraft tend to keep similar velocity in similar cruising altitude. The overtake is usual in *non-controlled airspace*.

IFR: The *Instrument Flight Rules* in annex 2. [4] and 11. [5] are defining the converging manual in detail:

1. $0^\circ \leq \text{the Angle of Approach} < 130^\circ$ - the minimal planar angle between aircraft position and expected collision point is in the interval $[0^\circ, 70^\circ[$
2. *Minimal Detection Range* - given as $2 \times \text{reactionTime} \times \text{speedDifference}$.

3. *Safety Margin* - during avoidance the overtaking aircraft keeps the minimal distance of *wake turbulence* of overtaken aircraft in own flight altitude.

Note. The *Safety Margin* is sufficiently small because speed difference is usually much lesser than in case of *Head-on approach*. The *Wake turbulence* can be avoided completely by taking the higher altitude level than overtaken aircraft.

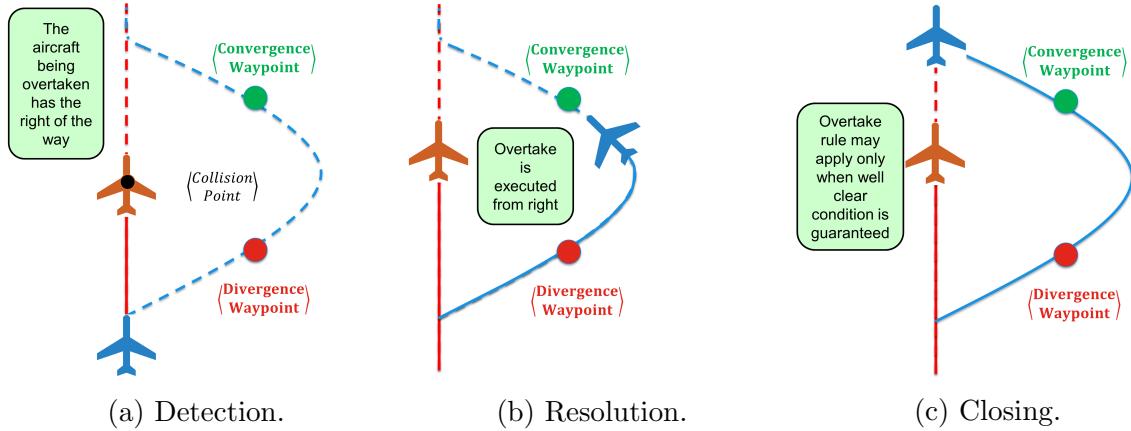


Figure 6.27: Overtake maneuver Detection/Resolution/Closing

Triggering events:

1. *Detection* (fig. 6.27a) - occurs when the distance between *overtaking* (blue) and overtaken (red) is approaching *minimal detection range* or double of *safety margin*. If the performance of *overtaking aircraft* (blue) allows taking *sharp right side to overtake* the *Maneuver starts*, otherwise *overtaking aircraft* (blue slows down) and keeps at least *safety margin distance* to avoid *wake turbulence*.
2. *Resolution* (fig. 6.27b) - *overtaken* (red) is keeping same heading and *speed* during overtake maneuver. The *overtaking* (blue) projects two waypoints: *Divergence* and *Convergence* keeping the required separation minimum during overtaking. Then the *overtaking* (blue) diverges heading to *Divergence waypoint*. When the *Divergence waypoint* is reached by *overtaking* (blue) aircraft, it changes to *original heading*.
3. *Closing* (fig. 6.27c) - the *closing* of *Overtake* starts when *overtaking* aircraft (blue) have sufficient lead over *overtaken* aircraft (red). The *overtaking* aircraft (blue) can safely change the heading to the original waypoint.

Constant Cruising Speed: Most of the traffic attendants at same flight level have similar (close to constant) cruising speed. Lower flight levels are for slower turbo-prop planes, and higher altitudes are for jet planes. It is stated that this principle will persist even when UAS will be integrated [172, 173, 174] in multiple air-traffic models.

6.7.5 Position Notification Implementation

Summary: There is a need to define a "minimal" data-set for UAS position notification. The base of such notification is the ADS-B message.

Motivation: The *position notification* (tab. 6.4) is designed for further *collision case resolution* (sec. 6.7.6). It is similar to ADS-B⁴ message information.

The main purpose is to broadcast the *position notification* in *controlled aerospace*. The broadcast for *non-controlled* airspace needs to contain *intruder properties, preferred separation mode* and *near-miss margin*.

Position: The position is defined in *Global Coordinate System* using GPS for latitude and longitude. The barometric altitude is required for controlled airspace, preferred for non-controlled airspace.

Heading: The *Linear Velocity* combined with heading in standard *North-East* coordinate frame is used.

Flight Levels: The *flight level* is notified to UTM for *collision detection* purposes. There is a *main flight level* where *aircraft* belong physically. There is a *passing flight level* form which/to which is aircraft emerging [3].

Aircraft Category: The aircraft category impacts the prioritization of *role assessment* by UTM/ATM. The following categorization is proposed by *manned aviation pilot community*, from the highest to the lowest right of the way priority:

1. *Manned aviation in distress* [4] - the aircraft with impaired capability switched to emergency mode. The emergency mode is usually acknowledged by the authority in controlled airspace.
2. *Balloon* (manned) [4] - the aircraft with *altitude* control and very slow dynamics implying very low maneuverability.
3. *Glider* (manned) [4] - the aircraft with *full control* but without own *propulsion*. The overall *maneuverability* is good, but the *velocity* changes are impossible with sufficient flexibility.
4. *Aerial towing* (manned) [4] - the towing aircraft usually have *own propulsion* and full maneuverability, the only constraint is *towed load*. The towed load decreases overall maneuverability.
5. *Airship* (manned) [4] - the airship have *own propulsion* and full maneuverability, the constraint is low acceleration/deceleration and huge turning radius.

⁴ADS-B versions and message containment: <https://mode-s.org/decode/adsb/version.html>.

6. *Other manned aviation* [4] - containing all vehicles with the required level of *airworthiness* for given operational *altitude*. They usually have required maneuverability.
7. *UAS Autonomous* (proposed) [175] - containing all autonomous UAS, the lower flexibility is expected at the beginning of integration.
8. *Remotely Piloted Aerial System (RPAS)* (proposed) [175] - has lesser priority due to the higher response rate of the pilot.

Note. This categorization reflects only Pilot community statement; the general priority rule is broken, because maneuverability and vulnerability should always be considered as a key decision factor.

Maneuverability: The maneuverability is the real key factor in priority assessment. The components of maneuverability are *maximal/mean acceleration/deceleration*, *climb/descent rate* and *turning ratio/radius*. The comparison can be made by solving *pursuit problem* using *Reach Sets* [69, 70].

The *Maneuverability categorization* is based on *original aircraft priority categorization* [4] accounting UAS/RPAS as equal to *manned aviation*. The ordered list from the highest to the lowest priority goes as follows:

1. *Impaired control* (Distress aircraft) - any aviation attendant in distress has the priority in case of the conflict occurrence.
2. *Altitude control/No* (Balloon, Hovering aircraft) - the balloon type crafts do not have any type of propulsion, and horizontal movements follow the airflow in given altitude.
3. *Full control/No propulsion* (Gliders of any sort) - the gliders can control their horizontal position, but there are limits to altitude control and acceleration/deceleration.
4. *Full control/Linear propulsion* (Any aircraft of plane type) - the *towing aircraft's* and *airplanes* belong there; the difference is the *flexibility of maneuvering*.
5. *Full control/VTOL capability* (Any aircraft with VTOL) - the *other aircraft* capable of doing on-spot-turn. The typical representative is *quad-rotor copter*.

| Position | |
|-----------------------|--|
| latitude | based on GPS/IMU sensor fusion. |
| longitude | based on GPS/IMU sensor fusion. |
| altitude | barometric altitude <i>Above Mean Sea Level</i> (AMSL). |
| Heading | |
| orientation | orientation in standard North-East coordinate frame. |
| velocity | relative UAS velocity. |
| Flight Levels | |
| main | flight level, where UAS mass center belongs |
| passing | flight level, during climb/ascend, or when distance of UAS mass center to flight level boundary $\leq 250ft$. |
| Registration | |
| registration ID | is unique registration number <i>to be issued</i> by local aviation authority for UTM communications purposes. |
| flight code | or mission code is a unique identification number for approved mission plan which is going to be flown by UAS. |
| UAS name | optional UAS identifier to increase human recognition. |
| Categorization | |
| craft category | ICAO main category, based on vehicle type. |
| maneuverability | secondary categorization is specifying size class, horizontal/vertical turning radius, minimal and maximal cruising speed. |
| Safety margins | |
| universal | minimal safety margin for any avoidance situation |
| head-on | minimal distance from other similar maneuverability class aircraft in case of a head-on approach. |
| converging | minimal distance from other similar maneuverability class aircraft in case of the converging maneuver. |
| overtake | minimal distance from other similar maneuverability class aircraft in case of overtake maneuver. |
| wake angle | for wake turbulence cone. |
| wake radius | for wake turbulence cone. |

Table 6.4: Time-stamped *position notification* structure.

There are other aspects like *minimal required* acceleration/deceleration/turn ratio to operate in a selected segment of the *airspace*. These should be specified later by *Minimum Operational Performance Standards* (MOPS).

Safety Margins: The *Safety Margin* for *Well Clear Condition* value is based on the *situation*. There is also a *universal safety margin* which guarantees the minimal safety for encountering intruder.

The most prevalent effect is *Wake turbulence*, therefore, *wake turbulence cone angle* [$0^\circ - 90^\circ$] and radius.

The *safety Margin* for situation-based avoidance is given by the list of supported maneuvers; there is converging (sec. 6.7.3), head-on (sec. 6.7.2), overtake (sec. 6.7.4) safety margins.

6.7.6 Collision Case Implementation

Summary: The UTM needs to detect and prevent possible collisions. The collision case is a record of such event detection, processing, and closure. Two detection methods are defined, one using linear intersection and other using planned trajectories intersection. The angle of approach and UAS relative speed determines the maneuver to be used in situation handling.

Collision Case Purpose: There is a need for detection and tracking of possible *controlled airspace traffic attendants* collisions. The presented *collision case structure* (tab. 6.6) is a minimalist reflection of *ATM* requirements. Following aspects of *collision case* life cycle are explained in this section:

1. *Base terminology* - the definition of *enforcement procedure* and difference between *Resolution* and *Mandate* from UTM authority. The *severity issue* is open.
2. *Calculation of single case for single decision frame* - step by step calculation and threat evaluation. Prequel to the *life-cycle*.
3. *Life cycle* gives outlook on how collision case data are handled through a longer period, notably: *Opening*, *collision point handling*, *safety margin handling*, and, *Closure*.
4. *Merge procedure for multiple cases in a single cluster* - the naive *merge procedure* to solve *multiple collision cases* via the *virtual roundabout*.

Resolution/Mandate Enforcement: *Enforcement procedure* is consisting from *Threat detection phase* and *Mitigation phase*. The *mitigation phase* is a time interval when *UTM* decision is enforced. The decision the UTM is enforcing is delivered in the form of *Resolutions* and *Mandates*.

A *Resolution* is an order from the *UTM* authority which is followed by subjected UAS. The *subjected UAS* can determine own behavior to some extent. When there is an emerging threat or another destructive event, like a new non-cooperative adversary, the UAS is allowed to break *resolution*.

A *Mandate* is an order from the *UTM* authority which cannot be broken at any cost. The example of the *mandate*: UAS is flying in the airspace, the passenger in distress needs it to safely land. The UAS must obey mandate even at the event of own destruction.

Threat Severity Evaluation: The threat severity evaluation is omitted partially, all threats are considered as equal. All commands from *UTM authority* will be considered as *resolutions*.

Calculation procedure: Collision case is calculated for two *Registered UAS systems* in *Unified UTM time-frame*. The *unified UTM time-frame* is a short period in future when the anticipated situations are predicted.

1st The *position* and *orientation* are adjusted according to the *mission plan*. Our implementation uses *Movement Automaton* as a predictor:

$$\begin{aligned} \text{adjustedPosition} &= \text{Position}(\text{Trajectory}(\text{notifiedState}, \text{futureMovements})) \\ \text{adjustedOrientation} &= \text{Orientation}(\text{Trajectory}(\text{notifiedState}, \text{futureMovements})) \end{aligned} \quad (6.109)$$

For other cases standard linear prediction can be used:

$$\begin{aligned} \text{adjustedPosition} &= \text{notificationPosition} \times \text{notificationVelocity} \times \text{timeDifference} \\ \text{adjustedOrientation} &= \text{notificationOrientation} \end{aligned} \quad (6.110)$$

2nd The *maneuverability*, *craft category*, *registration ID* are taken from *position notification*.

3rd *Collision case check procedure* goes like follows:

1. *Operation space checks* - the controlled airspace and flight level must match for proceeding.
2. *Maneuverability/Category check* - the maneuverability and UAS category must match. If there is mismatch, then the right of the way is forced to the vehicle with higher priority.

4th *Linear Intersection test* is designed to calculate *closest distance* and *time of linear trajectory projections*. First, for given *velocity* and *position* for UAS1 and UAS2 the helper variables are calculated:

$$\begin{aligned} A &= \|velocity_1\|^2 \\ B &= 2 * (velocity_1^T \times position_1 - velocity_2^T \times position_2) \\ C &= 2 \times velocity_1^T * velocity_2 \\ D &= 2 * (velocity_2^T \times position_2 - velocity_2' \times position_1); \\ E &= \|velocity_2\|^2; \\ F &= \|position_1\|^2 + \|position_2\|^2; \end{aligned} \quad (6.111)$$

Then the projection parameters can be calculated:

$$\begin{aligned} time &= \frac{-B - D}{2 \times A - 2 \times C + 2 \times E} \\ destination_i &= position_i + velocity_i \times time, \quad i \in \{1, 2\} \\ collisionPoint &= \frac{destination_1 + destination_2}{2} \\ collisionDistance &= \|destination_1 - destination_2\| \end{aligned} \quad (6.112)$$

If $time < 0$ the trajectories are diverging from each other (because the closest points already occurred). The procedure ends, the *collision flag* is not raised.

If $time > timeMargin$ the trajectories will get close to each other, but in further future and changes are anticipated. The procedure ends, the *collision flag* is not raised.

If $0 \leq time \leq timeMargin$ the trajectories are converging to each other and distance needs to be checked. If $distance \leq collisionMargin$ then *collision flag* is raised and *collision point* is set.

Note. *Collision Margin* is some number which is determined based on aircraft category and maneuverability. Our work defines collision margin as follow:

$$collisionMargin = \forall situation : \max \left\{ \begin{array}{l} safetyMargin(situation, UAS1) \\ + safetyMargin(situation, UAS2) \end{array} \right\} \quad (6.113)$$

Where the *safety margin* for every possible situation is evaluated for both *UAS*.

5th The *trajectory intersection* is *Movement Automaton* specific collision detection method. Its based on the assumption that *UTM* has the following information from *mission plan*:

1. *UAS state* - not only *position*, *orientation*, and, *velocity* vectors, but other mathematical model parameters mandatory for *movement automaton*.
2. *Movement Automaton* - movement automaton for our *UAS* system, so that *UTM* can use it in predictor mode.
3. *Future Movements set* - up to reasonable prediction horizon *timeMargin*.

The *Movement Automaton* can be used as trajectory prediction for initial system state and future movements. The prediction function (eq. 6.114).

$$Prediction : UAS \times state \times futureMovements \rightarrow [x, y, z, t] \in \mathbb{R}^4 \quad (6.114)$$

Note. Then prediction for *UAS1* is *Prediction*₁, and for *UAS 2* *Prediction*₂, the predictions are synchronized meaning that time at position *i* is equal in both discrete trajectory matrices.

The *collision distance* for predictor (eq. 6.114) is given as minimal distance of projected synchronized trajectories for UAS1 and UAS2. In our discrete environment, the *collision distance* is given as (eq. 6.115).

$$\text{collisionDistance} = \min \left\{ \| \text{point}_1 - \text{point}_2 \| : \forall \begin{cases} \text{point}_1 \in \text{Prediction}_1, \\ \text{point}_2 \in \text{Prediction}_2, \\ t_1 \sim t_2 \end{cases} \right\} \quad (6.115)$$

If $\text{collisionDistance} \leq \text{collisionMargin}$ condition is met, *collision flag* is set.

The collision point is then calculated as mean of *UAS positions* in prediction at a time when the distance is minimal. The final collision point is arithmetic mean of two positions (eq. 6.116).

$$\text{collisionPoint} = \frac{\text{point}_1 - \text{point}_2}{2} : \begin{pmatrix} \text{point}_1 \in \text{Prediction}_1, \\ \text{point}_2 \in \text{Prediction}_2, \\ t_1 \sim t_2 \text{ at minimal distance} \end{pmatrix} \quad (6.116)$$

Note. Collision point is overwritten by trajectory intersection (specific) method; the *linear intersection* is considered a *general collision detection method*. The collision detection method in future UTM system needs to be determined. The *Trajectory intersection* method presented in this work is one of the possible candidates.

6th *Role determination* phase is invoked if and only if previous conditions are met and *collision flag* with *collision point* exists.

There is *adjusted position* of each UAS used as verticals and *collision point* used as a center. The first step is normalization of adjusted position around collision point for both UAS:

$$\text{normalized}_i = \text{adjustedPosition}_i - \text{collisionPoint}, \quad i \in \{1, 2\} \quad (6.117)$$

Then the right-hand coordinate system internal angle calculation method is used:

$$\text{angleOfApproach} = \left| \text{atan2} \begin{pmatrix} \text{normalized}_1 \times \text{normalized}_2, \\ \text{normalized}_1 \circ \text{normalized}_2 \end{pmatrix} \right| \quad (6.118)$$

Based on the *angle of approach* the *scenario type* is decided like follows:

1. $130^\circ \leq \text{angleOfApproach} \leq 180^\circ$ - the scenario type is set as *Head On Approach* (sec.6.7.2)
2. $70^\circ \leq \text{angleOfApproach} < 130^\circ$ - the scenario type is set as *Converging Maneuver* (sec.6.7.3)

3. $0^\circ \leq angleOfApproach < 70^\circ$ and *different speed* - - the scenario type is set as *Overtake Maneuver* (sec.6.7.4)

Based on *relative position* and *scenario type*, the *avoidance role* like follows:

1. *Head On Approach* enforces the following:
 - a. The *avoidance role* is set as *RoundAbounting* for both UAS.
 - b. None of the *UAS* does have the *Right Of the Way*.
2. *Converging Maneuver* enforces the following:
 - a. *UAS* without free right side has a role set as *Converging*.
 - b. *UAS* with free right side has the *Right Of the Way*.
3. *Overtake Maneuver* enforces the following:
 - a. *Slower UAS* has *Overtaken* role with *Right Of the Way*.
 - b. *Faster UAS* has *Overtaking* without *Right Of the Way*.
 - c. *Faster UAS* mission plan is altered with *divergence and convergence waypoints*.

7th *Safety Margin Calculation* Is invoked when the collision case is *Active*. The *Active Collision Case* in this time-frame means that *Collision Flag* is raised. The *avoidance role* determines *safety margin calculation*.

If *Head-On Approach* is case type of *Head collision case* then *safety margin* is calculated as the maximum of the sum of *default* margins or *head on* margins:

$$safetyMargin = \max \left\{ \begin{array}{l} default(UAS1) + default(UAS2), \\ headOn(UAS_1) + headOn(UAS_2) \end{array} \right\} \quad (6.119)$$

If *Converging Maneuver* is case type of *Head collision case* then *safety margin* is calculated based on *avoiding UAS* as the maximum of opposing UAS *default margin* and avoiding *converging margin*:

$$safetyMargin = \begin{cases} uas1.role = Converging : \max \left\{ \begin{array}{l} default(UAS2), \\ converging(UAS1) \end{array} \right\} \\ uas1.role = Converging : \max \left\{ \begin{array}{l} default(UAS1), \\ converging(UAS2) \end{array} \right\} \end{cases} \quad (6.120)$$

If *Overtake maneuver* is case type of *Head collision case* then *safety margin* is calculated as the maximum of *default*, *overtaking*, *overtaken* margins of both UAS:

$$safetyMargin = \max \left\{ \begin{array}{l} default(UAS1), default(UAS2), \\ overtaken(UAS_1), overtaking(UAS_2), \\ overtaking(UAS_1), overtaken(UAS_2) \end{array} \right\} \quad (6.121)$$

Collision Case Chaining is procedure when multiple active collision cases for different *time-frame* are chained and creates the time ordered series of *collision cases*. There are two notable instances in the *chain*:

1. *Head Collision Case* - Collision case when the first danger was detected. The notable parameters are *collision point* and UAS *avoidance roles* because these are enforced by the *Rule engine* (sec. 6.8). The *head collision case* is first in the chain.
2. *Tail Collision Case* - Collision case when the *collision danger* was not detected. The *tail collision case* is last in the chain.

Note. The *Chaining* of *collision cases* is rather primitive and sensitive for errors/noise.

The *Consistency of Avoidance Maneuver* is ensured by enforcing *head collision case* parameters.

| Data for both attendants | |
|--------------------------|--|
| adjusted position | predicted from previous <i>position notifications</i> (6.4) data at the time of <i>UTM decision frame</i> start. |
| adjusted orientation | predicted from previous <i>position notifications</i> (6.4), <i>mission plan</i> , and <i>expected velocity</i> . |
| velocity | proclaimed velocity for given <i>UTM decision time frame</i> . |
| registration ID | is unique registration number issued by the local aviation authority |
| craft category | from <i>position notifications</i> (6.4). |
| maneuverability | from <i>position notifications</i> (6.4). |
| mission plan | is acquired from <i>allowed mission registers</i> where it has been registered prior UAS flight |
| safety margins | list of all safety margins derived based on craft categorization or overridden by <i>position notifications</i> (6.4). |
| avoidance role | is given based on situation evaluation. |
| trajectory prediction | simulated based on <i>position notification</i> (6.4) and <i>mission plan</i> . |

Table 6.5: Collision case structure attendant data.

Collision Cases Merge also known as *Collision Point Adjustment Procedure* purpose it to *merge* multiple collision cases into one general collision case. The clustering is used to identify *airspace congestion events* [176]. Example of *airspace clustering* is given in [177].

The main idea is to *encapsulate multiple collision cases* into one virtual roundabout to ease *traffic load* [178]. The potential risk on *turbo roundabouts* have been outlined in [179].

There are *active collision cases* in a focused *cluster* in *controlled airspace*. The

multiple collision cases can pop up at different *start times*, and they can be active for a different *period*.

The *Collision point* is replaced with the *roundabout center* point (eq. 6.122). The *roundabout center* is calculated as weighted average of *active collision cases* collision points. The *weight* $\in [0, 1]$ depending on severity rating of collision case.

$$\text{roundaboutCenter} = \frac{\sum_{\substack{\forall \text{collisionCase} \\ \text{case} \in \text{Cluster}}} \text{collisionCase.collisionPoint} \times \text{weight}}{|\text{collisionCase} \in \text{Cluster}|} \quad (6.122)$$

Note. The weight in (eq. 6.122) is set to 1 for all time; the weight calculation needs to be determined in future works.

The *smallest circle problem* defined and solved in [55, 163] is used to determine the safety margin in our approach. The *naive approach* determining *roundabout safety margin* is to take the maximum of all open case *safety margins* including default ones (eq. 6.123).

$$\text{safetyMargin} = \max \left\{ \begin{array}{l} \text{case.UAS}_i.\text{roundaboutSafetyMargin}, \\ \text{case.UAS}_i.\text{defaultSafetyMargin} \end{array} \right\}, \quad \forall \text{case} \in \text{Cluster}, \quad \text{UAS}_i \in \{1, 2\} \quad (6.123)$$

Collision case calculated data

| | |
|-------------------------------|---|
| linear intersection | is predicted on attendants <i>position</i> , <i>heading</i> , <i>velocity</i> , based on <i>maneuverability</i> certain thresholds are applied to determine safety properties. |
| trajectory intersection | is predicted on attendants <i>position</i> , <i>velocity</i> , <i>heading</i> , and <i>related mission plans</i> , based on <i>maneuverability</i> certain thresholds are applied to determine safety properties. |
| collision point | is created if there is the risk of medium/short period collision, if head collision case has not been closed, collision point is inherited. |
| adj. collision point | is created if there exists at least one active collision case in the nearby surroundings of this case collision point (cluster). |
| angle of approach(α) | is calculated based on attendants <i>velocity</i> and <i>position</i> , the range is $[0^\circ, 180^\circ]$, it determines <i>primary avoidance roles</i> . |
| safety margin | is calculated based on <i>avoidance roles</i> , <i>maneuverability</i> , collision indicators, and <i>angle of approach</i> . |
| margin adjustment | is calculated based on <i>linked collision cases</i> , <i>estimation errors</i> and <i>weather</i> . |
| linked cases | contains a list of collision cases which are active and can have an impact on this <i>collision case</i> . |
| head case | is a reference to collision case in the time frame when it was first opened. |

Collision case indicators

| | |
|-------------------------|---|
| linear intersection | indicates if there was a safety breach on linear trajectories estimation with the risk of direct collision. |
| trajectory intersection | indicates if there was a breach on trajectory estimation, with the risk of direct collision. |
| well clear breach | indicates if <i>linear projection</i> or <i>trajectory projection</i> breaches <i>well clear barrel</i> in <i>controlled airspace</i> . |
| active case | indicates if the case is still open. |

Table 6.6: Collision case structure for given decision time-frame.

6.8 UTM Directives Framework Implementation on UAS

Summary: The standard framework implementation (sec. 6.6) needs to be enhanced for UTM directives following. The rule engine software architecture supports the addition and removal of rules and regulations .

Introduction: This section is follow up of *UTM functionality definition* (sec. 6.7), outlining realization of *UTM directives* on *UAS* side (sec. 6.8.1, 6.8.2).

Reasoning: The *Avoidance* process and *UTM directives fulfillment* are different in every national airspace. The ICAO issues recommendation [3, 4] which are implemented by every member country, some of the procedures are stricter some are implemented differently.

The *UTM* collision case calculation and procedures may be universal, but their realization by *UAS* will be heavily impacted by local legislation and procedures. The *approach* must account the need for *variable parts* of *obstacle avoidance process*. The *dynamic parts* need to be woven to hard-coded processes.

Note. Please refer to *Template Programming* and *Aspect Oriented Programming* for further explanation.

Inspiration: There was a *Maritime Rules* implementation [180] in the form of *Movement Restrictions* and *Waypoint Changes*.

6.8.1 Rule Engine Architecture

Summary: The implementation of the rule engine architecture in our framework environment.

Purpose: The *core process* of *Avoidance Grid Run* (sec. 6.6.1) and *Mission Control Run* (sec. 6.6.2) needs to be enhanced based on the situation. The architecture is based on *aspect-oriented approach* [181]. The key ideas and concepts are taken from rule engine implementation for multiagent navigation system [182].

Rule Engine: The program module to inject and run *rules* modifying standard workflow based on triggering events. The *aspect-oriented* approach enables to configure rules in *run-time* via predefined process hooks - *Decision Points*.

A rules the in context of this work are pieces of code which have a semi-static structure consisting of following parts (fig. 6.28):

1. *Decision Point* - hook point in the process where the rule can be attached/detached.
If more than one rule is hooked the priority of execution needs to be defined.

2. *Context* - the *run time context* in a time of *invocation* in our case the *copy* of *Mission Control, Avoidance/Navigation Grid* and, *Collision Cases*.
3. *Parser Method* - optional helper method to parse interesting data set from *Context*. The *parsed data* have better readability.
4. *Condition Check Method* - implementation of the trigger. If the sufficient condition is met, the rule body is applied.
5. *Rule application* - calculations and data structure changes. Mainly, by *disabling trajectories* in *Reach Set* in our implementation.

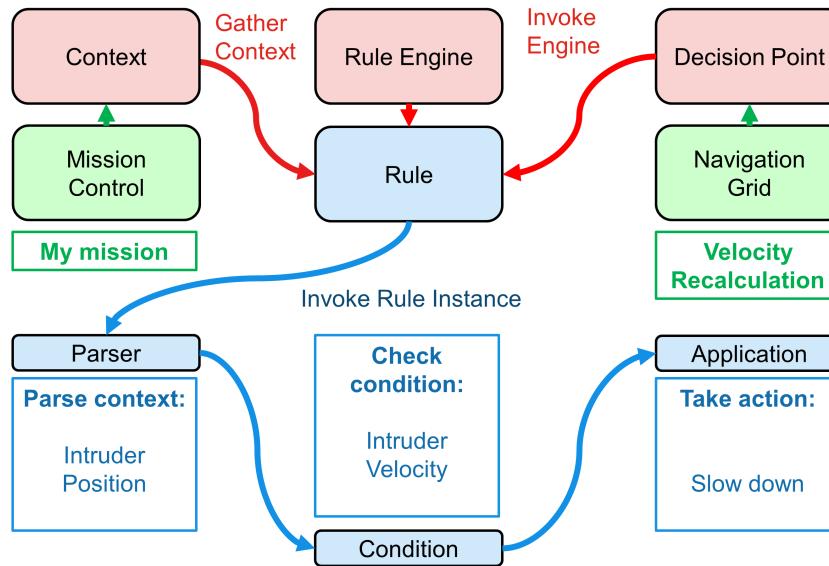


Figure 6.28: Rule engine components overview.

Example: The *UAS* is flying in controlled airspace. The *intruder* shows in front of *UAS*. The *UAS* is faster than an *intruder*. The *UAS* tries to obtain permission for *Overtake*. The *UTM* does not allow *overtake*, because of *insufficient UAS maneuverability capability*. The *Rule* (fig. 6.28) with:

1. *Context* - UAS Mission Control, containing the actual mission goal and UAS IMU parameters.
2. *Decision Point* (Joint Point) - Navigation grid, containing projected constraints and reach set approximation.
3. *The rule is invoked:*
 - a. *The parser* parses the context which is *intruder's Position Notification* containing its heading and velocity.
 - b. *The condition* is checked to *relative intruder velocity*. The *evaluation* is positiv, when the *UAS* is *pursuing the intruder*.
 - c. *Application* of *Rule* is the last step, in this case, the *UAS* will slow down.

Configurability: The *Rule Engine* enables real-time configuration. The *Enabled Rules Table* have been implemented to enforce specific rules in a specific context.

The *Rules* can be invoked from *Rule Application*; this enables effective rule chaining and piece-wise functionality split.

6.8.2 Rule Engine Setup

Summary: The setup to cover collision case resolution according to (sec. 6.7.6).

Configuration: The *Rule Engine Architecture* (fig. 6.28) is configured to handle *UTM* functionality for *Collision Case Resolution* (sec. 6.7.6). The overview of *Context* (Green), *Decision Points* (red) and *Rules to be Invoked* (cyan) is given in (fig. 6.29).

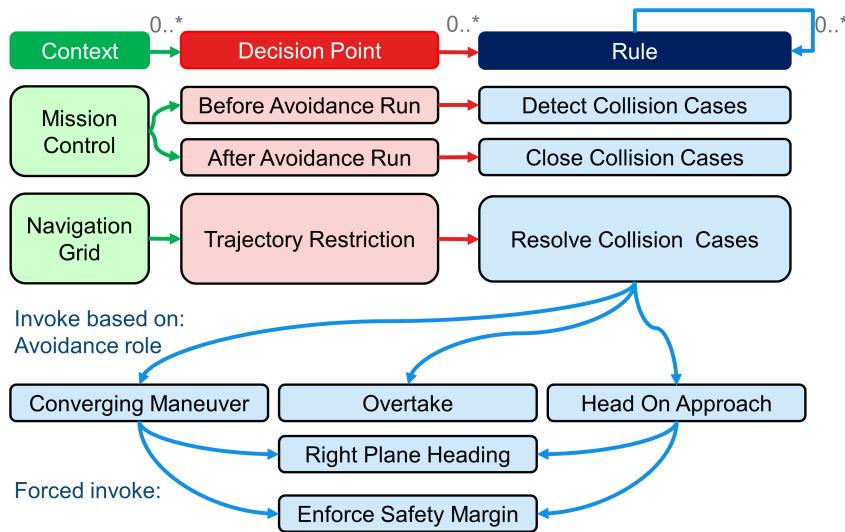


Figure 6.29: Rule engine initialization with Rules of the air.

Decision Points: The *Decisions* are bounded to *Mission Control Run Process* (fig. 6.22) in the following manner:

1. *Before Avoidance Run* (before step 7.) - Context: *Mission Control* (Received Collision Cases) - the *UTM* can send directives. It is required to find which ones are impacting our *UAS*.
2. *Trajectory Restrictions* (after step 7.) - Context: *Navigation Grid* (Trajectory Restrictions) - an adaptation of *behavior* imposed by *active collision cases*.
3. *After Avoidance Run* (after step 11.) - Context: *Mission Control* (Collision Case Resolutions) - our *UAS* will update the status of *Collision Cases* then it checks the *avoidance conditions*. The *Resolution Notification* resolution notifications are sent to *UTM* afterward.

Note. The *Weather Case* (app. E.1) is handled similarly. The mission control loop (fig. 6.22) have rules with separate *Decision Points* to enforce *hard constraints* (before step 9.) and *soft constraints* (before step 10.).

Road map: The *implemented rules*(cyan) are separated into the following categories:

1. *Management Rules* - managing collision cases (additional control flow):
 - a. *Detect Collision Cases* (app. E.2) - the detection of active participation in received *collision cases* and generation of *restrictions*.
 - b. *Resolve Collision Cases* (app. E.3) - the enforcement of *active avoidance roles* in *collision cases*. The one *Restriction Rule* is invoked directly.
 - c. *Close Collision Cases* (app. E.4) - impact calculation and *Resolution Notification* to *UTM authority*.
2. *Restriction Rules* - restricting the *Navigation Grid* trajectories or altering *goal waypoint* based on *selected collision cases*:
 - a. *Converging Maneuver* (app. E.6) implementation of *Converging Avoidance* (sec. 6.7.3).
 - b. *Head On Approach* (app. E.5) implementation of *Virtual Roundabout Enforcement* (sec. 6.7.2).
 - c. *Overtake* (app. E.7) implementation of *overtaking maneuver* for *Overtaking plane* (sec. 6.7.4).
3. *Miscellaneous Rules* - reused pieces of code in *Head-On* and *Converging Situations*:
 - a. *Right Plane Heading* (app. E.8) - restrict all trajectories heading to space separated by parametric plane in *Avoidance Grid* which is heading or belonging to plane.
 - b. *Enforce Safety Margin* (app. E.9) - restrict all *Trajectories Segments* which are in proximity of *Collision Point* lesser than *Enforced Safety Margin*.

Chapter 7

Simulations

The chapter presents the set of simulations developed according to a test plan (sec. 7.1). Test configuration (sec. 7.2) targets at exercising and evaluating proposed framework. The test cases are grouped in the following sections:

1. *Non-cooperative test cases* (sec. 7.3).
2. *Cooperative test cases* (sec. 7.4).
3. *Test cases conclusion* (sec. 7.5).
4. *Reach set approximation performance tests* (sec. 7.6).

7.1 Test Plan

The *Avoidance requirements* are given in (sec. 4.5), namely:

1. *Safety Margin Enforcement* (sec. 4.5.3) - keep UAS safe depending on situation.
2. *Path Tracking* (sec. 4.5.2) - track mission is given by a set of *waypoints* in the manner of *Energy Efficiency* (sec. 4.5.1).

These are given as nominal behavior (sec. 6.6.1), further enhanced by rule-based behavior (sec. 6.8.2).

The *Navigation requirements*, out of this scope, are given in (sec. 4.6). These are satisfied by *Mission Control Run* (sec. 6.6.2).

7.1.1 Testing approach

The purpose of this section is to show complex scenarios, not unit testing of framework functionality. The focus is on *borderline* cases for typical situations in an *expected environment*. The *mode switch* between *Navigation* and *Emergency Avoidance*.

The *Tests* are designed to focus on particular functionality in specific *operational environment* with main *obstacle/weather/intruder feature* with environment induced *constraints*. There is also *UTM* factor and *Navigation penalty*.

Operational Environment is classified according to:

1. *Operation space* - important for *Low Altitude Operations*, the difficulty of *Avoidance Maneuvers* is proportionally increasing with *Obstacle density*. There are following main categories
 - a. *Rural environment* - the relief and man-made structures are sparsely spread around the *operation space*; the UAS is operating on *very low altitude* (≤ 50 feet).
 - b. *Urban environment* - the concentration of the man-made structures are much higher, and they are more incorporated into land relief pattern, the UAS is operating on *very low altitude*.
 - c. *Open air* - the concentration of ground structures is very low, the concentration of *cooperative* and *non-cooperative intruders* is increased, the UAS is operating in altitude ranging from *50 feet* to *space border*. This brings us to:
2. *Airspace category* - when *Operation Space* pattern is categorized as *Open air* and depending on *altitude above mean sea level*. The UTM is *designed authority* for controlled airspace in current *F/G class airspace*.
 - a. *Controlled* - Open air where authority is present. The cases when *Authority* is not enforced due to the UTM malfunction, *C2 link loss* or other cause are not considered.
 - b. *Non-Controlled* - Open air operation space where is no central arbiter to determine or enforce traffic attendants behavior.

Static obstacles: Static obstacles with various features detectable by main *LiDAR* sensor. The main purpose is to show avoidance capabilities combined with heavy restrictions imposed by *soft* and *hard* constraints. The original purpose of our approach was to provide robust framework for static obstacle avoidance. Three tests with increasing obstacle density and navigation complexity are delivered.

Operational Space Constraints depends mainly on the *operational environment*. The standard set of constraints were taken into account for our test cases:

1. *Rural, Urban environment (low altitude)* are geo-fencing zones, ground (hard constraints), non-controlled airspace altitudes (soft constraints).
2. *Non-controlled airspace constraints (open air)* are geo-fencing zones (hard constraints), restricted airspace (hard constraint), weather (soft/hard constraint), controlled airspace (hard constraint), very low altitude border (soft constraints).
3. *Controlled airspace constraints (open air)* are restricted airspace (hard constraint), weather (soft/hard constraint), non-controlled airspace boundary (hard constraints), UTM Directives (hard constraints).

Air Traffic Attendants:

1. *Non-cooperative UAS* (Intruder) - there are some intruders with some degree of authority, size and *severity*. There were three test cases for non-cooperative intruders. Non-cooperative Intruders can be categorized as following based on behavior:
 - a. *Chaotic* intruders usually tend to behave unpredictable, for example, bird or *UAS in distress*, for this type of intruders *Maneuver Uncertainty Intersection Model* is used (app. C.3).
 - b. *Harmonic* intruder usually follows long straight paths, for example, UAS converging to waypoint, for this type of intruder *Body Volume Intersection Model* is used. (app. C.2).

Cooperative UAS (Intruder) - there are cooperative intruders who are obeying authority (UTM) or follow *common consensus*. The work focus on *UTM* authority implementation in four test cases. These test cases are reflecting the traffic management situations essential for successful UTM collision management

Weather impose *soft* and *hard* space constraint, which can be moving or static. The *soft constraint avoidance* is covered by *hard constraint avoidance*. The *static constrained area* is covered by *static obstacle avoidance* capability due to the *data fusion procedure* [13]. The only case which is not covered is *Moving constrained area*; small constraints can be covered by intruder models. The ideal candidate is a *storm*, because it covers quite a large area, the clouds are constantly moving, and severity is changing with time.

UTM: The *UAS Traffic Management* service should be implemented in *controlled airspace* by 2035. It is necessary to study impact of UTM services on the *Detect and Avoid* systems like ours.

The most basic service is *Identity provider* which should be implemented by 2020.

Then there are *location services*, which are necessary for coordinated collision avoidance, these were implemented in our solution up to necessary level for *Rules Of the Air* implementation.

Mission tracking is service tracking deviations from *declared mission plan* and *actual execution*. These statistics were used in all tests to track deviations from the reference trajectory.

Directives for *Traffic management* and *Collision prevention* are implemented as the functional life cycle of *Position notification* (sec. 6.7.5), *Collision Case* (sec. 6.7.6) for UTM. The directive handling is implemented as *Rule engine* (sec. 6.8.2) on UAS side.

Navigation: Navigation algorithm is depending on *Navigation mode*. UAS is usually in *Navigation mode* most of the time, despite this fact, UAS was forced into *Emergency Avoidance Mode* most of the time in test cases. The navigation complexity has been divined into following categories:

1. *Open space* - UAS has visibility to goal waypoint most of the time; there are no traps.
2. *Hidden waypoint* - UAS does not have visibility to goal waypoint, most of the time; there are irregular traps sometimes.
3. *Maze solving* - UAS line of sight for goal waypoint is hindered by multiple obstacles, there are irregular traps often.
4. *Rule following* - UAS navigation capabilities are constrained by rule enforcement.

7.1.2 Test Cases Summary

Test cases are summarized in (tab. 7.1).

| Test Case Name | Operational Environment | Air Traffic Attendants | Weather | UTM | Navigation | Scenario |
|-----------------------|--|--------------------------|---------|------|-----------------|---|
| Building Avoidance | Non-controlled (Rural) 4 × buildings | - | - | - | Open space | Fly mission around four buildings |
| Slalom | Non-controlled (Rural) 14 × buildings | - | - | - | Hidden waypoint | Navigate to hidden waypoint |
| Maze | Non-controlled (Urban) 30 × buildings | - | - | - | Maze structure | Solve maze with multiple curves |
| Storm | Non-controlled (Rural) 0 × buildings | - | Storm | - | Open Space | Avoid approaching storm |
| Emergency Converging | Non-controlled (Open air) | Non-cooperative UAS (1x) | - | - | Open Space | Converging situation resolution w. o. UTM |
| Emergency Head on | Non-controlled (Open air) | Non-cooperative UAS (1x) | - | - | Open Space | Head on situation resolution w. o. UTM |
| Emergency Multiple | Non-controlled (Open air) | Non-cooperative UAS (3x) | - | - | Open Space | Multi-collision case resolution w. o. UTM |
| Rule-based Converging | Controlled (Open air) | Cooperative UAS(1x) | - | Full | Follow Rules | Converging situation resolution with UTM |
| Rule-based Head on | Controlled (Open air) | Cooperative UAS(1x) | - | Full | Follow Rules | Head on situation resolution with UTM |
| Rule-based Multiple | Controlled (Open air) | Cooperative UAS(3x) | - | Full | Follow Rules | Multi-collision case resolution with UTM |
| Rule-based Overtake | Controlled (Open air) | Cooperative UAS (1x) | - | Full | Follow Rules | Overtake by UAS different speed ratio |

Table 7.1: Test Cases Summary.

7.1.3 Performance Evaluation

Evaluation method: *Test cases* were evaluated according to performance requirements defined in (sec. 4.5). The method was tracking critical parameter for *Safety* (sec. 4.5.3) (primary) and *Trajectory Tracking* (sec. 4.5.2) (secondary) including *Energy Efficiency* (sec. 4.5.1).

Safety Margin Performance Evaluation: The *safety of UAS* is main concern of *DAA system*. The common concept of *safety margin* is evaluated.

The *threat* is multidimensional; there are often multiple *static obstacles*, *intruders* or *weather constraints*. To reduce the multidimensional threats to one-dimensional value *crash distance* concept is used:

$$\text{crashDistance}(t) = \text{distance}(\text{UAScenter}(t), \text{threat})$$

where *selection the criterion* is:

$$\min \left\{ \begin{array}{c} \left(\text{distance}(\text{UAScenter}(t), \text{threat}) - \dots \right) \\ \dots - \text{threat.SafetyMargin} \\ : \forall \text{threat} \in \text{KnownWorld}(t) \end{array} \right\} \quad (7.1)$$

The *crash distance* (eq. 7.1) for given time is evaluated as shortest distance between UAS center and threat. The threat originates from the known world (sec. 4.1.10). The *threat* has safety margin. The distance to safety margin is used as a prioritization criterion in our test cases (tab. 7.1).

The *safety margin* evolution over time (eq. 7.2) is calculated similarly to *crash distance*. The most dangerous threat is selected based on *distance to the safety margin* criterion. The value of *safety margin* property is then used.

$$\text{safetyMargin}(t) = \text{threat.SafetyMargin}$$

where *the selection criterion* is:

$$\min \left\{ \begin{array}{c} \left(\text{distance}(\text{UAScenter}(t), \text{threat}) - \dots \right) \\ \dots - \text{threat.SafetyMargin} \\ : \forall \text{threat} \in \text{KnownWorld}(t) \end{array} \right\} \quad (7.2)$$

The *distance to safety margin* (eq. 7.3) is calculated as a difference between the *crash distance* (eq. 7.1) and *safety margin* (eq. 7.2). The *acceptance criteria* for safety is the *distance to safety margin* ≥ 0 .

$$\text{distanceToSafetyMargin}(t) = \text{crashDistance}(t) - \text{safetyMargin}(t) \geq 0 \quad (7.3)$$

Note. On Signed Distance: The most works are using *unsigned distance*. This work considers the *signed distance* with the following intervals:

1. + (away from the safety margin).
2. 0 (touching margin with UAS edge).
3. - (inside margin - crash/collision/broken boundary).

Distance to Safety Margin peaks are measured:

1. *Minimal* distance to safety margin indicates if *acceptance criterion* (eq. 7.3 is met).

2. *Maximal* distance to safety margin indicates the future *minimal detection range*. All scenarios were considered as borderline cases.

Trajectory Tracking Evaluation is a secondary priority after safety, following parameters were checked:

1. *Waypoint reach* - the *Mission* (4.6) is considered as completed if and only if \forall waypoints are reached and in the given order (check the output of 4.7). Moreover, if there is multiple UAS, each must meet the condition.
2. *Acceptable deviation* - for *tracking problem* (eq. 4.34) is a trajectory which in addition to *basic obstacle problem* (sec. 4.2) keeps deviation from the *reference trajectory* under a certain threshold (eq. 4.35).

Trajectory tracking deviation threshold (eq. 7.4) is defined as double of maximal distance between *goal waypoint* and *previous waypoint*.

$$\text{trackingDeviationThreshold} = 2 \times \text{distance}(\text{goalWaypoint}, \text{previousWaypoint}) \quad (7.4)$$

Note. If *goal waypoint* is first in the *mission*, the *UAS initial condition* is considered as a *previous waypoint*.

Computation Load: There is a theoretical definition of *intersection models* for *static obstacles and constraints* (sec. 6.5.1), *moving obstacles and constraints* (sec. 6.5.2), *avoidance run* (sec. 6.6.1), *mission control* (sec. 6.6.2) computational complexity.

The practical application requires to measure *computation load* in constrained environment. Let say that *avoidance framework* is running on stand alone embedded computer with 1.2 GHz processor and 1GB of dedicated RAM. This is simulated by *virtual machine*.

The *simulations* were executed in *Matlab/Simulink* environment¹ using: *UTM*², *Navigation loop*³, *Avoidance grid*⁴ and *Reach set*⁵ implementations.

The *decision frame* length is set to 1s which gives *computation load* (eq. 7.5). The *computation load* represents the portion of the *previous decision frame* used to current decision frame calculation.

$$\text{computationLoad} = \frac{\text{computationTime(frame)}}{\text{decisionFrameDuration}} \times 100, \quad [\%; s, s] \quad (7.5)$$

¹Prototype framework implementation: <https://github.com/logomo/Feature-based-ACAS/>

²UTM class: .../UavTraficManagement/UTMControl.m

³Navigation Loop main class: .../MissionControl/MissionControl.m

⁴Avoidance Grid class: .../AvoidanceGrid/AvoidanceGrid.m

⁵Reach set tree class: .../AvoidanceGrid/PredictorNode.m

Note. *Computation load* is depending on the actual situation; when the UAS is in *navigation mode*, it should be low, when the UAS is in a *clustered environment* it should be high.

Matlab implementation is quite ineffective; the Python/C++ implementation can give better results.

For *computational feasibility* there is *implicit* acceptance criterion (eq. 7.6): the computation of a feasible path for *this time-frame* must end in the *previous time-frame*.

$$\forall \text{time} \in \text{Mission} : \quad \text{computationLoad} < 100\% \quad (7.6)$$

7.2 Testing Configuration

All *simulations* are run with the configuration described in this *section*. The UAS used for the purposes is given by *model and control* (sec. 6.2).

UAS parameters: An *UAS system* (tab. 7.3) is modeled after small scale toy model with maximal body radius 30 cm, maximal speed 4 m.s⁻¹, weight 450 g., maximal flight duration 20 min, maximal turning rate 15 deg.s⁻¹. The *body margin* is set to 0.3m; the *near-miss radius* is double of *body margin*; thus 0.6 m, the *well clear radius* is set to 5 m. Margins can be set to any value if they are complaint with condition (7.7).

$$0 < \text{bodyMargin} \leq \text{nearMissRadius} \leq \text{wellClearRadius} \leq \text{gridDistance} \quad (7.7)$$

Note. The *safety margin* is broad term used to describe the *minimal distance* between UAS and *adversarial object*. The *Safety margin* is:

1. *Near miss radius* in case of *non-controlled airspace* or *emergency avoidance mode*.
2. *Well clear radius* in case of *controlled airspace* and *navigation mode*.

Decision time: Decision time can be set by the user to any positive non-zero value (7.8). The *Decision time* is equal 1 s, and *Decision frames* are synchronized.

$$\text{maxAlgorithmCalculationTime} \leq \text{decisionTome} \leq \infty \quad (7.8)$$

Speed: For all movements constant speed 1 m.s⁻¹ is used. Speed can be changed to any value in the given boundary (7.9).

$$0 \leq \text{speed} \leq \min \left(\begin{array}{l} 0.5 \times (\text{navigationGrid.distance}/\text{decisionFrame}) \\ 0.5 \times (\text{avoidanceGrid.distance}/\text{decisionFrame}) \end{array} \right) \quad (7.9)$$

Movement automaton: The *movement set* is given in (tab. 7.2). The *movement set* contains horizontal, vertical, and, combined movements.

Grids: Used *Navigation grid parameters* are given in (tab. 7.4). Selected *Navigation Reach set* is *ACAS-like* with enabled horizontal/vertical separation. Used *Avoidance grid parameters* are given in (tab. 7.5). Selected *Avoidance Reach set* is *combined* because of high *coverage ratio*.

The user can define own grid parameters according to the *space discretization rules* (sec. 6.3) and chose own *reach set type* according to preference (sec. 6.4).

| Movement | Roll | Pitch | Yaw |
|-----------|------|-------|------|
| Straight | 0° | 0° | 0° |
| Left | 0° | 15° | 0° |
| Right | 0° | -15° | 0° |
| Up | 0° | 0° | -15° |
| Down | 0° | 0° | 15° |
| UpLeft | 0° | 15° | -15° |
| UpRight | 0° | -15° | -15° |
| DownLeft | 0° | 15° | 15° |
| DownRight | 0° | -15° | 15° |

Table 7.2: Movement orientations.

| UAS parameters | |
|-----------------------|--------------|
| speed | 1 $m s^{-1}$ |
| horizontal turning r. | 3.82 m |
| vertical turning r. | 3.82 m |
| body radius | 0.3 m |
| near miss r. | 0.6 m |
| well clear r. | 5 m |

Table 7.3: UAS parameters.

| Navigation Grid | |
|------------------|-----------|
| RSA type | ACAS-like |
| distance range | 0 – 10 m |
| layer step | 1 m |
| horizontal range | ±45° |
| horizontal cells | 7 |
| vertical range | ±30° |
| vertical cells | 5 |

Table 7.4: Navigation Space parameters.

| Avoidance Grid | |
|------------------|----------|
| RSA type | combined |
| distance range | 0 – 10 m |
| layer step | 1 m |
| horizontal range | ±45° |
| horizontal cells | 7 |
| vertical range | ±30° |
| vertical cells | 5 |

Table 7.5: Avoidance Space parameters.

| Coloring | | |
|----------|----------|---------|
| Airc. | Executed | Planned |
| UAS 1 | blue | red |
| UAS 2 | cyan | magenta |
| UAS 3 | green | yellow |
| UAS 4 | black | green |

Table 7.6: UAS coloring.

7.3 Non-cooperative test cases

The *main* goal of this section is to show operative capabilities for *non-cooperative avoidance mode* in *emergency* and *solo situations*.

Test avoidance capabilities against *static obstacles*, *non-cooperative intruders*, *moving hard constraints* are covered.

Coverage of the *soft constraints*, *map obstacles*, and *detected obstacles* are implicitly covered due to the properties of *safety* and *body* margins (tab. 4.2).

1. *Building Avoidance* (sec. 7.3.1) covers *static obstacles* explicitly and *map obstacles*, *hard constraints*, *ground avoidance* implicitly.
2. *Slalom* (sec. 7.3.2) covers *open space navigation capabilities*, showing the determinism of the *avoidance loop run*, in addition to *building avoidance*.
3. *Maze* (sec. 7.3.3) covers *closed space navigation capabilities*, showing the higher level navigation properties of primitive *right-side* 2D maze solver. The main point is to show the possibility to enrich the *Navigation loop algorithm* (fig. 6.22).
4. *Storm* (sec. 7.3.4) covers *hard moving constraints avoidance* explicitly and *hard static constraints*, *soft static constraints*, *soft moving constraints* implicitly.
5. *Emergency converging scenario* (sec. 7.3.5) covers *non-cooperative intruder with the right of way* avoidance capability.
6. *Emergency head-on scenario* covers (sec. 7.3.6) *non-cooperative intruder without right of way* avoidance capability
7. *Emergency mixed scenario* (sec. 7.3.7) covers *multiple intruders with/without right of the way* avoidance capability.

7.3.1 Building avoidance

Scenario: The *UAS* is flying the mission given by (tab. 7.7) in the *open space environment*. There exists a map of obstacles with defined *safety* and *body margins*. *Reference trajectory* (direct interconnection of waypoints) is going through partially known space with some charted obstacles.

| Position | | Waypoints | | | |
|---------------|---------------------------------|------------------|-------------------|------------------|------------------|
| $[x, y, z]$ | $[\theta, \varpi, \psi]$ | \mathcal{WP}_1 | \mathcal{WP}_2 | \mathcal{WP}_3 | \mathcal{WP}_4 |
| $[0, 0, 0]^T$ | $[0^\circ, 0^\circ, 0^\circ]^T$ | $[100, 0, 0]^T$ | $[100, 100, 0]^T$ | $[0, 100, 0]^T$ | $[0, 0, 0]^T$ |

Table 7.7: Mission setup for *Building avoidance* scenario.

Obstacle set: Obstacles are discovered during a flight by *UAS LiDAR sensor*, the set of obstacles is defined in (tab. 7.8).

| Obstacle | | | Body Margin | | | Safety Margin |
|----------|------------------|-----------|-------------|------|------|---------------|
| id | position | type | min. | max. | avg. | |
| 1 | $[50, 0, 0]^T$ | polygonal | 14 | 20 | 16 | 5 |
| 2 | $[100, 50, 0]^T$ | hospital | 12 | 18 | 14 | 7 |
| 3 | $[50, 100, 0]^T$ | unusual | 10 | 20 | 15 | 8 |
| 4 | $[0, 50, 0]^T$ | square | 18 | 20 | 19 | 4 |

Table 7.8: *Obstacle set* for *Building avoidance scenario*.

Main Goal: Show *static obstacle avoidance capability* in an *open space environment*, using *LiDAR scanning* and *obstacle map* as the *information sources*.

Acceptance criteria:

1. Proper *algorithm mode switch*:
 - a *Avoidance mode* is active when the *UAS* is nearby to the obstacle (*distance* (*obstacleCenter*, *UASPosition*) $\leq 20m$).
 - b *Navigation mode* is active when the *UAS* is further away from any obstacle (*UAS* is actively converging to *goal waypoint*).
2. *Minimal safety margin distance* $\geq 0m$
3. *Reach each waypoint* (tab. 7.7) in the given order.

Testing Setup: The *standard test setup* defined in (tab. 7.2, 7.3, 7.4, 7.5, 7.6) is used with following parameter override:

1. *Avoidance grid - type - ACAS-like* with *horizontal enabled maneuvers*

Note. Enforced *safety margin* does not exceed the *avoidance grid range* (10 m). The concept of *Static obstacle avoidance* is in detail discussed in the *progress report* [13].

Simulation Run: Notable moments from the *simulation run* (fig. 7.1) are the following:

1. *1st building avoidance*. (fig. 7.1a) - UAS avoids the building from the left side because overall trajectory cost is cheaper. The first building is a convex obstacle.
2. *2nd building avoidance*. (fig. 7.1b) - UAS avoids the building from the right side while avoiding an active non-convex portion of the building.

3. 3rd building avoidance. (fig. 7.1c) - UAS avoids the building from the right side, missing both traps from it.
4. 4th building avoidance. (fig. 7.1d) - UAS avoids the building from the right side. This building is also a convex obstacle.

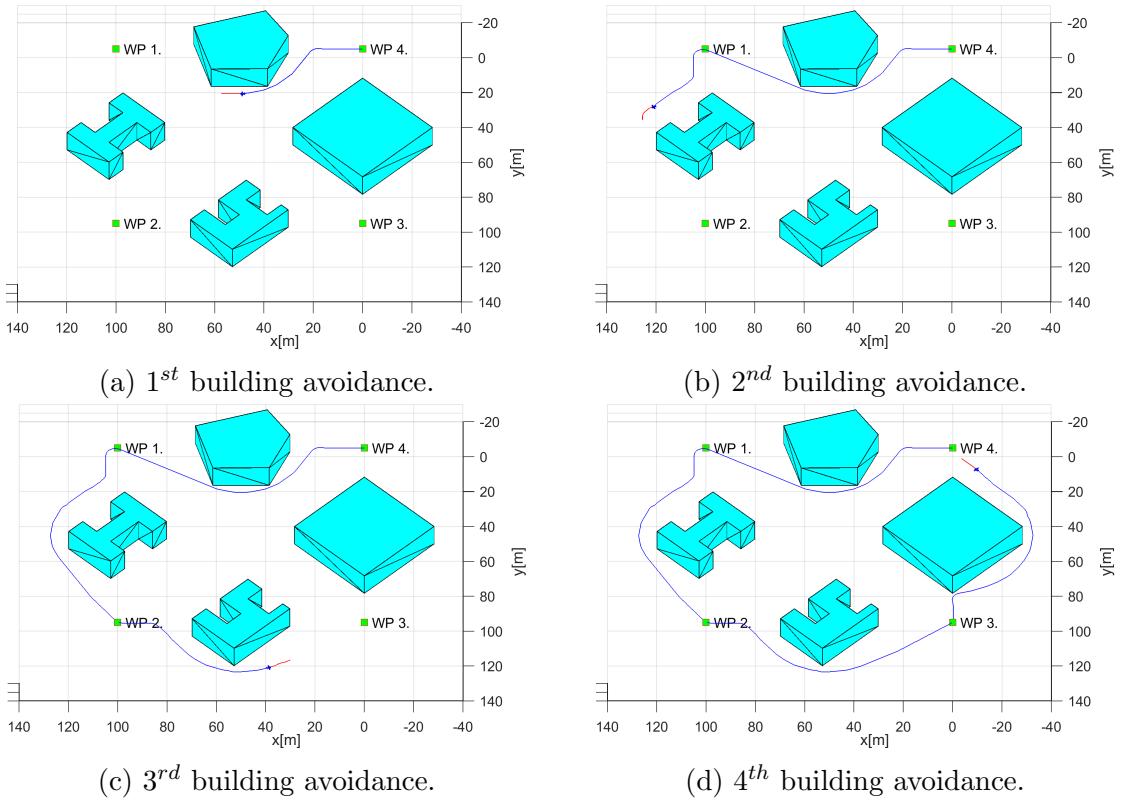


Figure 7.1: Test scenario for *Building avoidance* (static ground obstacles).

Distance to Body/Safety Margin Evolution: The distance of *UAS* center to the nearest obstacle (blue) does not break a *safety margin* (of the closest obstacle (yellow)) nor *body margin* of the closest obstacle (red) as it can be seen in (fig. 7.2). *Acceptance condition for algorithm mode switch* can be shown by *UAS active avoidance of obstacles*.

Note. The *body* and *safety margins* are changing depending on *UAS position and orientation*, is changing reflecting (tab. 7.8) margins.

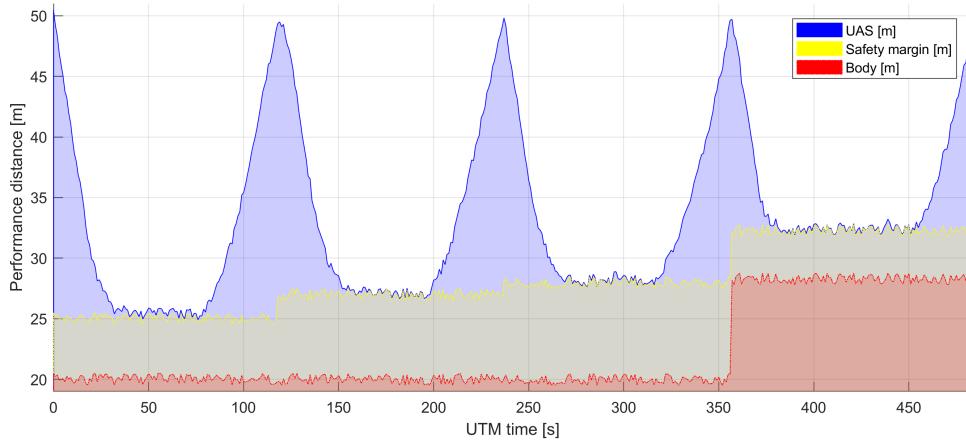


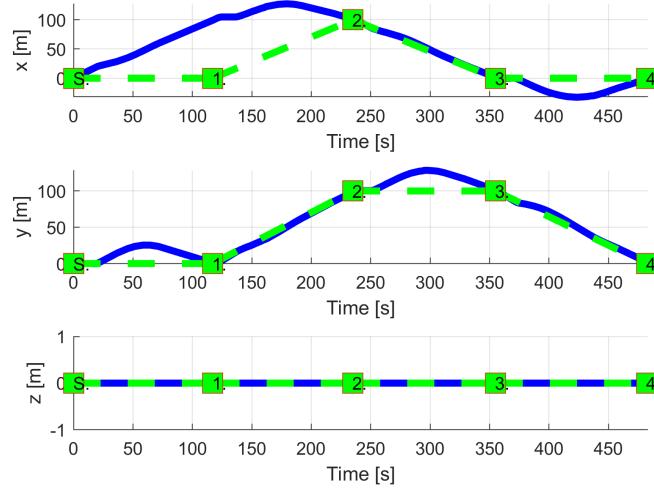
Figure 7.2: Distance to body/safety margin evolution for *Building avoidance scenario*.

Distance to Body/Safety Margin Peaks: Minimal distance to *safety margin* is 0.69 m. The *minimal distance to obstacle body* is 4.69 m which is more than sufficient for tested UAS type. *Safety margin acceptance criteria* have been achieved because the minimal distance is greater than zero. The minimal *body margin distance* is 4.69 m for obstacle no. 4 (tab. 7.8).

| Parameter | UAS 1 | |
|---------------------------|-------|-------|
| Distance to Safety Margin | min | 0.69 |
| | max | 24.98 |
| Distance to Body Margin | min | 4.69 |
| | max | 29.98 |

Table 7.9: Distance to Body/Safety Margin Peaks for *Building avoidance scenario*.

Path Tracking Performance: Reference path (green dashed line) is given as direct interconnection between waypoints (green numbered square). The real trajectory (solid blue line) is split into its XYZ components. *All mission waypoints* (fig. 7.3) have been reached in the given order. There are some deviations on $X - Y$ horizontal axes, while the UAS was in the *avoidance mode*.

Figure 7.3: *Building avoidance* path tracking.

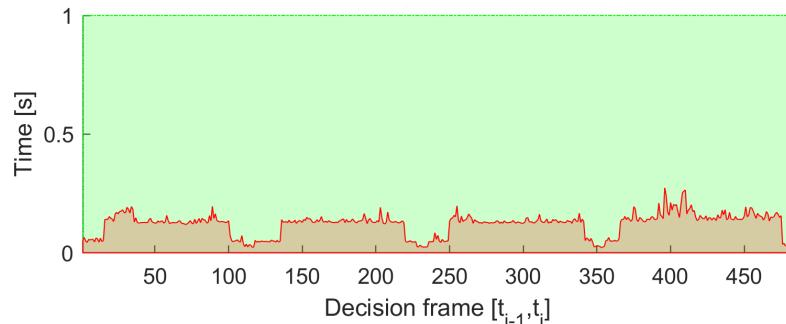
Path Tracking Deviations: Deviations (tab. 7.10) from the *reference trajectory* are in expected ranges considering the *mission plan* (tab. 7.7) and *obstacle properties* (tab. 7.8).

| Param. | UAS 1 | | | |
|--------------|------------------|----------------|------------------|------------------|
| | \mathcal{WP}_1 | \mathcal{WP} | \mathcal{WP}_3 | \mathcal{WP}_4 |
| $\max x $ | 104 | 86 | 5.34 | 32.52 |
| $\max y $ | 25.39 | 6.59 | 28.2 | 4.55 |
| $\max z $ | 0 | 0 | 0 | 0 |
| $\max dist.$ | 107.05 | 86.2 | 28.7 | 32.84 |

Table 7.10: Path tracking for properties *Building avoidance*.

Computation Load: The *computation load* for *scenario* (fig.7.4) shows used time (y-axis) over decision frame (x-axis).

There is a slight increase in *computation time* when UAS is in *Emergency Avoidance Mode*.

Figure 7.4: Computation time for *Building avoidance* scenario.

7.3.2 Slalom

Scenario: The *UAS* is flying the mission given by (tab. 7.11) in the *open-space environment*. An Operational space is more clustered than in the case of *Building Avoidance* (sec. 7.3.1). This map of notable *buildings* with defined *safety and body margins* imposing additional flight constraints. The *UAS* is flying through partially known space with some charted obstacles.

The *goal waypoint* is hidden behind the sensors line of sight. There are multiple cost equivalent trajectories to reach the goal.

| Position | | \mathcal{WP}_1 |
|----------------|----------------------------------|------------------|
| $[x, y, z]$ | $[\theta, \varpi, \psi]$ | |
| $[25, 5, 0]^T$ | $[0^\circ, 0^\circ, 90^\circ]^T$ | $[35, 75, 0]^T$ |

Table 7.11: Mission setup for *Slalom* scenario.

Obstacle set: Obstacles are discovered during a flight by *UAS LiDAR Sensor*. The set of obstacles is defined in (tab. 7.12) Some obstacles does not have *Line of Sight* during a flight, which causes additional constraints during the *avoidance trajectory selection* process.

| Obstacle | | Body Margin | | | Safety Margin |
|--------------|----------|-------------|--------------|------------|---------------|
| position | type | min. | max. | avg. | |
| multiple (4) | hospital | $[0.5, 1]$ | $[2.2, 3.1]$ | $[1.5, 3]$ | $[1, 3]$ |
| multiple (7) | unusual | $[0.3, 1]$ | $[2.3, 3.5]$ | $[2, 3]$ | $[1, 4]$ |
| multiple (3) | square | $[3, 4]$ | $[4, 5]$ | $[4, 5]$ | $[1, 4]$ |

Table 7.12: *Obstacle set* for *Slalom* scenario.

Main goal: Show *static obstacle avoidance* in a *clustered environment* with *shorter decision frames* due to the obstacle density. Show *hidden waypoint navigation capability* and *Behind Line of Sight* impact on decision making.

Acceptance Criteria are given as follow:

1. *Hidden waypoint reach* - the UAS will safely reach *goal waypoint*.
2. *Minimal safety margin distance* ≥ 0 .
3. *Hindered space* is accounted into decision making (BLOS impact).

Testing setup: The *standard test setup* defined in (tab. 7.2. 7.3. 7.4. 7.5. 7.6) is used with following parameter override:

1. *Avoidance grid - type* - *ACAS-like* with *horizontal enabled maneuvers*

Note. The *vertical separation* was disabled because *UAS* will increase its altitude to reach *goal waypoint*.

Simulation run: Notable moments from this *simulation run* (fig. 7.5) are the following:

1. *Open space obstacle* (fig. 7.5a) - avoidance of open space obstacle, while tracking *hidden waypoint*. This is standard navigation procedure, the middle building in front of *goal waypoint* is hidden by building in front of *UAS*.
2. *Hidden waypoint navigation* is shown in three stages start (fig. 7.5b), middle (fig. 7.5c), and end phase (fig. 7.5d). The *hidden goal waypoint* has been reached, and first acceptance criteria were fulfilled. The *Decision points* of the navigation loop are placed in very high density around this area. The avoided building had following traps which were avoided:
 - a. Trap (fig. 7.5b) on the left side of *UAS* was avoided because there was no turning point inside of space.
 - b. Trap (fig. 7.5c) on the left side of *UAS* was avoided because it was not wide enough to be considered as trajectory space.

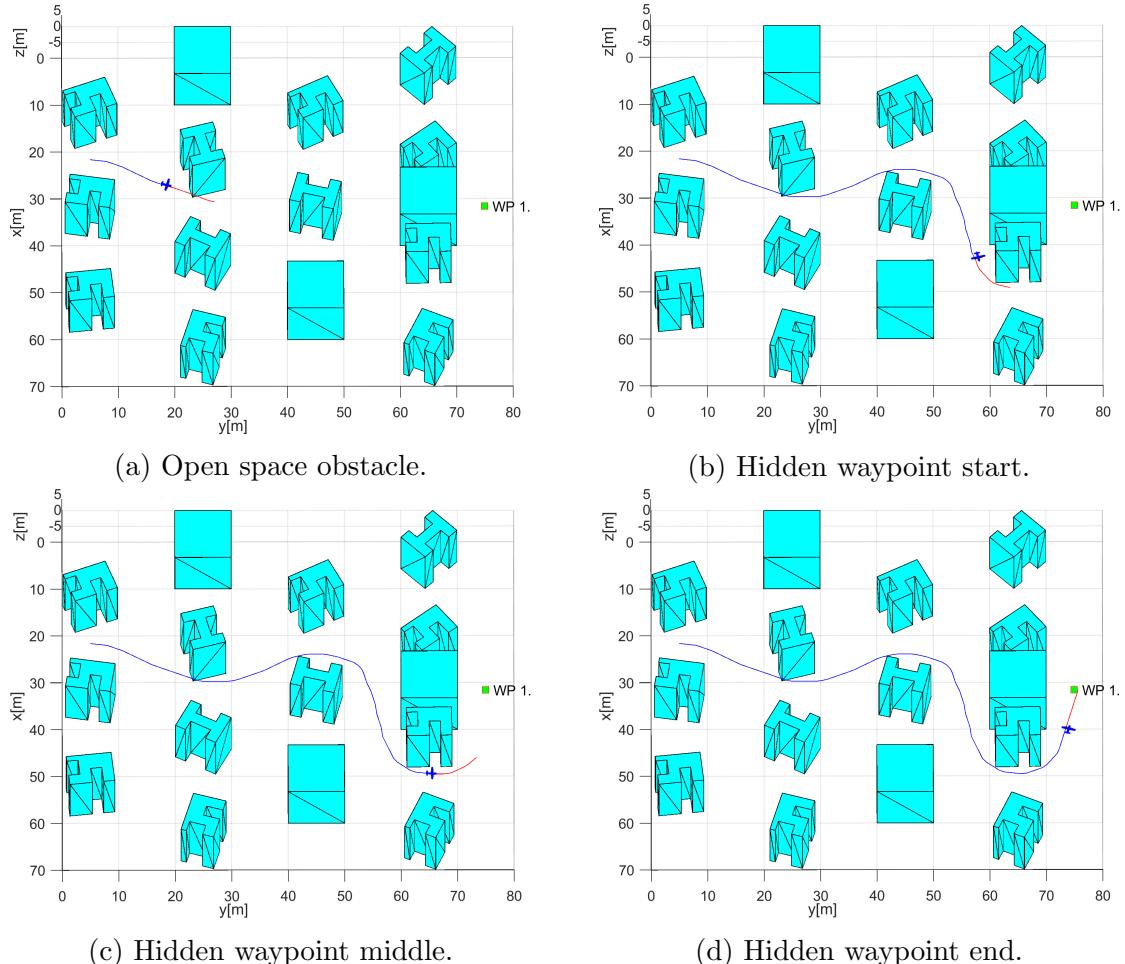


Figure 7.5: Test scenario for *Slalom* with a *hidden waypoint*.

Distance to Body/Safety Margin Evolution: The *UAS* (blue fill) does not break a *safety margin* (yellow fill) nor *body margin* (red fill) as you can see in (fig. 7.6). Hindered space is accounted into decision making because the distance to closest obstacle will never breach *safety margin* (yellow fill). If it was not, the UAS would break *safety* or *body* margin.

Body and *Safety margin* is changing values depending on the *nearest obstacle* and *mutual position of obstacle and UAS*. The ranges of *body* and *safety margins* are reflected in (tab. 7.12).

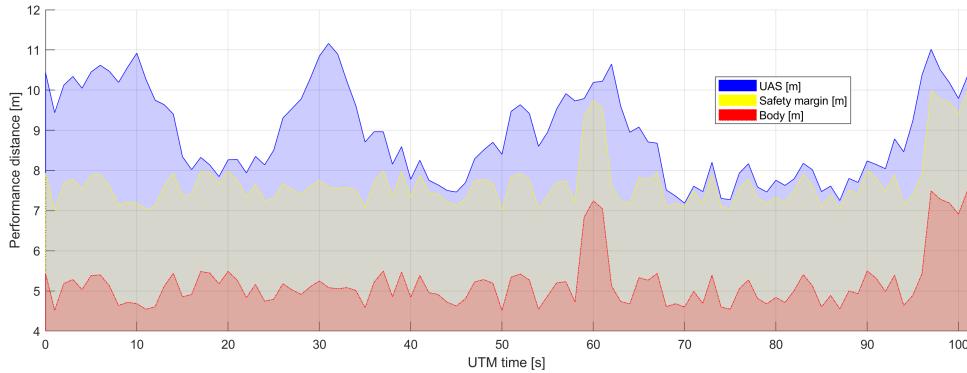


Figure 7.6: Distance to body/safety margin evolution for *Slalom scenario*.

Distance to Body/Safety Margin Peaks: The *UAS* distance to the boundary of *safety* and *body* margin is given in (tab. 7.13). The minimal distance of *UAS border*(blue line) to *safety margin boundary* (yellow line fig. 7.6) is 0.0856 m which can be considered as marginal 0. The minimal *body margin* distance is 2.5856 m ; it takes into account *safety margin* of 2.5 m . The condition $\text{safetyMarginDistance} \geq 0$ holds.

The difference between minimal and maximal *safety margin distance* is $\sim 3\text{ m}$ which indicate that the mission environment is tightly packed with obstacles.

| Parameter | UAS 1 | |
|---------------------------|-------|--------|
| Distance to Safety Margin | min | 0.0856 |
| | max | 3.7391 |
| Distance to Body Margin | min | 2.5856 |
| | max | 6.2391 |

Table 7.13: Distance to body/safety margin peaks for *Slalom scenario*.

Path tracking performance: Path tracking is given in (fig. 7.7). The line between a Starting position (green square, marked S) and goal waypoint (green square marked 1) is reference trajectory (green dashed line). The flown trajectory (blue solid line) is showing evolution over mission time (Time [s]) in global coordinate frame split into three axes ($x[\text{m}]$, $y[\text{m}]$, $z[\text{m}]$). The UAS was all time in *Emergency Avoidance Mode* due to the vicinity of dangerous obstacles.

The *UAS* reached final navigation waypoint, which fulfills acceptance criteria. The UAS has taken a significant detour (x [m] evolution) due to the hidden *waypoint*.

The test has been run multiple times to check if *Right-Up* preference for avoidance is always selected. *Small noise* (0.5-1m) was added to obstacle positions. The algorithm always chose a similar deterministic path. The higher noise levels were not possible due to the obstacle original size (tab. 7.12).

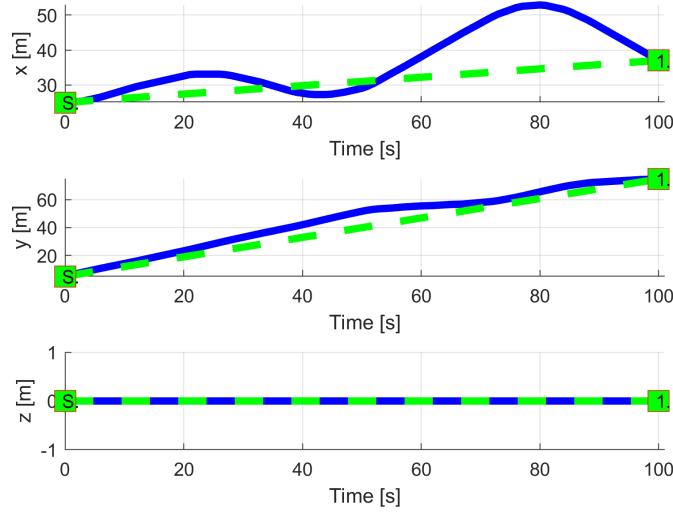


Figure 7.7: *Slalom* path tracking.

Path Tracking Deviations: Deviations given in (tab. 7.14) from *reference trajectory* (fig. 7.7) are in expected ranges considering the *mission plan* (tab. 7.11) and *obstacle properties* (7.12).

| Param. | UAS 1 |
|--------------|------------------|
| | \mathcal{WP}_1 |
| $\max x $ | 17.90 |
| $\max y $ | 12.41 |
| $\max z $ | 0 |
| $\max dist.$ | 20.06 |

Table 7.14: Path tracking properties for *Slalom* scenario.

Computation Load: The *computation load* for *scenario* (fig.7.8) shows used time (y-axis) over decision frame (x-axis).

The UAS is moving over *semi-clustered* environment the *computation load* is almost constant.

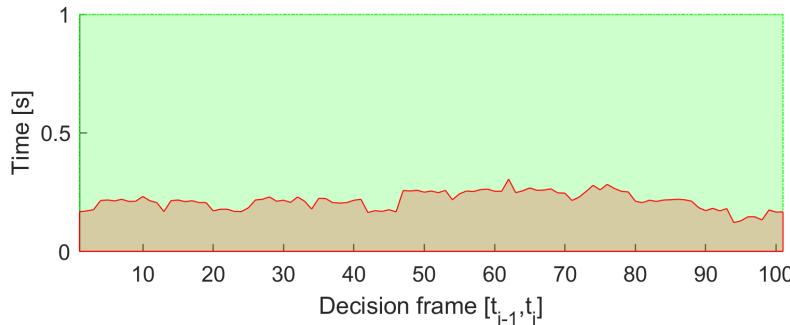


Figure 7.8: Computation time for *Slalom* scenario.

7.3.3 Maze

Scenario: The UAS is flying a mission given by (tab. 7.15) in *closed space* constrained by ground from the bottom, airspace constraint from top and building from sides. The maneuverable space is *maze-like* with *hidden goal waypoint*.

There exists an *Obstacle map* with defined *safety* and *body margins*. *Reference trajectories* (direct interconnection of the initial position and *goal waypoint*) is going through *partially known space* with some charted obstacles.

| Position | | \mathcal{WP}_1 |
|-----------------|---------------------------------|------------------|
| $[x, y, z]$ | $[\theta, \varpi, \psi]$ | |
| $[15, 15, 0]^T$ | $[0^\circ, 0^\circ, 0^\circ]^T$ | $[15, 75, 0]^T$ |

Table 7.15: Mission setup for *Maze* scenario.

Obstacle set: *Obstacles* are discovered during a flight by *UAS LiDAR* sensor. The *Obstacle set* is defined in (tab. 7.16). The obstacles are placed in a *virtual grid* with *cell size* $10 \times 10m$. There are following obstacles:

1. $5 \times$ *Hospital building* - H-shaped, with two open traps, with minimal body margin in the range $0.5 - 1m$, with maximal body margin in the range $2.2 - 3.1m$ and variable *safety margin* in the range $1 - 3m$.
2. $12 \times$ *Unusual trap building* - square-shaped building with two traps on the neighbouring side, with minimal body margin in the range $0.3 - 1m$, with maximal body margin in the range $2.3 - 3.5m$ and variable *safety margin* in the range $1 - 4m$.
3. $6 \times$ *Square building* - square-shaped building with minimal body margin in the range $3 - 4m$, with maximal body margin in the range $4 - 5m$ and variable *safety margin* in the range $1 - 4m$.
4. $7 \times$ *U-shaped Trap* - thin walled U shaped trap designed to catch incoming flying objects, with minimal body margin in the range $2 - 4m$, maximal body margin in the range $3 - 5m$ and various *safety margin* in the range $1 - 2m$.

The purpose of these *Obstacles* except *Square building* type is to create false positive path diversions. These diversions are designed to take *UAS* into an unsolvable situation. *Avoidance* of traps is possible due *Reach set properties* because many scenarios for avoidance can be evaluated at once.

| Obstacle | | Body Margin | | | Safety Margin |
|---------------|----------|-------------|------------|----------|---------------|
| position | type | min. | max. | avg. | |
| multiple (5) | hospital | [0.5, 1] | [2.2, 3.1] | [1.5, 3] | [1, 3] |
| multiple (12) | unusual | [0.3, 1] | [2.3, 3.5] | [2, 3] | [1, 4] |
| multiple (6) | square | [3, 4] | [4, 5] | [4, 5] | [1, 4] |
| multiple (7) | trap | [2, 4] | [3, 5] | [2, 4] | [1, 2] |

Table 7.16: *Obstacle set* for *Maze* scenario.

Main Goal: Demonstrate static obstacle avoidance in closed space navigation. Focus on determinism of *avoidance run*. Demonstrate the possibilities of primitives *right-hand* maze solver incorporated into *Navigation-loop*.

Acceptance Criteria:

1. *Do not break top/bottom boundaries* - the *UAS* Z coordinate should not leave range -5 to $5m$. The boundary break occurs when there is no feasible horizontal path, and *UAS* needs to climb up to resolve the situation.
2. Minimal safety margin distance $\geq 0m$.
3. *Reach hidden goal waypoint* by solving simple maze (tab. 7.15).

Testing Setup: The *standard test setup* defined in (tab. 7.2, 7.3, 7.4, 7.5, 7.6) is used with following parameter override:

1. *Avoidance grid - type - ACAS-like* with *horizontal enabled maneuvers*

Simulation Run: Notable moments from the simulation run (fig. 7.9) are the following:

1. *The Maze* consists from heavy constrained turns: 1st turn (fig. 7.9a), 2nd turn (fig. 7.9b), and 3rd turn (fig. 7.9c). The hidden waypoint reach is given by (fig. 7.9d).
2. *UAS* is constantly in *Emergency Avoidance mode* because there is always a presence of an obstacle.
3. *The Navigation path* is located in a slim corridor with width only 3-6 meters. Mutual distance of obstacles is 20 meters, and combined margins take 14-17 meters.

4. *Maze scenario* was very close to the urban environment concerning obstacle density and computational complexity.
5. *Avoidance run* computational complexity scaled linearly with a count of active obstacles in Field of View.
6. *Hidden Goal Waypoint* has been reached as shown in (fig. 7.9d). This satisfy *reach hidden waypoint* acceptance criterion.

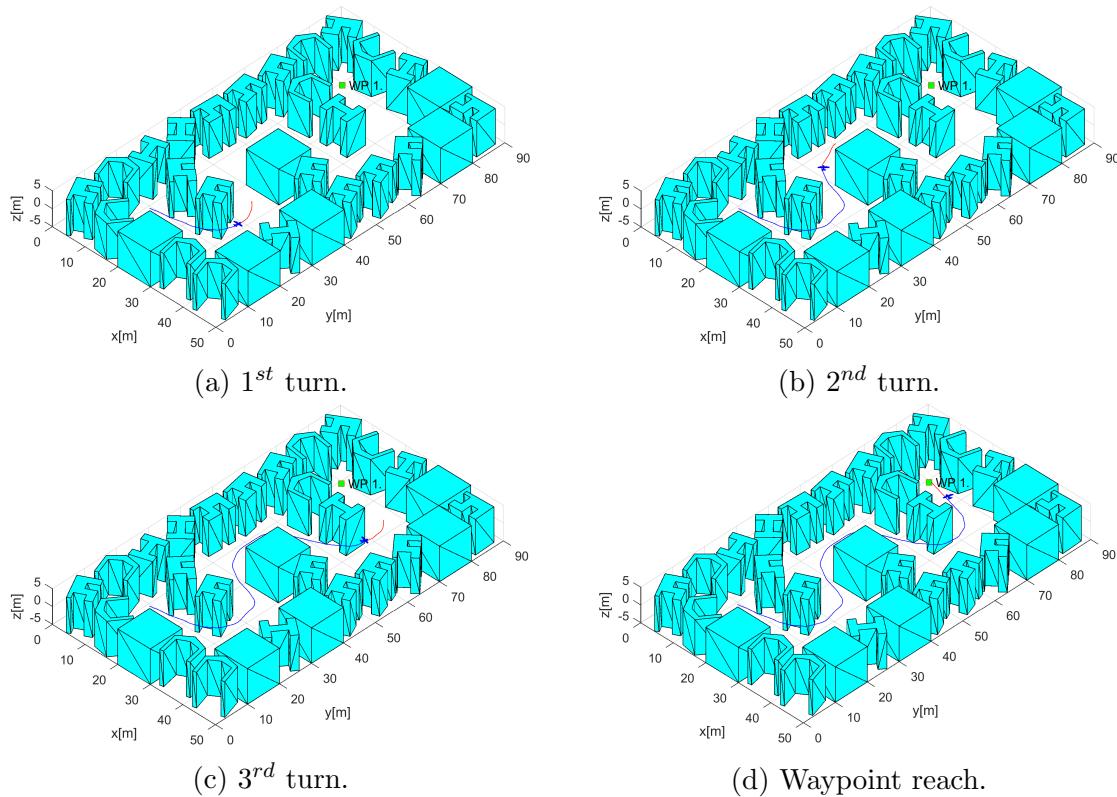


Figure 7.9: Test scenario for *Maze*.

Distance to Body/Safety Margin Evolution: The evolution of *body and safety margin* over time (x-axis, sec) given in meters distance (y-axis, m) is given in (fig. 7.10).

The *UAS* center distance to the nearest obstacle (blue line) does not break any *Safety Margin* (yellow line) of the closest obstacle. *Body Margin* of the closest obstacle (red line) has not been broken, because it always lies below of *Safety Margin* (yellow).

For *UTM period* 37 to 68 s, there is a *margin spike* due to avoidance of bloated *Rectangle buildings* (fig. 7.9b) during the 2nd turn. The *acceptance criterion* for *Safety Margin* is satisfied.

Note. The *body and safety margin* is changing depending on *UAS position and orientation*. The changes are reflected in (tab. 7.17).

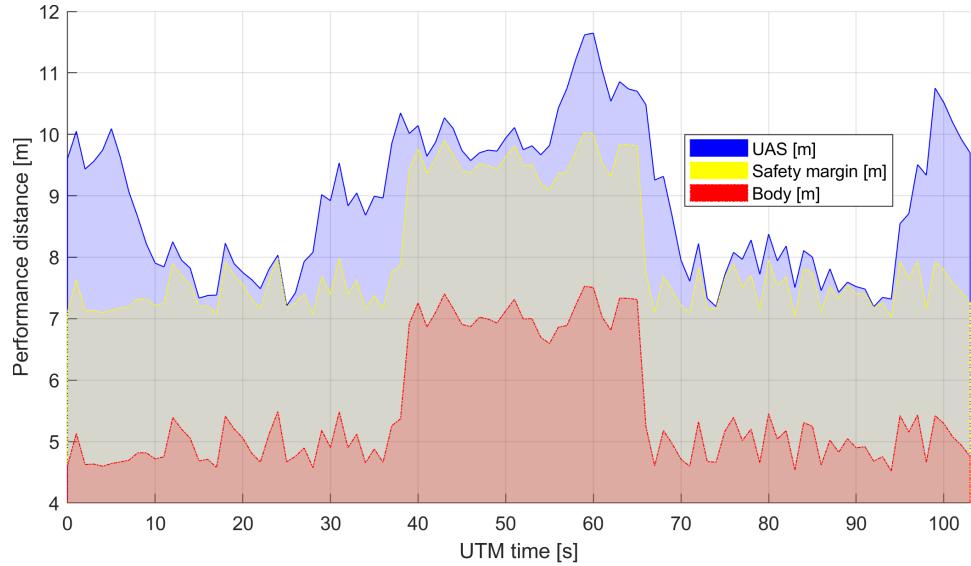


Figure 7.10: Distance to body/safety margin evolution for *Maze scenario*.

Distance to Body/Safety Margin Peaks: The minimal and maximal values for *UAS distance to safety margin* based on performance (fig. 7.10) is summarized in (tab. 7.17).

The *minimal distance to safety margin* is 0.0131m which can be taken as $\sim 0\text{m}$ due to the numerical error. The *maximal distance to safety margin* is 2.9513m which is $5 \times \text{UAS radius}$. The safety margin distance is $\leq 3\text{m}$ which means the scenario is tightly packed with obstacles. The *UAS* never left *Emergency Avoidance Mode* because of the condition: $\text{safetyMarginDistance} \geq \text{avoidanceGridLength}$ was never satisfied.

The *minimal body* distance is 5.0131m , while the *maximal body* distance is 8.7117m . The difference between minimal and maximal body distance is $\sim 4\text{m}$ which also indicates scenario packed with obstacles.

| Parameter | UAS 1 | |
|---------------------------|-------|--------|
| Distance to Safety Margin | min | 0.0131 |
| | max | 2.9513 |
| Distance to Body Margin | min | 5.0131 |
| | max | 8.7117 |

Table 7.17: Distance to body/safety margin peaks for *Maze scenario*.

Path Tracking Performance: Reference path (green dashed) line is given as direct interconnection of *initial position* (green square with S marker) and *hidden waypoint* (green square with 1 marker). The *UTM Reference Time* is given on x-axis. The evolution of the real trajectory (solid blue line) for each axis is given as follow:

1. *X-axis path tracking* - reflects the maneuvering in the curves of the maze.
2. *Y-axis path tracking* - shows horizontal progress to the *hidden goal waypoint*. The

expected linear tracking is not achievement due to the maneuvering delays on X-axis.

3. *Z-axis path tracking* - shows perfect linear tracking of the reference trajectory. The *altitude acceptance criterion*: $-5m \leq \text{altitude} \leq 5m$ have been fulfilled.

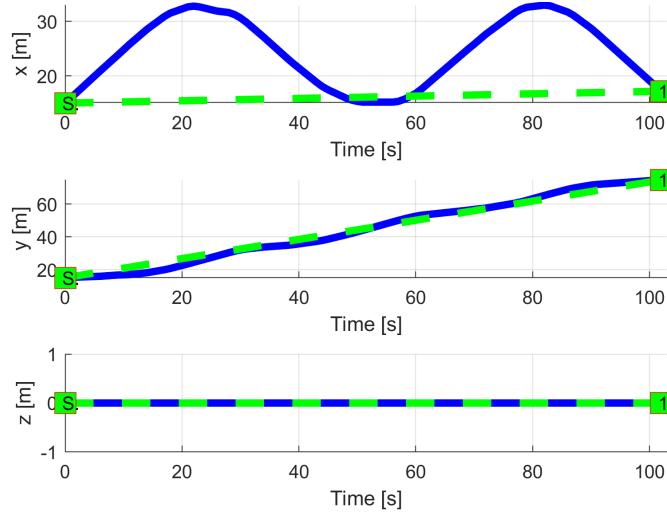


Figure 7.11: *Maze* path tracking.

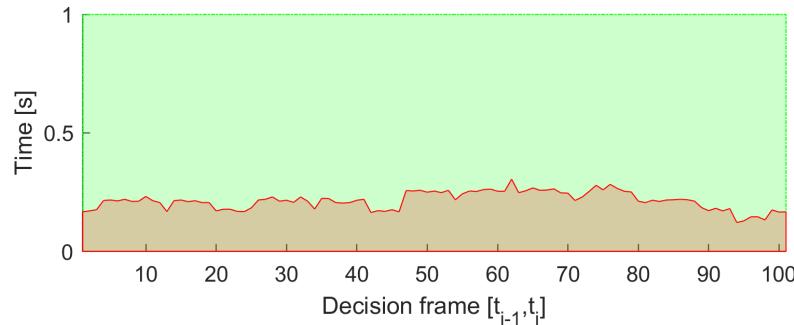
Path Tracking Deviations: Deviations (tab. 7.18) from *reference trajectory* are in expected ranges considering the *mission plan* (tab. 7.15) and *obstacle properties* (tab. 7.16).

| Param. | UAS 1 |
|---------------------|------------------|
| | \mathcal{WP}_1 |
| $\max x $ | 27.32 |
| $\max y $ | 2.41 |
| $\max z $ | 0 |
| $\max \text{dist.}$ | 28.06 |

Table 7.18: Path tracking properties for *Maze* scenario.

Computation Load: The *computation load* for *scenario* (fig.7.12) shows used time (y-axis) over decision frame (x-axis).

The UAS is constantly in *Emergency Avoidance Mode*; the *operational environment* is *cluttered* with obstacles. This causes very high *computation load*.

Figure 7.12: Computation time for *Maze* scenario.

7.3.4 Storm

Scenario: Small UAS is flying in open space in uncontrolled airspace (≤ 500 feet AGL (Above Ground Level)). A *Weather Service* notices UAS about *Dangerous Weather zone* (virtual constraint s. 6.5.3) which is moving in UAS direction. The *UAS* is executing mission given by (tab. 7.19).

| Position | | \mathcal{WP}_1 |
|---------------|----------------------------------|------------------|
| $[x, y, z]$ | $[\theta, \varpi, \psi]$ | |
| $[0, 0, 0]^T$ | $[0^\circ, 0^\circ, 90^\circ]^T$ | $[0, 60, 0]^T$ |

Table 7.19: Mission setup for *Storm* scenario.

Constraints: The *storm* is modeled as a *virtual constraint* with parameters given in (tab. 7.20). A constraint is modeled as a *convex polygon* for *horizontal boundary* and altitude for the *vertical boundary*.

The *Storm* is moving through an *operational region* with linear velocity $0.5ms^{-1}$. The *storm's center* was first detected at *decision frame* 0 at position $[0, 50, 0]^T$.

| Constraint | | | Body Margin | | | Safety Margin |
|----------------|----------------|---------|-------------|------|------|---------------|
| i. position | velocity | type | min. | max. | avg. | |
| $[0, 50, 0]^T$ | $[0, -0.5, 0]$ | polygon | 9 | 10 | 9.5 | 5 |

Table 7.20: *Constraint set* for *Storm* scenario.

Assumption: Every *avoidable moving constraint* is usually slower than an *Approaching UAS*, or its radius is smaller than the turning radius of an *Approaching UAS*.

Note. *Manned aviation* receives a permit to operate in *controlled airspace* only if it has capability outmaneuver every known threat in requested airspace.

The *Constrained space portion* is usually very large, therefore in the majority of cases the assumption *uasSpeed >> constraintSpeed* holds.

Main Goal: Show dynamic moving constraint avoidance capability in *uncontrolled airspace*.

Acceptance criteria:

1. *Hard constraint avoidance* - the *UAS* must not cross the body margin: $distance(stormCenter, UAS) \geq bodyMargin$.
2. *Soft constraint avoidance* - the *UAS* cannot cross the safety margin to get into proximity of *Storms surrounding area*: $distance(stormCenter, UAS) \geq safetyMargin$.

Testing setup: The *standard test setup* defined in (tab. 7.2, 7.3, 7.4, 7.5, 7.6) is used with following parameter override:

1. *Avoidance grid - type - ACAS-like* with *horizontal enabled maneuvers*.

Simulation run: Notable moments from a *simulation run* (fig. 7.13) are the following:

1. *Detection* (fig. 7.13a) - the *Storm* (magenta polygon) is detected prior to the engagement (retrieved from associated weather service). The *UAS* (blue) stays in *Navigation mode*. *Trajectories* in *Navigation grid* are constrained by rule *Enforce safety margin* (tab. E.9). The *Planned trajectory* (red) changes to avoid *Storm*.
2. *Avoidance start* (fig. 7.13b) - when UAS reaches optimal avoidance distance, the *navigation reach set* is constrained, forcing UAS to perform an evasive maneuver.
3. *Avoidance end* (fig. 7.13c) - navigation space is no longer constrained when the *minimal safe distance/heading* is achieved.
4. *Waypoint reached* (fig. 7.13d) - standard waypoint navigation procedure was used in this case.

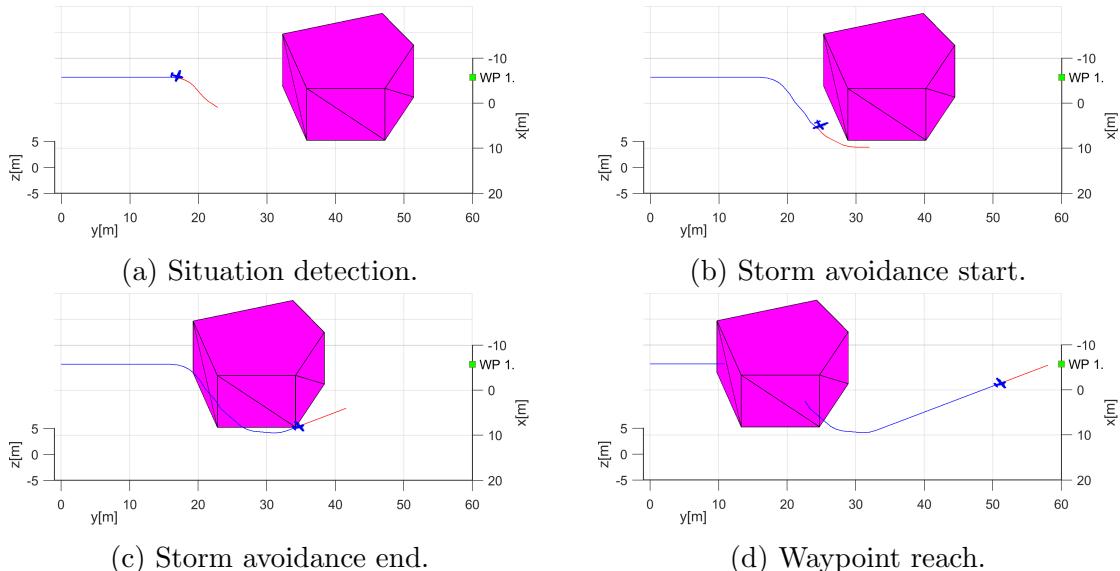


Figure 7.13: Test scenario for *Storm* (Dynamic hard constraint).

Distance to Body/Safety Margin Evolution: The *body margin* (red line) and *safety margin* (yellow line) and *UAS distance to storm center* (blue line) evolution over *UTM time* (x-axis) are given in (fig. 7.14) The *body* and *safety* margin was changing according to the mutual position of the *storm* and the *UAS* (see tab. 7.20).

The acceptance criteria for the *hard constraint avoidance* and *soft constraint avoidance* have been fulfilled.

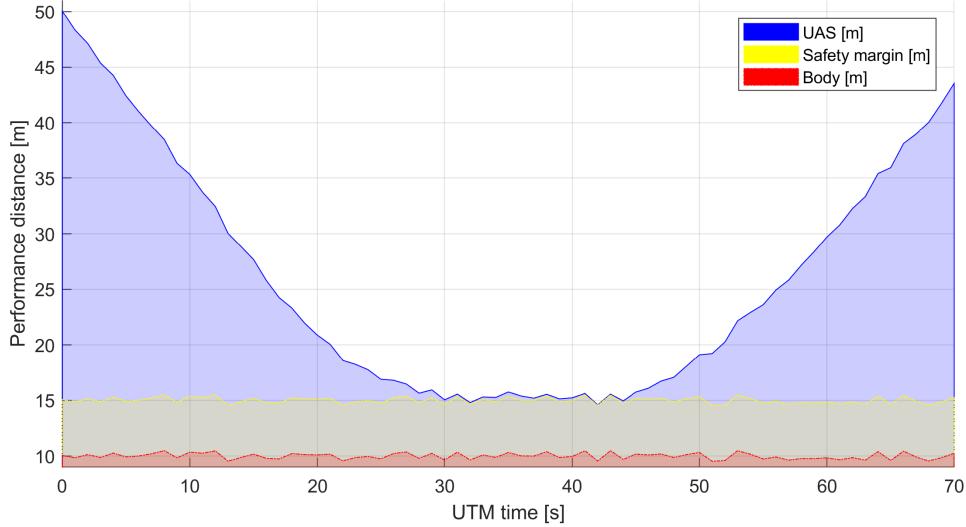


Figure 7.14: Distance to body/safety margin evolution for *Storm scenario*.

Distance to Body/Safety Margin Peaks: A *hard constraint* of *body margin* was not breached, because the $distance(UAS(t), stormBody(t))$ was all time greater than 0. Thus the *UAS* stayed well clear from *Storm*. The summary (tab. 7.21) shows that the *minimal body margin distance* was 5.0335 m, which proves *avoidance of hard constraint*.

A *soft constraint* represented as a *safety margin* (protective coating around storm body) was not breached, because the $distance(UAS(t), stormBody(t)) - safetyMargin(t)$ was all time greater than 0. The summary (tab. 7.21) show that the *minimal safety margin distance* was 0.0355 m, which proves *avoidance of soft constraints*.

| Parameter | UAS 1 | |
|---------------------------|-------|---------|
| Distance to Safety Margin | min | 0.0355 |
| | max | 34.9934 |
| Distance to Body Margin | min | 5.0355 |
| | max | 39.9934 |

Table 7.21: Distance to body/safety margin peaks for *Storm scenario*.

Path Tracking Performance: The *path tracking* (solid blue line) of *reference trajectory* (green dashed line) between *starting waypoint* (green square marked "S") and *final waypoint* (green square marked "1") is portrayed in (fig. 7.15). The *UAS* executes *horizontal right-side avoidance* of the *Storm* as is preferred.

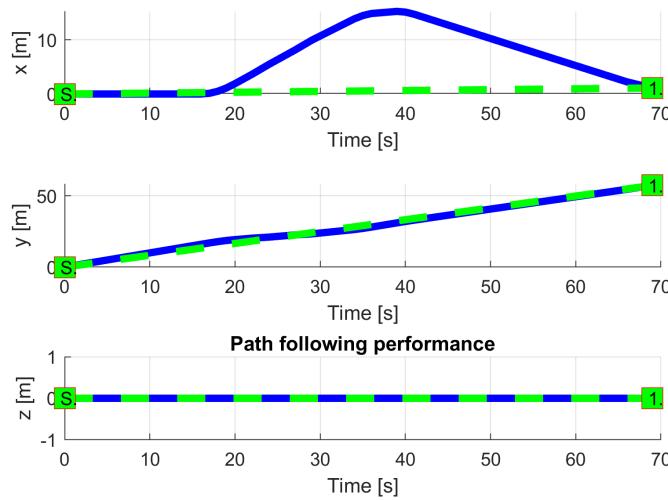


Figure 7.15: *Storm* avoidance scenario path tracking.

Path Tracking Deviations: *Deviations* (tab. 7.22) are in expected ranges considering the mission plan (tab. 7.19) and *body* and *safety* margins (tab. 7.20).

| Param. | UAS 1 |
|--------------|------------------|
| | \mathcal{WP}_1 |
| $\max x $ | 15.26 |
| $\max y $ | 1.32 |
| $\max z $ | 0 |
| $\max dist.$ | 15.76 |

Table 7.22: Path tracking properties for *Storm* scenario.

Computation Load: The *computation load* for *scenario* (fig.7.16) shows used time (y-axis) over decision frame (x-axis).

The *computation time* is low; it only increases slightly during avoidance maneuver.

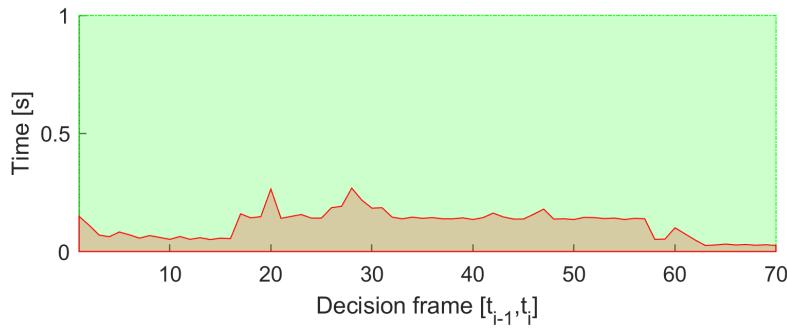


Figure 7.16: Computation time for *Maze* scenario.

7.3.5 Emergency Converging

Scenario: Two *UAS* are flying in *uncontrolled airspace* (altitude ≤ 500 ft. Above the Ground Level) with missions defined in (tab. 7.23). Both *UAS* are in the *Navigation*

mode with active *ADS-B-In/Out*, receiving position notification from each other. Cruising altitude is sufficient for horizontal separation (50-100 ft. Above the Ground Level). *Horizontal separation* is the preferred separation type for both *UAS*.

| UAS | Position | | \mathcal{WP}_1 |
|-----|----------------|----------------------------------|------------------|
| | $[x, y, z]$ | $[\theta, \varpi, \psi]$ | |
| 1 | $[0, 20, 0]^T$ | $[0^\circ, 0^\circ, 0^\circ]^T$ | $[40, 20, 0]^T$ |
| 2 | $[20, 0, 0]^T$ | $[0^\circ, 0^\circ, 90^\circ]^T$ | $[20, 40, 0]^T$ |

Table 7.23: Mission setup for *Emergency converging* scenario.

Note. *Collision point* is expected at $\mathcal{C} = [20, 20, 0]^T$. The *angle of approach* is 90° which classifies the situation as *Converging maneuver* (fig. 6.27).

Main Goal: Show two *non-cooperative* UAS avoidance capability for *Converging maneuver* scenario in *uncontrolled airspace*.

Acceptance criteria:

1. *Proper mode invocation* - when an intruder intersects the UAS with *Right of the Way* navigation grid, both UAS will switch into *Emergency Avoidance Mode*.
2. *Minimal safety margin distance* $\geq 0m$.
3. *Each UAS* will reach own goal waypoint (tab. 7.23).

Testing setup: The *standard test setup* for each UAS defined in (tab 7.2, 7.3, 7.4, 7.5, 7.6) is used with the following without parameter override.

Intruder intersection model has been chosen depending on UAS (tab. 7.24). Each UAS is equipped with *ADS-B In/Out* sensor obtaining/distributing the following information:

1. *Position* - in operational section coordinate frame.
2. *Velocity* - vector representation in the given coordinate frame.
3. *Class size* - class body radius based on UAS propulsion and size.
4. *Safety margin set* - set of safety margins for different collision cases.

Avoidance parameters for the *Emergency converging scenario* are given in (tab. 7.24). Each UAS has the same speed set to $1ms^{-1}$. Second UAS has the *Right of Way*.

The *safety margin* is considered as sum of both participants *near miss margins*. In this case, the default safety margin is considered as $1.2 m$.

| UAS | Parameters | | | Margins | | Separation |
|-----|------------|----------------|-------|---------|--------|------------|
| | velocity | intruder model | ROW | body | safety | |
| 1 | 1 | body + spread | false | 0.3 | 0.6 | horizontal |
| 2 | 1 | body + spread | true | 0.3 | 0.6 | horizontal |

Table 7.24: Avoidance parameters for *Emergency converging* scenario.

Note. Both UAS are using body (app. C.2) and spread (app. C.3) intersection models, reflecting both body volume and maneuverability of intruder. Both UAS have preferred separation mode as *horizontal*, typical for planes.

Simulation Run: Notable moments from the simulation run (fig. 7.17) are following

1. *Detection* (fig. 7.17a) - Intruder (UAS2 cyan) is approaching (UAS 1 blue) from the right side, Intruder (UAS2 cyan) has the right of way, because of $70^\circ \leq \text{angleOfApproach} < 130^\circ$. *Intruder intersection model* (for UAS 2) is created and propagated in *avoidance grid* (for UAS 1).
2. *Start Converging* (fig. 7.17b) - when *UAS 2 (cyan) parametric intruder intersection model* disables *trajectories*, converging maneuver for UAS 1 (blue) starts.
3. *Near miss case* (fig. 7.17c) - UAS 1 (blue) to UAS 2 (cyan) closest distance. The safety margin for *near miss* has not been breached. The safety margin for *well clear* in uncontrolled airspace is invalid.
4. *Waypoint reached* (fig. 7.17d) - the intruder intersection model for *UAS 2 (cyan)* is removed from UAS 1 (blue) *avoidance grid* after *converging maneuver competition*, standard navigation procedure is applied afterward.
5. Note that *UAS 2 (cyan)* has the *Right of way* in (tab. 7.24).
6. Note that *UAS 1 (blue)* used only horizontal separation (priority) in (fig. 7.19a).

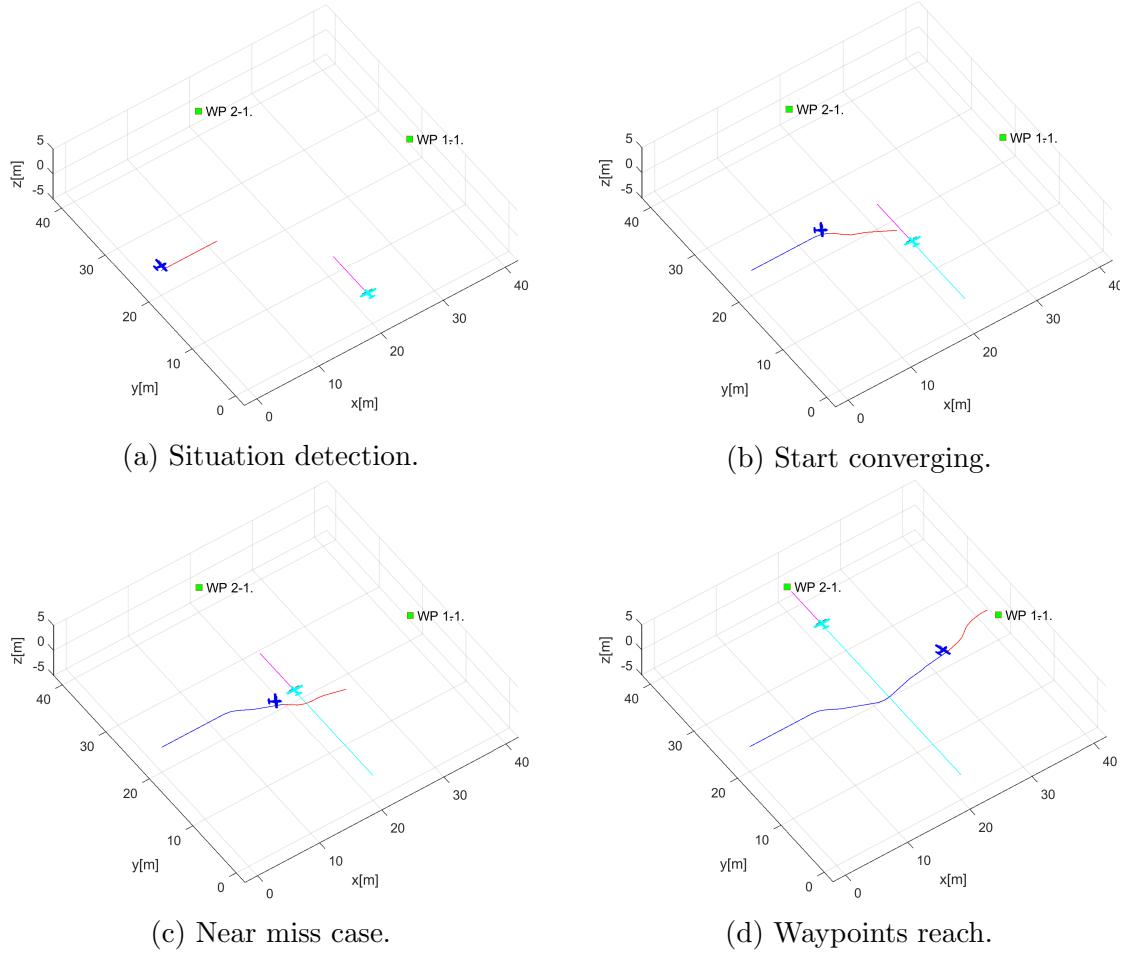


Figure 7.17: Test scenario for *Emergency converging* (Intruder avoidance).

Distance to Safety Margin Evolution: There is a need to compare the mutual distance between both UAS (y-axis [m]) and its evolution over UTM time (x-axis [s]). The *mutual distance* of *UAS 1* to *UAS 2* is given by *blue line*. The *Safety margin* value is denoted by the red line at a *constant value* of 1.2 m.

The *Proper avoidance Invocation* is shown when UAS systems are getting closer to each other, and they enter (Emergency Avoidance Mode) to provide *active separation*. The *Mutual distance evolution* (blue line) does not cross *safety margin* (red line).

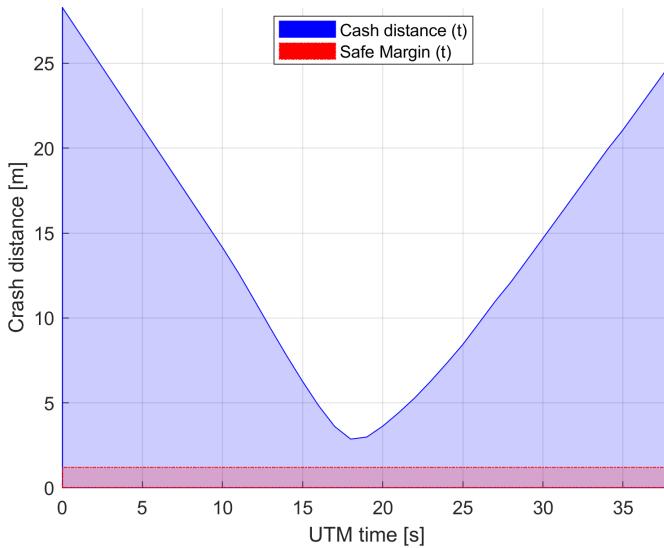


Figure 7.18: Distance to safety margin evolution for *emergency converging scenario*.

Distance to Safety Margin Peaks: Minimal and Maximal mutual distance to safety margin is summarized in (tab. 7.25). The closest to the collision are UAS systems when the distance to safety margin is 1.6676m.

The *minimal distance to safety margin* ≥ 0 which means that the *safety acceptance criterion* is fulfilled.

| UAS: | | 1-2 |
|---------------------------|-----|---------|
| Distance to Safety Margin | min | 1.6676 |
| | max | 27.0843 |

Table 7.25: Distance to safety margin peaks for the *emergency converging scenario*.

Path Tracking Performance: All waypoints (green numbered squares) for both UAS have been reached (fig. 7.19). *Reference trajectories* (green dashed lines), between the initial position (green square marked S) and goal waypoint (green square marked 1) are split into three XYZ values with respective figures. The tracked value is on y-axis [m] and time on x-axis [s]. The blue lines represent real parameter evolution over time.

Following observations can be made from path tracking (fig. 7.19) and preferred separations (tab. 7.24):

1. UAS 1 (fig. 7.19a) is using *horizontal separation* (y-axis). The UAS diverges from the reference trajectory to minimum necessary time.
2. UAS 2 (fig. 7.19b) has the right of way and is not using any active avoidance mechanism.

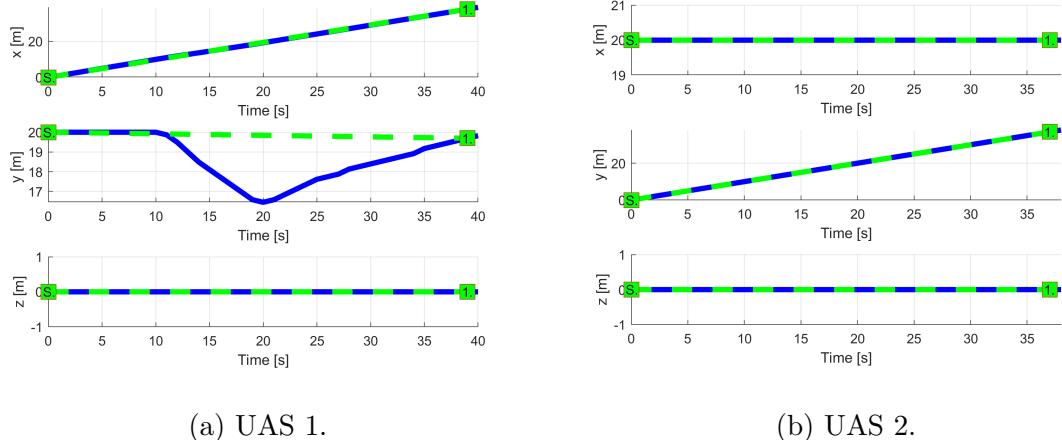


Figure 7.19: *Trajectory tracking for Emergency converging test case.*

Path Following Deviations: *Deviations* (tab. 7.26) are in expected ranges considering the *mission plans* (tab. 7.23) and *separation safety margin* (tab. 7.24).

| Param. | UAS 1 | UAS 2 |
|--------------|------------------|------------------|
| | \mathcal{WP}_1 | \mathcal{WP}_1 |
| $\max x $ | 0 | 0 |
| $\max y $ | 3.25 | 0 |
| $\max z $ | 0 | 0 |
| $\max dist.$ | 3.25 | 0 |

Table 7.26: Path tracking properties for the *Emergency converging* scenario.

Computation Load: The *computation load* for *scenario* (fig.7.20) shows used time (y-axis) over decision frame (x-axis).

The *computation time* is increased only for UAS 1 during the avoidance period. The UAS 2 remains unaffected because it has the right of way.

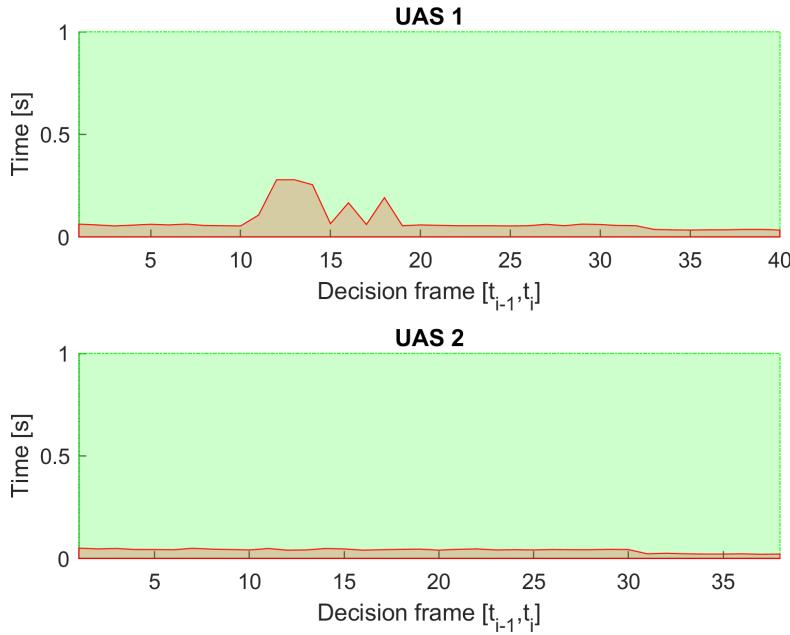


Figure 7.20: Computation time for *Emergency converging* scenario.

7.3.6 Emergency Head-On

Scenario: Two *UAS* systems are flying in *uncontrolled airspace* ($\text{altitude} \leq 500 \text{ ft. Above the Ground Level}$) with missions defined in (tab. 7.27). Both *UAS* are in the *Navigation mode* with active *ADS-B-In/Out*, receiving position notifications from each other. Cruising altitude is sufficient for horizontal separation (50-100 ft. Above Ground Level). *Horizontal separation* is preferred mode for both *UAS*.

| UAS | Position | | \mathcal{WP}_1 |
|-----|-----------------|-----------------------------------|------------------|
| | $[x, y, z]$ | $[\theta, \varpi, \psi]$ | |
| 1 | $[0, 20, 0]^T$ | $[0^\circ, 0^\circ, 0^\circ]^T$ | $[40, 20, 0]^T$ |
| 2 | $[40, 20, 0]^T$ | $[0^\circ, 0^\circ, 180^\circ]^T$ | $[0, 20, 0]^T$ |

Table 7.27: Mission setup for *Emergency head-on* scenario.

Note. *Collision point* is expected at $\mathcal{C} = [20, 20, 0]^T$. The *angle of approach* is 180° which classifies the situation as *Head-on maneuver* (fig. 6.25).

Main Goal: Show two *non-cooperative* *UAS* avoidance for *Head-on approach scenario* in *uncontrolled airspace*.

Acceptance criteria:

1. *Proper mode invocation* - when an intruder intersects the opposing *UAS* Navigation grid, bot intruder and *UAS* will switch to *Emergency Avoidance Mode*. None of the *UAS* have the *Right of Way*.

2. *Minimal Safety Margin distance* $\geq 0m$. That means the mutual distance of both *UAS centers* does not go below-given *safety margin*.
3. *Both UAS* will reach own goal waypoint (tab. 7.27).

Testing setup: The *standard test setup* for each UAS defined in (tab 7.2, 7.3, 7.4, 7.5, 7.6) is used with the following without parameter override.

Intruder intersection model has been chosen depending on UAS (tab. 7.28). Each UAS is equipped with *ADS-B In/Out* sensor obtaining/distributing the following information:

1. *Position* - in operational section coordinate frame.
2. *Velocity* - vector representation in the given coordinate frame.
3. *Class size* - class body radius based on UAS propulsion and size.
4. *Safety margin set* - set of safety margins for different collision cases.

Avoidance parameters for the *Emergency head-on scenario* are given in (tab. 7.28). Each UAS has the same speed set to $1ms^{-1}$. None of them have the *Right of Way*.

The *safety margin* is considered as a sum of both participants *near miss margins*. In this case, the default safety margin is considered as $1.2 m$.

| UAS | Parameters | | | Margins | | Separation |
|-----|------------|----------------|-------|---------|--------|------------|
| | velocity | intruder model | ROW | body | safety | |
| 1 | 1 | body (timed) | false | 0.3 | 0.6 | horizontal |
| 2 | 1 | body (timed) | false | 0.3 | 0.6 | horizontal |

Table 7.28: Avoidance parameters for *Emergency head on* scenario.

Note. Both UAS are using body (app. C.2) intersection model, reflecting both body volume along the expected trajectory. Both UAS have a preference for *horizontal* separation mode, typical for planes.

Simulation Run: Notable moments from the simulation run (fig. 7.21) are the following:

1. *Situation detection* (fig. 7.17a) - UAS 1 (blue) is approaching UAS 2 (cyan) with $130^\circ \leq angleOfApproach \leq 180^\circ$, this is considered head-on approach. Head-on approach gives the *right of the way* neither to *UAS 1* nor *UAS 2*. An *intruder intersection model* for opposite UAS is created in respective *avoidance grids*. *Head on emergency avoidance* starts independently in each UAS without intruders coordination. First *avoidance maneuver* is invoked when the *intruder intersection model* constraints any trajectory in the *avoidance grid*. When this happens *Navigation mode* switch to the *Emergency avoidance mode*.

2. *Before near miss* (fig. 7.21b) - both *UAS* are in *emergency avoidance mode*, sticking to right side avoidance maneuver.
3. *Near miss case* (fig. 7.21c) - UAS 1 to UAS 2 closest distance. The safety margin for *near miss* has not been breached. The safety margin for *well clear* in uncontrolled airspace is invalid. Both UAS are using also *Horizontal separation* to avoid each other, *Emergency avoidance mode* is switched to the *Navigation mode* when the risk of an *aerial clash* is voided.
4. *After near-miss* (fig. 7.21d) - both *UAS* are tracking back to respective waypoint, correcting *altitude* (Z-axis in (fig. 7.23)) first.
5. Note *Collision point* was expected at $\mathcal{C} = [20, 20, 0]^T$
6. Note *Both UAS* used *horizontal* (primary), *vertical* (secondary) separation (fig 7.23).
7. Note *Both UAS* decision times were *synchronized*, this is not an assumption, but it shows critical performance. Usually, safety margin is bloated for (eq.H.8).

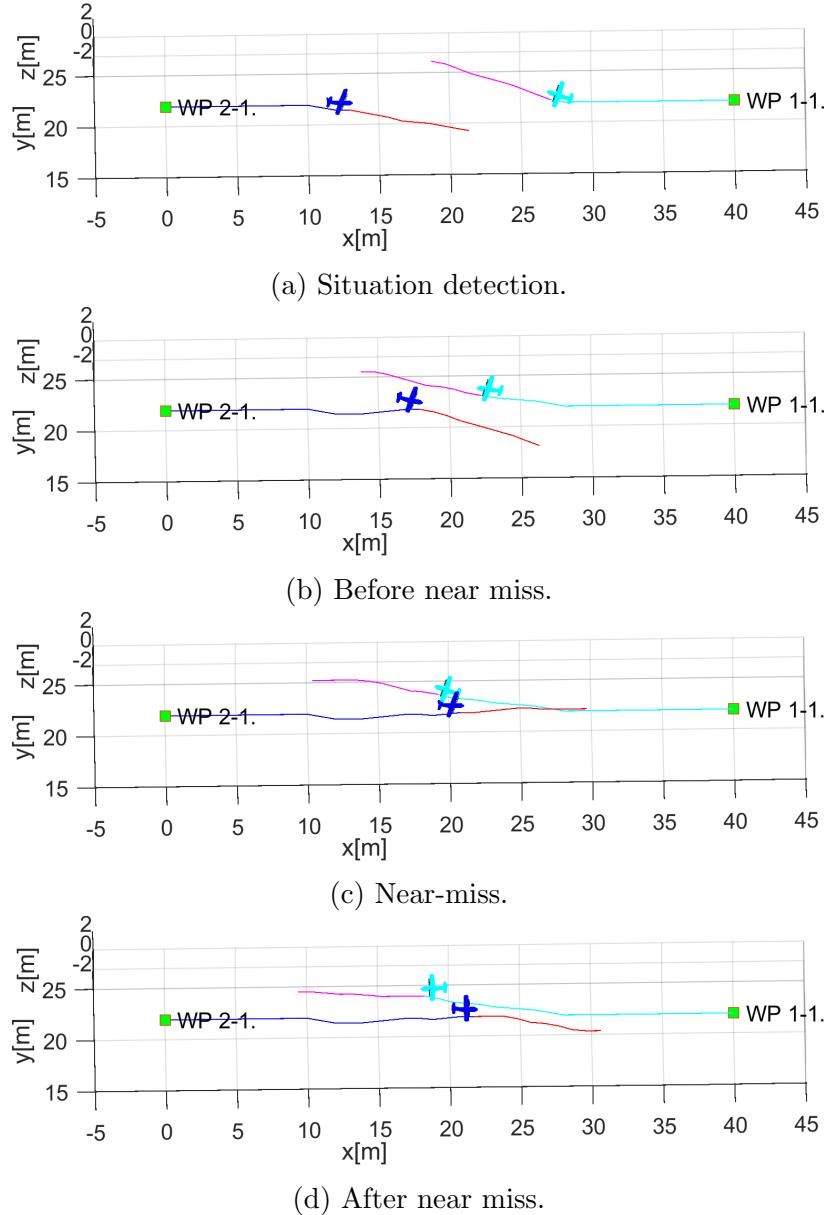


Figure 7.21: Test scenario for *Emergency head-on approach* (Intruder avoidance).

Distance to Safety Margin Evolution: There is a need to compare the mutual distance between both UAS (y-axis [m]) and its evolution over synchronized *UTM time* (x-axis [s].) The *mutual distance* between bodies of *UAS 1, UAS 2* (blue line) compared to *Safety Margin* (red line) is given in (fig. 7.22). The *Safety Margin* value was constant for all time at value 1.2 m which is double of *Near Miss Margin for UAS 1 UAS 2*.

The proper *Avoidance Invocation* is shown when *UAS* systems are getting closer to each other, and they start their *separation phase* (Emergency Avoidance Mode switch). The mutual distance (blue line) does not cross *safety margin* (red line).

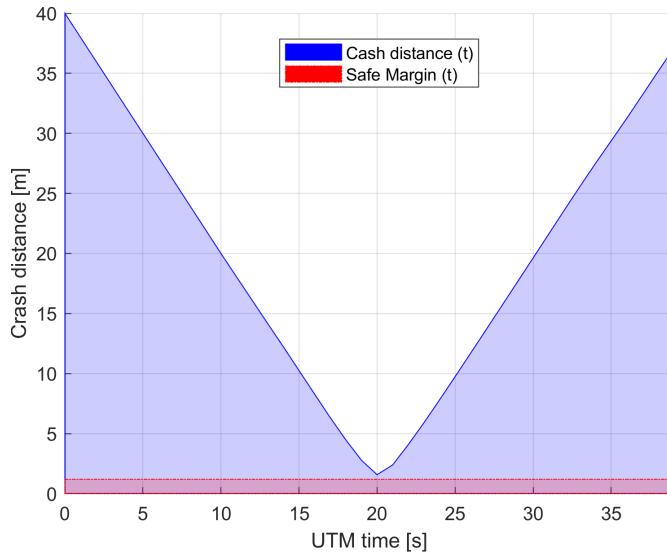


Figure 7.22: Distance to safety margin evolution for *emergency head-on scenario*.

Distance to Safety Margin Peaks: Minimal and Maximal mutual distance to safety margin is summarized in (tab. 7.29). The closest to the collision are UAS systems when the *distance to safety margin* is 0.3824m.

The *minimal distance to safety margin* ≥ 0 which means that the *safety acceptance criterion* is fulfilled.

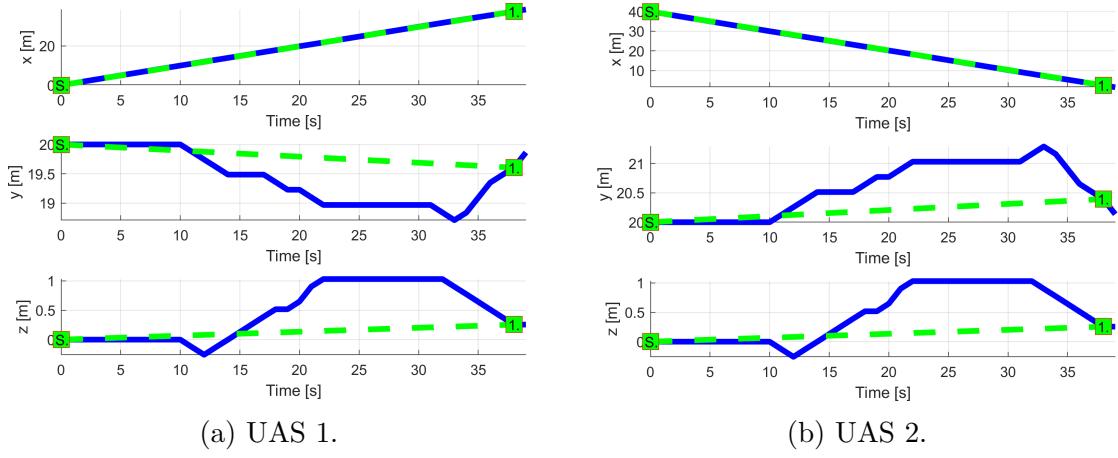
| UAS: | | 1-2 |
|---------------------------|-----|---------|
| Distance to Safety Margin | min | 0.3824 |
| | max | 38.8000 |

Table 7.29: Distance to safety margin peaks for *Emergency head-on scenario*.

Path Tracking Performance All waypoints (green numbered squares) for both UAS have been reached (fig. 7.23). *Reference trajectories* (green dashed lines), between the initial position (green square marked S) and goal waypoint (green square marked 1) are split into three XYZ values with respective figures. The tracked value is on y-axis [m] and time on x-axis [s]. The blue lines represent real parameter evolution over time.

Following observations can be made from path tracking (fig. 7.23) and preferred separations (tab. 7.28):

1. UAS 1 (fig. 7.23a) is using horizontal separation going to the right (y-axis) and a little bit up (z-axis).
2. UAS 2 (fig. 7.23b) is using horizontal separation going to the right (left in GCS, y-axis) and a little bit up (z-axis).

Figure 7.23: *Trajectory tracking for Emergency head-on test case.*

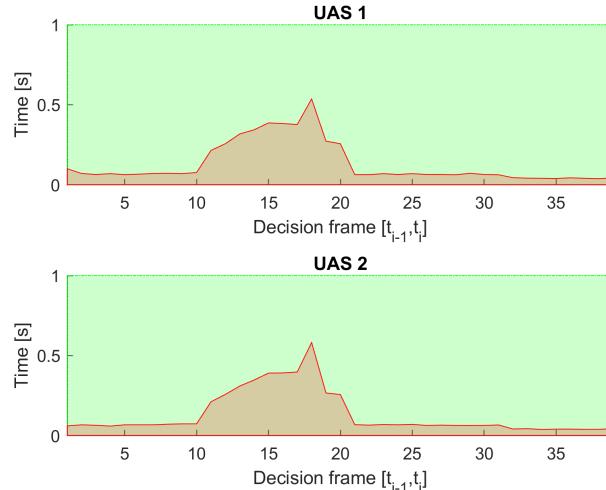
Path Following Deviations: *Deviations* (tab. 7.30) are in expected ranges considering the *mission plans* (tab. 7.27) and *separation safety margins* (tab. 7.28).

| Param. | UAS 1 | UAS 2 |
|--------------|--------|--------|
| | WP_1 | WP_1 |
| $\max x $ | 0.05 | 0.06 |
| $\max y $ | 1.37 | 1.48 |
| $\max z $ | 1.03 | 1.05 |
| $\max dist.$ | 1.39 | 1.52 |

Table 7.30: Path tracking properties for *Emergency head-on* scenario.

Computation Load: The *computation load* for *scenario* (fig.7.20) shows used time (y-axis) over decision frame (x-axis).

The *computation time* is increased only during the *avoidance phase*. The *load* is symmetric for both UAS systems.

Figure 7.24: Computation time for *Emergency head-on* scenario.

7.3.7 Emergency Mixed Head-On with Converging

Scenario: Four UAS are flying in *uncontrolled airspace* (altitude ≤ 500 ft. Above the Ground Level) missions defined in (tab. 7.31). All UAS are in the *Navigation mode* with active *ADS-B In*, receiving *position notifications* from each other. Cruising altitude is sufficient for *horizontal separation* (50-100 ft. Above the Ground Level).

| UAS | Position | | WP_1 |
|-----|-----------------|-----------------------------------|-----------------|
| | $[x, y, z]$ | $[\theta, \varpi, \psi]$ | |
| 1 | $[0, 20, 0]^T$ | $[0^\circ, 0^\circ, 0^\circ]^T$ | $[45, 20, 0]^T$ |
| 2 | $[40, 20, 0]^T$ | $[0^\circ, 0^\circ, 180^\circ]^T$ | $[-5, 20, 0]^T$ |
| 3 | $[20, 0, 0]^T$ | $[0^\circ, 0^\circ, 90^\circ]^T$ | $[20, 45, 0]^T$ |
| 4 | $[20, 40, 0]^T$ | $[0^\circ, 0^\circ, -90^\circ]^T$ | $[45, 20, 0]^T$ |

Table 7.31: Mission setup for the *Emergency mixed* scenario.

Note. Collision point is expected at $C = [20, 20, 0]^T$

Main Goal: Show *multiple non-cooperative intruders avoidance capability* in *uncontrolled* airspace.

Acceptance criteria:

1. Proper avoidance mode invocation - when an *intruder intersection model* impact the *Avoidance Grid*, UAS system will switch to an *Emergency avoidance mode*.
2. Minimal safety margin distance $\geq 0m$.
3. Each *UAS* will reach own goal waypoint (tab. 7.31).

Testing setup: The *standard test setup* for each UAS defined in (tab 7.2, 7.3, 7.4, 7.5, 7.6) is used with the following without parameter override.

Intruder intersection model has been chosen depending on UAS (tab. 7.32). Each UAS is equipped with *ADS-B In/Out* sensor obtaining/distributing the following information:

1. *Position* - in operational section coordinate frame.
2. *Velocity* - vector representation in the given coordinate frame.
3. *Class size* - class body radius based on UAS propulsion and size.
4. *Safety margin set* - set of safety margins for different collision cases.

Avoidance parameters for the *Emergency mixed scenario* are given in (tab. 7.32). Each UAS has different *intruder model* and separation combination. Each UAS has same the speed set to $1ms^{-1}$. None of UAS has the *Right of Way*.

The *safety margin* is considered as the sum of both participants *near miss margins*. In this case, the default safety margin is considered as $1.2 m$.

| UAS | Parameters | | | Margins | | Separation |
|-----|------------|----------------|-------|---------|--------|------------|
| | velocity | intruder model | ROW | body | safety | |
| 1 | 1 | body + spread | false | 0.3 | 0.6 | horizontal |
| 2 | 1 | body (timed) | false | 0.3 | 0.6 | vertical |
| 3 | 1 | body (timed) | false | 0.3 | 0.6 | horizontal |
| 2 | 1 | body + spread | false | 0.3 | 0.6 | vertical |

Table 7.32: Avoidance parameters for *Emergency mixed scenario*.

Note. Each *UAS* use different intruder intersection models and primary *separations* (defined in the tab. 7.32). UAS reactions are based on primary *Separation mode*, intruders intersection models this is reflected on major axial deviations in (fig. 7.27) and summarized in *path tracking deviation* (tab. 7.34).

Simulation Run: Notable moments from the simulation run (fig. 7.25) are the following:

1. *Situation detection* (fig. 7.25a) - UAS 1 (blue) is detecting UAS 2 (cyan), UAS 3 (green), and UAS 4 (black) as possible intruders. There are multiple converging and head on approaches depending on mutual positions (UAS and *angle of approach*). There exist at least one *converging case* where each UAS has the *Right of way*. Each UAS creates intruder intersection models depending on the intruder configuration (tab. 7.32). Each UAS enters into the *Emergency avoidance mode* independently, when at least one trajectory is constrained in the *avoidance grid*.
2. *Before near-miss* (fig. 7.25b) - all *UAS* are in *emergency avoidance mode*, using various *separation modes* and *intruder intersection models*. Each UAS is performing its own avoidance maneuver, constantly checking other intruders. If the same separation and the same intruder model were used, there would be a virtual round-about.
3. *After near-miss* (fig. 7.25c) - all *UAS* avoided each other which is covered in *safety margin performance* (fig. 7.26) and (tab. 7.33).
4. *Situation resolution* (fig. 7.25d) - all *UAS* returns to *Navigation mode* correcting *altitude* first and continuing to assigned waypoints.

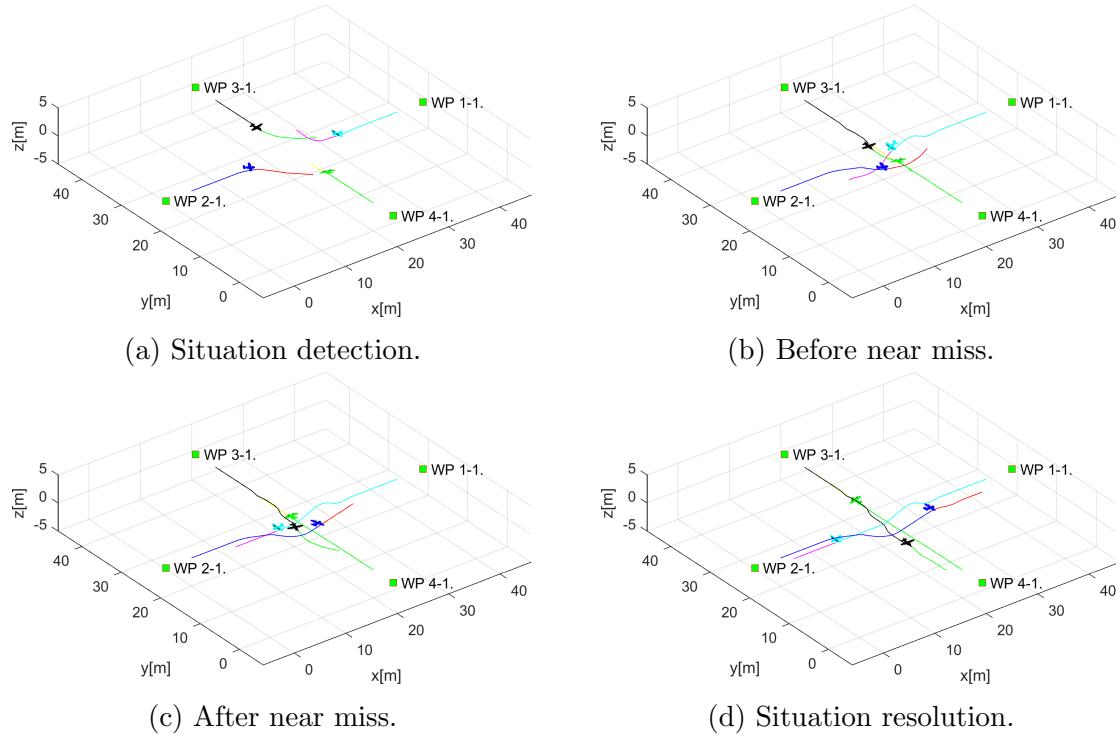


Figure 7.25: Test scenario for the *Emergency mixed* situation with the *self-separation mode*.

Distance to Safety Margin Evolution: There is a need to compare the mutual distance between each UAS. The graph (fig. 7.26) shows six figures for each *UAS systems* mutual distance (blue line) in this scenario. The *Safety Margin* (red line) (1.2 m) was not breached for any pair (case).

The *Proper avoidance invocation* is shown when UAS systems are getting closer to each other, and then they start separation phase (Emergency avoidance mode).

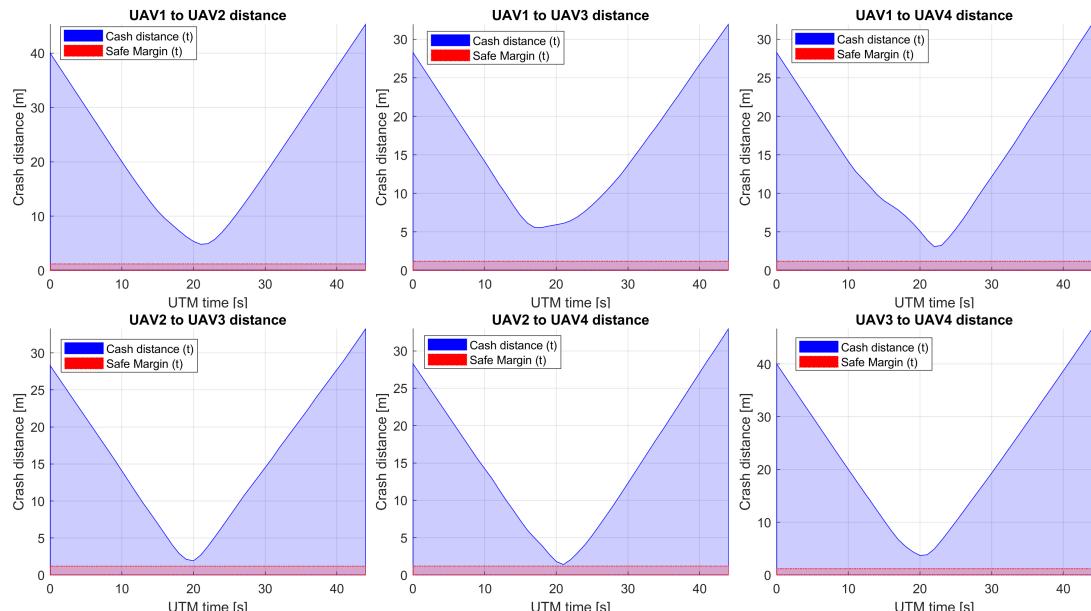


Figure 7.26: Distance to safety margin evolution for the *emergency mixed scenario*.

Distance to Safety Margin Peaks: Minimal and Maximal mutual distance to safety margin is summarized in (tab. 7.33). There is no detected breach for any combination.

The *closest to collision* is UAS pair 2 – 4 with mutual safety margin only 0.2019 m. On the other side is UAS pair 1 – 3 with mutual safety margin 4.3721 m.

The *minimal distance to safety margin* ≥ 0 which means that the *safety condition* is fulfilled.

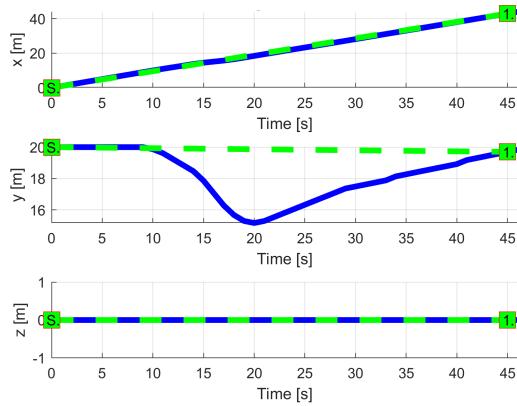
| UAS: | Distance to Safety Margin | | |
|------|---------------------------|---------|--------|
| | min | max | breach |
| 1-2 | 3.6231 | 44.0831 | false |
| 1-3 | 4.3721 | 30.7300 | false |
| 1-4 | 1.8959 | 30.7331 | false |
| 2-3 | 0.7331 | 32.0266 | false |
| 2-4 | 0.2019 | 31.7282 | false |
| 3-4 | 2.5171 | 45.4257 | false |

Table 7.33: Distance to safety margin peaks for the *emergency mixed scenario*.

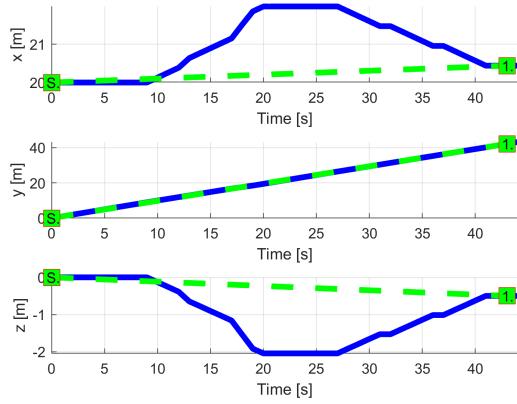
Path Tracking Performance: All waypoints (Green numbered squares) for all UAS have been reached (fig. 7.27). *Reference trajectories* (green dashed line) have been tracked by *UAS real path* (solid blue line) almost all time.

Following observations can be made from *path tracking* (fig. 7.27) and *preferred separations* (tab. 7.32):

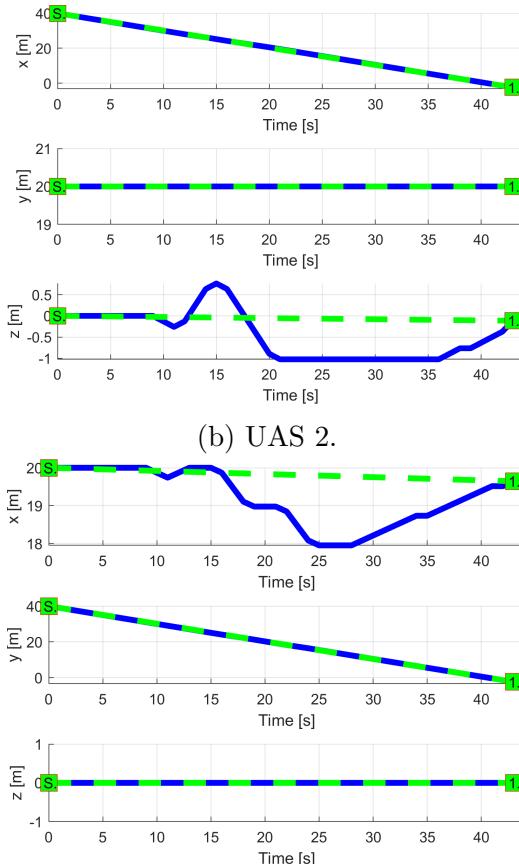
1. UAS 1 (fig. 7.27a) is using *horizontal separation* (y-axis right) having *preferred horizontal separation*.
2. UAS 2 (fig. 7.27b) is using vertical separation (z-axis up-down), having preferred vertical separation.
3. UAS 3 (fig. 7.27d) is using horizontal/vertical separation (x-right, z-down), having preferred horizontal separation. This UAS has used other than the preferred separation type.
4. UAS 4 (fig. 7.27c) is using horizontal separation (x-axis right/left), having preferred vertical separation. This UAS has used opposite separation type to preferred.



(a) UAS 1.



(c) UAS 3.



(d) UAS 4.

Figure 7.27: Trajectory tracking for the *Emergency mixed* situation test case.

Path Tracking Deviations: *Deviations* (tab. 7.34) are in expected ranges considering the *mission plans* (tab. 7.31) and *separation safety margins* (tab. 7.32).

| Param. | UAS 1 | UAS 2 | UAS 3 | UAS 4 |
|-----------|------------------|------------------|------------------|------------------|
| | \mathcal{WP}_1 | \mathcal{WP}_1 | \mathcal{WP}_1 | \mathcal{WP}_1 |
| max $ x $ | 0 | 0 | 1.98 | 2.05 |
| max $ y $ | 4.84 | 0 | 0 | 0 |
| max $ z $ | 0 | 1.23 | 2.43 | 0 |
| max dist. | 4.84 | 1.23 | 3.45 | 2.05 |

Table 7.34: Path tracking properties for the *Emergency mixed* scenario.

Computation Load: The *computation load* for scenario (fig.7.28) shows used time (y-axis) over decision frame (x-axis).

The *computation time* increases during periods of *active avoidance*. The *shortest* period of avoidance has UAS 1 and the *longest* period of avoidance has UAS 4.

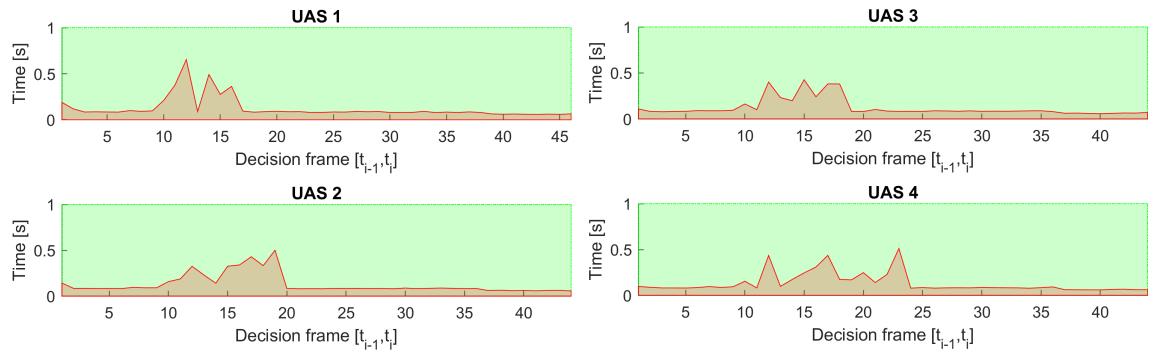


Figure 7.28: Computation time for *Emergency multiple* scenario.

7.4 Cooperative Test Cases

The *main goal* of this section is to show the operational capabilities of the *approach* under *UTM supervision*. The minimal UTM functionality set (sec. 6.7) has been implemented, including *position notifications mechanism*, *collision case calculation*, *resolution enforcement* components.

Test cases cover *well clear breach prevention*, *situation-based avoidance*, and *rules of the air enforcement*.

Coverage of *near miss situations*, *clash incidents* is given implicitly by *safety* and *body margins* (tab. 4.1).

1. *Rule-based converging* (sec. 7.4.1) covers *well clear breach* and the *converging rule of the air*, showing determinism and *UTM resolution execution*.
2. *Rule-based head-on* (sec. 7.4.2) covers *well clear breach* and the *head on rule of the air*, showing determinism and *UTM resolution execution*.
3. *Rule-based mixed head on with converging* (sec. 7.4.3) covers *well clear breach* and *head on and converging rules of the air*. The main focus is on a *virtual roundabout* concept when multiple collision cases are clustered into one avoidance maneuver.
4. *Rule-based overtake* (sec. 7.4.4) covers *well clear breach* during *overtaking* by faster UAS.

7.4.1 Rule-Based Converging

Scenario: Two *UAS* are approaching an *airway intersection* at the *same time* in *controlled airspace* (over 500 feet Above the Ground Level). The mutual position of *UAS* can be classified as *Side approach*. Following *collision hazards* are present:

1. *Active Converging Collision Hazard* - An *UAS* is approaching from the *right side*, which gives him *Right of the Way* and invokes the need to avoid *Intruder* actively.
2. *Passive Converging Collision Hazard* - An *UAS* is approaching from the *left side*, which gave us *Right of the Way* and imposes an obligation of *active avoidance* on other *UAS*.

Collision Hazards must be addressed by *UTM* service in the following manner:

1. *Each UAS* in particular *Controlled Space* periodically sends synchronized *Position Notification* messages (tab. 6.4).
2. *UTM* service receives *Position Notifications* and manages *Collision Case* (tab. 6.6) in *Controlled Space*.
3. *UTM* detects *Converging Collision Case* with *Collision Point* in the vicinity.

4. *UTM service Sends Mandate to UAS without Right of the Way and implements Normative Directive on all UAS in the area.*

Mission parameters for both UAS systems are defined in (tab. 7.35).

| UAS | Position | | \mathcal{WP}_1 |
|-----|----------------|----------------------------------|------------------|
| | $[x, y, z]$ | $[\theta, \varpi, \psi]$ | |
| 1 | $[0, 20, 0]^T$ | $[0^\circ, 0^\circ, 0^\circ]^T$ | $[40, 20, 0]^T$ |
| 2 | $[20, 0, 0]^T$ | $[0^\circ, 0^\circ, 90^\circ]^T$ | $[20, 40, 0]^T$ |

Table 7.35: Mission setup for *Rule based converging scenario*.

Assumptions: Following assumptions are valid for this test:

1. *Controlled Airspace Airworthiness* - UAS system is equipped with necessary controlled airspace equipment like ADS-B In/Out, Radar, Transponder, etc. Moreover, airworthy *UAS* can precisely follow *UTM directives* (max. 5 % deviation).
2. *C2 (Command & control) Link Established* - necessary for (*UAS* ↔ *UAS*) and (*UAS* ↔ *UTM*) communication. If *C2* link is lost the *UAS* will enter into *Emergency avoidance mode*.
3. *Decision frame synchronization with UTM* - necessary in discrete C2 environment otherwise *safety margins* needs to be *bloated*.
4. *Both UAS have identical cruising speed* - simplification impacting *UTM* service implementation. *Obstacle Avoidance Framework* can comprehend various intruders speed, with proper *UAS* directives.

Main Goal: Show possibility of *Converging situation resolution* with *forced safety margin* by *UAS Traffic Management* system. The *Obstacle Avoidance Framework based on Reach Sets* is used as a *Navigation Module*.

Acceptance Criteria: Following criteria must be met:

1. *Well Clear Condition valid for both UAS* - Both *UAS* must have *minimal required distance* from other *UAS* for all *Converging Maneuver enforcement time*.
2. *Fulfillment of UTM Directives* - Both *UAS* must stay in a *Navigation mode* for all *Converging Maneuver enforcement time*. *UAS without Right Of the Way* must stay away for the necessary time, before returning to *Original Navigation waypoint* \mathcal{WP}_1 following.

Testing Setup: The *standard test setup* for each UAS defined in (tab. 7.2, 7.3, 7.4, 7.5, 7.6) is used with following parameter override:

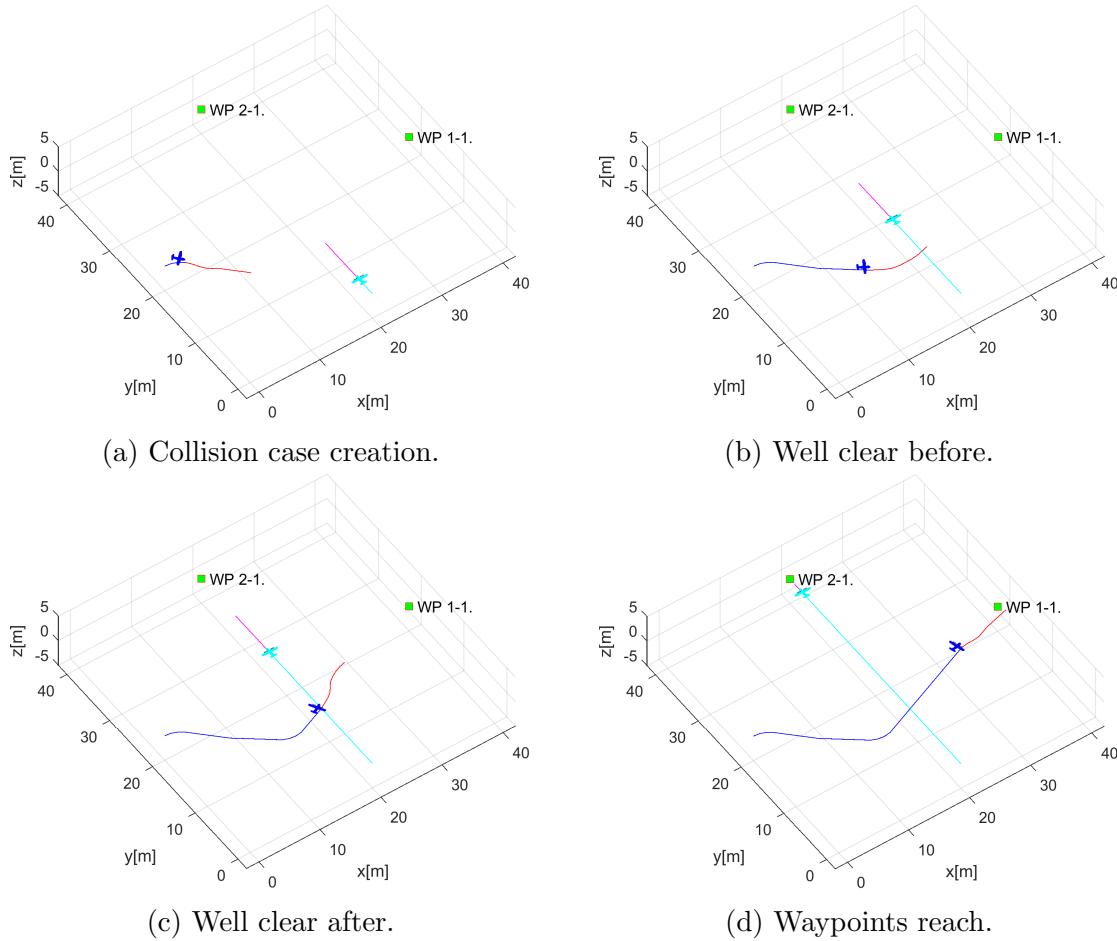
1. *Navigation grid - type - ACAS-like* with horizontal enabled *maneuvers*

This *configuration* is based on the assumption that every UAS is in *controlled airspace* in *FL450* (flight level 45000 feet Above Sea Level), without permission for a *climb or descent maneuver*. The *rule engine* is initialized in standard *Rules of the air* configuration (fig. 6.29).

There is *UTM* service for given *airspace cluster* calculating *collision cases* (tab. 6.6) based on incoming *UAS position notifications* (tab. 6.4).

Simulation Run: Notable moments from the *simulation run* (fig. 7.29) are the following:

1. *Collision Case creation* (fig. 7.29a) following events happens in this step:
 - a. Two *UAS* are approaching *airway intersection*: UAS 1 (blue) from left and UAS 2 (cyan) from the bottom.
 - b. They are going to *collide* at point $\mathcal{C} = [20, 20, 0]^T$ of *Flight Level* (elevation is 45, 000 feet Above Mean Sea Level).
 - c. UTM service notices future *Collision Situation* and creates *Collision Case*.
 - d. *Converging Directive* for 8 m from *Collision point* is issued for UAS 1 (blue) because UAS 2 (cyan) has the *Right Of the Way*.
 - e. *Keep Velocity/Heading Directive* is issued for UAS 2 (cyan) to ensure avoidance maneuver success.
 - f. UAS 1 (blue) corrects its heading according to *UTM* directive.
 - g. UAS 2 (cyan) stays on claimed course and if its necessary adjust its speed.
2. *Well clear before* (fig. 7.29b) UAS 1 (blue) check the *Collision Point* distance and keeps safe distance given by safety margin. UAS 2 (cyan) checks if there is no intruder in *Avoidance Grid* and if not, stays in *Navigation Mode*.
3. *Well clear after* (fig. 7.29c) UAS 2 (cyan) is *after Collision Point*, it can start negotiations of new speed and heading with UTM. UAS 1 (blue) is still enforced to follow *Converging Maneuver* directive until the outer boundary of *Collision Zone* is reached.
4. *Waypoints reach* (fig. 7.29d) UAS 1 (blue) leaves the outer boundary of the *Collision zone*. Leaving *Converging Maneuver Directive*. UTM closes *Collision Case*.

Figure 7.29: Test scenario for *Rule-based converging*.

Collision Case Calculation: For test scenario in (fig. 7.29) where UAS 1 (blue) is converging to avoid UAS 2 (cyan) the *Collision Case* (tab. 7.36) have been calculated.

The *Collision point* is at [20, 20, 0] in *Flight Level FL450* coordinate frame.

The *angle of approach* was evaluated as 90° which indicates *converging maneuver* in range $70^\circ \leq \text{angleOfApproach} < 130^\circ$.

The *mutual position* of UAS 1 (blue) and UAS 2(cyan) is giving the roles: *Right Of the Way* for UAS 2 (cyan) and *Converging* for UAS 1 (blue).

The *safety margin* for *Well Clear* was determined as 3m for UAS 1 and 5m for UAS 2. (Note: Well Clear Margin is usually much greater than Near Miss margin). The *Combined Case* margin which was enforced was 8m. The mutual distance cannot go below this threshold.

| Collision Case | | | | | | Margins | |
|----------------|-----|-------------|-----------------|-------------------|------------|---------|------|
| id | UAS | role | collision point | angle of approach | type | safety | case |
| 1-2 | 1 | Converging | $[20, 20, 0]^T$ | 90° | Converging | 3 | 8 |
| | 2 | Right o. W. | | | | 5 | |

Table 7.36: Collision case for *Rule-based converging* scenario.

Distance to Safety Margin Evolution: The safety margin values (well clear) (fig. 7.30) in controlled airspace are much greater than in non-controlled airspace (near miss) (fig. 7.18)

The enforced rule was (rule E.6) with parameters: Collision Point $[20, 20, 0]^T$ and *Safety Margin* 8 m as given by Collision Case (tab. 7.36).

The mutual *UAS distance* (blue line) does not go over *Safety Margin* (red line), which means UAS 1 well clear margin of 3 m and UAS 2 well clear margin of 5 m are not broken (fig. 7.30).

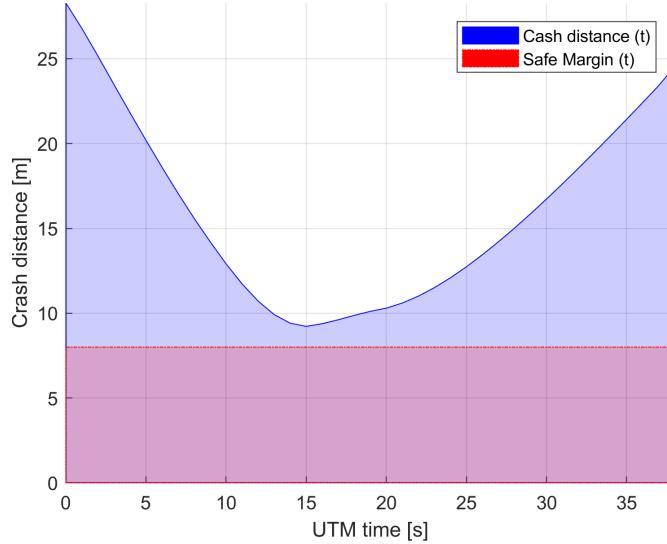


Figure 7.30: Distance to safety margin evolution for the *rule-based converging scenario*.

Distance to Safety Margin Peaks: *Distance to safety margin peaks* (tab. 7.37) represent the proximity on UAS mutual distance to *breach of well clear condition* (safety margin). The *breach of well clear condition* was not achieved. The *minimal distance to the safety margin* was 1.2240 m. The *maximal distance to safety margin* was 20.2843 m which represent distance in a time of *Collision Case Creation*.

| UAS: | Distance to Safety Margin | | |
|------|---------------------------|---------|--------|
| | min | max | breach |
| 1-2 | 1.2240 | 20.2843 | false |

Table 7.37: Distance to safety margin peaks for the *rule-based converging scenario*.

Path Tracking Performance: *Path tracking* is displayed in (fig. 7.31). The *UAS* trajectory is divided into *X, Y, Z axis tracking over UTM Time*. The *Reference Trajectory* (green dashed line) interconnect starting position of UAS (green square marked S) a goal waypoint (green square marked 1). The *Executed Trajectory* (solid blue line) reflects real UAS trajectory.

1. UAS 1. (fig, 7.31a) do steady right side *converging maneuver* (y-axis).

2. UAS 2. (fig. 7.31b) follows the reference trajectory precisely because it has the *Right Of Way*.

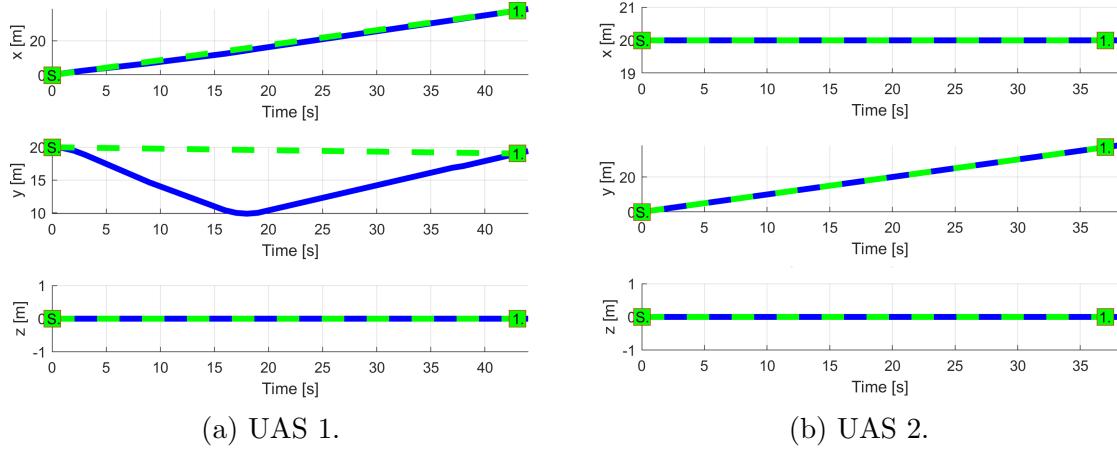


Figure 7.31: *Trajectory tracking for Rule-based converging test case.*

Path Tracking Deviations: Deviations (tab. 7.38) are in *expected ranges*, considering the *mission plans* (tab. 7.35) and *Collision Case* safety margin of 8m.

The minimal deviation distance was expected at the value of *safety margin* (8m). The maximal deviation was 10.22m which is acceptable due the space discretization, UAS dynamic, and, *dynamic decision time*.

| Param. | | |
|------------------|------------------|------------------|
| | UAS 1 | UAS 2 |
| \mathcal{WP}_1 | \mathcal{WP}_1 | \mathcal{WP}_1 |
| $\max x $ | 0 | 0 |
| $\max y $ | 10.22 | 0 |
| $\max z $ | 0 | 0 |
| $\max dist.$ | 10.22 | 0 |

Table 7.38: Path tracking properties for *Rule-based converging* scenario.

Computation Load: The *computation load* for *scenario* (fig.7.32) shows used time (y-axis) over decision frame (x-axis).

The *computation time* is slightly increased for avoiding UAS 1 during avoidance. The initial increase of computation time UAS 2 is caused by UTM communication demand.

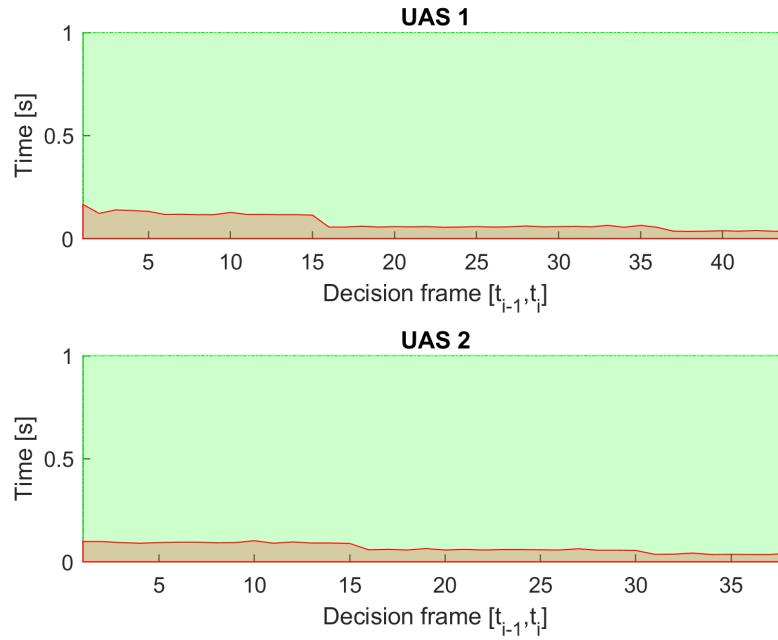


Figure 7.32: Computation time for *Rule-based converging* scenario.

7.4.2 Rule-Based Head-On

Scenario: Two *UAS* are going on the same *airway* in same *flight level* in the opposite direction in *controlled airspace* (over 500 feet Above the Ground Level). The *mutual position* of UAS can be classified as *Side Approach*. Following *collision hazard* is present:

1. *Head-on Collision Hazard* - An *UAS* is approaching from opposite direction which invokes need to avoid *Collision Point* actively.

Head on Collision Hazard must be addressed by *UTM* service in the following manner:

1. *Each UAS* in particular *Controlled Space* periodically sends synchronized *Position Notification* messages (tab. 6.4).
2. *UTM* service receives *Position Notifications* and manages *Collision Cases* (tab. 6.6) in *Controlled Space*.
3. *UTM* detects single *Head on Collision Cases* with *Collision Point* in the vicinity.
4. *UTM* service creates *Virtual Roundabout* and implements *Normative Directive* on both *UAS*.

Mission parameters for four UAS systems are defined in (tab. 7.39).

| UAS | Position | | \mathcal{WP}_1 |
|-----|-----------------|-----------------------------------|------------------|
| | $[x, y, z]$ | $[\theta, \varpi, \psi]$ | |
| 1 | $[0, 20, 0]^T$ | $[0^\circ, 0^\circ, 0^\circ]^T$ | $[45, 20, 0]^T$ |
| 2 | $[40, 20, 0]^T$ | $[0^\circ, 0^\circ, 180^\circ]^T$ | $[-5, 20, 0]^T$ |

Table 7.39: Mission setup for *Rule-based head-on* scenario.

Assumptions: Following assumptions are valid for this test:

1. *Controlled Airspace Airworthiness* - UAS system is equipped with necessary controlled airspace equipment like ADS-B In/Out, Radar, Transponder, etc. Moreover, airworthy *UAS* can precisely follow *UTM directives* (max. 5 % deviation).
2. *C2 (Command & control) Link Established* - necessary for (*UAS* ↔ *UAS*) and (*UAS* ↔ *UTM*) communication. If *C2* link is lost the *UAS* will enter into *Emergency avoidance mode*.
3. *Decision frame synchronization with UTM* - necessary in discrete *C2* environment otherwise *safety margins* needs to be *bloated*.
4. *Both UAS have identical cruising speed* - simplification impacting *UTM* service implementation. *Obstacle Avoidance Framework* can comprehend various intruders speed, with proper *UAS* directives.

Main Goal: Show possibility of *Head-on situation resolution* with *forced safety margin* by *UAS Traffic Management* system. The *Obstacle Avoidance Framework based on Reach Sets* is used as a *Navigation Module*.

Acceptance Criteria: Following criteria must be met:

1. *Well Clear Condition valid for both UAS* - Both *UAS* must have *minimal required distance* from *other UAS* for all *Virtual Roundabout enforcement time*.
2. *Fulfillment of UTM Directives* - Both *UAS* must stay in a *Navigation mode* for all *Virtual Roundabout enforcement time*. Both *UAS* must stay on *Virtual Roundabout* for the necessary time, before leaving for *Original Navigation waypoint* WP_1 .

Testing Setup: The *standard test setup* for each *UAS* defined in (tab. 7.2, 7.3, 7.4, 7.5, 7.6) is used with following parameter override:

1. *Navigation grid - type - ACAS-like with horizontal enabled maneuvers*.

This *configuration* is based on the assumption that both *UAS* is in *controlled airspace* in *FL450* (flight level 45000 feet Above Sea Level), without permission for a *climb or descent maneuver*. The *rule engine* is initialized in standard *Rules of the air* configuration (fig. 6.29).

There is *UTM* service for given *airspace cluster* calculating *collision cases* (tab. 6.6) based on incoming *UAS position notifications* (tab. 6.4).

Simulation Run: Notable moments from the *simulation run* (fig. 7.33) are the following:

1. *Collision Case creation* (fig. 7.33a) following events happens in this step:
 - a. Two UAS are on the same airway approaching each other from the opposite direction, UAS 1 (blue) from the left, UAS 2 (cyan) from the right.
 - b. They are going to *collide* at point $\mathcal{C} = [20, 20, 0]^T$ of *Flight Level* (Elevation is 45, 000 feet Above Mean Sea Level).
 - c. UTM service notices future *Collision Situation* and creates *Collision Case*.
 - d. *Virtual Roundabout* is created at *collision point* with radius 10m. UTM issues directive for both UAS to avoid collision point from different sides.
 - e. UAS 1 (blue) receives a directive to avoid *Collision Point* from the *right side* (downside in GCS). UAS 2 (cyan) receives a directive to avoid *Collision Point* from the *right side* (upside in GCS).
 - f. Both UAS enters into *Virtual Roundabout*.
2. *Well clear before* (fig. 7.33b) UAS 1 (blue) is keeping *enforced safety margin* (10 m) from *collision point* and *UAS 2 position*. The *Virtual Roundabout* is enforced until the (*Collision point*) is reached by both UAS. Both UAS stays in *Navigation Mode*.
3. *Well clear after* (fig. 7.33c) UTM notices that *Collision point level* has been reached by both UAS. UTM renounce *Directives* and enables a return to *Original Waypoint* WP_1 . Both UAS starts to converging to *Original waypoint* (because possible collision was averted).
4. *Waypoint reach* (fig. 7.33d) Both UAS reaches respective goal points.

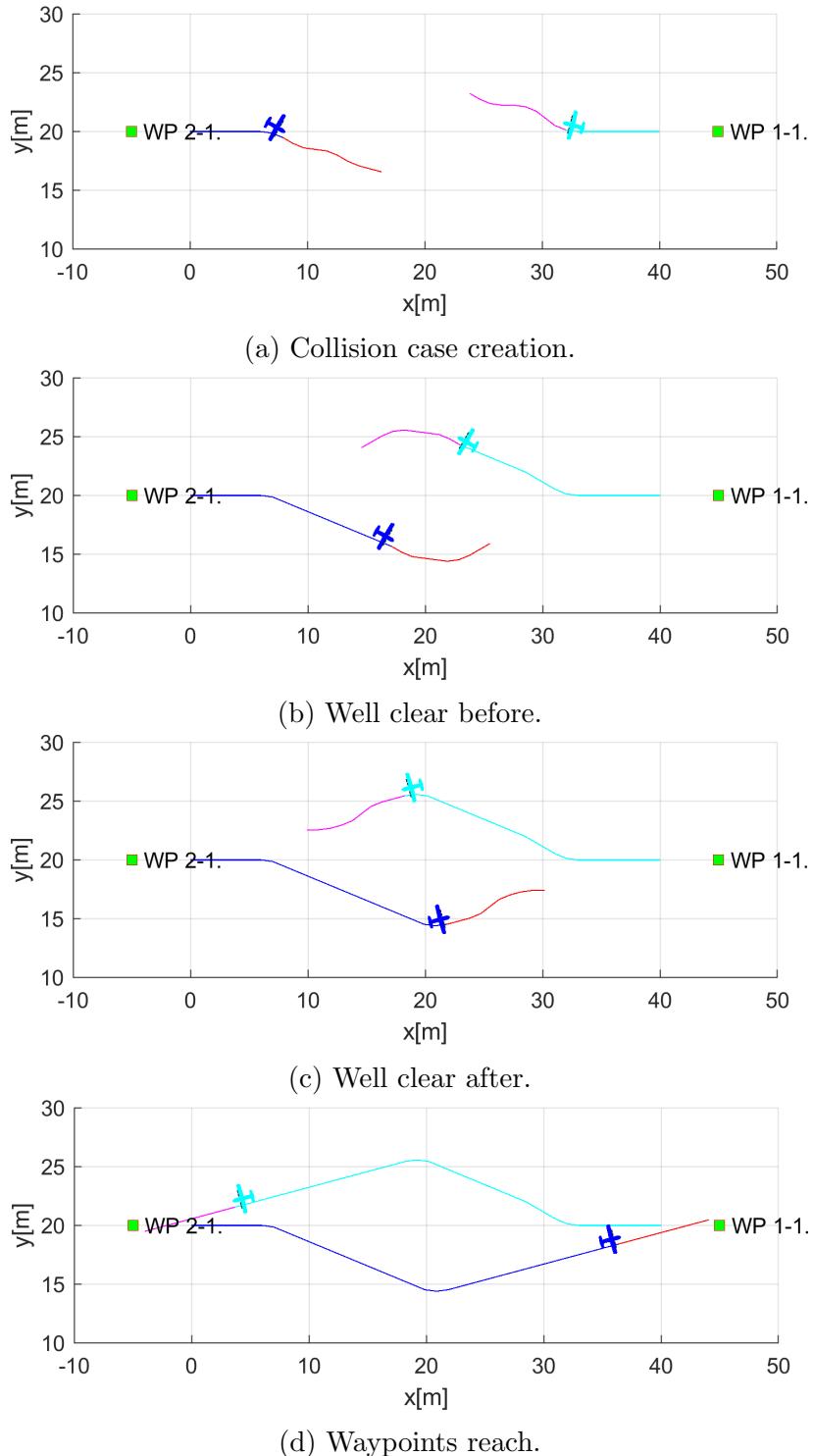


Figure 7.33: Test scenario for the *rule-based head-on approach* (virtual roundabout).

Collision Case Calculation: For test scenario in (fig. 7.33) where UAS 1 (blue) have head-on collision with UAS 2 (cyan), *Collision Case* have been calculated (tab. 7.40).

The *Collision point* is at $[20, 20, 0]^T$ in Flight Level *FL450* coordinate frame.

The *angle of approach* was evaluated as 180° which indicates *Head-on Approach* due to the $130^\circ \leq \text{angle of Approach} \leq 180^\circ$ conditions.

The *mutual position* of UAS 1 (blue) and UAS 2 (cyan) is giving the roles of *Round-about* to both UAS.

The *safety margin* for *Well Clear* was determined as 5m for UAS 1 and UAS 2. The combined *Case Margin* is 10 m, which is sum of both. The *mutual distance* cannot go below this threshold.

| Collision Case | | | | | | Margins | |
|----------------|-----|------------|-----------------|-------------------|---------|---------|------|
| id | UAS | role | collision point | angle of approach | type | safety | case |
| 1-2 | 1 | Roundabout | $[20, 20, 0]^T$ | 180° | Head on | 5 | 10 |
| | 2 | Roundabout | | | | 5 | |

Table 7.40: Collision case for the *rule-based head-on* scenario.

Distance to Safety Margin Evolution: The safety margin values (*well clear*) (fig. 7.34) in controlled airspace are much larger than in non-controlled airspace (*near miss*) (fig. 7.22).

The enforced rule was (rule E.5) with parameters: Collision Point $[20, 20, 0]^T$ and *Safety Margin* 10 m as given by Collision Case (tab. 7.40).

The mutual *UAS distance* (blue line) does not go over *Safety Margin* (red line) which means both UAS well clear margins are not broken by any means (fig. 7.33).

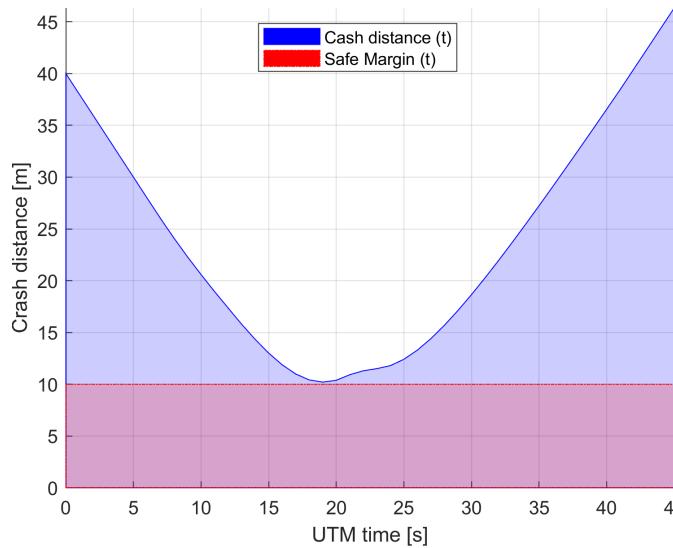


Figure 7.34: Distance to safety margin evolution for the *rule-based head-on scenario*.

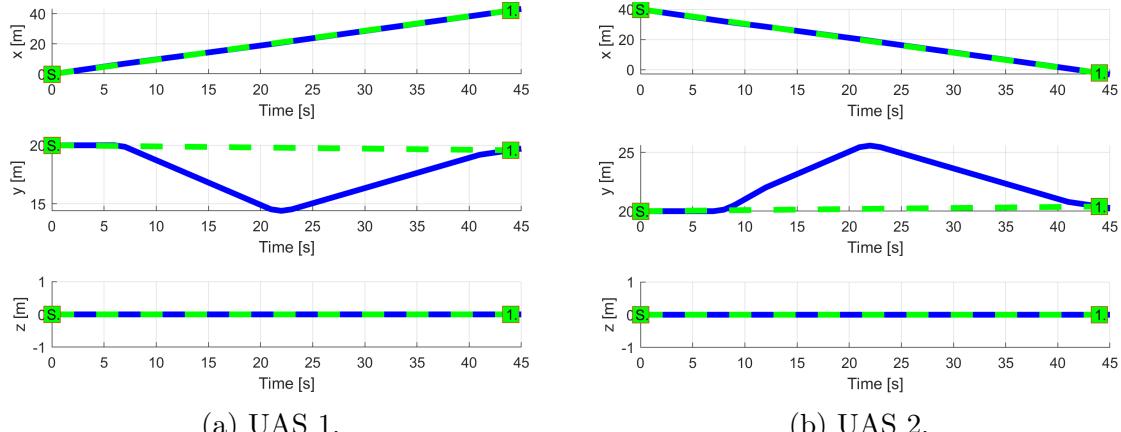
Distance to Safety Margin Peaks: Given by (tab. 7.41) represents the proximity on UAS mutual distance to *well clear condition* breach. The breach of *well clear condition* was not achieved. The *minimal distance to the safety margin* was 0.2084 m. The *maximal distance to safety margin* was 36.3253m which represents distance at *Collision Case* closing.

| UAS: | Distance to Safety Margin | | |
|------|---------------------------|---------|--------|
| | min | max | breach |
| 1-2 | 0.2084 | 36.3253 | false |

Table 7.41: *Rule-based head-on* safety margin distances.

Path Tracking Performance: *Path tracking* is displayed in (fig. 7.35). The *UAS* trajectory is divided into *X*, *Y*, *Z* axis tracking over *UTM Time*. The *Reference Trajectory* (green dashed line) interconnect starting position of UAS (green square marked S) a goal waypoint (green square marked 1). The *Executed Trajectory* (solid blue line) reflects real UAS trajectory.

1. UAS 1. (fig. 7.35a) do steady right side *roundabout maneuver* (y-axis).
2. UAS 2. (fig. 7.35b) do steady right side *roundabout maneuver* (y-axis).

Figure 7.35: *Trajectory tracking* for *rule-based head-on* test case.

Path Tracking Deviations: Deviations (tab. 7.42) are in *expected ranges*, considering the *mission plans* (tab. 7.39) and *Collision Case* safety margin of 10m.

| Param. | | |
|------------------|------------------|------------------|
| | UAS 1 | UAS 2 |
| \mathcal{WP}_1 | \mathcal{WP}_1 | \mathcal{WP}_1 |
| $\max x $ | 0 | 0 |
| $\max y $ | 5.40 | 5.40 |
| $\max z $ | 0 | 0 |
| $\max dist.$ | 5.40 | 5.40 |

Table 7.42: Path tracking properties for *rule-based head-on* scenario.

Computation Load: The *computation load* for *scenario* (fig.7.36) shows used time (y-axis) over decision frame (x-axis).

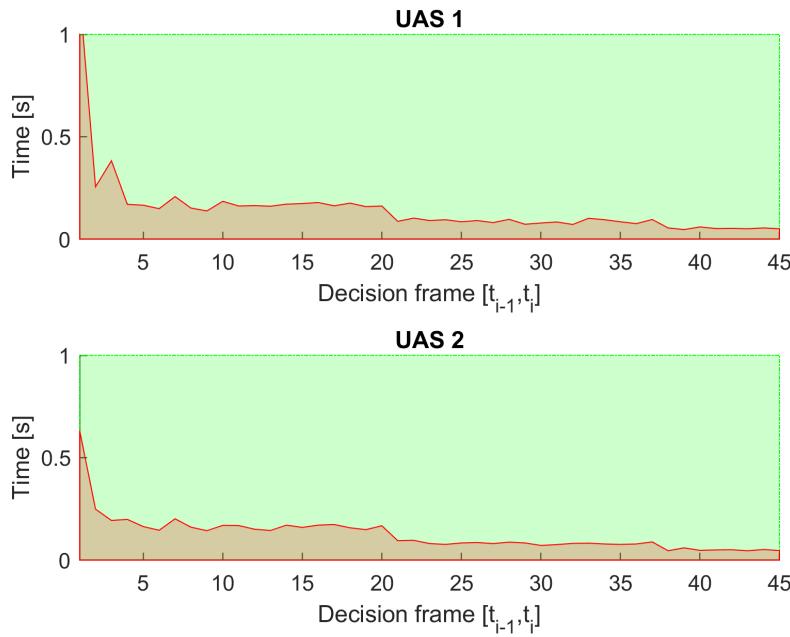


Figure 7.36: Computation time for *rule-based head-on* scenario.

7.4.3 Rule-Based Mixed Head-On with Converging

Scenario: Four *UAS* are approaching an airway *intersection* at the *same time* from *opposite direction* in *controlled airspace* (over 500 feet Above Ground Level). Each *UAS* have following *Collision Hazards*:

1. *Head on Collision Hazard* - An *UAS* is approaching from opposite direction which invokes need to avoid *Collision Point* actively
2. *Active Converging Collision Hazard* - An *UAS* is approaching from the *right side*, which gives him *Right of the Way* and invokes the need to avoid *Intruder* actively.
3. *Passive Converging Collision Hazard* - An *UAS* is approaching from the *left side*, which gave us *Right of the Way* and imposes an obligation of *active avoidance* on other *UAS*.

Note. Presented scenario is *the worst possible situation* in current *manned aviation ATM*.

Mentioned *Collision Hazards* must be addressed by *UTM* service in the following manner:

1. *Each UAS* in particular *Controlled Space* periodically sends synchronized *Position Notification* messages (tab. 6.4).
2. *UTM* service receives *Position Notifications* and manages *Collision Cases* (tab. 6.6) in *Controlled Space*.
3. *UTM* detects multiple *Collision Cases* with *Collision Points* in the vicinity.
4. *UTM* service creates *Virtual Roundabout* and implements *Normative Directive* on all *UAS* in the area.

Mission parameters for four UAS systems are defined in (tab. 7.43).

| UAS | Position | | \mathcal{WP}_1 |
|-----|-----------------|-----------------------------------|------------------|
| | $[x, y, z]$ | $[\theta, \varpi, \psi]$ | |
| 1 | $[0, 20, 0]^T$ | $[0^\circ, 0^\circ, 0^\circ]^T$ | $[45, 20, 0]^T$ |
| 2 | $[40, 20, 0]^T$ | $[0^\circ, 0^\circ, 180^\circ]^T$ | $[-5, 20, 0]^T$ |
| 3 | $[20, 0, 0]^T$ | $[0^\circ, 0^\circ, 90^\circ]^T$ | $[20, 45, 0]^T$ |
| 4 | $[20, 40, 0]^T$ | $[0^\circ, 0^\circ, -90^\circ]^T$ | $[45, 20, 0]^T$ |

Table 7.43: Mission setup for *rule-based mixed* scenario.

Assumptions: Following assumptions are valid for this test:

1. *Controlled Airspace Airworthiness* - UAS system is equipped with necessary controlled airspace equipment like ADS-B In/Out, Radar, Transponder, etc. Moreover, airworthy *UAS* has capability to precisely follow *UTM directives* (max. 5 % deviation).
2. *C2 (Command & control) Link Established* - necessary for $(\text{UAS} \leftrightarrow \text{UAS})$ and $(\text{UAS} \leftrightarrow \text{UTM})$ communication. If *C2* link is lost the *UAS* will enter into *Emergency avoidance mode*.
3. *Decision frame synchronization with UTM* - necessary in discrete C2 environment otherwise *safety margins* needs to be *bloated*.
4. *Every UAS have identical cruising speed* - simplification impacting *UTM service implementation*. *Obstacle Avoidance Framework* can comprehend various intruders speed, with proper *UAS* directives.

Main Goal: Show possibility of *Virtual Roundabout* invoked by *UTM directives* where *Obstacle Avoidance Framework based on Reach Sets* is used as a *Navigation Module*.

Acceptance Criteria: Following criteria must be met:

1. *Well Clear Condition valid for every UAS* - Each *UAS* must have *minimal required distance* from other *UAS* for all *Virtual Roundabout enforcement time*.
2. *Fulfillment of UTM Directives* - Each UAS must stay in a *Navigation mode* for all *Virtual Roundabout enforcement time*. Each *UAS* must stay on *Virtual Roundabout* for the necessary time, before leaving for *Original Navigation waypoint* \mathcal{WP}_1 .

Testing Setup: The *standard test setup* for each UAS defined in (tab. 7.2, 7.3, 7.4, 7.5, 7.6) is used with following parameter override:

1. *Navigation grid - type - ACAS-like with horizontal enabled maneuvers*

This *configuration* is based on the assumption that every UAS is in *controlled airspace* in *FL450* (flight level 45000 feet Above Sea Level), without permission for a *climb or descent maneuver*. The *rule engine* is initialized in standard *Rules of the air* configuration (fig. 6.29).

There is *UTM* service for given *airspace cluster* calculating *collision cases* (tab. 6.6) based on incoming *UAS position notifications* (tab. 6.4).

Simulation Run: Notable moments from the *simulation run* (fig. 7.37) are the following:

1. *Collision cases created* (fig. 7.37a) following events happen in this step:
 - a. Four *UAS* are approaching airways intersection: *UAS 1* (blue) from left, *UAS 2* (cyan) from right, *UAS 3* (green) from the bottom, *UAS 4* (black) from the top.
 - b. They are going to collide at point $[20, 20, 0]^T$ of *Flight level* (elevation is 45, 000 feet Above Mean Sea Level).
 - c. *UTM service* notices future *Collision Situations* and creates *Collision Cases*.
 - d. There are many *Collision Cases* in the near vicinity. The *Virtual Roundabout* is created with *Safety margin* 15 m.
 - e. The *UTM* service then sends a new *Roundabout Directives* to involved *UAS* systems.
 - f. Each *UAS* starts *Roundabout Entry Maneuver* by correcting own *Heading* and *Speed* (if its necessary).
2. *Roundabout entry* (fig. 7.37b) - Each *UAS* enters into *Virtual Roundabout* while sending *Roundabout Entrance Notification* to *UTM service*.
3. *Roundabout leave* (fig. 7.37c) following events happens in this step:
 - a. Each *UAS* when is going to approach the level of *Original Goal Waypoint* sends *Roundabout Leave Request*.
 - b. *UTM system* will check if there is *Sufficient Free Space* to leave *Virtual Roundabout*.
 - c. The *UTM Service* then issues *Virtual Roundabout Leave Approval*.
 - d. Each *UAS* will correct own heading and speed in the range of received permit.

4. *Situation resolution* (fig. 7.37d) - Each *UAS* is heading away from *Roundabout Center*, there is no active user of *Virtual Roundabout*. *UTM* will remove *Virtual Roundabout* and closes underlying *Collision Cases*. Each *UAS* will reach respective *Original Goal Waypoint*.

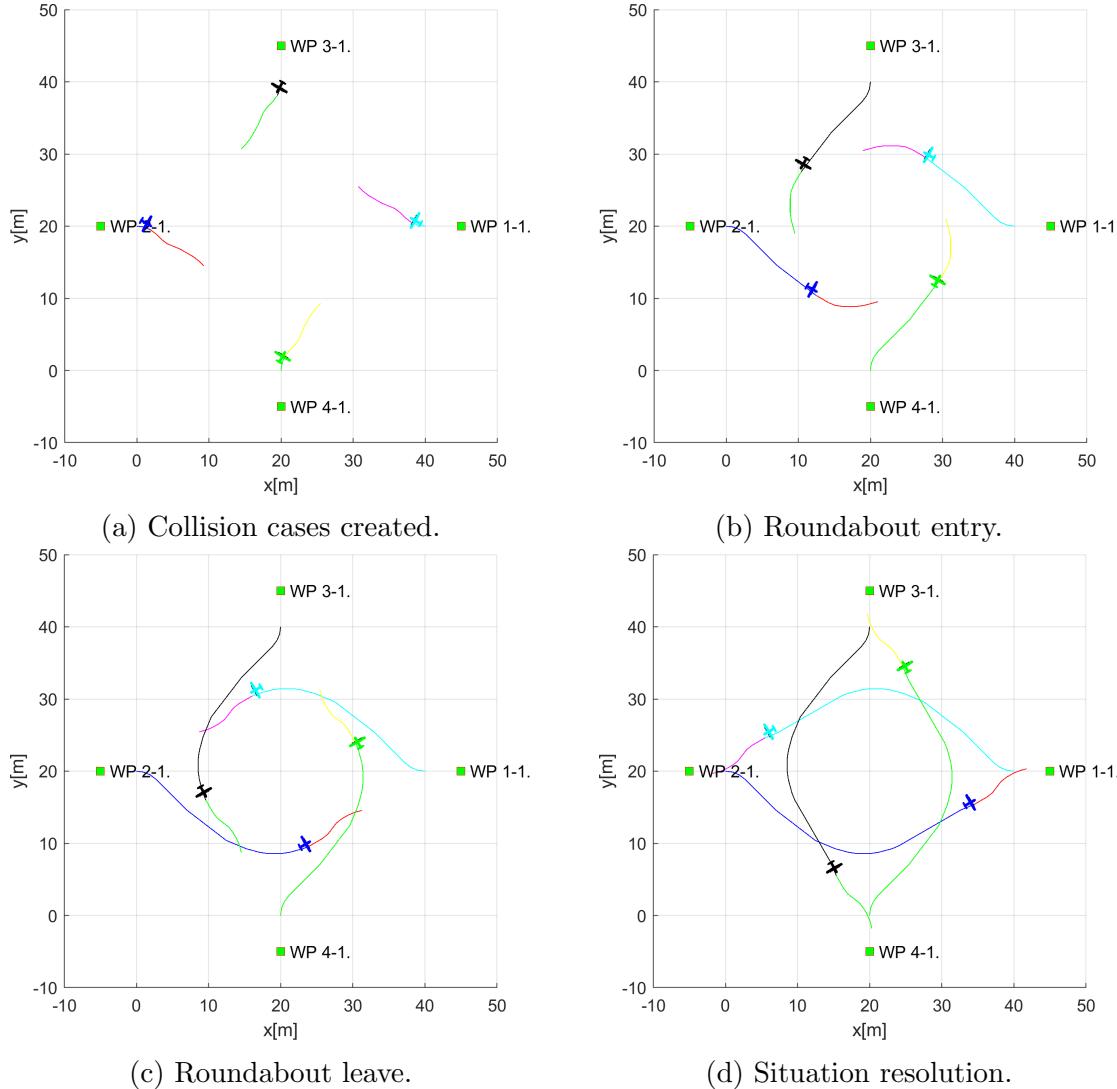


Figure 7.37: Test scenario for *rule-based mixed* situation with the *self-separation mode*.

Collision Cases Calculation: The set of original *Collision cases* is given in (tab. 7.44).

Each *UAS* has one *Head on*, *Converging passive*, *Converging active* collision hazard. For example *UAS 1* have a *head-on* with *UAS 2*, *converging passive* with *UAS 4*, *converging active* with *UAS 3*. For *UAS 2-4* check *role* in respective *Collision Cases*.

Note. *Collision cases* calculated by *UTM* are symmetric, which means that collision case for *UAS X, UAS Y* is identical to collision case calculated for *UAS Y, UAS X, X ≠ Y*.

Safety margin representing *Well Clear Margin* for single *UAS* in *Collision Case* ranges 5–8 m. *Case margin* representing the minimal mutual distance between two *UAS systems* to remain well clear ranges 12 – 15 m.

Merged Collision Case is oversimplified for demonstration purposes. *Merge Case Procedure* is out of the scope of this work due to its extent. Every *Collision Case* shares same *Collision Point* $[20, 20, 0]^T$ in flight level coordinate frame. *Merged Collision Case* type was set as *Roundabout*, due the number of collision case *attendants* is greater than 2. Each *UAS role* has been set as *Roundabout*. The enforced *safety margin* is equal to 15 m, which is the maximum of all *single collision case combined margins*.

| Collision Case | | | | | | Margins | |
|----------------|-----|------------|-----------------|-------------------|------------|---------------|------|
| id | UAS | role | collision point | angle of approach | type | safety | case |
| 1-2 | 1 | Roundabout | $[20, 20, 0]^T$ | 180° | Head on | 8 | 15 |
| | 2 | Roundabout | | | | 7 | |
| 1-3 | 1 | Converging | $[20, 20, 0]^T$ | 90° | Converging | 8 | 15 |
| | 3 | Right o.W. | | | | 5 | |
| 1-4 | 1 | Right o.W. | $[20, 20, 0]^T$ | 90° | Converging | 8 | 15 |
| | 4 | Converging | | | | 5 | |
| 2-3 | 2 | Right o.W. | $[20, 20, 0]^T$ | 90° | Converging | 7 | 12 |
| | 3 | Converging | | | | 5 | |
| 2-4 | 2 | Converging | $[20, 20, 0]^T$ | 90° | Converging | 7 | 12 |
| | 4 | Right o.W. | | | | 5 | |
| 3-4 | 3 | Roundabout | $[20, 20, 0]^T$ | 180° | Head on | 7 | 14 |
| | 4 | Roundabout | | | | 7 | |
| Merged cases | | | | | | Safety Margin | |
| id | UAS | role | collision point | type | | | |
| 1-2-3-4 | 1 | Roundabout | $[20, 20, 0]^T$ | Roundabout | | 15 | |
| | 2 | Roundabout | | | | | |
| | 3 | Roundabout | | | | | |
| | 4 | Roundabout | | | | | |

Table 7.44: Collision cases for *rule-based mixed* scenario.

Distance to Safety Margin Evolution: *Merged Collision Case Safety Margin* is 15 m, and it is valid for all *UAS mutual distances*. The simple condition for *Remain Well Clear* is:

$$\text{crashDistance}(UAS_X, UAS_Y, t) \geq 15m, X \neq Y \in \{1, 2, 3, 4\}, t \in \text{utmTime}$$

Safety Margin Performance is given in (fig. 7.38). The mutual distance (Crash Distance [m]) between two UAS is denoted as the *blue line*. The enforced safety margin for *Remain Well Clear* condition is denoted as the *red line*.

Note. Evolution of mutual crash distance is symmetric. In any case, the mutual distance goes under safety margin. Acceptance criterion for Well Clear condition is fulfilled.

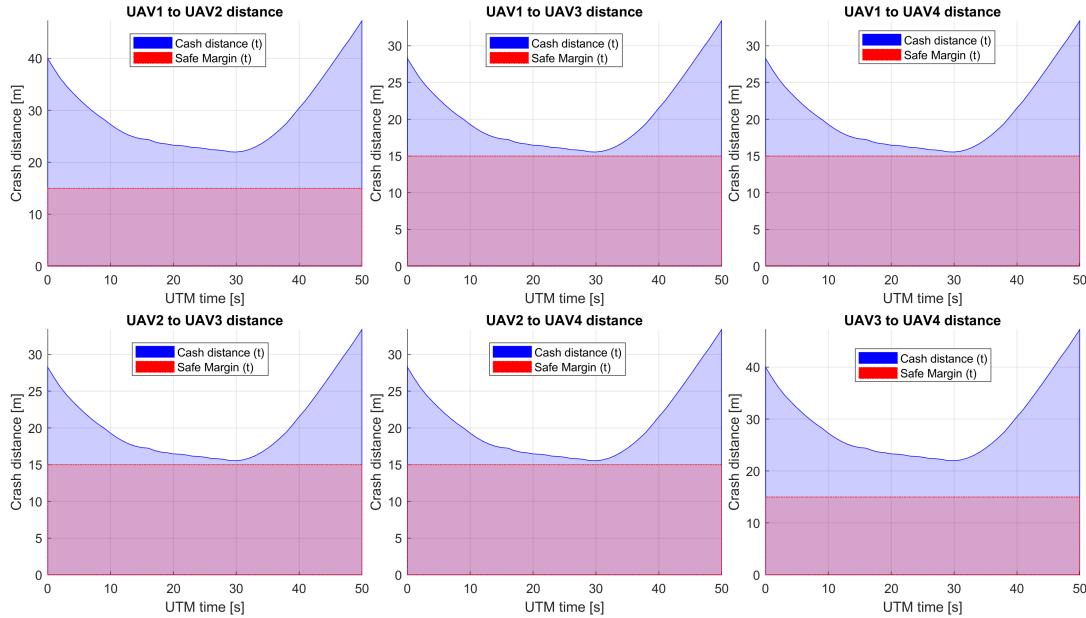


Figure 7.38: Distance to safety margin evolution for *rule-based mixed scenario*.

Distance to Safety Margin Peaks: *Distance to Safety Margin Peaks* (tab. 7.45) represents the proximity of *UAS* mutual distance to breach well clear condition. The *breach condition* was not fulfilled in any combination.

The *minimal distance to safety margin* was 0.5438 m between all four *UAS* systems. The *maximal distance to safety margin* ranges between 18 - 32 m which show advantages of the *virtual roundabout*.

| UAS: | Distance to Safety Margin | | |
|------|---------------------------|---------|--------|
| | min | max | breach |
| 1-2 | 6.9823 | 32.2369 | false |
| 1-3 | 0.5438 | 18.4015 | false |
| 1-4 | 0.5438 | 18.4015 | false |
| 2-3 | 0.5438 | 18.4015 | false |
| 2-4 | 0.5438 | 18.4015 | false |
| 3-4 | 6.9823 | 32.2369 | false |

Table 7.45: Distance to safety margin peaks for *rule-based mixed scenario*.

Path Tracking Performance: Path tracking is displayed in (fig. 7.39). The UAS trajectory is divided into *X, Y, Z axis tracking over UTM Time*. The *Reference Trajectory* (green dashed line) is represented as the interconnection between *Start Waypoint* (green square marked S) and *Goal Waypoint* \mathcal{WP}_1 (green square marked 1). The *Executed trajectory* (solid blue line) reflects real *UAS* movement.

1. *UAS 1* (fig. 7.39a) is using the bottom portion of *Virtual Roundabout* (-Y values), sticking to the boundary of the *Virtual Roundabout*.
2. *UAS 2* (fig. 7.39b) is using the upper portion of the *Virtual Roundabout*. (+Y values), sticking to the boundary of the *Virtual Roundabout*.
3. *UAS 3* (fig. 7.39c) is using the right portion of the *Virtual Roundabout*. (+X values), sticking to the boundary of the *Virtual Roundabout*.
4. *UAS 4* (fig. 7.39d) is using the left portion of the *Virtual Roundabout*. (-X values), sticking to the boundary of the *Virtual Roundabout*.

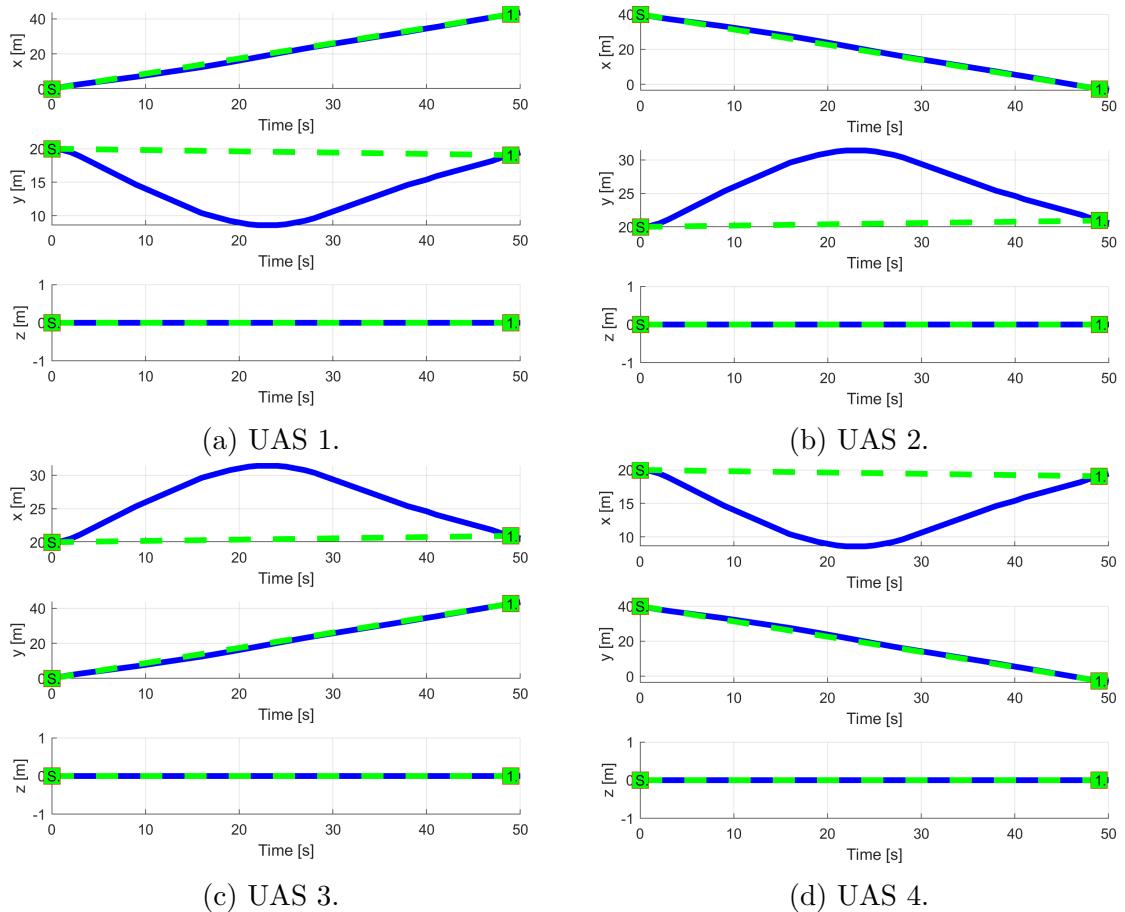


Figure 7.39: Trajectory tracking for *rule-based mixed* situation test case.

Path Tracking Deviations: *Deviations* (tab. 7.46) are in expected ranges, considering the mission plans (tab. 7.43) and *Merged Case Safety Margin* (15 m).

| Param. | UAS 1 | UAS 2 | UAS 3 | UAS 4 |
|-----------|------------------|------------------|------------------|------------------|
| | \mathcal{WP}_1 | \mathcal{WP}_1 | \mathcal{WP}_1 | \mathcal{WP}_1 |
| max $ x $ | 0 | 0 | 11.40 | 11.40 |
| max $ y $ | 11.40 | 11.40 | 0 | 0 |
| max $ z $ | 0 | 0 | 0 | 0 |
| max dist. | 11.40 | 11.40 | 11.40 | 11.40 |

Table 7.46: Path tracking properties for *rule-based mixed* scenario.

Computation Load: The *computation load* for *scenario* (fig.7.40) shows used time (y-axis) over decision frame (x-axis).

The *computation time* for each UAS has the same evolution. The *load* is higher during avoidance maneuver on the *virtual roundabout*.

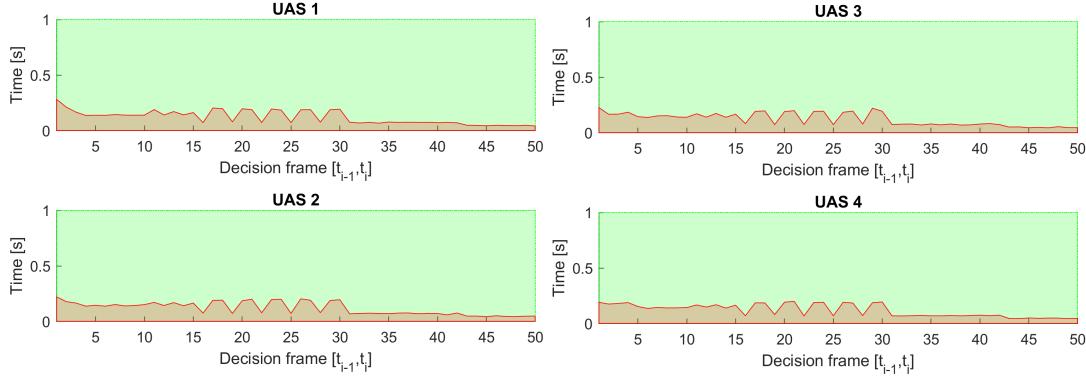


Figure 7.40: Computation time for *rule-based multiple* scenario.

7.4.4 Rule-Based Overtake

Scenario: Two UAS are flying in the *controlled airspace* (over 500 feet Above Ground Level) on the *airway* (in the same direction). *Slower UAS* is in front of *Faster UAS*. There is possibility of a *collision* or a *near miss incident* or a *well clear breach*. The *Faster UAS* (Overtaking) must contact *UTM* service and ask for *overtake permission*. Scenario steps:

1. *Faster UAS* (Overtaking) notices *UTM* service about *Slower UAS* (Overtaken). (This step is Optional.)
2. *UTM* service issues *Directives* to all *UAS* in the area.
3. *Overtake Directive* is received by *Faster UAS* (Overtaking) and *Slower UAS* (Overtaken).
4. *Faster UAS* (Overtaking) mission plan is altered to reflect *Overtake directive*, *Divergence Waypoint* and *Convergence Waypoint* are added.

5. *Faster UAS* (Overtaking) safely overtakes *Slower UAS* (Overtaken) without breaking *Well clear* condition.

Mission parameters for both *UAS* systems are defined in (tab. 7.47).

| UAS | Position | | \mathcal{WP}_1 |
|-----|------------------|---------------------------------|------------------|
| | $[x, y, z]$ | $[\theta, \varpi, \psi]$ | |
| 1 | $[-40, 20, 0]^T$ | $[0^\circ, 0^\circ, 0^\circ]^T$ | $[110, 20, 0]^T$ |
| 2 | $[-20, 20, 0]^T$ | $[0^\circ, 0^\circ, 0^\circ]^T$ | $[80, 20, 0]^T$ |

Table 7.47: Mission setup for all *Rule based overtaking* scenarios.

Assumptions: Following assumptions are valid for this test:

1. *Controlled Airspace Airworthiness* - UAS system is equipped with necessary controlled airspace equipment like ADS-B In/Out, Radar, Transponder, etc. Moreover, airworthy *UAS* has capability to precisely follow *UTM directives* (max. 5 % deviation).
2. *C2 (Command & control) Link Established* - necessary for $(\text{UAS} \leftrightarrow \text{UAS})$ and $(\text{UAS} \leftrightarrow \text{UTM})$ communication. If *C2* link is lost the *UAS* will enter into *Emergency avoidance mode*.
3. *Decision frame synchronization with UTM* - necessary in discrete C2 environment otherwise *safety margins* needs to be *bloated*.

Main Goal: Show possibility of *Overtake Maneuver* invoked by the *UTM Directive* (event-based flight constraint).

Acceptance Criteria: Following criteria must be met:

1. *Proper passing of Divergence/Convergence Waypoint* - a minimal distance of *UAS trajectory* to *Divergence/Convergence waypoint* must be below the passing threshold. Waypoints need to be passed in given order (Divergence 1st, Convergence 2nd).
2. *Slower UAS (Overtaken) keeps Right of the Way* - the UAS with lesser maneuverability does not stand a chance in avoidance situation, it needs to keep its *Right of the Way*.
3. *Both UAS does not breach Well Clear (safety) Margin* - mutual distance does not get through *calculated Safety Margin*.

Testing Setup: The *standard test setup* for each UAS defined in (tab. 7.2, 7.3, 7.4, 7.5, 7.6) is used with following parameter override:

1. *Navigation grid - type - ACAS-like with horizontal enabled maneuvers.*

This *configuration* is based on the assumption that every UAS is in *controlled airspace* in *FL450* (flight level 45000 feet Above Sea Level), without permission for a *climb or descent maneuver*. The *rule engine* is initialized in standard *Rules of the air* configuration (fig. 6.29).

There is *UTM* service for given *airspace cluster* calculating *collision cases* (tab. 6.6) based on incoming *UAS position notifications* (tab. 6.4).

Simulation Run: Notable moments from the *simulation run* (fig. 7.41) are the following:

1. *Collision case creation* (fig. 7.41a) - *Faster UAS* (blue) receives *UTM Directive* to invoke *Overtake Rule* (tab. E.7). *Slower UAS* (magenta) receives *UTM Directive* to keep *Right of the Way* and warning that is going to be *Overtaken*. *Faster UAS* (blue) creates two *virtual waypoints*:
 - a. *Divergence waypoint* at position $[0, 14, 0]^T$.
 - b. *Convergence waypoint* at position $[24, 14, 0]^T$.

Faster UAS then sets *Divergence waypoint* as *Goal waypoint*, and It starts to overtake maneuver while checking mutual distance.

2. *Divergence waypoint reach* (fig. 7.41b) - *Faster UAS* (blue) successfully reached *Divergence Waypoint*, setting *Convergence Waypoint* as new *Goal waypoint*.
3. *Convergence waypoint reach* (fig. 7.41c) - *Faster UAS* (blue) successfully reached *Convergence Waypoint*, setting *Original Goal Waypoint* as new *Goal waypoint*. The *UTM* service is notified from *Faster UAS* (blue) that *Overtaken Maneuver* has been completed. *UTM* acknowledges maneuver competition and It sends a notification to *Slower UAS* (magenta) that *Overtake Maneuver* is finished. *Slower UAS* (magenta) was successfully overtaken.
4. *Original waypoint reach* (fig. 7.41d) - *Faster UAS* (blue) successfully reached *Original Waypoint*, Starting landing Sequence.

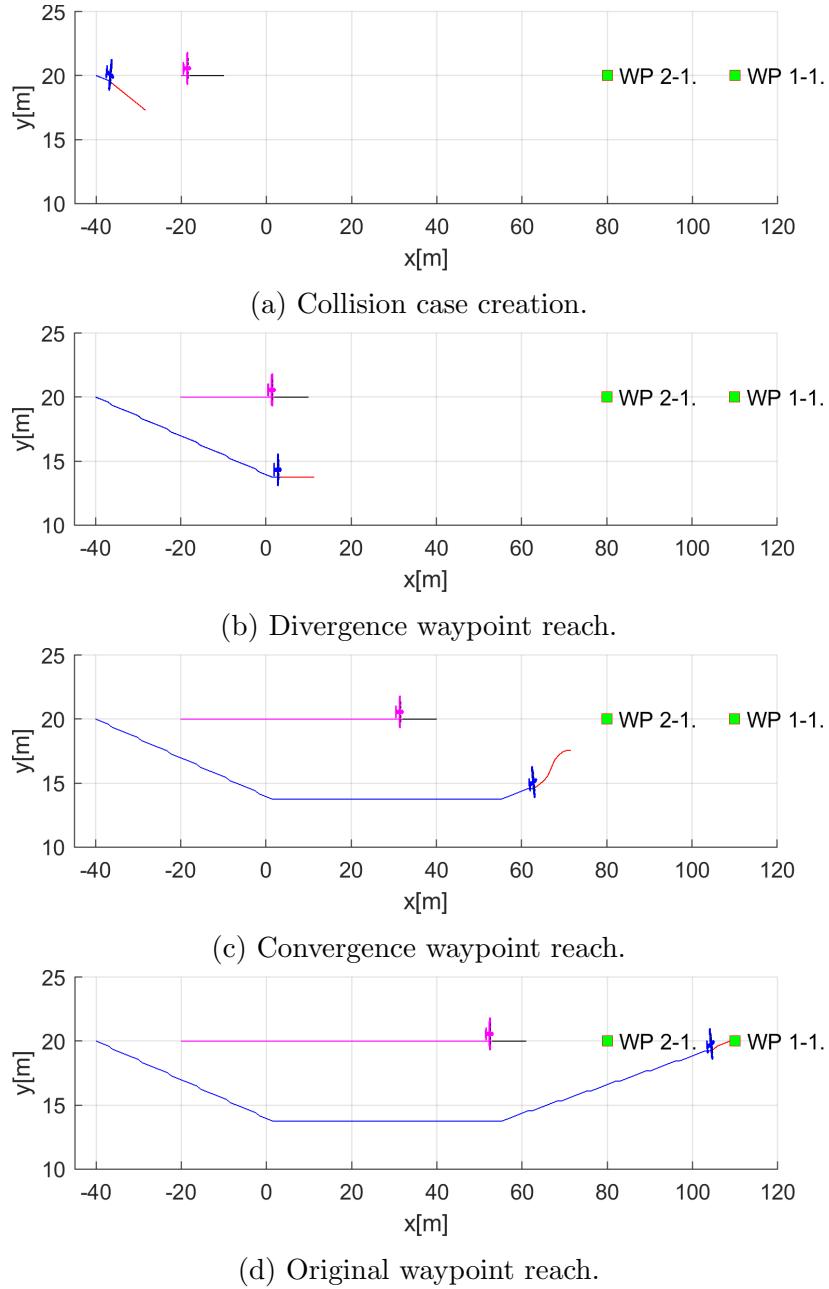


Figure 7.41: Test scenario for *rule-based Overtake* (double speed of overtaking aircraft).

Collision Case Calculation: The *Collision Case* (tab. 7.48) was calculated according to the *Collision Calculation process* (sec. 6.7.6). *Faster UAS* (1) has *Overtaking* role, and *Slower UAS* has the *Right of Way*. *Collision Point* is direct type at $[0.20.0]^T$. *Collision case type* was set based on *angle of approach* 0° as *Overtake*. The *Safety Margin* was set as 5 m.

| Collision Case | | | | | | Margins | |
|----------------|-----|------------|-----------------|-------------------|----------|---------|------|
| id | UAS | role | collision point | angle of approach | type | safety | case |
| 1-2 | 1 | Overtaking | $[0, 20, 0]^T$ | 0° | Overtake | 5 | 5 |
| | 2 | Right o.W. | | | | 5 | |

Table 7.48: Collision case for *Rule-based Overtake* scenario 2x speed.

Overtake Speed: Divergence/Convergence Waypoints *Divergence waypoints* have been calculated according to (eq. E.15), and, *Convergence Waypoints* have been calculated according to (eq. E.16). Following *Speed Differences* were taken into account (Faster/Slower UAS speed ratio): $2x$, $3x$, $4x$. Following observations can be made:

1. *The distance between Divergence and Convergence waypoint is decreasing with increasing speed difference.*
2. *Divergence waypoint is moving back/right in UAS Local Coordinate Frame with Increasing speed difference.*
3. *Convergence waypoint is moving like Divergence waypoint but a little bit faster.*

| Speed diff. | Divergence | | Convergence | | Final waypoint |
|-------------|--------------------|-------------------|-------------------|-------------------|------------------|
| | waypoint | difference | waypoint | difference | |
| 2x | $[0, 14, 0]^T$ | | $[24, 14, 0]^T$ | | $[110, 20, 0]^T$ |
| | | $[-10, -1, 0]^T$ | | $[-8, -1, 0]^T$ | |
| 3x | $[-10, 13, 0]^T$ | | $[16, 13, 0]^T$ | | $[110, 20, 0]^T$ |
| | | $[-3.4, -1, 0]^T$ | | $[-1.3, -1, 0]^T$ | |
| 4x | $[-13.4, 12, 0]^T$ | | $[14.7, 12, 0]^T$ | | $[110, 20, 0]^T$ |

Table 7.49: Convergence and divergence waypoints for various speed differences.

Overtake Speed: Impact on Trajectory Overtake speed difference is visible in (fig. 7.42). The *Slower vehicle trajectory*(cyan) is following *standard mission waypoints*. The *Faster vehicle trajectory* for $2x$ (blue), $3x$ (green), $4x$ (black) are following *Divergence/Convergence waypoints* from (tab. 7.49).

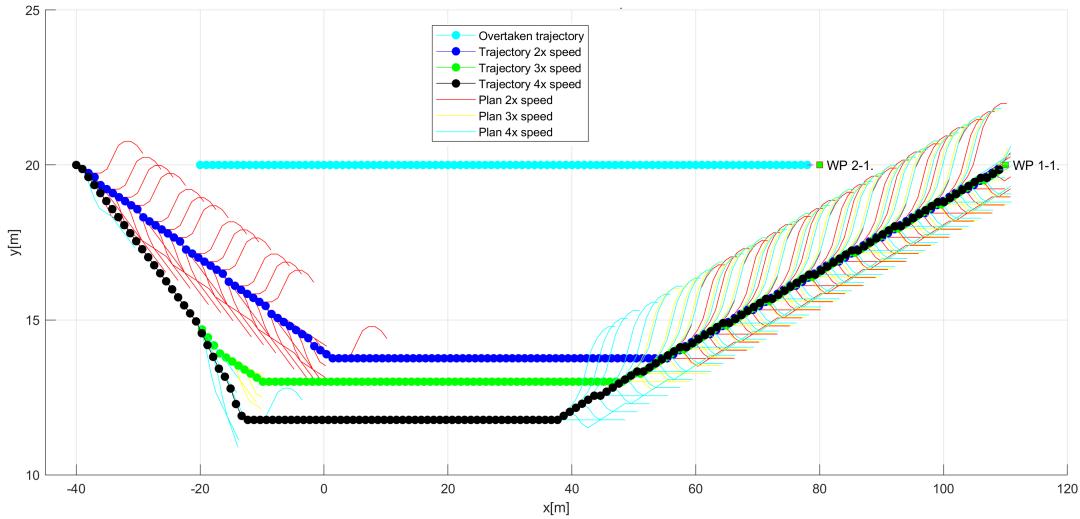


Figure 7.42: *Rule-based overtake trajectories at a different speed.*

Overtake Speed: Impact on Distance to Safety Margin Evolution *Safety margin* (red line) is set to 5 m. It is obvious that *Faster UAS* will take down *Slower UAS* if there was not for an *Overtake maneuver*. The distance of *Faster UAS* to *Slower UAS* evolution is depending on *Speed difference*. *Inflection point* (closest point of two UAS) is reached sooner with *Higher speed*. *Safety margin performance* was measured for the *UTM performance time* in the interval [0, 35] s and *Speed difference* of 2x (blue), 3x (green), 4x (black).

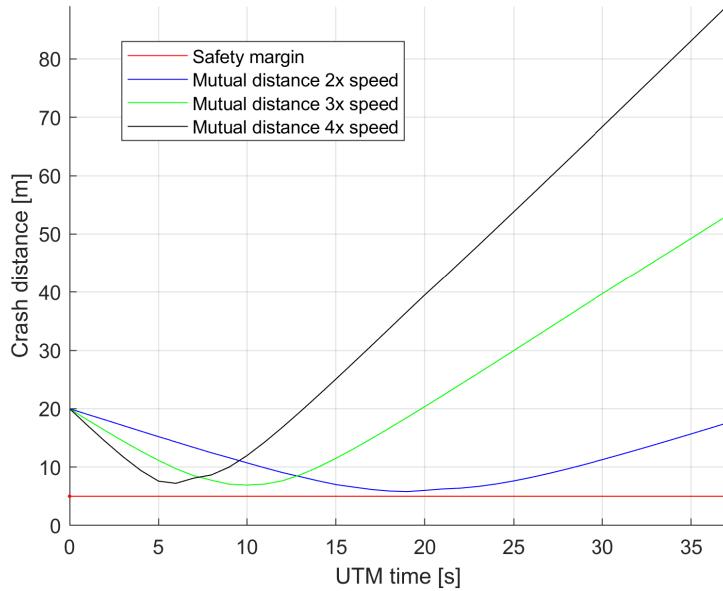


Figure 7.43: Overtake speed-dependent distance to safety margin evolution for *rule-based overtake scenario*.

Overtake Speed: Impact on Distance to Safety Margin Peaks There is summary table (tab. 7.50) for measurement of minimal and maximal values for *Distance to Safety Margin* over *UTM time* (fig.7.43). The minimal *Overtake Distance to Safety Margin* is 0.7991 m for 2x *Speed Difference*. The minimal *Overtake closest point reach time* is 7 s

for 4x *Speed Difference*.

For each *Speed difference* (2x, 3x, 4x), the *Well Clear Margin* (Safety Margin) was not reached by the *Faster UAS Body boundary*.

| Speed diff. | Minimal | | Maximal | | Breach |
|-------------|----------|------|----------|------|--------|
| | distance | time | distance | time | |
| 2x | 0.7991 | 20 | 48.8508 | 76 | false |
| 3x | 1.9180 | 11 | 73.5336 | 51 | false |
| 4x | 2.2154 | 7 | 84.0721 | 38 | false |

Table 7.50: Distance to safety margin peaks for various overtake speed in *rule-based overtaking scenario*.

Path Tracking Performance: 2x Speed Performance was only evaluated for the case when *Faster/Slower UAS speed ratio* is 2x. All waypoints are marked as green numbered *squares* with a number. Initial waypoint is marked as a green square with *S*. Reference trajectory is annotated as *green dashed line*. The *executed trajectory* is annotated as *solid blue line*.

Following observations can be made from path tracking (fig. 7.44):

1. *UAS 2 has the Right of Way* (fig. 7.44b) - reference trajectory and executed trajectory are identical.
2. *UAS 1 is Overtaking* (fig. 7.44a) - the following waypoints are marked on reference trajectory:
 - a. *Collision Point* (WP 1.) - this is not used for navigation, it is marking of *Collision Point*.
 - b. *Divergence waypoint* (WP 2.) - there will *Faster UAS* navigate to avoid *Collision*.
 - c. *Convergence waypoint* (WP 3.) - there will *Faster UAS* navigate to gain *Safe Return Distance*.
 - d. *Original Goal Waypoint* (WP 4.) - there will *Faster UAS* continue until *original goal* is reached.

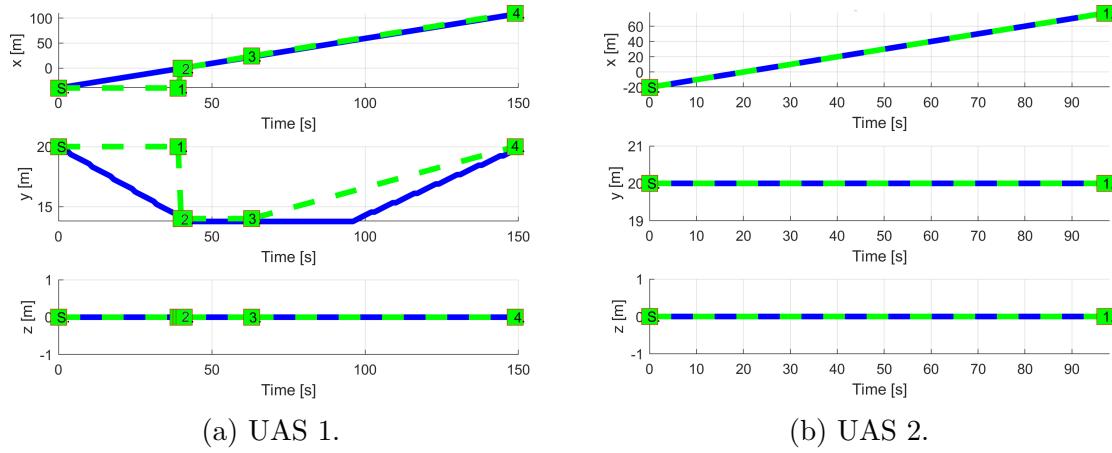


Figure 7.44: Trajectory tracking for *rule-based overtake double speed* situation test case.

Path Tracking Deviations: 2x Speed Path tracking deviations (tab. 7.51) are interesting for an *Overtake Maneuver* performance.

Maximal deviation distance is for important waypoints: Divergence (WP 2.), Convergence (WP 3.) and Original Goal Waypoint (WP 4.), equal to 0 m. This is the *desired effect* for *Overtake maneuver*.

Collision point (WP 1.) is avoided at minimal distance 5.7991 m (tab. 7.50) and maximal distance 24.5 m (tab. 7.51).

Other *Speed Difference Ratios* yields similar results.

| Param. | UAS 1 | | | | UAS 2 |
|-----------|-----------------|-----------------|-----------------|-----------------|-----------------|
| | WP ₁ | WP ₂ | WP ₃ | WP ₄ | WP ₁ |
| | col. | div. | conv. | orig. | nav. |
| max x | 20 | 0 | 0 | 0 | 0 |
| max y | 6 | 0 | 4 | 5 | 0 |
| max z | 0 | 0 | 0 | 0 | 0 |
| max dist. | 24.5 | 0 | 4 | 5 | 0 |

Table 7.51: Path tracking properties for *rule overtaking 2x speed* scenario.

Computation Load: The *computation load* for *scenario* (fig.7.45) shows used time (y-axis) over decision frame (x-axis).

The load is minimal on both UAS because the rule calculates only the divergence (eq. E.17) and convergence (eq. E.18) waypoints.

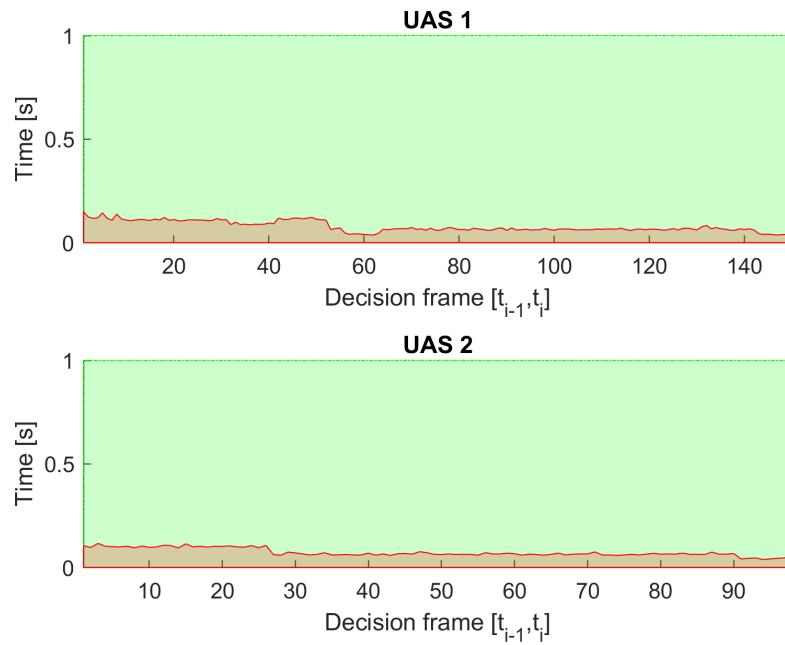


Figure 7.45: Computation time for *rule-based overtaking* scenario.

7.5 Test Cases Conclusion

This section contains summary of performance evaluation (sec. 7.1.3), adversary behavior impact on our approach (sec. 7.5.2), *calculation load* in (sec. 7.5.3).

7.5.1 Performance Evaluation

Performance of test cases was evaluated according to criteria given by (sec. 7.1.3). The performance for *test cases* from test plan (tab. 7.1) has been summarized in (tab. 7.52).

| Scenario name | Safety Margin | | | Trajectory tracking | | | Pass / Fail | |
|-----------------------------------|-------------------|--------------------|-----------|--|--|----------------------|-------------|--|
| | Distance | | Breach | Waypoint Reach | Reference Deviation | Acceptable Deviation | | |
| | min | max | | | | | | |
| Building avoidance (sim. 7.1) | 0.69 m UAS 1 | 24.98 m UAS 1 | No (7.2) | Yes/UAS 1/(7.3) | WP ₁ : 107.05m WP ₂ : 86.20m WP ₃ : 28.70m WP ₄ : 32.84m | Yes (7.10) | Pass | |
| Slalom (sim. 7.5) | 0.09 m UAS 1 | 3.74 m UAS 1 | No (7.6) | Yes/UAS 1/(7.7) | WP ₁ : 20.06m | Yes (7.14) | Pass | |
| Maze (sim. 7.9) | 0.01 m UAS 1 | 2.95 m UAS 1 | No (7.10) | Yes/UAS 1/(7.11) | WP ₁ : 28.06m | Yes (7.18) | Pass | |
| Storm (sim. 7.13) | 0.04 m UAS 1 | 34.99 m UAS 1 | No (7.14) | Yes/UAS 1/(7.15) | WP ₁ : 15.76m | Yes (7.22) | Pass | |
| Emergency Converging (sim. 7.17) | 1.67 m UAS 1-2 | 27.08 m UAS 1-1 | No (7.18) | Yes/UAS 1/(7.19a) Yes/UAS 2/(7.19b) | WP ₁ : 3.25m WP ₁ : 0.00m | Yes (7.26) | Pass | |
| Emergency Head On (sim. 7.21) | 0.38 m UAS 1-2 | 38.00 m UAS 1-2 | No (7.22) | Yes/UAS 1/(7.23a) Yes/UAS 2/(7.23b) | WP ₁ : 3.25m WP ₁ : 0.00m | Yes (7.30) | Pass | |
| Emergency Multiple (sim. 7.25) | 0.20 m UAS 2-4 | 45.46 m UAS 3-4 | No (7.26) | Yes/UAS 1/(7.27a) Yes/UAS 2/(7.27b) Yes/UAS 3/(7.27c) Yes/UAS 4/(7.27d) | WP ₁ : 4.84m WP ₁ : 1.83m WP ₁ : 3.45m WP ₁ : 2.05m | Yes (7.34) | Pass | |
| Rule-based Converging (sim. 7.29) | 1.22 m UAS 1-2 | 20.28 m UAS 1-2 | No (7.37) | Yes/UAS 1/(7.31a) Yes/UAS 2/(7.31b) | WP ₁ : 10.22m WP ₁ : 0.00m | Yes (7.38) | Pass | |
| Rule-based Head On (sim. 7.33) | 0.21 m UAS 1-2 | 36.33 m UAS 1-2 | No (7.34) | Yes/UAS 1/(7.35a) Yes/UAS 2/(7.35b) | WP ₁ : 5.40m WP ₁ : 5.40m | Yes (7.42) | Pass | |
| Rule-based Multiple (sim. 7.37) | 0.54 m UAS 2-3 | 32.24 m UAS 1-2 | No (7.38) | Yes/UAS 1/(7.39a) Yes/UAS 2/(7.39b) Yes/UAS 3/(7.39c) Yes/UAS 4/(7.39d) | WP ₁ : 11.40m WP ₁ : 11.40m WP ₁ : 11.40m WP ₁ : 11.40m | Yes (7.46) | Pass | |
| Rule-based Overtake (sim. 7.41) | 0.80 m UAS 1-2 | 48.85 m UAS 1-2 | No (7.43) | Yes/UAS 1/(7.44a) Yes/UAS 2/(7.44b) | WP ₁ : 24.00m WP ₂ : 0.00m WP ₃ : 4.00m WP ₄ : 5.00m WP ₁ : 0.00m | Yes (7.51) | Pass | |

Table 7.52: Test cases *performance evaluation*.

Highlights: Each *scenario* contains the reference to notable simulation moments and results. The scenarios were grouped according to the *Operational Space* category, and each category is separated by strike line.

Non-cooperative test cases for the Rural/Urban environment:

1. *Static obstacle avoidance* (Building/Slalom/Maze) - the buildings were correctly avoided without security breach; navigation algorithm was sufficient for given scenarios and obstacle density.
2. *Weather avoidance* (Storm) - the moving *storm* have been avoided in both *soft constraint* and *hard constraint* state. The assumption of *early detection/notification* is key in successful weather avoidance.

Non-cooperative test cases for Intruder Avoidance - the key assumptions are early intruder detection in *Avoidance Grid* and *non-adversarial* behavior. Each UAS was running own instance of *Navigation loop* (fig. 6.22). The summary of test cases is going like follow:

1. *Emergency converging* - both UAS identified correct roles according to the rules of the air. The UAS 2 kept *right of the way*.
2. *Emergency head on* - both UAS identified correct roles according rules of the air, both of them uses full separation with *Combined Reach Set Approximation* (sec. 6.4.7).
3. *Emergency mixed* - all four UAS enters into emergency avoidance mode intermediately after intruders detection. The *non-cooperative* consensus of separation is reached (fig. D.2)

Cooperative test cases with UTM supervision are working according to *UTM architecture* (fig. 6.24), where the *UTM* is considered as main authority. The key assumptions are UTM Resolution fulfillment and *non-adversary behavior*. Each UAS was running own instance of *Navigation loop* (fig. 6.22) with enabled *Rule Engine* (sec. 6.8). The summary of test cases is going like follow:

1. *Rule-based converging* - correct handling of *converging maneuver* (fig. 6.26), proper rule invocation (rule E.6) on UAS side.
2. *Rule-based head on* - correct handling of *head on maneuver* (fig. 6.25), proper rule invocation (rule E.5) on UAS side.
3. *Rule-based multiple* - proper *Collision case Merge* (tab. 7.44) with new collision point (eq. 6.122) and *safety margin calculation* (eq. 6.123).
4. *Rule-based overtake* - correct handling of *overtaking maneuver* (fig. 6.27), proper rule invocation (rule E.7). Divergence/Convergence (eq. E.15,E.16) for multiple waypoints calculation works for various speed difference (fig. 7.43).

7.5.2 Adversary Behaviour Impact

The *abuse* of UAS for *ill intentions* realization is expected. The *UAS* is cheap, disposable and does not have ethic boundaries.

One of the *assumptions* was that there are only intruders who do not actively look to harm our *UAS*. Breaking this assumption can be lethal for our system and also for other systems.

Let us take *Rule-based Head on* test case (sec. 7.4.2), changing only following aspects:

1. *UAS 2 position spoofing* - the adversarial vehicle is *faking its position* according to expected behavior.
2. *UAS 2 Navigation goal* - set as *UAS 1 position* from intercepted *position notifications* (tab. 6.4).

Simulation: The *simulation* (fig. 7.46) have been run with defined condition. UAS 2 (magenta) has been chosen as the *adversary*. UTM sees the expected trajectory of UAS 2 (grey plane/trajectory) based on spoofed *position notifications*. The *navigation/avoidance grid range* (black dashed line boundary) is shown. The notable moment of the simulation are:

1. *Deviation detection (UAS2 \leftrightarrow UTM)* (fig. 7.46a) - the *collision case* (tab. 7.40) is active and *enforced* by UTM. The *adversary* UAS 2 (magenta) starts deviating from expected trajectory (grey). UAS 1 (blue) does not register any foreign object in *avoidance grid range* (black dashed line).
2. *Adversary attacking (UAS2 \rightarrow UAS1)* (fig. 7.46b) - the adversary UAS 2 (magenta) starts actively pursuing UAS 1 (blue) by changing the original heading. This can be considered as the beginning of *active pursuit*. UAS 1 (blue) does not detect any foreign object in *avoidance grid* (black dashed line boundary). UTM is receiving expected UAS position (grey plane/line).
3. *Emergency avoidance (UAS1 \rightarrow UAS2)* (fig. 7.46c) following happens:
 - a. *Adversary UAS 2* (magenta) is spotted by *UAS 1* (blue), it entered into UAS 1 avoidance grid (black dashed line boundary).
 - b. *UAS 1* (blue) enters into *Emergency Avoidance Mode* because there is a *foreign object* in *avoidance grid*.
 - c. *UTM* notices a warning to *UAS 1* (blue) because it entered into *Emergency Avoidance Mode*. UTM is not aware of any breach, because of expected UAS 2 position (grey plane/line)
 - d. *Adversary UAS 2* (magenta) has UAS 1 (blue) locked in *navigation grid* as the goal (which guarantees optimal pursuit).

4. *Blind spot (\odot UAS1)* (fig. 7.46d) following happens:

- UAS 1* (blue) returns to *Navigation Mode* because there is no *foreign object* in *avoidance grid* (black dashed line boundary).
 - UTM* receives the mode change, and it starts enforcing *resolutions* for *collision case*, Adversary *UAS 2* is considered clear due to *expected position* (grey plane/line) compliance with the resolution.
 - Adversary UAS 2* (magenta) is on *UAS 1* blind spot. The target *UAS 1* (blue) is locked in the *UAS 2* navigation grid (black dashed line boundary).
5. *Collision detail ($UAS1 \leftrightarrow UAS2$)* (fig. 7.46e) - Target *UAS 1* (blue) is hit by *Adversary UAS 2* (magenta) on left wing tip. Both *UAS* are going down. *UTM* will detect sudden loss of both *UAS* systems.

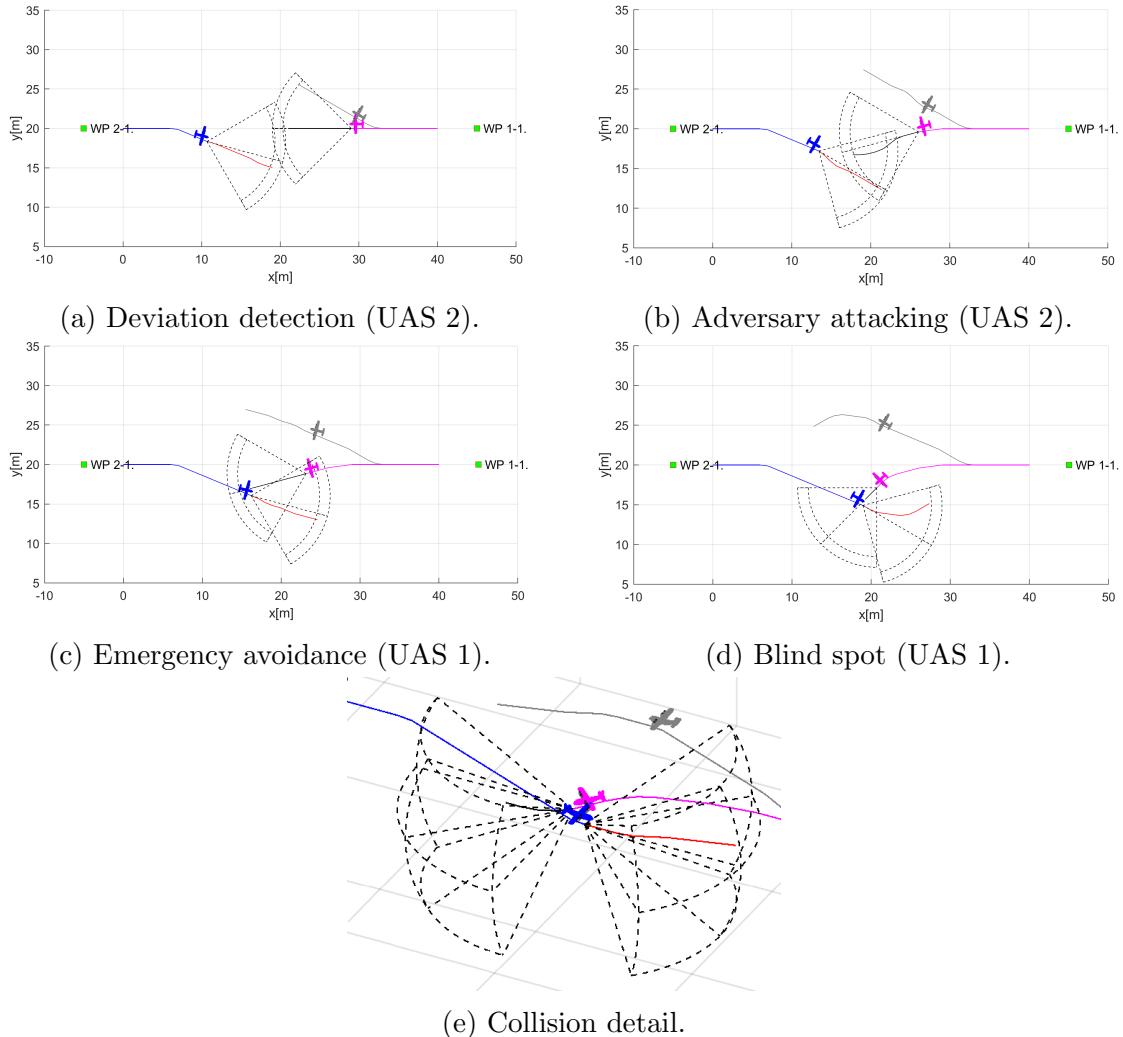


Figure 7.46: Adversarial behaviour of *UAS 2* (magenta) to compliant *UAS 1* (blue)

Performance Parameters Evaluation: Performance parameters (y-axis) are tracked over *UTM time* (x-axis). The evolution of *performance* (fig. 7.47) is tracking following parameters:

1. *Expected crash distance* (gray line) - defined as (eq . 7.1) between UAS 1 (blue)and expected UAS 2 position (grey plane/line) over mission time $t \in [0, 22]$.
2. *Crash distance* (blue line) - defined as (eq . 7.1) between UAS 1 (blue) and real UAS 2 position (magenta plane/line) over mission time $t \in [0, 22]$.
3. *Safety margin* (yellow line) - constant value according to *collision case* (tab. 7.40) as the value of 10 m. The safety margin is considered as a *soft constraint*.
4. *Body margin* (red line) - constant value according to (tab. 7.28) as value of 1.2 m. The body margin is considered as a *hard constraint*. The breaking of *body margin* means an effective *collision* UAS 1 and UAS 2.

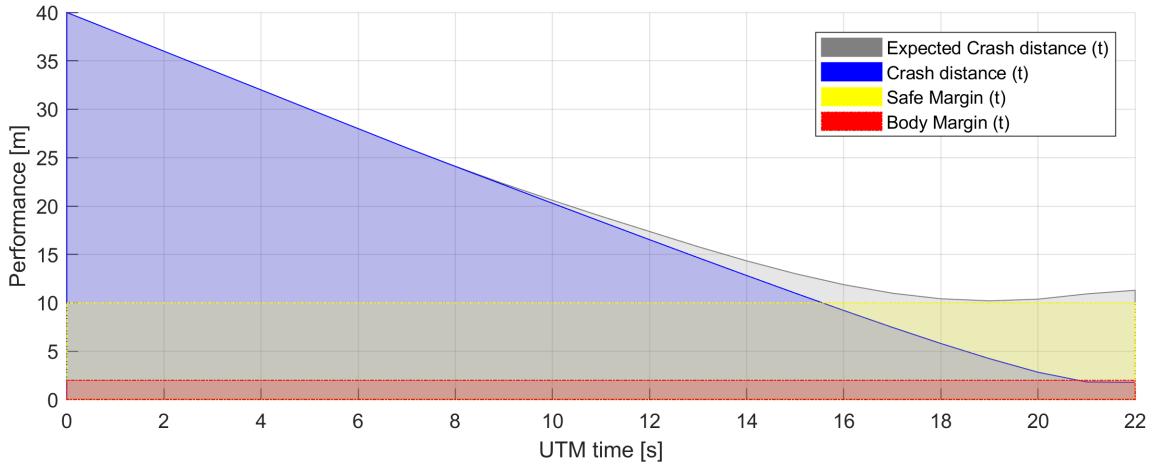


Figure 7.47: Expected/Real Distance to body/safety margin evolution for *adversarial behavior* of UAS 2.

Safety criteria for both *body* and *safety margins* in case of *expected behavior* are satisfied (eq. 7.10). This means that *UAS 1* fulfilled the *UTM directive* even though it entered *Emergency Avoidance Mode* (fig. 7.46c).

$$\begin{aligned} \text{expectedDistanceToSafetyMargin}(t) &\geq 0, & \forall t \in [0, 22] \\ \text{expectedDistanceToBodyMargin}(t) &\geq 0 & \forall t \in [0, 22] \end{aligned} \quad (7.10)$$

Safety Margin is broken at UTM time 15 s, *body margin* is broken at UTM time 21 s, the collision happens at UTM time 22 s. This is summarized in *Distance Condition Breach* (eq. 7.11).

$$\begin{aligned} \text{distanceToSafetyMargin}(t) &< 0, & \forall t \in [21, 22] \\ \text{distanceToBodyMargin}(t) &< 0 & \forall t \in [15, 22] \end{aligned} \quad (7.11)$$

Note. An *adversary behavior* needs to be addressed on:

1. *UAS Traffic Management Level* - our UTM implementation failed to detect *deviation* (fig. 7.46a) and *start of attack* (fig. 7.46b). UAS 2 (magenta) had clean

intention from the beginning and did not change pursuit even when *safety margin* was breached.

2. *Emergency Avoidance Level* - our *navigation loop implementation* does not consider the *ill-intentions*. The UAS 1 (blue) properly switched to *Emergency avoidance mode* (fig. 7.46c) after detection of UAS2 (magenta). UAS 2 (magenta) then used the blind spot to exploit UAS 1 vulnerability.

7.5.3 Computation Footprint

The *computation footprint* is summarized in computation load (tab. 7.53). The *computation load* (eq. 7.5) was calculated for each *time-frame* in scenarios. There is the summary of *minimal*, *maximal*, *average* and *median* values.

The *computational load* never exceed more than 55.95% in case of *emergency Head On* (eq. 7.6), which means that *every path* was calculated on time.

| Scenario | Computation load | | | |
|-----------------------------------|------------------|--------|--------|--------|
| | min. | max. | avg. | med. |
| Building avoidance (fig. 7.4) | 2.20% | 27.40% | 12.11% | 13.20% |
| Slalom (fig. 7.8) | 12.20% | 30.50% | 21.42% | 21.50% |
| Maze (fig. 7.12) | 24.90% | 46.10% | 31.51% | 30.80% |
| Storm (fig. 7.16) | 2.60% | 26.90% | 11.57% | 13.90% |
| Emergency Converging (fig. 7.20) | 2.75% | 16.50% | 5.84% | 4.95% |
| Emergency Head On (fig. 7.24) | 3.90% | 55.95% | 13.19% | 6.90% |
| Emergency Multiple (fig. 7.28) | 5.90% | 52.35% | 12.77% | 8.56% |
| Rule-based Converging (fig. 7.32) | 3.60% | 13.50% | 7.32% | 5.97% |
| Rule-based Head on (fig. 7.36) | 4.65% | 41.60% | 13.64% | 9.30% |
| Rule-based Multiple (fig. 7.40) | 4.37% | 23.30% | 11.96% | 10.93% |
| Rule-based Overtake (fig. 7.45) | 3.85% | 13.40% | 7.62% | 6.70% |

Table 7.53: *Computation load statistics* for all test cases.

Following observations can be made:

1. *Building avoidance*, *Slalom*, and *Maze* scenarios - the computation load is increasing with the *number of static obstacles*. The *average load* for *Emergency avoidance mode* in *clustered environment* is 31.51% (*Maze*).
2. *Storm scenario* - the overall *computation load* is very low due to the *moving constraint implementation* (def. 20).
3. *Emergency Converging/Head On/Multiple* scenarios - the *overall computation load* is quite high due to the ineffective *body volume intersection* (app. C.2) implementation.

4. *Rule-based Converging/Head On/Multiple* scenarios - the *median computational load* is low, because of the linear *rule implementation* (sec. 6.8.2)
5. *Rule-based Overtake* - the *average computation load* is very low because only *divergence/convergence* (rule. E.7) waypoints are calculated and UAS stays in *navigation mode*.

7.6 Reduced Reach Sets Performance

Constrained Expansion Method (alg. 6.1) is creating *Reach Sets* from the *Root Node* as a tree expansion using *Expansion Constraint function* (depending on type).

The *Reach set creation procedure* is creating the following artifacts:

1. *Nodes* - tree *Node* containing necessary data for discrete Trajectory portion, notably *System State Evolution*, *buffer*, and, *Reachability Rating*.
2. *Trajectories* - leaf *Node* containing *unique buffer* which is not *prefixed* in others *Node* buffer.

The *Reach Set Computation Time* depends strongly on *Movement Automaton* prediction complexity and Node count. The *Constrained Expansion Method* (alg. 6.1) is separating all nodes entering into $cell_{i,j,k}$ into two distinctive groups: *Candidates for expansion* and *Leftover Nodes*.

The *Leftover Nodes* are thrown away every expansion. The *Leftover Nodes* are not expanded in the next *Wave-front* iteration, but they leave a notable *computation* and *memory* footprint.

Note. *Average Trajectory Smoothness Rate* (def. 19) is important only in *Navigation Mode*; this aspect has been covered over (sec. 6.4.5, 6.4.7, 6.4.6).

Approach: For the same conditions (*Testing Avoidance Grid*, *UAS initial state*, *Movement Automaton*) compare the performance of *Reach Set Approximations* created by various methods for the following parameters:

1. *Coverage Ratio* - defined in (def. 18) shows how versatile *Reach Set Approximation* is (up to 100% of complete reach set coverage).
2. *Node count* - count of Nodes in *Reach Set Approximation* counted like:
 - a. full - all active nodes existing over computation time,
 - b. pruned - active nodes for real-time use.
3. *Count of Trajectories* - count of Trajectories (leaf Nodes) counted like:
 - a. full - all active trajectories existing over computation time,
 - b. pruned - active trajectories are leading to coating cells of *Avoidance Grid*.

Testing Avoidance Grid with *Distance 10 m*, *Layer count 10*, *Horizontal range* $[-45^\circ, +45^\circ]$, *Horizontal Cell Count 7*, *Vertical range* $[-30^\circ, +30^\circ]$, and *Vertical Cell Count 5*.

Note. The sizing of the *Avoidance Grid* was chosen a small scale because the property of *Coverage Ratio* can be calculated exactly up to some scale, after that it can be only assumed. Various sizes of *Avoidance Grid* was tested in [13].

The UAS is at *Back-side* of figure (the initial state is at all *Trajectory Origins*). The *black dashed line* marks *Avoidance Grid* space boundary. Each trajectory has own color and ends at *Front-side* of *Avoidance Grid Boundary*.

Coverage-Maximizing Reach Set (sec. 6.4.4) is used in *Emergency Avoidance Mode* for *Non-Controlled Airspace*. The *full* set of trajectories is given in (fig. 7.48a). The *Pruned* set of trajectories is given in (fig. 7.48b).

Tuning parameters were selected like follow: *Spread Ratio* is 15 (unique footprint trajectories in the cell), and *trajectory footprint length* is 3 (last three unique passing cells).

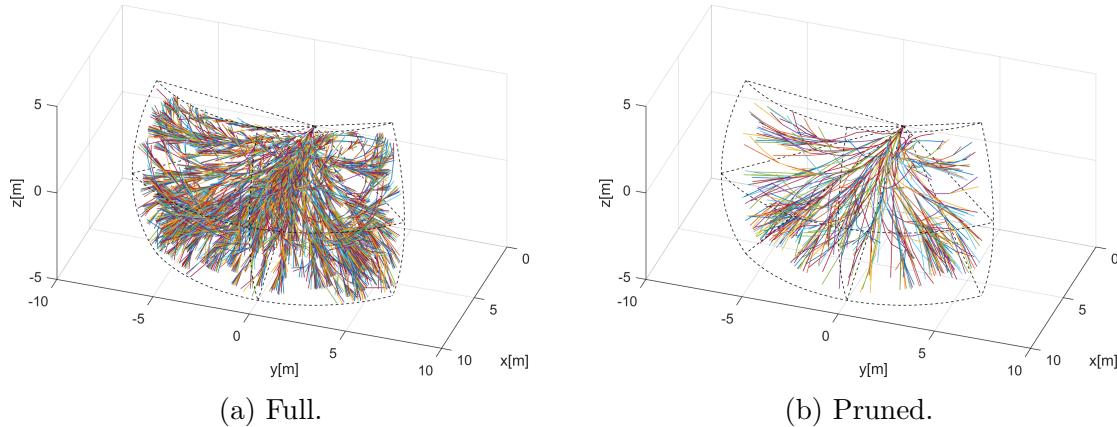


Figure 7.48: Coverage-maximizing reach set computation example.

Turn-Minimizing Reach Set (sec. 6.4.5) is used in *Navigation Mode* for *Non Controlled Airspace*. The *full* set of trajectories is given in (fig. 7.49a). The *Pruned* set of trajectories is given in (fig. 7.49b).

Tuning parameter for harmonic spread ratio was set to 9 (which implies low coverage).

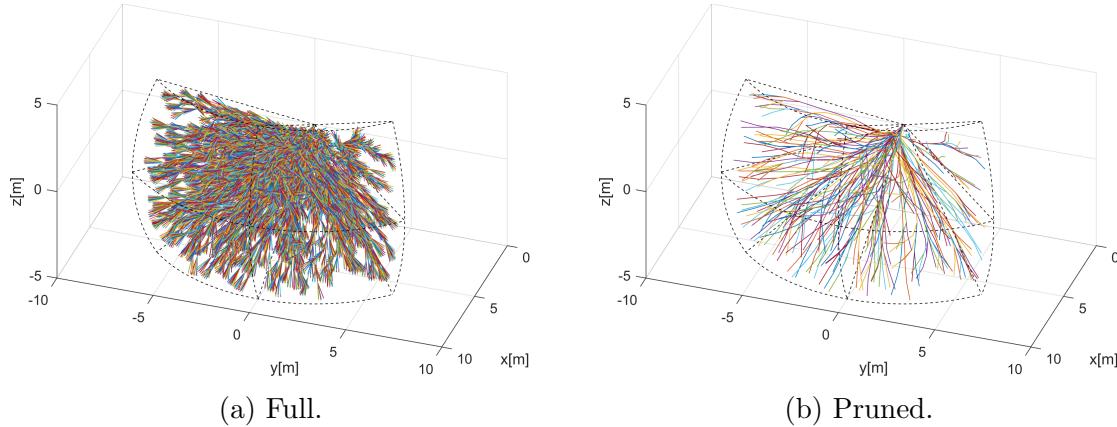


Figure 7.49: Turn-minimizing reach set computation example.

Combined Reach Set (sec. 6.4.7) is combination of *Coverage-Maximizing Reach Set* (fig. 7.48) and *Turn-Minimizing Reach Set* (fig. 7.49). The *tuning parameters* are

the same for the respective methods. It is used for both *Emergency Avoidance* and *Navigation*.

ACAS-like Reach Set (sec. 6.4.6) is used in *Navigation Mode* for *Controlled Airspace*. The separations used are *Horizontal*, *Vertical*, *Slash*, and, *Backslash*, to give the worst possible nodes and trajectories count. The *full* set of trajectories is given in (fig. 7.50a). The *Pruned* set of trajectories is given in (fig. 7.50b).

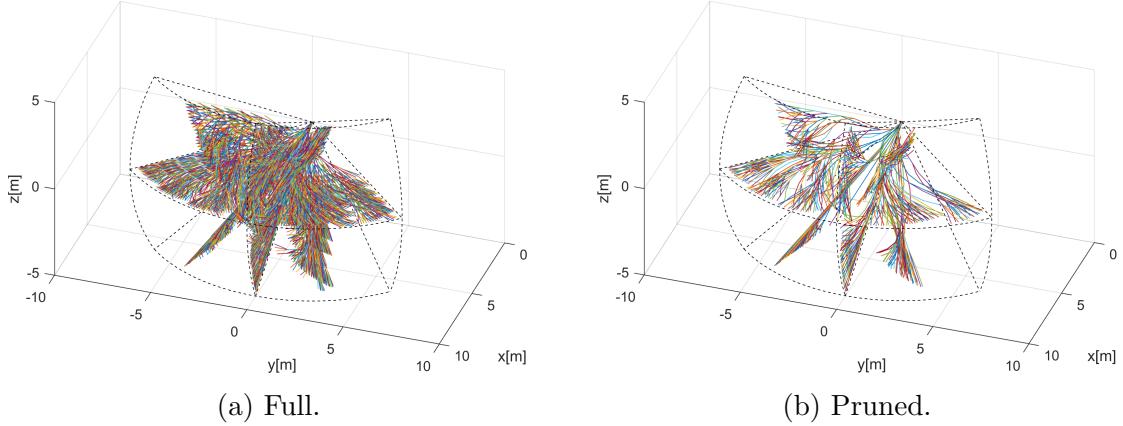


Figure 7.50: ACAS-like reach set computation example.

Computation Methods Performance Comparison (tab. 7.54) gives overview of memory consumption and *Coverage Ratio*.

Node count: *Full Node Count* shows how much memory it takes to compute *Reach set*. *Pruned Node Count* shows how much *memory* is needed for storage.

Note. The total size of *full/pruned Reach Set* depends on Node implementation. The Object-oriented prototype implementation in Matlab for example avoidance grid took up to 1 megabyte of system memory. The effective implementation would take up to 100 kilobytes.

Constrained expansion (alg. 6.1) have different selection rate, depending on the method. The survival rate directly reflects strictness of selection criteria. The rate of *node pruned* is summarized in (eq.7.12)

| Nodes pruned | |
|--------------------|----------|
| <i>CM – RSA</i> | : 78.93% |
| <i>TM – RSA</i> | : 18.50% |
| <i>ACAS – like</i> | : 79.05% |

(7.12)

The *interpretation of results* for each reach set estimation method is like follow:

1. *Coverage-Maximizing* - the main exploration drive is *Coverage Rate*, the *Trajectory* segments are not usually smooth. For our *Movement Automaton*, there is only

one *Smooth Movement*: Straight. Other eight are considered *Chaotic Movements*. Impact of this fact is significant because 4/5 of nodes were pruned.

2. *Turn-Minimizing* - the main exploration drive is *Smoothness* of contained *Trajectories*. The *Trajectory segments* which are getting further away from *cell center* are not feasible. If *Smooth Movements* set size is considered, the Smooth/Chaotic movement ratio is 1/8 for our *Movement Automaton* implementation. The low node count was expected in this approach. Another Contributing factor is *Trajectory Footprint Length* for uniqueness selection, which is not a tuning parameter in this method, and it is set to the most strict selection.
3. *ACAS-like* - the main drive is to create set consisting from *multiple 2D separation planes*. The expansion method applies full movement set on the *candidate node*. The *Separation plane movement subset* is determining, which node will be selected for further expansion. The size of separation plane subset to the size of movement set rate is 1 : 3. There are four separation planes: horizontal, vertical, slash and backslash each containing full 2D plane reach set approximation which caused high node prune rate. Nodes used rate should get lower with increasing grid size.

Trajectories count: *Full trajectories count* shows how many *leaf nodes* were existing during the calculation process without pruning. The difference between *full node count* and *full trajectories count* is count of inner tree nodes.

Pruned trajectories count shows how many *leaf nodes* are used in run-time of *avoidance algorithm*. The difference between *pruned node count* and *pruned trajectories count* shows the count of inner nodes in active reach set.

The most of *waste leaf nodes* are removed during *layer pruning*: function *reachSet.purge- SameFootprint()* (alg. 6.1). The *Waste trajectories* or *unused leaf nodes count* have significant impact. Because *leaf nodes* are a side product of *Node Expansion procedure* the amount of *pruned trajectories* is around 90 % regardless of the used method. The results are summarized in (eq. 7.13)

| Trajectories pruned | |
|---------------------|----------|
| <i>CM – RSA</i> | : 91.24% |
| <i>TM – RSA</i> | : 88.21% |
| <i>ACAS – like</i> | : 89.43% |

(7.13)

| Calculation method | Node count | | Trajectories | | Coverage ratio | Parameters |
|--------------------|------------|--------|--------------|--------|----------------|---|
| | full | pruned | full | pruned | | |
| CM-RSA | 6727 | 1417 | 4557 | 399 | 90% | spread:15 |
| TM-RSA | 1724 | 1405 | 1528 | 180 | 30% | spread:9 |
| combined | - | 2405 | - | 435 | 95% | CH spread:15 H spread:9 tree comb. |
| ACAS-like | 11294 | 2366 | 7437 | 786 | 74.95% | Separations: H/V/S/BS Coverage pruning: disabled |

Table 7.54: *Reduced reach set* computation methods performance

Coverage ratio: (def. 18) is showing how much maneuvering versatility of *Reach Set*. *Full Reach Set Approximation* have coverage ratio of 100 %. It is possible to construct *Reference Reach Set* without constrained expansion method which contains all possible *trajectory footprints*. Following observations for *coverage ratio* can be made:

1. *Coverage-maximizing* reach set estimation method by design select *Nodes* which have the high probability of *trajectory footprint* diversification. The high coverage ratio was achieved at values around 90 %.
2. *Turn-Minimizing* reach set estimation method by design selects most smooth trajectories which cause low *trajectory footprint* diversity. The fairly high coverage ratio of 30 % has been achieved.
3. *Combined* reach set estimation method takes two reach set and combines their trajectory trees into a single trajectory tree. It is given that *Coverage ratio* will achieve at least maximal coverage ratio of original reach sets. Harmonic reach set supplemented narrow smooth trajectories which were throw away previously; this increased overall *coverage ratio* to 95 %.
4. *ACAS-like* reach set estimation method contained four separation planes, which caused that it was similar to *Coverage-Maximizing Reach Set Approximation* for given *Avoidance Grid*, concerning of performance. The coverage ratio For 2D plane was 100 %.

Chapter 8

Conclusion and Future Work

This section summarizes *obstacle avoidance framework* functionality (sec. 8.1), provides the *comparison to other methods* (sec. 8.2), outlines *approach reusability* (sec. 8.3), summarizes *lessons learned* (sec. 8.4) and introduces possible *future research heading* (sec. 8.5).

8.1 Summary

The approach presented in (ch. 6) is covering the challenges (fig. 6.1) related to:

1. *Reactive avoidance* - covering static obstacles, moving intruders, geo-fencing, weather effects.
2. *Event-based avoidance* - covers core UTM functionality and cooperative avoidance.
3. *Pre-emptive avoidance* - not covered in this work, the assumption about initial waypoint reachability still holds (ass. 4).

Reactive Avoidance Test Coverage: Testing avoidance capabilities against *static obstacles*, *non-cooperative intruders*, *moving hard constraints* are covered. Coverage of the *soft constraints*, *map obstacles*, and *detected obstacles* are implicitly covered due to the properties of *safety* and *body* margins (tab. 4.2).

1. *Building Avoidance* (sec. 7.3.1) covers *static obstacles* explicitly and *map obstacles*, *hard constraints*, *ground avoidance* implicitly.
2. *Slalom* (sec. 7.3.2) covers *open space navigation capabilities*, showing the determinism of the *avoidance loop run*, in addition to *building avoidance*.
3. *Maze* (sec. 7.3.3) covers *closed space navigation capabilities*, showing the higher level navigation properties of primitive *right-side* 2D maze solver. The main point is to show possibility to enrich the *Navigation loop algorithm* (fig. 6.22).

4. *Storm* (sec. 7.3.4) covers *hard moving constraints avoidance* explicitly and *hard static constraints, soft static constraints, soft moving constraints* implicitly.
5. *Emergency converging scenario* (sec. 7.3.5) covers *non-cooperative intruder with the right of the way* avoidance capability.
6. *Emergency head-on scenario* covers (sec. 7.3.6) *non-cooperative intruder without right of the way* avoidance capability
7. *Emergency mixed scenario* (sec. 7.3.7) covers *multiple intruders with/without right of the way* avoidance capability.

Event-Based Avoidance Coverage: Test cases covers *well clear breach prevention, situation-based avoidance, and rules of the air enforcement*. Coverage of *near miss situations, clash incidents* is given implicitly by *safety* and *body* margins (tab. 4.1).

1. *Rule-based converging* (sec. 7.4.1) covers *well clear breach* and *the converging rule of the air*, showing determinism and *UTM resolution execution*.
2. *Rule-based head-on* (sec. 7.4.2) covers *well clear breach* and *head on rule of the air*, showing determinism and *UTM resolution execution*.
3. *Rule-based mixed head on with converging* (sec. 7.4.3) covers *well clear breach* and *head on and converging rules of the air*. The main focus is on the *virtual roundabout* concept when multiple collision cases are clustered into one avoidance maneuver.
4. *Rule-based overtake* (sec. 7.4.4) covers *well clear breach* during *overtaking* by faster UAS.

8.1.1 Hierarchical Decision Making

The key feature of the approach is hierarchical decision making. The hierarchy is introduced not only to threat prioritization but also to the time sequence of the decisions. The framework scheme (fig. 6.2) shows that situational assessment in the form of *data fusion* is done in (sec. 6.5.4). The trajectory selection process is separated from the final avoidance strategy pick-up process. The decision making separation is done like follows:

- *Avoidance Run* (sec. 6.6.1) - for one specific data set, for a fixed time, for a fixed goal find the cheapest trajectory which is safe (reachable).
- *Navigation Run* (sec. 6.6.2) - for multiple data sets (threat hierarchy), for a fixed time, for context depending goal try to find existing solution as a safe trajectory. Select trajectory breaking least restrictions from candidates.

The decision making process on threat assessment level (fig. 6.22) is the configuration of threat type application in 7th-10th step. The idea is that UAS:

1. must avoid any *static obstacles* (sec. 6.5.1),
2. can skip *intruders* (sec. 6.5.2) if there are no other options
3. can enter into *hard constraints* (sec. 6.5.3, def. 20) area if its necessary,
4. can enter into *soft constraints* (sec. 6.5.3, def. 20) area if its necessary.

Note. The *implementation of soft/hard constraints* and their evaluation process is same in *avoidance run*; the only difference is when they are applied by *navigation run*.

8.1.2 Use of Developed Reach Set Approximations

The reach set approximation (sec. 6.4) represents a set of *maneuvering* strategies (avoidance/movement) which can be used in a different context. The properties of the trajectories are given based on *constrained expansion* (sec. 6.4.3) and performance parameters priority (sec. 6.4.2).

The conditions and priorities are changing with the operational environment, to successful integration into nonsegregated airspace is necessary to reflect this. For this purpose, the multiple reach set approximation approaches were developed. The performance tests and longer summary can be found in (sec. 7.6). The summary of each reach set approximation method is given like follow:

- *Coverage-maximizing reach set approximation* (sec. 6.4.4) - contains a lot of *curved trajectories* with high space coverage ratio, used for *emergency reactive avoidance* in non-controlled/controlled airspace, *last resort reach set*.
- *Turn-minimizing reach set approximation* (sec. 6.4.5) - contains a lot of *straight trajectories* with low space coverage ratio, used for navigation in *non-controller airspace*.
- *Combined reach set approximation* (sec. 6.4.7) - the reach set approximation is represented as a tree, there is a possibility to combine two or more reach sets with the same initial state, the outcome of such merge is a new reach state with different properties.
- *ACAS-like reach set approximation* (sec. 6.4.6) - contains *mandatory separations* (depending on aircraft type), has low trajectory count and low coverage ratio, it emulates ACAS maneuver table in reach set data structure, used for navigation and avoidance in ATC controlled airspace.

8.2 Comparison to Other Methods

The *testing* (sec. 7.1) is strongly *capability* oriented, some performance testing regarding computational load (sec. 7.5.3) is on the place. The advantages of our approach regarding scalability are outlined in (sec. 8.2.2).

This section summarizes the results achieved by one of the master students [183]. Who compares other avoidance methods concerning performance. The *conservative* vector field avoidance [184] and *adaptive* potential field [185] methods are compared to our approach (sec. 8.2.1).

8.2.1 Conservative and Adaptive Method Comparison

Testing scenario: The testing scenario is similar to *building avoidance* (sec. 7.3.1), using the default framework testing configuration (sec. 7.2). The *reach set approximation* used is a *combination of TM-RSA and CM-RSA* (fig. 6.9) filling the role for *navigation/avoidance* in non controlled airspace. The *mission* is given by waypoints defined in (tab. 8.1), the UAS is starting at \mathcal{WP}_1 .

| Position | | Waypoints | | | | |
|------------------------|---------------------------|------------------------|-------------------------|--------------------------|-------------------------|-------------------------|
| [x, y, z] | [\theta, \varpi, \psi] | \mathcal{WP}_1 | \mathcal{WP}_2 | \mathcal{WP}_3 | \mathcal{WP}_4 | \mathcal{WP}_5 |
| [0, 0, 0] ^T | [0°, 0°, 0°] ^T | [0, 0, 0] ^T | [20, 0, 0] ^T | [20, 20, 0] ^T | [0, 20, 0] ^T | [0, 0, 10] ^T |

Table 8.1: Mission setup for *Performance test* scenario.

The *static obstacle set* (tab. 8.2) contains three obstacles in the middle of waypoints. The obstacles are balls with uniform radius. The concept of *body* and *safety margin* was not used in this test.

| Obstacle | | | Radius |
|----------|--------------------------|------|--------|
| id | position | type | |
| 1 | [10, 0, 0] ^T | ball | 2 |
| 2 | [20, 10, 0] ^T | ball | 2 |
| 3 | [10, 20, 0] ^T | ball | 2 |

Table 8.2: *Obstacle set* for *Building avoidance* scenario.

Note. The z coordinate value of zero does not represent the ground in this test scenario, because some methods do not support ground avoidance like ours.

Scenario example: The *scenario* was run for the following approaches:

1. *Static obstacle avoidance based on reach sets* - implementation [183] of [9], using own framework.
2. *Conservative* - vector field avoidance [184], adapted implementation [183].

3. *Adaptive - potential field* [185], adapted implementation [183].

The example of the testing framework developed by my student is shown in (fig. 8.1). The *obstacles* are big red balls consisting from smaller red balls, representing LiDAR sensor hit zones. The *UAS* sensing range is represented as an area outlined with a black dashed line. The waypoints are marked as green squares. The flew trajectory is represented as blue line. The *planned trajectories* (red line) for decision frames (magenta circle) which are not executed are also shown.

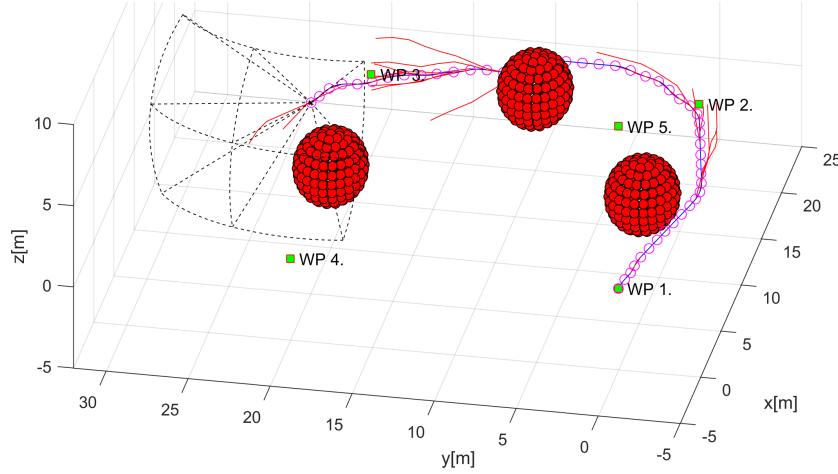


Figure 8.1: A testing scenario for method performance comparison [183].

Conservative Method Performance Margin: The *principle* of the conservative method is like follows: *Every time there must be a place to turn without the context*. The conservative method performance margin was estimated by [183] like follow:

$$\text{conservative}_{\text{margin}} = 2 \times \text{turningRadius} + 3 \times \text{uasRadius}$$

Using parameters from our testing configuration (sec. 7.2) where the turning radius is 2 and UAS body radius is 0.6 the theoretical $\text{conservative}_{\text{margin}}$ is 4.6.

Conservative Method Performance Margin: The *principle* of the conservative method is like follows: *Every mass point has a potential equal to its expected mass, those potentials repel each other with proportional force*. The adaptive method performance margin was estimated by [183] like follow:

$$\text{adaptive}_{\text{margin}} = 3 \times \text{uasRadius}$$

Using parameters from our testing configuration (sec. 7.2) where the UAS body radius is 0.6 the theoretical $\text{adaptive}_{\text{margin}}$ is 1.8.

Note. The *real conservative and method performance* was close to *theoretical best performance* [183].

Distance to Body Margin Evolution: The performance of an *obstacle avoidance framework based on reach set* for the mission (tab. 8.1) is shown in (fig. 8.2). The critical margin (crash zone) is 0.6 (UAS body radius) and is denoted as a red dashed line. The *adaptive margin* id 1.8 and its denoted as a yellow dashed line. The *conservative margin* is 4.6 and its denoted as green dashed line.

The distance to the *body margin* of the UAS center stayed mostly in *adaptive zone* (yellow line) and two times entered into *outperforming zone* (red line). The *outperforming zone* is distance values between:

$$\text{criticalMargin}(0.6) \geq \text{outperformingZone} < \text{adaptiveMargin}(1.8)$$

One can say that reach set method outperforms selected conservative and adaptive methods representatives.

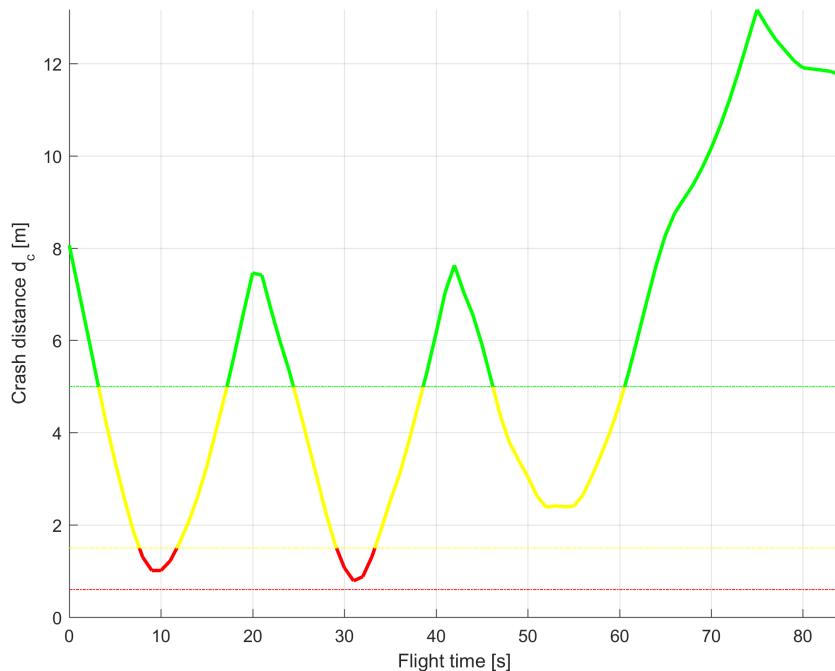


Figure 8.2: Distance to body margin evolution [183].

Disclaimer: Further investigation regarding *cost-effectiveness* is required, the approach needs to be tested in real environment and on the multiple scenarios. The testing concerning approach feasibility have been proven (tab. 7.52).

8.2.2 Scalability

The *many methods* do not offer *scalability*. The scalability means that method applies to multiple problems with different scale or there is a sufficient amount of tuneable parameter to make it work in different environments. Many avoidance concepts have hardwired margins.

The *Detect and Avoid* system must offer scalability to some extent, because of changing regulations and performance criteria on the UAS systems.

Range scalability: The physical constraints need to be sensed by sensor array in avoidance grid, and the UAS cannot get to close to them.

Static obstacles: The body margin or the minimal distance of UAS center to obstacle body is constrained like follow:

$$uasRadius < bodyMargin \leq gridRange$$

The *tuning parameter* for static obstacles is *safety margin* is then constrained like follow:

$$bodyMargin < safetyMargin \leq \infty$$

The constraints for safety margin calculation and some guidelines were outlined in (app. H.2).

Intruders: The *intruders* (sec. 6.5.2) are limited by *detection range* and intersection models parameters ranges.

The *body volume intersection* (app. C.2) has a following constraint:

$$intruderRadius < bodyRadius \leq minimalDetectionRange - 2 \times ourUASradius$$

Note. The *body radius* does not need to fit exactly on the intruder real body radius; usually, some padding space is added.

The *maneuverability uncertainty intersection* (app. C.3) is tuned through the uncertainty spread on the horizontal and vertical plane. This spread cannot be infinite, and the only limitation is given by implementation:

$$\begin{aligned} 0^\circ \leq horizontalSpread &< 90^\circ \\ 0^\circ \leq verticalSpread &< 90^\circ \end{aligned}$$

Constraints: The constraints are considered as *airspace restrictions*, *weather zones*, *geo-fencing zones* in this work. The constraints have two implementations *Static constraints* (sec. 6.5.3) and *Moving constraints* (def. 20). The only real constraint is the detection range; the constraint must be detected before making any impact on *grid*:

$$impact(uasPosition, constraintCenter) < detectionRange - gridRange$$

8.3 Approach Reusability

The *avoidance framework* can be used as a whole, or some of the concepts can be transferred as modules to other approaches. This section picks such reusable concepts,

UTM Services: The constrained *UTM functionality* is outlined in (sec. 6.7) including:

1. *Future UTM Communication Architecture* (fig. 6.24) as the authority over *airspace segment* (fig. 2.4)[41].
2. *Cooperative Conflict Resolution Under UTM Supervision* (fig. D.1) designed as mild/feasible directives (commands) with *constant supervision*.
3. *Rules of the Air Enforcement* (sec. 6.7.2, 6.7.3, 6.7.4) including designs of *Position Notification* (sec. 6.7.5) and *Collision Case Structure/Calculation* (sec. 6.7.6).
4. *Divergence/Convergence Waypoints* concept is showcased in *Overtake Rule* (rule E.7).
5. *Weather Avoidance* (app. E.1) is using a similar concept to *Collision Case: Weather Case*. The information is provided by *Local Airspace Authority*.

Emergency Avoidance Functionality: The standard framework implementation (fig. 6.22) can handle the situations given in non-cooperative test cases (sec. 7.3). The list of threats is given by (tab. 4.2).

Event-Based Avoidance Functionality: The standard framework implementation (fig. 6.22) with active *C2 link* and rules setup (fig. 6.29) can handle the situations given in cooperative test cases (7.4). The list of threats is given by (tab. 4.1). The *Avoidance Mode Concept* enables to switch between *Event-Based Avoidance* (Navigation) and *Emergency Avoidance*.

Note. The emergency Avoidance Functionality is included in *Event-Based Avoidance* (Navigation) mode. The prioritization of *threats* may differ (tab. 4.1).

Reusability for More Complex Systems: The framework (fig. F.1) with implemented rule engine (fig. 6.28) can be used on *any system*, with appropriate *Movement automaton* (sec. B.3) enabling *wave-front* propagation (alg. 6.1) for reach set estimation. Following artifacts needs to be delivered for concept reuse:

1. The *Movement Automaton* is used to generate *thick series of waypoints* which guarantees desired degree of safety.
2. The *complex UAS system* is following the *reference trajectory* (sec. B.4).

3. The *Sensor Fusion* (sec. 4.1.8) implementation including classification to *Free*, *Occupied*, *Restricted* space type.
4. The *sensor field* is supporting detection of threats. There should be at least one sensor with the capability of feeding *Avoidance Grid*. Our implementation was based on LiDAR/ADS-B feeds.
5. The *Information Sources* are supporting the online/offline threat processing. This one is completely optional.

Note. On UTM integration: The future UTM system will not be giving the extreme commands, the directives are more like constraints; therefore our system can provide the guidance and constraint evaluation

Note. On Safety Margin: The disparity between real flown trajectory (nonlinear dynamics) and planned trajectory (Movement Automaton) needs to be accounted into *Safety Margin*.

Reach Set Approximations: The *wave-front* approach (alg. 6.1) can be used with *Constrained expansion function* (sec. 6.4.3) to create own *Reach set Approximation Method*. Existing reach set approximation methods are always following a different goal; they can be reused for other tasks (perf. 7.6):

1. *Coverage-maximizing* (sec. 6.4.4) - high space coverage, ideal for unpredictable and complex avoidance maneuvers.
2. *Turn-minimizing* (sec. 6.4.5) - smooth trajectories, medium space coverage, ideal for navigation maneuvers.
3. *Combined* (sec. 6.4.7) - a combination of the *CM-RSA* and *TM-RSA*, the cost function defines preferred trajectories. The procedure is reusable for any reach set approximation types (2^+) combination.
4. *ACAS-X Like* (def. 6.4.6) - following *TCAS/ACAS separation modes*, can be used as an alternative for *controlled avoidance* and *navigation*.

8.4 Lessons Learned

During the approach development, some mistakes were made. This section summarizes the most notable design choices with the reasoning behind them.

Euclidean vs. Planar Space Partitioning: The *operational space discretization* was planned from the beginning. The initial approach was to use a *uniform Euclidean grid* where the smallest space portion was represented as cubic cell.

Euclidean grid shortcomings: The problem was with *LiDAR* reading assessment because for each point it was necessary to do *transformation* defined as a series of the functions (eq. A.6, A.7, A.8, A.9). The continuous updating of the *space assessment* usually timed-out.

The other issue was with wavefront algorithm (alg. 6.1). The intersection (def. 14) function was not problematic, due the unlimited estimation time of *reach set approximation*. The *reach set approximation* is shaped usually like the cone; therefore each propagation caused hitting more cells in Euclidean grid. This caused the following challenges:

1. *An exponential growth of trajectory count* - the *trajectory footprint* (def. 16) is almost always unique in Euclidean Grid. The problem is that two very similar trajectories concerning avoidance were considered unique.
2. *Low avoidance strategies variability* - the generated trajectories by *TM-RSA* (sec. 6.4.5) or *CM-RSA* (sec. 6.4.4) reach set approximation did not follow the desired properties of coverage.
3. *Threat impact reflection* - the space which is closer to UAS needs to be surveillance with greater detail than space which is further away from UAS. The Euclidean grid does not reflect this.

Euclidean Grid Benefits: The intersection model performance was good because of most of the numeric/analytically intersection solutions for:

1. *Line* - used in *linear intruder intersection model* (app. C.1).
2. *Tunnel* - used in *body-volume intersection model* (app. C.2).
3. *Polygon* - used in constraints (sec. 6.5.3, def. 20)

Planar Grid Deployment: The shortcomings and benefits of *Euclidean grid* are obvious, the main reason why the *planar grid* was employed is the *performance of the wave-front algorithm* (alg. 6.1) for UAS system dynamic. The main focus of this work lies on *Reach Set* properties, the planar grid (fig. 6.4) properties:

$$\uparrow \text{distance} \implies \uparrow \text{cellSize} \implies \downarrow \text{importance}$$

are ideal for *avoidance problems*. The cell size is increasing proportionally with increasing distance. The space partitioning at the lower distance is dense, reflecting the importance of proximity threats. The space partitioning at the greater distance is getting looser, reflecting the unimportant of threats at greater distance.

The *planar grid* employment has following benefits:

1. Fast *LiDAR* (and other ranging sensors) clustering.
2. Reflection of space importance.
3. A significant increase of *wave-front* algorithm.
4. A significant decrease in reach set approximation node count.

The *planar grid* employment has the following shortcomings:

1. Implementation complexity of intruder intersection models.
2. Implementation complexity of constraints intersection models.

Note. The count of intruder and constraints is not so great concerning *aerial transpiration*; the situation is opposite in case of *ground transportation* especially road vehicles.

Intruder Intersection Models: The *implemented intruder intersection models* (app. C) are basic and sufficient. There are few ideas which do not make it into the final concept:

1. *Intruder as line/curve intersection* - the idea is to use a parametric curve instead of the *linear intersection model* (app. C.1). The problem is those curve parameters are hard to obtain; they are not mandatory part of any position notification proposal so far.
2. *Intruder as movement automaton simulated trajectory* - the idea is to have access to movement automaton of an intruder in *predictor mode* (sec. B.4) to generate future expected trajectory. The problem is in some situations the trajectory of intruder changes a lot and *maneuverability uncertainty intersection* (app. C.3) outperforms this approach significantly. The other problem is that most of the *general aviation* does not want to share this kind of information due to security reasons. This concept makes it into the final solution, but not in *avoidance framework*, but as *UTM collision case intersection calculation* (sec. 6.7.6).
3. *Intruder well clear implementation as a rule* - the idea is to implement *potential fields principle* as core mechanism of avoidance. The problem is determinism of this approach, and the idea was scrapped.

Approach to data fusion: To make our approach universal, the decision to use input/output interfaces were made (fig. 6.2). The choice of the *output control interface* is clear, the movement automaton (sec. 6.2.1) was performing well from the beginning [14, 15].

The *input interface* represented as *data fusion* (theory sec. 4.1.9, implementation sec 6.5.4), was more problematic.

Deterministic approach: The article [9] and technical reports [14, 15] are using a deterministic approach where the property of the avoidance grid cell is given as a Boolean value. This approach is too restrictive because all the values needed to be determined on the spot. The overall data fusion was nonexistent because the approaches were used only static obstacles and *LiDAR* sensor as the base.

Probabilistic approach: The technical report [13] introduced the *intruders* and *map obstacles* [156]. This requires *probabilistic implementation*. The probabilistic density functions with all that theoretical apparatus are established in order to cover the processes of the data fusion and decision making. The problem was the formalization level; much *effort* is wasted to formalize very simple calculations and thresholds. The tuning parameters were nonexistent, preventing the customization.

Final Approach as a Synthesis: The benefits of both approaches for data fusion are used in final implementation (sec. 6.5.4), the useful aspects of probabilistic approach like ratings and addition rules persisted. The useful aspects of deterministic, like threshold, is kept through predicates (tab. 6.3). The concept is the following:

1. *Ratings reflects probabilities* - the property of visibility (eq. 6.85), obstacle encounter in cell (eq. 6.86), map obstacle in cell (eq. 6.87), intruder intersection with cell (eq. 6.88), constraint intersection of the cell (eq. 6.89) are formalized as threat property (def. 21).
2. *Man-made rules incorporation* - man-made rules can be incorporated into any part of the data fusion process.
3. *Situation assessment* - the known world classification through data fusion (eq. 6.93) is implemented as cell set classification:
 - a. *Uncertain* (eq. 6.93) - the state of this space cannot be determined by sensors or prior knowledge.
 - b. *Occupied* (eq. 6.94) - these cells are occupied by a physical object, entering into this space will cause real damage to UAS.
 - c. *Constrained* (eq. 6.95) - intruder or other kinds of threat may be present, this space is better to avoid.
 - d. *Free* (eq. 6.96) - these cells are free of any threat, this fact is verified by sensors and prior knowledge.
 - e. *Reachable* (eq. 6.97) - these cells are free and moreover there exists at least one *safe trajectory* leading there, the safe trajectory is a trajectory from *reach set approximation* which goes through free space.

Note. The formal definition for data fusion (sec. 4.1.9), known world (sec. 4.1.10) and their role in problem solution (sec. 4.2) are impacting the final data fusion implementation.

8.5 Future Work

The future work needs to address issues of *adversarial avoidance* (sec. 7.5.2) and *practical implementation* (sec. 5.5).

Adversarial Avoidance: The counterexample (sec. 7.46) showed that the approach is vulnerable to *adversarial behavior*. The adversarial UAS just simply avoided our field of the vision to proceed with side-hit.

This opens the possibility to solve the problem as *differential game* [69, 70]. Our UAS should pose as the *defender*; adversary should pose as the *attacker*. The *reach set* of the system given as:

$$\text{differentialGame} = \text{model}(\text{positions}, \text{defender}, \text{attacker})$$

This reach set will give us the options in our decision-making process to avoid the pursuer. This adversarial will bring additional scientific challenges which can yield interesting results.

Real System Implementation: The testing proved the *capability of approach* for wide range of applications (tab. 7.52). To proceed further with comparative testing, beyond theoretical implementations (sec. 8.2.1), it is necessary to deploy it in real environment.

The *ideal candidate* is *LSTS-tooolchain* [186]. The LSTS offers all necessary base for *approach software architecture* (fig. 6.2). The parts of our approach can be distributed over LSTS-toolchain in the following manner:

1. *LSTS Dune* (UAS on-board control) - the implementation of main *navigation loop* (sec. 6.6.2), including sensor integration with data fusion (sec. 6.5.4). The *rule engine* (sec. 6.8.1) can be deployed after UTM services implementation, to support *cooperative maneuvers*.
2. *LSTS Neptus* (UTM equivalent) - the implementation of *UTM* services and calculations (sec. 6.7.1).
3. *LSTS IMC* (Messaging implementation) - the messaging support for cooperative (fig. D.1) and non-cooperative (fig. D.2) communication schemes.
4. *LSTS Ripples* (Long term data storage) - the flight-log storage.

The real system implementation will enable to:

1. *Compare approach performance* - the theoretical performance of approach is good, only the real challenges can show the flaws and strengths of approach.
2. *Develop fault-tolerant and recovery procedures* - the process of *event-based* and *reactive* avoidance have been developed and tested in theoretical environment. The real implementation can improve the process weak points by "*learning by doing*" method
3. *Advertise the approach benefits* - the successful implementation on a real system will increase the outreach and visibility of the approach significantly.

Appendix A

Complementary Definitions

Cartesian Space: 3D Cartesian space defined by an X, Y, and Z axes (describing position based on horizontal placement, vertical placement, and depth respectively). The coordinates for any point within this space are shown as a vector $[x, y, z]$. The *coordinate system* used this work is the right-handed system (thumbs points at the positive direction of x-axis, the index finger is pointing to the positive direction of y-axis, the positive of z-axis given by remaining fingers).

Base Works: *Euler outlined universal rotation theorem* which was presented in [187]. Rigid body dynamics and rotation matrices defined by Schaub [188].

UAS Coordinate System: *Local Coordinate frame* is defined by UAS mass center as space center, $Z-$ in the direction of gravitational force, $X+$ in the direction of UAS heading. This local coordinate system is called Euler Normalized Unit-frame (ENU).

Rotation Matrices: Following *Rotation Matrices* are used to transform between two displaced coordinate systems. Roll angle rotation is defined around X-axis by matrix (eq. A.1) on YZ-plane. Pitch angle rotation matrix is defined around Y-axis by matrix (eq. A.2) on XZ-plane. Yaw angle rotation matrix is defined around Z axis by matrix (eq. A.3) on XY-plane.

$$R_{YZ} = R_{roll} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\text{roll}) & -\sin(\text{roll}) \\ 0 & \sin(\text{roll}) & \cos(\text{roll}) \end{bmatrix} \quad (\text{A.1})$$

$$R_{XZ} = R_{pitch} = \begin{bmatrix} \cos(\text{pitch}) & 0 & \sin(\text{pitch}) \\ 0 & 1 & 0 \\ -\sin(\text{pitch}) & 0 & \cos(\text{pitch}) \end{bmatrix} \quad (\text{A.2})$$

$$R_{XY} = R_{yaw} = \begin{bmatrix} \cos(\text{yaw}) & -\sin(\text{yaw}) & 0 \\ \sin(\text{yaw}) & \cos(\text{yaw}) & 1 \\ 0 & 0 & 1 \end{bmatrix} \quad (\text{A.3})$$

The full rotation matrix in X, Y, Z is given by (eq. A.4).

$$R_{XYZ} = R_{roll,pitch,yaw} = R_{XY} * R_{XZ} * R_{YZ} = R_{yaw} * R_{pitch} * R_{roll} \quad (\text{A.4})$$

Note. The rotation matrix R_{XYZ} (eq. A.4) and its inverse R_{XYZ}^{-1} which gives identity $R_{XYZ} * R_{XYZ}^{-1} = I$ are used all over this work in transformation.

Gimbal Lock Prevention: To keep solution numerically stable and rotations numerically stable gimbal lock prevention is necessary [189]. Gimbal lock occurs when one of the matrices (eq. A.1, A.2, A.3) is singular or final matrix for X, Y, Z rotation is singular (eq. A.4). Gimbal lock leads to the lost of one or more degree of freedom, depending on rank and space dimension of a singular matrix. To prevent gimbal lock, it is necessary to introduce a mechanism to check if the rotation matrix is regular. For this purpose normative reset function is introduced:

$$[roll, pitch, yaw]^T = f(t, roll^-, pitch^-, yaw^-), \quad \text{norm}(R_{roll,pitch,yaw}) = 3 \quad (\text{A.5})$$

Function resets yaw or roll angle to the initial position to keep a degree (rank) of the rotation matrix. The simpler but not fault-tolerant solution is to keep angles $roll, pitch, yaw \in (-\pi, \pi]$ range.

Polar coordinates: A *polar coordinate system* represents a point in the form of vector:

$$\text{point}_{polar} = [distance, horizontalDislocationAngle, verticalDislocationAngle]^T$$

which is ideal for representation of LiDAR scanned point because usually total point distance and a pair of dislocation angles are returned. Using most common LiDAR with horizontal rotation $horizontal^\circ$ and vertical mirror inclination $vertical^\circ$, one can define polar coordinate $\text{point}_{polar} = [distance_{x,y,z}, horizontal^\circ, vertical^\circ]$ which is dual to Cartesian coordinate $\text{point}_{cartesian} = [x, y, z]$. If rotation angle ranges are $horizontal^\circ, vertical^\circ \in (-\pi, \pi]$ transformation function is a bijection.

Polar → Cartesian: Transformation from polar to Cartesian representation is defined by following series of functions (eq. A.6, A.7, A.8, A.9).

$$distance_{xy} = \cos(horizontal^\circ) \times distance_{xyz} \quad (\text{A.6})$$

$$z = \sin(horizontal^\circ) \times distance_{xyz} \quad (\text{A.7})$$

$$y = \sin(horizontal^\circ) \times distance_{xy} \quad (\text{A.8})$$

$$x = \cos(\text{vertical}^\circ) \times \text{distance}_{xy} \quad (\text{A.9})$$

Cartesian → Polar: Transformation from Cartesian to polar representation is defined by following series of functions (eq A.10, A.11, A.12, A.13).

$$\text{distance}_{xyz} = \sqrt{x^2 + y^2 + z^2} \quad (\text{A.10})$$

$$\text{distance}_{xy} = \sqrt{x^2 + y^2} \quad (\text{A.11})$$

$$\text{horizontal}^\circ = \arctan\left(\frac{y}{x}\right) \quad (\text{A.12})$$

$$\text{vertical}^\circ = \arctan\left(\frac{z}{d_{xy}}\right) \quad (\text{A.13})$$

Definition 24. *Global Coordinate System (GCS) \mathcal{X}_g takes as center c_{g0} well-known point (for example center of geo-reference model in GNSS systems) every reference distance, plane or angle is calculated taking this center to mind.*

Definition 25. *Local Coordinate System (LCS) \mathcal{X}_l takes as center c_{l0} frame of the vehicle and can be changing position and orientation in global coordinate frame \mathcal{X}_g .*

Definition 26. *Global position of polar obstacle $o_i \in \mathcal{O}_{3D}$. Let $o_i = [d_o, \theta_o, \varphi_o]^T$ be polar position of obstacle o_i in local coordinate frame of vehicle with global Cartesian position $[x_v, y_v, z_v]^T$ and normalized orientation angles $[roll_v, pitch_v, yaw_v]$.*

Then Cartesian position of obstacle oi , $[x_o, y_o, z_o]^T$ in the local coordinate frame is given by transformation functions x_o (eq. A.9), y_o (eq. A.8), z_o (eq. A.7).

Global position of polar obstacle oi , $[x_g, y_g, z_g]^T$ is given by the following equation:

$$\begin{bmatrix} x_g \\ y_g \\ z_g \end{bmatrix} = \left[R_{XYZ}(roll_v, pitch_v, yaw_v) \begin{bmatrix} x_o \\ y_o \\ z_o \end{bmatrix} + \begin{bmatrix} x_v \\ y_v \\ z_v \end{bmatrix} \right] \quad (\text{A.14})$$

Definition 27. *Local position of global coordinate $[x_g, y_g, z_g]^T \in \mathbb{R}^3$. Let there be a vehicle with global Cartesian position $[x_v, y_v, z_v]^T$ and normalized orientation angles $[roll_v, pitch_v, yaw_v]$. in global coordinate frame \mathcal{X}_x .*

Then local Cartesian coordinate position $[x_l, y_l, z_l]^T$ of point $[x_g, y_g, z_g]^T$ is given by

the the following equation:

$$\begin{bmatrix} x_l \\ y_l \\ z_l \end{bmatrix} = \left[R_{XYZ}(-roll_v, -pitch_v, -yaw_v) \left(\begin{bmatrix} x_g \\ y_g \\ z_g \end{bmatrix} - \begin{bmatrix} x_v \\ y_v \\ z_v \end{bmatrix} \right) \right] \quad (\text{A.15})$$

The local polar position is given as $[distance_l, horizontal_l^\circ, horizontal_l^\circ]$, where $distance_l$ is given by (eq. A.10), $vertical_l^\circ$ is given by (eq. A.12). $vertical_l^\circ$ is given by (eq. A.13), where $[x_l, y_l, z_l]$ are used as local coordinates.

Polar surface calculation: The problem is to calculate intersected surface dA of ball subsurface defined by radius r , horizontal span ϕ , and vertical span θ . From classical mechanics, one can formulate the problem as given by (fig. A.1a). The intersection plot is in (fig. A.1b).

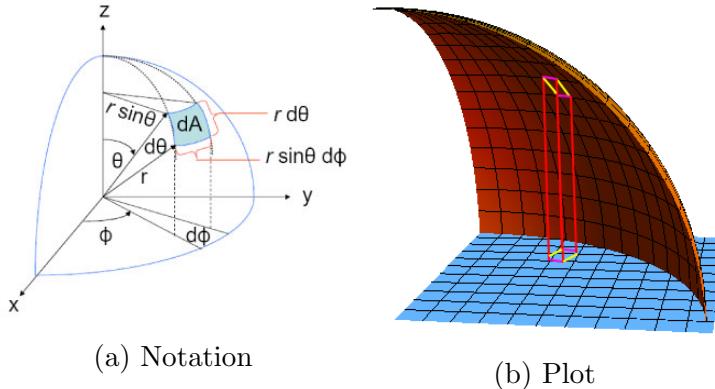


Figure A.1: Polar surface calculation notation and plot

One can use the first fundamental form to determine the surface area element. Recall that this is the metric tensor, whose components are obtained by taking the inner product of two tangent vectors in polar space $g_{ij} = X_i \cdot X_j$, for tangent vectors X_i, X_j . Following identification for the components of metric tensor will be used:

$$g_{ij} = \begin{bmatrix} E & F \\ F & G \end{bmatrix} \quad (\text{A.16})$$

Where $E = \langle X_u, X_u \rangle$, $F = \langle X_u, X_v \rangle$, and $G = \langle X_v, X_v \rangle$. Lagrange's identity can be used, which tells us that the squared area of a parallelogram in space is equal to the sum of the squares of its projections onto the Cartesian plane:

$$|X_u \times X_v|^2 = |X_u|^2 |X_v|^2 - (X_u \cdot X_v)^2 \quad (\text{A.17})$$

Given example is displayed in (fig. A.1b). The area element is given as:

$$\begin{aligned} dA &= |X_u \times X_v| \quad dudv \\ &= \sqrt{\left| |X_u|^2 |X_v|^2 - (X_u \cdot X_v)^2 \right|} \quad dudv \\ &= \sqrt{EG - F^2} \quad dudv \end{aligned} \quad (\text{A.18})$$

We will find tangent vectors via the usual parametrization which give, $X(\phi, \theta) = [r \cos \phi \sin \theta, r \sin \phi \sin \theta, r \cos \theta]$, so that tangent vectors are simply defined as:

$$\begin{aligned} X_\phi &= [-r \sin \phi \cos \theta, r \cos \phi \sin \theta, 0] \\ X_\theta &= [-r \cos \phi \sin \theta, r \sin \phi \cos \theta, -r \sin \theta] \end{aligned} \quad (\text{A.19})$$

Computing the elements of the first fundamental form gives us:

$$E = r^2 \cos^2 \theta, \quad F = 0, \quad G = r^2 \quad (\text{A.20})$$

Thus final difference is given as:

$$dA = \sqrt{r^4 \cos^2} \quad d\theta d\phi = r^2 \cos \theta \quad d\theta d\phi \quad (\text{A.21})$$

Note. *Polar Surface* is used in *Detected Obstacle Rating Calculation* (eq. 6.50). The final formula used in the *surface integral calculation* in *non-compact notation* is given as the following:

$$dA = r^2 \cos(\text{horizontal}^\circ) \quad d\text{horizontal}^\circ d\text{vertical}^\circ \quad (\text{A.22})$$

Appendix B

Movement Automaton Theory

This appendix covers theory related to *Movement Automaton*.

1. *Specialization of Hybrid Automaton* (sec. B.1) - the specialization of the hybrid automaton to fulfill control/approximation roles in our approach.
2. *Formal Movement Automaton Definition* (sec. B.2) - the formal definition of *movement automaton* used in our approach.
3. *Segmented Movement Automaton* (sec. B.3) - for more complex systems the *State Space* can be *separated into Segments*, and *segment movement automaton* is used to generate a *thick reference trajectory*.
4. *Reference Trajectory Generator* (sec. B.4) - other use of *Movement Automaton* as the predictor for *reference trajectory calculation*.

B.1 Specialization of Hybrid Automaton

Idea: There is a need for *fast trajectory approximation* method. The basic idea is taken from the pilot steering a plane. The pilot has issued commands from a navigator in a very short and precise manner. The movement has its primitive phase when steering is static, and its transition phase when steering is moving from one static position to another.

Imagine having vertical and horizontal flaps on an airplane (fig. B.1). The *navigator* is issuing a command every second to a pilot. Hands of the pilot translate each command to an input signal (blue line). The command validity period (black frame) is split into *transition period* (red frame) when the input signal is changing and *primitive period* (magenta frame) when the position of the input signal is static.

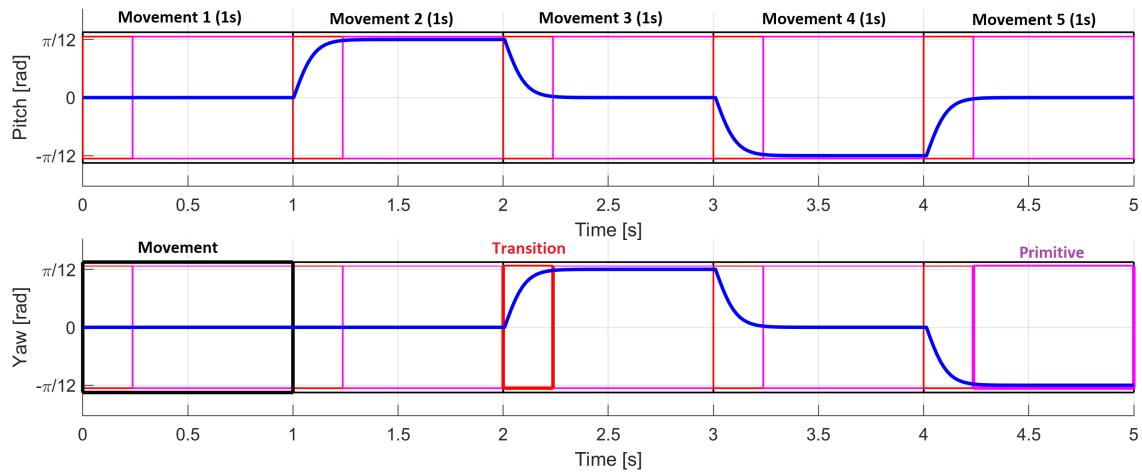


Figure B.1: Example of input signal segmentation to movements.

Note. The hybrid automaton (sec. 3.3) can be used as the base for simple control mechanism imitating navigators command execution by the pilot. The automaton states can be mapped to primitives and transitions. The reset map needs to be replaced with external order issuer to ensure smooth execution of commands.

The future commands will be stacked in the buffer from which they will be picked for execution.

Definition 28. *Movement Primitive:*

States from Hybrid automaton can be taken as Movements in Movement Automaton. MovementPrimitive (eq. B.1) is describing the Movement behavior as transfer function VectorField enriched with parameters.

$$\text{MovementPrimitive}(\text{vectorField}, \text{minimalDuration}, \text{parameters}) \quad (\text{B.1})$$

VectorField : SystemState × parameters → SystemState

Example: Let say that *UAS* system is given as *position = velocity*, then let us have two *MovementPrimitives*:

1. *Stay* - *minimalTime* = 1s, *parameters* = {}, *VectorField* : *position* = 0.
2. *Move* - *minimalTime* = 1s, *parameters* = {*velocity*}, *VectorField* : *position* = *velocity*.

Trajectory from Movement Primitives: The *UAS* should *Move* for 5s with velocity 10m/s, then *Stay* for 10s, then move for 7s with velocity 4m/s, with initial position *position*₀ = 0 and initial time *t*₀ = 1. The standard approach is to derive transfer function *position* = Θ(...)

$$\text{position}(t) = \Theta(\dots) \begin{cases} t \in [0, 5] & : 10 \times t + \text{position}(0) \\ t \in (5, 15] & : 0 \times (t - 5) + \text{position}(5) \\ t \in (15, 22] & : 4 \times (t - 15) + \text{position}(15) \end{cases} \quad (\text{B.2})$$

The *example* given by (eq. B.2) is fairly primitive, but imagine UAS system given by nonlinear dynamics [190]. Then defining transfer function for a given command chain can be impossible.

Definition 29. *Movement Transition:*

System state *can be different from intended movement application*, the notion of Transition is therefore introduced as stabilizing element in movement chaining (eq. B.3).

$$\text{Transition} : \text{MovementPrimitive} \times \text{SystemState} \rightarrow \text{MovementPrimitive} \quad (\text{B.3})$$

Trajectory with Transitions: Introducing two transitions $\text{Transition}(\text{Move}, \text{Stay})$ and $\text{Transition}(\text{Stay}, \text{Move})$ reflecting periods when vehicle stop moving or speed-up to desired velocity. The transfer function (eq. B.2) can be rewritten as combination of MovementPrimitives (eq. B.1) and Transitions (eq. B.3):

$$\begin{aligned} & \text{Transition}(\text{Stay}, \text{Move}), \text{Move}(5s, 10m/s), \\ & \text{Transition}(\text{Move}, \text{Stay}), \text{Stay}(10s), \\ & \text{Transition}(\text{Stay}, \text{Move}), \text{Move}(7s, 4m/s) \end{aligned} \quad (\text{B.4})$$

Note. There are two types of *movement primitives*:

1. *Stationary* - when the system state is considered neutral, and they are considered an entry point for automaton.
2. *Dynamic* - when the system state is considered evolving, and they need to be terminated with a *stationary* transition.

Movement Mapping Example: Transition/MovementPrimitive pairs (eq. B.3) can be mapped into movements (eq. B.5).

$$\begin{aligned} & \text{Move}(5s, 10m/s) : \text{Transition}(\text{Stay}, \text{Move}), \text{Move}(5s, 10m/s), \\ & \text{Stay}(10s) : \text{Transition}(\text{Move}, \text{Stay}), \text{Stay}(10s), \\ & \text{Move}(7s, 4m/s) : \text{Transition}(\text{Stay}, \text{Move}), \text{Move}(7s, 4m/s) \end{aligned} \quad (\text{B.5})$$

Definition 30. *Movement:*

Movement can consist from multiple Transitions (eq. B.3) and one MovementPrimitive (eq. B.1), the duration of MovementPrimitive can be shortened by Transitions duration. Movement is defined as follows:

$$\text{Movement} \begin{pmatrix} \text{initialState}, \\ \text{initialTime}[0..1], \\ \text{duration}, \\ \text{parameters}[0..1] \end{pmatrix} = \text{Chain} \left(\begin{array}{l} \text{InitialTransition}(\dots)[0..*], \\ \text{MovementPrimitive} \begin{pmatrix} \text{transitionState}, \\ \text{remainingDuration}, \\ \text{parameters} \end{pmatrix}, \\ \text{LeaveTransition}(\dots)[0..*], \end{array} \right) \quad (\text{B.6})$$

Chain function *connects multiple initial Transitions which are applied at initialState at initialTime. Then own MovementPrimitive (eq. B.1) is invoked with transitionnsState. Transitions state is state changed by Initial Transitions. After Movement Primitive there can be Leave Transitions Movement*

Minimal Movement Time: Given by (eq. B.7) for *movement* is given as the sum of *MovementPrimitive* (eq. B.1) minimal time, and *Transition* (eq. B.3) in/out combined minimal time.

$$\text{minimalTime}(\text{Movement}) = \frac{\text{minimalTime}(\text{MovementPrimitive}) +}{\max_{in/out} \{ \text{time}(\text{Transition}) \}} \quad (\text{B.7})$$

Movement Chaining: *Movements* can be *chained* and applied to initial *system state* to generate *system trajectory*. Example of the trajectory is given by (eq. B.2). Movements are reversibly obtained by participation such a *trajectory* into *Movement primitives* and *Transitions*. Then sample *Trajectory* for $n \in \mathbb{N}^+$ movements looks like (eq. B.8).

$$\begin{aligned} \text{Trajectory}(t_0) &= \text{State}(t_0) \\ \text{Trajectory}(t_0, t_1] &= \text{Movement}_1(\text{Trajectory}(t_0), t_0, \text{duration}_1, \text{parameters}_1) \\ \text{Trajectory}(t_1, t_2] &= \text{Movement}_2(\text{Trajectory}(t_1), t_1, \text{duration}_2, \text{parameters}_2) \\ \text{Trajectory}(t_2, t_3] &= \text{Movement}_3(\text{Trajectory}(t_2), t_2, \text{duration}_3, \text{parameters}_3) \\ &\vdots \\ \text{Trajectory}(t_{n-1}, t_n] &= \text{Movement}_n(\text{Trajectory}(t_{n-1}), t_{n-1}, \text{duration}_n, \text{parameters}_n) \end{aligned} \quad (\text{B.8})$$

Given *Trajectory* at time t_0 is given as the initial *State* of *System*. For time interval $(t_0, t_1]$, which length is equal to duration_1 , the *State* is given by Movement_1 with parameters_1 and base time t_0 . This behavior continues for movements $2, \dots, n$.

Definition 31. Movement Buffer:

Movements can be chained into Buffer with the assumption of continuous movement execution. Continuous movement executions each movement in the chain (eq. B.8) is executed in time interval $\tau_i = (t_{i-1}, t_i]$ where i is movement order and $\forall \text{Movement}_i$ starting time is t_0 or t_{i-1} from the previous movement. With given assumption Buffer is given as (eq. B.9) with parameters t_{i-1}, t_i omitted, due t_0 and duration_i dependency.

$$\text{Buffer} = \{\text{Movement}_i(\text{duration}_i, \text{parameters}_i)\} i \in \mathbb{N}^+ \quad (\text{B.9})$$

Definition 32. *Movement Automaton Trajectory:*

Let say system $\text{State} \in \mathbb{R}^n$ which *Trajectory* is defined by movement chaining (eq. B.8), applied on some initial time $t_0 \in \mathbb{R}^+$ and final time $t_f = t_0 + \sum_{i=1}^I \text{duration}_i$, with movements contained in Buffer (eq. B.9) is given as *Trajectory* (eq. B.10).

$$\text{Trajectory}(t_0, \text{State}(t_0), \text{Buffer}) \text{ or } \text{Trajectory}(\text{State}_0, \text{Buffer}) \text{ if } t_0 = 0 \quad (\text{B.10})$$

Note. The space dimension of *Trajectories* is \mathbb{R}^{n+1} if the space dimension of state *Space* is R^n , because *Trajectory space* contains evolution of *Space* in time interval $T[t_0, t_f]$.

The transformation from *transfer function* (eq. B.2) to *trajectory* (eq. B.10) is natural, only set of *Movement primitives* (eq. B.1) and set of *Transitions* (eq. B.3) is required.

State Projection: *Trajectory* (eq. B.10) is natural evolution of space over time, then there exists *StateProjection* function (eq. B.11) which returns *State* for specific *Time*.

$$\text{StateProjection} : \text{Trajectory} \times \text{Time} \rightarrow \text{State}(\text{Time}) \quad (\text{B.11})$$

B.2 Formal Movement Automaton Definition

Definition 33. Movement Automaton is given as follow:

$$\text{InitialState} : \in \mathbb{R}^h, h \in \mathbb{N}^+ \quad (\text{B.12})$$

$$\text{System} : \text{State} = f(\text{Time}, \text{State}, \text{Input}) \text{ or vectorField} \quad (\text{B.13})$$

$$\text{Primitives} = \left\{ \text{MovementPrimitive}_i \begin{pmatrix} \text{vectorField}, \\ \text{minimalDuration}, \\ \text{parameters} \end{pmatrix} \right\} i \in \mathbb{N}^+ \quad (\text{B.14})$$

$$\text{Transitions} = \left\{ \text{Transition}_j \begin{pmatrix} \text{MovementPrimitive}_l, \\ \text{MovementPrimitive}_k \end{pmatrix}_{k \neq l} \right\} j \in \mathbb{N}^+ \quad (\text{B.15})$$

$$\text{Movements} = \left\{ \text{Movement}_m \begin{bmatrix} \text{Transition}_o[0..*], \\ \text{MovementPrimitive}_p \\ \text{Transition}_r[0..*], \end{bmatrix}_{o \neq r} \right\} m \in \mathbb{N}^+ \quad (\text{B.16})$$

$$\text{Buffer} = \{\text{Movement}_s(\text{duration}_s, \text{parameters}_s)\} s \in \mathbb{N}^+ \quad (\text{B.17})$$

$$\text{Executed} = \{\text{Movement}_s(\text{duration}_s, \text{parameters}_t)\} t \in \mathbb{N}^+ \quad (\text{B.18})$$

$$\text{Builder} : \text{Movement} \times \text{MovementPrimitive} \rightarrow \text{Movement} \quad (\text{B.19})$$

$$\text{Trajectory} : \text{InitialState} \times \text{Movement}^u \rightarrow \text{State} \times \text{Time}, u \in \mathbb{N}^+ \quad (\text{B.20})$$

$$\text{StateProjection} : \text{Trajectory} \times \text{Time} \rightarrow \text{State}(\text{Time}) \quad (\text{B.21})$$

System (eq. B.13) is given in form of differential equations $\dot{x} = f(t, x, u)$ or other transformable equivalent, with initial state (eq. B.12).

Movements (eq. B.8) are defined as sequence of necessary initial transitions (eq. B.15), movement primitive (eq. B.14), and, leave transitions (B.15).

The Buffer contains a set of movement primitives (eq. B.14) to be executed in order to achieve the desired goal. Builder (eq. B.19) assures that first movement primitive (eq. B.1) from Buffer (eq. B.17) is transformed into next movement (eq. B.16) based on current movement (eq. B.16).

The system trajectory (eq. B.20) is defined in (eq. B.10). State projection (eqs. B.11, B.21) is giving State variable for time $t \in [t_0, t_{max}]$ where t_{max} is given by:

$$t_{max} = t_0 + \sum_{i=1, u} \text{Buffer.Movement}(i).\text{movementDuration} \quad (\text{B.22})$$

Note. From Continuous Reach set to Movement Automaton Control Reach Set:

The reach set R (B.23) for system $d/dt \text{state} = \text{model}(\text{state}, \text{input})$ with initial state $\text{state}_0 = \text{state}(t_i)$ in time interval $[t_i, t_{i+1}[$ is with existing control strategy $\text{input}(t) \in$

ControlStrategy(t). The reach set $R(state_0, t_0, t_1)$ where $t_1 > t_0$.

$$R(state_0, t_0, t_1) = \bigcup \{state(s) : input(s) \in ControlStrategy(s), s \in (t_0, t_1]\} \quad (\text{B.23})$$

The reach set \mathcal{R} (B.24) of the system under the control of the movement automation consist from the set of trajectories $Trajectory(initialState, buffer)$, which are executed in constrained time period $[t_i, t_{i+1}[$.

$$\begin{aligned} ReachSet(state_0, t_i, t_{i+1}) = \\ \{Trajectory(state_0, buffer) : duration(buffer) \leq (t_{i+1} - t_i)\} \end{aligned} \quad (\text{B.24})$$

Note. Weak Invariance:

When the UAS is under the control of the movement automaton for the obstacle avoidance problem, by design of the avoidance algorithm, the trajectories of the UAS will not intersect any threat. This means that the controlled system $d/dt state = model(state, input)$ is *weakly invariant* to the complement of the threats, and with respect to the free space. A pair $(state, SafeSpace)$, where $d/dt state = model(state, input)$ and *SafeSpace* is a closed set, is weakly invariant if there exist controls such that a trajectory starting inside $State_0 \in SafeSpace$ remains inside $State(t) \in SafeSpace$ [147].

B.3 Segmented Movement Automaton

Motivation: Constructing *Movement Automaton* for the more complex system can be tedious. Used *Movement Automaton* for *UAS system* (6.4) has decoupled control which is not true for most of the copters/planes [190].

Partitioning UAS State Space: Proposed movement automaton is defined by its Movement set (tab. 6.1,6.2). Those can be scaled depending on maneuverability in the *Initial state state(0)*:

1. *Climb/Descent Rate* $\delta pitch_{max}(k)$ - the maximal climb or descent rate for Up/Down movements.
2. *Turn Rate* $\delta yaw_{max}(k)$ - the maximal turn rate for Left/Right movement.
3. *Acceleration* $\delta v_{max}(k)$ - the maximal acceleration in cruising speed range.

Definition 34. *State Space partition Maneuverability is depending on Initial State. There cannot be the infinite count of Movement Automatons.*

The state space StateSpace $\in \mathbb{R}^n$ can be separated into two exclusive subsets:

$$StateSpace = [ImpactStates, NonImpactingStates] \quad (\text{B.25})$$

The Impacting states are states which bounds the Maneuverability: $\delta pitch_{max}(k)$, $\delta yaw_{max}(k)$, $\delta v_{max}(k)$. For each impact state is possible to define upper and lower boundary:

$\forall impactState \in ImpactStates, \exists :$

$$lower(impactState) \leq value(impactState) \leq upper(impactState) \quad (B.26)$$

The bounded interval of impact state can be separated into distinctive impact state segments like follow:

$impactState \in [lower, upper] :$

$$\begin{aligned} \{[lower, separator_1[\dots \cup \dots [separator_i, separator_{i+1}[\dots \cup \dots \\ \dots \cup \dots [separator_n, upper]\}] = \\ = impactStateIntervals(impactState) \end{aligned} \quad (B.27)$$

Note. The interval length depends on model dynamics. The rule of thumb is to keep maximal climb/descend/turn/acceleration rates near constant value.

When partitioning of all impact States finishes, the count of partitions is given as the product of count of partitions for each member of Impact States:

$$partitionCount = \prod_{impactState \in ImpactStates} |impactStateIntervals(impactState)| \quad (B.28)$$

Note. Try to keep the count of partitions to a minimum; each new interval increases the count of partitions geometrically.

There is finite number n of Impacting States, these are separated into $impactStateIntervals_i$ with respective index $i \in 1 \dots n$. The segment with index defining position used impacting state intervals is given as constrained space:

$$Segment(index) = \left[\begin{array}{l} impactState_1 \in impactStateIntervals_1[index_1], \\ \vdots \\ impactState_n \in impactStateIntervals_n[index_n], \\ \vdots \\ NonImpactingStates \end{array} \right] \quad (B.29)$$

Each Segment covers one of impacting state intervals combination because the original intervals are exclusive, also Segments are exclusive. The union of all segments covers State Space:

$$StateSpace = \bigcup_{\forall index \in |impactStateIntervals|^n} Segment(index) \quad (B.30)$$

Segmented Movement Automaton: The segmentation of *state space* is done in (def. 34) any *state* belongs exactly to *Segment* of *State Space*. For each *Segment* in *State Space* it is possible to assess: *Climb/Descent Rate* $\delta pitch_{max}(k)$, *Turn Rate* $\delta yaw_{max}(k)$, and, *Acceleration* $\delta v_{max}(k)$.

Definition 35. *Movement Automaton for Segment(index)*

For for Model(eq. 6.10) with State (eq. 6.9) the input vector (eq. 6.8) is for position $[x, y, z]$ and velocity defined like:

$$\begin{aligned} \delta x(k) &= (v(k) + \delta v(k)) \cos(\delta pitch(k)) \cos(\delta yaw(k)) \\ \delta y(k) &= (v(k) + \delta v(k)) \cos(\delta pitch(k)) \sin(\delta yaw(k)) \\ \delta z(k) &= -(v(k) + \delta v(k)) \cos(\delta pitch(k)) \\ \delta v(k) &\in [-\delta v(k)_{max}, \delta v(k)_{max}] \end{aligned} \quad (B.31)$$

The acceleration $\delta v(k)$ is in interval $[-\delta v(k)_{max}, \delta v(k)_{max}]$, usually set to 0 ms^{-1} . The change of the orientation angles for *Movement Set* (eq. 6.11) is given in (tab. B.1,B.2).

| <i>input(movement)</i> | Straight | Down | Up | Left | Right |
|-----------------------------|----------|----------------------|-----------------------|--------------------|---------------------|
| $\delta roll(k)[^{\circ}]$ | 0 | 0 | 0 | 0 | 0 |
| $\delta pitch(k)[^{\circ}]$ | 0 | $\delta pitch_{max}$ | $-\delta pitch_{max}$ | 0 | 0 |
| $\delta yaw(k)[^{\circ}]$ | 0 | 0 | 0 | δyaw_{max} | $-\delta yaw_{max}$ |

Table B.1: Orientation input values for main axes movements.

| <i>input(movement)</i> | Down-Left | Down-Right | Up-Left | Up-Right |
|-----------------------------|-----------------------|-----------------------|----------------------|----------------------|
| $\delta roll(k)[^{\circ}]$ | 0 | 0 | 0 | 0 |
| $\delta pitch(k)[^{\circ}]$ | $-\delta pitch_{max}$ | $-\delta pitch_{max}$ | $\delta pitch_{max}$ | $\delta pitch_{max}$ |
| $\delta yaw(k)[^{\circ}]$ | δyaw_{max} | $-\delta yaw_{max}$ | δyaw_{max} | $-\delta yaw_{max}$ |

Table B.2: Orientation input values for diagonal axes movements.

Note. The *Trajectory* is calculated the same as in (eq. B.33). The *State Projection* is given as in (eq. B.36).

Then the *Movement Automaton* for $Segment \in StateSpace$ is defined.

Definition 36. *Segmented Movement Automaton For system with segmented state space* (eq. B.30) *there is for each state(k) in StateSpace injection function:*

$$ActiveMovementAutomaton : StateSpace \rightarrow MovementAutomaton \quad (B.32)$$

Trajectory of non-segmented movement automaton (eq. B.33) is then given as the time-series of discrete states:

$$\text{Trajectory}(\text{state}(0), \text{Buffer}) = \left\{ \begin{array}{l} \text{state}(0) + \sum_{j=0}^{i-1} \text{input}(\text{movement}(j)) : \\ i \in \{1 \dots |\text{Buffer}| + 1\}, \\ \text{movement}(\cdot) \in \text{Buffer} \end{array} \right\} \quad (\text{B.33})$$

Selecting appropriate movement automaton implementation (def. 35) for state(k) \in Segment \subset State Space. The mapping function (eq. B.32) is injection mapping every state(k) to Segment then Movement Automaton Implementation. The trajectory generated is then given:

$$\text{Trajectory} \left(\begin{matrix} \text{state}(0), \\ \text{Buffer} \end{matrix} \right) = \left\{ \begin{array}{l} \text{state}(0) + \dots \\ \sum_{j=0}^{i-1} \text{ActiveMovementAutomaton}(\text{state}(j-1)). \\ \quad .\text{input}(\text{movement}(j)) \\ i \in \{1 \dots |\text{Buffer}| + 1\}, \\ \text{movement}(\cdot) \in \text{Buffer} \end{array} \right\} \quad (\text{B.34})$$

B.4 Reference Trajectory Generator

Reference Trajectory Generator: Segmented Movement Automaton (def. 36) with trajectory function (eq. B.34) is used as a *reference trajectory generator* for *complex systems*.

There is an assumption that precise *path tracking* implementation exist for such system which with *thick reference trajectory* gives similar results to *plain movement automaton control*.

The *Reference trajectory* (eq. B.35) for *Planned* movement set is given as projection of *Trajectory* time series to position time series $[x, y, z, t]$:

$$\text{ReferenceTrajectory} : \text{Trajectory} \left(\begin{matrix} \text{state}(\text{now}), \\ \text{Planned} \end{matrix} \right) \rightarrow \left[\begin{array}{l} x_{ref} \in \mathbb{R}^{|\text{Planned}|} \\ y_{ref} \in \mathbb{R}^{|\text{Planned}|} \\ z_{ref} \in \mathbb{R}^{|\text{Planned}|} \\ t_{ref} \in \mathbb{R}^{|\text{Planned}|} \end{array} \right] \quad (\text{B.35})$$

Predictor: The *Reference Trajectory Generator* (eq. B.35) can also be used as a predictor.

Note. The *Segmented Movement Automaton* (def. 36) is used in this work with one Segment equal to State space with input function given by (6.1, 6.2). The predictor used

in *Reach set computation* is given by (eq. B.35).

State Projection (eq. B.36) for the *Trajectory* (eq. B.33) is given as follow:

$$\text{StateProjection}(\text{Trajectory}, \text{time}) = \text{Trajectory.getMemberByIndex}(\text{time} + 1) \quad (\text{B.36})$$

Note. *Movement Automaton* for system (eq. 6.4) with given (as. 6) is established with all related properties (sec. 33).

Appendix C

Intruder Intersection Models

C.1 Small-Body Direct-Movement Intruder Intersection

Idea: There are *small intruders* which have body *smaller* than average $cell_{i,j,k}$ cell size. Its trajectory will stick to *linear trajectory* prediction with high probability.

Space Intersection Rate: The *Space Intersection Rate* for $cell_{i,j,k}$ is implemented as simple point cloud intersection. Where *sufficiently thick* point cloud is defined along *line* (eq. C.1):

$$position(time) = position(time_0) + velocity \times time, \quad time \in [0, \infty[\quad (C.1)$$

Then there exist projection function from local Euclidean coordinates to local polar coordinates (eq. C.2). The function projects intruder trajectory (eq. C.1) to planar coordinates [*distance*, *horizontal* $^\circ$, *vertical* $^\circ$] as a set of sufficiently thick point cloud.

$$polarSet : position(t) \rightarrow \{[distance, horizontal^\circ], vertical^\circ\} \quad (C.2)$$

The *space intersection rating* $SpaceIntersection(\circ)$ for line type is given as (eq. C.3). If there exist non empty intersection of $polarSet \cap cell_{i,j,k}$ there is space intersection rate equal to 1, if intersection $polarSet \cap cell_{i,j,k} = \emptyset$ then the rate is zero.

$$space \left(\begin{array}{c} Intruder, \\ cell_{i,j,k} \end{array} \right) = \begin{cases} 1 : & \exists point \in polarSet(eq.C.2) : point \in c_{i,j,k} \\ 0 : & \text{otherwise} \end{cases} \quad (C.3)$$

Note. The *intruder intersection rate* is multiplication of *space intersection rate* and time intersection rate. The *intersection rate* is calculated for *every intruder* and *selected intersection model* separately.

C.2 Notable-Body Direct-Movement Intruder Intersection

Idea: The *Intruder* has body volume greater than *average cell_{i,j,k}* volume. The *intruder body* is considered as the ball moving along *intruder position*. The *intersection* of the intruder body is realized as sufficiently thick *point-cloud intersection*.

Space Intersection Rate - Body Volume: The *body volume mass* with center at *position(t)* is moving along intruder trajectory prediction (eq. C.4) in time interval $[0, \infty[:$

$$\text{position}(\text{time}) = \text{position}(\text{time}_0) + \text{velocity} \times \text{time} \quad (\text{C.4})$$

The body *Volume ball Body(position(t), radius)* (eq. C.5) is defined as set of points in \mathbb{R}^3 euclidean space. The center is moving along the *position(t)*. The body *volume ball* is a set of points sufficiently thick including also inner points. The *thickness* is guaranteed by existence of neighbour point which is close enough.

$$\text{Body}(\text{position}(t), \text{radius}) = \left\{ \begin{array}{l} \|\text{position}(t) - \text{point}\| \leq \text{radius} \\ \text{point} \in \mathbb{R}^3 : \forall \text{point}_i \exists \text{point}_{j \neq i}, \\ \text{distance}(\text{point}_i, \text{point}_j) \leq \text{thickness} \end{array} \right\} \quad (\text{C.5})$$

The *polar volume ball polarBody* (eq. C.6) is projection of body volume ball set *Body(position(t), radius)* to a set of planar coordinates in avoidance grid coordinate frame:

$$\text{polarBall}(t) : \text{Body}(\text{position}(t), \text{radius}) \rightarrow \left\{ \begin{bmatrix} \text{distance}, \text{horizontal}^\circ, \\ \text{vertical}^\circ, \text{intersectionTime} \end{bmatrix} \right\} \quad (\text{C.6})$$

The *space intersection rate for vehicle body space(Intruder, cell_{i,j,k})* (eq. C.7) is calculated as intersection of polar body volume ball and *cell_{i,j,k}*. If intersection is non empty then base probability is one, zero otherwise:

$$\text{space} \begin{pmatrix} \text{Intruder}, \\ \text{cell}_{i,j,k} \end{pmatrix} = \begin{cases} 1 : & \exists \text{point} \in \text{polarBall}(\text{eq.C.6}) : \text{point} \in c_{i,j,k} \\ 0 : & \text{otherwise} \end{cases} \quad (\text{C.7})$$

Intersection Time: The *intersection time* id depending on point cloud (eq. C.6) where each point *have intersection time* given as *body-center position* time (eq. C.4).

Note. The *body-volume* intersection model can insert the *multiple intersection times* into one *cell_{i,j,k}*. The *interval length* considers all of these for intersection rates (eq. 6.71).

C.3 Maneuvering-Intruder-Intersection

Idea: The *intruders* are not bullets they are not sticking to predicted linear paths. The *intruder* maneuverability is given as horizontal and vertical spread. Therefore *intruder reach set* will form an *elliptic cone*. This cone can be transformed into *finite discrete* point-cloud, each *point* should have assigned *severity* impact value. The point cloud intersection with *Avoidance Grid* will give us space impact of an *uncertain* intruder.

Note. The following section will use condensed notation, due to the equation complexity. The *terminology* is consistent with the rest of the section.

Space Intersection Rate - Body Volume Intersection: $P_T(i_k(x_s, v, \theta, \varphi), c_{i,j,k})$ computation is less straight-forward than other space intersection rates. First let us define the linear intruder i_k positions x at time t (eq. C.8) model, where $x(t)$ defines intruder position in *avoidance grid euclidean coordinate frame* at time t_i , v defines intruder velocity, and t is a time offset.

$$x(t) = x_s + v_I \cdot t \quad (\text{C.8})$$

Intruder *horizontal spread* θ and *vertical spread* φ are introduced. These spreads represent intruder deviation limits along from linear trajectory prediction $x(t) \in \mathbb{R}^3$. The example is given by (fig. C.1) where the intruder starts at point x_s with fixed velocity v , the linear trajectory prediction is outlined by blue line. The *predicted intruder position* at time $t = 10s$ is given by $x(10)$ (blue point). The ellipsoidal space $E(x)$ is projected on the plane $D(x(t))$. The plane D (eq. C.9) for point $x(t)$ and velocity v is defined as an orthogonal plane to velocity vector $v \in \mathbb{R}^3$ with origin at intruder position $x(t)$.

$$D(x(t), v) = \{a \in \mathbb{R}^3 : (a - x(t)) \perp v, \} \quad (\text{C.9})$$

To construct ellipsoidal space boundary on orthogonal plane $D(x(t), v)$ some parameters are defined in (eq. C.10). The *scalar distance* $d_d(x(t))$ is simple Euclidean norm, *maximal horizontal offset* $d_\theta(x(t))$ is given as product of sinus of horizontal offset angle θ and scalar distance d_d , and *maximal vertical offset* $d_\varphi(x(t))$ is given a product of sinus of vertical offset angle φ and scalar distance d_d .

$$\begin{aligned} d_d &= d_d(x(t), x_s) = \|x(t) - x_s\|_2 \\ d_{\theta_{\max}} &= d_\theta(x(t)) = \sin \theta(i_k) \cdot d_d(x(t)) \\ d_{\varphi_{\max}} &= d_\varphi(x(t)) = \sin \varphi(i_k) \cdot d_d(x(t)) \end{aligned} \quad (\text{C.10})$$

The *Ellipsoid* $E(x(t), v)$ (eq. C.11) for fixed intruder position $x(t)$ and fixed intruder velocity v is given as constrained portion of orthogonal plane $D(x(t), v)$. The constraint is defined by an internal coordinate frame $p \in \mathbb{R}^2$ which is space reduction of plane $D(x(t), v)$.

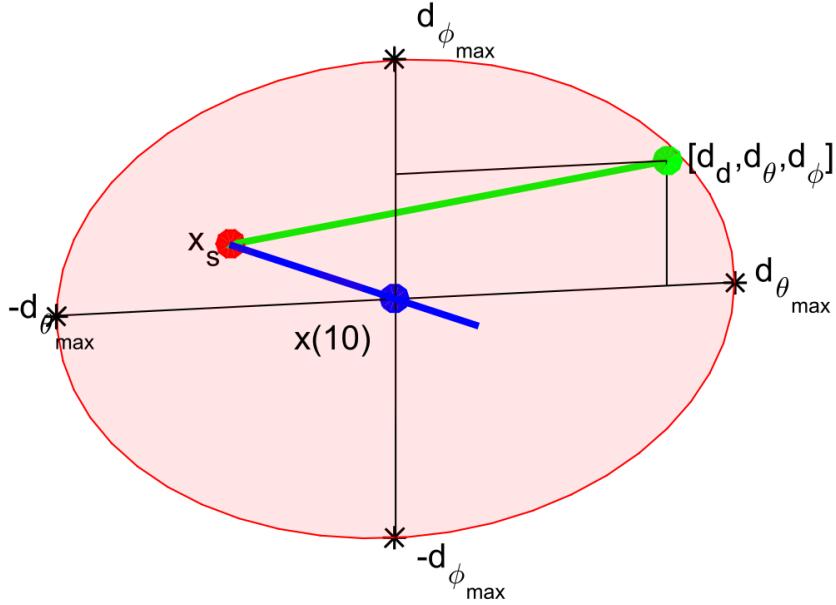


Figure C.1: One rate position $[d_d, d_\theta, d_\varphi]$ (green). deviated from linear trajectory (blue line) at point $x(10)$.(blue) with initial position x_s (red)

The internal coordinate frame $p \in \mathbb{R}^2$ has origin in $x(t) \rightarrow \mathbb{R}^2$. The points of plane p are bounded by projection $p = (b - x(t)) \rightarrow \mathbb{R}^2$, where $b \in D(x(t), v)$. The point of ellipsoidal p is then given as standard ellipse boundary with vertical span $d_\theta(x(t))$ and horizontal span $d_\varphi(x(t))$.

The 2D *Ellipsoid* $E(x(t), v)$ for specific time $t = 10s$ example is portrayed as red ellipsoid (in fig. C.1).

$$E(x(t), v) = \left\{ b \in \mathbb{R}^3 : b \in D(x(t), v), p = (b - x(t)) \rightarrow \mathbb{R}^2, \left(\frac{p(1)^2}{d_\theta(x(t))^2} + \frac{p(2)^2}{d_\varphi(x(t))^2} \right) \leq 1 \right\} \quad (\text{C.11})$$

The expected behavior of an intruder i_k is to stick to predicted linear trajectory $x(t)$ (C.8). The probability of deviation should be decreasing with distance from the ellipse center (fig. C.2.).

Probability density function for ellipsoid $E(x(t), v)$ defined in (eq. C.11) is depending on maximal horizontal spread $d_\theta(x(t))$, maximal vertical spread $d_\varphi(x(t))$, defined by (eq. C.10).

Two standard probabilistic distributions are established $\mathcal{N}(\mu_\theta, \sigma_\theta)$ (eq. C.12) for horizontal spread $\theta(x(t))$ and $\mathcal{N}(\mu_\varphi, \sigma_\varphi)$ (eq. C.13) for vertical spread $\varphi(x(t))$. The means μ_θ and μ_φ are set to zero, and internal coordinate frame $p \in \mathbb{R}^2$ where $x(t) \rightarrow \mathbb{R}^2$ is frame center. The variances σ_θ and σ_φ are set as maximal distances on horizontal/vertical spread axes $d_\theta(x(t))$ and $d_\varphi(x(t))$.

$$P(x(t), d_\theta) = \mathcal{N}(\mu_\theta, \sigma_\theta) = \mathcal{N}(0, d_\theta(x(t))) \quad (\text{C.12})$$

$$P(x(t), d_\varphi) = \mathcal{N}(\mu_\varphi, \sigma_\varphi) = \mathcal{N}(0, d_\varphi(x(t))) \quad (\text{C.13})$$

The combined *probability density function* for maximal spreads d_θ and d_φ is given by (eq. C.14). Because probability density function is defined for internal space $p \in \mathbb{R}^2$ and one may need to calculate impact rate for cell space $c_{i,j,k} \in \mathbb{R}^3$.

The reduction from two parameter probability distribution function to scalar rate distribution function is needed. A scalar rate distribution function $P(x(t), d_\theta, d_\varphi)$ over ellipsoid $E(x(t), v)$ is defined as (eq.C.14), where the final rate is given as an average of two partial probabilities.

Final space intersection rate $P(x(t), d_\theta, d_\varphi)$ needs to be normalized to hold *normal distribution condition* (eq. C.15). Normal distribution condition value (eq. C.15) is given as surface integral over ellipsoid $E(x(0), v)$ with rate distribution function $P(x(t), d_\theta, d_\varphi)$.

$$P(x(t), d_\theta, d_\varphi) = \frac{\mathcal{N}(\mu_\theta, \sigma_\theta) + \mathcal{N}(\mu_\varphi, \sigma_\varphi)}{2} \quad (\text{C.14})$$

$$\iint_{E(x(\tau))} P(x(t), d_\theta, d_\varphi) dd_\theta dd_\varphi = 1 \quad (\text{C.15})$$

Final space intersection rate $P(x(t), c_{i,j,k}, \theta, \varphi)$ (space portion, time portion is calculated in (eq.6.72) is given by (eq. C.17). Its mean value of all intersection rates $P(x(\tau), c_{i,j,k}, \theta, \varphi)$ where $\tau \in [i_e(c_{i,j,k}), i_l(c_{i,j,k})]$ is fixed point in intersection time interval.

An $P(x(\tau), c_{i,j,k}, \theta, \varphi)$ (C.16) is integration of rate density function $P(x(\tau), d_\theta, d_\varphi)$ (eq. C.14) in surface $E(x(\tau), v)$ to cell $c_{i,j,k}$ volume intersection.

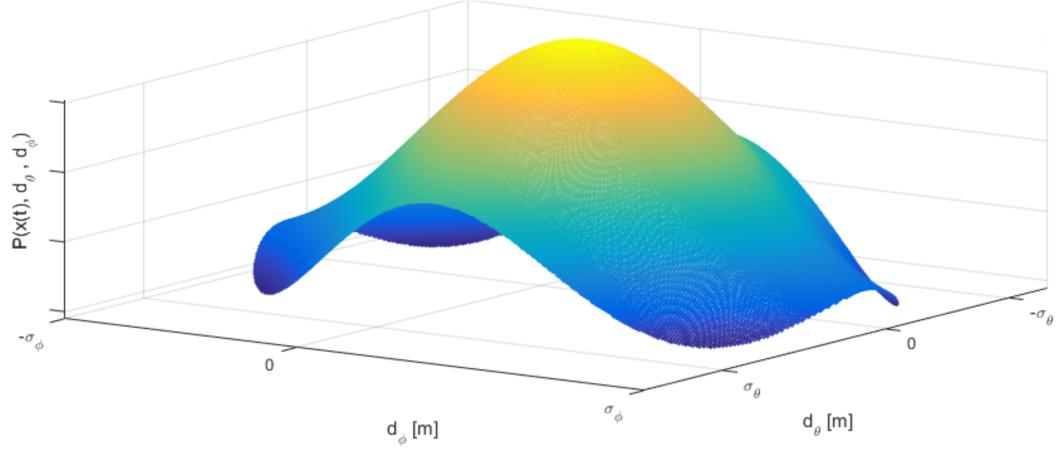


Figure C.2: Probability of intruder i_k position in ellipsoid $E(x(t), v)$

To get a volume integration partial rate in surface intersection must be integrated and normalized in time interval $\tau \in [i_e(c_{i,j,k}), i_l(c_{i,j,k})]$, the *base intersection probability* $P_T(i_k(x_s, v, \theta, \varphi), c_{i,j,k})$ is given by (eq. C.17). Example of intersection of intruder i_r uncertain ellipsoid cone with avoidance grid $\mathcal{A}(t_i)$ is given in (fig. C.3).

$$P(x(\tau), c_{i,j,k}, \theta, \varphi) = \iint_{E(x(\tau), v) \cap c_{i,j,k}} P(x(\tau), d_\theta, d_\varphi) \quad (\text{C.16})$$

$$P_T(i_k(x_s, v, \theta, \varphi), c_{i,j,k}) = \frac{\int_{i_e(c_{i,j,k})}^{i_l(c_{i,j,k})} P(x(\tau), c_{i,j,k}, \theta, \varphi) d\tau}{i_l(c_{i,j,k}) - i_e(c_{i,j,k})} \quad (\text{C.17})$$

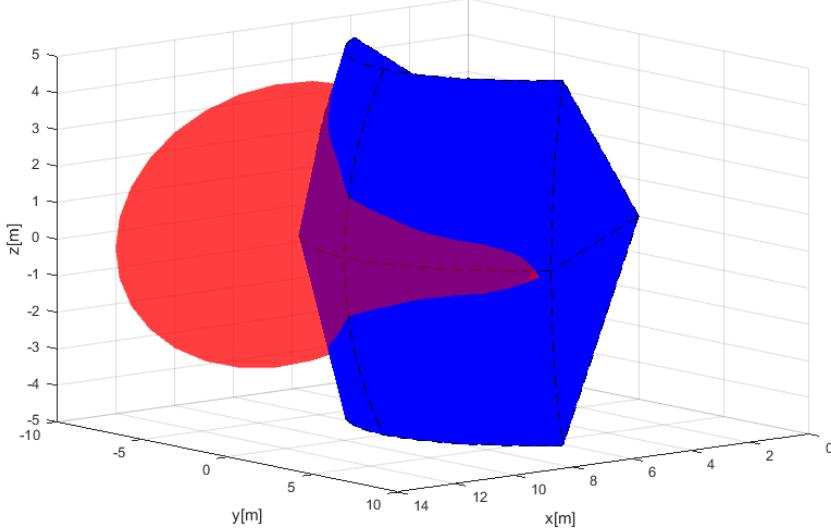


Figure C.3: Avoidance grid $\mathcal{A}(t_i)$ (blue) intersection with elliptic cone intruder $i_k(x, v, \theta, \varphi)$ (red) example.

A numeric approximation of space intersection rate $P_T(i_k(x_s, v, \theta, \varphi), c_{i,j,k})$ is more implementation feasible than symbolic calculation due to the multiple intersection constraints and bad intersection algorithm complexity.

Let us define a homogeneous discrete subset of real numbers \mathcal{R} which is a non-empty subset of real numbers \mathbb{R} . The set \mathcal{R} (eq. C.18) is homogeneous that means for an equal interval $(i, i + 1], i \in \mathbb{Z}$ subset the count of members is equal to some positive natural number k . The parameter k can be understood as *unit approximation density*.

Similarly, the power sets $\mathcal{R}^2 \subset \mathbb{R}^2, \mathcal{R}^3 \subset \mathbb{R}^3, \dots, \mathcal{R}^i \subset \mathbb{R}^i, i \in \mathbb{N}^+$ keeps homogeneous distribution.

$$\mathcal{R} = \left\{ a \in \mathbb{R} : \forall i \in \mathbb{Z}, |i < a \leq i + 1| = k, k \in \mathbb{N}^+, \right. \\ \left. \forall j \in \mathbb{N}^+ a_{j+1} - a_j = m, m \in \mathbb{R}^+ \right\}, \mathcal{R} \subset \mathbb{R} \quad (\text{C.18})$$

The orthogonal plane for $x(t), v, t \in \mathbb{R}$ is defined by (eq. C.9). The orthogonality property is also kept for any subspace $\mathcal{R}^n \in \mathbb{R}^n, n \in \mathbb{N}^+$. Numeric approximation of $D(x(t), v)$ is given as $D_D(x(t), v)$ (eq. C.19).

The only difference is that discrete approximation is countable $|D_D| = m, m \in \mathbb{N}^+$, but continuous representation $|D| \approx \infty$ is uncountable. Because ellipsoid is a subset of orthogonal plane it keeps its countability property; therefore E_D is also countable and

must contain at least one member.

$$D_D(x(t), v) = \{a \in \mathcal{R}^3 : (a - x(t)) \perp v, \}, t \in \mathcal{R} \quad (\text{C.19})$$

The *base ellipsoid* $E(x(t), v)$ for continuous-space is given by (eq. C.11). Every element, expect the base of internal projection \mathcal{R}^2 and orthogonal plane D_D is same in discrete case $E_D(x(t), v)$ (eq. C.20).

$$\bar{E}_D(x(t), v) = \left\{ b \in \mathcal{R}^3 : b \in D_D(x(t), v), p = (b - x(t)) \rightarrow \mathcal{R}^2, \left(\frac{p(1)^2}{d_\theta(x(t))^2} + \frac{p(2)^2}{d_\varphi(x(t))^2} \right) \leq 1 \right\}, t \in \mathcal{R} \quad (\text{C.20})$$

The *numeric calculation disproportion* can occur in case that ellipsoid $\bar{E}_D(x(t), v)$ (C.20) in case of $d_\theta(x(t)) \approx 0$ and $d_\varphi(x(t)) \approx 0$. The count of ellipsoid members can be $|\bar{E}_D(x(t), v)| = 0$, which is in contradiction with assumption $|\bar{E}_D(x(t), v)| \neq 0$.

Let assume for discrete times $\tau = \{t_1, t_2, \dots, t_i\}$, $i \in \mathbb{N}^+$ there exists ellipsoids $\bar{E}_D(x(t_1), v), \bar{E}_D(x(t_2), v), \dots, \bar{E}_D(x(t_i), v)$ which are non empty and in space \mathcal{R}^2 in internal coordinate frame and space \mathcal{R}^3 in avoidance grid $\mathcal{A}(t_i)$ coordinate frame. The intersection of these partial ellipsoids in both spaces is equal to:

$$\bar{E}_D(x(t_1), v) \cap \bar{E}_D(x(t_2), v) \cdots \cap \dots \bar{E}_D(x(t_i), v) = \emptyset \quad (\text{C.21})$$

An *empty intersection* enables us to keep homogeneity property of ellipsoids by adding points so it is safe to add specific point $x(t)$ into empty ellipsoid. But only one, because it does not impact probability density functions $\mathcal{N}(\mu_\theta, \sigma_\theta)$ and $\mathcal{N}(\mu_\varphi, \sigma_\varphi)$, neither space intersection rate density function $P(x, d_\theta, d_\varphi)$.

The final ellipsoid used forward $E_D(x(t), v)$ (eq. C.22) is keeping all properties of ellipsoid $E(x(t), v)$ (eq. C.22).

$$E_D(x(t), v) = \begin{cases} |\bar{E}_D(x(t), v)| = 0 & : \{x(t)\} \\ |\bar{E}_D(x(t), v)| \geq 0 & : \bar{E}_D(x(t), v) \end{cases} \quad (\text{C.22})$$

The normal distribution condition for rate distribution function $P_D(x(t), d_\theta, d_\varphi, p)$, which is instance of to rate density function $P(x(y), d_\theta, d_\varphi)$ (eq. C.14) is used. This rate distribution must be normalized according to (eq. C.23).

$$\sum_{p \in E_D(x(t))} P_D(x(t), d_\theta, d_\varphi, p) = 1, \forall t \in \mathcal{R}^+ \quad (\text{C.23})$$

The equations for *space intersection rate* are similar to (eq. C.16, C.17). For cell $c_{i,j,k}$ there exist intruder entry time $i_e(c_{i,j,k})$ its the earliest intersection with ellipsoid $E_D(x(i_e(c_{i,j,k}))), v$. Same situation occurs with intruder leave time $i_l(c_{i,j,k})$. Because E_D is countable set, it means additional attributes can be attached to each point $p \in E_D$.

Based on system dynamic (eq. 6.68) the *Time Of Arrival* (TOA) can be calculated. The example of TOA is given in fig. C.4.

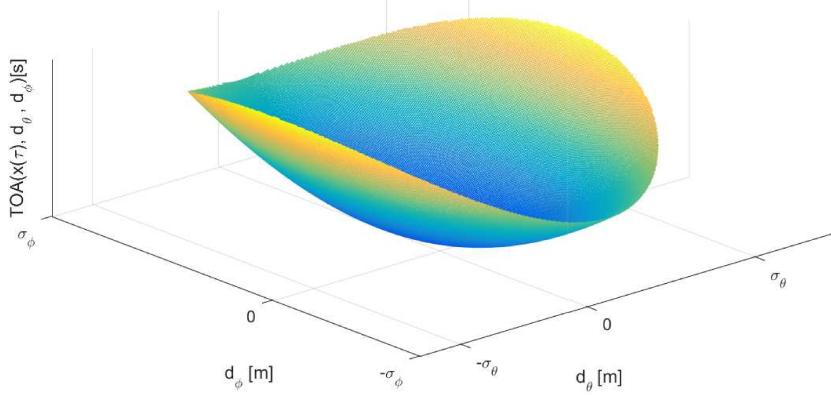


Figure C.4: Time Of Arrival (TOA) for one ellipsoid $E_D(x(\tau), v)$.

The intersection rate $P_D(x(\tau), c_{i,j,k}, \theta, \varphi)$ for one time sample τ is given by (eq. C.24), which has similar notation to (eq. C.16), sums are used instead of integrals and discrete rate density function $P_D(x(\tau), d_\theta, d_\varphi, p)$ for points form ellipse and cell intersection are used as iterator base set $p \in \{E_D(x(\tau), v) \cap c_{i,j,k}\}$.

$$P_D(x(\tau), c_{i,j,k}, \theta, \varphi) = \sum_{p \in \{E_D(x(\tau), v) \cap c_{i,j,k}\}} P_D(x(\tau), d_\theta, d_\varphi, p) \quad (\text{C.24})$$

The *space intersection rate* $P_{TD}(i_k(x_s, v, \theta, \varphi), c_{i,j,k})$ (eq. C.25) is given as mean intersection rate of partial intersections $P_D(x(\tau), c_{i,j,k}, \theta, \varphi)$ where step set $T = \{i_e(c_{i,j,k}), \dots, i_l(c_{i,j,k})\}$ contains all viable intersection times with ellipsoids $E(x(\tau \in T), v)$. The denominator is basically count of samples in sample time set T .

$$P_{TD}(i_k(x_s, v, \theta, \varphi), c_{i,j,k}) = \frac{\sum_{\tau=i_e(c_{i,j,k})}^{i_l(c_{i,j,k})} \sum_{p \in E_D(x(\tau), v)} P_D(x(\tau), c_{i,j,k}, \theta, \varphi, p)}{\sum_{\tau=i_e(c_{i,j,k})}^{i_l(c_{i,j,k})} 1} \quad (\text{C.25})$$

An *intersection of intruder cone and cell* $c_{i,j,k}$ cell is defined by (eq. C.26) The set of point $p \in \mathbb{R}^3$ where condition of intersection between ellipsoids $E_D(x(\tau), v)$ for times $\tau \in \mathbb{R}^+$ and cell space $c_{i,j,k}$ is met.

$$\mathcal{P}(i_k(x_s, v, \theta, \varphi), c_{i,j,k}) = \bigcup_{\forall \tau \in \mathbb{R}^+} \{p \in \mathbb{R}^3 : p \in c_{i,j,k} \cap E_D(x(\tau), v)\} \quad (\text{C.26})$$

An *intruder time of entry* $i_e(i_k, c_{i,j,k})$ (eq. C.27), for intruder i, k and cell $c_{i,j,k}$ is approximated for discrete point set $\mathcal{P}(i_k(x_s, v, \theta, \varphi), c_{i,j,k})$ (eq. C.26) as minimal time of arrival $t_{TOA}(p)$ of member points p .

$$i_e(i_k, c_{i,j,k}) \approx \min \{t_{TOA}(p) : p \in \mathcal{P}(i_k(x_s, v, \theta, \varphi), c_{i,j,k})\} \quad (\text{C.27})$$

An *intruder time of leave* $i_l(i_k, c_{i,j,k})$ (eq. C.28), for intruder i, k and cell $c_{i,j,k}$ is approximated for discrete point set $\mathcal{P}(i_k(x_s, v, \theta, \varphi), c_{i,j,k})$ (eq. C.26) as maximal time of arrival $t_{TOA}(p)$ of member points p .

$$i_l(i_k, c_{i,j,k}) \approx \max \{t_{TOA}(p) : p \in \mathcal{P}(i_k(x_s, v, \theta, \varphi), c_{i,j,k})\} \quad (\text{C.28})$$

Combined intersection model: The *combined intersection model* $P_{OI}(i_k, c_{i,j,k}, l, b, s, \tau)$ is defined for intruder i_k with parameters:

1. *Starting position* x_s - expected position of intruder i_r in 3D space at time of avoidance t_i in avoidance grid frame $\mathcal{A}(t_i)$.
2. *Velocity vector* v - oriented velocity of intruder i_r at time of avoidance t_i in avoidance grid frame $\mathcal{A}(t_i)$.
3. *Horizontal uncertainty spread* θ - defines how much can intruder i_r deviate on horizontal axis of intruder local coordinate frame (if X+ is the main axis, then Y is horizontal axis in right-hand euclidean coordinate frame), due the properties of intersection definition, the horizontal uncertainty spread can have following values $\theta \in [0, \pi/2]$.
4. *Vertical uncertainty spread* φ - defines how much can intruder i_r deviate on vertical axis of intruder local coordinate frame (if X+ is the main axis in local right-hand euclidean intruder coordinate frame, then Z is horizontal-vertical axis), due to the intersection definition, the vertical uncertainty spread can have following values $\varphi \in [0, \pi/2]$.
5. *Body volume radius* r - defines the body volume of an intruder in meters and it has \mathbb{R}^+ value.

The *flag vector* $l, b, s, \tau \in \{0, 1\}$ is a parametrization of rate calculation: l stands for the *lined intersection*, b stands for *body intersection*, s stands for the *spread intersection*, τ stands for *time account*.

The *space intersection for line* $P_L(i_k, c_{i,j,k})$ is defined as $P_T(i_k(x, v), c_{i,j,k})$, where i_k is intruder with properties of initial position x , velocity vector v and $c_{i,j,k}$ is target cell. (eq. C.3).

The *space intersection rate for body volume* $P_B(i_k, c_{i,j,k})$ is defined as $P_T(i_k(x, v, r), c_{i,j,k})$ (eq. C.7), where intruder i_r has additional property of the intruder body volume radius r .

The *space intersection probability for maneuverability uncertainty* $P_S(i_k, c_{i,j,k})$ is defined as $P_{TD}(i_k(x_s, v, \theta, \varphi), c_{i,j,k})$ (eq. C.25), where intruder properties θ, φ stands for intruder horizontal and vertical uncertainty spread.

The *time intersection rate* $P_{\tau,x}(i_k, c_{i,j,k}) \in [0, 1]$ is defined in (eq. 6.72). This probability has two calculation modes, first is for 1D intersection (line), second is for volume intersection (body volume, spread elliptic cone).

UAS cell entry time t_e and cell leave time t_l time for a vehicle in avoidance grid $\mathcal{A}(t_i)$ is given by (eq. 6.69) and (eq. 6.70).

Intruder leave and entry time for 1D intersections is trivial and is omitted in this section. Intruder entry i_e and intruder leave i_l for 3D intersection is given by (eq. C.27, C.28).

All partial rates with respective definition references are summarized in (eq. C.29)

$$P_L(i_k, c_{i,j,k}) = P_T(i_k(x, v), c_{i,j,k}) \quad (C.3)$$

$$P_B(i_k, c_{i,j,k}) = P_T(i_k(x, v, r), c_{i,j,k}) \quad (C.7)$$

$$P_S(i_k, c_{i,j,k}) = P_{TD}(i_k(x_s, v, \theta, \varphi), c_{i,j,k}) \quad (C.25) \quad (C.29)$$

$$P_{\tau,x}(i_k, c_{i,j,k}) = \frac{\|[i_e(c_{i,j,k}), i_l(c_{i,j,k})] \cap [t_e, t_l]\|}{\|[t_e, t_l]\|} \quad (6.72)$$

With definition of all space and time intersection rates (eq. C.29) and given flag vector $l, b, s, \tau \in \{0, 1\}$ one can formulate combined intersection rate $P_{O_I}(i_k, c_{i,j,k}, l, b, s, \tau)$ (eq. C.30) for intruder i_k and cell $c_{i,j,k}$. The principle is following: *maximum of selected rates product based on flag vector is final intersection rate of intruder i_k in the cell.*

The time-use flag τ is adding time intersection rate $P_{\tau,x}(i_k, c_{i,j,k})$, where time intersection rate is defined by $x = \{L, B, S\}$ for line, body volume, spread ellipse time intersections ($P_{\tau,L}(i_k, c_{i,j,k}) \neq P_{\tau,B}(i_k, c_{i,j,k}) \neq P_{\tau,S}(i_k, c_{i,j,k})$ for one intruder i_k).

$$P_{O_I}(i_k, c_{i,j,k}, l, b, s, \tau) = \begin{cases} \tau = 0 & : \max \left\{ P_L(i_k, c_{i,j,k}).l, P_B(i_k, c_{i,j,k}).b, P_S(i_k, c_{i,j,k}).s \right\} \\ \tau = 1 & : \max \left\{ P_{\tau,L}(i_k, c_{i,j,k}).P_L(i_k, c_{i,j,k}).l, P_{\tau,B}(i_k, c_{i,j,k}).P_B(i_k, c_{i,j,k}).b, P_{\tau,S}(i_k, c_{i,j,k}).P_S(i_k, c_{i,j,k}).s \right\} \end{cases} \quad (C.30)$$

Appendix D

Conflict Resolution Schemes

D.1 Cooperative Conflict Resolution

Idea: There is a *final decision maker* (absolute authority) in conflict resolution. This authority is *UTM* or *air traffic attendant* with higher priority. The future *UTM system* is such authority. The approach to mixed conflict resolution is mentioned in [191], based on navigation [192]. This is similar to our approach.

Note. Open Issue: Decentralized model with UTM as an approver of directives is possible, but that is a topic for own research.

Goal: UAS is obligated to follow up committed mission plan with given precision. There is one to five percent allowed deviations for ATM mission plans. Similar rates are achievable according to [191]. This requirement is given by [3] ICAO 4444 document for ATM operations.

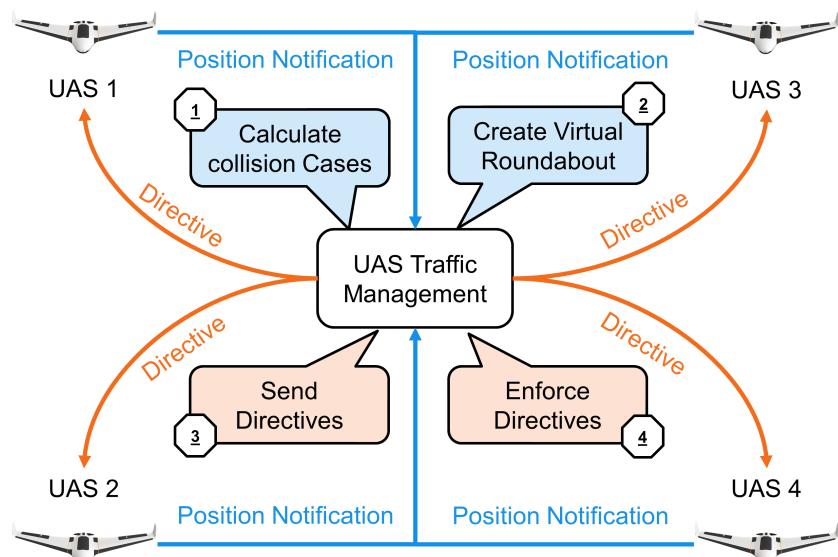


Figure D.1: Cooperative conflict resolution via UTM authority.

Cooperative Conflict Resolution (fig. D.1) shows a functional diagram of one *UTM time-frame* there are following actors:

1. *Unmanned Autonomous System* (UAS) equipped with necessary navigation and communication modules, providing the unique *identification number*.
2. *UAS Traffic Management* (UTM) posing as the central authority for given *airspace cluster*.

The following steps are executed during *Cooperative conflict resolution*:

1. $UAS_* \rightarrow UTM$ *Send position notification* - each *UAS* is notifying the authority (UTM)
2. $\circlearrowleft UTM$ *Calculate collision Cases* - UTM gathers data and predicts possible collisions then it tries to link them and manage the situation.
3. $\circlearrowleft UTM$ *Create virtual Roundabout* - active collision cases are aggregated into a virtual roundabout.
4. $UTM \rightarrow UAS_*$ *Send directives* - UTM sends commands to UAS systems which need to change their planned trajectories.
5. $UTM \rightarrow UAS_*$ *Enforce directives* - UTM is periodically checking constraints imposed in previous *decision frames*.

D.2 Non-Cooperative Conflict Resolution

Idea: There is *main UAS(1)* which is flying in open *non-controlled* airspace. Other UAS are operating in its vicinity. It is expected that they are claiming their *planned trajectories*. The *Main UAS(1)* detects the collision with other *UAS(2-4)*.

There is no *final decision maker* nor *supervising authority*; all communication participants have a similar level of rights.

Note. There is an assumption that other airspace users are behaving like intruders, without intent to destroy or harm. The *adversarial behavior* is not accounted. The response from an *intruder* is not mandatory in *non-controlled* airspace.

Goal: Provide *mutual avoidance mechanism* in *non-controlled* airspace. Let us consider the equal standpoint of all airspace attendants.

Conflict Resolution: The conflict resolution depends on current mode and *handshake* between airspace attendants. The non-cooperative behavior has been implemented as follows:

1. *Navigation mode* - every *airspace attendant* is calculating own *collision cases* and checking the behavior of the other (virtual UTM).
2. *Emergency avoidance mode* - is depending on communication mode:
 - a *Response mode* - claiming separation methods and using avoidance mechanism (Avoidance grid with intruder model in our case).
 - b *Blind mode* - every conflict side picks own strategy respecting given *rules of the air*.

Note. Intruder Intersection model selection: UAS based on Event detects possible collision for some reason UTM directive is out of the question, then try to claim separation (body volume intruder model (app. C.2)), If separation fails, go full survival mode (uncertain intruder model (app. C.2)).

Special Cases in Manned Aviation: There are IFALPA reports which can give us an overview of *enforced non-cooperative* mode causes in *controlled airspace*:

1. *VFR disabled* - flying in fog or thick clouds can render pilot vision, similar to UAS cameras/LiDAR.
2. *IFR equipment broke* - the sensor malfunction is more likely to happen due to the lesser redundancy in UAS systems.
3. *C2C Link disabled* - communication loss is more likely to happen, due to the lesser redundancy.

4. ATM failure - the ground control module of UTM can also fail.

Note. Traffic management related fails are lesser than 0.001 cases per one flight (according to IFALPA [193]).

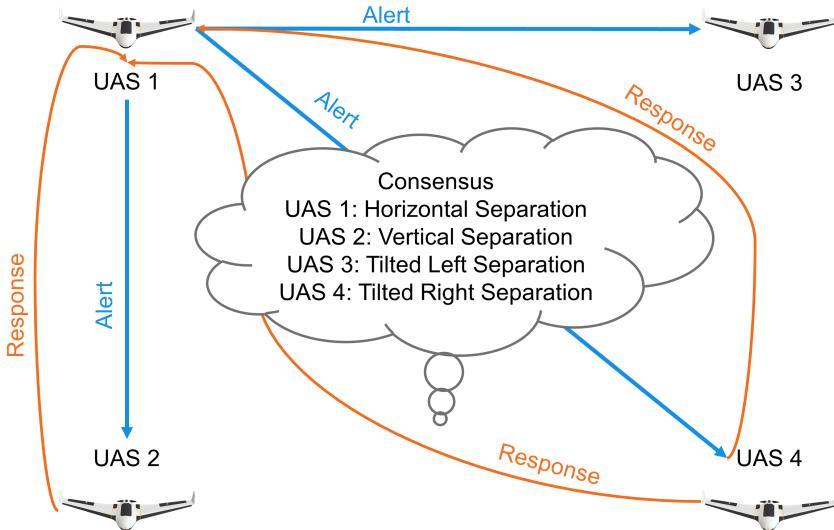


Figure D.2: Non-cooperative conflict resolution via UAS claims.

Response mode scenario example: The *main UAS(1)* is going to collide with other *UAS(2-4)*:

1. $UAS(1) \rightarrow UAS(2 - 4)$ sends position and heading notification.
2. $\bigcirc UAS(2 - 4)$ calculates possible collisions.
3. $UAS(2 - 4) \rightarrow UAS(1)$ sends a response to the *main UAS(1)* with claimed separation mode.
4. $\bigcirc UAS(1)$ acknowledges proposed *separation modes*.
5. $\bigcirc UAS(1 - 4)$ avoids each other using claimed separation mode because every *UAS* achieved *consensus*.

Note. The mutual consensus is not usually achieved via C2 communication. The most common case is *assuming separation mode*. This case is shown in (sec. 7.3.7)

Appendix E

Additional UTM functionality

This appendix contains additional UTM functionality description and detailed rule body implementations.

E.1 Weather Case Implementation

Motivation: The weather, as defined in (eq. 4.5), impacts flight and system dynamics; therefore it impacts the *reach set* is impacted. The *weather impact* can be solved by policy application:

1. *Weather Acceptance* - for bigger *UAS* the normal weather impact does not pose a significant risk. The *segmented movement automaton* (def. 35) with *Weather situation* as the discrete state is used.
2. *Weather Avoidance* - all *weather impact zones* are considered as hard constraints with protective *soft-constraint* around.
3. *Combined approach* - depending on the type of impact and declared *UAS* impact resistance the zones are divided into *soft* and *hard* constraints.

Note. This work handles small *UAS* avoidance; these are very sensitive to any weather impact; therefore *Weather impacted areas* will be considered as *hard constraints with soft constraint protection zone*.

The original *weather impact zone* is considered as obstacle body and enforces the body margin.

The surroundings of *weather impact zone* up to *safety margin* distance are considered as *soft constraint zone* (implemented as a bloated polygon).

Purpose: The *weather case* (tab. E.1) is broadcasted by *Airspace Authority* to *impacted area*, each *UAS* then change their mission according to *their maneuvering capabilities*. Each trajectory must lead away from the *constrained area*. The algorithm used for intersection selected based on [160] the selected algorithm *Shamos-Hoey* [161].

Constrained Area: Constrained area can be defined as *static* (sec. 6.5.3) or dynamic constraint (def. 20). The *constraint center* is defined on horizontal plane like follow:

$$\text{ConstraintCenter} = \text{center} \in [\text{latitude}, \text{longitude}] \quad (\text{E.1})$$

The *Convex Polygon* boundary is defined on horizontal plane, contains at least 3 vertexes:

$$\text{ConvexPolygon} = \{\text{point}_i : \text{point}_i \in [\text{latitude}, \text{longitude}], i \geq 3\} \quad (\text{E.2})$$

The *Vertical constraint* is defined as *range of barometric altitude* (Above Mean Sea Level):

$$\text{VerticalConstraint} = [\text{startAltitude}, \text{endAltitude}] \quad (\text{E.3})$$

| Constrained area | |
|------------------------------|---|
| center position | is given as a geometrical <i>center point of the boundary</i> . |
| boundary | is represented as a <i>convex polygon</i> on the latitude-longitude plane. |
| start altitude | is lower boundary barometric altitude given at above mean sea level, where given weather factor has a significant impact. |
| end altitude | is upper boundary barometric altitude given at above mean sea level, where given weather factor has a significant impact. |
| Additional parameters | |
| type(s) | lists weather events occurring in the <i>constrained area</i> . |
| severity list | is recorded for each plane <i>category</i> |
| start | indicates when weather constraint was established. |
| expected end | of weather constraint. |
| velocity | indicates if weather phenomenon is moving. |
| Miscellaneous | |
| previous | reference to <i>weather constraint</i> decision time-frame data. |
| impacted | list of possibly impacted attendees (planes which obtained divergence order or warning from UTM). |

Table E.1: Static/Dynamic weather constraint for given decision time-frame.

Additional parameters : Following additional parameters with additional purpose can be attached to *Weather Constraint*.

1. *Type* - defines required resistance - moisture, temperature, wind.
2. *Severity* - defines the impact for each *aircraft category*, this is used in soft/hard type assessment.
3. *Duration* - start and end of *constraint* validity, if not defined valid for all *UAS mission time*.

4. *Velocity* - velocity and last position assessment time.

Note. Our implementation does not consider the *type* or *severity*. All *weather impact* is considered as a *hard constraint*. The velocity differentiates *static* ($= 0$)/*moving* (> 0) *constraints*.

Avoidance System: Resolve similar to *Converging/Overtake Maneuver* depending on the *angle of approach*. The *virtual roundabout* is utilized for *static constraints*; the *intruder model* is utilized for *dynamic constraints*.

E.2 Rule: Detect Collision Cases

This rule is activated each *UAS avoidance run*. *UTM* sent out all related *collision cases* (E.4) based on our *UAS identifier*. Creation of *collision case* is given in (sec. 6.7.6) based on air traffic periodical *position notifications* (sec 6.7.5).

$$UTM \times timeFrame \rightarrow UTMCollisionCases \quad (E.4)$$

If there are available *position notifications* (sec 6.7.5) from surrounding air-traffic, UAS will calculate own *collision cases* (E.5).

$$uasStatus \times positionNotification \times utmTimeFrame \rightarrow UASCollisionCases \quad (E.5)$$

Then UAS merges *own collision cases* with *UTM collision cases*, if there exist following disparities UAS will take action:

1. $distamce(ownCollisionPoint, utmCollisionPoint) \geq threshold$, send UTM notification, use *utmCollisionPoint*
2. $utmMargin \geq ownMargin$, use safety margin from UTM.
3. $utmAvoidanceRole == active, ownAvoidanceRole == inactive$, use UTM avoidance role.
4. $utmCollisionCase == active, ownCollisionCase == uncertain$, use UTM provided collision case, not all *position notifications* are available.
5. $utmCollisionCase == inactive, ownCollisionCase == active$, notify UTM with new collision case, ignore collision case until UTM approves.

Note. *Avoidance role* is classified as *inactive* if and only if UAS has the *right of way*, it is classified as *active* otherwise.

Safety margin determined by UTM has priority because not all calculations factors are available for UAS.

Collision Case unknown to UTM are ignored, due to safety reasons (false data spoofing), collision case is activated after UTM confirmation. If there is real intruder not confirmed by UTM, it is handled via *non-cooperative* or *emergency* avoidance procedure

The *selection process* of active *collision cases* is based on UAS *avoidance role* in each *collision case*.

1. If the *avoidance roles* are following: *Head On Approach*, *Converging Maneuver*, or *Overtake* in all *collision cases* UAS system will stay in cooperative mode.
2. If there exists at least one *collision case* with *own avoidance role* or *intruder avoidance role* set as *avoid-emergency*, the UAS will notify UTM and ask for *diversion order*; meanwhile it sets itself into *Emergency avoidance mode*.
3. If there exist multiple *Overtake avoidance roles* or combination of *Overtake avoidance role* and *Another active role*, the UAS will decrease its cruising speed like follows:

$$UASSpeed = \max \left\{ \begin{array}{l} \text{minimalUASCruisingSpeed}, \\ \min \{ \text{intruderSpeed} \} \quad \forall \text{activeCollisionCases} \end{array} \right\} \quad (\text{E.6})$$

During *slow-down* UAS switches to *emergency avoidance mode* and asks for *divergence order* from UTM.

The *ordering of collision cases* starts if and only if the *UAS* is in *cooperative avoidance mode*. The cases are ordered for processing based on severity rating which is calculated based on:

1. *Safety Margin* - the greater safety margins are prioritized.
2. *Intruder vehicle class* - the more dangerous intruders are prioritized.
3. *Collision point distance* - closer collision points are prioritized.
4. *UAS avoidance role* - *Head on Approach* is favored upon *Converging maneuver*, due to direct collision severity.

Rule engine invocation for each *active collision case* is then applied on *descending severity sorted list*.

The rule is summarized in table E.2.

| |
|---|
| <p><i>Invocation:</i> Every <i>Decision point</i> in <i>UAS main loop</i></p> <p><i>Objective:</i></p> <ol style="list-style-type: none"> 1. Fetch UTM <i>Collision cases</i> for a given decision time frame. 2. Create/update <i>own collision cases</i> based on received <i>Position notifications</i> from surrounding <i>Intruders</i>. 3. Merge <i>Collision cases</i> based on <i>UTM priority order</i>. 4. Select active <i>collision cases</i> based on the following conditions: <ol style="list-style-type: none"> a. <i>Active participation</i> in <i>collision case</i> where <i>avoidance role</i> \neq <i>Right of the way</i>. b. <i>Collision point</i> is in the front of UAS. c. <i>Emergency mode detection</i> there exists at least one non-cooperative participant. 5. Order <i>collision cases</i> based on <i>severity</i>. 6. If there is at least one <i>active collision case</i> enforce rule <i>Resolve collision case</i> (tab E.3) for each <i>active collision case</i>. |
|---|

| <i>Context</i> | <i>Condition</i> | <i>Application</i> |
|--|---------------------------------------|---|
| UAS Mission control, Before Avoidance Run, UTM/UAS collision cases | Clean avoidance grid, No emergency | Active collision case selection, Prioritization |

Table E.2: Detect collision cases rule definition.

E.3 Rule: Resolve Collision Case

Active collision cases are processed one by one. All collision cases are applied to *Navigation grid*. *Navigation grid* contains all possible *trajectories* in the form of *Reach set*. All *trajectories* are *reachable* at the beginning of the UAS *avoidance frame*. Each application of *collision case resolution* rule disables some subset of feasible *trajectories*. For this reason are *active collision cases* sorted by severity.

It is assumed that UAS is in *cooperative avoidance mode*. If the previous application of this rule forced UAS into *emergency mode*, the rule is not applied to save system resources. *Emergency mode* is invoked if *rule application* disables all *trajectories* in *Navigation grid*. If there is at least one *feasible trajectory* in *avoidance grid* follow-up rule is invoked based on UAS *avoidance role*.

The rule is summarized in table E.3.

Invocation: This rule is invoked if exists at least one *active collision case* in given *navigation grid time-frame*; moreover *avoidance grid* must be empty and *cooperative avoidance mode* is enforced.

Objective: Based on *active collision case* and *UTM directives* enforce behavior based on *own avoidance role*:

1. *Head on approach* - rule E.5.
2. *Converging maneuver* - rule E.6.
3. *Overtake* - rule E.7.
4. *Emergency mode* - switch from *active avoidance mode* to *emergency mode*.

| <i>Context</i> | <i>Condition</i> | <i>Application</i> |
|---|---|---|
| UAS mission control, Trajectory restriction, Collision cases, | Active merged collision case, Resolution mandate from UTM | Enforce Rules of Air or Enforce emergency |

Table E.3: Resolve collision case rule definition.

E.4 Rule: Close Collision Cases

Collection of rule results detected by rule E.2 and resolved by rule E.3 is done via the context of the rule engine. For each *time-frame* and each *trajectory* \in *NavigationGrid*, there exists rule engine *context* query (E.7) which returns *trajectory status* and *list of applied rules on trajectory*.

$$\text{Context}(\text{trajectory}, \text{timeFrame}) \rightarrow \{\text{State : Enabled/Disabled, Rule(s)}\} \quad (\text{E.7})$$

Calculation of possible trajectories in navigation grid is using *collected rule results* (E.7). If the *trajectory state* and linked *rule reason* are sufficient, the *trajectory* is disabled for the the given *time frame*. *Standard navigation algorithm* is used (sec. 6.6.2) to select *feasible trajectory*.

Rules of the air and their application in *General Aviation* cases is consistent. Increasing traffic density can impose new layers of rules, which may cause the *soft deadlock* in *maneuverability*. In this case, *Navigation grid* will have all *possible trajectories* exhausted. The following procedure is executed:

1. UAS switch into *Non-cooperative avoidance mode* or *Emergency avoidance mode* depending on situation severity (One conflict can be handled with *vertical separation* of conflicting aircraft).
2. UAS broadcasts *warning message* to all nearby aircraft, and *separation message(s)* to conflicting aircraft. *Separation message* contains an *expected collision point* and *preferred separation type*. Each conflicting aircraft then reacts and sends *action notification* to UTM.
3. If UAS switches into *emergency mode*, non-cooperative avoidance using *avoidance grid* is induced. Each relevant intruder is projected as *timed body volume intruder* (app. C.2), where *safety margin* is used as *body radius*.

UAS notifies *UTM* with *course change*, *planned avoidance trajectory*, *avoidance mode*. *UTM* approves planned changes or sends *plan corrections* (out of scope). The rule summary is given in table E.4.

Invocation: There exists at least one *active collision case* which had an impact on *Navigation grid*.

Objective: Ensure that multiple *avoidance rules* application gives feasible *avoidance strategy*, enter into *emergency avoidance mode* otherwise. Following steps are executed:

1. *Collect rules applied on navigation grid from active collision cases.*
2. *Calculate possible trajectories for avoidance;* there may be none.
3. If there is no *feasible route*, for each *intruder* from related *collision cases*:
 - a. Issue *warning message* containing *expected collision point* and *preferred separation type*.
 - b. Create appropriate *intruder object* for *avoidance grid*.
 - c. Calculate *evasive maneuver* based on the expected *separation type*.
4. *Notify UTM with collision case resolution* for each *active collision case*. *Notify UTM with planned trajectory and avoidance mode*

| <i>Context</i> | <i>Condition</i> | <i>Application</i> |
|---|---|---|
| UAS Mission control, After avoidance run, Collision resolutions | At least one trajectory in Navigation grid, Emergency check | Force <i>Emergency mode</i> OR Close Collision Case |

Table E.4: Close collision case rule definition.

E.5 Rule: Head on Approach

Rule (E.5) is invoked based on the *angle of approach* range condition, defined *collision case* section 6.7.6. The handling of *head on* avoidance is given in section 6.7.2.

Virtual round-abound for UAS and intruder is created by UTM. The center of virtual round-abound and *corrections for participants margins* are determined based on:

1. *Collision case center* - contributes to the round-abound center median point.
2. *UAS and intruder maneuverability* - determines *attendants avoidance mode* and *maximal avoidance margins*.
3. *Surrounding air-traffic* - contributes to round-abound center median point, determines ideal *ideal avoidance margins* due to *wake turbulence prevention*.

Invocation: When *UAS avoidance role* is *Head on avoidance* and *avoidance grid* is empty.

Objective: Ensure that the *UAS body* does not enter into *intruder's well clear zone*.

1. Prevent *left-side leading* maneuvers (rule E.8).
2. Prevent head on *safety margin* breach(rule E.9).
3. Return to original course, when *navigation grid* is clear.
4. Prevent *wake turbulence* (by safety margin correction).
5. Enforce *Round-about* behavior (by clustering collision cases).

| <i>Context</i> | <i>Condition</i> | <i>Application</i> |
|--|------------------|--|
| UAS Navigation Grid, Collision Point, Avoidance role | None | Run rules referenced in objective listing. |

Table E.5: Head on Approach rule definition.

The *virtual round-about center* is calculated as *corrected median* (E.8) taking *cluster of collision cases* and calculates the median of their collision points corrected by *weather* and *wake turbulence* factor.

$$\begin{aligned} \text{correctedMedian} = & \sum_{c_i \in \text{collisionCases}} (c_i.\text{center} + \text{correction}) / \text{count}(\text{collisionCases}) + \\ & + \text{correction}(\text{Weather}) + \text{correction}(\text{WakeTurbulence}) \end{aligned} \quad (\text{E.8})$$

Corrected margin needs to be calculated for each *participating aircraft*, because of the *virtual roundabout center* correction (E.8). Each *round-about participant* is ordered based on importance (lowest maneuverability first). Then for each *round-about participant* obtains *corrected margin* (E.9) calculated from *collision case safety margin*, corrections based on other *more important vehicles*, *weather*, *wake turbulence*.

$$\text{correctedMargin} = \min \left[\begin{array}{l} \text{caseMargin} + \text{correction} \begin{pmatrix} \text{ImportantVehicles}, \\ \text{Weather}, \\ \text{WakeTurbulence} \end{pmatrix} \\ \text{maximalAvoidanceMargin} \end{array} \right] \quad (\text{E.9})$$

E.6 Rule: Converging Maneuver

The rule is invoked based on the *angle of approach* range defined in *collision case calculation* (sec. 6.7.6). Behavior enforced to this rule is equal to rule E.5 except the *intruder* stays on his original path. UAS behavior is described in (sec 6.7.3). The *rule summary* is given by (tab. E.6).

| | | |
|--|------------------|---------------------------|
| <p><i>Invocation:</i> When <i>UAS avoidance role</i> is <i>Converging</i>, and <i>avoidance grid</i> is empty.</p> <p><i>Objective:</i> Ensure that the <i>UAS body</i> does not enter into <i>intruder's well clear zone</i>.</p> <ol style="list-style-type: none"> 1. Prevent <i>left-side leading</i> maneuvers (rule E.8). 2. Prevent head on <i>safety margin</i> breach(rule E.9). 3. Return to original course, when <i>navigation grid</i> is clear. 4. Prevent <i>wake turbulence</i> encounter (by safety margin correction). | | |
| <i>Context</i> | <i>Condition</i> | <i>Application</i> |
| UAS Navigation grid, Collision point, Avoidance role | None | Run rules from objective. |

Table E.6: Converging maneuver rule definition.

E.7 Rule: Overtake

During overtake maneuver there is our *UAS* and *Intruder* cruising at same *flight level*. The *angle of approach* (α) is lesser than 70° . *UAS* absolute velocity is much greater than *overtaken* absolute velocity.

It is assumed that during *overtake* maneuver *overtaken* intruder will keep constant heading and velocity. If this assumption is broken, the *UAS* system will invoke *Emergency avoidance* procedure. *UTM* will calculate such *divergence* and *convergence* waypoints that *overtake safety condition* (E.10) is satisfied.

$$\text{distance}(\text{uasPosition}, \text{overtakenPosition}) \geq \text{utmMargin}, \forall t \in \text{manueverTime} \quad (\text{E.10})$$

Where *utmMargin* is calculated based on *Collision case* resolution. The *main idea* is to

calculate *Safe offset for Overtake maneuver*, let us have:

$$\text{velocityDifference} = \|uasVelocity - overtakenVelocity\| \quad [\text{ms}^{-1}, \text{ms}^{-1}, \text{ms}^{-1}] \quad (\text{E.11})$$

Decision distance (E.12) is given as distance when *UTM mandate* takes effectiveness, its assumed that *UAS* knows *UTM decision frame* [s]:

$$\text{decisionDistance} = \text{velocityDifference} \times \text{uasDecisionFrame} \quad [\text{m}, \text{ms}^{-1}, \text{s}] \quad (\text{E.12})$$

Overtake middle distance (E.13) is a length of the hypotenuse for triangle where *positional difference* and *utm margin* for overtake are cathetus:

$$\text{overtakeMiddle} = \sqrt{\|uasPosition - collisionPoint\|_2^2 + \text{safetyMargin}^2} \quad [\text{m}, \vec{m}, \vec{m}, \text{m}] \quad (\text{E.13})$$

Safe offset (E.14) is considered as a combination of *overtake middle distance* (E.13), *decision distance* and *uas waypoint reach margin*.

$$\begin{aligned} & \text{overtakeMiddle} \\ \text{safeOffset} = & +\text{decisionDistance} \quad [\text{m}, \text{m}, \text{m}, \text{m}] \quad (\text{E.14}) \\ & +\text{waypointReachMargin} \end{aligned}$$

Note. *Waypoint reach margin* [m] is the property of own *UAS navigation algorithm*. It represents the maximal distance of vehicle position and a waypoint at a time when the waypoint is considered reached.

Local coordinate frame: UAS and Overtaken are in Local coordinate frame heading in X^+ axis direction (X^+ front of aircraft, X^- back of vehicles, Y^- right side, Y^+ left side, $\text{flightLevel} \rightarrow Z = 0$), Collision Point is considered as $\vec{0}$,

Divergence point (E.15) in local coordinates is given as right offset of (*UTM margin*) and *decision distance*:

$$\text{divergence} = \begin{bmatrix} 0 \\ -\text{decisionDistance} - \text{utmMargin} \\ 0 \end{bmatrix} \quad [\vec{m}, \text{m}, \text{m}] \quad (\text{E.15})$$

Convergence point (E.16) in local coordinates is given frontal *safe offset* (E.14) and right offset of *UTM margin* and *decision distance*:

$$\text{convergence} = \begin{bmatrix} \text{safeOffset} \\ -\text{decisionDistance} - \text{utmMargin} \\ 0 \end{bmatrix} \quad [\vec{m}, \text{m}, \text{m}] \quad (\text{E.16})$$

Convergence (E.17) and *Divergence* (E.18) waypoint in global coordinate frame is

obtained via transformation function R_{XYZ} as follow:

$$\begin{aligned} divergenceWaypoint &= collisionPoint \\ &+ R_{XYZ}(overtakenOrientation, divergence) \end{aligned} \quad (\text{E.17})$$

$$\begin{aligned} convergenceWaypoint &= collisionPoint \\ &+ R_{XYZ}(overtakenOrientation, convergence) \end{aligned} \quad (\text{E.18})$$

Overtake rule is summarized in (tab. E.7).

| | | |
|--|--|--|
| <i>Invocation:</i> Invoked by rule <i>Collision Case Resolution</i> (rule E.3) | | |
| <i>Divergence Waypoint</i> (E.17): waypoint to diverge from original UAS path to ensure Intruder safety, with unchanged intruder velocity and heading. | | |
| <i>Convergence Waypoint</i> (E.18): waypoint when convergence to original UAS path is enabled, within unchanged intruder velocity and heading. | | |
| <i>Objective:</i> | | |
| <ol style="list-style-type: none"> 1. Calculate <i>Divergence Waypoint</i> and <i>Convergence Waypoint</i>. 2. Enforce Divergence/Convergence waypoint during avoidance. | | |

| <i>Context</i> | <i>Condition</i> | <i>Application</i> |
|--|--|--|
| UAS Navigation Grid, Collision Point, Avoidance Role | $UASVelocity$ \gg $IntruderVelocity$ | Calculate & Enforce: <ul style="list-style-type: none"> • Divergence waypoint, • Convergence waypoint |

Table E.7: Overtake rule definition.

E.8 Rule: Right Plane Heading

There is a need to check if the *trajectory* is heading to the *right-side* from *collision point*. For this purpose, one may need to define a *separation plane in the 3D environment*. *Separation plane* will be defined according to Samuelson *hyperplane separation theorem* [194].

Separation plane (E.19) is defined by three points in *global coordination frame*:

1. *UAS Position* which is fixed to given *time-frame*.
2. *Collision point* which is not equal to *uas position* by definition.
3. *Gravitational acceleration vector* fitted to *UAS position* and orthogonal to vector (*uasPosition* → *collisionPoint*).

The properties of these three points guarantees that $scale.usasPosition \neq scale.collisionPoint \neq scale.gravitationalAcceleration$ for any linear $scale \neq 0$.

$$\text{SeparationPlane} = \text{Plane} \left(\begin{array}{l} uasPosition, collisionPoint, \\ loc2glob(uasPosition, gravitationalAcceleration) \end{array} \right) \quad (\text{E.19})$$

Separation plane (E.19) in *right-hand coordinate frame* where $\text{center} = uasPosition$ X^+ is given by vector \vec{x}^+ ($uasPosition, collisionPoint$) and Z^- is given by vector \vec{z}^- ($uasPosition, gravitationalAcceleration$). Then *right subspace* can be defined as all points where $y \leq 0$ and *left subspace* as all points where $y > 0$.

Reach set contains *trajectories*, the minimal dataset for trajectory is time-series of *position* and *heading* regardless *underlying nonlinear model*. Let us have *transformation function* which can map UAS *position* and *heading* into *separation plane coordinate frame*.

The *first condition* (E.20) says that each trajectory *point* must lie within the *right space portion*.

$$\forall position \in trajectory, \quad position \in rightSubspace \quad (\text{E.20})$$

The *second condition* (E.21) needs to be applied for each *decision point* when *trajectory* can be re-planned. It must be ensured that in time of reaching *decision point* vehicle is not heading into *left subspace* with given *turning time horizon*. The *minimal information* contains a heading (velocity) vector. Checking if linear projection from *position* point with *heading* in given time-frame $[0, horizon]$ is sufficient.

$$\forall t \in [0, horizon], \quad (position + velocity * t) \in rightSubspace \quad (\text{E.21})$$

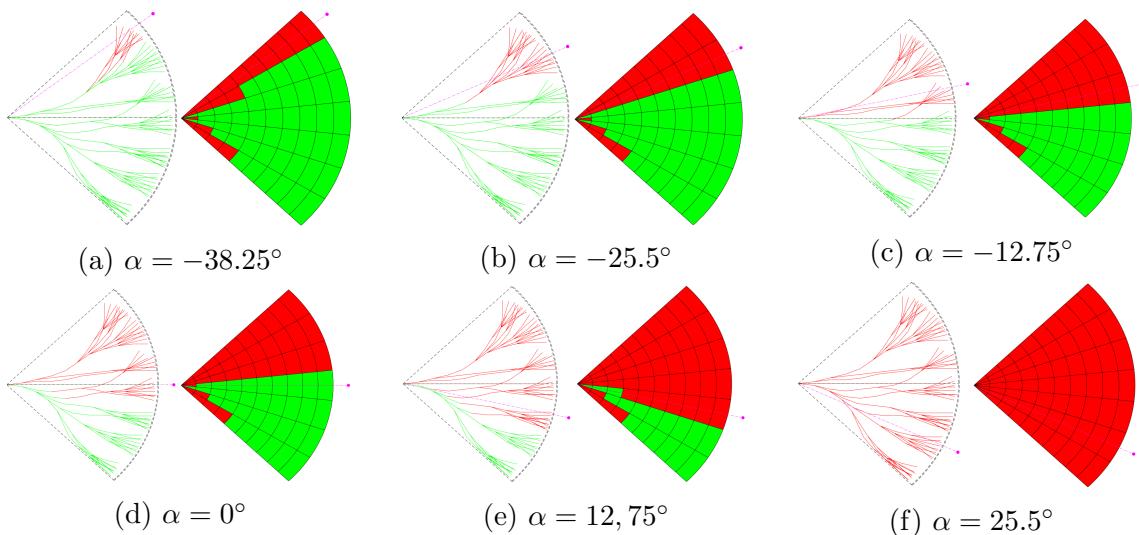


Figure E.1: Right plane heading rule evaluation for various angles of approach α .

Figure E.1. shows *enabled* (green line) and *disabled* (red lines) *trajectories* (left sub-

figure). These trajectories are divided according to the *separation line* (magenta dashed line), given by *vehicle position* and *collision point* (magenta circle). *Space segmentation* (right subfigure) show *reachable* (green fill) *unreachable* (red fill) space. The situation is shown for various *collision point angles of approach* α .

The rule for a right plane heading check is summarized in (tab. E.8).

| | | |
|--|---------------------------------------|--|
| <i>Invocation:</i> Invoked by other <i>maneuver rules</i> . | | |
| <i>Objective:</i> Disable all <i>trajectories</i> in <i>Navigation grid's reach set</i> which are: | | |
| 1. <i>Heading into collision zone</i> | 2. <i>Leading into collision zone</i> | |

| Context | Condition | Application |
|---|---|--|
| UAS Navigation Grid, Collision point (LOC) | There are feasible trajectories in Navigation Grid. | Disable trajectories in Navigation Grid. |

Table E.8: Right plane heading rule definition.

E.9 Rule: Enforce Safety Margin

Rule E.8. checks right plane heading for a single mass point along *trajectories*. The rule needs to account *body mass* of *intruder* and UAS, other factors like safe distance, regulations, etc. All mentioned factors are included in the *safety margin*. The *safety margin* is applied as *radius ball* around *collision point*.

Collision point can be mapped from *global coordinate frame* to *reach set coordinate frame*, based on UAS *position and orientation* in a *decision time*. Then a comparison of distance between *collision point* and every *trajectory decision point* is trivial.

Trajectory feasibility condition for *non-controlled airspace* (E.22) is given as follow:

$$\forall \text{position} \in \text{trajectory}, \quad \text{distance}(\text{position}, \text{collisionpoint}) \geq \text{safetyMargin} \quad (\text{E.22})$$

Controlled airspace must maintain *well clear condition*. To enforce protective barrel around *collision point* one must compare *global coordinates*. *Trajectory feasibility condition* for *controlled airspace* (E.23) is given as follow:

$$\begin{aligned} & \forall \text{position} \in \text{trajectory}, \\ & \quad XY \text{distance}(\text{position}, \text{collisionPoint}) \geq \text{safetyMargin} \quad (\text{E.23}) \\ & \quad \text{flightLevelStart} \geq Z(\text{position}) \geq \text{flightLevelEnd} \end{aligned}$$

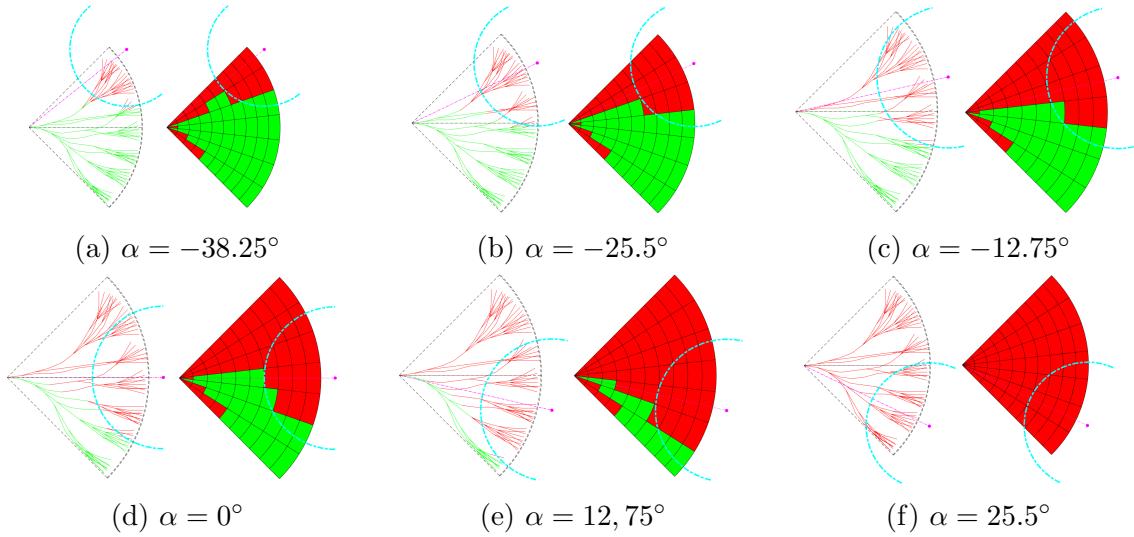


Figure E.2: Enforce safety margin rule evaluation for various angles of approach α .

Figure E.2. shows *enabled* (green line) and *disabled* (red lines) *trajectories* (left subfigure). These trajectories are divided according to the *separation line* (magenta dashed line), given by *vehicle position* and *collision point* (magenta circle). More trajectories are disabled due to *safety margin* (teal dashed line) around the *collision point*. *Space segmentation* (right subfigure) show *reachable* (green fill) *unreachable* (red fill) space. The situation is shown for various *collision point angles of approach* α .

The rule for safety margin check is summarized in (tab. E.9).

Invocation: Invoked by other *maneuver rules*.

Objective: Based on the type of airspace, for the given *collision point* and *safety margin* disable trajectories in:

1. Ball radius for *non-controlled* airspace (E.22).
2. Well-clear barrel *controlled* airspace (E.23).

| Context | Condition | Application |
|---|--|--|
| UAS Navigation Grid Collision point Safety Margin | There are feasible trajectories for condition application. | Disable trajectories in Navigation Grid. |

Table E.9: Enforce safety margin rule definition.

Appendix F

Comparison to the Previous Version of the Framework

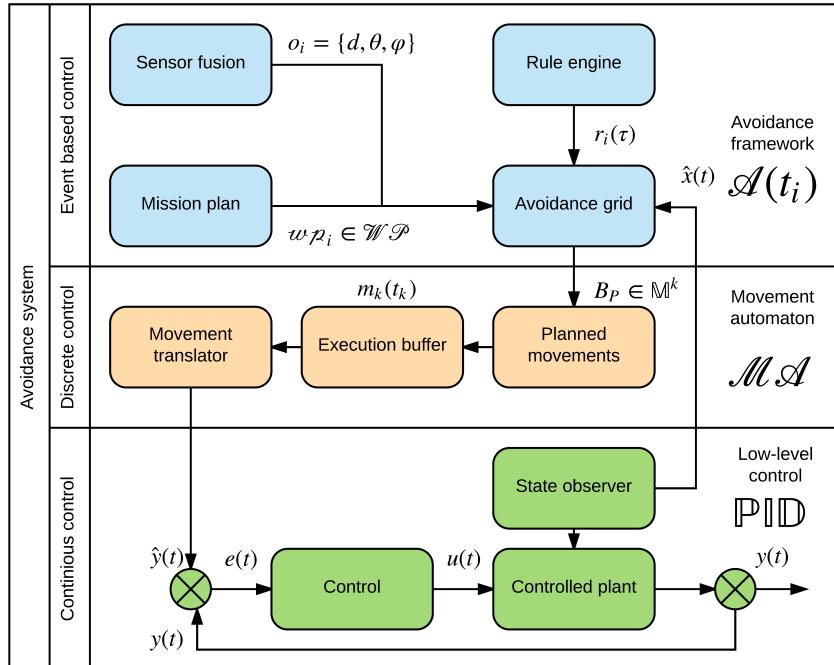


Figure F.1: Obstacle avoidance based on Reach sets concept [9].

Conceptual scheme: The overall concept of *Detect and Avoid Framework* (fig. F.1) is taking architecture from LSTS toolchain [186, 195]. The UAS part is based on *LSTS Dune*, and it can be easily integrated in the future.

1. *Continuous control* - is not solved in this work, its kept in the scheme for reference.
2. *Discrete control* - it bridges event based *Detect & Avoid* core functionality with *Continuous control*. Its covered by *Movement Automaton* (sec. 5.1).

3. *Event-based control* - covers major functionalists:

- a. *Sensor (Data) fusion* - the main feed of information, implementation of *sensor fusion* (sec. 4.1.8) and *data fusion* (sec. 4.1.9) contributing the avoidance events, introduced in (sec. 5.2).
- b. *Mission plan* - feeding actual goal and objectives to *Navigation Algorithm* (sec. 5.3) and obeying *UTM directives* (sec. 5.6).
- c. *Avoidance Grid* - using mainly *Approximation of Reachable Space* (sec. 5.4) in *Avoidance Maneuver Estimation*.
- d. *Rule engine* - enforcing UTM directives (sec. 5.6).

Surveillance Improvements in Our Work: *Hierarchical calculation* is addressed in *Mission Control run* (sec: 6.6.2) where threats are hierarchically applied based on *severity*.

Source reliability evaluation is addressed in *Static Obstacles* (sec. 6.5.1) and *Moving Obstacles* 6.5.2). The main rating for *Detected obstacle*, *Map Obstacle* and *Visibility* of space are established there.

Clear rating definition - the *Reachability* of space portion and *Safety* rating for trajectory are established in *Avoidance Grid Run* (sec. 6.6.1)

Reach Set Improvements in Our Work: *Limited system dimension* - the discretization due to the higher system dimension and increased maneuver complexity goes hand-in-hand with *pre-calculation* of the *Reach Set*. This shortcoming is addressed in (sec. 6.4.3).

Real-time optimization - replaced by *Discrete offline optimization problem*. The *general cost function* is given in (eq. 4.30). The optimization problem solved in this work is defined in (eq. 4.34).

Continuous space disparity - The *pre-calculated reach set estimation* can be valid with a small *marginal error* for some region in *system state space*. The dynamic method for state space segmentation can be used [196]. This aspect is not addressed in this work, because it strongly depends on the system behind movement automaton.

Trajectory Tracking - The *movement automaton* (def. 33) in Control Mode can be used to track a reference trajectory in form of the *Movement Buffer*(def. 31). Another option is to use *thick waypoint trajectory tracking for UAS* like in [197] or [198]. The work will use only *Movement Automaton* as controller/predictor.

Appendix G

Framework - Matlab Implementaiton

The presented framework covers a wide variety of the functionality to enable future DAA methods development. The framework enables to integrate any partial method into complex infrastructure. The user can play with reach set approximation and trajectory properties.

There is a possibility to add additional information sources, sensor reading assessments and reach set approximations. The scalability of the approach is possible thanks to movement automaton; there is only a requirement of the proper movement set implementation.

The amount of the work in this framework is wide; it has been developed over past two years, it started as a simple proof of concept¹ for optimal control report. The predictive properties of the *movement automaton* were added in need of *Predictive control report*². The cooperation with Linkoping University yielded *Data fusion procedure*³. The final increment in the form of UTM implementation is summarized in this thesis. The initial build of theframework⁴ has been released on 9th October 2018 for the public.

G.1 Functionality Description

Core Framework: The *core framework* reflects the implementation used in (ch. 6).

[Module] *Vehicle* - UAS Model and Movement Automaton implementation (sec. 6.2).

[Class] *Vehicle* - UAS model with movement automaton.

[Method] *fly(MovementType)* - executes one movement for one second.

[Method] *plot()* - displays trajectory, UAS body and planned trajectory in GCF.

[Class] *LinerizedModel* - used as movement automaton predictor.

¹Optimal Control <https://github.com/logomo/Optimal-Control>

²Predictive Control <https://github.com/logomo/Predictive-control---Final-report>

³Data fusion procedure <https://github.com/logomo/Data-Fusion-Report>

⁴Feature-based ACAS <https://github.com/logomo/Feature-based-ACAS>

[Class] *State* (Class Role) - represent UAS system state evolution as a set of state-input time series.

[Method] *plot* - shows state evolution over time in figures.

[Module] *AvoidanceGrid* - Avoidance Grid (sec. 6.3) and Reach set approximation (sec. 6.4).

[Class] *AvoidanceGrid* - the space segmentation class containing the following methods:

[Method] *putObstacle(AbstractObstacle)* intersects static/map obstacle object simulates LiDAR reading in case of *static obstacle*.

[Method] *intersectAdversary(AbstractAdversary)* - insert intruder intersection into avoidance grid cells.

[Method] *applyConstraint(AbstractConstraint)* - insert constraint hard/soft static/moving into avoidance grid cells.

[Method] *precalculate[ReachSetType](PredictorNode)* - creates specific reach set type for initial node (offline/online).

[Method] *plotGridSlice(*)* - plots specific status of the cells on horizontal/vertical layer specified by cell indexes.

[Method] *recalculate(*)* - enforces data fusion procedure on avoidance grid cells and reach set approximation trajectories.

[Class] *GridLayer* - represents one distance layer in avoidance grid, used in wave-front algorithm, contains methods for reach set estimation and rating calculations, no notable methods to describe.

[Class] *GridCell* - represents one cell in avoidance grid, contains a set of passing trajectories, low-level support methods for reach set estimation, no notable methods to describe.

[Class] *PredictorNode* - represent one piece-wise segment of trajectory between before-after unitary movement application it can be linked into tree/graph data structure:

[Method] *expand(AvoidanceGrid)* - applies full movement set to create a possible frontal expansion of the trajectories. Various expansion constraints are applied to enhance functionality.

[Method] *plotTrajectories(*)* - shows trajectories coming out from the *root* node.

[Method] *calculateCost(*)* - applies a cost function to calculate trajectory segment expenses, this is used later in avoidance path selection process.

[Module] *Obstacles* - Static/Map Obstacles (sec. 6.5.1) and All Constraints types (sec. 6.5.3).

[Class] *AbstractConstraint* - abstract class representing constraint in the 3D environment the notable methods:

[Method] *getPoints(*)* - gets 3D point cloud in GCF, representing LiDAR reading of the constraint (constraint also be used as obstacles and vice-versa).

[Method] *dynamize(*)* - enables/disables movement property of constraint, this is used for weather behavior simulation, it can comprehend any kind of linear movement.

[Method] *applyMovement(*)* - applies the movement defined by *dynamize(*)* method. This is to force the movement of constraint if it is necessary.

[Method] *isIntersection(*)* - Description - calculates the impact on *UAS* operation space (threat verification method).

[Class] *PolyConstraint* - the standard weather map is represented as 2D flight level slices, where in leveled flight the weather constraints are 2D polygons. The 3D representation uses a 2D polygon for a horizontal boundary and altitude range for a vertical boundary. The interface methods of *AbstractConstraint* are implemented.

[Class] *ExamplePolyConstraint* - the set of example constraints/obstacles, buildings weather areas prototypes, refer to *ExamplePolygonType* enumeration, extension of the *PolyConstraint* class.

[Class] *AbstractObstacle* - abstract class representing a map or detected obstacle in a 3D environment, the complete listing of implementations can be found in [156], the notable methods:

[Method] *getPoints(*)* - get obstacle points valued with *ObstacleType* enumeration. The intersection algorithm is depending on this,

[Method] *isCollision(*)* - checks if there is an inevitable collision with an obstacle body.

[Method] *isIntersection(*)* - checks if there is an intersection with UAS operation space.

[Class] *SphereObstacle* - the *AbstractObstacle* implementation representing sphere with fixed point center and fixed radius in time.

[Class] *BarellObstacle* - the *AbstractObstacle* implementation representing sphere with fixed point center and circle constraint on horizontal GCF plane and altitude limitation on Y GCF axis.

[Class] *MazeMatrix* (Class Role) - helper class to create a city-like landscapes in a mesh grid, the plan is defined by *mazemap*, the notable methods:

[Method] *generateMazeObstacles(mazeMap)* - returns the set of *AbstractObstacle* implementations, representing the landscape of the maze.

[Module] *AdversaryVehicle* - Intruder Intersection Implementation (sec. 6.5.2).

[Class] *AbstractAdversaryVehicle* (Class Role) - abstract intruder implementation defining base functions for registering and intruder and outlining intersection models representation, notable method:

[Method] *registerSelf(avoidanceGrid)* - the intruder register itself at the UAS avoidance grid.

[Class] *AdversaryVehicle* (Class Role) - the specification of *AbstractAdversaryVehicle* interface implementing space related intersection calculations, notable methods:

[Method] *findLinearIntersection(*)* - finds linear intersection with avoidance grid.

[Class] *TimedAdversaryVehicle* - the specification of *AdversaryVehicle* adding meeting time in avoidance grid cell aspect, notable methods:

[Method] *findLinearIntersection(*)* - timed linear intersection with avoidance grid search method.

[Method] *findIntersectionBalls(*)* - timed body volume intersection with avoidance grid search method.

[Method] *findIntersectionEllipseCells(*)* - timed uncertainty spread intersection implementation.

[Class] *IntersectionConfig* - intersection model configuration class containing setting class.

[Module] *MissionControl* - Avoidance/Navigation loop implementation (sec. 6.6).

[Class] *MissionControl* - the control concept implementation of the *non-cooperative* avoidance for one UAS working with Obstacle/Intruders/Constraints, optimized for LiDAR/ADS-B equipment, notable methods:

[Method] *runOnce(*)* - mission control implementation with all data processing, hierarchies and event handling.

[Method] *runOnceWithPlot(*)* - mission control run with situation plot of mission dynamic content - plane position, flew trajectory, planned trajectory.

[Method] *findBestPath(*)* - avoidance grid run implementation, to search the best path for one time, one threat setup and fixed goal waypoint, underlying functions shows the logic of the trajectory selection.

[Method] *plotMissionStaticContent(*)* - plots static mission content - waypoints, goals initial position.

[Method] *plotObstacleToDistanceStatistic(*)* - plots and calculates the *crash distance to nearest threat* statistic during the mission.

[Method] *plotRealvsPlanTrajectoryStatistics(*)* - plots and calculates *trajectory tracking* statistics.

[Method] *plotAndCalculateComputationTime(*)* - plots and calculates *computational load* statistics.

[Method] *notifyIntruder(*)* - ADS-B notification method corpus, contains *PositionNotification* object creation and *Intruder* registration procedure.

[Method] *getVehiclePositionNotification(*)* - returns actual position notification for UTM implementation.

[Class] *FlightLog* - wrapper class for notable flight parameters snapshots on the *decision time*. The flight log is created by mission control *runOnce(*)* method.

[Class] *Intruder* (Class Role) - mission control non-cooperative intruder data structure (ADS-B like message).

[Class] *Waypoint* (Class Role) - the waypoint representation in 3D GCF, including the status flags.

[Module] *UTM* - UAS Traffic Management leveled flight traffic management services (sec. 6.7) implementation in one airspace cluster.

[Class] *UTMControl* - the UTM core services implementation to cover necessary *Rules Of the Air* implementation.

[Method] *registerMission(*)* - registers the mission in active airspace segment.

[Method] *runSimulations(*)* - runs active mission controls for one *UTM decision* frame, this includes the mission dynamic content plot, the missions static content plot must be run prior to this function call.

[Method] *createCollisionCases(*)* - creates collision cases for active collisions, closes inactive cases and runs overall management of the ongoing situations.

[Method] *linkCollisionCase(*)* - when new collision case pops out, the ongoing collision cases are checked for a possible link, also issues the directives to active UAS systems.

[Method] *showCollisioncaseTrace(*)* - shows decision making trace for linked collision cases, the process is printed out in the console.

[Class] *CollisionCase* - collision case data structure wrapper.

[Class] *VehiclePositionNotification* - position notification with additional information data-wrapper.

[Module] *RuleEngine* - UTM directives implementation over *navigation loop* (sec. 6.8) (MissionControl/AvoidanceGrid/PredictorNode).

[Class] *RuleEngine* - the *rule engine* implementation with notable functions:

[Method] *activateRule(jointPointCode,ruleCode)* -for given decision point in algorithm activate rule given by *RuleCode* enumeration member, the next trigger of decision point will invoke rule behaviour.

[Method] *deactivateRule(jointPointCode,ruleCode)* - for given decision point in algorithm deactivate rule given by *RuleCode* enumeration member, the running instances of the rule will finish, the new instance will not be invoked.

[Method] *invoke(rullable,jointPoint)* - invoke rule engine on specific joint (decision) point, the rullable is reference to *RullableObject* interface implementation.

[Method] *invokeRule(context,jointPointCode,ruleCode)* - forced rule invocation, the standard conditions are checked, the context can be modified. This method is used in rule chaining.

[Class] *AbstractRule* - the interface class of rule implementation, notable functions:

[Method] *parseContext(*)* - the context of *RullableObject* is passed as a map, the internal structures of the rule needs to initialized.

[Method] *testCondition(*)* - the parsed context is checked to triggering conditions, if the conditions are met, the rule continues with *invokeRuleBody()*.

[Method] *invokeRuleBody(*)* - the business logic of the rule, context changes, calculations, etc.

[Class] *TestRule* - extends *AbstractRule*, the template for new rule implementation.

[Class] *RulePriorRulesOfAir* - extends *AbstractRule*, gathers and verifies directives in the form of active collision cases, prepares the list of *active collision cases*.

[Class] *RuleCollisionCaseResolution* - extends *AbstractRule*, process the list of active collision cases, based on *avoidance role* invokes specific situation resolution commands.

[Class] *RuleConvergingManeuver* - extends *AbstractRule*, enforces converging role on UAS, to avoid collision point.

[Class] *RuleHeadOnApproachManeuver* - extends *AbstractRule*,enforces head on avoidance, to avoid collision point.

[Class] *RuleOvertakeManeuver* - extends *AbstractRule*, forces overtaking UAS to follow divergence/convergence points.

[Class] *RulePostRulesOfAir* - extends *AbstractRule*, notifies the resolutions of active collision cases to UTM.

Miscellaneous: The additional modules of the framework supporting the basic functions, the modules are ordered by importance:

[Module] *Scenarios* - Implementation of all presented scenarios (tab. 7.1) over UTM/Mission Control artifacts.

[Module] *Utilities* - common functionality package, logging, file exports, coordinate frame transformations, etc.

[Module] *Enumerations* - Aircraft categorizations, Airspace categorizations, Reach set types, Collision Case Resolution related, and Intruder related enumerations.

[Module] *Tests* - development test, functionality showcases, and scenarios prototypes (not just documented ones).

[Module] *Dijkstra* - graph representation of reach set implementation.

Appendix H

Approach Guidelines

This appendix contains guidelines topics useful for framework setup.

H.1 Guideline - Grid Size Calculation

Note. This is done for specific type of the system and it is recommended to start with full boundary defined by sensor and then reduce the Avoidance grid according performance capabilities (reach set/computational)

The grid size calculation is done by hand. The following approach has been used in our work.

For *Sensor Field* there is *effective sensor boundary* given as set:

$$\text{Boundary}(\text{Sensor} \in \text{SensorField}) = \{\text{points} \in \text{polarCoordinates}\} \quad (\text{H.1})$$

The *Boundary* for sensor fields is then given as *union of all singe sensor boundaries*:

$$\text{Boundary}(\text{SensorField}) = \bigcap_{\forall \text{Sensors}} \text{Boundary}(\text{Sensor} \in \text{SensorField}) \quad (\text{H.2})$$

Depending on boundary properties it can be projected into maximal avoidance grid boundary values:

$$\begin{aligned} & \max(\text{distanceRange}) \\ \text{Boundary}(\text{SensorField}) \rightarrow \text{AvoidanceGrid} : & \max(\text{horizontalRange}) \quad (\text{H.3}) \\ & \max(\text{verticalRange}) \end{aligned}$$

Our approach taken worst LiDAR performance into account [27] and following parameters for avoidance grid were calculated:

1. distance range $[0m, 10m]$,
2. horizontal range $[-180^\circ, 180^\circ]$,

3. vertical range $[-30^\circ, 30^\circ]$.

The *count of layers* is derived from *average distance traveled by one movement application*:

$$\text{layerCount} = \frac{|\text{distanceRange}|}{\text{avg. } \text{length}(\text{movement} \in \text{MovementSet})} \quad (\text{H.4})$$

The *layer length* is based on *our movement set* (tab. 6.1, 6.2) the average movement length is 1 m; therefore the *layer count* is 10.

The *efficient boundary* is given by *Reach Set*. Estimate reach set coverage space using *ellipsoidal toolbox* [199] up to given *sensor field* maximal distance:

$$\text{Boundary}(\text{ReachSet}) = \text{Ellipsoid}(\text{UASSystem}, \text{distance}) \quad (\text{H.5})$$

The values for *Reach Set Boundary* with distance 10 m was following:

1. distance range $[0m, 10m]$,
2. horizontal range $[-45^\circ, 45^\circ]$,
3. vertical range $[-45^\circ, 45^\circ]$,

The *Avoidance Grid* boundary is given as *intersection* of all boundaries:

$$\text{Boundary}(\text{AvoidanceGrid}) = \text{Boundary}(\text{ReachSet}) \cap \text{Boundary}(\text{SensorField}) \quad (\text{H.6})$$

The values for *Avoidance Grid Boundary* for our UAS system (sec. 6.2.2) following:

1. distance range $[0m, 10m]$,
2. horizontal range $[-45^\circ, 45^\circ]$,
3. vertical range $[-45^\circ, 45^\circ]$,
4. layer count 10, layer distance 1m.

The *horizontal cell count* and *vertical cell count* was estimated by the *rule of thumb* to have value 7 and 5.

H.2 Guideline - Safety Margin Calculation

Safety Margin Determination: To determine *safety Margin* the *Rule of Thumb* is used:

$$\text{maximalBodyRadius} \leq \text{safetyMargin} \leq 2 \times \text{turningRadius} \quad (\text{H.7})$$

The *lower boundary* is given by *UAS* construction because the *UAS* body is considered as a *unit ball* with the radius given as *maximal body radius*.

The *upper boundary* is optional, The *double of turning radius* is used by the *conservative approach* [184].

Safety Margin Bloating: The *discretization* of *Reach Set*, *Operation Space* and *Decisions* imposes standard *mixed integer* problem considering *safety*. This section covers a *non-exhaustive* list of possible *Safety Margin Bloats* in our approach.

Own Position Uncertainty Bloat: The *sensor fusion* is precise, but not *exact* in own *UAS* position determination. The usual maximal disparity needs to be accounted into *Safety Margin*.

Intruder Position Uncertainty Bloat: The *sensor fusion* of *Intruder* is precise, but not *exact* in own *UAS* position determination. The usual maximal disparity needs to be accounted into *Safety Margin*.

Weather bloat: The *Weather* impact type may result in increased *safety margin*. Example: *UAS* is not humidity resistant, the clouds will be avoided from a greater distance.

Airspace bloat: The *Airspace* depending on cluster or *country* may require greater separation distances, depending on circumstances. The example can be *UAS* directive to keep minimal separation from obstacles. The *Safety Margin* is usually overridden by *UTM* directive value.

UTM Synchronization Bloat: Both *UAS* decision times were *synchronized*. The *intruder* can be offset for the *full decision frame*. This is not an assumption, but it shows critical performance. Usually, safety margin is bloated for (worst case offset):

$$\text{safetyMarginBloat} = \begin{pmatrix} \text{intruderVelocity} \times \dots \\ \text{intruderDecisionFrame} \end{pmatrix} [m, ms^{-1}, s] \quad (\text{H.8})$$

Bibliography

- [1] Michael Huerta. Integration of civil unmanned aircraft systems (uas) in the national airspace system (nas) roadmap. *Federal Aviation Administration, Retrieved Dec, 19:2013*, 2013.
- [2] Karthik Balakrishnan, Joe Polastre, Jessie Mooberry, Richard Golding, and Peter Sachs. The roadmap for the safe integration of autonomous aircraft. Blueprint for the sky - Airbus, www.utmbblueprint.com, sep 2018.
- [3] ICAO. 4444: Procedures for air navigation services. Technical report, ICAO, 2018.
- [4] ICAO. Annex 2 (rules of the air). Technical report, ICAO, 2018.
- [5] ICAO. Annex 11 (air traffic services). Technical report, ICAO, 2018.
- [6] Remotely piloted aircraft systems atm concept of operations. obtained from: <https://www.eurocontrol.int/publications/remotely-piloted-aircraft-systems-rpas-atm-concept-operations-conops> on 14th November 2018, February 2017.
- [7] JARUS regulations. <http://jarus-rpas.org/regulations>. Accessed: 2018-10-28.
- [8] Yazdi I Jenie, Erik-Jan Van Kampen, Coen C de Visser, and Q Ping Chu. Velocity obstacle method for non-cooperative autonomous collision avoidance system for uavs. In *AIAA Guidance, Navigation, and Control Conference*, page 1472, 2014.
- [9] Alojz Gomola, João Borges de Sousa, Fernando Lobo Pereira, and Pavel Klang. Obstacle avoidance framework based on reach sets. In *Iberian Robotics conference*, pages 768–779. Springer, 2017.
- [10] Juraj Števek, Michal Kvasnica, Miroslav Fikar, and Alojz Gomola. A parametric programming approach to automated integrated circuit design. *IEEE Transactions on Control Systems Technology*, 26(4):1180–1191, 2018.
- [11] Kristian Klausen, Jostein Borgen Moe, Jonathan Cornel van den Hoorn, Alojz Gomola, Thor I Fossen, and Tor Arne Johansen. Coordinated control concept for recovery of a fixed-wing uav on a ship using a net carried by multirotor uavs. In

- Unmanned Aircraft Systems (ICUAS), 2016 International Conference on*, pages 964–973. IEEE, 2016.
- [12] Alojz Gomola. Aspect-oriented solution for mutual exclusion in embedded systems. In Alena Kozakova, editor, *ELITECH15: 17th Conference of doctoral students*, pages 964–973, Bratislava, Slovak Republic, may 2015. Online publication.
 - [13] Alojz Gomola, Pavel Klang, and Jan Ludvik. Probabilistic approach in data fusion for obstacle avoidance framework based on reach sets. In *Internal publication collection*, pages 1–93. Honeywell, 2017.
 - [14] Alojz Gomola. Model predictive control of unmanned air vehicles with obstacle avoidance capabilities. Technical report, FEUP, 2017.
 - [15] Alojz Gomola. Optimal control of unmanned air vehicles with obstacle avoidance capabilities in partially known environment. Technical report, FEUP, 2017.
 - [16] Alessandro Gardi, Roberto Sabatini, Subramanian Ramasamy, Matthew Marino, et al. Automated atm system for 4-dimensional trajectory based operations. In *AIAC16: 16th Australian International Aerospace Congress*, page 190. Engineers Australia, 2015.
 - [17] Artur Zolich, David Palma, Kimmo Kansanen, Kay Fjørtoft, João Sousa, Karl H. Johansson, Yuming Jiang, Hefeng Dong, and Tor Johansen. Survey on communication and networks for autonomous marine systems. *Journal of Intelligent & Robotic Systems*, page tba, 04 2018.
 - [18] Francesco Nex and Fabio Remondino. Uav for 3d mapping applications: a review. *Applied geomatics*, 6(1):1–15, 2014.
 - [19] William Kress Bodin, Jesse Redman, and Derral Charles Thorson. Navigating a uav with obstacle avoidance algorithms, June 5 2007. US Patent 7,228,232.
 - [20] Jonathan P How, BRETT BEHIHKE, Adrian Frank, Daniel Dale, and John Vian. Real-time indoor autonomous vehicle test environment. *IEEE control systems*, 28(2):51–64, 2008.
 - [21] Anouck R Girard, Adam S Howell, and J Karl Hedrick. Border patrol and surveillance missions using multiple unmanned air vehicles. In *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, volume 1, pages 620–625. IEEE, 2004.
 - [22] Fabio AA Andrade, Rune Storvold, and Tor Arne Johansen. Autonomous uav surveillance of a ship’s path with mpc for maritime situational awareness. In *Unmanned Aircraft Systems (ICUAS), 2017 International Conference on*, pages 633–639. IEEE, 2017.

- [23] Kristian Klausen, Thor I Fossen, and Tor Arne Johansen. Nonlinear control with swing damping of a multirotor uav with suspended load. *Journal of Intelligent & Robotic Systems*, 88(2-4):379–394, 2017.
- [24] Thomas W Kennedy Jr and Donald F Fenstermaker. Resolution advisory display instrument for tcas guidance, January 17 1995. US Patent 5,382,954.
- [25] Mike Marston and Gabe Baca. Acas-xu initial self-separation flight tests, 2015.
- [26] Jon A Blaskovich and Stephen G McCauley. Declutter of graphical tcas targets to improve situational awareness, December 11 2007. US Patent 7,307,578.
- [27] Roberto Sabatini, Alessandro Gardi, and Mark A Richardson. Lidar obstacle warning and avoidance system for unmanned aircraft. *International Journal of Mechanical, Aerospace, Industrial and Mechatronics Engineering*, 8(4):702–713, 2014.
- [28] Subramanian Ramasamy, Roberto Sabatini, Alessandro Gardi, and Jing Liu. Lidar obstacle warning and avoidance system for unmanned aerial vehicle sense-and-avoid. *Aerospace Science and Technology*, 55:344–358, 2016.
- [29] Subramanian Ramasamy, Roberto Sabatini, and Alessandro Gardi. Avionics sensor fusion for small size unmanned aircraft sense-and-avoid. In *Metrology for Aerospace (MetroAeroSpace), 2014 IEEE*, pages 271–276. IEEE, 2014.
- [30] John D Lee, Daniel V McGehee, Timothy L Brown, and Michelle L Reyes. Collision warning timing, driver distraction, and driver response to imminent rear-end collisions in a high-fidelity driving simulator. *Human factors*, 44(2):314–334, 2002.
- [31] Ronald Miller and Qingfeng Huang. An adaptive peer-to-peer collision warning system. In *Vehicular technology conference, 2002. VTC Spring 2002. IEEE 55th*, volume 1, pages 317–321. IEEE, 2002.
- [32] Lisa C Thomas, Christopher D Wickens, and IL Savoy. Effects of cdti display dimensionality and conflict geometry on conflict resolution performance. In *Proceedings of the 13th International Symposium on Aviation Psychology*. Citeseer, 2005.
- [33] David H Williams. Self-separation in terminal areas using cdti. In *Proceedings of the Human Factors Society Annual Meeting*, volume 27, pages 772–776. Sage Publications Sage CA: Los Angeles, CA, 1983.
- [34] Civil Aviation Safety Authority. Human injury model for small unmanned aircraft impacts, 2013.
- [35] Roland Weibel and R John Hansman. Safety considerations for operation of different classes of uavs in the nas. In *AIAA 4th Aviation Technology, Integration and Operations (ATIO) Forum*, page 6244, 2004.

- [36] ICAO Doc. 8168 ops/611 aircraft operations: Procedures for air navigation services-volume ii construction of visual and instrument flight procedures, 2006.
- [37] Commission implementing regulation (eu) no 923/2012. obtained from: <https://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2012:281:0001:0066:EN:PDF> on 15th November 2018, February 2017.
- [38] Kimon P Valavanis and George J Vachtsevanos. Uav sense, detect and avoid: Introduction. In *Handbook of Unmanned Aerial Vehicles*, pages 1813–1816. Springer, 2015.
- [39] United States. Federal Aviation Administration. *Pilots' Role in Collision Avoidance*. Number 48 in 1. US Department of Transportation, Federal Aviation Administration, 1983.
- [40] Alessandro Gardi, Roberto Sabatini, Subramanian Ramasamy, and Trevor Kistan. Real-time trajectory optimisation models for next generation air traffic management systems. In *Applied Mechanics and Materials*, volume 629, pages 327–332. Trans Tech Publ, 2014.
- [41] Ingrid Gerdes, Annette Temme, and Michael Schultz. Dynamic airspace sectorization using controller task load. *Sixth SESAR Innovation Days*, 2016.
- [42] EUROCONTROL NETALERT. N17-acas x-the future of airborne collision avoidance, 2013.
- [43] ICAO Annex. 10-vol. 4 - aeronautical telecommunications, surveillance radar and collision avoidance systems, 2007.
- [44] Federal Aviation Administration. Introduction to tcas ii, version 7.1. 2011.
- [45] César Munoz, Anthony Narkawicz, and James Chamberlain. A tcas-ii resolution advisory detection algorithm. In *AIAA Guidance, Navigation, and Control (GNC) Conference*, page 4622, 2013.
- [46] EASA. Prototype commission regulation on unmanned aircraft operations. obtained from: <https://www.easa.europa.eu/sites/default/files/dfu/UAS%20Prototype%20Regulation%20final.pdf> on 14th November 2018, August 2016.
- [47] Andrew Hately. Concept of Operations for U-space. Eurocontrol draft., June 2018.
- [48] European Commision. Regulation (ec) no 923/2012 of the european parliament and of the council of 26 september 2012 laying down common common rules of the air... including amendment by regulation (ec) no 1185/2016 of 20 july 2016. obtained from: <http://eur-lex.europa.eu/legalcontent/EN/TXT/?uri=CELEX:02012R0923-20171012> on 11th November 2018, July 2012.

- [49] European Commision. Regulation (ec) no 216/2008 of the european parliament and of the council of 20 february 2008 on common rules in the field of civil aviation and establishing a european aviation safety agency. obtained from: <http://eur-lex.europa.eu/legal-content/en/ALL/?uri=CELEX%3A32008R0216> on 11th November 2018, February 2008.
- [50] EASA. Easa opinion 01-2018. obtained from: <https://www.easa.europa.eu/document-library/opinions/opinion-012018> on 11th November 2018, January 2018.
- [51] Andrew C Cone, David P Thipphavong, Seung Man Lee, and Confesor Santiago. Uas well clear recovery against non-cooperative intruders using vertical maneuvers. In *17th AIAA Aviation Technology, Integration, and Operations Conference*, page 4382, 2017.
- [52] Seung Man Lee, Chunki Park, Andrew Clayton Cone, David P Thipphavong, and Confesor Santiago. Nas-wide fast-time simulation study for evaluating performance of uas detect-and-avoid alerting and guidance systems. 2016.
- [53] David Thipphavong, Andrew Cone, and Seungman Lee. Ensuring interoperability between unmanned aircraft detect-and-avoid and manned aircraft collision avoidance. 2017.
- [54] Andrew Cook. *European air traffic management: principles, practice, and research*. Ashgate Publishing, Ltd., 2007.
- [55] Emo Welzl. Smallest enclosing disks (balls and ellipsoids). In *New results and new trends in computer science*, pages 359–370. Springer, 1991.
- [56] Mark J Koetse and Piet Rietveld. The impact of climate change and weather on transport: An overview of empirical findings. *Transportation Research Part D: Transport and Environment*, 14(3):205–221, 2009.
- [57] Hiroshi Yamashita, Volker Grewe, Patrick Jöckel, Florian Linke, M Schaefer, and D Sasaki. Climate impact assessment of routing strategies: Interactive air traffic in a climate model. *Climate Proceedings*, 2015.
- [58] Travis M Smith, Valliappa Lakshmanan, Gregory J Stumpf, Kiel L Ortega, Kurt Hndl, Karen Cooper, Kristin M Calhoun, Darrel M Kingfield, Kevin L Manross, Robert Toomey, et al. Multi-radar multi-sensor (mrms) severe weather and aviation products: Initial operating capabilities. *Bulletin of the American Meteorological Society*, 97(9):1617–1630, 2016.
- [59] Edward Balaban, Indranil Roychoudhury, Lilly Spirkovska, Shankar Sankararaman, Chetan S Kulkarni, and Matthew Daigle. Dynamic routing of aircraft in presence

- of adverse weather using a pomdp framework. In *17th AIAA Aviation Technology, Integration, and Operations Conference*, page 3429, 2017.
- [60] Gregory Thompson, Marcia K Politovich, and Roy M Rasmussen. A numerical weather model's ability to predict characteristics of aircraft icing environments. *Weather and Forecasting*, 32(1):207–221, 2017.
 - [61] Glenn K Rutledge, Jordan Alpert, and Wesley Ebisuzaki. Nomads: A climate and weather model archive at the national oceanic and atmospheric administration. *Bulletin of the American Meteorological Society*, 87(3):327–341, 2006.
 - [62] Thomas P Spriesterbach, Kelly A Bruns, Lauren I Baron, and Jason E Sohlke. Unmanned aircraft system airspace integration in the national airspace using a ground-based sense and avoid system. *Johns Hopkins APL Technical Digest*, 32(3):572–583, 2013.
 - [63] Adrian Muraru. A critical analysis of sense and avoid technologies for modern uavs. In *Mechanical, Industrial, and Manufacturing Engineering—Proceedings of 2011 International Conference on Mechanical, Industrial, and Manufacturing Engineering (MIME 2011)*, 2011.
 - [64] John Lai, Jason J Ford, Luis Mejias, Peter O'Shea, and Rod Walker. See and avoid using onboard computer vision. *Sense and Avoid in UAS Research and Applications*, Plamen Angelov (ed.), John Wiley and Sons, West Sussex, UK, 2012.
 - [65] Mykel J Kochenderfer, Leo P Espindle, J Daniel Griffith, and James K Kuchar. Encounter modeling for sense and avoid development. In *2008 Integrated Communications, Navigation and Surveillance Conference*, pages 1–10. IEEE, 2008.
 - [66] Edward A Lee. *Structure and interpretation of signals and systems*. Lee & Seshia, 2011.
 - [67] John Charles Butcher. *The numerical analysis of ordinary differential equations: Runge-Kutta and general linear methods*. Wiley-Interscience, 1987.
 - [68] Lawrence F Shampine and Mark W Reichelt. The matlab ode suite. *SIAM journal on scientific computing*, 18(1):1–22, 1997.
 - [69] Nikolai Nikolaevich Krasovskij, Andrei Izmailovich Subbotin, and Samuel Kotz. *Game-theoretical control problems*. Springer-Verlag New York, Inc., 1987.
 - [70] NN Krasovskii and AI Subbotin. Game-theoretical control problems. translated from the russian by samuel kotz, 1988.
 - [71] Conrado Daws, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. The tool kronos. In *Hybrid Systems III*, pages 208–219. Springer, 1996.

- [72] Thomas A Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Information and computation*, 111(2):193–244, 1994.
- [73] Thomas A Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. Hytech: the next generation. In *Real-Time Systems Symposium, 1995. Proceedings., 16th IEEE*, pages 56–65. IEEE, 1995.
- [74] G Leitmann. Optimality and reachability via feedback controls. *Dynamic systems and mycrophysics*, pages 119–141, 1982.
- [75] LS Pontryagin, VG Boltyanskii, and RV Gamkrelidze. Ef mischenko the mathematical theory of optimal processes (english translation by k. n. trirogoff). *Interscience, New York*, 1962.
- [76] Igor Vladimirovich Girsanov. *Lectures on mathematical theory of extremum problems*, volume 67. Springer Science & Business Media, 2012.
- [77] Martino Bardi and Italo Capuzzo-Dolcetta. *Optimal control and viscosity solutions of Hamilton-Jacobi-Bellman equations*. Springer Science & Business Media, 2008.
- [78] Mircea Lazar. Model predictive control of hybrid systems: Stability and robustness, 2006.
- [79] Francesco Borrelli, Alberto Bemporad, Michael Fodor, and Davor Hrovat. An mpc/hybrid system approach to traction control. *IEEE Transactions on Control Systems Technology*, 14(3):541–552, 2006.
- [80] Pedro Casau, David Cabecinhas, and Carlos Silvestre. Autonomous transition flight for a vertical take-off and landing aircraft. pages 3974–3979, 12 2011.
- [81] S Martin, J Bange, and F Beyrich. Meteorological profiling of the lower troposphere using the research uav “m 2 av carolo”. *Atmospheric Measurement Techniques*, 4(4):705–716, 2011.
- [82] Qi Chen. Airborne lidar data processing and information extraction. *Photogrammetric engineering and remote sensing*, 73(2):109, 2007.
- [83] Theodore C Yapo, Charles V Stewart, and Richard J Radke. A probabilistic representation of lidar range data for efficient 3d object detection. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW’08. IEEE Computer Society Conference on*, pages 1–8. IEEE, 2008.
- [84] Alexander B. Kurzhanski and Pravin Varaiya. Dynamic optimization for reachability problems. *Journal of Optimization Theory and Applications*, 108(2):227–251, 2001.

- [85] Pravin Varaiya. Reach set computation using optimal control. In *Verification of Digital and Hybrid Systems*, pages 323–331. Springer, 2000.
- [86] Emilio Frazzoli, Munther A Dahleh, and Eric Feron. Trajectory tracking control design for autonomous helicopters using a backstepping algorithm. In *American Control Conference, 2000. Proceedings of the 2000*, volume 6, pages 4102–4107. IEEE, 2000.
- [87] Emilio Frazzoli. *Robust hybrid control for autonomous vehicle motion planning*. PhD thesis, Massachusetts Institute of Technology, 2001.
- [88] George J Pappas, Gerardo Lafferriere, and Shankar Sastry. Hierarchically consistent control systems. *IEEE transactions on automatic control*, 45(6):1144–1160, 2000.
- [89] Lester E Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of mathematics*, 79(3):497–516, 1957.
- [90] James Reeds and Lawrence Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific journal of mathematics*, 145(2):367–393, 1990.
- [91] Philippe Souères and J-D Boissonnat. Optimal trajectories for nonholonomic mobile robots. In *Robot motion planning and control*, pages 93–170. Springer, 1998.
- [92] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.
- [93] Héctor J Sussmann. Lie brackets, real analyticity and geometric control. *Differential geometric control theory*, 27:1–116, 1983.
- [94] Fredrik Gustafsson. *Statistical sensor fusion*. Studentlitteratur,, 2010.
- [95] Subramanian Ramasamy, Roberto Sabatini, Alessandro Gardi, and Trevor Kistan. Next generation flight management system for real-time trajectory based operations. *Applied Mechanics and Materials*, 629:344–349, 2014.
- [96] Yuriy Chynchenko, Tatyana Shmelova, and Oksana Chynchenko. Remotely piloted aircraft systems operations under uncertainty conditions. *Proceedings of the National Aviation University*, 1(1):18–22, 2016.
- [97] T Shmelova, D Bondarev, and Y Znakovska. Modeling of the decision making by uav’s operator in emergency situations. In *Methods and Systems of Navigation and Motion Control (MSNMC), 2016 4th International Conference on*, volume 1, pages 31–34. IEEE, 2016.

- [98] Volodymyr Kharchenko, Tatyana Shmelova, Yevgeniya Znakovska, Dmitriy Bondarev, and Andriy Stratyi. Modelling of decision making of unmanned aerial vehicle's operator in emergency situations. *Proceedings of the National aviation university*, 1(1):20–28, 2017.
- [99] Kwang-Yeon Kim, Jung-Woo Park, and Min-Jea Tahk. Uav collision avoidance using probabilistic method in 3-d. In *Control, Automation and Systems, 2007. ICCAS'07. International Conference on*, pages 826–829. IEEE, 2007.
- [100] Rene Vidal, Omid Shakernia, H Jin Kim, David Hyunchul Shim, and Shankar Sastry. Probabilistic pursuit-evasion games: theory, implementation, and experimental evaluation. *IEEE transactions on robotics and automation*, 18(5):662–669, 2002.
- [101] Barbara Pfeiffer, Rajan Batta, Kathrin Klamroth, and Rakesh Nagi. Path planning for uavs in the presence of threat zones using probabilistic modeling. *IEEE Trans. Autom. Control*, 43:278–283, 2005.
- [102] Mangal Kothari and Ian Postlethwaite. A probabilistically robust path planning algorithm for uavs using rapidly-exploring random trees. *Journal of Intelligent & Robotic Systems*, pages 1–23, 2013.
- [103] Lars Blackmore, Hui Li, and Brian Williams. A probabilistic approach to optimal robust path planning with obstacles. In *American Control Conference, 2006*, pages 7–pp. IEEE, 2006.
- [104] Steven M LaValle, Michael S Branicky, and Stephen R Lindemann. On the relationship between classical grid search and probabilistic roadmaps. *The International Journal of Robotics Research*, 23(7-8):673–692, 2004.
- [105] Ira M Gessel. A probabilistic method for lattice path enumeration. *Journal of statistical planning and inference*, 14(1):49–58, 1986.
- [106] Floriana Esposito, Donato Malerba, Giovanni Semeraro, and J Kay. A comparative analysis of methods for pruning decision trees. *IEEE transactions on pattern analysis and machine intelligence*, 19(5):476–491, 1997.
- [107] Kaoru Hirota. Concepts of probabilistic sets. *Fuzzy sets and systems*, 5(1):31–46, 1981.
- [108] Jacco M Hoekstra, Ronald NHW van Gent, and Rob CJ Ruigrok. Designing for safety: the ‘free flight’air traffic management concept. *Reliability Engineering & System Safety*, 75(2):215–232, 2002.
- [109] Alessandro Gardi, Roberto Sabatini, and Trevor Kistan. Multi-objective 4d trajectory optimization for integrated avionics and air traffic management systems. *IEEE Transactions on Aerospace and Electronic Systems*, 2018.

- [110] Chad Goerzen, Zhaodan Kong, and Bernard Mettler. A survey of motion planning algorithms from the perspective of autonomous uav guidance. *Journal of Intelligent and Robotic Systems*, 57(1-4):65–100, 2010.
- [111] Yixiang Lim, Alessandro Gardi, and Roberto Sabatini. Energy efficient 4d trajectories for terminal descent operations. In *International Symposium on Sustainable Aviation*, 2018.
- [112] Swati Mishra and Pankaj Bande. Maze solving algorithms for micro mouse. In *Signal Image Technology and Internet Based Systems, 2008. SITIS’08. IEEE International Conference on*, pages 86–93. IEEE, 2008.
- [113] Ibrahim Elshamarka and Abu Bakar Sayuti Saman. Design and implementation of a robot for maze-solving using flood-fill algorithm. *International Journal of Computer Applications*, 56(5), 2012.
- [114] Quentin Chatelais, Horatiu Vultur, and Emmanouil Kanellis. Maze solving by an autonomous robot. *Aalborg University*, 2014.
- [115] Nathan Richards, Manu Sharma, and David Ward. A hybrid a-star automaton approach to on-line path planning with obstacle avoidance. In *AIAA 1st Intelligent Systems Technical Conference*, page 6229, 2004.
- [116] Zhuoning Dong, Zongji Chen, Rui Zhou, and Rulin Zhang. A hybrid approach of virtual force and a* search algorithm for uav path re-planning. In *Industrial Electronics and Applications (ICIEA), 2011 6th IEEE Conference on*, pages 1140–1145. IEEE, 2011.
- [117] Allison Ryan and J Karl Hedrick. A mode-switching path planner for uav-assisted search and rescue. In *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC’05. 44th IEEE Conference on*, pages 1471–1476. IEEE, 2005.
- [118] Oskar Ljungqvist, Niclas Evestedt, Marcello Cirillo, Daniel Axehill, and Olov Holmer. Lattice-based motion planning for a general 2-trailer system. In *Intelligent Vehicles Symposium (IV), 2017 IEEE*, pages 819–824. IEEE, 2017.
- [119] Oskar Ljungqvist, Daniel Axehill, and Anders Helmersson. Path following control for a reversing general 2-trailer system. In *Decision and Control (CDC), 2016 IEEE 55th Conference on*, pages 2455–2461. IEEE, 2016.
- [120] Niclas Evestedt, Oskar Ljungqvist, and Daniel Axehill. Path tracking and stabilization for a reversing general 2-trailer configuration using a cascaded control approach. In *Intelligent Vehicles Symposium (IV), 2016 IEEE*, pages 1156–1161. IEEE, 2016.

- [121] Mihail Pivtoraiko, Ross A Knepper, and Alonzo Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, 2009.
- [122] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. William “red” whittaker. *Ziv Wolkowicki, Jason Ziglar, Hong Bae, Thomas Brown, Daniel Demirrish, Bakhtiar Litkouhi, Jim Nickolaou, Varsha Sadekar, Wende Zhang, Joshua Struble, Michael Taylor, Michael Darms, Dave Ferguson, Autonomous driving in urban environments: Boss and the Urban Challenge*, *Journal of Field Robotics*, 25(8):425–466, 2008.
- [123] Marcello Cirillo. From videogames to autonomous trucks: A new algorithm for lattice-based motion planning. In *Intelligent Vehicles Symposium (IV), 2017 IEEE*, pages 148–153. IEEE, 2017.
- [124] Claire J Tomlin, John Lygeros, and S Shankar Sastry. A game theoretic approach to controller design for hybrid systems. *Proceedings of the IEEE*, 88(7):949–970, 2000.
- [125] Hong Chen, Carsten W Scherer, and F Allgower. A game theoretic approach to nonlinear robust receding horizon control of constrained systems. In *American Control Conference, 1997. Proceedings of the 1997*, volume 5, pages 3073–3077. IEEE, 1997.
- [126] Boris Houska, Hans Joachim Ferreau, and Moritz Diehl. Acado toolkit—an open-source framework for automatic control and dynamic optimization. *Optimal Control Applications and Methods*, 32(3):298–312, 2011.
- [127] Diego Merani, Alessandro Berni, John Potter, and Ricardo Martins. An underwater convergence layer for disruption tolerant networking. In *Internet Communications (BCFIC Riga), 2011 Baltic Congress on Future*, pages 103–108. IEEE, 2011.
- [128] Kanna Rajan, Frédéric Py, and Javier Barreiro. Towards deliberative control in marine robotics. In *Marine Robot Autonomy*, pages 91–175. Springer, 2013.
- [129] José Queirós Pinto, Paulo Sousa Dias, Rui Gonçalves, Gil Manuel Gonçalves, João Tasso de Figueiredo Borges Sousa, Fernando Lobo Pereira, et al. Neptus a framework to support a mission life cycle. In *Proceedings of the 7th Conference on Manoeuvring and Control of Marine Craft*, 2006.
- [130] Paulo Sousa Dias, Rui MF Gomes, and José Pinto. Mission planning and specification in the neptus framework. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 3220–3225. IEEE, 2006.

- [131] Paulo Sousa Dias, Sergio Loureiro Fraga, Rui MF Gomes, Gil M Goncalves, Fernando Lobo Pereira, Jose Pinto, and Joao Borges Sousa. Neptus-a framework to support multiple vehicle operation. In *Oceans 2005-Europe*, volume 2, pages 963–968. IEEE, 2005.
- [132] Ricardo Martins, Paulo Sousa Dias, Eduardo RB Marques, José Pinto, Joao B Sousa, and Fernando L Pereira. Imc: A communication protocol for networked vehicles and sensors. In *Oceans 2009-Europe*, pages 1–6. IEEE, 2009.
- [133] Lorenzo Marconi, Roberto Naldi, and Luca Gentili. A control framework for robust practical tracking of hybrid automata. In *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*, pages 661–666. IEEE, 2009.
- [134] Virtual arena mpc framework for advanced uav-uas simulations. <https://github.com/andreaalessandretti/VirtualArena>. Accessed: 2016-05-30.
- [135] Andrea Alessandretti. Notes on trajectory-tracking and path-following controllers for constrained underactuated vehicles using model predictive control.
- [136] Brian L Stevens, Frank L Lewis, and Eric N Johnson. *Aircraft Control and Simulation: Dynamics, Controls Design, and Autonomous Systems*. John Wiley & Sons, 2015.
- [137] Andrew Hately. Concept of operations for u-space. Exploratory research call, EUROCONTROL, sep 2018.
- [138] AAMI Standard. Recommended practices. *Operation of Aircraft, Annex*, 6, 1986.
- [139] Maria Prandini and Jianghai Hu. Application of reachability analysis for stochastic hybrid systems to aircraft conflict prediction. In *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, pages 4036–4041. IEEE, 2008.
- [140] Ondřej Šantin and Vladimir Havlena. Combined partial conjugate gradient and gradient projection solver for mpc. In *Control Applications (CCA), 2011 IEEE International Conference on*, pages 1270–1275. IEEE, 2011.
- [141] Frangois G Pin and Hubert A Vasseur. Autonomous trajectory generation for mobile robots with non-holonomic and steering angle constraints. Technical report, Oak Ridge National Lab., 1990.
- [142] Ralph G Andrzejak, G Widman, K Lehnertz, C Rieke, P David, and CE Elger. The epileptic process as nonlinear deterministic dynamics in a stochastic environment: an evaluation on mesial temporal lobe epilepsy. *Epilepsy research*, 44(2-3):129–140, 2001.

- [143] Yoshiaki Kuwata, Justin Teo, Gaston Fiore, Sertac Karaman, Emilio Frazzoli, and Jonathan P How. Real-time motion planning with applications to autonomous urban driving. *IEEE Transactions on Control Systems Technology*, 17(5):1105–1118, 2009.
- [144] Florian Homm, Nico Kaempchen, Jeff Ota, and Darius Burschka. Efficient occupancy grid computation on the gpu with lidar and radar for road boundary detection. In *Intelligent Vehicles Symposium (IV), 2010 IEEE*, pages 1006–1013. IEEE, 2010.
- [145] Sandeep Gupta, Holger Weinacker, and Barbara Koch. Comparative analysis of clustering-based approaches for 3-d single tree detection using airborne fullwave lidar data. *Remote Sensing*, 2(4):968–989, 2010.
- [146] Osmar R Zaïane and Chi-Hoon Lee. Clustering spatial data when facing physical constraints. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pages 737–740. IEEE, 2002.
- [147] Franco Blanchini. Set invariance in control. *Automatica*, 35(11):1747–1767, 1999.
- [148] John Birmingham and Peter Kent. Tree-searching and tree-pruning techniques. In *Computer chess compendium*, pages 123–128. Springer, 1988.
- [149] Chad Goerzen, Zhaodan Kong, and Bernard Mettler. A survey of motion planning algorithms from the perspective of autonomous uav guidance. In *Selected papers from the 2nd International Symposium on UAVs, Reno, Nevada, USA June 8–10, 2009*, pages 65–100. Springer, 2009.
- [150] Catherine Plaisant, Jesse Grosjean, and Benjamin B Bederson. Spacetree: Supporting exploration in large node link tree, design evolution and empirical evaluation. In *Information Visualization, 2002. INFOVIS 2002. IEEE Symposium on*, pages 57–64. IEEE, 2002.
- [151] John Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine learning*, 4(2):227–243, 1989.
- [152] Jay Shively. Uas integration in the nas: Detect and avoid. 2018.
- [153] Kyle D Julian, Jessica Lopez, Jeffrey S Brush, Michael P Owen, and Mykel J Kochenderfer. Policy compression for aircraft collision avoidance systems. In *Digital Avionics Systems Conference (DASC), 2016 IEEE/AIAA 35th*, pages 1–10. IEEE, 2016.
- [154] Patrick O’Neil, Edward Cheng, Dieter Gawlick, and Elizabeth O’Neil. The log-structured merge-tree (lsm-tree). *Acta Informatica*, 33(4):351–385, 1996.

- [155] Pascal Van Hentenryck and Laurent Michel. *Constraint-based local search*. The MIT press, 2009.
- [156] Maria Cerna. Usage of maps obtained by lidar in uav navigation. Master thesis, Institute of Automotive Mechatronics, Faculty of Electrical Engineering and Information Technology, Slovak University of Technology, Ilkovicova 3, Bratislava. Slovak Republic, jun 2018.
- [157] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760–772, 1998.
- [158] Michael Ian Shamos and Dan Hoey. Closest-point problems. In *Foundations of Computer Science, 1975., 16th Annual Symposium on*, pages 151–162. IEEE, 1975.
- [159] Jon Louis Bentley, Bruce W Weide, and Andrew C Yao. Optimal expected-time algorithms for closest point problems. *ACM Transactions on Mathematical Software (TOMS)*, 6(4):563–580, 1980.
- [160] Jon Louis Bentley and Thomas A Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on computers*, (9):643–647, 1979.
- [161] Michael Ian Shamos and Dan Hoey. Geometric intersection problems. In *17th annual symposium on foundations of computer science*, pages 208–215. IEEE, 1976.
- [162] Duncan McLaren Young Sommerville. *Analytical geometry of three dimensions*. Cambridge University Press, 2016.
- [163] Jack Ritter. An efficient bounding sphere. *Graphics gems*, 1:301–303, 1990.
- [164] Roberto Sabatini, Subramanian Ramasamy, Alessandro Gardi, and Leopoldo Rodriguez Salazar. Low-cost sensors data fusion for small size unmanned aerial vehicles navigation and guidance. *International Journal of Unmanned Systems Engineering.*, 1(3):16, 2013.
- [165] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *International Conference on Principles and Practice of Constraint Programming*, pages 417–431. Springer, 1998.
- [166] Roberto Sabatini, Celia Bartel, Anish Kaharkar, Tesheen Shaid, and Subramanian Ramasamy. Navigation and guidance system architectures for small unmanned aircraft applications. *International Journal of Mechanical, Industrial Science and Engineering*, 8(4):733–752, 2014.
- [167] Roberto Sabatini, Alessandro Gardi, and M Richardson. Lidar obstacle warning and avoidance system for unmanned aircraft. *International Journal of Mechanical, Aerospace, Industrial and Mechatronics Engineering*, 8(4):718–729, 2014.

- [168] Jon Kleinberg, Christos Papadimitriou, and Prabhakar Raghavan. A microeconomic view of data mining. *Data mining and knowledge discovery*, 2(4):311–324, 1998.
- [169] Nico Zimmer, Jens Schiefele, Keyvan Bayram, Theo Hankers, Sebastian Frank, and Thomas Feuerle. Rule-based notam & weather notification. In *Integrated Communications, Navigation and Surveillance Conference (ICNS), 2011*, pages O1–1. IEEE, 2011.
- [170] Thomas Prevot, Joseph Rios, Parimal Kopardekar, John E Robinson III, Marcus Johnson, and Jaewoo Jung. Uas traffic management (utm) concept of operations to safely enable low altitude flight operations. In *16th AIAA Aviation Technology, Integration, and Operations Conference*, page 3292, 2016.
- [171] Nico Zimmer and Keyvan Bayram. Selective weather notification, March 18 2014. US Patent 8,674,850.
- [172] Alexandre Bayen, Pascal Grieder, George Meyer, and Claire J Tomlin. Langrangian delay predictive model for sector-based air traffic flow. *Journal of guidance, control, and dynamics*, 28(5):1015–1026, 2005.
- [173] Parimal Kopardekar and Sherri Magyarits. Dynamic density: measuring and predicting sector complexity [atc]. In *Digital Avionics Systems Conference, 2002. Proceedings. The 21st*, volume 1, pages 2C4–2C4. IEEE, 2002.
- [174] MP Helme, K Lindsay, SV Massimini, and G Booth. Optimization of traffic flow to minimize delay in the national airspace system. In *Control Applications, 1992., First IEEE Conference on*, pages 435–437. IEEE, 1992.
- [175] Confesor Santiago and Eric R Mueller. Pilot evaluation of a uas detect-and-avoid system’s effectiveness in remaining well clear. In *Eleventh UAS/Europe Air Traffic Management Research and Development Seminar (ATM2015)*, 2015.
- [176] Karl Bilimoria and Hilda Lee. Analysis of aircraft clusters to measure sector-independent airspace congestion. In *AIAA 5th ATIO and 16th Lighter-Than-Air Sys Tech. and Balloon Systems Conferences*, page 7455, 2005.
- [177] CR Brinton and S Pledgie. Airspace partitioning using flight clustering and computational geometry. In *Digital Avionics Systems Conference, 2008. DASC 2008. IEEE/AIAA 27th*, pages 3–B. IEEE, 2008.
- [178] M Ebrahim Fouladvand, Zeinab Sadjadi, and M Reza Shaebani. Characteristics of vehicular traffic flow at a roundabout. *Physical Review E*, 70(4):046132, 2004.
- [179] Raffaele Mauro and Marco Cattani. Potential accident rate of turbo-roundabouts. In *4th International Symposium on Highway Geometric DesignPolytechnic University of Valencia Transportation Research Board*, 2010.

- [180] Michael R Benjamin, Joseph A Curcio, John J Leonard, and Paul M Newman. Navigation of unmanned marine vehicles in accordance with the rules of the road. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 3581–3587. IEEE, 2006.
- [181] Ernest Friedman Hill. *Jess in action: Java rule-based systems*. Manning Publications Co., 2003.
- [182] Georg S Seyboth, Dimos V Dimarogonas, and Karl H Johansson. Event-based broadcasting for multi-agent average consensus. *Automatica*, 49(1):245–252, 2013.
- [183] Martin Hrdlik. Advanced obstacle detection and navigation methods of unmanned vehicles. Master thesis, Institute of Automotive Mechatronics, Faculty of Electrical Engineering and Information Technology, Slovak University of Technology, Ilkovicova 3, Bratislava. Slovak Republic, jun 2018.
- [184] Johann Borenstein and Yoram Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3):278–288, 1991.
- [185] Yoram Koren and Johann Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 1398–1404. IEEE, 1991.
- [186] José Pinto, Paulo S Dias, Ricardo Martins, Joao Fortuna, Eduardo Marques, and Joao Sousa. The lsts toolchain for networked vehicle systems. In *OCEANS-Bergen, 2013 MTS/IEEE*, pages 1–9. IEEE, 2013.
- [187] Leonhard Euler. Formulae generales pro translatione quacunque corporum rigidorum. *Novi Acad. Sci. Petrop*, 20:189–207, 1775.
- [188] Hanspeter Schaub and John L Junkins. *Analytical mechanics of space systems*. Aiaa, 2003.
- [189] Manuel Kramer and Douglas J Dapprich. Gyro stabilized inertial reference system with gimbal lock prevention means, October 4 1977. US Patent 4,052,654.
- [190] Thor I Fossen. Mathematical models for control of aircraft and satellites. *Department of Engineering Cybernetics Norwegian University of Science and Technology*, 2011.
- [191] Subramanian Ramasamy, Roberto Sabatini, and Alessandro Gardi. Towards a unified approach to cooperative and non-cooperative rpas detect-and-avoid. In *Fourth Australasian Unmanned Systems Conference*, 2014.

- [192] Subramanian Ramasamy, Roberto Sabatini, A Gardi, and Yifang Liu. Novel flight management system for real-time 4-dimensional trajectory based operations. In *proceedings of AIAA Guidance, Navigation, and Control Conference*, 2013.
- [193] Branka Subotic, Arnab Majumdar, and Washington Y Ochieng. Recovery from equipment failures in air traffic control (atc): The findings from an international survey of controllers. *Air Traffic Control Quarterly*, 15(2):157–181, 2007.
- [194] Hans Samelson, Robert M Thrall, and Oscar Wesler. A partition theorem for euclidean n-space. *Proceedings of the American Mathematical Society*, 9(5):805–807, 1958.
- [195] José Pinto, Pedro Calado, José Braga, Paulo Dias, Ricardo Martins, Eduardo Marques, and JB Sousa. Implementation of a control architecture for networked vehicle systems. *IFAC Proceedings Volumes*, 45(5):100–105, 2012.
- [196] Yasutake Takahashi, Minoru Asada, and Koh Hosoda. Reasonable performance in less learning time by real robot based on incremental state space segmentation. In *Intelligent Robots and Systems' 96, IROS 96, Proceedings of the 1996 IEEE/RSJ International Conference on*, volume 3, pages 1518–1524. IEEE, 1996.
- [197] Isaac Kaminer, Antonio Pascoal, Eric Hallberg, and Carlos Silvestre. Trajectory tracking for autonomous vehicles: An integrated approach to guidance and control. *Journal of Guidance, Control, and Dynamics*, 21(1):29–38, 1998.
- [198] Marina H Murillo, Alejandro C Limache, Pablo S Rojas Fredini, and Leonardo L Giovanini. Generalized nonlinear optimal predictive control using iterative state-space trajectories: Applications to autonomous flight of uavs. *International Journal of Control, Automation and Systems*, 13(2):361–370, 2015.
- [199] Alex A Kurzhanskiy and Pravin Varaiya. Ellipsoidal toolbox (et). In *Decision and Control, 2006 45th IEEE Conference on*, pages 1498–1503. IEEE, 2006.