

Diese Arbeit wurde vorgelegt am
Lehrstuhl für Mathematik (MathCCES) und bei Airbus

Entwicklung eines effizienten Wettervermeidungsalgorithmus für ein unbemanntes Flugzeug

Developing an Efficient Weather Avoidance Algorithm for a Remotely Piloted Aircraft

Masterarbeit
Simulation Sciences

Februar 2018

Vorgelegt von Presented by	Rohit Lad, 359892 rohit.lad@rwth-aachen.de
Erstprüfer First examiner	Prof. Dr. Martin Frank Lehrstuhl für Mathematik (SSC) KIT
Zweitprüfer Second examiner	Prof. Dr. Manuel Torrilhon Lehrstuhl für Mathematik (MathCCES) RWTH Aachen
Externeprüfer External examiner	Dr. Sven Martin TL3. Flight Management Systems (TEAYC-TL3) Airbus

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich diese Masterarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Stellen meiner Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen sind, habe ich in jedem Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht. Dasselbe gilt sinngemäß für Tabellen und Abbildungen. Diese Arbeit hat in dieser oder einer ähnlichen Form noch nicht im Rahmen einer anderen Prüfung vorgelegen.

Aachen, im Februar 2018

Rohit Sunil Lad
359892

Abstract

The weather has a considerable impact on an aircraft. Flying through severe weather conditions can damage the structural integrity, may lead to the malfunctioning of the onboard electronic equipment and could also lead to loss of the aircraft and thus it is important to avoid severe weather conditions to ensure a safe flight. While conducting an unmanned flight, there is a need for a weather avoidance algorithm which is responsible for detecting severe weather conditions, propose waypoints for avoidance manoeuvre considering airspace and general dynamic criteria and return to its original path to stay predictable.

An aircraft flying in upper airspaces (A-C) is granted free airspace corridor and a pre-planned flight plan by the Air Traffic Control (ATC). However, when a deviation due to weather is required under link-loss, it is subjected to the following constraints:

- **Airspace constraints:** An aircraft violating its airspace could pose a threat to other aircraft by causing a loss of separation. Thus it is important for an RPA (Remotely Piloted Aircraft) enabled with weather avoidance capabilities to be able to integrate with airspace regulations while avoiding weather. An RPA, while having a communication loss with its remote pilot should remain predictable whenever it's doing a weather avoidance manoeuvre so that the ATC could clear the airspace to avoid possible separation issues
- **Dynamic constraints:** Unlike a ground-based robot, an aircraft cannot stay still and has to keep flying with a minimum speed. Furthermore, an aircraft also cannot fly with rapid twists and turns

This thesis work begins with an introduction to aviation terminologies, flight rules, weather phenomena and hazards posed to an aircraft. Later, a review of four major categories of path planning algorithms is done mentioning their pros and cons and an algorithm from each of the categories is picked. Four different algorithms are modified to accommodate the airspace rules, detect and handle weather information, address the motion of weather clouds and intent reporting to generate an optimal path for weather avoidance. The project ends with a presentation of several test-cases and results which demonstrate the functioning of the algorithms with time complexity analysis, operational test cases and a comparison of benefits and shortcomings of the algorithms.

Acknowledgement

First and foremost, I would like to thank my family who supported me in pursuing a Master's degree in Simulation Sciences which ultimately led to an opportunity of working on this project. I would like to thank my mentor and direct supervisor at Airbus, Dr. Sanchito Banerjee for guiding me throughout this project and helping me bridge gap between the theoretical and operational understanding of the topic of weather avoidance. I would like to thank respected Prof. Dr. Martin Frank to guide me with his extensive knowledge and experience in Computational Sciences and Mathematical methods. I would like to thank my team leader at Airbus, Dr. Sven Martin for giving me an opportunity to work at Airbus and contribute to this project.

Contents

List of Figures

List of Tables

1 Introduction

This project deals with developing a weather avoidance algorithm for a Remotely Piloted Aircraft. This section briefly introduces one to the terminologies and standards used in the aviation industry, different weather phenomena and the hazards caused by adverse weather. It also describes the problem statement of the thesis while clearly stating the scope and limitations of the project.

1.1 Remotely Piloted Aircraft System

The International Civil Aircraft Organization (ICAO) defines an aircraft which operates with no on board pilot as an Unmanned aircraft [1]. An unmanned aircraft piloted from a Remote Pilot Station (RPS) is called a Remotely Piloted Aircraft (RPA). An RPA is piloted from an RPS by a Remote Pilot (RP) via a *Command and Control* (C^2) link and the system of RPA, RPAS, C^2 link, navigation, surveillance equipment together is termed as a Remotely Piloted Aircraft System(s) (RPAS).

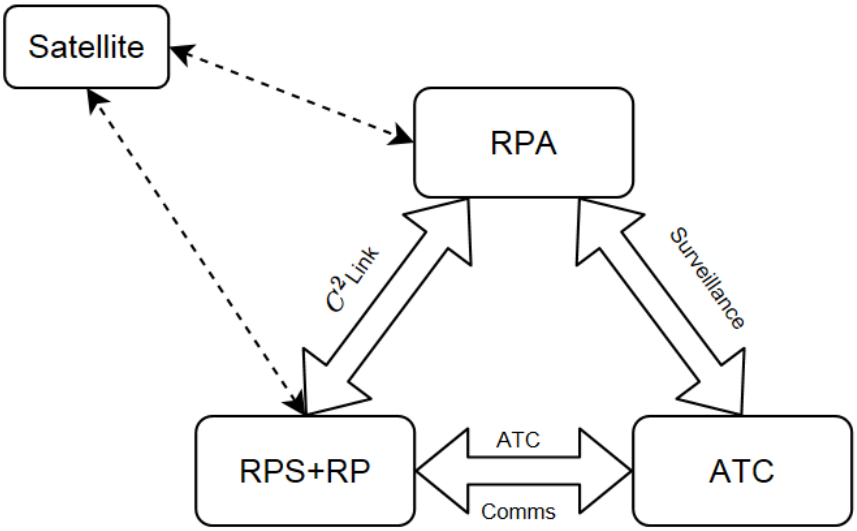


Figure 1: RPAS architecture [2]

The Manual on RPAS [1] mentions that the C^2 link communication may be in a direct Radio Line Of Sight (RLOS) in which the transmitter and receiver are directly within mutual radio link coverage or Beyond Radio Line-Of-Sight (BRLOS) in which the transmitter and receiver are not in RLOS. The RPS is in direct connection with the Air Traffic Control (ATC) which exchange instructions, status and intent reports of the RPA. The ATC possesses the information about all such manned and unmanned aircraft to manage, organize safe and collision-free flights, therefore to keep the ATC updated about it's status, the RPA is also in connection with the ATC. An RPA, before beginning the flight has to submit an intended flight path to the ATC and has to get it approved beforehand. Weather phenomenon like hail, thunderstorms, heavy

precipitations etc. may disrupt the intended flight plan of an aircraft and therefore, rules are set forth to address hazardous weather occurrences.

1.2 Instrument Flight Rules

IFR are a set of operation rules when flying under visual references is not safe or flight by referring the instruments in the flight deck and navigation is accomplished by referring to electronic signals [3, 4].

- **Requirements:** Aircraft must be equipped with navigation devices and equipment while flying with IFR. ATC clearance is required to fly in controlled areas. Pilot must adhere to the flight plan submitted and the RPA must report position whenever needed.
- **IFR level restrictions:** sets minimal flying altitudes and cruising levels while flying with IFR
- **Change to VFR:** sets rules to be followed when pilot decides to switch to VFR

While flying under IFR, the ATC assists the pilot for separation from other aircraft

1.3 Visual Flight Rules

VFR are a set of operation rules [5] for a pilot in weather conditions which the pilot is able to see. The pilot follows VFR to respond to visible weather situations to navigate, control altitude and avoid other aircraft.

- **Visual Meteorological Conditions:** or VMC are a set of conditions which describe the minimum flight visibility and minimum distance of clouds to conduct flights in VFR
- **VFR Level restrictions:** set a maximum altitude or flight level at which VFR are followed
- **Change to IFR:** sets rules to be followed while pilot decides to switch to IFR

Altitude band	**Airspace class**	**Minimum flight visibility**	**Minimum distance from clouds**
At or above 3050m AMSL	A,B,C,D,E, F,G	8 km	1500m horizontally, 300m vertically
Below 3050m AMSL and above 900m ASML or 300m above terrain, whichever is higher	A,B,C,D,E, F,G	5 km	1500m horizontally, 300m vertically
At or below 900m AMSL or 300m above terrain, whichever is higher	A,B,C,D,E	5km	1500m horizontally, 300m vertically
	F,G	5[1]km	clear of cloud and with the surface in sight

Table 1: Visual Meteorological Conditions (VMC) [6]

The table 1 gives the minimum distance from clouds and conditions of visibility to conduct a flight in VFR. If the minimum separation conditions are not met, the flight has to be conducted with IFR. Pilots flying under VFR are usually not given assistance by the ATC and are responsible for separation from other aircraft. (Above Mean Sea Level abbr. AMSL)

1.4 Airspace classes

The navigable airspaces all over the world are divided into several classes on the basis of flight rules and the communication between ATC and aircraft. Other than the above mentioned flight rules, some of the aspects addressed are

- *Separation:* The process of maintaining a minimum gap between terrain, obstacles and other aircraft
- *Clearance:* It is the approval the ATC gives to an aircraft to proceed to perform an action under the terms and conditions mentioned within the clearance
- *Traffic Information:* The information about position and intentions of aircraft which might probably pose a threat to flight

The airspace classes from with the roles of ATC, flight rules, separation [7] is provided in table 2:

[1]Flight visibilities of upto a minimum of 1500 m are permitted in some countries given that the aircraft speed gives sufficient opportunity to observe other traffic or obstacles and avoid collision

Class	IFR	VFR	Separation	Clearance	Traffic Information
A	Yes	No	For all flights	Required	No
B	Yes	Yes	For all flights	Required	No
C	Yes	Yes	For all IFR to IFR/VFR	Required	For all VFR
D	Yes	Yes	For all IFR to IFR	Required	For all IFR and VFR
E	Yes	Yes	For all IFR to IFR	Required for all IFR	For all IFR and VFR is possible
F	Yes	Yes	For all IFR to IFR as far as possible	No	On request if possible
G	Yes	Yes	No	Not provided	On request if possible

Table 2: Overview of Airspace Classes

When an RPA flying in the upper airspaces detects weather phenomenon, the RP is expected to request a clearance for it's intentions to the ATC. Once the ATC gives a clearance, the RP signals the RPA to execute the avoidance measures. If the ATC rejects the measures suggested by the RP due to reasons like separation issues, flight path violations and so on, the RPA is expected to keep flying on the cleared flight path.

1.5 Weather

A flight-path is submitted to the ATC and a study of the weather along the planned path is done. But even then, an RPA may face unexpected weather conditions and the RP has to take decisions while consulting with the ATC accordingly. The following are some of the commonly occurring weather phenomena and their effect on an aircraft and a flight [2, 8].

- **Heavy Precipitations:** Precipitation is any type (solid or liquid) of water particles falling from the atmosphere to reach the ground. Precipitation types include snow, rain, hail etc. Precipitations cause reduced visibility and could be hazardous because of their impromptu start. Hail specifically has hazardous effect on aircraft since it causes severe structural damage which may lead to damage of aircraft body
- **Icing:** is a deposit of ice forming on the aircraft body. It is a cumulative hazard i.e. the hazard worsens the longer it is allowed to happen

- **Lightening:** occurs within clouds, between clouds, from cloud to sky, cloud to ground being the most severe. Lightening may cause loss of communication due to electromagnetic interference, structural damage and damage to electronics
- **Turbulences:** occur due to chaotic atmospheric convection and clear air turbulences caused due to jet streams at high altitude. Turbulences may cause disturbances in aircraft control like chaotic accelerations
- **Thunderstorms:** is a storm produced by cumulonimbus clouds followed by thunder and lightening. Thunderstorms can bring on nearly every aviation hazard together like lightening, hail, turbulence, tornadoes, rains

The occurrences of thunderstorms, rains, snow is followed by presence of moisture or droplets in the air. These precipitations are detected by a weather radar. A RADAR meaning 'RAdio Detection And Ranging' has always played a major role in detecting precipitations. A radar emits radio waves which strike the target(s), which are heavy precipitations, cumulonimbus clouds etc [8] in our case. The stricken waves return back in the form of backscattered energy which helps in detecting weather phenomena. An RPA considered in this project is equipped with an on-board weather radar which gives it real-time weather data within a prescribed range. Moreover, the RPS also obtains weather information from ground-based sources like Digital Vol Meteo (D-VOLMET), NEXRAD [2]. With the availability of on-board weather radar and ground based weather information sources, the RP can decide whether to avoid the weather by conducting avoidance manoeuvre by consulting with the ATC.

1.6 Scope and Limitations

The RP can decide to implement a weather avoidance manoeuvre after obtaining clearance from the ATC. But in case when the RPA has a C^2 link-loss with the RP, the RPA is expected to have autonomous decision making capability in order to decide whether to avoid or fly through a weather situation. Proposing this decision making capability is the main objective of this thesis project.

1.6.1 Scope

The scope of this project is to develop a Weather Avoidance (WA) algorithm for an RPA. The objective is not to develop an algorithm which avoids weather and leads the RPA to the destination, but rather to develop an algorithm which sets standards for weather avoidance manoeuvres, makes the RPA more predictable under link-loss and always follows the rules of air as outlined in ICAO Annex 2 [3]. The scope can be subdivided into following points:

- **Detect:** To first avoid a weather occurrence, it is important to detect it. The project assumes the presence of an on-board weather radar as well as a ground based weather information system. For this purpose, a radar emulator and a

dataset named OPERA to simulate ground based weather service was obtained from Airbus Group Innovations (AGI)

- **Plan:** Once the weather is detected, the algorithm is supposed plan a path to avoid the weather and propose a standardized manoeuvre to the next weather free zone on the original flight path. The WA path should be generated by considering the severity of the weather, airspace violations caused by avoidance and operational feasibility of the path

- **Intent reporting:** If the algorithm decides to conduct an avoidance manoeuvre, it's intent of diverging from the original flight path should be reported to the ATC well in advance so that the ATC has got enough time to act on this change. The developed algorithm should have a functionality of intent reporting

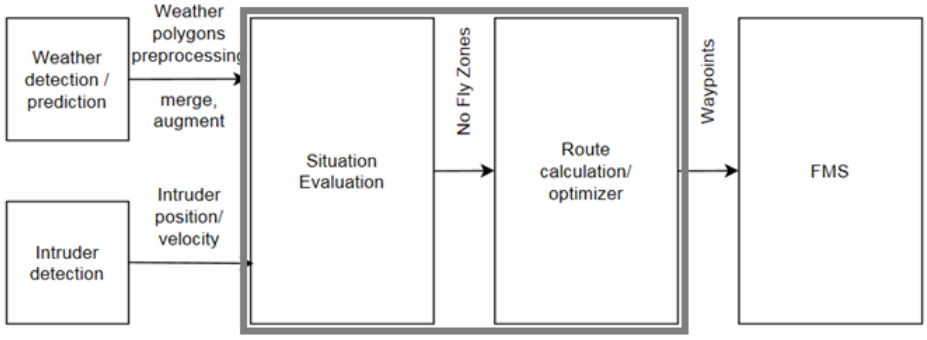


Figure 2: Detect and Avoid block diagram

The figure above describes a functional diagram of the overall 'Detect and Avoid' procedure. The intruder (other aircraft) and weather scenarios are detected and input to the situation evaluation block. This is where the scope of this project lies. Evaluating the already available weather situation, planning an avoidance path and submitting the path to the Flight management System(FMS). The FMS takes takes care of the inner loop, autopilot controls of the RPA.

1.6.2 Limitations

The project is limited to certain criteria which can be subdivided as follows:

- **Lateral Deviations:** The weather avoidance algorithm is limited to thunderstorms and cumulonimbus clouds. Turbulences, Icing etc. situations require altitude changing manoeuvres which is out of the scope of the project. Any manoeuvres proposed in this project are lateral, considering the altitude remains the same

- **Inner-loop control:** The goal of this project is only to propose avoidance manoeuvre in the form of waypoints. The inner loop control aspects, i.e. the

autopilot flight control is out of scope of the project, the WA algorithm acts as a guidance function which proposes manoeuvre waypoints in terms of geographical coordinates to the autopilot.

- **Weather Forecast:** Weather forecast is a complicated topic since the weather phenomena are dynamic, they break into multiple events and join together to regroup into a bigger weather event. The models used for weather forecasting are complicated and are not always reliable and therefore there is no certifiable weather forecasting service yet available to be implemented on a RPA. Thus, the effect of forecasting moving weather is limited to self made weather scenarios assuming that they can be forecasted with a certain degree of accuracy.

- **Weather:** Although the weather data from radar and ground based source is available, the processing of this weather data like merging, augmentation to create convex polygons is out of the scope of the project

- **Flightpath:** The original flightpath is always linear and does not involve any type of curvatures. The *source* and *target* are always collinear

The scope of this project limits us to *on-line* planning since the WA algorithm plans a manoeuvre to avoid dynamic weather and is therefore expected to address changes in the weather scenarios. It would be clear later that there has been no research available on *on-line* path planning for weather avoidance while considering the rules of air and standardized procedures and therefore this project stands out as one of a kind.

The upcoming section discusses categories of path planning, contribution of researchers into path planning for weather avoidance and presents briefly various algorithms categorized into four major sections for path planning. The pros and cons of each category are discussed and an algorithm is picked from each category to be developed so as to accommodate the changing weather data, standardized path planning methods for weather avoidance and intent reporting.

2 Survey of Path Planning Algorithms

Path and trajectory planning have been a keen area of interest for researchers as it provides room for improvement and innovation in many aspects. There is an essential difference between path and trajectory planning. *Path planning* is all about finding an optimal way of connecting two locations whereas *Trajectory planning* addresses how an agent will move from one position to another [9]. While considering the planning term w.r.t UAV applications, path planning would concern itself with calculating an optimal path for the UAV to fly from a source position to a target position. Trajectory planning would involve determining an optimal combination the input state signals, acceleration, velocity, degrees of freedoms of the UAV which allows it to fly from source to target.

Path or trajectory planning is further classified into *off-line* and *on-line* planning [9]. Off-line planning allows the algorithm an access to huge data and sufficient time to process the data to plan a path or trajectory. Generally, the obstacles are static or show considerably less movement. On the contrary, On-line path planning methods use simple methods for modeling the problem and thus can find solution faster than off-line methods. On-line methods can easily address moving obstacles and can readily plan a time dependent path.

The essential goal of Path Planning is to calculate a path from a initial state X_{init} to a goal state X_{end} while avoiding infeasible intermediate states. The infeasible states may include an impractical velocity or acceleration, a position coinciding with an obstacle, a *No-Fly Zone* for a UAV etc. The factors to be considered during path planning are *Optimality* (Is the obtained solution globally or locally optimal), *Completeness* (Will a solution be found in finite time if one exists) and *Complexity* (How difficult it is to calculate the path) [10].

2.1 Path Planning for Weather Avoidance

A number of researchers have contributed for developing standardized weather avoidance methods while addressing safety considerations. Matthews et al. [11] presented a study to quantify the performance of weather avoidance fields in predicting the impact of convective weather on en route airspace. The weather model identified the reagions of convective weather that pilots are likely to avoid and provided a probabilistic weather avoidance field to automate decision support systems of the future impact of weather on ATC. Prevot et al. [12] discussed about the development and evaluation of procedures for air traffic control operations with separation and weather avoidance assurance. Gauci et al. [13] worked on the pilot evaluation of a decision support tool to assist the crew when dealing with the problem of weather detection and avoidance. Krozel et al. [14] compared methods to generate weather avoidance paths. They used three methods namely Standard Arrival Routes (STARs), geometric optimization methods for planing non-intersecting flight paths and greedy prioritization methods for

the same. Xie et al. [15] proposed an A* based algorithm to improve safety operation while avoiding weather.

The above cited literature focuses upon the weather avoidance and seperation problem from the perspective of the ATC and therefore qualifies for *off-line* path planning. Some of the research dealing with *on-line* path planning for weather avoidance is Ramu et al. [16] work on creating cloud ranging and detection algorithm to avoid clouds. The algorithm was able to differentiate between ground, sky and cloud regions in order to plan deviation manoeuvre. Pannequin et al. [17] proposed a non-linear model predictive control based algorithm for aircraft motion planning. Mattei et al. [18] described a method to generate continuously differentiable optimal flight trajectories in the presence of obstacles by generating a weighted oriented graph and solving complex quadratic programming optimization problems for generating arcs. [10] [19] [20] addressed *on-line* trajectory optimization and obstacle avoidance for UAVs using methods like MILP.

Over the recent years, there have been lots of contribution to 2D path planning methods which are basically classified into four different categories namely *Graph based*, *Mathematical Model based*, *Sampling based* and *Bio inspired* algorithms [21]. These categories differ with respect to the methods used to represent the environment, optimal searching, inspiration, complexity and several other factors. A brief introduction of each of these categories and some well known developments are given in the upcoming section.

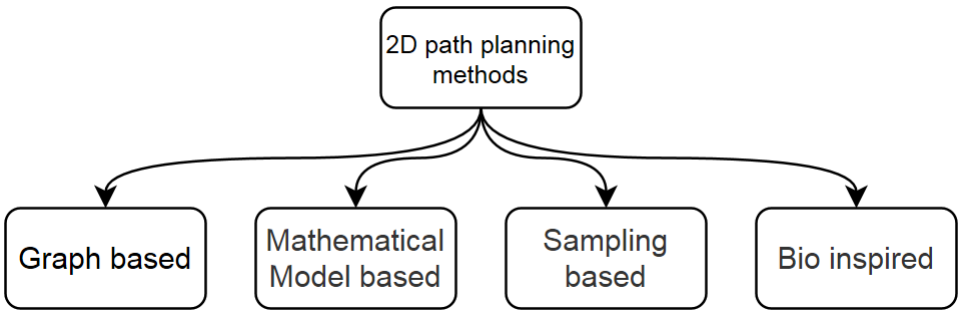


Figure 3: General categories of path planning methods

2.2 Graph based Algorithms

Graph based algorithms or *Node based algorithms* involve the environment represented in terms of a graph composed of nodes. There are costs involved to move from one node to another and the search method calculates a set of nodes starting from *source* to *target* resulting into lowest cost.
To generalize, consider a set of nodes called a *Graph* with N_{graph} number of nodes u_i. Two nodes u_i *and* u_j are same if $i = j$ and different if $i \neq j$. There involves a node for the agent to begin with u_{source} and the target node u_{target} both $\in Graph$. Let E be a set of edges leaving the node u_i and thus, $e_{ij} \in E$ is an edge leaving from v_i to $v_j \in Graph$. The graph can either be directed i.e. $e_{i,j} \neq e_{j,i}$ or undirected i.e. $e_{i,j} = e_{j,i}$ depending upon the problem that is being solved. The search algorithm is called *forward search* if the search begins from u_{source} to u_{target} and *backward search* if the search begins from u_{target} and ends at u_{source} [9]. Some of the important node based terminologies are

- **Unexpanded Node:** The node which has not yet been reached by the search algorithm
- **Open Node:** The node which has been reached by the search algorithm and has atleast one neighbour which is Unexpanded
- **Closed Node:** The node which has been reached by the search algorithm and all of it's neighbours are Open Nodes

The pseudocode for Generic Graph Search algorithm is given as follows:

Algorithm 1 Generic Graph Search

```
Require Graph, u_source, u_target
Initialize OPEN, CLOSED
Add u_target to OPEN list
while OPEN is not empty do
    Pick a node u_i from OPEN and add to CLOSED
    if u_i ∈ u_target then
        Break
    end if
    for each neighbour u_j of u_i not in CLOSED do
        if u_j is not in OPEN then
            Add u_j to OPEN list
            Set parent of u_j to u_i
        end if
    end for
end while
```

- The algorithm begins with selecting the u_{source} node and starts exploring it's neighbours

- The algorithm ends when the target node u_{target} becomes a Closed Node
- Each closed node has a parent which is used to backtrack the path from u_{target} to u_{source}
- Notice that the method for picking up a node u_i from $OPEN$ list is not mentioned. This is the core of any graph search algorithm and there have been several variety of algorithms depending upon the policy to pick a node from $OPEN$ list

Some of the versions of graph search method are *Depth-First Search*, *Breadth First Search*, *Best-First Search* [9], One of the earliest *optimal Best-First* search method namely *Dijkstra's algorithm* [22] was the foundational basis for most of the path planning research work in the field of robotics.

2.2.1 Dijkstra's algorithm

The algorithm aims at finding the optimal path when the cost for travelling through the edges $e_{ij} \in E$ are known. The costs to travel from u_i to u_j along the edge e_{ij} would be denoted by $c(u_i, u_j)$ henceforth. This cost depends upon the formulation of the problem, usually the Euclidean distance between two nodes for path planning, but may also involve other aspects like the amount of work done to travel, amount of money spent, time required to travel etc. in general optimization problems. The main property to be noted here is that each node u_i knows the cost to travel between itself and all its neighbours $u_j \in Neighbour_i$, a set of all the neighbours of u_i. The pseudocode for Dijkstra's algorithm is given as follows:

Algorithm 2 Dijkstra's algorithm

Require $Graph, u_{source}, u_{target}$
Initialize: $OPEN, CLOSED, cost(Graph) \leftarrow \infty$
Add u_{source} to $OPEN$ list and set $cost(u_{source}) \leftarrow 0$
while $OPEN$ list is not empty **do**
  Pick node $u_i \leftarrow min(cost(OPEN))$
  if $u_j \in u_{target}$ **then**
    Break
  end if
  for each neighbour $u_j \in Neighbour_i$ not in $CLOSED$ **do**
    if u_j is not in $OPEN$ **then**
      Add u_j to $OPEN$ list
    end if
    if $cost(u_j) < cost(u_i) + c(u_i, u_j)$ **then**
      $cost(u_j) \leftarrow cost(u_i) + c(u_i, u_j)$
      Set parent of u_j to u_i
    end if
  end for
end while

- The algorithm begins by selecting u_{source} and all its unexpanded neighbours
- It can be seen that the algorithm creates a search tree by choosing a parent with minimum cost of travel
- What makes it different from the Generic Graph Search algorithm is the node picking criterion. Dijsktra's algorithm always picks the node which has the lowest cost in the entire graph
- As seen from the algorithm, the search tree is reordered if a cheaper path is found by setting a different parent
- Dijkstra's algorithm is complete for finite graphs [9]

Although the Dijkstra's algorithm was a great development over other Graph Search methods, it has a couple of drawbacks. Dijkstra's algorithm does a non-directed search i.e. the algorithm does not take the advantage of the location of the target node. This increases the computation cost since the nodes which could have been avoided for expansion are also evaluated.

2.2.2 A* algorithm

In 1968 Hart et al. [23] published a complete, optimal path planning algorithm which picks up nodes in an informed manner. They introduced a heuristic estimation of the cost from any particular node to the target node to achieve faster convergence. The heuristic estimate between current node u_i and the target node u_{target} would be denoted by h_i henceforth.
Considering the coordinates of nodes in the environment $\in \mathbb{R}^2$ the Heuristic estimate can be given by several ways . Let $x_{i1}, x_{i2}, x_{t1}, x_{t2}$ be the x and y coordinates of node u_i and u_{target} respectively.

- **Euclidean Distance:** $h_i = \sqrt{(x_{i1} - x_{t1})^2 + (x_{i2} - x_{t2})^2}$
- **Manhattan Distance:** $h_i = |x_{i1} - x_{t1}| + |x_{i2} - x_{t2}|$
- **Diagonal Distance:** $h_i = \max(|x_{i1} - x_{t1}|, |x_{i2} - x_{t2}|) + (\sqrt{2} - 1)\min(|x_{i1} - x_{t1}|, |x_{i2} - x_{t2}|)$

where $|f|$ is the absolute value of f.

Algorithm 3 A* algorithm

Require: $Graph, u_{source}, u_{target}$
Initialize: $OPEN, CLOSED, cost(Graph) \leftarrow \infty$
Add u_{source} to $OPEN$ list
Set $cost(u_{source}).g \leftarrow 0$, $cost(u_{source}).h \leftarrow h_{source}$
$cost(u_{source}).f \leftarrow cost(u_{source}).g + cost(u_{source}).h$
while $OPEN$ list is not empty **do**
  Pick node $u_i \leftarrow min(cost(OPEN).f)$
  if $u_j \in u_{target}$ **then**
    Break
  end if
  for each neighbour $u_j \in N_i$ not in $CLOSED$ **do**
    if u_j is not in $OPEN$ **then**
      Add u_j to $OPEN$ list
    end if
    if $cost(u_j).g < cost(u_i).g + c(u_i, u_j)$ **then**
      $cost(u_j).g \leftarrow cost(u_i).g + c(u_i, u_j)$
      Set parent of u_j to u_i
    end if
    $cost(u_j).h \leftarrow h_j$
    $cost(u_j).f \leftarrow cost(u_j).g + cost(u_j).h$
  end for
  Remove u_i from $OPEN$ and Add it to $CLOSED$
end while

- As observed from the algorithm, A* uses three different costs namely f, g and h. g is the true cost of reaching the node same as in Dijkstra's algorithm, h is the heuristic estimate from current node to the target node and f is the sum of g and h
- A* picks up a node with minimum f cost as contrary to Dijkstra where the true cost g is used. This makes A* better since it also considers the heuristic while picking new nodes
- Since A* considers heuristic, it performs a directed search in the direction of minimum sum of cost and heuristics
- If $h_j \leq c(u_j, u_{target})$ then the heuristic is said to be *optimistic* and *admissible*. When the algorithm uses admissible heuristic, the obtained path is guaranteed to be optimal w.r.t cost [23]

There are several modifications of A* like *Dynamic A** [24] which is a sensor based algorithm that changes the weights of it's edges to form a temporal map. *D* LITE* [25], *Lifelong A** [26], *Anytime A** [27] which have their own specialities and advantages

over A*. These methods are not discussed or used anywhere later in this project since A* gives a sufficient platform to develop WA algorithm which is discussed later.

2.3 Mathematical Model Based Algorithms

Node based methods or Graph Search algorithms decompose the environment into a set of nodes. Thus, the agent can only provide a solution in terms of the available decomposition and therefore such methods might not be able to consider all the system dynamics and kinodynamic constraints. This is where *Model Based Algorithms* come into picture. These methods model the dynamics, path constraints, obstacles and in general the environment mathematically. An objective cost function is bounded by these constraints and minimized using optimization techniques. Some of the most commonly used methods include the following:

2.3.1 Mixed Integer Linear Programming (MILP)

Optimal trajectory planning with collision avoidance is represented as a linear objective function subjected to linear and mixed integer constraints and is therefore called *MILP- Mixed Integer Linear Programming*. MILP approach has been used for aircraft trajectory planning with hard obstacle avoidance [19], multi agent path planning [28], receding horizon approach for UAVs [20] etc.
A standard MILP optimization problem has the following formulation:

$$\begin{aligned} \min_x f(x) &= c^T x \\ b_L &\leq Ax \leq b_U \\ x_L &\leq x \leq x_U \end{aligned}$$

Where $A \in \mathbb{R}^{m \times n}; c, x, x_L, x_U \in \mathbb{R}^n; b_L, b_U \in \mathbb{R}^m$. A subset of $x_I \in x$ are restricted to take integer values. Optimal solution is the vector x^* for which the value of the objective function $f(x^*)$ is the minimum.
For a path planning problem, aircraft dynamics are modeled by a set of discrete system of equations:

$$\bar{x}(k+1) = A\bar{x}(k) + B\bar{u}(k)$$

where $\bar{x}$ is the state vector, $\bar{u}$ is the command vector, A is the state matrix and B is the command matrix.
The objective function is given as the sum of performance function $l(...)$ over time T and goal function $f(...)$ as follows:

$$J_T = \sum_{k=0}^{T-1} l(\bar{x}(k), \bar{u}(k)) + f(\bar{x}(T))$$

The constraints imposed are as follows:

$$\bar{x}(k+1) = A\bar{x}(k) + B\bar{u}(k)$$
$$\bar{x}(k) \leq \mathbb{X}(k)$$
$$\bar{u}(k) \leq \mathbb{U}(k)$$
$$\bar{x}(k) \neq \mathbb{O}_i(k)$$

Where, $\mathbb{X}(k)$ are the set of boundary constraints $\mathbb{U}(k)$ are the set of command constraints $\mathbb{O}_i(k)$ are the set of Obstacle constraints
For the i^{th} obstacle defined by with $N_{ob,i}$ number of nodes to form a convex polygon, each edge is considered a half plane. If the position state variables $\bar{x}_{out}(k) \subset \bar{x}(k)$ lie outside at least one half plane, the agent can be considered to be outside the obstacle. Thus the set of obstacle constraints $\mathbb{O}_i(k)$ can be modeled as follows:

$$K\bar{x}_{out}(k) - Mb \leq d$$
$$\sum_{h=1}^{N_{ob,i}} b_h \leq N_{ob,i} - 1$$

Where, $K \in \mathbb{R}^{N_{ob,i} \times n_{pos}}$, $\bar{x}_{out}(k) \in \mathbb{R}^{n_{pos} \times 1} \subset \bar{x}(k)$ is the column vector of position states, $M >> 0$ is an arbitrarily large number, $b \in \mathbb{R}^{N_{ob,i} \times 1}$ are the set of binary variables, $d \in \mathbb{R}^{N_{ob,i} \times 1}$. The next constraint guarantees that the position state $\bar{x}_{out}(k)$ satisfies at least one half plane constraints.

- MILP generates an optimal path over each time-step for the planned time T
- Algorithms which guarantee to find an optimal solution like *cutting planes, branch and bound, dynamic programming* can be used to solve the MILP problem but these algorithms may take exponential number of iterations
- Several non-linear constraints have to be linearized to formulate the MILP criterion. Moreover, a general purpose optimization solver is not suitable for MILP, thus instead of linearizing functions, a similar path planning problem can be formulated with Optimal Control path planning

2.3.2 Optimal Control

Optimal control problems are a subclass of dynamic optimization problems which involve time-dependent variables and time-dependent constraints. The general system model can be represented in the form of ordinary differential equations as

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

Where $x(t), u(t)$ are called *state variables* and *control variables* respectively. The objective function to be minimized is given as

$$\min_{x(.)u(.)} \phi(x(t), u(t))$$

Subjected to necessary initial and final constraints

$$\psi(x(t_0), t_0) = a$$
$$\psi(x(t_f), t_f) = b$$

where t_0, t_f are the beginning and final time for which the problem has to be optimized. Thus, the above problem can be converted into unconstrained Lagrangian form as:

$$L = \phi(x(t), u(t)) + \int_{t_0}^{t_f} \lambda^T \{f(x(t), u(t)) - \dot{x}\} dt$$

Where λ is the Lagrangian multiplier vector. The Hamiltonian can be given as

$$H(x, \lambda, u, t) = \lambda^T f(x(t), u(t))$$

The Hamiltonian is used to solve the optimal control problem based on the maximum principle to find an optimal path within the planned time frame $[t_0, t_f]$ [21]. Tisdale et al. [29] proposed a variant of optimal control problem for receding horizon approach, Gan et al. [30] proposed a multi agent target search using gradient based negotiation using the optimal control framework.

2.4 Sampling Based Algorithms

These methods sample the environment as a set of nodes or cells and then search the samples to find the feasible path on the contrary to node based methods where a graph is already present. There is no provision to add or delete a node in graph search algorithms and thus the environment decomposition cannot be refined, but with intelligent sampling algorithms, these methods can refine the environment over time. Methods like *Rapidly exploring Random Trees (RRT)*, *Probabilistic Road Maps (PRM)*, *Visibility Graphs*, *Artificial Potential Fields* [21] are some of the most popular sampling based algorithms. Some of them are explained here:

2.4.1 Rapidly expanding Random Trees (RRT)

RRT [31] searches the configuration space to generate a path from the source node to the target node. A *Configuration Space* C is a set of all the possible transformations applicable to an agent (robot or a UAV). A configuration space consists of obstacle free C_{free} and obstacle filled region $C_{obs} = C \backslash C_{free}$. At every iteration, a new node $u_{rand} \in C_{free}$ is sampled and it's feasibility with nearest node is checked to create a path tree T. The RRT algorithm is given as follows:

Algorithm 4 RRT

Require: $u_{source}, u_{target}, N_{itr}$
Initialize: Add vertex u_{source} to the tree T
for $i = 1 : N_{itr}$ **do**
  $u_{rand} \leftarrow$ Randomly sample in the C_{free} space
  $u_{near} \leftarrow$ The nearest vertex to u_{rand} in tree T
  $u_{new} \leftarrow$ Reconfigure$(u_{near}, u_{rand}, C_{obs})$
  if c$(u_{new}, u_{target}) \leq \epsilon$ **then**
    $u_{new} \leftarrow u_{target}$
  end if
  Add vertex u_{new} to the tree T
  Set parent of u_{new} to u_{near}
end for

- As seen from the basic RRT algorithm, random node u_{rand} is sampled every iteration in C_{free}
- Nearest node u_{near} is calculated to u_{rand} and since u_{rand} might not always be feasible, maybe because the edge from u_{near} might pass through C_{obs} or if system constraints aren't followed, a new node u_{new} is generated which is feasible and lies in C_{free}
- The new node is added to the tree T with it's parent set to u_{near}
- This process continues until the number of random sampling nodes N_{itr} are reached. The sampling process could also be ended when the u_{target} node is found, but the obtained path is not always optimal
- RRT does not guarantee asymptotic optimality [32]
- For non-asymptotic approach, the solution obtained is non-unique due to the randomness induced in sampling

Several improvements have been done in RRT algorithm w.r.t sampling techniques over years [33], [34], [35] to improve it's efficiency until 2010 when RRT*, a method confirming asymptotic optimality was introduced.

2.4.2 Rapidly exploring Random Tree Star (RRT*)

RRT* [32] is an extension of classical RRT which finds the initial rough path fairly quicker than other algorithms and then keeps on optimizing the path with more number of samples [36] and ensures asymptotic optimality. The RRT* pseudocode is given as follows:

Algorithm 5 RRT*

Require: $u_{source}, u_{target}, N_{itr}, rad$
Initialize: Add vertex u_{source} to the tree T
for $i = 1 : N_{itr}$ **do**
  $u_{rand} \leftarrow$ Randomly sample in the C_{free} space
  $u_{near} \leftarrow$ The nearest vertex to u_{rand} in tree T
  $u_{new} \leftarrow$ Reconfigure$(u_{near}, u_{rand}, C_{obs})$
  $cost(u_{new}) \leftarrow \infty$
  if $c(u_{new}, u_{target}) \leq \epsilon$ **then**
    $u_{new} \leftarrow u_{target}$
  end if
  $U_{set} \leftarrow$ Nodes from tree T lying within radius rad from u_{new}
  for each node $v \in U_{set}$ **do**
    if $cost(u_{new}) < cost(v) + c(u_{new}, v)$ **then**
      Set parent of u_{new} to v
      $cost(u_{new}) \leftarrow cost(v) + c(u_{new}, v)$
    end if
  end for
  Add vertex u_{new} to the tree T
  for each node $v \in U_{set}$ **do**
    if $cost(v) > cost(u_{new}) + c(u_{new}, v)$ **then**
      Set parent of v to u_{new}
    end if
  end for
end for

- As observed, the RRT* algorithm is almost similar to the basic RRT algorithm
- The only difference is when a set of nearest nodes U_{set} lying in the neighbourhood of radius rad are calculated
- RRT* tries to remove the connections with large costs by connecting via u_{new} because of which, the tree turns out to be dense
- The time complexity increases and it cannot generate multipath [37]

2.5 Bioinspired Algorithms and Machine Learning

Such algorithms try to mimic the biological behaviour in order to solve complex optimization problems, do not require complex mathematical models and aim at finding out the solution by strong searching methods [21]. Bioinspired algorithms are mainly classified into two types namely *Evolutionary Algorithms* and *Neural Networks*.

2.5.1 Evolutionary Algorithms (EAs)

Evolutionary algorithms were conceptualized to solve problems which the usual linear programming, dynamic programming failed to solve. EAs [38] begin by selecting feasible solutions in the first generation randomly and then takes into consideration various constraints like obstacles, movement limitations to evaluate the fitness of each individual. A sample of these individuals are selected to reproduce for the next generation according to their fitness. The final step is the mutation and crossover step. *Mutation* is an operator used to maintain the genetic diversity whereas *Crossover* operator is used to vary the genetic encoding from one generation to the next. The process is iterative and it stops when a desired pre-set goal is achieved. Then, the individuals with the best fitness are decoded as the optimal path nodes. *Genetic Algorithms* are some of the most widely used population based Evolutionary algorithms, a few others are *Ant Colony Optimization* [39], *Particle Swarm Optimization* [40], *Memetic Algorithm* [41] etc.

2.5.2 Neural Networks and Machine Learning

Where EAs tend to mimic the biological behaviour of living creatures, Neural Networks aim at mimicking the neural activities in a brain. NNs just like EAs do not require a mathematical model and thus cannot develop rules for decision making which raises doubts about their reliability. There have been attempts to develop efficient NN models until recently with the huge rise in computational capacities and Big Data availability, the *Deep Neural Networks* have gained much popularity.

Machine Learning is a field of Computer Science which enables a system to learn without defining a set of pre programmed rules [42]. ML is a set of algorithms which focus primarily on the two paradigms of *Classification* and *Regression*.

- *Classification*: The problem of determining with a degree of certainty to which specific class an unknown observation belongs
- *Regression*: The problem of determining the relationship between dependent and independent variables, i.e. how the value of a dependent variable changes with a change in the independent variable values

Machine learning can also be categorized depending upon the availability of data [43]:

- *Supervised Learning:* Type of ML in which the dataset contains the inputs as well as the desired outputs. The goal is to find the relationship between inputs and outputs
- *Unsupervised Learning:* Type of learning where the inputs are known, but the desired output in unknown. The goal is to determine some sensible relationship between the input data

- *Reinforcement Learning:* The type of learning where data is obtained by interacting with an environment and improving over time by getting a feedback or reinforcement. There have been discussions about RL belonging to Supervised or unsupervised learning, but it is better distinguish it into a whole new class since it belongs to a different paradigm

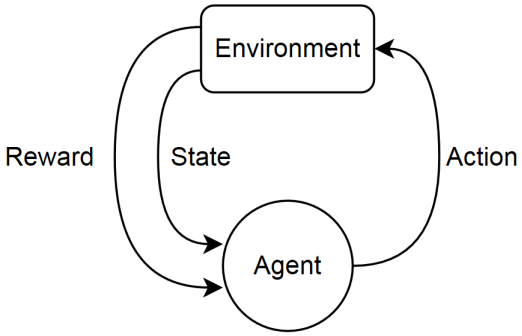


Figure 4: Reinforcement Learning sketch

There have been several different Machine learning algorithms specializing in their individual tasks, not all will be discussed in this project. RL on one hand has proved to be a tool for path planning and is analogous to animal behaviour where for e.g. a dog is offered a treat for performing a rolling action and it learns to perform better rolls to get treat which is reward.
Once the agent in RL learns from the feedbacks, it can aptly decide which action to take in order to get the maximum reward. This learning from experience can be used to develop a path planning algorithm. Although training the agent takes too much time, once the agent is trained, this approach gives to one of the quickest solutions.

2.6 Analysis

Summarizing the four classes of algorithms above, one can conclude that

- Node based algorithms:
  - Decompose the environment into a set of nodes to make a graph
  - A* holds a strong case since it is directed by using heuristic estimate whereas Dijkstra's algorithm is undirected
  - A* is one of the chosen algorithms since it guarantees *optimality* and *completeness*
- Mathematical Model based algorithms:
  - MILP and optimal control problems have been applied in several path planning problems. These methods can model the environment very well and have been proven for *online* planning
  - Such methods require a complicated solver in order to solve mixed integer and constrained problems, global *optimality* can be attained and the *complexity* depends upon the polynomial equation [21]
  - A Non-Linear programming method used to model the weather obstacles and relatively less number of constraints is proposed in the upcoming section
- Sampling based algorithms:
  - These methods explore the C space by sampling nodes but lack in the time complexity
  - RRT* guarantees asymptotic convergence which RRT lacks and since the convergence is not guaranteed in a finite time, the direct application of RRT* is not a good choice for our application
  - A method combining the completeness guarantee by A* and sampling properties of RRT* has been proposed in the upcoming section
- Bio Inspired algorithms:
  - Evolutionary algorithm holds no case for *online* path planning because of it's high time complexity
  - Reinforcement learning on the other hand can prove to be a useful tool once the agent is sufficiently trained
  - A well trained agent by using Deep Neural Networks is proposed in the upcoming section as an innovative approach to path planning

Although this is the initial critical analysis of path planning algorithms, their true performance w.r.t time complexity and optimality would be estimated with the help of static weather cases and the algorithms which stand out as the best choices would then encounter dynamic weather. The steps followed to develop the algorithms to address the scope of this project are explained in the next section.

3 Methodology

Having had a look at different categories of path planning algorithms in the previous section, it is now time to conceptualize and develop over the problem at hand. As discussed before, the objective is to develop a *complete*, *certifiable* path planning algorithm for weather avoidance. The Radio Technical Commission for aeronautics (RTCA) has set up guidelines for the safety of safety-critical software approval [44] for certification agencies like EASA. A software used by airborne services is certified when it meets certain requirements like a thorough documentation of software Code Standard, Software Design document, Test Cases and Procedures, Verification results etc.
Commercial model based design, validation and code generation software packages like SCADE [45] are used for the purpose of conducting simulations. SCADE generates certifiable C code and has a drag and drop modeling interface which makes it time consuming to perform simple tasks. Although the algorithms and test cases in this project have been developed in MATLAB [46], a general idea of keeping the algorithms simple and basic is always considered with the goal of being certifiable in the near future.

3.1 Conceptualization

An RPA flying in it's cleared flight path when detects weather phenomena has to rely upon the WA algorithm to suggest manoeuvre waypoints. Being airborne comes with a set of responsibilities for an RPA as follows:

Airspace Corridors: An aircraft is provided with corridors, analogous to a highway in the air. An aircraft is not expected to violate its corridor boundaries because it may interfere with another aircraft's airway. Thus, in the case of occurrence of a hazardous weather situation which lies in the original flight path, it's supposed to conduct an avoidance manoeuvre considering the least violation of it's corridor.

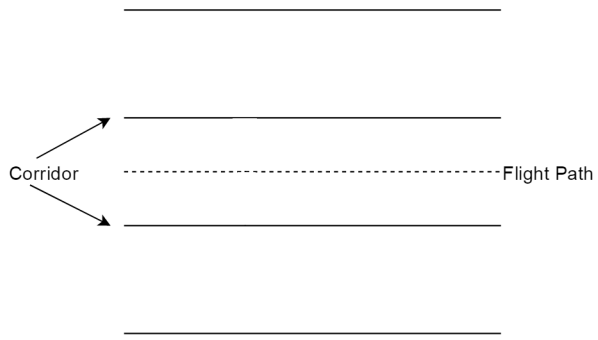
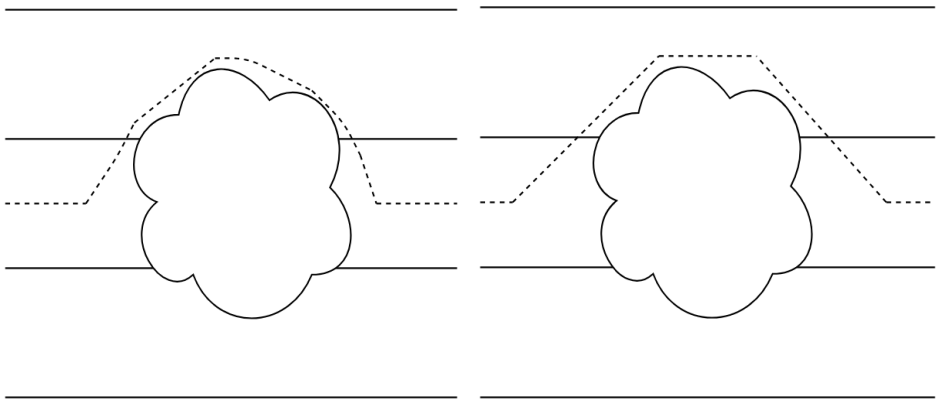


Figure 5: Airspace corridor

The figure 5 shows an example of an airspace corridor which is a boundary lying parallel to the flight path [47].

Link Loss: An RPA is linked to it's RPS via a C^2 link. When the link is active, the RP has a full responsibility of the RPA, but in case the RPA loses it's link, the RPA depends solely upon the autonomous functionalities programmed within it. During a link loss phase, if the RPA detects a hazardous weather situation, it has to perform an autonomous avoidance manoeuvre while considering the rules and regulations described.

Predictability: One of the main concerns while conducting autonomous flights is predictability during link loss. An RPA is not only responsible for conducting a safe flight by itself, but it's also responsible for not causing any disruptions in the flight plans of another aircraft. While avoiding weather under linkloss, the ATC should be able to know about the exact manoeuvres the RPA does in advance so that it can take measures for the safety of other aircraft. Thus, when flying autonomously, the aircraft should always take standardized measures thus remaining predictable.



(a) Non-Standard manoeuvre (b) Standard manoeuvre

Figure 6: WA manoeuvres

The figure 6(a) shown a non-standardized manoeuvre. Although the goal of weather avoidance has been achieved, this type of manoeuvre makes the RPA unpredictable during link loss. The time/location where the aircraft starts to divert and converge is unknown. Whereas in figure 6(b), although the RPA spends more time outside it's own airspace, all the actions taken are standardized. The intent of the RPA to conduct standardized manoeuvres are reported to the ATC beforehand which is termed as *intent reporting* which gives it enough time to take into consideration the safety of own as well as other aircraft. Thus, the goal of this project is not only to develop a weather avoidance algorithm, but also to standardize the processes and tasks needed for conducting a weather avoidance manoeuvre.

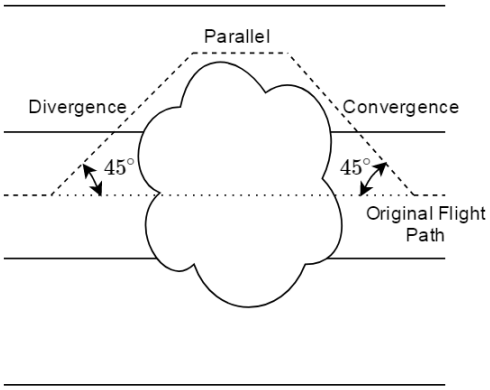


Figure 7: Standardized manoeuvre †

45° Deviation: An avoidance manoeuvre can basically begin at any feasible angle w.r.t the original flight path, but to standardize the process, for the scope it was decided that in any of the algorithms to be developed, a deviation will begin and terminate at angles ±45 degrees to the original flight path since the flight path is considered straight. This standardizes the procedure of *Divergence* and *Convergence* of an RPA with the original flight path while conducting an avoidance manoeuvre.
Divergence: is when the aircraft departs from the original flight path to conduct an avoidance manoeuvre
Convergence: is when the aircraft arrives back to follow it's original flight path to complete it's avoidance manoeuvre

Parallel manoeuvre: Once the RPA has conducted the divergence manoeuvre, it would conduct a flight parallel to the original flight path at a distance suitable to avoid the weather phenomenon. These standards are sufficient to plot a perfect avoidance plan. Since the angle and location at which an RPA deviates is fixed and also the manner in which it flies in order to avoid weather is fixed, it's easier to visualize a trapezium as shown in the figure 7. These standards will always be considered while developing any of the upcoming weather avoidance algorithms. Also, the effect on path planning without following standards is shown in one of the test-cases in figure 53.

The concepts/ implementations indicated by † are original ideas and aren't referred from any literature

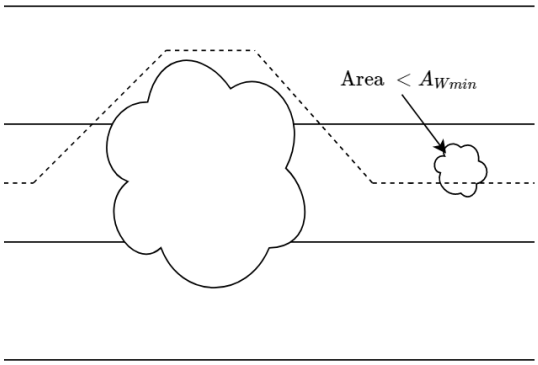
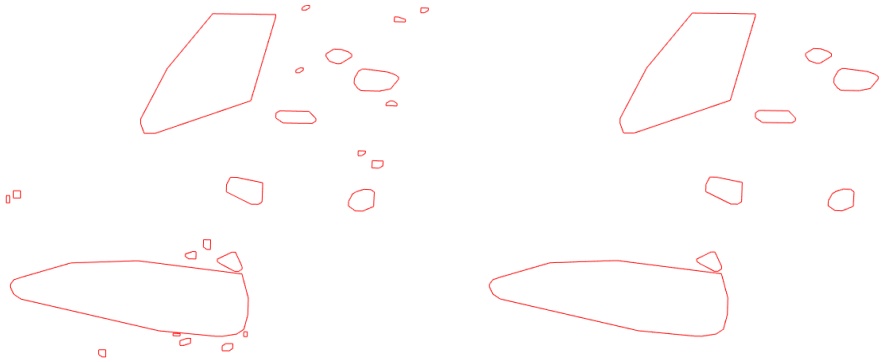


Figure 8: Fly-through weather

Fly-through Weather: Not every weather phenomenon poses a threat to the aircraft. Some weather occurrences last long while other disappear relatively quickly while most of the times, it is not worth deviating for an insignificant weather phenomena. Usually, in manned flights, some weather occurrences are ignored and the aircraft flies through. In this project, the weather phenomena below a certain area A_{Wmin} are not considered significant enough in order to deviate by the avoidance algorithm and the path is planned without considering them.



(a) Full weather scenario (b) Filtered weather scenario

Figure 9: Weather filtering

The weather data can be filtered to accomplish the above task. The figure 9(a) shows an unfiltered weather scenario whereas (b) shows the same weather scenario after filtering. In case of mathematical model based algorithms where the model complexity increases with each weather phenomena, filtering severely reduces the complexity since a less number of weather polygons are considered for calculation.

Waypoints: An aircraft flies along it's flightpath which is defined by a set of waypoints. Mathematically defining, a waypoint is a state boundary condition which is

defined by geographical coordinates, the altitude, velocity, acceleration etc. that the aircraft is expected to be in at a particular point. For this application, a waypoint $\in \mathbb{R}^2$ would be defined only by the latitude and longitudinal coordinates. Some of the general waypoint terminologies used which are used several times further in the project are:

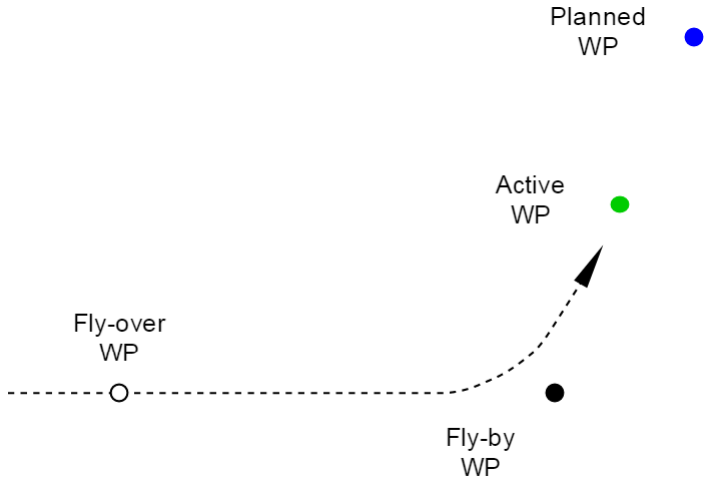
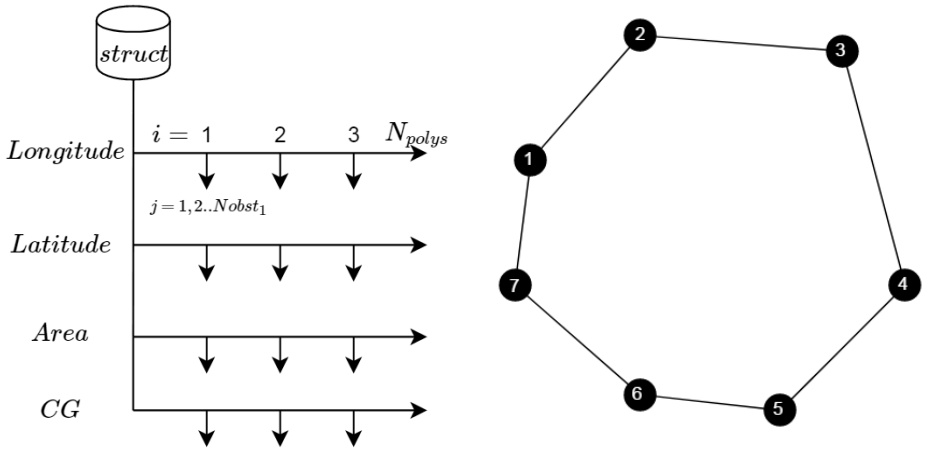


Figure 10: Waypoints

- **Fly-over Waypoint** The waypoint which must actually be crossed by an aircraft while following it's flight path
- **Fly-by Waypoint** The waypoint which marks the intersection of two paths and must not necessarily be passed through an aircraft
- **Active Waypoint** The waypoint which is being followed (always called as WP_{active} henceforth)
- **Planned waypoint** The waypoint which is planned to be followed after passing the Active Waypoint (always called as $WP_{planned}$ henceforth)

Weather data: The RPA is equipped with an onboard weather radar and also gets periodic weather information from ground based sources. The weather data is obtained in the form of a data structure in xml format defining cloud polygons. The polygons are always guaranteed to be convex and are defined by a set of nodes with their geographical latitude and longitudinal coordinates, total area occupied by each polygon and the center of gravity of each polygon.

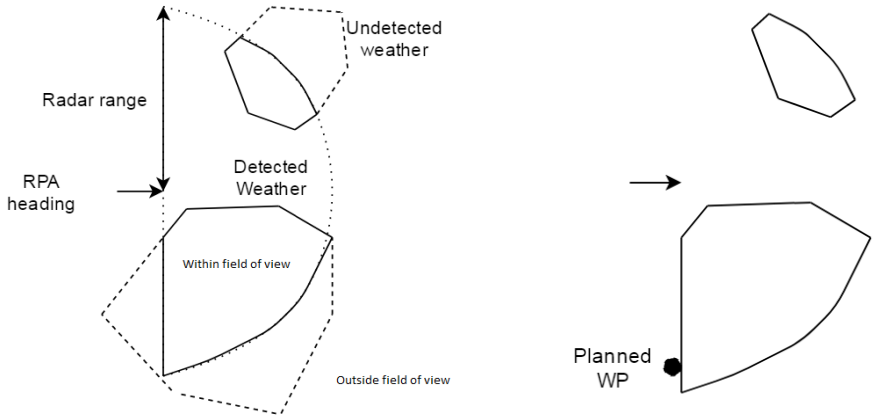


(a) Weather structure (b) Weather polygon

Figure 11: Weather data-structure

Consider a weather scenario with N_{polys} number of detected polygons. As shown in figure 11(a), The data structure *struct* consists of four fields $Longitude, Latitude, Area$ and CG. Considering that i^{th} polygon consists of $Nobst_i$ number of nodes and edges, the figure 11(b) shows a convex polygon with seven nodes. There are two main types of weather services available depending upon their sampling time and the availability range as follows

Weather Radar: provides weather data w.r.t the heading direction of the RPA. In our application, a simulated weather radar data model is provided which returns a set of weather cloud polygons within a range of $\approx 60\ kms$. The Radar data has a visibility range, i.e. the RPA can only sense weather which lies within it's visibility range at a particular instant. The RPA has no clue about the weather phenomenon that lies beyond it's radar range and this poses challenges to the planning algorithm.



(a) Radar range detection (b) Effect of radar data

Figure 12: Radar weather data

As seen in figure 12(a), the weather radar model only detects weather phenomenon lying within it's range. It has no clue about the nature of the undetected weather, this causes the weather polygons to look like in 12(b). This loss of weather data due to radar limitations causes problems for the WA algorithm. As seen in case(b), the algorithm may return the planned waypoint just along the weather range boundary and it's feasible since the waypoint does not lie inside the detected weather phenomenon either. This makes it necessary to implement some constraints which would be discussed in the upcoming sections. The mis-planning which occurs due to radar data-loss is also mentioned in the test cases section.

OPERA weather data: As discussed, RADAR provides the real-time weather data but is limited to it's range and the heading direction of the aircraft, there exists a weather data provider which gives the weather information of the whole scenario. The OPERA weather provides the weather cloud scenario although it lacks the real-time aspect of the radar. OPERA provides a static image of the weather scenario at every quarter of an hour i.e. the sampling time for this service is 15 minutes.

(a) OPERA at time T (b) OPERA at time T+15 mins

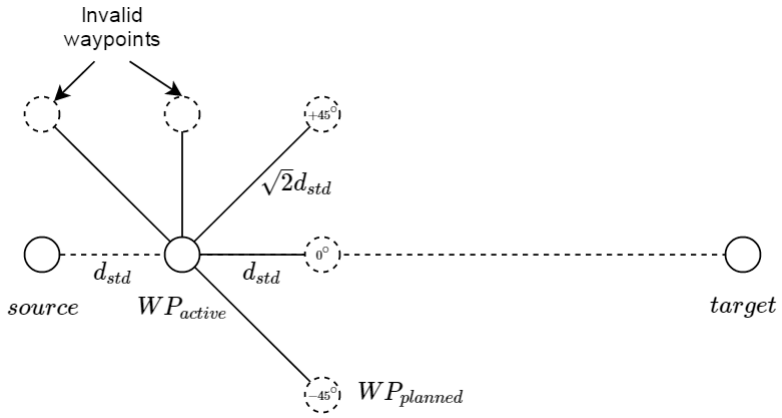
Figure 13: OPERA weather data

The figure 13 shows two consecutive OPERA scans which are 15 minutes apart. Since the OPERA data is not updated in real-time, it is hard to extrapolate to calculate forecasts, moreover the path planning algorithm faces problems because of the sampling time whenever there is drastic change in the weather polygons. Since the WA has no idea of the weather change status in between samples, it happens that the RPA find itself suddenly inside a polygon after the latest update. This case is mentioned in the test cases section later in figure 55.

Time Dependence of path: Most of the algorithms discussed above do not address time dependence. Dijkstra, A*, RRT prove to be effective whenever the obstacles are static whereas Optimal Control, MILP address dynamic obstacles. In this project, the first stage of development was to proceed considering static weather situation, i.e. a single scan of OPERA data was conducted and it was considered that the weather scenario stays the same throughout the journey of the RPA. But this hardly happens in real scenarios. Weather phenomena are constantly changing, thunderstorms are constantly moving, clouds keep moving, splitting and joining from each other. Thus it is absolutely necessary to take into consideration the movement of weather scenarios.

In an RPA with link-loss, since the human decision making capability is absent, there is a need to take into account a way to address moving weather. It would help if there was a service which provides a weather forecast which could be taken into consideration for path planning but at the moment, there is no such certifiable service upon which the WA algorithm can always rely. Thus, taking into consideration operational aspects, a qualitative study of the effect of No Forecast vs. Forecast has been done later in this project.

Before we start with developing path planning algorithm, it is necessary to put forth a set of rules and regulations addressing the previously discussed problems and



standardization concepts. All the proposed algorithms are developed considering these rules as their basic foundation. Summarizing the above mentioned standardizations

- An aircraft can only deviate $0, \pm 45$ to the $target - source$ vector and the planned waypoint resulting these deviations shall be placed at standardized distances $d_{std}, \sqrt{2}d_{std}$ from the active waypoint. The $target - source$ vector is henceforth called v_{ts}
- Conducting a U-turn manoeuvre is out of the scope of the planning algorithms and thus, the planned waypoint would be in the direction such that $0 < \frac{(WP_{planned} - WP_{active}) \cdot (v_{ts})}{||WP_{planned} - WP_{active}|| \ ||v_{ts}||} \leq 1$
- Since the dynamics of the aircraft are controlled by the FGS, WA is not a trajectory planning problem but rather a path planning problem while addressing limitations
- The algorithm developed should always be *complete* in the sense that under no circumstances should the algorithm return a null solution

While performing a Weather Avoidance Manoeuvre (WAM), a new flight path has to be planned such that the hazardous weather situation is avoided. This new flight path can be easily described by a set of new waypoints. Ideally, according to the White Paper [2], a Weather Avoidance algorithm after encountering a hazardous weather situation is supposed to provide a set of manoeuvre waypoints until the next weather-free situation. After performing this manoeuvre, the FGS follows the original flight path.
Since the actual FGS system was unavailable to perform the simulations on time, a simple waypoint control algorithm has been used. All the waypoints used are Follow-by, for a particular case, the aircraft flies with a constant average speed v, No dynamics of the RPA have been considered. In the actual application, the newly planned waypoints will be fed into the FGS and it will carry out the control operations.

3.1.1 Waypoint Control

A pure pursuit guidance law is used to follow waypoints and vaguely imitate the FGS. The geometric rule for pure pursuit is to always keep the aircraft oriented towards the Active Waypoint. This is demonstrated in the following figure [48]

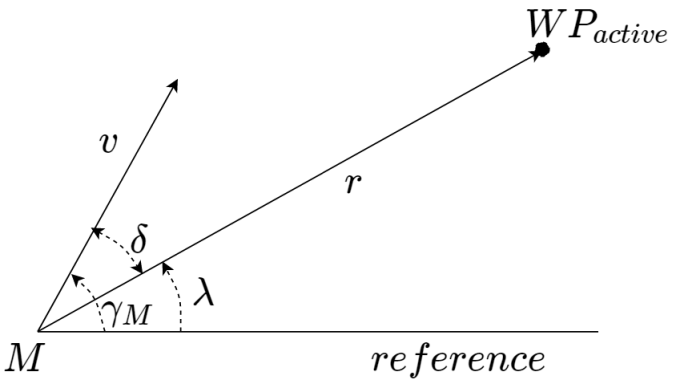


Figure 14: Waypoint control sketch

Given, WP_{active} is the Active Waypoint, vector v denotes the aircraft velocity, r is the vector from the object M to WP_{active} also known as LOS or *Line Of Sight* vector, λ is the angle r makes w.r.t $reference$, γ_M is the angle the velocity vector v makes with the same $reference$. δ is the error between λ and γ_M and thus geometrically, $\gamma_M = \lambda + \delta$.

Pure pursuit would have to control the object M such that the error angle δ is minimum so that the vector v coincides with r. The guidance law for implementing this [48] is given as

$$a_{M_c} = v\dot{\gamma}_M = -f(\delta) = -f(\gamma_M - \lambda) \tag{1}$$

Where $f(*)$ is an odd-symmetric function like $\sin(*)$ and a_M is the lateral acceleration of M

A simple *velocity pursuit* guidance law could be applied here for a case where the target to be followed is stationary since WP_{active} is not moving. The kinematic equations for the moving object M can be given as

$$\begin{aligned} \dot{r} &= -v\cos(\delta) \\ r\dot{\lambda} &= -v\sin(\delta) \end{aligned} \tag{2}$$

Using $sin(*)$ as the odd-symmetric function $f(*)$ in the guidance law in equation 1 would give

$$\dot{\gamma_M} = -\frac{Kv}{r}\sin(\delta) \tag{3}$$

Where K is a constant. From the equation 2, we have $\dot{\lambda} = -(v/r)\sin(\delta)$ and $\dot{\lambda} = \dot{\gamma_M} - \dot{\delta}$ which results into

$$r\frac{d\delta}{dr} = (K-1)\tan(\delta) \tag{4}$$

The differential equation 4 can be solved by

$$\frac{\sin(\delta)}{\sin(\delta_0)} = \left(\frac{r}{r_0}\right)^{K-1} \tag{5}$$

Thus, according to this guidance law, as $r \to 0$ $\delta \to 0$ given $K > 1$.

However, since

$$a_M = v\dot{\gamma} = -\frac{Kv^2\sin(\delta_0)}{r_0}\left(\frac{r}{r_0}\right)^{K-2} \tag{6}$$

it can be concluded that as $r \to 0$, $a_M \to 0$ given that $K > 2$.

In the actual implementation, the $reference$ is always considered to be $x\ axis$ which means that all the angles are measured w.r.t the positive $x\ axis$. The angles are measured as usual i.e counter-clockwise and are always positive i.e. they lie within the domain $[0, 2\pi)$

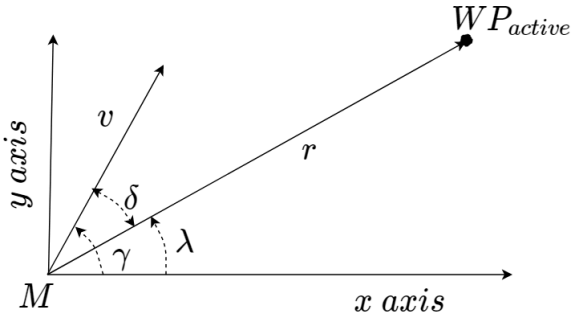


Figure 15: Guidance law implementation

Based upon the guidance law in equation 3 and the figure, 15 the following kinematic equations can be written

$$\dot{x} = v\cos(\gamma)$$

$$\dot{y} = v\sin(\gamma)$$
$$\dot{\gamma} = -\frac{Kv\sin(\lambda)}{r}$$

Considering the sampling time be T, the above equations can be discretized as follows

$$x(t+1) = x(t) + Tv\cos(\gamma(t)) \tag{7}$$
$$y(t+1) = y(t) + Tv\sin(\gamma(t)) \tag{8}$$
$$\gamma(t+1) = \gamma(t) - T\frac{Kv\sin(\lambda)}{r} \tag{9}$$

The Waypoint Control algorithm is given as follows:

Algorithm 6 Waypoint Control

Require: $source, target, v, T, K, N_{wpc}$
Initialize: $X(source_x, source_y, \gamma_{start}), t$
Scan $\leftarrow$ Perform Weather Scans
Forecasts $\leftarrow$ Calculate Forecast(Scan)
$WP_{active} \leftarrow$ WA($source$, Forecasts)
for $k = 1 : N_{wpc}$ **do**
  $r \leftarrow ||WP_{active} - X(k, 1:2)||$
  if $||X(k, 1:2) - target|| \leq \epsilon$ **then**
    break
  end if
  if $2Tv \leq r < 3Tv$ **then**
    Scan$\{1\} \leftarrow$ Perform Weather Scan
  end if
  if $Tv \leq r < 2Tv$ **then**
    Scan$\{2\} \leftarrow$ Perform Weather Scan
    Forecasts $\leftarrow$Calculate Forecast(Scan)
    $WP_{planned} \leftarrow$ WA(WP_{active}, Forecasts)
    $WP_{active} \leftarrow WP_{planned}$
    $r \leftarrow ||WP_{active} - X(k, 1:2)||$
  end if
  $\lambda \leftarrow$ Calculate λ
  $X(k+1, 1) = X(k, 1) + Tv\cos(X(k, 3))$
  $X(k+1, 2) = X(k, 2) + Tv\sin(X(k, 3))$
  $X(k+1, 3) = X(k, 3) + T\frac{-Kv\sin(\lambda)}{r}$
  $t \leftarrow t + T$
end for

- From the Waypoint Control algorithm 6, it is noted that N_{wpc} is the number of maximum number of iterations set for the algorithm, T is the sample time,

$K > 2$ is the constant from the guidance law, v is the aircraft velocity which is constant throughout the simulation

- The state of the RPA X can be defined by it's location x, y coordinates i.e. *Longitude* and *Latitude* coordinates and γ, it's inclination w.r.t x *axis* or the equator
- After the initialization, weather scans with a time gap of 2 minutes are done which are used to generate weather forecasts
- The active waypoint WP_{active} is calculated using WA algorithm, this waypoint is to be followed by the WP control algorithm
- LOS vector r is updated every iteration which is used to perform more weather scans when the RPA reaches in the vicinity of radius $3Tv$ of WP_{active}
- When the RPA is closer to WP_{active}, the previous weather scans are used to generate a forecast which is used to plan the planned waypoint $WP_{planned}$ by the WA algorithm
- The WP_{active} is switched by $WP_{planned}$ which is now to be followed by the RPA, the state of the aircraft X is then calculated according to the guidance law
- Time variable t is incremented with every iteration by sample time T

3.1.2 Weather Forecast

As described in the previous section about the importance of time-dependence in aviation, there is currently no certifiable forecast service. Thus, for the sake of simulations, a simple linear extrapolation has been done on moving weather polygons which helps us later in the qualitative study.

Although *forecast* may sound like a complicated procedure, what it actually means is to include the equations of motions of the obstacles into the planning algorithm. This could also be termed as a non-communication or sensor based obstacle avoidance since the obstacles and the agent do not communicate but rather the obstacles are detected by sensors or basically the radar.
The following steps have been mentioned in the Waypoint Control algorithm [6]

$$\begin{aligned} \text{Scan}\{1\} &\leftarrow \text{Perform Weather Scan} \\ \text{Scan}\{2\} &\leftarrow \text{Perform Weather Scan} \\ \text{Forecasts} &\leftarrow \text{Calculate Forecast(Scan)} \end{aligned}$$

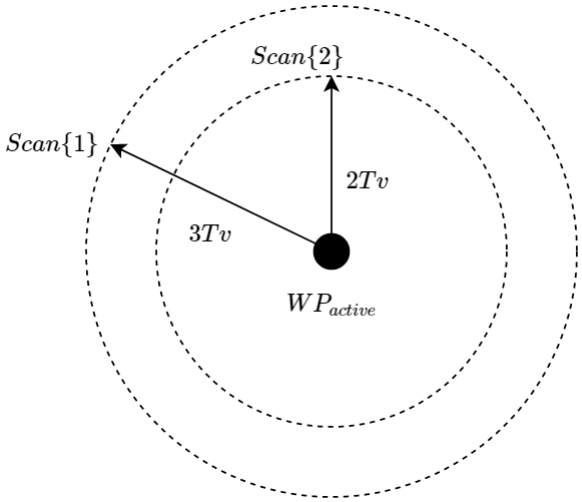


Figure 16: Weather scan neighbourhood[†]

When the RPA approaches the WP_{active}, the first weather scan Scan$\{1\}$ is done at any time $t_1 = t$ when it enters the outer neighbourhood of radius $3Tv$ and the second scan Scan$\{2\}$ is conducted at time $t_2 = t+T$ when it further enters the inner neighbourhood of radius $2Tv$. These two scans are used by the forecasting algorithm to conduct an extrapolation for the time $t_2 + N_t\Delta T$ where ΔT is the time step while N_t is a real number ≥ 0

Consider a weather data structure obtained during the first scan be $struct_{t1}$ at time t_1 and another obtained during the second scan be $struct_{t2}$ at time t_2. Since the two scans were conducted in a considerably less time difference T, they are considered to have same number of polygons.

Consider a particular i^{th} polygon for a cloud from $struct_{t1}$ and $struct_{t2}$ with it's respective CGs CG_i^{t1} and CG_i^{t2}. Let the polygon contain $Nobst_i^{t1}$ and $Nobst_i^{t2}$ number of defining nodes. The simple linear extrapolation to calculate the forecast $CG_i^{t2+N_t\Delta T}$ at time $t_2 + N_t\Delta T$ is given as

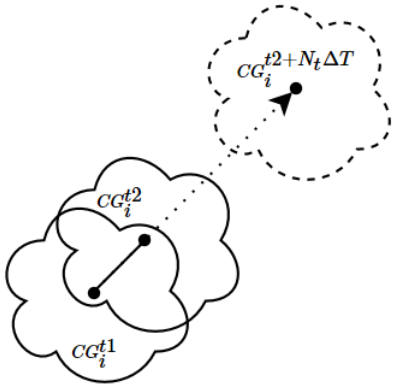


Figure 17: Weather extrapolation

$$
\begin{aligned}
dt &= T \\
ds &= CG_i^{t2} - CG_i^{t1} \\
v_i &= \frac{ds}{dt} \\
CG_i^{t2+N_t\Delta T} &= CG_i^{t2} + N_t \Delta T v_i
\end{aligned}
$$

The overall forecasting algorithm is given by

Algorithm 7 Calculate Forecast

Require: $T, struct_{t1}, struct_{t2}, N_t, \Delta T$
Initialize: $struct_{t2+N_t\Delta T}$
for each polygon i in $struct_{t2}$ **do**
  $v_i \leftarrow \frac{CG_i^{t2} - CG_i^{t1}}{T}$
  for each node in polygon i **do**
    Extrapolate node and save in $struct_{t2+N_t\Delta T}$
  end for
end for

- Since algorithm [7] is not a sophisticated weather forecasting algorithm, the reliability is unknown and therefore self created polygons moving with a constant velocity are considered wherever forecast is switched on. This algorithm is used to demonstrate the effects of plugging in motion of weather in WA algorithm.

3.2 A* Path Planning algorithm

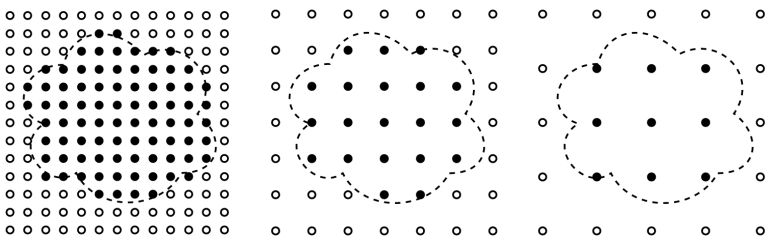
A generic A* algorithm works for a simple path planning case, but in order to include the rules of air and airspace as mentioned before, some modifications were done in the algorithm. This includes a perfect planning of the graph to be generated, considering airspace violation, reducing the number of twists developed in the planned path, guaranteeing 100% solution existence. Moreover, since weather is always dynamic, a new approach has been adopted in order to address polygon movements. Following is the stepwise description of the algorithm development

3.2.1 Grid

One of the most important aspects of A* algorithm is the graph. A* provides solutions only w.r.t the decomposition of the environment i.e. the solution will consist coordinates present only in the graph nodes. Considering the above set standards of deviating manoeuvre, parallel manoeuvre, airspace violation etc. it makes sense to create a graph in the shape of a square grid. This makes it possible to make $\pm 45°$ and even parallel deviations.

One aspect to be considered before constructing a grid is that the solution for optimal path between *source* and *target* without any obstacles should lead to a series of nodes which lie on a straight line. Thus, if the grid is rectangular, it makes sense to construct it such that the *source* and *target* lie either on the opposite vertices or lie on opposite edges with symmetric intersection ratio. To address this aspect, a square grid is constructed with odd number of nodes so that the *source* and *target* nodes lie exactly in the center of opposite edges.

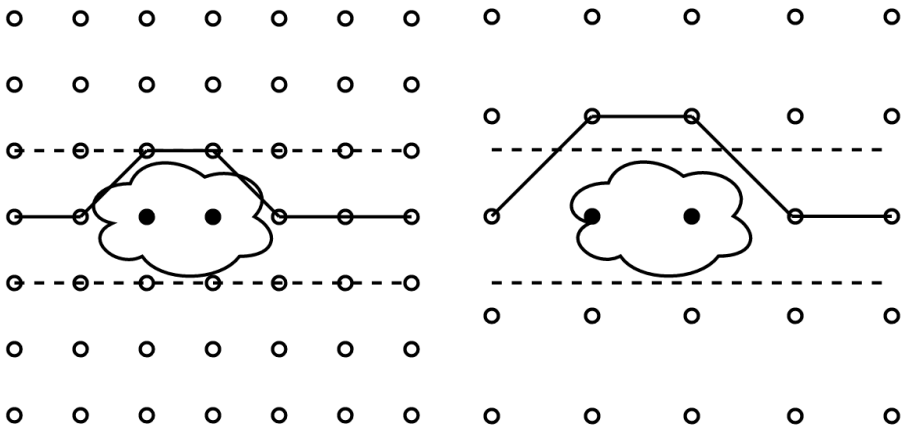
Another important aspect is deciding the node distance. The grid nodes store the information of obstacles, i.e. in this particular case, if a node lies within or on the edge of a weather cloud be noted. The more dense the grid is, the more computationally complex it is for the algorithm to calculate the optimal path as more number of nodes are to be evaluated. The sparse the grid is, the more information is lost as seen in the figure 18.



(a) Dense grid (b) Optimal grid (c) Sparse grid

Figure 18: Grid density

The dense grid stores precise information about the cloud shape, but it increase the computational cost. The medium grid loses some information about the cloud shape, but it is less computationally expensive than the denser grid. On the other hand, the sparse grid would be the least expensive, but the cloud shape information is considerably lost which might prove risky in the case of path planning.



(a) Optimal grid (b) Sparse grid

Figure 19: Grid density and airspace corridors

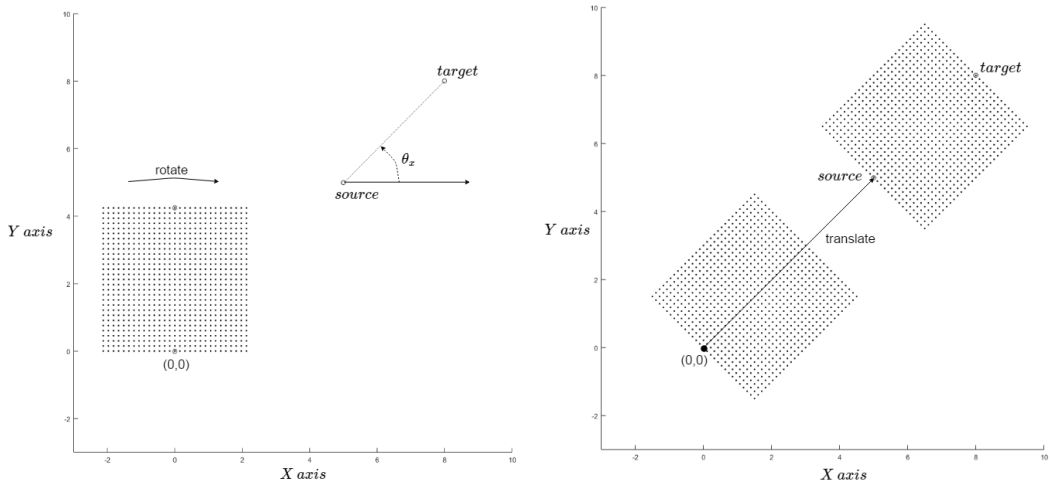
The node distance also depends upon the airspace corridors. As seen from the figure 19(a) an optimally distributed grid reduces the airspace violations, although some of the manoeuvred path passes through the weather cloud. Whereas in 19(b), since the node distance is greater than the half airspace corridor length, any avoidance manoeuvre would always cause an airspace violation.
Thus, there is always a trade-off between the weather information to be retained, the computational cost and the airspace violation cost involved while considering the node distance for the grid. While planning and conducting simulations, the total number of nodes in the grid were calculated by

$$N_{nodes} \leftarrow max\left(ceil(||v_{ts}||/d_{corr}) + 1, 3\right)^2 \tag{10}$$

$$NodeDistance \leftarrow ||v_{ts}||/(\sqrt{N_{nodes}} - 1) \tag{11}$$

- The $*^2$ at the end ensures that the total number of nodes is a square number, $||.||$ denotes a standard euclidean norm
- *ceil* guarantees that the result of $||v_{ts}||/d_{corr}$ is always an integer > 0 while d_{corr} is the half of total corridor width i.e. 10 Nautical miles
- Since in any case, there should be atleast 3 nodes along the grid edge, one each for *source* and *target* and one in between. This is guaranteed by $max(*, 3)$ function
- The equation 10 creates grid with an approximate node distance of d_{corr} which creates an ideal case as shown in figure 19(a)

To create a grid such that the *source* and *target* nodes lie exactly on the center nodes of opposite edges,



(a) Rotation (b) Translation

Figure 20: Transformation operations

- A square grid is created with edge distance of $||v_{ts}||$ and N_{nodes} number of total nodes. The grid spans on $\pm$ x $axis$ and positive y $axis$ in which the origin $(0, 0)$ lies on the center node of the bottom edge
- θ_x, the inclination of the v_{ts} vector is calculated w.r.t positive x $axis$ and every node in the grid is rotated by $\theta \leftarrow \theta_x - \frac{\pi}{2}$ along the origin
- After the rotation operation, each node is translated by the *source* vector such that the grid matches the *source* and *target* coordinates
- Considering $\begin{bmatrix} node_{x,i} \\ node_{y,i} \end{bmatrix}$ be the x and y coordinates of i^{th} node, the new coordinates $\begin{bmatrix} node_{x',i} \\ node_{y',i} \end{bmatrix}$ is given by

$$\begin{bmatrix} node_{x',i} \\ node_{y',i} \end{bmatrix} \leftarrow \begin{bmatrix} source_x \\ source_y \end{bmatrix} + \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} node_{x,i} \\ node_{y,i} \end{bmatrix}$$

A coordinate matrix of size $[N_{nodes} \times 2]$ is created to store the coordinate information of all the nodes. The index of each node is the node number of that particular node.

Since a node in a graph is connected to a couple of other nodes called it's neighbours, the construction of a square grid allows each i^{th} node to have a maximum of eight neighbours. To store this relation between nodes, a connection matrix is created

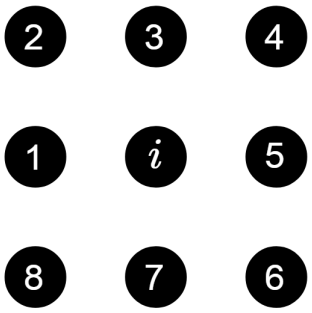


Figure 21: Neighbouring nodes

The size of the connection matrix is $[N_{nodes} \times 8]$ and the row index is the node number. Thus i^{th} node will be stored in the i^{th} row of the matrix and each of it's eight columns will contain the node numbers of it's neighbouring nodes, the pattern is shown in the above figure. In case a node does not have a particular neighbour, it's column will have -1 value.

3.2.2 Obstacles

As described in the WPC algorithm 6, the function 'Perform Weather Scan' simulates a weather scan performed by the RPA. Generally in A* the nodes falling inside the obstacles are inserted into the $CLOSED$ nodes list. Thus, these nodes are never the part of the solution space and guarantees 100% obstacle avoidance. In the cases where no solution exists, A* does not provide any solution as expected. It would not be good if our algorithm gives an indeterministic solution and thus, in this implementation, every obstacle node coordinates are not inserted in the $CLOSED$ list, but rather a different method is used to avoid obstacles.

Algorithm 8 Obstacle Cost†

Require: $Node$, polygon $struct$
Initialize: $cost_{obst} \leftarrow 1$
for each polygon P in $struct$ **do**
  if Node lies inside or on P **then**
    $cost_{obst} \leftarrow W$ $\{W >> 1\}$
    break
  end if
end for

This is done by checking whether a particular node lies inside any of the obstacle polygons in a given scenario and assigned a $cost_{obst}$

- If $Node$ lies within or on edge of any of the polygons, it's $cost_{obst}$ is assigned a high value W else it assumes the value of 1. If $cost_{obst}$ has a high value, it will take a relatively higher cost to pass through that particular node.

- Since the cloud nodes are not inserted in the $CLOSED$ list, all the nodes are available for evaluation. In a case where there's a cloud passing over the *source* or *target*, the algorithm would never return a $Null$ path. The algorithm would in a worst case, consider passing through a cloud with the lowest cost but would always guarantee a path

3.2.3 Corridor

According to the previous implementation, the algorithm would try to avoid hazardous weather situations but flying out of own airspace corridor while avoiding clouds is not recommended. Thus, another cost $cost_{corr}$ for addressing corridor violation is added.

Algorithm 9 Corridor Cost†

Require: $Node, source, target$
$d \leftarrow \perp$ distance of $Node$ from the segment joining *source* and *target*
$cost_{corr} \leftarrow \frac{d}{d_{corr}}$ $\{d_{corr}$ is half of the total corridor width as described before$\}$

- The $cost_{corr}$ is directly proportional to the perpendicular distance from the segment joining *source* and *target* nodes, which motivates the algorithm to reduce it as much as possible, thus reducing the airspace violations if possible

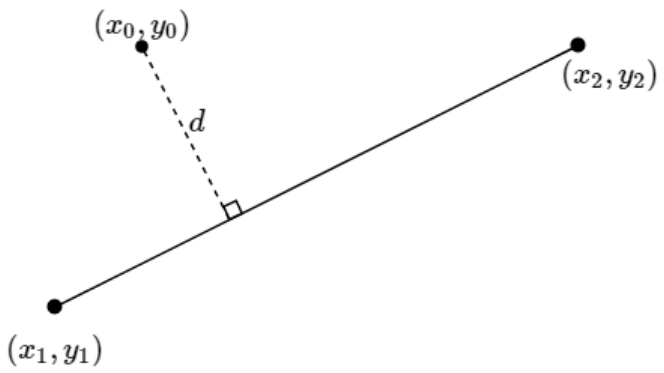


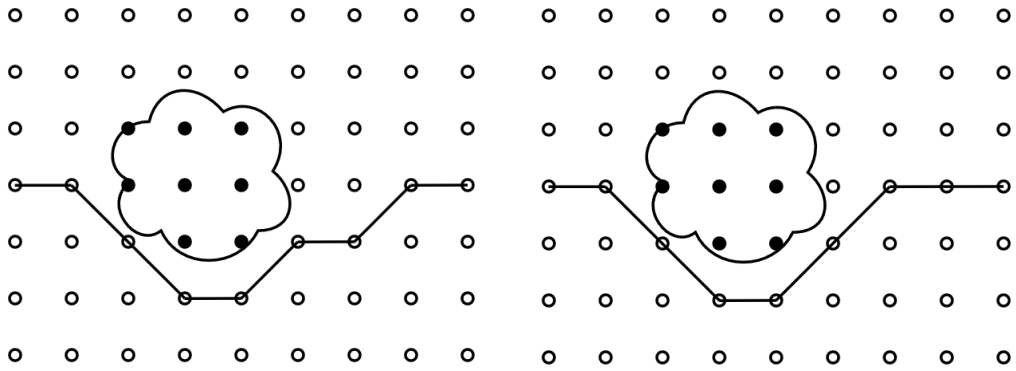
Figure 22: Perpendicular distance of a point from a segment

If $x_0, y_0, x_1, y_1, x_2, y_2$ are the x and y coordinates of given $Node$, *source* and *target*, the perpendicular distance between the node and the segment is given by

$$d = \frac{|(y_2 - y_1)x_0 - (x_2 - x_1)y_0 + x_2y_1 - y_2x_1|}{\sqrt{(x_1 - x_2)^2 + (y_2 - y_1)^2}} \tag{12}$$

3.2.4 Twists

A* is a cost minimizing algorithm and finds a path with the help of heuristic estimate. Consider a path planning problem where the distance between the nodes is the only cost of travelling.



(a) Path with more twists (b) Twist optimal path

Figure 23: Twist optimality

In the figure 23, since node distance is the only considered cost, both the paths cost the same. But from the perspective of an aircraft, 23(a) is not optimal since it has more twists along it's path. On the other hand, 23(b) is optimal since it has the least number of twists possible. Thus, there's a need to include twist cost $cost_{twist}$ into the planning algorithm.

Consider the search algorithm has picked a node u which has a parent u_{par} and u_{ci} $i = 1..8$ neighbours. Let's consider the case where $u_{c1} = u_{par}$ and is thus *closed* node and u_{ci} $i = 2..8$ are *open* nodes.

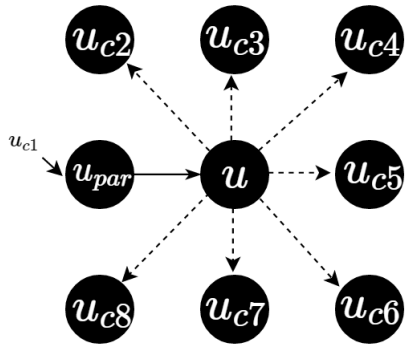


Figure 24: Twist formulation

It is clear from the figure 24 that

- Path $u_{par} \rightarrow u_{c5}$ incurs the lowest cost since it's not twisted

- Paths $u_{par} \rightarrow u_{c4}, u_{c6}$ incur more because they have a 45° twist
- Paths $u_{par} \rightarrow u_{c3}, u_{c7}$ incur even higher cost since they have a 90° twist, which should not be recommended by the algorithm since it violates our standards
- Paths $u_{par} \rightarrow u_{c2}, u_{c8}$ are practically not feasible for an aircraft since they have a 135° twist
- Path $u_{par} \rightarrow u_{c1}$ won't be allowed by the algorithm since u_{c1} is the parent of u

Here, the construction of the connection matrix comes handy. That is the reason to store the children nodes starting from left in a clockwise fashion. It can be seen that the absolute difference between the children and parent node indices has a pattern

- $abs(4 - abs(5 - 1)) = 0$ gives no twist
- $abs(4 - abs(4 - 1)) = abs(4 - abs(6 - 1)) = 1$ gives 45° twist
- $abs(4 - abs(3 - 1)) = abs(4 - abs(7 - 1)) = 2$ gives 90° twist
- $abs(4 - abs(2 - 1)) = abs(4 - abs(8 - 1)) = 3$ gives 135° twist

It can also be noted that this pattern remains the same irrespective of the parent and child node indices. Thus, $cost_{twist}$ can be derived as

Algorithm 10 Twist Cost†

Require: u, u_{par}, u_{ci}, connection matrix
$\theta_{twist} \leftarrow abs(4 - abs($node indices difference of u_{par} and $u_{ci}))$ from the connection matrix(u)
$cost_{twist} \leftarrow \sin(\theta_{twist}\ \frac{\pi}{8})$

- Thus, the $cost_{twist}$ assigned to twist-less path is 0, 45° twist ≈ 0.3827, 90° twist ≈ 0.7071 and 135° twist is ≈ 0.9239

***Note*:** Therefore the overall *optimality* of the algorithm will not be decided only by the length of the planned path but also a combination of *corridor optimality* and *twist optimality*. Any of the discussed algorithms are *optimal* if they generate a path which successfully avoids harmful weather situations with minimum number of twists along the path while reducing the corridor violations as much as possible

3.2.5 Radar Disappearance

It is known that the RADAR data is limited to it's range. The weather data beyond the radar's range is lost and may lead to improper planning as shown in figure 12

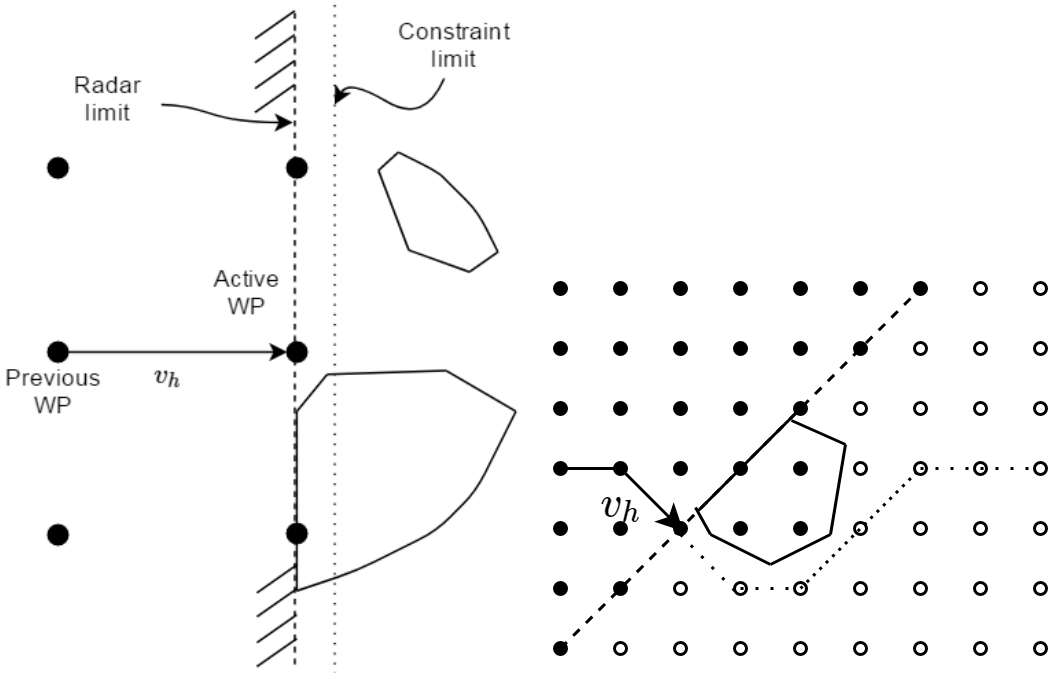


Figure 25: Radar disappearance

This problem can be solved in A* by adding a constraint limiter. As shown in the figure 25(a), the Previous Waypoint WP_{prev}, the Active Waypoint WP_{active} are given and the algorithm is supposed to plan it's path further. Consider the vector $v_h = WP_{active} - WP_{prev}$ and let m be it's slope. The radar limit which leads to data disappearance is perpendicular to the vector v_h and thus has a slope $m' = m^{-1}$.

A quick trick that can be used to avoid the mis-planning due to data disappearance is to use the half-plane created by the radar limit. The half-plane which consists of the WP_{prev} can be considered to be heavy-weather zone, let's call it the $non - feasible$ half-plane. As shown in the figure 25(b), all the nodes that lie in the $non-feasible$ half-plane are considered to be lying inside a weather polygon and thus those particular nodes have $cost_{obst} \leftarrow W$, $W >> 1$. Considering $WP.x$ and $WP.y$ are the x and y coordinates of any waypoint WP, the algorithm to check if a $Node$ lies in the $non - feasible$ half plane is given as:

Algorithm 11 Constraint limit†

Require: $WP_{active}, WP_{prev}, Node$
$v_h \leftarrow WP_{active} - WP_{prev}$
$m' \leftarrow m^{-1} \leftarrow -v_h(1)/v_h(2)$
$a \leftarrow sign(WP_{prev}.y - WP_{active}.y + m'(WP_{active}.x - WP_{prev}.x))$ {equation of the half-plane}
if $a == sign(Node.y - WP_{active}.y + m'(WP_{active}.x - Node.x))$ **then**
  $cost_{obst} \leftarrow W\{W >> 1\}$
else
  $cost_{obst} \leftarrow 1$
end if

The nodes lying on the radar-limit such as $Node_{active}$ itself could cause a truncation error, depending upon the number of significant digits after decimal considered. But just to be safe, the half-plane boundary could be raised by a little bit to avoid any kind of unexpected errors by $v_h = (1 + \epsilon)v_h$ where $0 < \epsilon \leq 0.1$. Thus as shown in figure 25(a), the radar limit is raised upto the constraint limit which safely guarantees that all the nodes on the strict $non - feasible$ half-plane have a high $cost_{obst}$. This method helps in avoiding mis-planning due to data-disappearance as shown in 25(b) since the back nodes carry a high cost.

3.2.6 Time Dependence

Time-dependent A* algorithm is a paradigm in which the path has to be planned considering the movement of obstacles. Likhachev et al. [27] worked with Anytime A* and Wang et al. [49] worked on A* with moving obstacles. In time dependent path planning for robots, there is an easier option to stop at a position or even move back, which is not possible to do with an aircraft. The available possible options provide much more freedom as compared to the problem that has to be addressed in this project.
A new method for time dependent path planning for this specific case has been proposed in this project. This method takes the advantage of the path planning constraints that the algorithm has been subjected to.

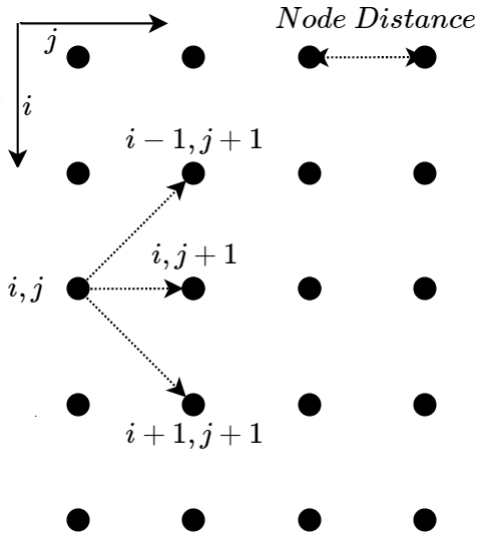


Figure 26: A* grid pattern

Consider a grid with N_{nodes} number of total nodes or $n_{nodes} = \sqrt{N_{nodes}}$ number of edge nodes. Considering that the agent starts from the i^{th} row and j^{th} column at time t, the possible next position for the agent always has to be in the $j + 1^{th}$ column and $i - 1, i, i + 1$ row. The reason being that the agent (RPA) cannot stop, take backward turns and only has to move ahead. Thus, considering constant speed v, the estimated time required to reach these three nodes can be calculated. which is:

$$(i, j) \rightarrow (i - 1, j + 1) \quad \text{needs} \quad \frac{\sqrt{2}NodeDistance}{v}$$

$$(i, j) \rightarrow (i, j + 1) \quad \text{needs} \quad \frac{NodeDistance}{v}$$

$$(i, j) \rightarrow (i + 1, j + 1) \quad \text{needs} \quad \frac{\sqrt{2}NodeDistance}{v}$$

Thus, while planning the path for $i \pm 1^{th}$ row, a weather forecast for that time $t + \Delta T$ could be used. But this requires the system to maintain two different sets of forecasts i.e. for $i \pm 1$ and i^{th} row in $j + 1^{st}$ column. In this case, one could take the advantage of the obstacles not being rigid. Thus assuming that the agent takes same time to reach $i - 1, i$ and $i + 1$ row, the system would have to maintain only one forecast. The time thus taken to reach any of these three nodes in $j + 1^{th}$ column could be approximated as the average of the time it takes to reach diagonal node $(i + 1, j + 1)$ or $(i - 1, j + 1)$ and the horizontal node $(i, j + 1)$

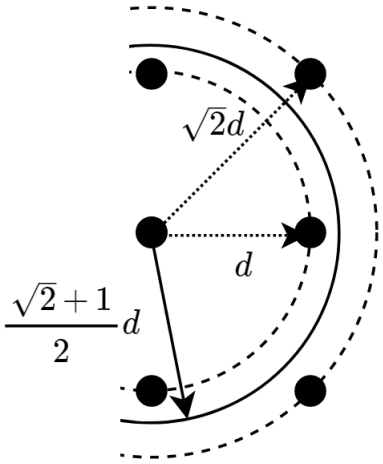


Figure 27: Time approximation[†]

Therefore the time ΔT it takes to reach column $j+1$ would be

$$\Delta T \leftarrow \frac{\sqrt{d}+d}{2v} \leftarrow \frac{NodeDistance_{avg}}{v}$$

Now, for the full path to be time-dependent the system would have to store $n_{nodes}-1$ number of forecasts. The accuracy of forecasts also reduces for the farther future, i.e the prediction for $t+\Delta T$ is more reliable than the prediction for $t+20\Delta T$. The way in which this problem is constructed produces opportunities to tackle with the problem of reliability of predictions.

As the agent's grid column position increments serially $j \rightarrow j+1 \rightarrow j+2 \rightarrow ....$, one could assume that the predictions for the next three nodes, i.e. the ones in the columns $j+1, j+2$ *and* $j+3$ are fairly reliable. Thus, the path can be planned considering only the upcoming three nodes to be time dependent i.e. only until $t+3\Delta T$. All the columns after $j+3$ could use the forecasts for $j+3^{rd}$ column. Now, the system has to store less forecasts, i.e. only three which sounds fair.

***Note*:** One could argue why a set of forecasts is being maintained rather than simply substituting the time in the forecasting algorithm 7. The reason being that since a certifiable forecasting service is not yet available in practise, the way in which forecasts will be available is unknown. Weather forecasts being a complex topic, the forecasts will certainly not be available in a model which maps *time* to *weather data*. It is assumed that RPA would receive the forecast data in the form of same weather polygon data-structure at times $t+\Delta T, t+2\Delta T$ and $t+3\Delta T$ where t is the time the RPA receives the forecast data.

Time dependence can be implemented in A* by simply beginning the planning with j^{th} column of the WP_{active} at time t = 0. Then the $j+1, j+2$ column nodes are

planned w.r.t forecasts for $t + \Delta T, t + 2\Delta T$. All the column nodes from $j + 3...n_{nodes}$ can are planned w.r.t forecasts for $t+3\Delta T$. This is the easiest possible time dependence implementation for A*, but another approach named '$time - cost^{\dagger}$' has been used here since it is helpful for the next proposed sampling based algorithm RRT-A*

Consider a node which denotes the active waypoint

- All the nodes are assigned a *time cost*, $cost(*).t$ and is initialized to ∞. The time cost for beginning node $cost(Node_{begin}).t$ is initialized to 0 since it's the node from where planning begins
- While evaluating a neighbour node u, the time calculated to reach this node can be assigned the as the sum of time cost of it's parent u_{par} and ΔT

$$t_{calc} \leftarrow cost(u_{par}).t + \Delta T$$

- This can be done since it's assumed that in a square grid, the time taken to travel from a parent to it's child is ΔT
- The node cost denotes the time it takes to reach a particular node u. Since each node is desired to be reached in minimum possible time, The lowest time to reach is chosen

$$cost(u).t \leftarrow min(cost(u).t, t_{calc})$$

- This approach favours minimal time required to reach a node u and once the node is closed, the time cost for that node never changes

Using the *time-cost* approach, the forecast for evaluating a particular node u can be chosen easily as

Algorithm 12 Choose Forecast

Require: $cost(u).t$, Forecasts for $t + \Delta T, t + 2\Delta T, t + 3\Delta T$
if $0 < cost(u).t \leq \Delta T$ **then**
  $Weather \leftarrow Forecast$ for $t + \Delta T$
else if $\Delta T < cost(u).t \leq 2\Delta T$ **then**
  $Weather \leftarrow Forecast$ for $t + 2\Delta T$
else if $\Delta 2T < cost(u).t$ **then**
  $Weather \leftarrow Forecast$ for $t + 3\Delta T$
end if

3.2.7 Full A* Algorithm

Considering all the discussed aspects so far, the full A* algorithm can be given as follows:

Algorithm 13 A*

Require: $WP_{active}, source, target, N_{nodes}$, Forecasts, $NodeDistance, Node_{prev}$
Initialize $coord\_mat, conn\_mat, CLOSED, OPEN, cost \leftarrow \infty$
$Node_{begin} \leftarrow WP_{active}$
Add $Node_{begin}$ to $OPEN$ list
Initialize f, g, h, t cost of $Node_{begin}$
while $OPEN$ is not empty **do**
  pick node $u_i \leftarrow min(cost(OPEN).f)$
  if $u_i \in target$ **then**
    break
  end if
  for each neighbour $u_j \in N_i$ not in $CLOSED$ **do**
    if u_j is not in $OPEN$ **then**
      Add u_j to $OPEN$ list
    end if
    $cost(u_j).t \leftarrow min(cost(u_j).t, cost(u_i).t + \Delta T)$
    $Weather \leftarrow$Choose Forecast$(cost(u_j).t,$ Forecasts$)$
    $cost_{obst} \leftarrow$ Calculate Obstacle Cost
    $cost_{corr} \leftarrow$ Calculate Corridor Cost
    $cost_{twist} \leftarrow$ Calculate Twist Cost
    $cost_{obst} \leftarrow$ Calculate Constraint Limit
    $cost_{temp} \leftarrow cost_{obst}(1 + \zeta_c cost_{corr} + \zeta_t cost_{twist})c(u_i, u_j)$
    if $cost(u_j).g < cost_{temp}$ **then**
      $cost(u_j).g \leftarrow cost_{temp}$
      Set parent of u_j to u_i
    end if
    $cost(u_j).h \leftarrow h_j$
    $cost(u_j).f \leftarrow cost(u_j).g + cost(u_j).h$
  end for
  Remove u_i from $OPEN$ and Add it to $CLOSED$
end while

- The path planning begins from $Node_{begin}$ being added to the $OPEN$ list and it's f, g, h, t costs being initialized.
- For each iteration, a node u_i with the minimum f cost from the $OPEN$ list is chosen and it's neighbours are evaluated for path costs.
- Each neighbour favours minimum cost with which a weather forecast is chosen for planning, $cost_{obst}, cost_{corr}, cost_{twist}$ and constraint limits are added

- $cost_{temp}$ is calculated by assigning weights ζ_c and ζ_t for $cost_{corr}, cost_{twist}$ respectively. These weights determine the contribution of corridor and twist costs while planning. The less weight a particular cost is given, the less it's contribution is and vice versa. A study showing the effect of these weights on path planning is shown in the test-cases section
- Minimum g cost is favoured for each neighbouring node $u_j \in N_i$ and corresponding h and f costs are calculated
- Node u_i is then $CLOSED$
- The actual path from $Node_{begin}$ to u_{target} can be tracked in reverse by looking at each node's parent
- For each time the Waypoint Control algorithm calls A*, such path is calculated and the child node of $Node_{begin}$ is returned as $WP_{planned}$

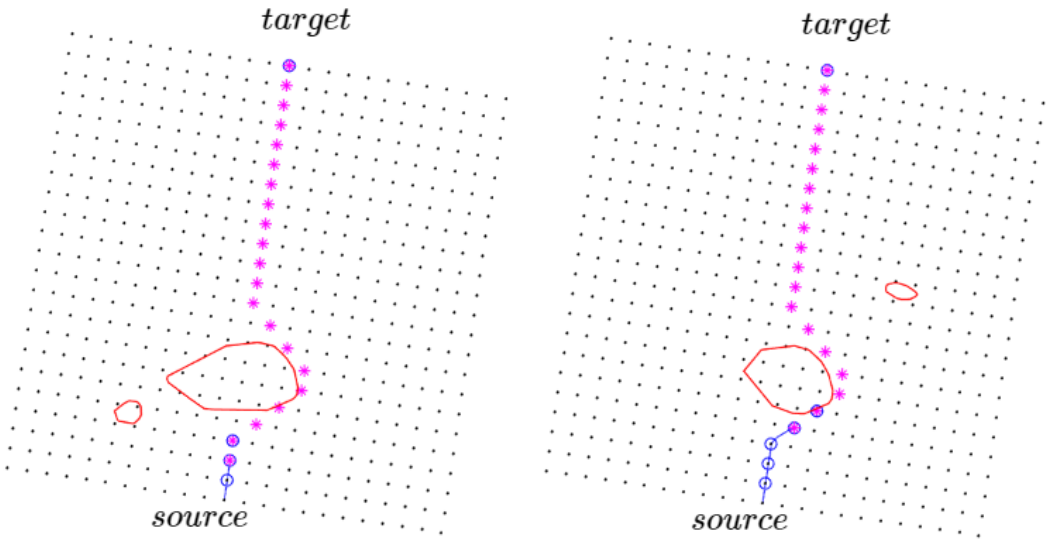


Figure 28: A* algorithm path planning

The A* algorithm when used with the Waypoint Control algorithm 6 looks like he figure 28. One can see the square grid of nodes and the path planned for each call of A* algorithm as well as the planned waypoint returned by the algorithm. This algorithm is ready to address data disappearance due to radar limitations, movement of weather, twist and corridor optimality, standardization procedures while always being complete and optimal.

3.3 RRT-A* Path Planning Algorithm

It is known that the RRT* algorithm searches the configuration space C to find the best path from source to target nodes. Let's remind again that C_{obs} is the obstacle filled region and the free configuration space $C_{free} = C \backslash C_{obs}$ and at every iteration of the algorithm, a new node $u_{rand} \in C_{free}$ is sampled. The RRT* algorithm guarantees asymptotic convergence, but since the project is about *online* path planning, the optimal path is supposed to be calculated in finite time, moreover it is supposed to be deterministic, following all the standards that have been described before.

There have been several improvements in the Random Forest algorithms which focus mainly on the sampling algorithm for faster convergence. In the method proposed here, the sampling algorithm has been modified such that there is no more randomness involved. What it means that the u_{rand} which is sampled is deterministic and not random anymore, thus making the algorithm faster to converge and deterministic.

The search for optimal path is done by combining the elements of RRT* and A* algorithms together. The main aspect of the proposed algorithm is that it does not have to maintain a set of grid points as opposed to A*. The algorithm also guarantees returning an optimal path if one exists since the search algorithm in the same as that of A*. It would be seen later how easy is it to implement the obstacle detection, corridor cost, twist costs, time dependence involved for path planning in this algorithm since it generates nodes which create a similar mesh structure as A*.

3.3.1 Sampling

The main cause of randomness in the Random Forest algorithms is the randomness of sampling. It makes sense for the sampling to be random since that is the basic differentiating factor as the name suggests. But since we have talked about the standards and foundation rules in the previous sections, it gives enough room to make intelligent sampling decisions.

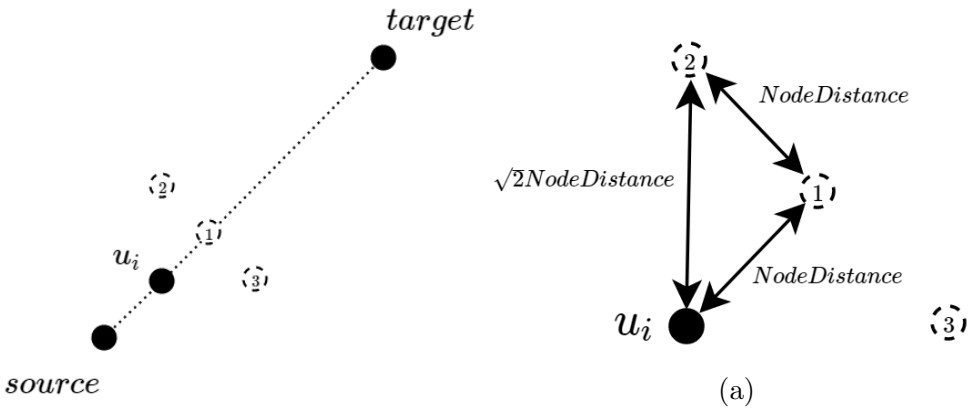


Figure 29: Standardized node sampling[†]

It is known that the A* grid is square, i.e. a node has maximum of eight neighbours. Having a glance at the standardization rules, the aircraft is allowed to take $0, \pm 45°$ manoeuvres w.r.t original flight path, thus it makes sense to sample new nodes befitting these rules. Since the search algorithm is same as A*, consider the node picked by the algorithm to be u_i. Now, in the algorithm, we do not sample only one node u_{rand}, but rather three nodes and let $U_{neighbours,i}$ be a set of neighbouring nodes sampled. As shown in the figure 29(a), the new nodes sampled always lie at $0, \pm 45°$ to the v_{ts} vector.

While remembering A*, it can be seen that the sampling is analogous to evaluating three neighbouring nodes. The three nodes coordinates can be obtained in the same way the coordinate matric is generated in A* as

- Calculate θ_x, the inclination of $v_{ts} = target - source$ vector w.r.t positive x axis
- Spawn three nodes at $y = NodeDistance$ and x=0,-$NodeDistance$,$NodeDistance$ (node 1,2,3 respectively)
- Rotate every node by $\theta \leftarrow \theta_x - \frac{\pi}{2}$ along the origin and then translate to the current node u_i

Since the $NodeDistance$ has already been calculated in A* by equation 11, this type of sampling addresses the issues of weather data loss due to sparse sampling, corridor violation, huge computations due to dense sampling as discussed previously.

3.3.2 Search

The RRT-A* algorithm does not have to maintain a grid unlike A*. There is no connection-matrix which stores the neighbours of a particular node. Instead, the neighbours $\in U_{neighbours}$ are directly sampled and a $NodeList$ is maintained.

- It has been mentioned that the path search methodology of this algorithm is similar to A*, therefore each node has a f, g, h cost to be maintained and the node picked up for evaluation u_i every iteration is the node present in $OPEN$ list with the minimum f cost
- After the node u_i is picked, three neighbours $\in U_{neighbours,i}$ are sampled. It might happen that one or many of these sampled nodes have already been sampled in the previous iterations, thus the $NodeList$ is checked for duplicate nodes

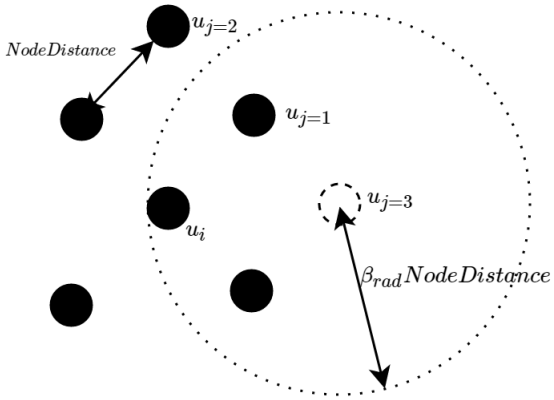


Figure 30: Node neighbourhood

- *Neighbourhood search* is conducted around the node u_j with a radius of $\beta_{rad}NodeDistance$. As shown in the above figure for a certain i^{th} iteration, for $u_{j=3}$, the farthest neighbour u_i lies at a distance of $\sqrt{2}NodeDistance$.
- Thus to avoid computations over all $OPEN$ list, only the nodes lying in the neighbourhood are evaluated (figure 30). Thus β_{rad} can take a value of $(1+\epsilon)\sqrt{2}$ where ϵ lies in the range of $0 < \epsilon \leq 0.1$
- The cost of travelling to u_j via each node in it's neighbourhood and $OPEN$ list is calculated. The minimum cost is retained while assigning the particular node as the parent of u_j
- In this algorithm, the costs namely $cost_{obst}, cost_{corr}$ are evaluated in the same way as in A*. $cost_{twist}$ cannot be calculated directly since there is no connection matrix maintained.

3.3.3 Twists

Calculating twist costs for this algorithm has the same basics as that for A*, but since the connection matrix is not available, a different approach is used.

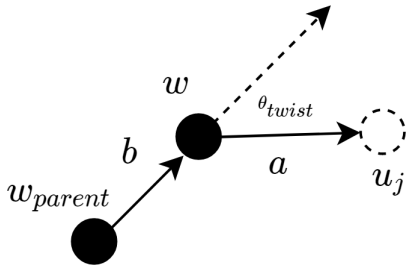


Figure 31: Twist for RRT-A* †

Given the newly spawned node u_j and the neighbourhood node from $OPEN$ list w. If w_{parent} is the parent node of w, the two vectors a, b can be given by $u_j - w$ and $w - w_{parent}$. Simple arccos would give the angle between the two vectors and the twist cost $cost_{twist}$ is given by

$$\theta_{twist} \leftarrow abs\left(\cos^{-1}\left(\frac{a \cdot b}{||a||||b||}\right)\right)$$
$$cost_{twist} \leftarrow \sin(\theta_{twist}\ \frac{\pi}{8})$$

- (.) is a vector *dot product* while ||.|| is the usual *Euclidean norm*. Since the nodes are sampled in such a way that they would create a square mesh grid, it can be assured that the twist angle θ_{twist} would always take discrete values among $0°, 45°, 90°, 135°$ which would give the same twist cost as discussed before for A*

3.3.4 Time Dependence

The time dependence of RRT-A* is same as that for A*. The reason for not assigning time costs according to columns in A* is clear here. Since in RRT-A*, there exists no connection matrix and therefore assigning costs to nodes as per the grid columns would not have worked. In RRT-A*, the *time-cost* principle works with the same principle of favouring the lowest time cost. The weather forecast can be chosen accordingly using the 'Choose Forecast' function.

3.3.5 Full RRT-A* Algorithm

Summing up the thoughts for RRT-A* together, the pseudocode for the algorithm can be given as:

Algorithm 14 RRT-A*

Require: $WP_{active}, u_{source}, u_{target}$, Forecasts, $N_{itr}, Node_{prev}$,
Initialize $CLOSED, OPEN, NodeList, cost$
$Node_{begin} \leftarrow WP_{active}$
Add $Node_{begin}$ to $OPEN$ list
Initialize f, g, h, t cost of $Node_{begin}$
for $i = 1 : N_{iter}$ **do**
  pick node $u_i \leftarrow min(cost(OPEN).f)$
  if $u_i \in target$ **then**
    break
  end if
  $U_{neighbours} \leftarrow$ Sample Neighbours(u_i)
  for each neighbour $u_j \in U_{neighbours,i}$ not in $NodeList$ **do**
    Initialize f, g, h, t costs of u_j
    for each node w in $OPEN$ list **do**
      $rad \leftarrow c(u_j, w)$
      if $rad \leq \beta_{rad} NodeDistance$ **then**
        $cost(u_j).t = min(cost(u_j).t, cost(w).t + \Delta T)$
        $Weather \leftarrow$ Choose Weather$(cost(u_j).t,$ Forecasts$)$
        $cost_{obst} \leftarrow$Calculate Obstacle Cost
        $cost_{corr} \leftarrow$Calculate Corridor Cost
        $cost_{twist} \leftarrow$Calculate Twist Cost
        $cost_{obst} \leftarrow$Calculate Constraint Limit
        $cost_{temp} \leftarrow cost_{obst}(1 + \zeta_c cost_{corr} + \zeta_t cost_{twist})c(u_j, w) + cost(w).g$
        if $cost_{temp} < cost(u_j).g$ **then**
          $cost(u_j).g \leftarrow cost_{temp}$
          Set parent of u_j to w
        end if
        $cost(u_j).h \leftarrow h_j$
        $cost(u_j).f \leftarrow cost(u_j).g + cost(u_j).h$
      end if
    end for
    Add u_j to $OPEN$ list
  end for
  Add u_i to $CLOSED$ and Remove it from $OPEN$ list
end for

- The algorithm starts planning from $node_{begin}$ by initializing it's f, g, h, t costs and adding it to $OPEN$ list

- For each iteration, since the search is same as A* the node with minimum f cost is picked from the $OPEN$ list and a set of neighbouring nodes is sampled
- Duplicate nodes are ignored and the f, g, h, t costs are initialized for each sampled node u_j and a neighbourhood search is carried over the $OPEN$ list
- All the required costs are calculated while favouring minimum *time-cost* and choosing suitable forecast. Weight terms are added to *twist* and *corridor* costs
- u_j is added to $OPEN$ and u_i is then $CLOSED$

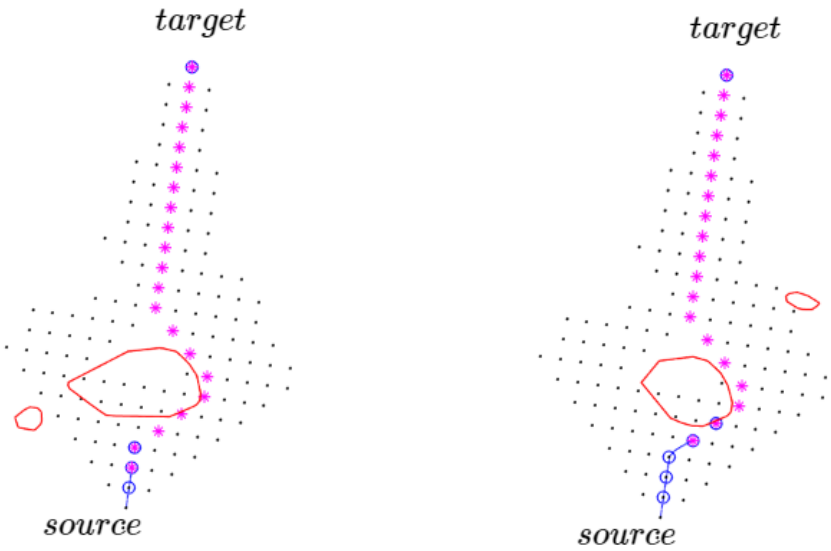


Figure 32: RRT-A* algorithm path planning

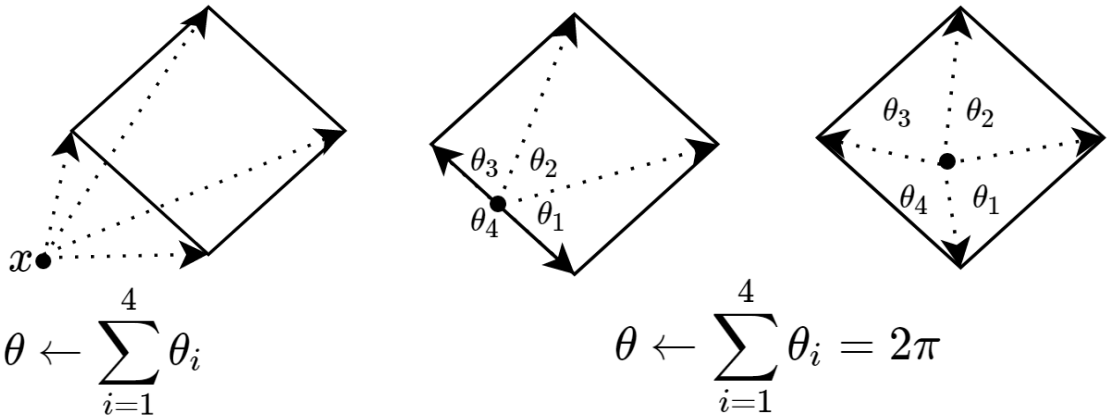
The RRT-A* algorithm when coupled with the Waypoint Control algorithm 6 looks like the figure 32. One can see all the nodes sampled to create a search tree. Similar to A*, the planned path can be traced by reverse search and the planned waypoint which is the child of WP_{begin} is returned for each call by WPC algorithm. This algorithm being almost same as A* w.r.t search criteria, *NodeDistance*, time-dependence always produces exactly the same results like A*. A time-complexity comparison study is done later in the test-cases section.

3.4 Non-Linear Programming

The node-based and sampling-based methods being discrete optimization methods are faster but always produce results according to the decomposition of the environment. Whereas MILP and Optimal control methods have been widely used for path planning, another approach with non-linear gradient based optimization algorithm has been proposed to generate optimal path. Ingersoll et al. [50] proposed UAV path planning with Bezier curves and receding horizon approach using non-linear programming which forms the basis of the method proposed in this project. Although they used simple circular obstacles and non-standardized motion which contradicts our requirements, nevertheless it was a good attempt to motivate us to use NLP in this project.

3.4.1 Obstacle model

Most of the published research [19, 50] model obstacles in terms of constraints. One of the methods[†] to model convex obstacle constraints is:



(a) x outside polygon

(b) x on or inside polygon

Figure 33: Obstacles as constraints

Consider a convex polygon with four nodes and $x \in \mathbb{R}^2$ be the optimization variable representing position. Let the angle subtended by x with i^{th} edge be θ_i. It is seen that in figure 33(a), when x lies outside the polygon, the sum of all the subtended angles $\theta \leftarrow \sum_{i=1}^{4}$ is less than 2π. In the 33(b) as x approached to the edge of the polygon θ becomes 2π and remains the same when x is anywhere inside the polygon. For a j^{th} polygon with $Nobst_j$ number of nodes, this property can be formulated as an inequality constraint. So for x to remain outside the j^{th} polygon in a scenario with N_{polys} number of total polygons

$$\sum_{i=1}^{Nobst_j} \theta_{ij} < 2\pi \;\; j = 1, 2....N_{polys}$$

The subtended angles θ_{ij} can be calculated by simple cosine law. The problem with this formulation is the optimization problem becomes highly constrained. This approach leads to a total of N_{polys} number constraints and $\sum_{j=1}^{N_{polys}} Nobst_j$ number of inverse cosine evaluations for the constraint calculations.
Our problem has an advantage that the obstacles aren't rigid and there could be a way to exploit that property. To reduce the complexity, the obstacles can be viewed as mountains which the optimizer would find hard to climb. One inspiration[†] to model such mountains comes from Multivariate Normal distribution.

Bivariate Normal Distribution is a generalization of the one-dimensional normal distribution to two dimensions. A bivariate normal distribution can be characterized by it's mean μ and covariance $\sum_{cov}$, a semi definite matrix. A bivariate distribution has two random variables since $x \in \mathbb{R}^2$ and the probability distribution is given as

$$\phi(x) = \frac{1}{\sqrt{2\pi|\sum_{cov}|}} exp\left\{-\frac{1}{2}(x-\mu)^T \sum\nolimits_{cov}^{-1} (x-\mu)\right\} \tag{13}$$

Where $|\sum_{cov}|$ is the determinant and $\sum_{cov}^{-1}$ is the inverse of covariance matrix $\sum_{cov}$. For a given dataset of vectors, to calculate it's probability distribution it is necessary to estimate the mean and covariance matrix.

Since the weather polygons are defined a set of nodes, these node data can be used to calculate the mean and covariance. For simplicity, node coordinates of the polygons are considered independent from each other and thus, For an i^{th} weather polygon defined by $Nobst_i$ nodes in a scenario with $Npolys$ number of polygons, let $X_{i,j} = [poly_i.lon(j), poly_i.lat(j)]$ denote the j^{th} node coordinates. The mean and variance can be calculated as

$$\begin{aligned} \mu_i &= \frac{\sum_{j=1}^{Nobst_i} X_{i,j}}{Nobst_i} \\ \sum_{cov,i} &= \frac{1}{Nobst_i - 1} \sum_{j=1}^{Nobst_i} (X_{i,j} - \mu_i)^T (X_{i,j} - \mu_i) \end{aligned} \tag{14}$$

A bivariate normal distribution or a Gaussian distribution function is also a continuous probability distribution function. Thus the integral of the function w.r.t the coordinate axes is always 1. This means that at any particular coordinate, the function value is always less than 1. To increase the magnitude of our obstacle polygons, the normalization term of the distribution function is excluded. Thus, for the i^{th} polygon, the distribution function would look like

$$DF_i(X) = e^{-\frac{1}{2}(X-\mu_i)^T \sum_{cov,i}^{-1} (X-\mu_i)}$$

and for a scenario with N_{polys} number of polygons, each polygon would contribute to the function. Thus the mathematical model for a full scenario would be a superposition of distribution functions $DF_i(X)$ for all the N_{polys} polygons.

3.4.2 Objective Function

The basic aim of any optimization problem is to minimize the Objective function while considering the constraints. In this application, the algorithm is required to calculate an optimal path while avoiding weather and obeying the rules of airspace as mentioned above.

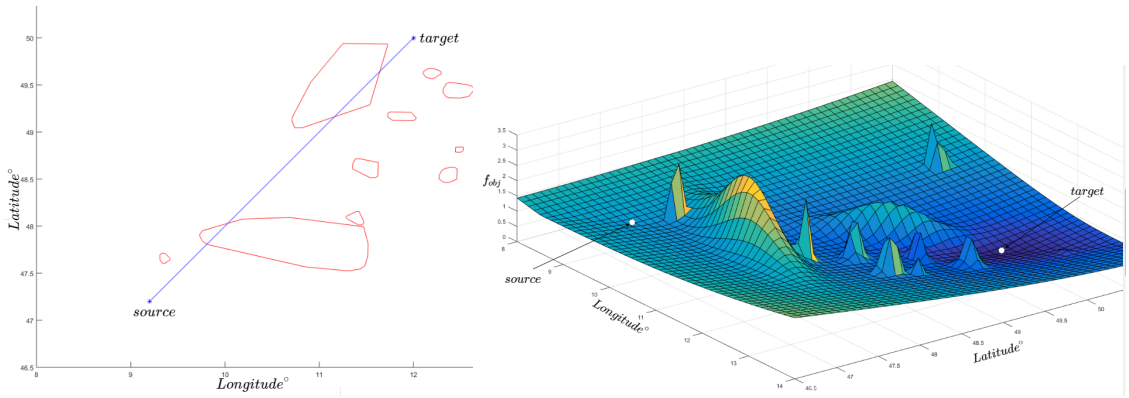
The airspace corridor violation cost is included in the objective function itself. The perpendicular distance of the point $[x_0, y_0]$ from a line segment joining the two points $[x_1, y_1]$ and $[x_2, y_2]$ is given in the equation 12. Thus, to reduce the airspace corridor violation, it is necessary to minimize this distance, let's call it d_{pl} to symbolize the distance of a point x from the segment joining $source$ and $target$.

Considering the mathematical model for obstacles, corridor cost and the fact that the RPA is supposed to eventually move towards the $target$, the objective function could be given as

Algorithm 15 Objective

Require: $x, target, source, \mu$ and $\sum_{cov}$ for N_{polys} polygons
$d_{pl} \leftarrow$ Perpendicular distance between x and $target - source$ line segment
$d \leftarrow ||target - x||$
$d_{ts} \leftarrow ||v_{ts}||$
for $i = 1 : N_{polys}$ **do**
  $f \leftarrow f + DF_i(x)$
end for
$obj \leftarrow \frac{\zeta_c d_{pl} + d}{||v_{ts}||}(1 + f)^2$

- As seen in the algorithm 15, the final objective function obj involves d_{pl} term to reduce corridor violations, d as in the distance between optimum point x to $target$. The sum of these distances is normalized by the total $target - source$ distance
- ζ_c is the corridor weight which addresses the contribution of d_{pl} in the objective function
- the superposition of all the polygon values is given by f and is squared after adding 1 otherwise obj would hold a value close to zero at places with no polygons. This also make the polygons to be avoided by the path planner appear more distinct



(a) Weather scenario (b) Objective function

Figure 34: Objective function with obstacles model

- The figure 34(a) shows the scenario with given weather polygons, the *source* and *target*. The 34(b) shows the objective function *obj*

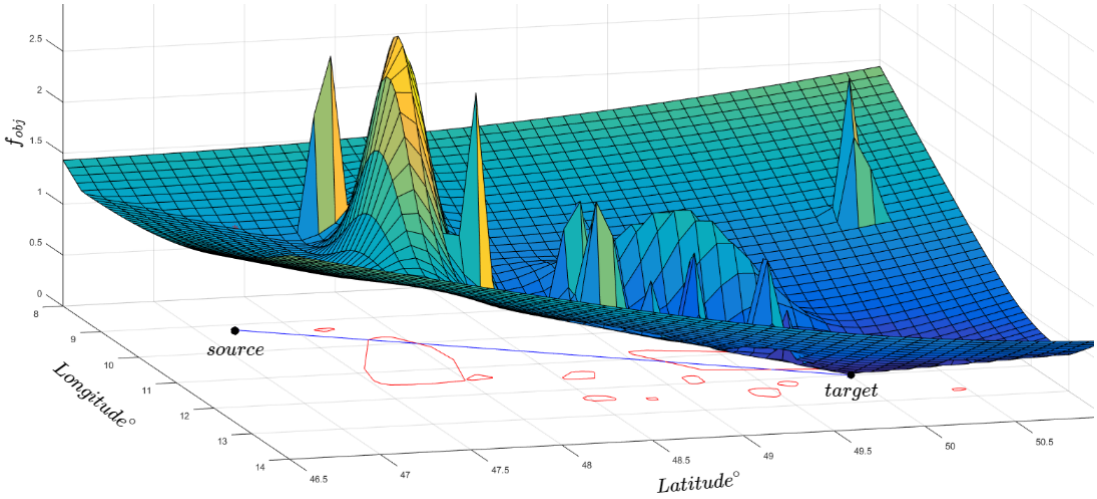


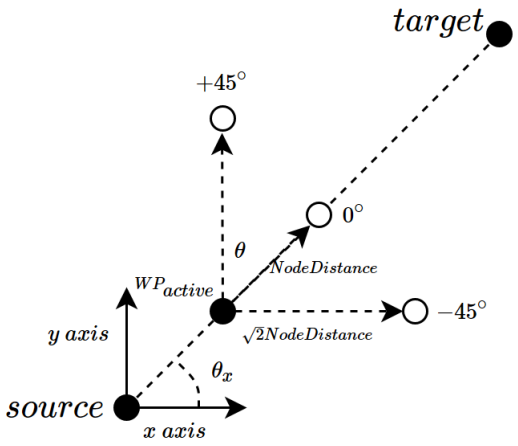
Figure 35: Objective function visualization

- As seen in the figure 35, the objective function *obj* is relatively high at *source* and is zero at the *target*. The objective function also has a higher value for each polygon, now the goal of the optimizer is clearly defined which is to generate a path starting from *source* till *target* while always reducing the objective function. It is similar to finding a path for climbing down from a mountain while avoiding climbing up any intermediate hills

3.4.3 Constraints

Till now, we have taken the obstacles, corridor violation and heading towards *target* into consideration. This would have been sufficient if the goal of our problem did not include the standards that we fixed. The aspects still to be considered are standardized deviation of $0°, \pm 45°$, reducing the twist in the generated path, fixing the step-size between two Waypoints to either $NodeDistance$ or $\sqrt{2}NodeDistance$, addressing the radar data disappearance and time dependence.

Deviation: Since the aircraft has to conduct deviations at standardized angles, it's necessary to apply a few constraints which address this aspect. Consider the active waypoint WP_{active} and the optimization variable x which determines the geographical coordinates of the planned waypoint should always make an angle of either $0°, +45°, -45°$ with the v_{ts} vector. When it makes $0°$, the WP_{active} and x should be separated by $NodeDistance$ whereas if it makes $\pm 45°$, the separation should be $\sqrt{2}NodeDistance$.



The angles between vectors could be calculated by the cosine law, but it does not distinguish between $+45°$ and $-45°$ since cosine is an even function. Thus to address this issue, the *multi-valued inverse tangent* function $atan2$ is used. Always remember that any angles calculated lie in the domain $[0, 2\pi)$ and is measured counter-clockwise. Let θ_x be the inclination of the v_{ts} vector w.r.t positive x *axis* and let θ be the inclination of $x - WP_{active}$ vector w.r.t positive x *axis*. Then the standardization procedure can be formulated into constraints as follows

- In case of $+45°$ inclination

$$\begin{aligned} \theta - \theta_x &= \frac{\pi}{4} \\ ||x - WP_{active}|| &= \sqrt{2}NodeDistance \end{aligned} \tag{15}$$

- In case of $0°$ inclination

$$\begin{aligned} \theta - \theta_x &= 0 \\ ||x - WP_{active}|| &= NodeDistance \end{aligned} \tag{16}$$

- In case of $-45°$ inclination

$$\theta - \theta_x = -\frac{\pi}{4}$$
$$||x - WP_{active}|| = \sqrt{2}NodeDistance \tag{17}$$

Waypoint separation: The above constraints were applied to calculate the next direct waypoint to be planned $WP_{planned}$. But it has to be considered that the position of $WP_{planned}$ not only depends upon a small neighbourhood that it lies in i.e. using only local minima is insufficient. Rather, for path planning, the future expected scenario has to be considered. The graph based and sampling based algorithms have the capability to search throughout the environment without much effort. But since solving an optimization problem with gradient based iterative methods is the foundation of mathematical model based methods, it may involve big computations for a rather simple task.

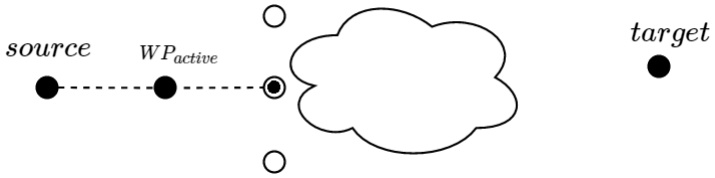


Figure 36: Local planning

In the figure 36, the $WP_{planned}$ has to be the one with $0°$ deviation since the optimization problem obtained the minimum value for that particular node. But as it can be seen that it would be a rather bad choice since to avoid the weather in a standardized manner, the aircraft has to pass through the cloud under any circumstances.

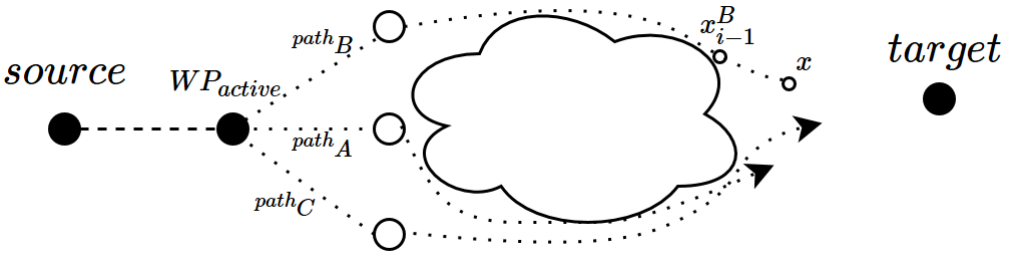


Figure 37: Extended path planning

To take into consideration the whole scenario, it makes sense to extend the paths for the three choices into $path_A, path_B$ and $path_C$ as shown in the figure 37 and then make an informed decision about which path would be the best. Let the j^{th} path be constructed by $N_{ext}+1$ number of nodes $x_i^j,\ i = 1..N_{ext}+1,\ j = A, B, C$. Now, each path begins from WP_{active} and ends at the node $x_{N_{ext}+1}^j$ with x_1^j being $WP_{planned}$ and so on.

While calculating planning for the x_i^j node, the separation between it's previous node x_{i-1}^j is not required to be defined by a strict equality constraint as before. Rather one can define the separation range

$$\begin{aligned} ||x - x_{i-1}^j|| &\leq lim_{max} NodeDistance \\ ||x - x_{i-1}^j|| &\geq lim_{min} NodeDistance \end{aligned} \tag{18}$$

Where x is the optimization variable for x_i^j. Since in any case, the distance between two nodes is either $NodeDistance$ or $\sqrt{2}NodeDistance$, the minimum and maximum limits can be considered to be

$$NodeDistance \leq ||x - x_{i-1}^j|| \leq \sqrt{2}NodeDistance$$

A study about the time complexity v.s. N_{ext} has been conducted which is shown later in this report.

3.4.4 Radar Disappearance

In this method, the effect of radar disappearance is addressed automatically due to the standardization constraints mentioned before.

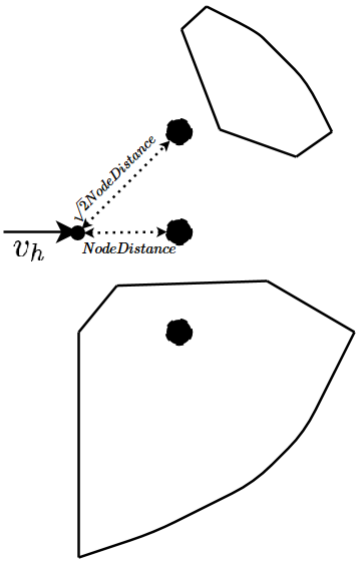


Figure 38: Radar disappearance for NLP

as shown in the figure 38, due to the separation constraints in equations 15,16 and 17 the planned waypoint $WP_{planned}$ will never lie in the *non-feasible* half plane. Therefore there is no need to add an explicit constraint limit for radar disappearance problem,

rather to make sure that the extended path always heads towards the *target* another simple constraint is added

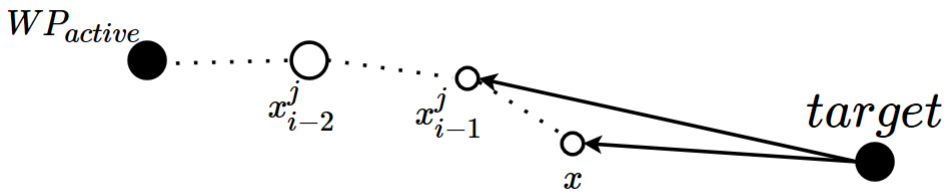


Figure 39: Extended path heading

As shown in the figure 39, for the path to head towards the *target* it is necessary for the distance $||x - target||$ to be less than $||x^j_{i-1} - target||$ which can be expressed by the following constraint

$$||x - target|| - ||x^j_{i-1} - target|| \leq 0 \tag{19}$$

The equation 18 makes it sure that x and x^j_{i-1} do not overlap.

3.4.5 Path Evaluation

So, using the approach to create three different paths for each possible deviation, a criterion for path selection has to be decided. A good approach is to integrate the objective function along the developed path and choose a path which gives an over all minimum value.

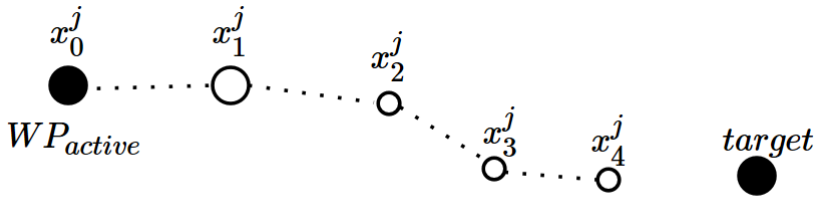


Figure 40: Extended path evaluation

Consider a path generated from WP_{active} with $N_{ext} = 3$ This leads to $N_{ext} + 1$ number of path segments in total. For the j^{th} path, consider WP_{active} to be x^j_0 and the path extends till the node $x^j_{N_{ext}+1}$. The objective function could be integrated along the extended path j and the path leading to the minimum value could be chosen. The simplest method of integrating the objective function along a node segment $x^j_{i-1} \rightarrow x^j_i$ is to consider the average of the objective function at both the nodes times the euclidean

distance between the two nodes. Thus, the function $f(j)$ to be evaluated would look like

$$f(j) = \frac{1}{2}\sum_{i=0}^{N_{ext}}(obj(x_i^j) + obj(x_{i+1}^j))||x_{i+1} - x_i|| \tag{20}$$

Twist: It was easier to apply twist cost along the full path in the previous two methods. Since in this method, the full path may or might not be evaluated in a standardized manner, a different approach is used to include twist costs.

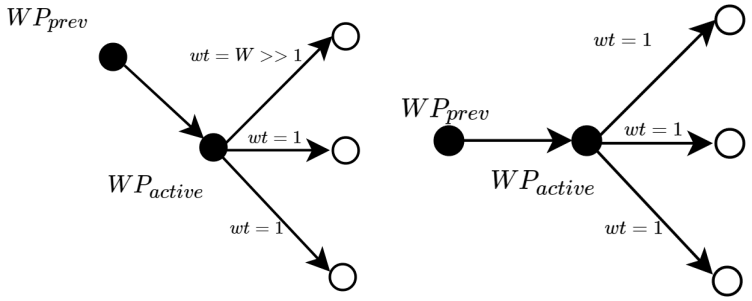


Figure 41: Twist cost for NLP†

While calculating the function $f(j)$ along the path j, a weight wt is added to each segment. Consider the figure 41 in which the planned waypoint which gives the maximum twist is given a weight $W >> 1$ so that it is inherently neglected whereas the other two waypoints are given a weight of 1.

3.4.6 Time Dependence

Since the gradient based optimization problem does not consider for decomposition of environment into samples or graph, the distance between consecutive nodes in a path may vary. According to the previous discussions, since it has been decided to only consider three forecasts, NLP has to be modified such that it accommodates the decision. Since for a path, the total number of nodes to be planned are $N_{ext} + 1$ where for every i^{th} node, $WP_{planned}$ is $i = 1^{st}$ node, the next node is $i = 2$ and so on. The forecast to be chosen for i^{th} waypoint can be given as

Algorithm 16 Choose Forecast NLP†

Require: i, Forecasts for $t + \Delta T, t + 2\Delta T, t + 3\Delta T$
$N_t \leftarrow min(3, i)$
$Weather \leftarrow Forecast$ for $t + N_t\Delta T$

- Thus from the algorithm 16, the $WP_{planned}$ is calculated for forecast at $t + \Delta T$, the successive node for for the forecast at $t + 2\Delta T$, and all the next nodes for $t + 3\Delta T$
- This can be done since the extended nodes are separated by a minimum of $NodeDistance$ and maximum of $\sqrt{2}NodeDistance$
- The solution provided is not as precise as an *optimal control* problem but considering the non rigid nature of obstacles, this simple approach gives satisfactory results

3.4.7 Optimization problem

Thus, summing up all the factors considered till now, the problem can be converted into a standard constrained non-linear optimization problem

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & obj(x) \\ \text{subject to} \quad & g_m(x) \leq 0, \ m = 1, ...N_{ineq} \\ & h_n(x) = 0, \ n = 1...N_{eq} \end{aligned}$$

While planning the $i = 1^{st}$ node i.e. the $WP_{planned}$, equality constraints for deviation are applied depending upon which path option is being evaluated
While planning the $i > 1^{th}$ node i.e. the extended nodes, the inequality constraints are applied independent of which path is being planned
For all the nodes and paths being planned, the inequality constraint for path heading is applied.
This can be summarized as for j^{th} path and i^{th} node being planned and considering x_0^j to be the WP_{active},

- Require i, j, x_{i-1}^j

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & obj(x) \\ \text{subject to} \quad & ||x - target|| - ||x_{i-1}^j - target|| \leq 0 \end{aligned}$$

- If $WP_{planned}$ is being planned i.e. $i == 1$
  - If $Path_A$ is being planned i.e. $j == 1$

$$\begin{aligned} \theta - \theta_x &= 0 \\ ||x - WP_{active}|| &= NodeDistance \end{aligned}$$

  - If $Path_B$ is being planned i.e. $j == 2$

$$\begin{aligned} \theta - \theta_x &= \frac{\pi}{4} \\ ||x - WP_{active}|| &= \sqrt{2}NodeDistance \end{aligned}$$

– If $Path_C$ is being planned i.e. $j == 3$

$$\theta - \theta_x = -\frac{\pi}{4}$$
$$||x - WP_{active}|| = \sqrt{2}NodeDistance$$

- Else if extended nodes are being planned i.e. $i > 1$

$$NodeDistance - ||x - x_{i-1}^j|| \leq 0$$
$$||x - x_{i-1}^j|| - \sqrt{2}NodeDistance \leq 0$$

Optimization Solver: The above optimization problem can be solved using the interior-point approach for solving constrained minimization problems [51]. For actual implementation, MATLAB's *fmincon(*)* solver is used for optimization.
Considering the Non-Linear optimization problem with N_{ineq} and N_{eq} number of inequality and equality constraints respectively,

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & obj(x) && (21)\\ \text{subject to} \quad & g(x) \leq 0 && (22)\\ & h(x) = 0 && (23) \end{aligned}$$

Where $obj : \mathbb{R}^2 \to \mathbb{R},\ g : \mathbb{R}^2 \to \mathbb{R}^{N_{ineq}},\ h : \mathbb{R}^2 \to \mathbb{R}^{N_{eq}}$
The inequality constraints can be converted to equality constraints by adding slack variables s. Thus the above problem can be reformulated as

$$\begin{aligned} \underset{x,s}{\text{minimize}} \quad & obj(x)\\ \text{subject to} \quad & g(x) + s = 0\\ & h(x) = 0\\ & s \geq 0 \end{aligned}$$

The above problem is converted into an approximate problem by adding logarithmic barrier terms thus resulting into solving

$$\begin{aligned} \underset{x,s}{\text{minimize}} \quad & obj(x) - \mu_p \sum_{i=1}^{N_{ineq}} ln(s_i)\\ \text{subject to} \quad & g(x) + s = 0\\ & h(x) = 0 \end{aligned}$$

where $\mu_p > 0$ is the penalty parameter and the vector $s = (s_1....s_{N_{ineq}})^T$ is considered positive. Denoting Lagrange multipliers for the above problem,

$$L(x, s, \lambda_h, \lambda_g) = obj(x) - \mu_p \sum_{i=1}^{N_{ineq}} ln(s_i) + \lambda_g^T (g(x) + s) + \lambda_h^T h(x) \tag{24}$$

Where $\lambda_g \in \mathbb{R}^{N_{ineq}}$ and $\lambda_h \in \mathbb{R}^{N_{eq}}$ are the Lagrange parameters corresponding to inequality and equality constraints respectively. The *Karush-Kuhn-Tucker* or simply KKT conditions which are analogous to the gradient being zero are given as

$$\begin{aligned} \nabla_x L(x, s, \lambda_h, \lambda_g) &= \nabla obj(x) + H(x)\lambda_h + G(x)\lambda_g = 0 \\ \nabla_s L(x, s, \lambda_h, \lambda_g) &= -\mu_p S^{-1} e + \lambda_g = 0 \\ h(x) &= 0 \\ g(x) + s &= 0 \end{aligned} \tag{25}$$

Where $H(x) = (\nabla h_1(x) .... \nabla h_{N_{eq}})$, $G(x) = (\nabla g_1(x) .... \nabla g_{N_{ineq}})$, $e = ones(N_{ineq}, 1)$ and $S = diag(s_i),\ i = 1...N_{ineq}$. Multiplying the second row by S, The above system can be solved by applying Newton's method to obtain the iteration

$$\begin{bmatrix} \nabla^2_{xx} L & 0 & H(x) & G(x) \\ 0 & \Lambda & 0 & S \\ H^T(x) & 0 & 0 & 0 \\ G^T(x) & I & 0 & 0 \end{bmatrix} \begin{bmatrix} d_x^k \\ d_s^k \\ d_{\lambda_h}^k \\ d_{\lambda_g}^k \end{bmatrix} = \begin{bmatrix} -\nabla obj(x) - H(x)\lambda_h - G(x)\lambda_g \\ \mu_p e - S\lambda_g \\ -h(x) \\ -g(x) - s \end{bmatrix}$$

Where $d_x^k, d_s^k, d_{\lambda_h}^k, d_{\lambda_g}^k$ are the step sizes for x, s, λ_h and λ_g for the k^{th} iteration respectively and $\Lambda = diag(\lambda_{g,1} .... \lambda_{g,N_{ineq}})$. To convert the above matrix into a symmetric form, multiply the second block of equation by S^{-1}

$$\begin{bmatrix} \nabla^2_{xx} L & 0 & H(x) & G(x) \\ 0 & S^{-1}\Lambda & 0 & I \\ H^T(x) & 0 & 0 & 0 \\ G^T(x) & I & 0 & 0 \end{bmatrix} \begin{bmatrix} d_x^k \\ d_s^k \\ d_{\lambda_h}^k \\ d_{\lambda_g^k} \end{bmatrix} = \begin{bmatrix} -\nabla obj(x) - H(x)\lambda_h - G(x)\lambda_g \\ \mu_p S^{-1} e - \lambda_g \\ -h(x) \\ -g(x) - s \end{bmatrix} \tag{26}$$

Thus the parameters are incremented by solving the above symmetric system of equations

$$\begin{aligned} x^{k+1} &= x^k + \alpha_k d_x^k \\ s^{k+1} &= s^k + \alpha_k d_s^k \\ \lambda_h^{k+1} &= \lambda_h^k + \alpha_k d_{\lambda_h}^k \\ \lambda_g^{k+1} &= \lambda_g^k + \alpha_k d_{\lambda_g}^k \end{aligned} \tag{27}$$

At each iteration, the algorithm decreases the merit function [52]

$$obj(x) - \mu_p \sum_{i=1}^{N_{ineq}} ln(s_i) + \nu||h(x), g(x) + s|| \tag{28}$$

The parameter ν may increase with each iteration in order to force the solution towards feasibility. If a step size α_k not reduce the merit function, the particular step is rejected and a shorter step is tried. The convergence is checked if the KKT conditions are satisfied with suitable tolerance as

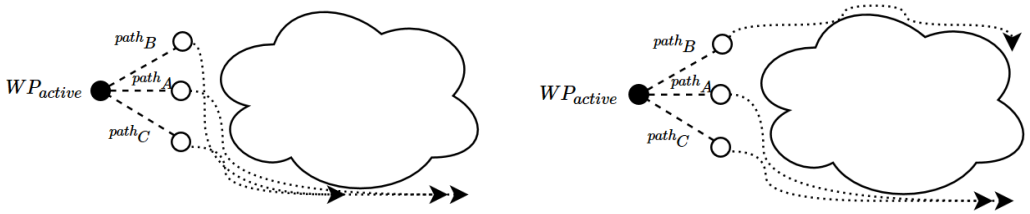
$$\begin{aligned} max|\nabla obj(x) + H(x)\lambda_h + G(x)\lambda_g| &\leq \epsilon_{tol} \\ max|-\mu_p S^{-1}e + \lambda_g| &\leq \epsilon_{tol} \\ max|h(x)| &\leq \epsilon_{tol} \\ max|g(x) + s| &\leq \epsilon_{tol} \end{aligned} \tag{29}$$

The full optimization algorithm can be given as follows

Algorithm 17 Dual Primal Barrier method

Require: $obj(x), h(x), g(x), x^0$
Initialize: $k \leftarrow 1, \lambda_h^0, \lambda_g^0, s^0, \mu_p > 0$
$error \leftarrow$ Check KKT convergence criterion
while $error > \epsilon_{tol}$ **do**
  Solve the system for $d_x^k, d_s^k, d_{\lambda_h}^k, d_{\lambda_g}^k$
  Calculate $x^{k+1}, s^{k+1}, \lambda_h^{k+1}, \lambda_g^{k+1}$ while determining α_k with Merit Function
end while

Initial Guess: As one is already familiar that gradient based optimization methods are sensitive to initial guess, therefore it is important to give an informed initial guess for solving the optimization problem.



(a) Poor initial guess (b) Improved initial guess

Figure 42: Initial guess

Since it has been decided to construct extended paths for each possible choice of deviation, it is important that these paths explore the environment as widely as possible.

As shown in the figure 42(a) since all the three extended paths were given the same initial conditions, the amount of environment explored is less and therefore all the three paths explored in the same direction. Whereas on the other hand, in 42(b), the three paths explore more environment due to different initial guesses thus giving the algorithm versatile path choices.
To achieve this, consider v_{ts}^{A} be the normalized $target - source$ vector. Two other vectors v_{ts}^{B},v_{ts}^{C} inclined at $\pm 45°$ to v_{ts}^{A} be calculated by rotation transformation respectively. Thus for planning any node x_i^j, the initial guess provided would be $x_{i-1}^j + v_{ts}^j NodeDistance$. This allows the algorithm for a wide spread exploration of the environment leading to a better path planning

3.4.8 Full NLP Algorithm

Summarizing the points discussed till now, the full algorithm for Non Linear Programming can be written as:

Algorithm 18 NLP

Require: $WP_{active}, WP_{prev}, u_{target}, u_{source}, Forecasts$
Initialize: $f \leftarrow \infty$
Calculate μ and $\sum_{cov}$ for each polygon in each forecast
for $j = 1 : 3$ **do**
  $x_0^j \leftarrow WP_{active}$
  for $i = 1 : N_{ext} + 1$ **do**
    $Weather \leftarrow$ Choose Forecast NLP
    if $||x_{i-1}^j - u_{target}|| < \epsilon$ **then**
      $x_i^j \leftarrow u_{target}$
      Calculate $f(j)wt$ w.r.t x_i^j and x_{i-1}^j
      break
    else
      Generate initial guess
      Generate objective function and constraints
      $x_i^j \leftarrow$ Solve respective optimization problem
      Calculate $f(j)wt$ w.r.t x_i^j and x_{i-1}^j
    end if
  end for
end for
Select appropriate Waypoint $WP_{planned}$ w.r.t $min(f)$

- The algorithm begins by calculating the estimation parameters μ and $\sum_{cov}$ for each polygon in all of the three forecasts and f is initialized to Infinity
- For each path planned, optimization problem is constructed for each node to be planned, initial guess is calculated

- The total path value for each j^{th} path $f(j)$ is calculated and the path resulting in the minimum value is selected. $WP_{planned}$ from this particular path is then returned at each call by the WPC algorithm

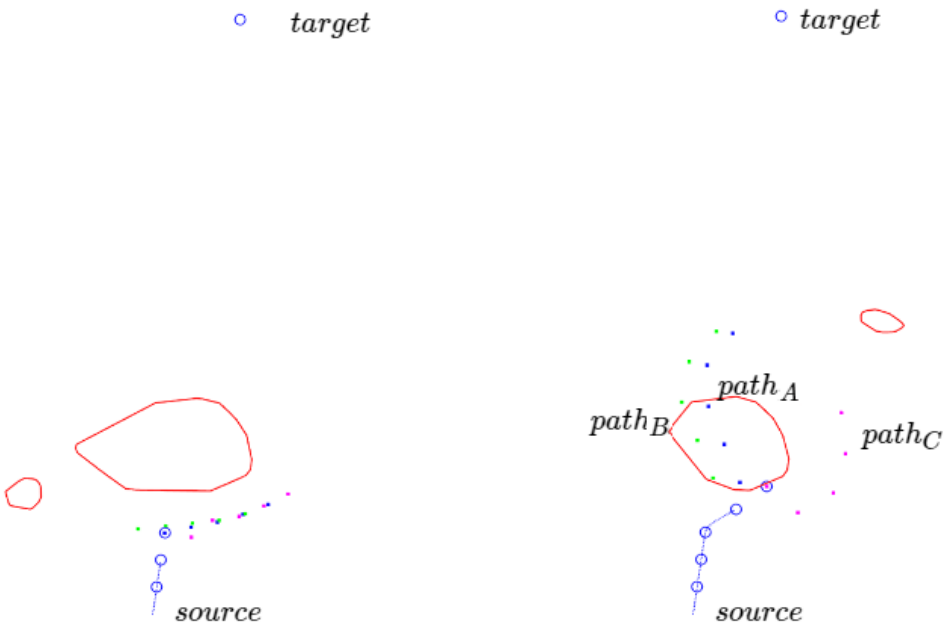


Figure 43: NLP algorithm path planning

The NLP algorithm when coupled with the Waypoint Control algorithm 6 looks like the figure 43. One can see the three paths generated for $N_{ext} = 4$. The three paths are compared in quality by equation 20 and a corresponding $WP_{planned}$ is returned to the WPC. Since NLP being a model based method, the optimizer takes the most time to solve the problem until it converges. This makes NLP much time consuming when compared to the previous two algorithms which is shown in a test case later.

3.5 Reinforcement Learning

Reinforcement Learning (RL) is a field of Machine learning inspired by behaviourist psychology. It is a method which determines how an agent would take an action based upon a positive or negative reinforcement received in terms of a reward. There are many different solutions to a RL problem, the widely used involve choosing an action which will maximize the reward in a long term and not only in the intermediate future. Let's start with some important definitions:

Agent: An agent in RL could be termed as the player. The agent is an object that takes actions and gets a reinforcement or feedback in return. The feedback recieved lets the agent learn whether the action it took was good or bad.

Environment: In RL, an environment is a playground in which an agent could take actions and expect a reward. The states of the agent is determined by the environment it's present in.

State: A state of an agent is the observed condition in which it is present. For example, a state of a car agent would be defined by it's velocity, acceleration and so on.
In RL, the agent does not have an idea which actions leads to win or lose but it explores the environment is a certain random manner by taking random actions. The agent will then remember these actions and the effect of these actions to help it take better actions in the future.

Markov Decision Process: simply a MDP provides a framework to model decision making where the outcome is the result of a combination of randomness and the actions of the agent. A MDP is defined by $\mathbb{S}, \mathbb{A}, \mathbb{P}, \mathbb{R}, \gamma$

$\mathbb{S}$: A set of all the possible observations which describe the state of the agent

$\mathbb{A}$: A set of actions from which the agent can choose one a to interact with the environment

$\mathbb{P}$: $P(\frac{s'}{s}, a)$ The transition probability matrix which describes how the environment changes when agent enters the state s' by performing action a in state s

$\mathbb{R}$: $P(\frac{r}{s}, a)$ Models the reward r given the current state s and the action a taken in the current state

γ : Also known as discount factor which takes a value in the range (0,1]. The discount factor is used to mathematically weigh the far future rewards. A discount factor of 1 would mean that all the immediate and far future rewards are given same importance whereas $0 < \gamma < 1$ means that the immediate rewards are given more importance than the far future rewards

Markov Property: Given the present state and action, the future state is independent of the previous states. This means that the current state is all the agent needs to decide the next state when an action is received. The goal of the agent is to solve the MDP by finding *Optimal policy* i.e. to find the sequence of actions to be taken in order to maximize the total reward.
Consider agent to be in state s_0 at time t_0, since the objective is to find a *policy* meaning a sequence of actions in time $t_0 \rightarrow t_n$ the MDP follows as following:

- At time t_0, the Agent observes the state s_0, takes an action a_0 and enters the state s_1. It recieves a reward r_1 depending upon the new state
- At time t_1, the agent observes the state s_1 and as before, takes an action a_1 to enter a state s_2 and recieves a reward r_2
- The steps keep on continuing until some stopping criterion or state s_{stop} is reached.

The total reward obtained through the process is given by

$$r_{total} = \sum_{i=1}^{stop} r_i$$

Considering the discounted rewards i.e. to weigh the rewards over future time, the total reward now can be given by

$$r_{total} = \sum_{i=1}^{stop} \gamma^{i-1} r_i$$

Value Function: A *value function* gives the idea of how good it is for an agent to be in a particular state. The value function $V(s)$ is equivalent to the expected total reward for an agent. If an agent uses a policy π to choose the actions, the value function can thus be given by

$$V^{\pi}(s) = \mathbb{E}[\sum_{i=1}^{stop} \gamma^{i-1} r_i] \quad \forall s \in \mathbb{S} \tag{30}$$

Thus, there exists an *optimal value function* $V^*(s)$ which has the maximum value than all the other existing value functions for all possible states and the policy corresponding to the optimal value function is called *optimal policy* π^*

$$V^*(s) = \max_{\pi}(V^{\pi}(s)) \qquad \forall s \in \mathbb{S} \tag{31}$$

$$\pi^*(s) = arg\ \max_{\pi}(V^{\pi}(s)) \qquad \forall s \in \mathbb{S} \tag{32}$$

Q Function: A state value function is a quantification of how good a state s is for the agent whereas the Q function quantifies how good it is for the agent to be in state s and choose an action a. The Q function is a function of state s and the corresponding action a which maps to a reward r

$$Q : \mathbb{S} \times \mathbb{A} \rightarrow \mathbb{R}$$

The optimal Q value $Q^*(s, a)$ implies the expected total reward for choosing action a while being in the state s. Thus logically, the optimal value function $V^*(s)$ should be maximum of $Q^*(s, a)$ for all possible actions and the corresponding policy is the optimal policy

$$V^*(s) = \max_{a} Q^*(s, a) \qquad \forall s \in \mathbb{S} \tag{33}$$

$$\pi^*(s) = arg\ \max_{a}(Q^*(s, a)) \qquad \forall s \in \mathbb{S} \tag{34}$$

Bellman Equation: The Bellman equation states that the expected value for being in a state s and choosing a policy π is given by sum of rewards obtained by choosing the policy π in the current state s and discounted expected value of being in the new state s'

$$V^{\pi}(s) = R(s, \pi) + \gamma \sum_{s'} P(s'|s, \pi) V^{\pi}(s') \tag{35}$$

Thus, the optimal policy is given by

$$V^*(s) = \max_{a} V^{\pi}(s)$$

$$V^*(s) = \max_{a}\{R(s, a) + \gamma \sum_{s'} P(s'|s, a) V^*(s')\} \tag{36}$$

Since by definition of optimal Q value, $Q^*(s, a)$ is the expected total reward while choosing action a,

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s'} (P(s'|s, a) V^*(s')) \tag{37}$$

Thus the equations [36] and [37] can be rewritten to get recursive equation as

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} (P(s'|s,a) \underset{a'}{max} Q^*(s',a')) \tag{38}$$

3.5.1 Q Learning

The cases where the state transition and reward models are unknown, rather only the set of possible states and actions are known, the agent has to learn actively through it's experience of interaction with the environment. In Q Learning, the agent is considered to have no idea about the state-transition and reward models. The agent learns good and bad actions over time by trial and error.

Temporal-Difference Learning: The idea behind Q Learning [53] is to approximate the state-action pairs from the samples of Q(s,a) which were observed during exploration. A table with for possible states-action pairs with Q values for each corresponding pair is created . The agent interacts with the environment by choosing actions a and getting rewards r for transitioning from state s to s'. The observed Q value $Q_{obs}(s,a)$ for taking action a in state s is calculated by the equation [38] where the transition model $P(s'|s,a)$ has a value 1 since it is unknown. The Q value $Q(s,a)$ is updated by $Q_{obs}(s,a)$ with a learning rate α. The pseudocode is given as follows:

Algorithm 19 Q Learning

Initialize Q(s,a) table ($\mathbb{S} \times \mathbb{A}$)
for $i = 1 : N_{train}$ **do**
  Initialize Agent state $s \leftarrow s_{init}$
  while s is Not Terminal condition **do**
    Select one action $a \in \mathbb{A}$ for the current state using exploration strategy
    Update the state s to next state s' by taking the action a
    Get reward $r \leftarrow R(s,a)$ for transiting from s with action a
    Get $Q(s',a')$ from the Q table
    $Q_{obs}(s,a) \leftarrow r + \gamma \underset{a'}{max} Q(s',a')$
    $Q(s,a) \leftarrow (1-\alpha) Q(s,a) + \alpha\ Q_{obs}(s,a)$
    $s \leftarrow s'$
  end while
end for

- The number of training iterations N_{train} is kept high so that the agent is exposed to as many possibilities of $s-a$ pairs as possible
- The $Q(s,a)$ is updated until all the Q values converge to a roughly constant value, the iteration can be stopped if the convergence is achieved first

- One of the major problems faced by Q learning is maintaining the state-action pairs table. As the size of possible states and actions increase, it becomes complicated to search through the table for a possible $s - a$ pair for updating the possible $Q(s, a)$ value
- This problem could be solved if there is a function which replicated the behaviour of the Q table. Defining such a function can be done by neural networks which are explained in the next section

3.5.2 Deep Learning

Deep Learning [54] is a part of the broader Machine Learning which has been gaining attention since the recent availability of BIG DATA and huge computing capabilities. Deep Learning involves a set of Neural Networks which roughly model the functioning of an actual brain.
The basic goal of a Deep Learning Neural Network is to approximate a function. Considering a function $y \leftarrow f(x)$ which maps the inputs x to a category y, a Deep Neural Network is a model $y \leftarrow f^*(x, \theta)$ which learns the best configuration of the parameters θ that results in the best approximation of the function $f(x)$.
A brain has a particular connection of neurons which are activated or 'fired' while taking a particular decision. Similarly, a Deep Learning model has a set of Neurons arranged in layers. Every neuron produces an output based upon the output from the previous layer and a set of weights.

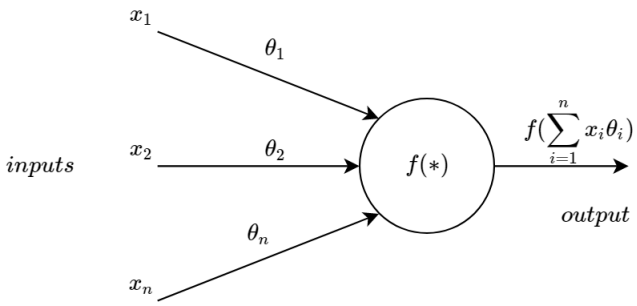


Figure 44: Neuron model

A single neuron model can be depicted in the figure 44 considering n inputs to the neuron with θ_i weight to the i^{th} input, the transfer function Z could be given as $Z = \sum_{i=1}^{n} x_i\theta_i$ and the output of the neuron holding a function $f()$ is $f(Z)$. The function which a neuron holds is called an activation function a. The activation function roughly models the firing phenomenon of a neuron and was initially a step function. But since a step function is discontinuous, several continuous versions of a step function are used, the most common one is the sigmoid function $\sigma(Z)$

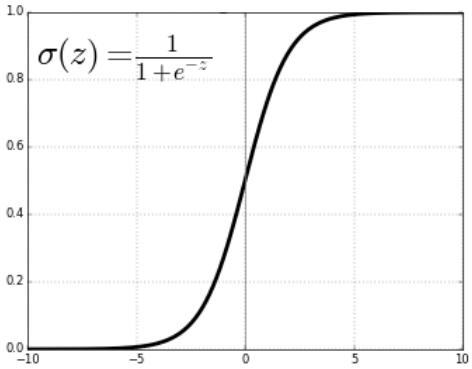


Figure 45: Sigmoid activation function

$$a = \sigma(Z) = \frac{1}{1 + e^{-Z}} \tag{39}$$

The sigmoid function is said to be activated when the transfer function reaches value above a certain threshold. The derivative of this activation function is given by

$$\sigma'(Z) = \frac{d}{dZ}(\frac{1}{1 + e^{-Z}}) = \sigma(Z)(1 - \sigma(Z)) \tag{40}$$

Such activation functions are used because it's easier to calculate the derivative $\sigma'(Z)$ once the function $\sigma(Z)$ is calculated.

Feedforward Multilayer Neural Network: An actual Deep Neural Network model is a combination of several such neuron models.

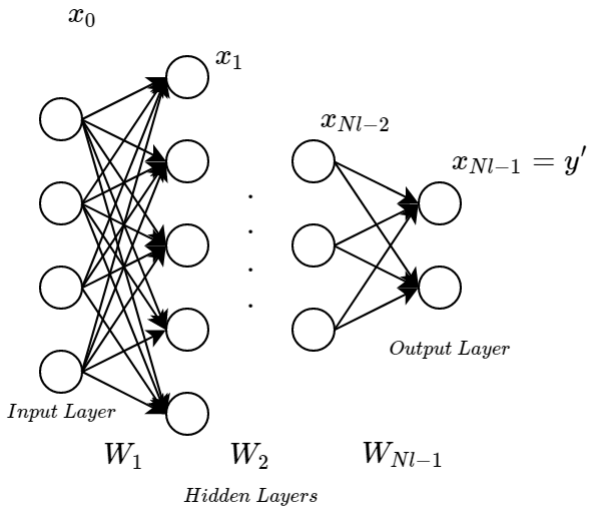


Figure 46: Deep Neural Network

As shown in the figure 46, there is an input layer x_0 with neurons equal to the size of input $n_0 \times 1$. All the neurons in the input layer are connected to each neuron in the immediate next layer x_1 called Hidden Layer. There can be any number of hidden layers [0,∞) depending upon the complexity of the network. All the neurons in the last hidden layer x_{Nl-2} are connected to each neuron in the output layer x_{Nl-1}.

Forward Propagation: Considering y' to be the actual output vector of a neural network, each layer connection can be defined with a weight matrix W_i which is the weights of the connection from $i-1^{th}$ layer to the i^{th} layer. Since all the neurons in i^{th} layer have the same activation function a_i the output x_i for any layer could be given as $x_i = a_i(W_i x_{i-1})$. The matrix W_i has the size $[n_i \times n_{i-1}]$.
Forward propagation is the process in which a neural network is fed with input vector x_0 to get the output vector y' or x_{Nl-1} which is a sequence of operations

$$\begin{aligned} x_1 &= a_1(W_1 x_0) \\ x_2 &= a_2(W_2 x_1) \\ &\vdots \\ y' = x_{Nl-1} &= a_{Nl-1}(W_{Nl-1} x_{Nl-2}) \end{aligned}$$

Backpropagation Algorithm: The gist behind the 'Learning' part is adjusting these individual neuron to neuron connection weights so that the output matches the expected output. Backpropagation is used to reduce the error contribution of individual neuron for a given input-output $x_0 \rightarrow y$ dataset.

Loss Function: Also known as the *Error Function* or *Cost Function*, it calculates the difference between the obtained network output and the expected network output. Let y be the network expected output vector. The error function $E(y, y')$ measuring the difference between the obtained and the expected values or simply the loss could be given as the *Euclidean Distance* between the two vectors. Thus the Loss function and it's gradient could be given as

$$\begin{aligned} E(y, y') &= \frac{1}{2}||y - y'||^2 \\ \frac{\partial E}{\partial y'} &= y' - y \end{aligned}$$

To update the neuron-neuron weights is indeed to update each of the individual matrices $W_1 ... W_{Nl-1}$ i.e. for all the weights w

$$w \leftarrow w - \alpha_w \frac{\partial E}{\partial w}$$

Here, α_w is the step size for the update also known as the *learning rate*. This relates to *GradientDescent* algorithm in which condition variables are updated by subtracting the gradient of the objective function w.r.t condition variables times a step size α. The main task is thus to calculate the partial derivative of error E w.r.t individual weights, which can be done by applying chain rule. Calculating the error derivative w.r.t the last weight matrix W_{Nl-1},

$$\begin{aligned}
\frac{\partial E}{\partial W_{Nl-1}} &= \frac{\partial E}{\partial x_{Nl-1}} \frac{\partial x_{Nl-1}}{\partial W_{Nl-1}} \\
&= \frac{\partial E}{\partial y'} \frac{\partial a_{Nl-1}(W_{Nl-1} x_{Nl-2})}{\partial W_{Nl-1}} \\
&= [(y' - y) \circ a'_{Nl-1}(W_{Nl-1} x_{Nl-2})] \frac{\partial W_{Nl-1} x_{Nl-2}}{\partial W_{Nl-1}} \\
&= [(y' - y) \circ a'_{Nl-1}(W_{Nl-1} x_{Nl-2})] x_{Nl-2}^T \\
\text{Let } \delta_{Nl-1} &= (y' - y) \circ a'_{Nl-1}(W_{Nl-1} x_{Nl-2}) \\
\frac{\partial E}{\partial W_{Nl-1}} &= \delta_{Nl-1} x_{Nl-2}^T
\end{aligned} \tag{41}$$

Here $\circ$ is the Hadamard product. For the second last weight matrix W_{Nl-2},

$$\begin{aligned}
\frac{\partial E}{\partial W_{Nl-2}} &= \frac{\partial E}{\partial x_{Nl-1}} \frac{\partial x_{Nl-1}}{\partial W_{Nl-2}} \\
&= [(y' - y) \circ a'_{Nl-1}(W_{Nl-1} x_{Nl-2})] \frac{\partial W_{Nl-1} x_{Nl-2}}{\partial W_{Nl-2}} \\
&= \delta_{Nl-1} \frac{\partial W_{Nl-1} x_{Nl-2}}{\partial W_{Nl-2}} \\
&= W_{Nl-1}^T \delta_{Nl-1} \frac{\partial x_{Nl-2}}{\partial W_{Nl-2}} \\
&= [W_{Nl-1}^T \delta_{Nl-1} \circ a'_{Nl-2}(W_{Nl-2} x_{Nl-3})] \frac{\partial W_{Nl-2} x_{Nl-3}}{\partial W_{Nl-2}} \\
\frac{\partial E}{\partial W_{Nl-2}} &= \delta_{Nl-2} x_{Nl-3}^T
\end{aligned} \tag{42}$$

This goes on repeating until the first layer weights $\frac{\partial E}{\partial W_1} = \delta_1 x_0^T$ where one can see the relation, for any i^{th}, $i = 1...Nl - 2$ layer

$$\delta_i = W_{i+1}^T \delta_{i+1} \circ a'_i(W_i x_{i-1}) \tag{43}$$

whereas for $i = Nl - 1$

$$\delta_{Nl-1} = (y' - y) \circ a'_{Nl-1}(W_{Nl-1} x_{Nl-2}) \tag{44}$$

The partial derivative of error E w.r.t i^{th} weight matrix is given by

$$\frac{\partial E}{\partial W_i} = \delta_i x_{i-1}^T \tag{45}$$

The full algorithm for training a neural network can be given as

Algorithm 20 Training a Deep Network

Require: Network with Nl layers, a_i activations and a dataset $x \rightarrow y$
Initialize: All the weight matrices W, α_w
for Number of epochs **do**
  for each pair from dataset **do**
    Conduct forward pass to calculate y'
    Calculate error $E(y, y')$
    Conduct a backward pass to update weights
  end for
  Increment the learning rate α_w
end for

While, it takes a huge time to train a network as the number of epochs and the dataset increases, usually instead of choosing a single pair from dataset, a batch of size N_{batch} is chosen and the optimizer used Batch Gradient Descent to optimize. Many more versions of solvers are available and many techniques are used in Deep Learning to avoid over-fitting, reduce the training time, obtain faster convergence which are not discussed in this project. Training a neural net is a huge topic and is out of the scope of this project.

3.5.3 Deep Q Learning

As it is now known how a Deep Neural Network works as a function approximator by updating it's individual weights to reduce the error, it is time to use it in Q Learning. But before that, it's time to define the state of the agent (RPA).

Environment: The environment in the case of deep Q learning could be completely described by *source*, *target* and the weather polygons to be avoided *polys*. Inspired from the grid generation in A* algorithm, a similar grid with nodes separated by $NodeDistance$ can be created with *source* and *target* on exact centres of the opposite edges.

Actions: A set of all possible actions $\mathbb{A}$ in this case is pretty simple to describe. Looking at the previous development of standards, rules and algorithms it is clear that the agent can take three actions i.e. to deviate in either $0°, +45°, -45°$ to the $target - source$ vector.

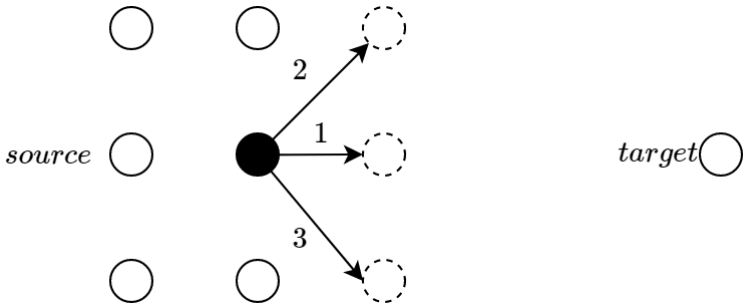


Figure 47: Actions for DQL

Let the actions $\in \mathbb{A}$ be named as

$$action1 \rightarrow 0^{\circ} deviation$$
$$action2 \rightarrow +45^{\circ} deviation$$
$$action3 \rightarrow -45^{\circ} deviation \text{ to the } v_{ts} \text{ vector}$$

Thus the agent, starting from the *source* will choose a particular set of actions to reach the $target$

States: Defining the state in which the agent is present is a tricky job. If the states are large in number, it would take extremely large time to train the model. Moreover, redundant states lead to unnecessary complexity. Less number of states lead to information loss which leads to the system not converging at any time.

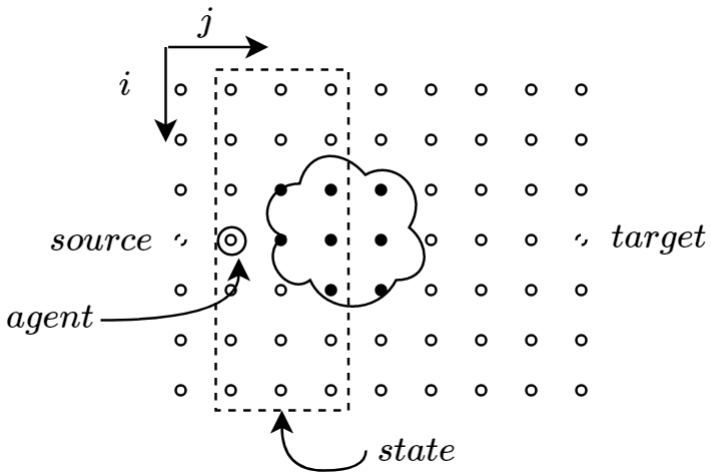


Figure 48: State definition for DQL

Consider the environment in the figure 48 defined by *source*, $target$ weather polygon. Let the grid be defined by a set of total N_{nodes} just like in A*. The state of the agent can be defined by the current column it is present in and a set of immediate next

columns. Let the total number of columns considered to describe a state be 3 as shown in the above figure. Thus, the total state would be a *flattened* vector of $3\sqrt{N_{nodes}} \times 1$ with a combination of i^{th} column appended by $i + 1^{th}$ and $i + 2^{nd}$ column respectively. The values the state vector would take are

- $0 \rightarrow$ If a particular node is free
- $1 \rightarrow$ If a particular node is inside or on the edge of a polygon
- $2 \rightarrow$ If a particular node is the $target$ node
- $3 \rightarrow$ If a particular node is the agent

Rewards: Rewards determine the reinforcement the agent gets for performing certain action and being in a particular state. In this case, the rewards the agent gets are defined as follows

- If the agent enters a free node, it receives a reward of $-10(d + \zeta_c d_{pl})/||target - source||$
- If the agent enters a node inside or on the edge of a polygon it receives a reward of -100 points
- If the agent takes an action a which leads it to move outside the grid it receives a reward of -120 points
- If the agent enters the $target$ node it receives a reward of 200 points

Here, d is the euclidean distance between agent and $target$, d_{pl} is the perpendicular distance between agent and the segment joining $source$ and $target$ whereas ζ_c is the term used to weigh the d_{pl} cost. Thus it is clear that the agent receives a negative reinforcement in terms of a negative reward for choosing the wrong actions whereas it always gets a positive reward for reaching the target. The agent receives a negative reward, not as severe as moving into a polygon which increases as it moves closer to the target while trying to reduce the corridor violations.

Training the agent: As discussed before, when the number of state-action pairs increase, the Q-Table size also increases, thus making it harder to search the specific $s - a$ pair. Let us call the neural network as a function $Q \leftarrow f_{net}(s, w)$ which maps state set $\mathbb{S}$ to Q-values $\mathbb{Q}$ whereas the parameter w symbolizes the weight configuration. Since the output of the neural net is then Q-values, a linear activation is used on the output layer making it a regression network.

The Q value obtained by taking an action a in state s i.e. $Q(s, a)$ is calculated using the neural network. Entering the new state s' and receiving a reward r, the observed Q value $Q_{obs}(s, a)$ is calculated by equation [38] where the neural network again calculates $Q(s', a')$. In Deep-Reinforcement learning, the difference between observed and predicted Q-values can be considered as the error function. This error can be later used to update the weights of the network w. This process is continued for a given number of iterations or until the Q-values for state-action pairs converge.
Recalling the previously discussed Q Learning algorithm without the Q-table can be given as [53]

Algorithm 21 Deep Q Learning

Require: Neural Net $Q \leftarrow f_{net}(s, \theta), x \in \mathbb{S}, y \in \mathbb{A}$
Initialize weight matrices w
for Number of training iterations N_{iter} **do**
  Initialize Environment, Agent state $s \leftarrow s_{init}$
  while s is Not Terminal state **do**
    $Q_{pred}(s, a) \leftarrow f_{net}(s, w)$
    Select one action $a \in \mathbb{A}$ for the current state using exploration strategy
    Update the state s to next state s' by taking the action a
    Get reward $r \leftarrow R(s, a)$ for transiting from s with action a
    Get $Q(s', a') \leftarrow f_{net}(s', w)$
    if s' is a non terminal state **then**
      $Q_{obs}(s, a) \leftarrow r + \gamma \underset{a'}{max} Q(s', a')$
    else
      $Q_{obs}(s, a) \leftarrow r$
    end if
    Calculate error $E \leftarrow ||Q_{pred}(s, a) - Q_{obs}(s, a)||$
    Update the weights $w \leftarrow w - \alpha_w \frac{\partial E}{\partial w}$
    $s \leftarrow s'$
  end while
end for

- As seen in the algorithm [21], for each iteration a Q-value is predicted Q_{pred} for an agent being in state s
- For the action selection strategy, an ϵ *greedy* policy can be used where a random action is selected for a proportion of ϵ trials and the action resulting maximum Q-value is selected for the rest of the trials. This is done so that the agent can take false decisions and get a chance to explore the environment, The factor ϵ is reduced iteratively so as to always choose an best action
- When the agent enters a state s' due to action a, a reward is calculated and the new Q-value $Q(s', a')$ is obtained from the neural net which is then used to calculate $Q_{obs}(s, a)$

- Now, since ideally, the $Q_{obs}(s, a)$ and $Q(s, a)$ are supposed to be same, their error E is calculated and is used to update the weights of the neural network

The problem with this type of learning is that the network cannot address changes in the environment. For instance if an agent is trained for a particular position of weather polygons, it does not behave well if the polygon positions change. To address this problem, the paradigm of experience replay is considered.

Experience Replay: During a gameplay or iteration, all the experiences i.e. states s, s', reward r and the action a are stored in a memory. While training the network, random batches of such experiences are picked up. This makes to network train to different experiences it faced in the past [53].

Algorithm 22 DQL with Experience Replay

Require: Neural Net $Q \leftarrow f_{net}(s, w), x \in \mathbb{S}, y \in \mathbb{A}$
Initialize weight matrices w
for Number of training iterations N_{iter} **do**
  Initialize Randomize the environment, Agent state $s \leftarrow s_{init}$
  while s is Not Terminal state **do**
    $Q_{pred}(s, a) \leftarrow f_{net}(s, w)$
    Select one action $a \in \mathbb{A}$ for the current state using ϵ greedy strategy
    Update the state s to next state s' by taking the action a
    Get reward $r \leftarrow R(s, a)$ for transiting from s with action a
    if $s \neq s'$ **then**
      Store the experience $< s, a, s', r >$ into memory
    end if
    if memory $>$ batchsize **then**
      pick random samples of batchsize and initialize batch error E to zero vector
      for each experience from sample **do**
        Get $Q_{pred}(s, a) \leftarrow f_{net}(s, w)$
        Get $Q(s', a') \leftarrow f_{net}(s', w)$
        if s' is a non terminal state **then**
          $Q_{obs}(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a')$
        else
          $Q_{obs}(s, a) \leftarrow r$
        end if
        Calculate batch error $E \leftarrow E + \frac{1}{2 \text{ batchsize}} ||Q_{pred}(s, a) - Q_{obs}(s, a)||$
      end for
      Update the weights for a batch of errors $w \leftarrow w - \alpha_w \frac{\partial E}{\partial w}$
    end if
    $s \leftarrow s'$
  end while
end for

- The above algorithm gives a pseudocode for experience replay with a memory and a batchsize used to save and pick experiences
- Until the memory size is greater than batchsize, the network is not trained and the experiences are saved in the memory
- An experience is stored only if the new and previous states of the agent are different. This helps in keeping diversity of experiences in the memory. Once the memory reaches maximum limit, the older experiences can be replaced with newer ones
- Once the memory size surpasses batchsize, a random sample of experiences is picked from the memory, the predicted Q-value $Q_{pred}(s, a)$ and observed Q-value $Q_{obs}(s, a)$ is calculated for each instance
- The error is then calculated for the specific experience and the overall error is added to calculated the error for total batch
- The batch error is then minimized by updating the weights of the neural network

Once the network is trained, one can save the weights of the neural network. Let's call the trained network $Q \leftarrow f_{net}(s, w_{trained})$ and using this net as a function approximator, one can simply use the agent to plan a weather free path

Algorithm 23 Deep Q Agent

Require: Trained net $Q \leftarrow f_{net}(s, w_{trained}), source, target, Weather$
Initialize: Randomize the environment, $s \leftarrow s_{init}$
while s is Not Terminal state **do**
  $Q(s, a) \leftarrow f_{net}(s, w_{trained})$
  $a_{opt} \leftarrow \underset{a \in \mathbb{A}}{argmax} Q(s, a)$
  Update s to s' by taking action a_{opt}
  $s \leftarrow s'$
end while

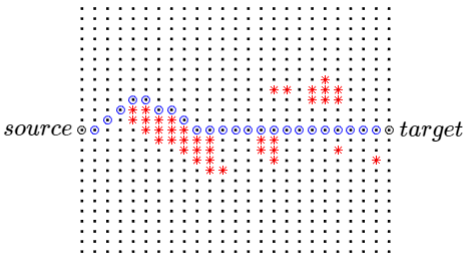


Figure 49: Deep Reinforcement Learning path planning

- Figure 49 shows the path planned by the agent from *source* to *target*

- The environment was discretized into 625 nodes three columns in total were used to define the state thus resulting into a state vector of size 75×1
- The agent was trained from a dataset of 20 weather situations chosen from the OPERA data for 20000 iterations using DeepLearnToolbox [55] for MATLAB
- The Deep Q Learning algorithm has not been trained for time-dependent path planning. For a network to be time dependent, i.e. for it to remember a time sequence, another paradigm called recurrent neural nets are used
- Deep Reinforcement learning demonstrates an innovative method of self learning and is being used to develop self driving cars [56] and training game bots to learn human like gaming abilities [57]
- The reduction of neural net's error function and convergence of Q values are demonstrated in the next section

3.6 Intent Reporting

Intent reporting is the process of letting know the ATC about the intentions of the RPA [58] under link-loss. Although standardized manoeuvre and minimum corridor violation have been considered, it is important to report the intention of the RPA to ATC well in advance. This allows the ATC to become aware of the RPA's situation as well as take suitable actions to consider the safe flight of self and other aircraft. The whitepaper [2] for Automated Weather Avoidance SubSystem project at Airbus mentions a list of reports that could be used by the RPA for WA during link loss. It mentions that the Weather Avoidance Subsystem could replace the pilot element from the 'Controller Pilot Datalink Communication' (CPDLC) [58] and send messages to ATC to notify where it is going.

All the algorithms implemented before request the WA algorithm when the RPA enters the inner neighbourhood $2Tv$ of WP_{active} as seen in the WPC algorithm 6. This means that when the RPA decides to follow the $WP_{planned}$, it has planned it's path for the time $\frac{||WP_{planned}-WP_{active}||}{v}$

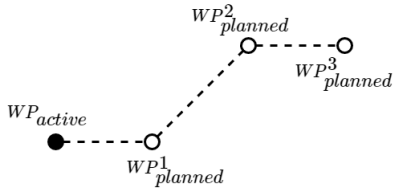


Figure 50: Intent reporting concept

Considering T_{intent} to be the intent reporting time, one method† to implement intent reporting is to already plan the path of length vT_{intent} in advance. As shown in the figure 50, instead of planning for a single $WP_{planned}$, a series of waypoints $WP^i_{planned}, i = 1..N_{intent}$ can be planned.
The A* and RRT-A* algorithms for each request not only plan for $WP_{planned}$, but for the whole time-dependent path. Therefore when the WPC algorithm begins, the WA can return $WP^i_{planned}, i = 1..N_{intent}$. When the RPA reaches the neighbourhood of $WP^1_{planned}$, the WPC requests for WA algorithm where $WP^{N_{intent}}_{planned}$ is used as WP_{begin} for planning and the algorithms can return only one waypoint. Therefore under any circumstances, the RPA has always planned for time

$$T_{intent} = \frac{1}{v}\left(||WP_{active} - WP^1_{planned}|| + \sum_{i=1}^{N_{intent}-1} ||WP^i_{planned} - WP^{i+1}_{planned}||\right)$$

Thus, the intention of the RPA for time T_{intent} are already fixed and cannot be changed once reported to the ATC. This on one hand creates safety standards but also lead to mis-planning. Since the RPA does not know the future weather situations in advance, freezing the path for the next time T_{intent} leads to the RPA finding itself inside weather polygons. This case study is demonstrated in the next section.

4 Tests and Result Analysis

This section produces a set of results obtained from the algorithms developed in the previous sections. It presents the effect of several weight parameters used in developing the algorithms, the functioning of time-dependence, weather data-loss measures for radar and intent reporting. This section also presents test cases to demonstrate the time-complexity of the algorithms and operational test case to determine the benefit of forecast. The objective for each test case, observations from the results and the analysis is clearly mentioned.

4.1 No Weather

Objective: To study the outcome when the WA does not detect any weather event.

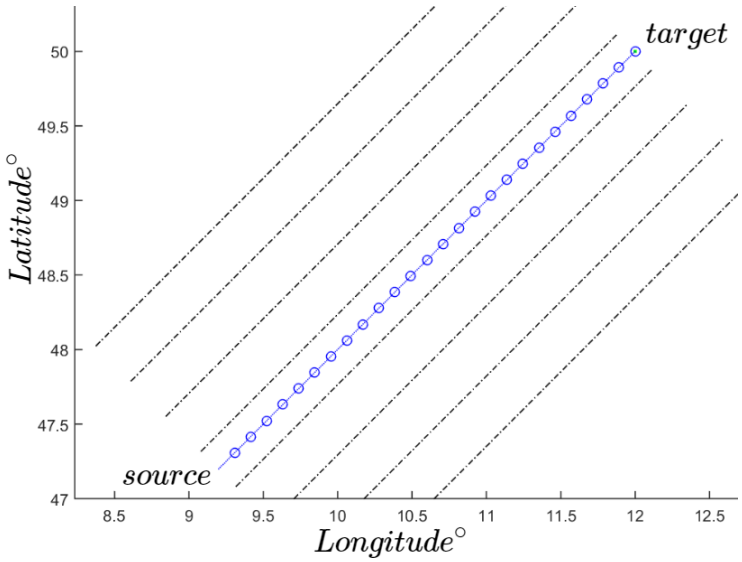
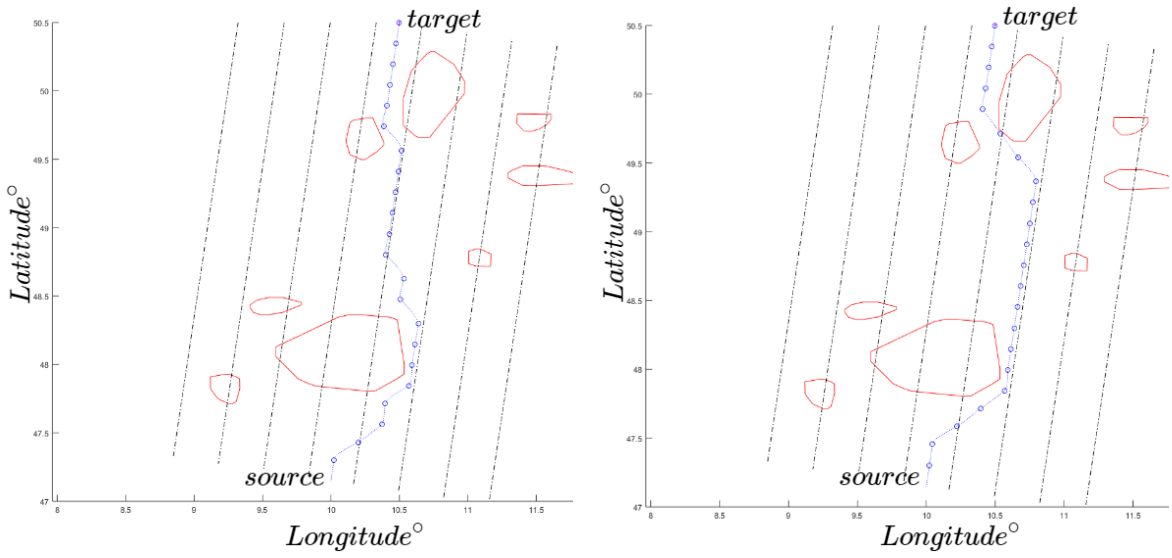


Figure 51: No weather

Observation and Analysis: The WA algorithm does not detect any weather occurrence and the path between *source* and *target* is a straight line which is the shortest possible path

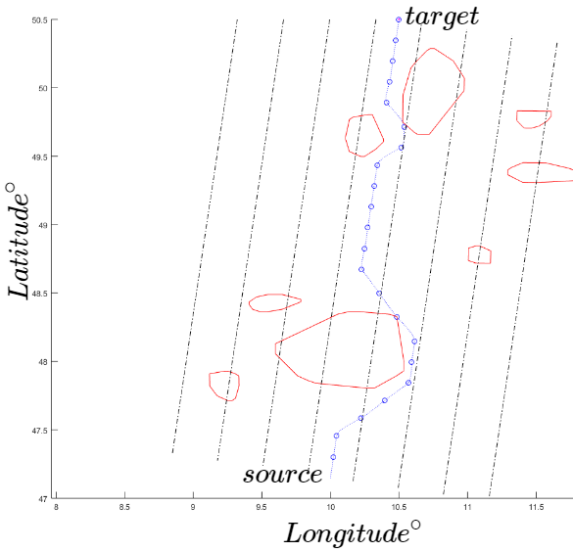
4.2 Effect of weight parameters

Objective: To demonstrate the effect of the weight parameters ζ_c and ζ_t for A* and RRT-A*.
In the following test-case, the twist weight and corridor weights were varied in a stepwise manner. The results were exactly same for both A* and RRT-A* since the basic search algorithm is same for both and henceforth, the plots belong to both the algorithms for static weather scenario

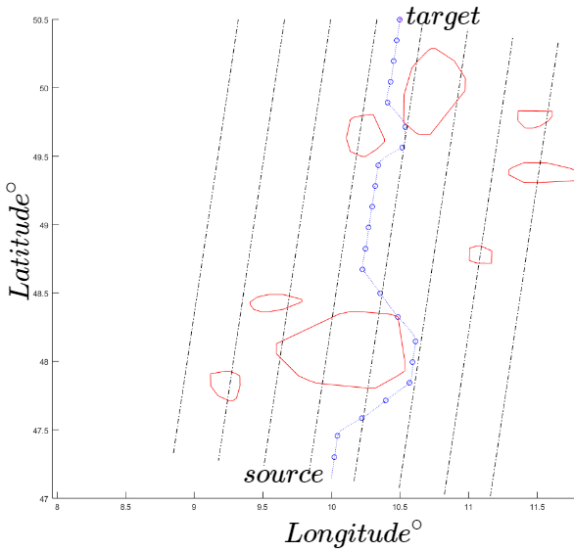


(a) $\zeta_c = 0, \zeta_t = 0$

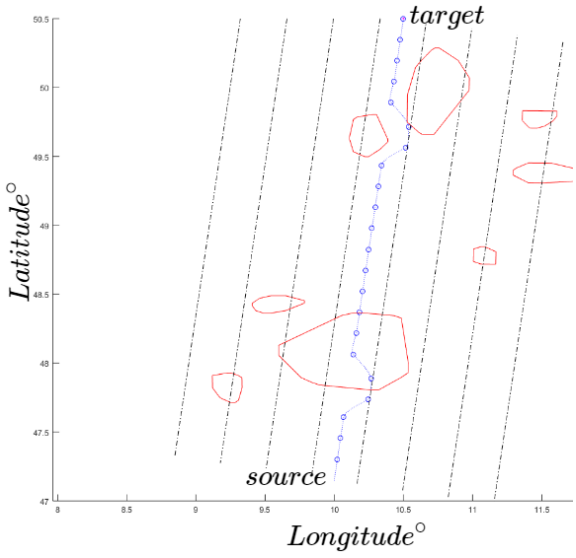
(b) $\zeta_c = 0, \zeta_t = 0.5$



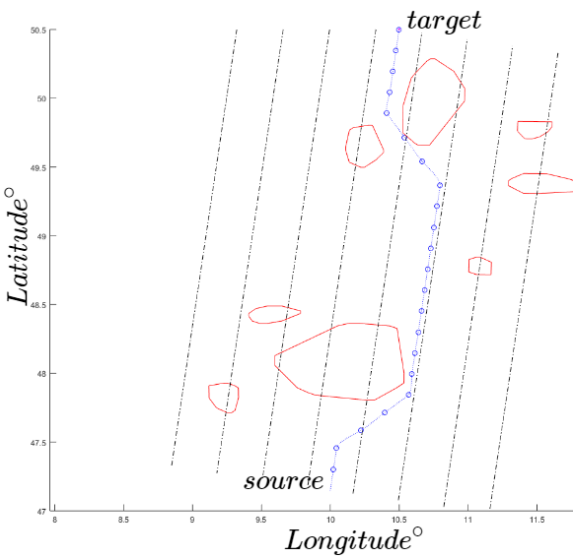
(c) $\zeta_c = 0.5, \zeta_t = 0$



(d) $\zeta_c = 0.5, \zeta_t = 0.5$



(e) $\zeta_c = 1, \zeta_t = 0$



(f) $\zeta_c = 0, \zeta_t = 1$

Figure 52: Effect of ζ_c and ζ_t on A* and RRT-A*

Observations:

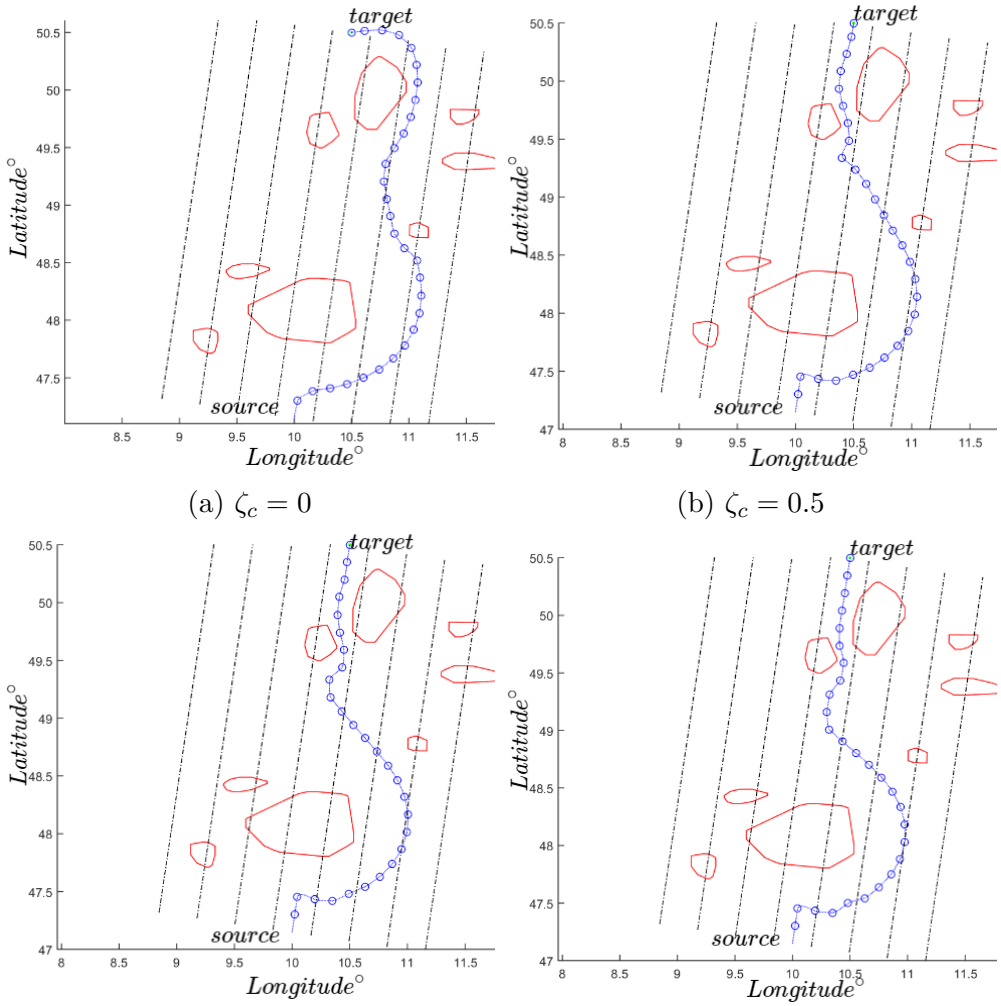
- In the figure 52, case(a) shows the path planned without considering any twist and corridor costs. As seen, the algorithm plans the minimum possible path length but it's not optimal w.r.t airspace violations and number of twists in the path
- In case(b), ζ_c is zero and ζ_t is 0.5 which results in the path with the least possible twists as expected. This path is optimal w.r.t twists but not w.r.t airspace violations
- In case(c), ζ_t is zero and ζ_c is 0.5 which results in a path which is optimal w.r.t airspace violations
- Case(d) has equal weights of 0.5 for both ζ_t and ζ_c which results the same path as in case c since there is no other way in which the path be planned while staying optimal with either of the criteria
- Case(e) ζ_c is 1 whereas ζ_t is zero. A high corridor weight while the twist weight staying zero results in a bad path passing through the weather polygon
- Case(f) ζ_c is zero and ζ_t is 1. This results into an optimal path w.r.t twists as case(b)
- This test case demonstrates that the weights of the algorithms can be changed in order to obtain a path optimal w.r.t a certain criteria. for the application in this project, it is always recommended to keep both the weights to a same value not equal to zero

Analysis:

- As the twist-cost weight ζ_t increases, the algorithm prefers reducing the twists whereas when the corridor-cost weight ζ_c increases, the algorithm prefers reducing corridor violations
- Case(e) demonstrates the requirement that the algorithm should always provide a solution and never return *null* solution. Although the weight ratio $\zeta_c : \zeta_t$ is not recommended, the algorithm prefers to pass through polygon, but surely returns a solution
- To perform standardized manoeuvres, it is recommended to keep the ratio $\zeta_c : \zeta_t$ as 1 so that the algorithm gives equal preference to reduce the corridor violations as well as twists

4.3 Effect of standardization constraints

Objective: This test case demonstrates the effect of applying standardization constraints to the NLP algorithm. It provides a good demonstration to prove the importance of predictability and standardizations in WA.
The following results show the path planed by NLP algorithm without the standardization constraints with increasing corridor weight ζ_c for static weather scenario



(a) $\zeta_c = 0$ (b) $\zeta_c = 0.5$

(c) $\zeta_c = 1$ (d) $\zeta_c = 1.5$

Figure 53: Effect of ζ_c and *No Standardization* on NLP

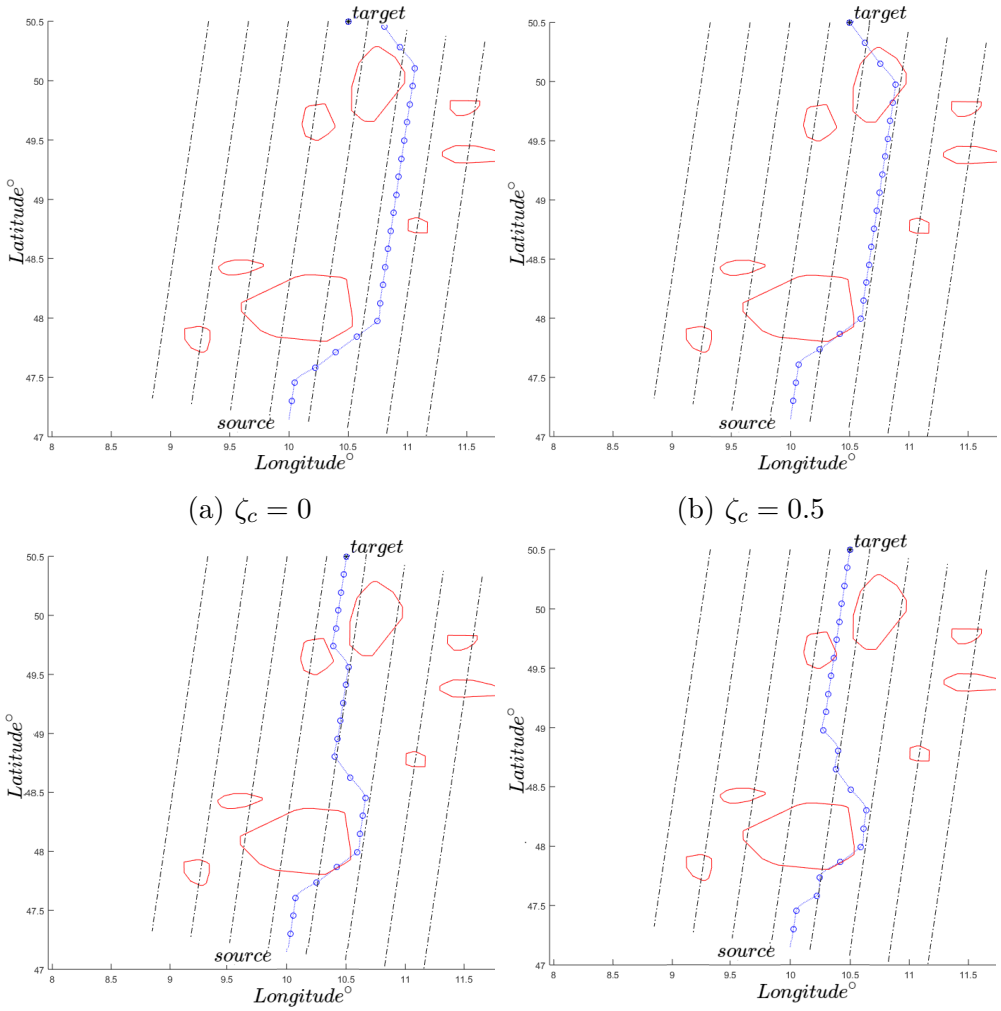
Observations:

- The corridor cost weight ζ_c is zero in case(a) and is incremented by 0.5 thereafter
- The figure 53(a) to (d) show that as ζ_c increases, the planned path lies more and more into the corridor

Analysis:

- If an RPA flying in upper classes carried out such WA manoeuvres, it would pose a threat to other aircraft due to airspace violations and the ATC would find it hard to manage the safe flight of the aircraft. This demonstrates how important it is to set standards for weather avoidance as well as the algorithm to be predictable
- The algorithm even without any standards successfully avoids the weather. This can be used in the lower airspaces where ATC clearance is not always required

The following results in figure 54 show the path planned by NLP algorithm while considering the standardization constraints with increasing corridor weight ζ_c



(a) $\zeta_c = 0$ (b) $\zeta_c = 0.5$

(c) $\zeta_c = 1$ (d) $\zeta_c = 1.5$

Figure 54: Effect of ζ_c and *With Standardization* on NLP

Observations:

- As observed from the figure 54, the length of path which lies within own airspace corridor increases with the increase in ζ_c
- The standardization constraints relatively improve the path developed in terms of predictability

Analysis:

- The path produced by standardized NLP is not optimal as opposed to the one planned by RRT-A* and A* algorithms as is clear from figure 52(d) and figure 54(d)
- This happens since NLP's exploring capability limits itself to the length of the extended paths. But the more the length of extended path, the more time complex is the algorithm. This is shown in a later test case

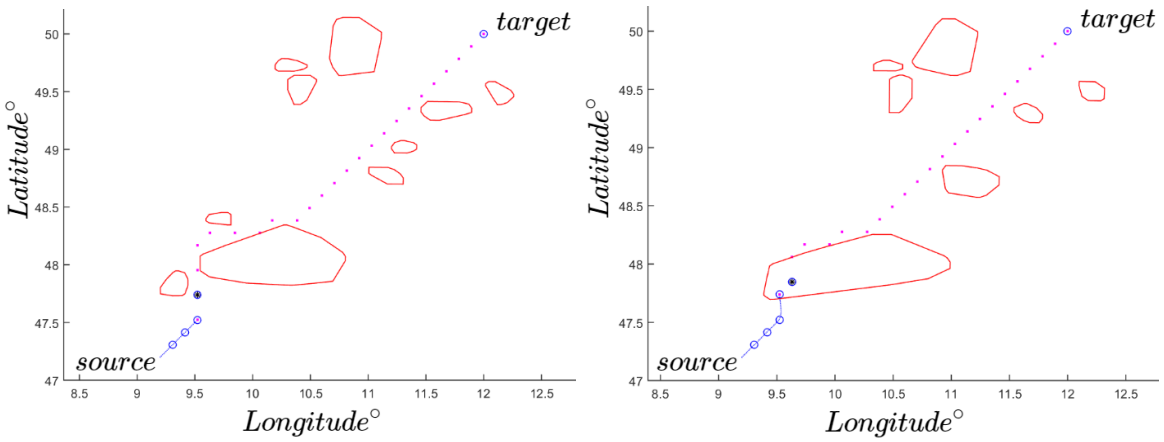
***Note*:** The test cases presented in section 4.2 and section 4.3 lets the A*, RRT-A* and NLP algorithm face the same static weather scenario so that their performances can be compared. As observed, A* and RRT-A* algorithms generate an overall *optimal* path whereas NLP lacks optimality and is therefore excluded from further evaluation for effect of dynamic weather.

4.4 Effect of Weather data

This test-case demonstrates the effect of weather data limitations on path planning.

4.4.1 Effect of OPERA Sampling Rate

Objective: To demonstrate the problems faced by WA algorithm due to the slow sampling rate (15 mins) of OPERA data



(a) OPERA weather data at time T (b) OPERA weather data at time T+ 15 mins

Figure 55: Effect of OPERA sampling

Observation:

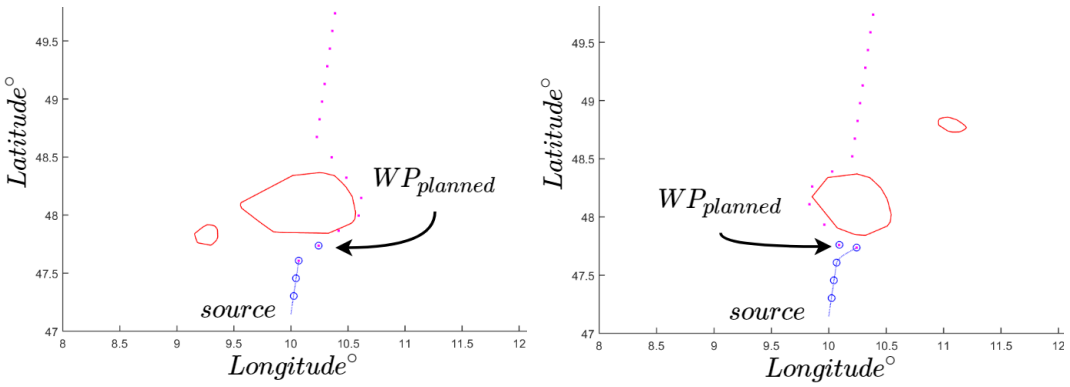
- Figure 55(a) shows the path planned by the RPA for an initial OPERA weather data sample. The path is weather free and standardized. Figure (b) shows an updated OPERA sample in which the RPA finds itself inside the weather polygon

Analysis:

- Although the OPERA data provides the weather information of the whole scenario, the high sampling time renders this benefit moot

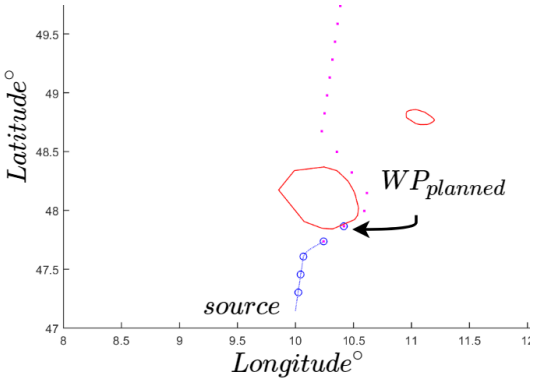
4.4.2 Effect of RADAR Range

Objective: This test-case demonstrates the effect of radar data-loss faced by A* and RRT-A* algorithms and how implementing the radar constraint limit helps in getting rid of this problem.



(a) No data loss

(b) No constraint limit



(c) With constraint limit

Figure 56: Effect of RADAR range

Observations:

- The case(a) shows the initial state with a planned waypoint $WP_{planned}$, in this state, there is no information loss because of radar limit
- Case(b) shows when the RPA reaches the previous $WP_{planned}$, there occurs an information loss due to the radar limit. The new $WP_{planned}$ occurs in the *non-feasible* half plane where the weather cannot be seen because of the radar visibility limit
- Case(c) shows the path planned by applying radar-limit which causes the $WP_{planned}$ to fall in the *feasible* half-plane thus leading to proper path planning

Analysis:

- The radar information-loss can be overcome by applying a radar-limit constraint whereas there is no solution for the OPERA sampling problem
- Therefore, although the radar provides limited weather data when compared to OPERA, to assure a weather free path using only weather radar can be sufficient for WA

4.5 Forecasting

Objective: To demonstrate the effect of using forecast information for path planning with A* and RRT-A*. Also to demonstrate that path planning with forecasting is not limited to the number of obstacles.

The following test-case demonstrates the effect of time dependence on a single weather polygon moving with a speed of 50 kmph in the North-West ([-1,1]) direction. The red polygon(s) are the instantial weather situation whereas the green polygons denote the forecast data

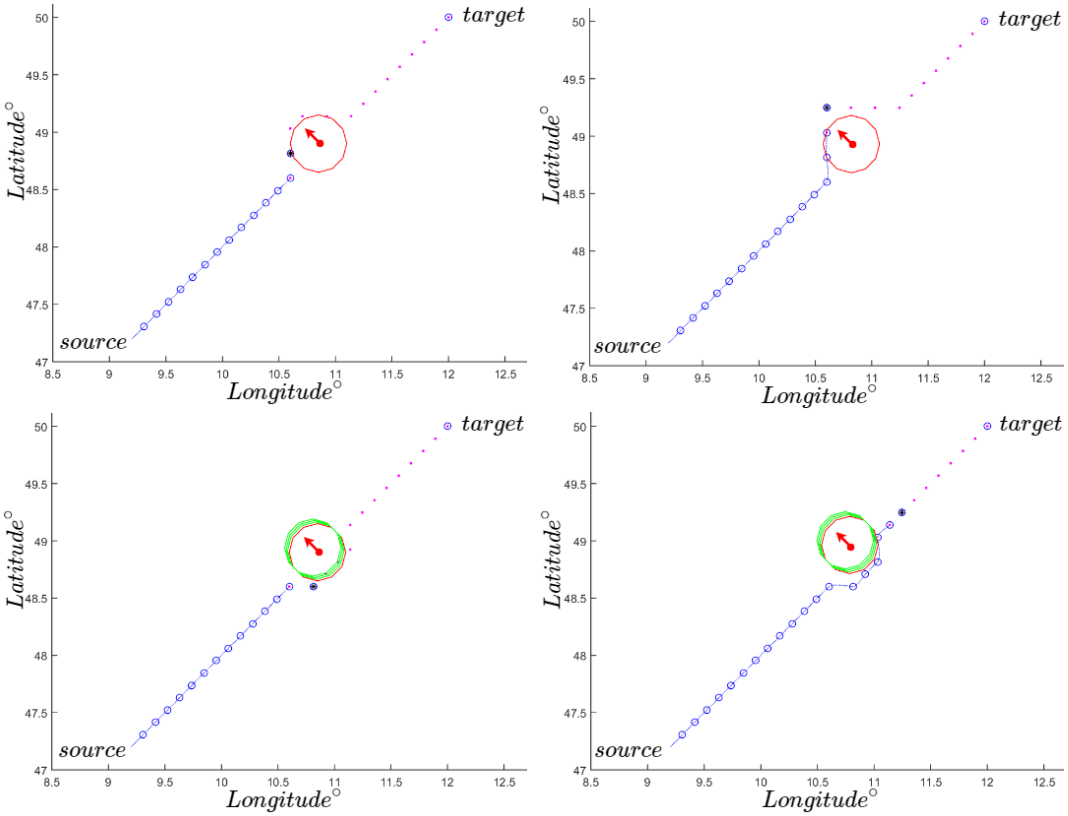


Figure 57: Effect of forecast with one obstacle

Observations:

- The figure 57(a) shows the instance when the algorithm decides to conduct a divergence manoeuvre or take action(2) as discussed previously. The figure(b) demonstrates that the previous decision was rather a poor choice since the RPA finds itself inside the moving weather polygon
- Case(c) shows the effect of forecast switched on. As opposed to (a), the algorithm now chooses to take action(3) since it knows about the movement of obstacle polygon. case(d) shows the RPA successfully avoided the polygon

Analysis:

- With no forecasting, the algorithm plans with the instantial weather situation without considering the equations of motion of the polygons whereas with forecasting turned on, the algorithm successfully can use the information of motion of a particular polygon
- Therefore the number of unnecessary violations and instances of flying through weather can be avoided by using forecast data

The following test-case demonstrates the effect of forecasting on multiple weather polygons

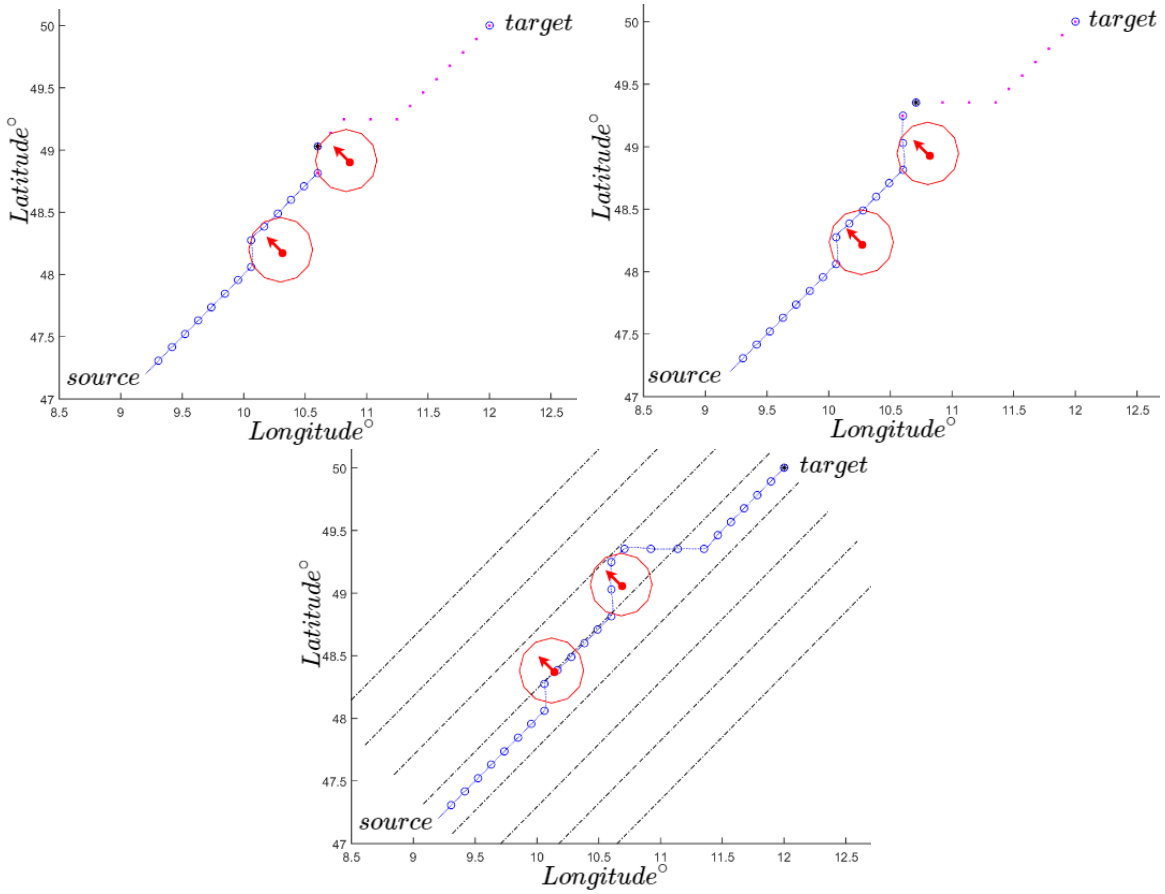


Figure 58: Effect of multiple dynamic obstacles without forecast

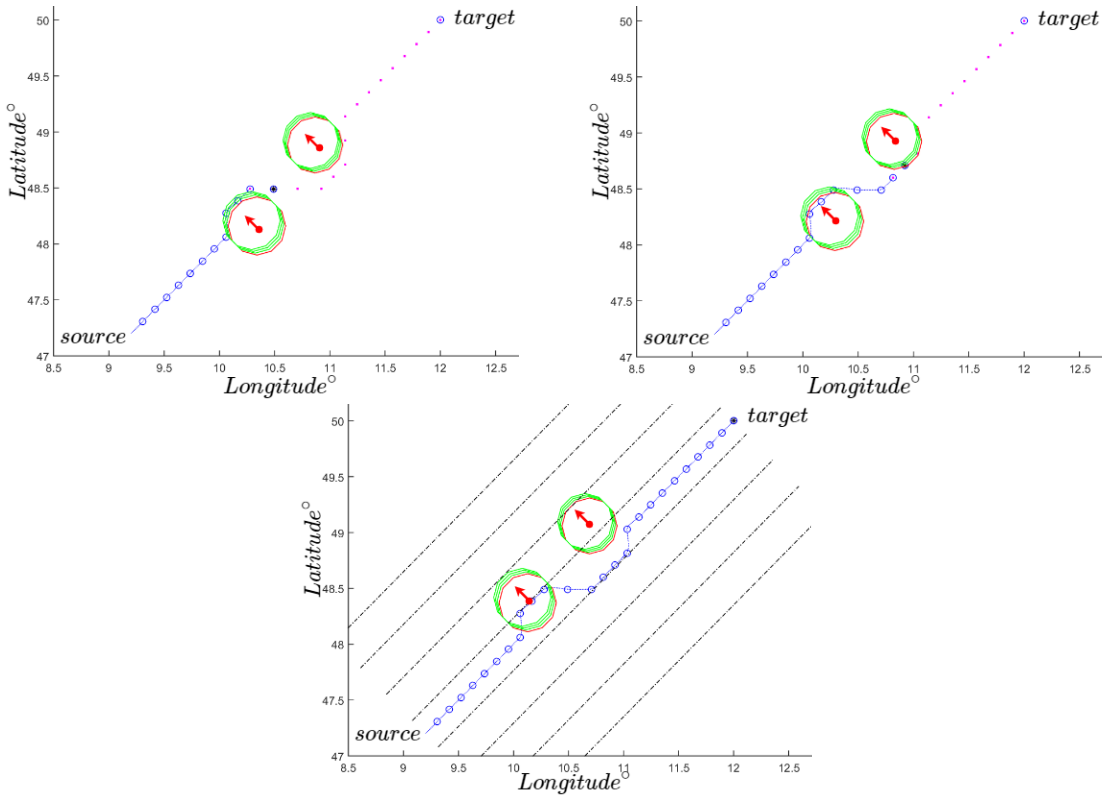


Figure 59: Effect of multiple dynamic obstacles with forecast

Observations:

- The figure 59 (a) and (b) show the instance when the algorithm chooses to take a rather poor divergence action due to lack on weather movement information. This leads to the RPA passing through the weather polygons at some instances. The total airspace violation due to uninformed decision making where the RPA had to violate it's own airspace and had to pass through polygon
- Figure 59(a) and (b) show the instances when the algorithm takes informed decisions to avoid the moving weather polygons. The RPA never enters the actual weather polygon and moreover does not violate it's own airspace as shown in 59(c)

Analysis:

- These test-cases demonstrate that the time-dependence implemented in the algorithms while not being an exactly sophisticated solution, rather works well with multiple non-rigid obstacles such as weather

4.6 Effect of Intent Reporting

Objective: To demonstrate the effects of implementing intent-reporting on path planning with A* and RRT-A* by freezing N_{intent}= 1,2,3 and 4 waypoints respectively.

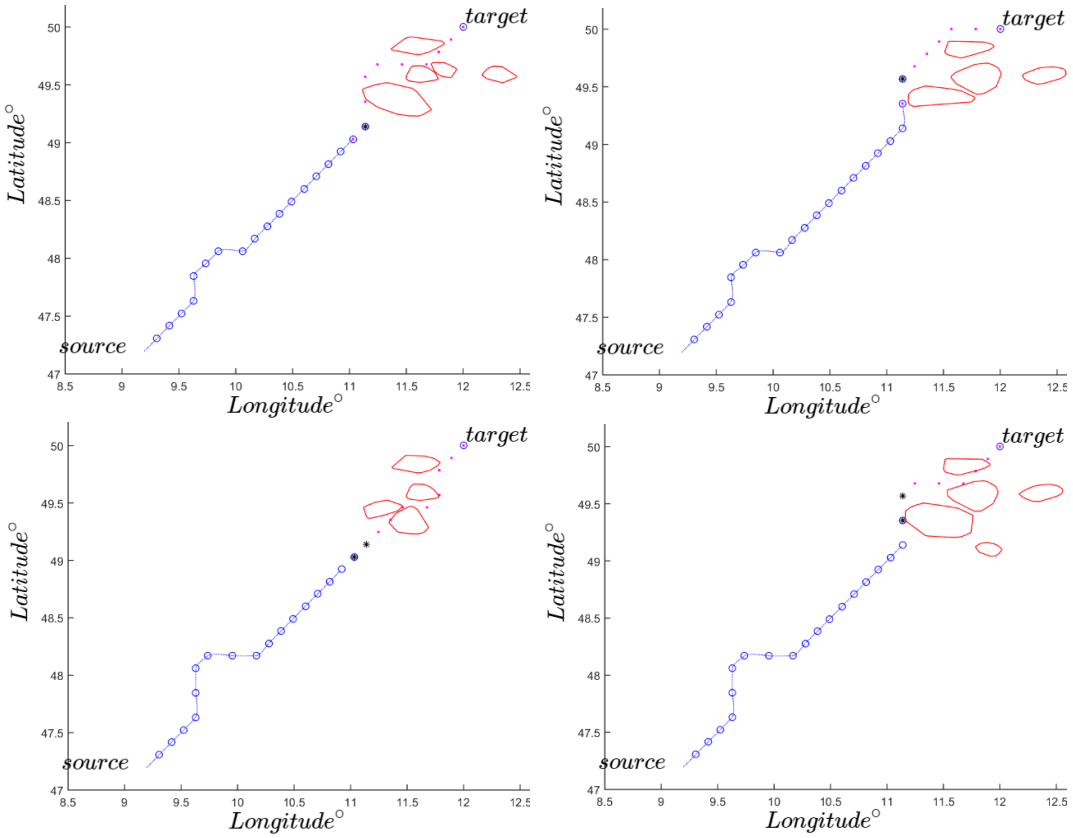


Figure 60: T_{intent} = 2.2 and 4.41 mins ($N_{intent} = 1, 2$)

Observations:

- The figure 60(a) and (b) show the path planning with intent reporting by freezing only one waypoint i.e. the $WP_{planned}$ and has been the usual case so far. This case shows how successfully a changing weather situation is avoided by the algorithm
- The figure(c) and (d) show the path planning by freezing two waypoints which allows more time for intent reporting

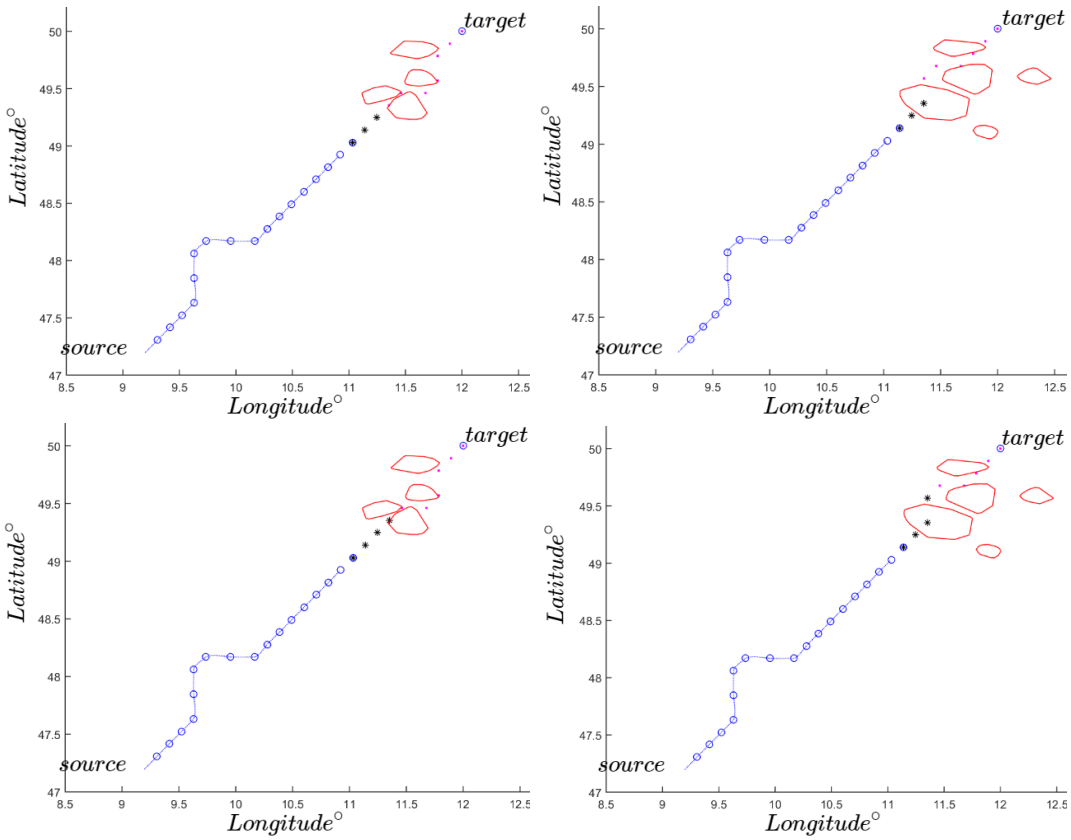


Figure 61: $T_{intent} = 6.62$ and 8.83 mins ($N_{intent} = 3, 4$)

Observations:

- Since the waypoints were frozen at a longer period beforehand, the algorithm could not address the changes in weather conditions after freezing the waypoints. Therefore as shown in figures (b) and (d), the RPA had to pass through the weather polygon with no option to avoid it

Analysis:

- Although intent-reporting sets forth safety standards, a longer intent reporting time creates unreliable path planning situations

4.7 Intent reporting with forecast

Objective: To show the effect of intent reporting with weather forecast.
The moving polygons are same as considered in the previous case i.e. it is moving with 50 kmph in the North-West direction

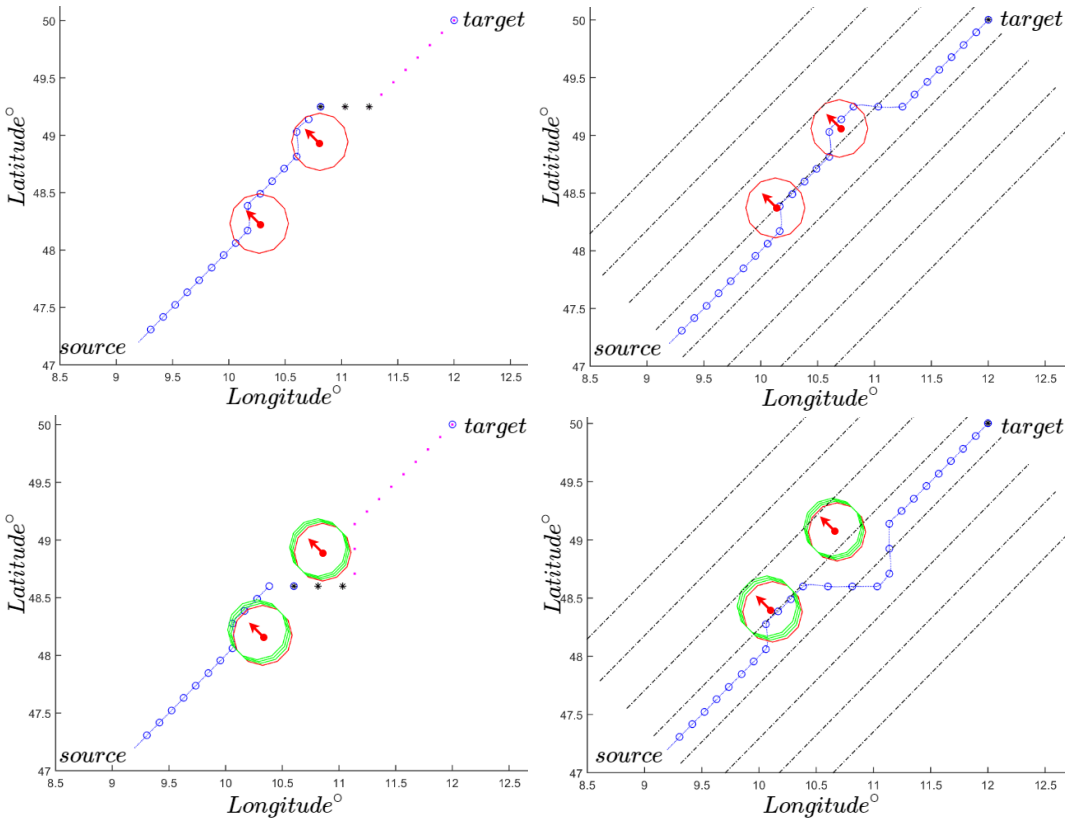


Figure 62: Effect of intent reporting with forecast

Observations:

- The figure 62(a) and (b) shows that without forecasting and $N_{intent} = 3$ the RPA finds itself within a weather polygon. The intent reporting causes more airspace violation as compared to the figure 58
- For multiple polygons with forecast, the RPA successfully avoids the weather polygons but because of the intent-reporting, the RPA violates it's airspace when compared to figure 59

Analysis:

- Increasing the intent reporting time increases the chances of passing through a weather situation but it can be reduced by using weather forecast

- Since the time-dependence algorithm uses three forecasts for the consecutive three waypoints planned, using forecast while maintaining $N_{intent} = 3$ would produce a time-dependent path with sufficient intent reporting time

4.8 NLP extended node complexity

Objective: To demonstrate the time complexity for the number of extended nodes N_{ext} in NLP algorithm

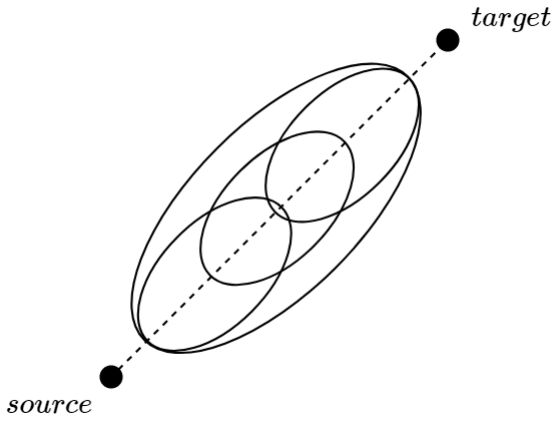


Figure 63: Polygon arrangement for test-case

Setup: The test-case consists of a single elliptical polygon lying on the original flight path. The polygon changes it's position along the $target - source$ vector and also it's size within limits such that it spans a 20% to 80% of the original flight path. For every N_{ext}, ten time readings are taken with different polygon size and position and then average time for every N_{ext} is calculated and then normalized w.r.t time for $N_{ext} = 1$

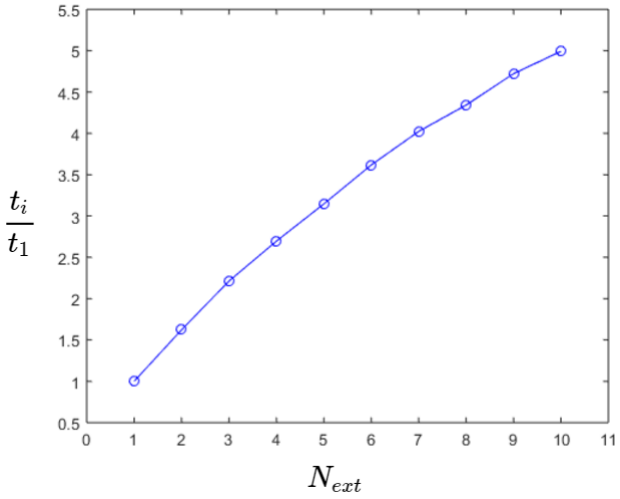


Figure 64: N_{ext} vs. $\frac{t_i}{t_1}$

Observations and Analysis:

- As seen in the figure 64, there is a roughly linear relationship between the calculation time vs. N_{ext}
- The increase in N_{ext} produces a better path but also increases the calculation time linearly
- Throughout experimentation and developments till date, it has been seen that the algorithm produces fairly good results for $N_{ext} = 5$

4.9 Time Complexity of algorithms

Objective: To determine the time complexity of the A*,RRT-A* and NLP algorithms

Setup: Similar to the above test-case, the total time required to plan a path from *source* to *target* has been calculated for a variety of elliptical polygons lying on the original flight path. The polygon's size is varied such that it spans a 20% to 80% of the original flight path same as figure 63. There are ten such positions of polygon for which the computation time is measured and the values are averaged to calculate the average end-to-end path calculation time for each algorithm and normalized w.r.t minimum time required.

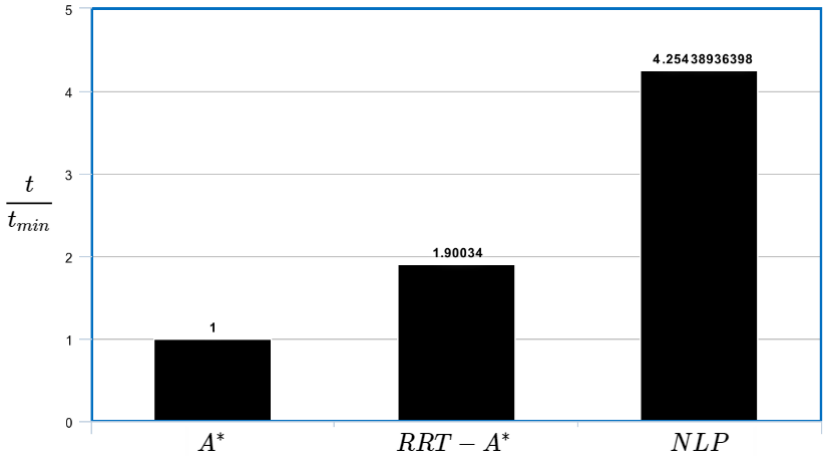


Figure 65: Time Complexity of algorithms

Observations:

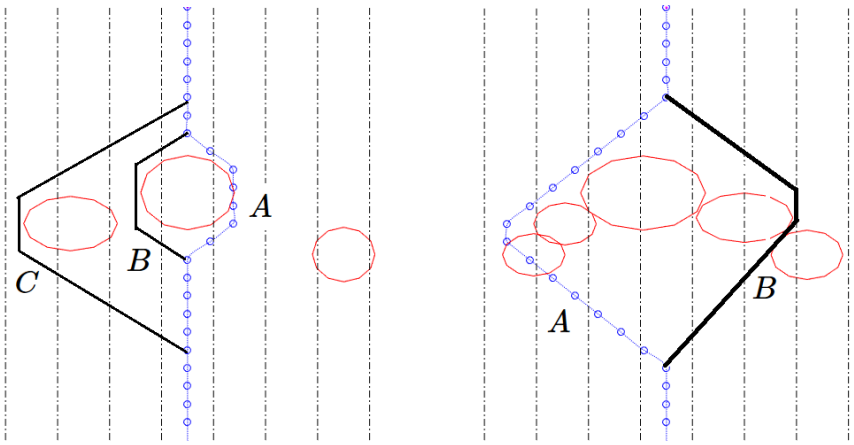
- From the figure 65, it is clear that A* requires the least average time for end-to-end path planning, up next is RRT-A* algorithm whereas the NLP algorithm requires the most time
- The NLP algorithm analysis was conducted with $N_{ext} = 5$

Analysis:

- Since NLP is a model based process, the solver consumes the most time until the optimization process has converged making it almost four times time consuming than A*
- RRT-A* although using A* heuristic based search methodology, the sampling of new nodes and maintaining the *NodeList* takes the most time. This makes it almost twice as time consuming as A*
- A* has a fixed set of nodes saved in a grid and thus it does not have to maintain a *NodeList*, moreover the heuristic based search, *completeness*, *optimality* makes it the best fit for the given application

4.10 Path choice balance

Objective: To prove the requirement of the weather avoidance algorithm mentioned in the White Paper [2]. The first requirement being that the WA algorithm must plan a path maintaining a balance between reducing weather hazard risk and the operational aspects of the RPA. The second requirement is for the WA algorithm to choose the least worst option while avoiding weather



(a) Risk-Operational aspects balance (b) Least worst choice

Figure 66: White Paper test-cases

Observations and Analysis:

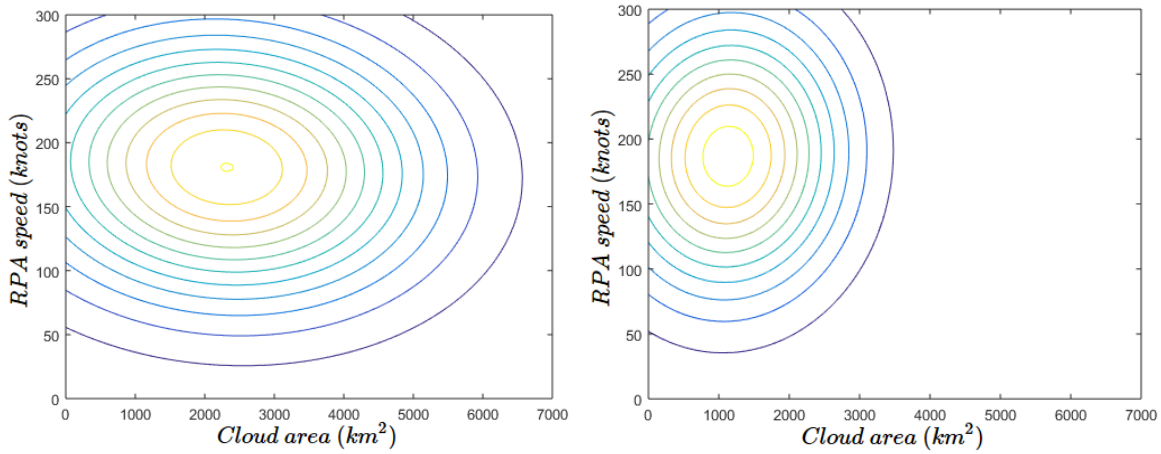
- The Whitepaper [2] proposes a scenario as shown in figure 66(a) where there are three possible choices *A,B and C* to avoid the weather. The algorithm chooses the take the path A since it has a balance between risk posed to the aircraft and operational aspects like airspace violation. Path B lies mere between two weather polygons causing more risk to the aircraft whereas path C causes the most corridor violation

- The Whitepaper [2] proposes a scenario as shown in figure 66(b) where the aircraft faces huge cumulonimbus cells before it. Path B is avoided since it passes between cells where the weather severity changes rapidly. Path A is the least worst choice the algorithm can take when heading back is not an option
- These test cases demonstrate the requirements proposed in the White Paper of project AWASS

4.11 Operational test-case

Objective: To get an estimate of weather avoided by the algorithm for several RPA speeds.

Setup: In this test case, the OPERA and radar data over Bayern region for the date 27.06.2015 from 02:45 till 23:00 is used. The *source* and *target* coordinates in *longitude*° and *latitude*° being [9.2,47.2] and [12.0,50] are used for RPA speed of 300,250,200,150 and 100 knots. These coordinates are then swapped and again simulated for the same speeds. The number of times an RPA encounters itself inside a weather polygon is measured and the corresponding area of the polygon is measured and a bivariate Gaussian distribution is plotted



(a) OPERA distribution (b) RADAR distribution

Figure 67: Operational test case statistics

Observations:

- As seen from figure 67(a) and (b), the encounter of RPA inside a weather polygon is much widespread for all speeds and areas of polygons whereas it is much narrow for radar data

- The mean and covariance for OPERA test case is $\begin{bmatrix}1131.809 & 186.806\end{bmatrix}$ and $\begin{bmatrix}1.167e+06 & 2.287e+03\\ 2.287e+03 & 4.861e+03\end{bmatrix}$ and for RADAR test case is $\begin{bmatrix}2316.093 & 180.841\end{bmatrix}$ and $\begin{bmatrix}3.634e+06 & -7.426e+03\\ -7.426e+03 & 4.843e+03\end{bmatrix}$ respectively
- This signifies that with OPERA data, the RPA encounters itself more times inside a polygon. This is because of the sampling issue and no forecast. On the other hand although the radar data provides less weather information, but since it's real-time the RPA's weather encounters are very less

Setup: To check if the RPA with mean speed 180 knots equipped with forecast information would be able to avoid polygons with mean area 2316 km^2, a circular polygon with radius of 27.1515 kms $\approx 0.25°$ of Earth's geographical coordinates is constructed. The RPA begins at $[9.2, 47.5]°$ and ends at $[12.0, 50.0]°$ always encounters the polygon moving with 50kmph at the center of flight path $p = [10.6, 48.6]°$. With random velocity directions of the polygon and calculated starting positions so as to meet the RPA at p, it was seen that the RPA flew with forecast switched on.

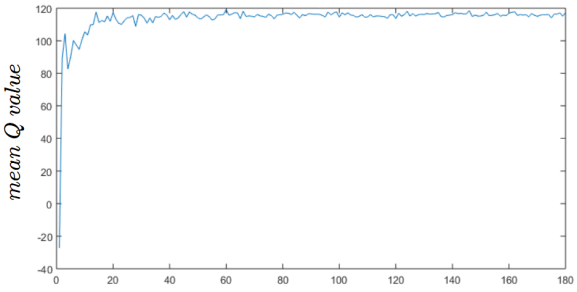
Observation and Analysis:

- The RPA always successfully avoided the weather polygon
- This test case demonstrates that Forecast information although not currently available, would certainly help in autonomous weather avoidance for an RPA in the future

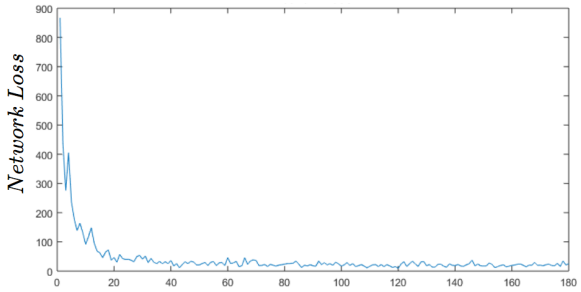
4.12 Q learning agent training

Objective: To demonstrate the process of agent training with experience replay in order to reduce the network error

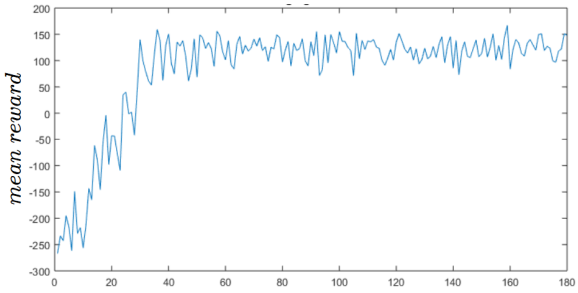
Setup: In the implementation, a simple environment with $N_{nodes} = 49$ nodes was constructed with *source* and *target* on the center of opposite edges of the grid. Four columns were chosen to define the state, including he column in which the agent is present making the state vector to be 28×1 in size, A deep network of $28 \times 1000 \times 500 \times 3$ was used to train the agent with experience replay. Circular weather polygon with randomized size and position lying along the $source - target$ path was used to train the agent for $N_{iter} = 3000$. Once the training iteration starts in the experience replay, for each hundred moves a batch is chosen and the mean value of maximum Q values for each input s and the batch error E is plotted. The total reward earned by the agent for each move is calculated after every hundred moves and it's mean is plotted.



(a) Mean Q value convergence



(b) Network loss reduction



(c) Mean reward

Figure 68: Agent training performance

Observations:

- As seen in the figure 68(a), the mean Q values for a batch changes rapidly in the beginning but continue to become stable over time
- As shown in the figure 68(b) the batch error goes on decreasing over time
- It is observed that in the initial iterations, the agent obtains negative mean reward. But over time, the agent learns how to avoid obstacles and gain positive rewards in each training iteration by reaching the *target* successfully

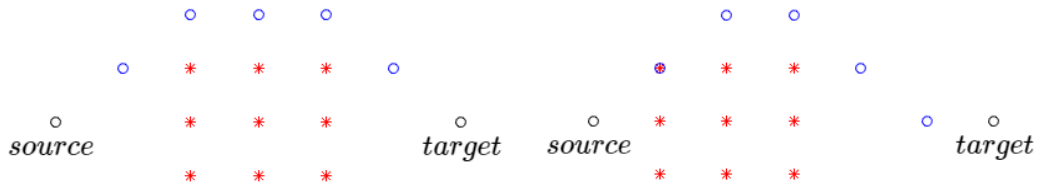


Figure 69: Deep Q agent path planning

Observations and Analysis:

- Figure 69 shows the path planned by the agent trained as before. The agent successfully avoids weather and whenever no choice is available, it passes through the weather to reach the *target*
- The neural network successfully trains the agent to maximize the long term reward while reducing it's loss function. The neural network approximates itself to the function which maps states $\mathbb{S}$ to Q values $\mathbb{Q}$

Although Neural Networks are not certifiable since they do not meet the guidelines set forth by RTCA [44] wrt. software life cycle data, providing software verification test cases and procedures, providing software verification plan [59] as mentioned by Jacklin. There is a hope of using machine learning algorithms to automate processes as the world is moving towards implementing ML techniques in several disciplines.

5 Conclusion

The aim of the thesis project was to develop a weather avoidance algorithm for an RPA flying under link-loss within the mentioned scope and limitations. Four algorithms are developed which try to address the requirements of the project. The algorithms developed in the methodology section can be summarized in table 3

Algorithm	A*	RRT-A*	NLP	DRL
Optimal	Yes	Yes	No	No
Complete	Yes	Yes	Yes	Yes
Complexity	Less	Moderate	High	Very high
Standardizations	Inherent	Inherent	Optional	Inherent
Address Radar Loss	Yes	Yes	Yes	Yes
Address Moving Weather	Yes	Yes	Yes	No
Intent Reporting	Yes	Yes	No	Yes
Certifiable	Yes	Yes	χ	No

Table 3: Comparison of algorithms

The static weather test-cases for optimality 4.3 and time-complexity 4.9 prove that A* and RRT-A* are the best choices and with respect to certifiability, they can be implemented in SCADE to produce certified C codes thus making them certifiable. NLP requires the optimization solver to be written in certified C and only then can the NLP algorithm be certified(χ). Since RRT-A* is twice as complex as A*, it is the best choice for weather avoidance. A* addresses all the aspects of *Detect, Plan* and *Intent Reporting* as mentioned within the project's scope and is thus the most efficient algorithm and with today's available equipment, allows to do *on-line* path planning for weather avoidance.

The results and test-cases show that the algorithms are generic w.r.t number of weather polygons to be avoided [4.5] and the built-in cost parameters ζ_c, ζ_t can be modified in order to change the optimality preferences of the algorithms [4.2]. Since the use of OPERA data causes problems due to slow sampling rate, the mis-planning [4.4.1] can be avoided if a ground-based weather service with better sampling rate is chosen. Nonetheless, using only on-board radar is sufficient [4.4.2] unless the RPA faces huge cumulonimbus cloud wall which could not be fully recognized by the algorithm due to radar range limitations.

Contributions: This project proposes *on-line* path planning algorithms for weather avoidance while considering the rules of airspace, intent reporting and standardizations whereas other literatures mention *off-line* path planning from the perspective of ATC. It demonstrates some of the aspects mentioned in the White Paper [2] like automatic flight path generation by avoiding thunderstorm cells [4.2], least worst of the possible

choices for path planning [4.10], weather avoidance in lower airspaces [4.3]. The project proposes a functional, conceptual background for weather avoidance considering dynamic weather and demonstrates the importance of weather forecasting services while conducting an autonomous flight under link-loss [4.11].

Future Work: Since the project only considers lateral deviations for thunderstorms, automatic weather avoidance for icing, hail, turbulence can be considered in the future. Automated WA during take-off and landing as well as including self-separation measures in the same algorithm can be focused upon. Introducing a certified weather forecasting service could be considered for automated WA. Since this project only proposes a theoretical implementation of Intent Reporting, a thorough study with more tests and simulations needs to be done in the future. Jacklin [59] proposed methods to close the certification gaps in adaptive flight control software, such methods could be used to introduce more self-learning, intelligent autonomy into aircraft in the future.

Appendix

Data for NLP extended node complexity

N_{ext}	1	2	3	4	5	6	7	8	9	10
case 1	5.12	7.92	10.54	13.06	15.24	17.52	19.85	21.10	22.74	24.56
case 2	3.95	6.77	9.28	11.37	13.12	14.98	16.51	17.98	19.47	20.67
case 3	3.94	6.76	9.19	11.39	13.11	14.92	16.48	17.85	19.64	20.57
case 4	4.94	8.42	12.03	14.45	17.12	19.68	21.58	23.78	25.99	27.58
case 5	5.31	8.68	13.02	14.90	16.99	20.50	21.78	23.55	25.37	26.63
case 6	5.76	8.79	11.88	14.88	17.24	19.45	22.47	23.88	26.35	28.01
case 7	3.21	5.27	7.20	8.77	10.52	12.08	13.65	14.79	15.96	16.73
case 8	3.78	5.10	6.87	8.49	10.11	11.33	12.75	13.83	15.01	15.72
case 9	3.26	6.00	7.05	8.69	10.28	11.87	13.07	14.02	15.40	16.06
case 10	3.23	5.24	6.97	8.56	9.96	11.41	12.82	13.74	14.74	15.64
t *mean*	4.25	6.90	9.40	11.46	13.37	15.37	17.10	18.45	20.07	21.22
$\frac{t}{t_1}$	1.00	1.62	2.21	2.70	3.15	3.62	4.02	4.34	4.72	4.99

The above table gives the runtime in seconds for the NLP algorithm to generate a full *source* to *target* path which refers to figure 64.

Data for time complexity of algorithms

Algorithm	A*	RRT-A*	NLP
case 1	2.6055	4.5146	15.2358
case 2	3.0752	5.8977	13.1211
case 3	3.0801	5.8818	13.1085
case 4	2.2450	4.1520	17.1199
case 5	2.9520	5.3058	16.9901
case 6	4.0220	7.6005	17.2440
case 7	2.9350	5.5745	10.5248
case 8	3.9423	8.0316	10.1081
case 9	3.3960	6.6485	10.2786
case 10	3.1724	6.1120	9.9649
t *mean*	3.1426	5.9719	13.3696
$\frac{t}{t_{min}}$	1.0000	1.9000	4.2543

The above table gives the runtime in seconds for A*, RRT-A* and NLP algorithm to full *source* to *target* path which refers to figure 65.

References

[1] ICAO, "Icao doc 10019: Manual on remotely piloted aircraft systems (rpas),"

[2] S. Banerjee, "Automated weather avoidance sub-system - white paper," 2017.

[3] ICAO, "Icao annex 2: Rules of the air chapter 5,"

[4] F. A. Administration, *Instrument Procedures Handbook FAA-H-8083-16B*.

[5] ICAO, "Icao annex 2: Rules of the air chapter 4,"

[6] "Eu ops 1, regulation (ec) no 859/2008 (oj, 20.09.2008),"

[7] ICAO, "Icao annex 11: Air traffic services,"

[8] F. A. Administration, "00-6b - aviation weather," 2016.

[9] M. W. Otte, "A survey of machine learning approaches to robotic path-planning,"

[10] W. Pereira Coutinho, M. BATTARRA, and J. Fliege, "The unmanned aerial vehicle routing and trajectory optimisation problem," 2017.

[11] M. Matthews and R. DeLaura, "Assessment and interpretation of en route weather avoidance fields from the convective weather avoidance model," in *10th AIAA Aviation Technology, Integration, and Operations Conference, Fort Worth, TX*, 2010.

[12] T. Prevot, J. S. Mercer, L. H. Martin, J. R. Homola, C. D. Cabrall, and C. L. Brasil, "Evaluation of high density air traffic operations with automation for separation assurance, weather avoidance and schedule conformance," in *Proceedings of the 11th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference*, 2011.

[13] J. Gauci, K. Theuma, and D. Z. Mangion, "Pilot evaluation of a decision support tool for weather and terrain avoidance during departure," in *16th AIAA Aviation Technology, Integration, and Operations Conference*, p. 3295, 2016.

[14] J. Krozel, S. Penny, J. Prete, and J. S. Mitchell, "Comparison of algorithms for synthesizing weather avoidance routes in transition airspace," in *AIAA Guidance, Navigation and Control Conf*, 2004.

[15] Z. Xie and Z. Zhong, "Aircraft path planning under adverse weather conditions," in *MATEC Web of Conferences*, vol. 77, p. 15001, EDP Sciences, 2016.

[16] P. Ramu, H. Nguyen, S. Avadhanam, J. Newton, J. Yadegar, and A. Ganguli, "Passive camera based cloud detection and avoidance for aircraft systems," June 30 2015. US Patent 9,070,285.

[17] J. J. Pannequin, A. M. Bayen, I. M. Mitchell, H. Chung, and S. Sastry, "Multiple aircraft deconflicted path planning with weather avoidance constraints," in *AIAA Guidance, Navigation and Control Conference*, 2007.

[18] M. Mattei and L. Blasi, "Smooth flight trajectory planning in the presence of no-fly zones and obstacles," *Journal of guidance, control, and dynamics*, vol. 33, no. 2, p. 454, 2010.

[19] A. Richards and J. P. How, "Aircraft trajectory planning with collision avoidance using mixed integer linear programming," in *American Control Conference, 2002. Proceedings of the 2002*, vol. 3, pp. 1936–1941, IEEE, 2002.

[20] T. Schouwenaars, É. Féron, and J. How, "Safe receding horizon path planning for autonomous vehicles,"

[21] L. Yang, J. Qi, J. Xiao, and X. Yong, "A literature review of uav 3d path planning," in *Intelligent Control and Automation (WCICA), 2014 11th World Congress on*, pp. 2376–2381, IEEE, 2014.

[22] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, Dec 1959.

[23] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[24] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pp. 3310–3317, IEEE, 1994.

[25] S. Koenig and M. Likhachev, "D* lite.," *AAAI/IAAI*, vol. 15, 2002.

[26] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong planning a∗," *Artificial Intelligence*, vol. 155, no. 1-2, pp. 93–146, 2004.

[27] M. Likhachev, G. J. Gordon, and S. Thrun, "Ara*: Anytime a* with provable bounds on sub-optimality," in *Advances in Neural Information Processing Systems*, pp. 767–774, 2004.

[28] T. Schouwenaars, A. Stubbs, J. Paduano, and E. Feron, "Multivehicle path planning for nonline-of-sight communication," *Journal of Field Robotics*, vol. 23, no. 3-4, pp. 269–290, 2006.

[29] J. Tisdale, Z. Kim, and J. K. Hedrick, "Autonomous uav path planning and estimation," *IEEE Robotics & Automation Magazine*, vol. 16, no. 2, 2009.

[30] S. K. Gan and S. Sukkarieh, "Multi-uav target search using explicit decentralized gradient-based negotiation," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 751–756, IEEE, 2011.

[31] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[32] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[33] R. J. Geraerts, *Sampling-based motion planning: Analysis and path quality*. 2006.

[34] R. Geraerts, "On experimental research in sampling-based motion planning," in *Workshop on Benchmarks in Robotics Research*, pp. 31–34, 2006.

[35] S. Khanmohammadi and A. Mahdizadeh, "Density avoided sampling: An intelligent sampling technique for rapidly-exploring random trees," in *Hybrid Intelligent Systems, 2008. HIS'08. Eighth International Conference on*, pp. 672–677, IEEE, 2008.

[36] M. Zucker, J. Kuffner, and M. Branicky, "Multipartite rrts for rapid replanning in dynamic environments," in *Robotics and Automation, 2007 IEEE International Conference on*, pp. 1603–1609, IEEE, 2007.

[37] S. Choudhury, S. Scherer, and S. Singh, "Rrt*-ar: Sampling-based alternate routes planning with applications to autonomous emergency landing of a helicopter," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 3947–3952, IEEE, 2013.

[38] I. Hasircioglu, H. R. Topcuoglu, and M. Ermis, "3-d path planning for the navigation of unmanned aerial vehicles by using evolutionary algorithms," in *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pp. 1499–1506, ACM, 2008.

[39] C.-T. Cheng, K. Fallahi, H. Leung, and K. T. Chi, "Cooperative path planner for uavs using aco algorithm with gaussian distribution functions," in *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on*, pp. 173–176, IEEE, 2009.

[40] J. L. Foo, J. Knutzon, V. Kalivarapu, J. Oliver, and E. Winer, "Path planning of unmanned aerial vehicles using b-splines and particle swarm optimization," *Journal of aerospace computing, Information, and communication*, vol. 6, no. 4, pp. 271–290, 2009.

[41] N. Shahidi, H. Esmaeilzadeh, M. Abdollahi, and C. Lucas, "Memetic algorithm based path planning for a mobile robot.," in *International Conference on Computational Intelligence*, pp. 56–59, 2004.

[42] Machine Learning, "Machine learning — Wikipedia, the free encyclopedia," 2017.

[43] G. Machado, "Ml basics: supervised, unsupervised and reinforcement learning," 2016.

[44] L. A. Johnson *et al.*, "Do-178b, software considerations in airborne systems and equipment certification," *Crosstalk, October*, vol. 199, 1998.

[45] "Technical data sheet scade suite® 17.0," 2016.

[46] M. Inc, "Matlab r2017 primer," 2017.

[47] I. Doc, "4444," *PANS ATM*, 2007.

[48] N. Shneydor, *Missile Guidance and Pursuit: Kinematics, Dynamics and Control*. Horwood engineering science series, Elsevier Science, 1998.

[49] Z. Wang and S. Zlatanova, "An a*-based search approach for navigation among moving obstacles," in *Intelligent systems for crisis management*, pp. 17–30, Springer, 2013.

[50] B. Ingersoll, K. Ingersoll, P. DeFranco, and A. Ning, "Uav path-planning using bezier curves and a receding horizon approach," 2016.

[51] R. H. Byrd, J. C. Gilbert, and J. Nocedal, "A trust region method based on interior point techniques for nonlinear programming," *Mathematical Programming*, vol. 89, no. 1, pp. 149–185, 2000.

[52] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical programming*, vol. 106, no. 1, pp. 25–57, 2006.

[53] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[54] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[55] R. B. Palm, "Prediction as a candidate for learning deep hierarchical models of data," Master's thesis, 2012.

[56] L. Fridman, "Mit 6.s094: Deep learning for self-driving cars."

[57] G. Lample and D. S. Chaplot, "Playing fps games with deep reinforcement learning.," 2017.

[58] ICAO, *ICAO Doc 10037: GOLD*.

[59] S. Jacklin, "Closing the certification gaps in adaptive flight control software," in *AIAA Guidance, Navigation and Control Conference and Exhibit*, p. 6988, 2008.