Czech Technical University in Prague
Fakulty of Electrical Engineering
Department of Cybernetics

BACHELOR THESIS

Název práce

Autor: Matěj Račinský

Thesis supervisor: Dr. Martin Saska

In Prague, 2015

Název práce: Název bakalářské práce

Autor: Matěj Račinský

Katedra (ústav): Katedra kybernetiky

Vedoucí bakalářské práce: Dr. Martin Saska

e-mail vedoucího: saska@labe.felk.cvut.cz

Abstrakt V předložené práci studujeme... Uvede se abstrakt v rozsahu 80 až 200 slov. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut sit amet sem. Mauris nec turpis ac sem mollis pretium. Suspendisse neque massa, suscipit id, dictum in, porta at, quam. Nunc suscipit, pede vel elementum pretium, nisl urna sodales velit, sit amet auctor elit quam id tellus. Nullam sollicitudin.

Klíčová slova: klíčová slova (3 až 5)

Title: Název bakalářské práce v angličtině

Author: Matěj Račinský

Department: Department of Cybernetics

Supervisor: Dr. Martin Saska

Supervisor's e-mail address: saska@labe.felk.cvut.cz

Abstract In the present work we study ... Uvede se anglický abstrakt v rozsahu 80 až 200 slov. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut sit amet sem. Mauris nec turpis ac sem mollis pretium. Suspendisse neque massa, suscipit id, dictum in, porta at, quam. Nunc suscipit, pede vel elementum pretium, nisl urna sodales velit, sit amet auctor elit quam id tellus. Nullam sollicitudin. Donec hendrerit. Aliquam ac nibh. Vivamus mi. Sed felis. Proin pretium elit in neque. Pellentesque at turpis. Maecenas convallis. Vestibulum id lectus.

Keywords: klíčová slova (3 až 5) v angličtině

Prohlašuji, že jsem svou bakalářskou práci napsal(a) samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne December 30, 2015 Jméno Příjmení + podpis

CONTENTS

Abstrakt ii

Zadání práce iii

1 Algorithm v

2 Grouping of goals for guiding path vii

3 Paths narrowing ix

4 Dubins maneuvers xi

5 Implementation xii

5.1 External libraries xii

5.2 Code structure and services xii

CHAPTER
ONE

ALGORITHM

Basis of whole algorithm is here in pseudocode

Configuration variable is instance of Configuration class, which holds all configuration variables, including selected map. Map holds all Aeras of Interest (AoI) and obstacles. All obstacles and AoIs are represented now as rectangles.

Even if we want to find as short path to AoI as possible, path too near to obstacles is not convenient for realization, because UAVs do not use car like motion model used in this simulation, so they can not exactly follow found trajectories. So in real environment, it is convenient for the swarm to have path planned with safe distance from obstacles. Because of that fact, we need to increase size od obstacles, which is done in line 2 in function amplifyObstacles.

Line 3 represents discretization of map to graph. Discretizaion divides map to squares with size set in configuration and eac square is represented by node. In this graph, there are 4 types of nodes: Free, Obstacle, UAV and Goal. If part of square of whole square is covered by obstacle, corresponding node has type Obstacle. If part of square or whole square is covered by AoI, corresponding node has type Goal. If square contains UAV, corresponding node has type UAV and rest of squares have corresponding nodes qith type Free.

Edges in this graph are only between nodes of neighboring squares, so each node has maximally 8 edges. Obstacle nodes do not have any edges.

After converting map to nodes, optional grouping of goals for guiding path can be turned on. I will cover the grouping in chapter 2.

Algorithm 1.1 Basis of whole algorithm

```
1:  map := configuration.getMap();
2:  map := amplifyObstacles(map);
3:  nodes := mapToNodes(map);
4:  paths := createGuidingPaths(nodes);
5:  rrtPath := rrtPath(paths, map, nodes);
6:  lastState := getBestFitness(rrtPath, map);
7:  path := getPath(lastState);
8:  path = straightenCrossingTrajectories(path);
9:  path := optimizePathByDubins(path, map);
10: savePathToJson(path);
```

Line 4 calculates the guiding paths for rrt path algorithm using the A* algorithm. Algorithm has modified cost function and in addition to cost function of A* algorithm, cost of current node is added during the calculation. Nodes neighboring with obstacles has bigger cost than nodes which do not have obstacles as neighbors. Thanks to this modification, guiding path avoids obstacles and has bigger distance to obstacles.

On line 5 the rrt path algoritm takes place. This function returns structure with tree with root at starting position of UAVs and with array containing leaves of this three, where all UAVs are in Areas of Interest.

On line 6 the leaf, where UAVs have best coverage of AoI is chosen. Quality of coverage is determined by cost function, which will be mentioned later.

todo: možná sem odkaz na kapitolu

On line 7 the path is built from last state.

On line 8 is optional preparation before optimization using Dubins maneuvers. In the preperation, all crossings of paths of individual UAVs are straigtened, so UAVs do not cross other UAVs trajectories during whole path. During implementation of this method were complications, which are covered in chapter, and thus the row was removed from the algorithm. 3

todo: možná sem odkaz na kapitolu

On line 9 is optimization by Dubins maneuvers. Optimizations is covered in chapter.

Last line is only persisting of path to file for usage of path by different program.

CHAPTER
TWO

GROUPING OF GOALS FOR GUIDING PATH

During this processing of map (method MapProcessor::getEndNodes in codebase) all AoIs are grouped to one big AoI, which is the smallest rectangle covering all AoIs.

If this modification is turned on, instead of one goal for every AoI (node in middle of AoI rectangle is considered as goal node), only one goal is used for all AoIs. This prevents swarm to split and whole swarm has only one guiding path. The main reason to have one big swarm instead of more smaller swarms is that smaller swarms (or individual UAVs in case of same count of AoI and UAVs) is relative localization, which can be used better when having only one swarm.

Below are images of maps with goals and obstacles. Goals have grren color and obstacles have grey color.

This approach has advantage, when individual AoIs are near to global goal of whole gorup, as seen in 2.2.

Disadvantage of this method is, when individual AoIs have big distance from each other than can be covered by UAVs, this approach totally fails, because RRT-Path, which is much faster than RRT, has goal very distant from AoIs, as can be seen in 2.1.

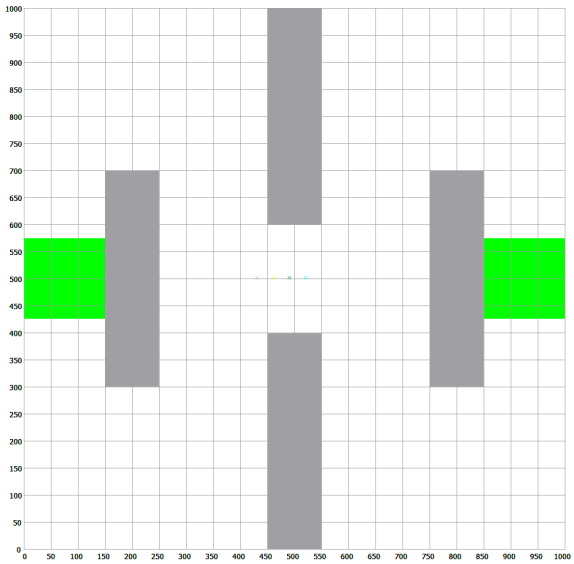


Figure 2.1: Map with goals unsuitable for grouping

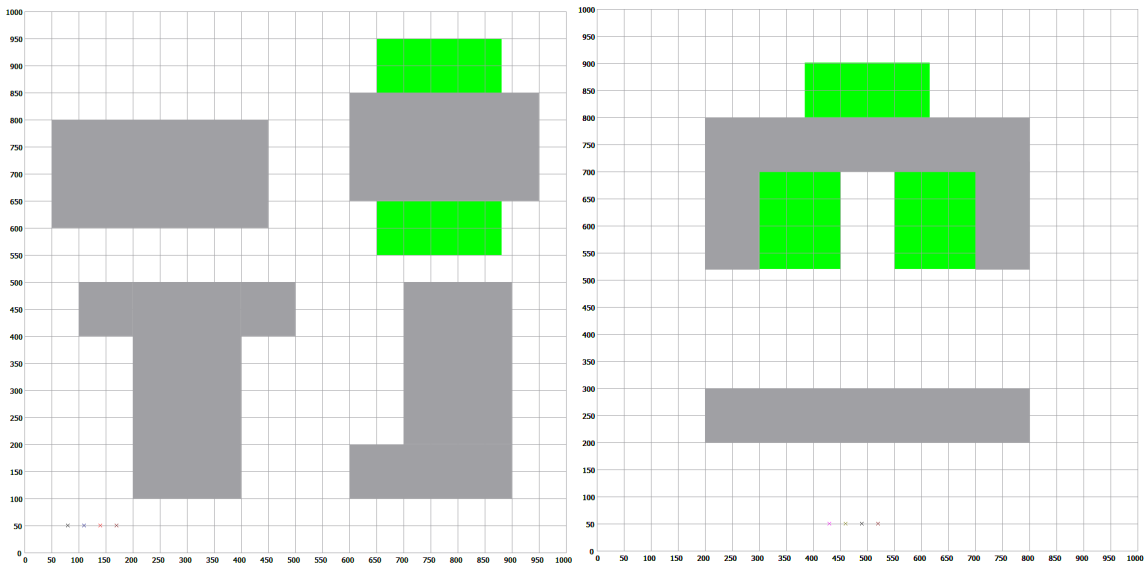


Figure 2.2: Maps with goals suitable for grouping

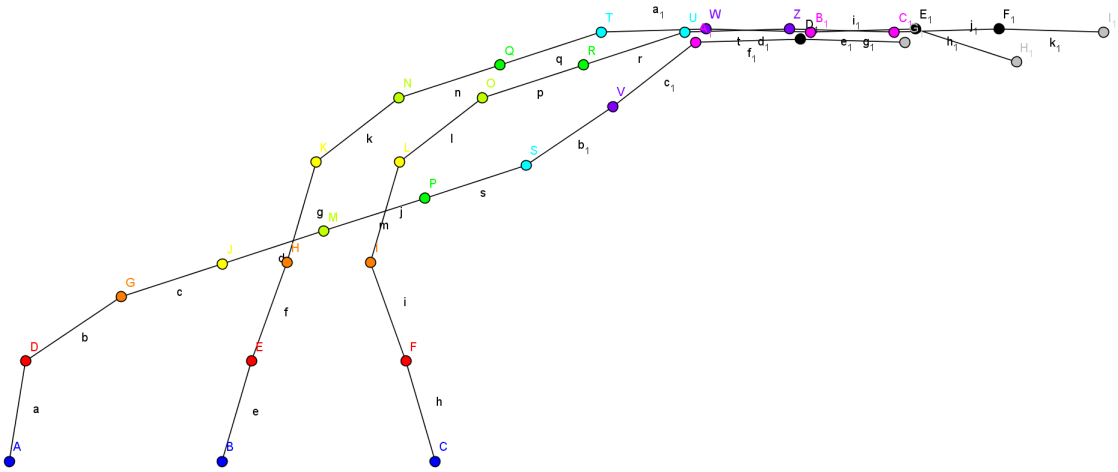
CHAPTER
THREE

PATHS NARROWING

RRT-Path algorithm checks crossing paths between neighboring states, so between n-th state and n+1-th state no trajectories are crossing each other. But between states trajectories are still crossing. In this image 3.1 is shown path found by RRT-Path. Every color marks one state in RRT-Path. As we can see, check in algorithm prevents from crossing path between neighboring states, but crossing of paths in different times can not be easily prevented. We can see in image, that paths cross between points J (yellow), M (light green) and H (orange), K (yellow), so there is no easy approach to prevent path collisions between n-2-th state and n-th state. Optimization by Dubins maneuvers shortens trajectory of UAVs, so UAVs could be in these trajectories in different time, so there could be collisions after optimization. Another complication occurs, when time difference between two states is too low, then uavs could collide, because in reality UAV can not follow path precisely, but only with some errors.

I tried to straighten crossing trajectories, but all attempts failed. Straightening was done by switching parts of crossing paths from the earliest crossing state to end. But then paths had different leghts, which was unsuitable for path planning for swarm. This can be done by adding "waiting" points, points in different state but with same position in different time. But then is really hard sto traighten longer path with many crossings (this path is really short, paths in other maps are much longer and more complicated). For motion model with inertia is also really hard to deal with waiting states, which complicates following of straightened trajectories. Due to all complications mentioned above, this part was removed from algorithm. But it is possible to add it, when better aproach will be found.

Figure 3.1: Crossing paths



CHAPTER
FOUR

DUBINS MANEUVERS

Dubins maneuvers, also called Dubins curves or Dubins were published by Lester Eli Dubins in 1957

CHAPTER
FIVE

IMPLEMENTATION

This part will cover implementation of algorithm, which was used for simulations. Whole codebase can be found at this github repository.

5.1 External libraries

In implementation are used some external libries. Every used library is mentioned here.Boost libraries is used for smart pointers, libraries for Dubins maneuvers are from Master Thesis by Petr Váňa[1]. Generating of JSON from C++ object is done via Json Spirit library. Another external library is V-Collide from The University of North Carolina at Chapel Hill.

Because V-Collide sources were written in 1997 and because I used C++11 compiler to compile my source codes, I had to rewrite part of this library for compatibility and to make public API easier to use. Modifications can be seen in this github repository.

Last use external library is QT, which was used to create platform independent GUI.

5.2 Code structure and services

Here is shown brief UML scheme demonstrating dependency diagram of codebase. To keep diagram simple, only services are displayed, other classes, thich are not services, were left out for lucidity. Diagram was generated using software StarUML

Core class holds core of whole Application and has all other classes as dependencies, as is shown in image 5.1.

As mentioned in 1 chapter Configuration is DTO for all configuration variables, but to keep reasonable amount of classes, Configuration is also service, which delegates all configuration changes from GUI to Core class. Configuration and GuiDrawer implementation LoggerInterface are the only connections between Core and GUI.

State factory creates State classes according to Factory pattern. State class represents state in RRT-Path algorithm. State has coordinates and rotations for all UAVs.

Persister persists found path to JSON using Json Spirit library.

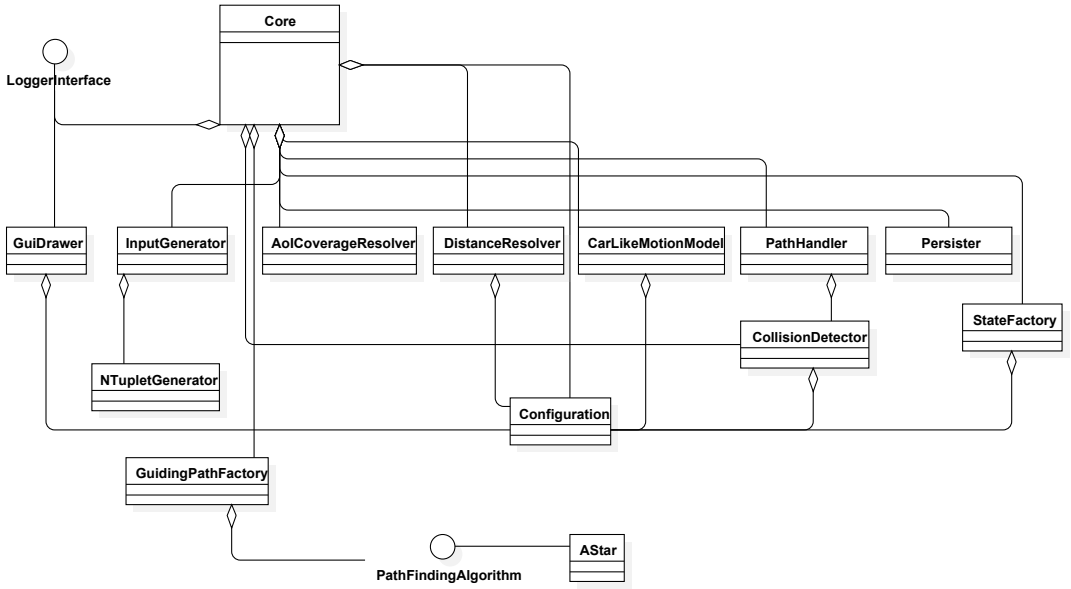
PathHandler serves as utils class for manipulations with path (vector of State classes).

CarLikeMotionModel holds motion model algorithm.

InputGenerator is used to generate inputs to motion model.

NTupletGenerator only generates variation with repeating for given input.

Figure 5.1: Dependency diagram



DistanceResolver counts distances between two states and distance of path.

AoICoverageResolver determines cost function for states,where all UAVs are in AoIs.

GuidingPathFactory is wrapper for PathFindingAlgorithm interface and is used by Core to find guiding path.

Implementation of PathFindingAlgorithm is AStart class.

BIBLIOGRAPHY

[1] Petr Váňa, *Path Planning for Non-holonomic Vehicle in Surveillance Missions* [online]. [cited 2015-12-29]. URL: https://dspace.cvut.cz/bitstream/handle/10467/61814/F3-DP-2015-Vana-Petr-thesis.pdf

[2] Dubins, L.E. (July 1957). "On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents"