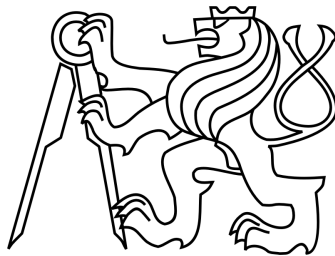


CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF ELECTRICAL ENGINEERING
DEPARTMENT OF CYBERNETICS



BACHELOR THESIS

RRT-path method used for cooperative surveillance by
group of helicopters

Author: Matěj Račinský

Thesis supervisor: Dr. Martin Saska

In Prague, 2016

Název práce: Aplikace algoritmu RRT-path v úloze autonomního dohledu skupinou helikoptér

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, consectetur 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: RRT-path method used for cooperative surveillance by group of helicopters

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, consectetur 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 17th April 2016

Jméno Příjmení + podpis

CONTENTS

Abstrakt	2
Zadání práce	3
1 Algorithm	5
2 Grouping of goals for guiding path	7
3 Paths straightening	9
4 Motion model	11
5 Dubins curves	12
6 Optimization using Dubins curves	14
7 Path re-sampling	15
7.1 Influence of re-sampling on Dubins curves optimization	15
8 Implementation	19
8.1 External libraries	19
8.2 Code structure and services	19
Bibliography	20

ALGORITHM

Basis of whole algorithm is here in pseudocode todo: dodat odkaz na algoritmus

Configuration variable is instance of Configuration class, which holds all configuration variables, including selected map. Map holds all Areas 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 of obstacles, which is done in line 2 in function `amplifyObstacles`.

Line 3 represents discretization of map to graph. Discretization divides map to squares with size set in configuration and each 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 with type Free.

Edges in this graph are only between nodes of neighbouring 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.

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 := resamplePath(path, map);
10: path := optimizePathByDubins(path, map);
11: savePathToJson(path);
```

I will cover the grouping in chapter 2.

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 neighbouring with obstacles has bigger cost than nodes which do not have obstacles as neighbours. Thanks to this modification, guiding path avoids obstacles and has bigger distance to obstacles.

On line 5 the RRT path algorithm takes place. This function returns structure with tree with root at starting position of UAVs and with array containing leaves of this tree, 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.

On line 7 the path is built from last state.

On line 8 is optional preparation before optimization using Dubins curves. In the preparation, all crossings of paths of individual UAVs are straightened, 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

On line 9, re-sampling of path is made, mainly due to requirements of real UAVs, which is mentioned in chapter 7.

On line 10 is optimization by Dubins curves. Optimizations is covered in chapter.

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

todo: možná
sem odkaz na
kapitolu

todo: možná
sem odkaz na
kapitolu

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 green colour and obstacles have grey colour.

This approach has advantage, when individual AoIs are near to global goal of whole group, 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.

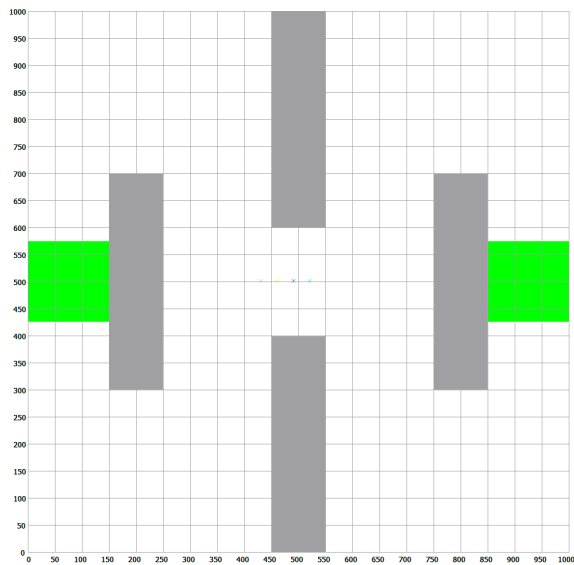


Figure 2.1: Map with goals unsuitable for grouping

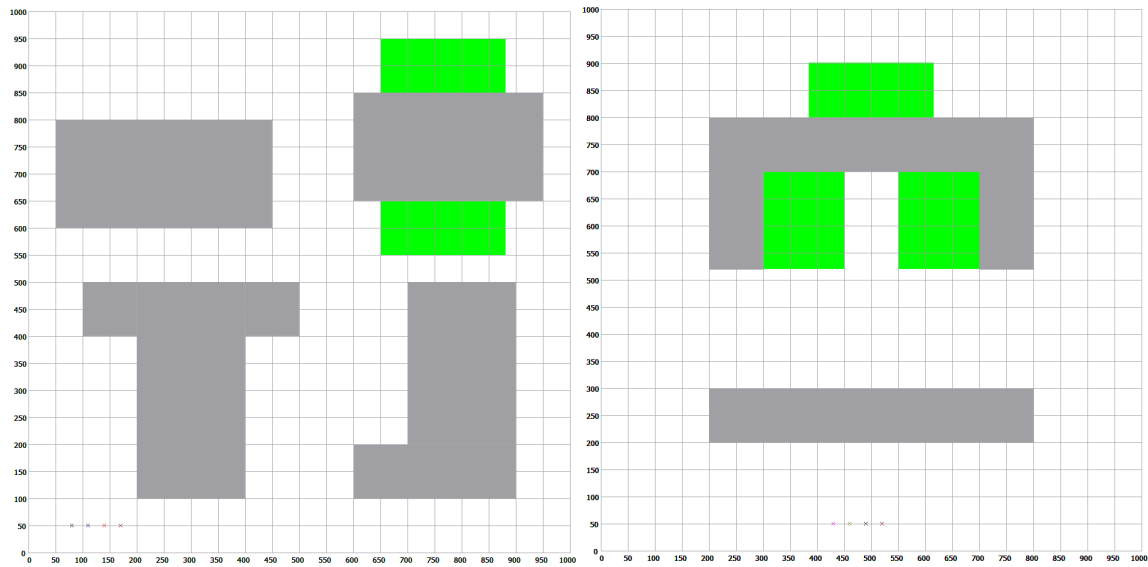


Figure 2.2: Maps with goals suitable for grouping

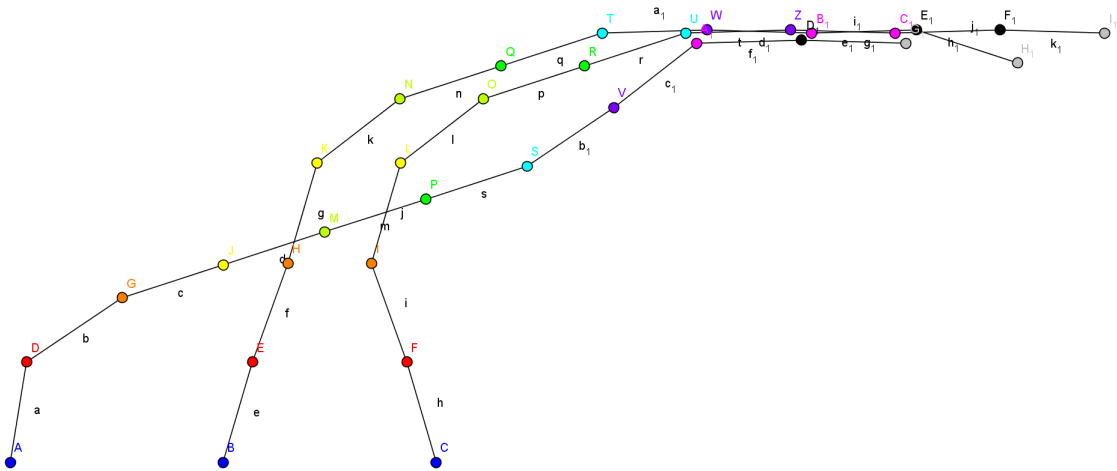
PATHS STRAIGHTENING

RRT-Path algorithm checks crossing paths between neighbouring 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 colour marks one state in RRT-Path. As we can see, check in algorithm prevents from crossing path between neighbouring 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 curves 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.

todo: zjistit, jak
je to správně
anglicky

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 lengths, 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 to straighten 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 approach will be found.

Figure 3.1: Crossing paths



MOTION MODEL

The RRT-Path algorithm is universal and works without motion model, which is fine for holonomic robot, but usage of motion model allows us to find path more feasible for swarm of UAVs than without using of holonomic robot. For this purpose, car like model was chosen. Differential equations of motion model in 3D from [3] are

$$\begin{aligned}\dot{x}(t) &= v(t) \sin \varphi(t) \\ \dot{y}(t) &= v(t) \cos \varphi(t) \\ \dot{z}(t) &= w(t) \\ \dot{\varphi}(t) &= K(t) v(t)\end{aligned}\tag{4.1}$$

where $x(t)$, $y(t)$, $z(t)$ are coordinates of UAV, $\varphi(t)$ represents heading of UAV, $v(t)$ is forward velocity, $K(t)$ is curvature, $w(t)$ is ascent velocity. $\begin{bmatrix} K(t) & w(t) & v(t) \end{bmatrix}$ represent input vector of motion model. Differential equations are useful for representation in equations, but not useful for computer algorithm. For usage in algorithm are better difference equations instead of differential equations. When inputs are held constant in each time interval between two time steps, difference equations are

$$\begin{aligned}x(k+1) &= \begin{cases} \text{if } K(k+1) \neq 0 \\ x(k) + \frac{1}{K(k+1)} (\sin(\varphi(k) + K(k+1)v(k+1)\Delta t(k+1)) - \sin(\varphi(k))) \\ \text{if } K(k+1) = 0 \\ x(k) + v(k+1) \cos(\varphi(k)) \Delta t(k+1) \end{cases} \\ y(k+1) &= \begin{cases} \text{if } K(k+1) \neq 0 \\ y(k) - \frac{1}{K(k+1)} (\cos(\varphi(k) + K(k+1)v(k+1)\Delta t(k+1)) - \cos(\varphi(k))) \\ \text{if } K(k+1) = 0 \\ y(k) + v(k+1) \sin(\varphi(k)) \Delta t(k+1) \end{cases} \\ z(k+1) &= z(k) + w(k+1) \Delta t(k+1) \\ \varphi(k+1) &= \varphi(k) + K(k+1) v(k+1) \Delta t(k+1)\end{aligned}\tag{4.2}$$

DUBINS CURVES

Dubins curves, also called Dubins manoeuvres or Dubins path were published by Lester Eli Dubins in 1957 [1]. Dubins path is optimal path for car like motion model. Path is optimal, when car moves at constant forward speed. The other important constraint is the maximum steering angle ϕ_{max} , which results in a minimum turning radius ρ_{min} . As the car travels, consider the length of the curve in $\mathcal{W} = \mathbb{R}^2$ traced out by a pencil attached to the centre of the car. The task is to minimize the length of this curve as the car travels between any q_I and q_G . Due to ρ_{min} , this can be considered as a bounded-curvature shortest-path problem. If $\rho_{min} = 0$, then there is no curvature bound, and the shortest path follows a straight line in \mathbb{R}^2 . In terms of a cost function, the criterion to optimize is

$$L(\tilde{q}, \tilde{u}) = \int_0^{t_F} \sqrt{\dot{x}(t)^2 + \dot{y}(t)^2} dt \quad (5.1)$$

, where t_F is the time at which q_G is reached, and a configuration is denoted as $q = (x, y, \theta)$, \tilde{x}_t denotes the function $\tilde{x}_t : [0, t] \rightarrow X$, which is called the state trajectory (or state history). This is a continuous-time version of the state history, which was defined previously for problems that have discrete stages. Similarly, \tilde{u}_t denotes the action trajectory (or action history). If q_G is not reached, then it is assumed that $L(\tilde{q}, \tilde{u}) = \infty$. [2]

When considering constraints of inputs (actions) for motion model, the system can be simplified to

$$\begin{aligned} \dot{x} &= \cos \theta \\ \dot{y} &= \sin \theta \\ \dot{\theta} &= u \end{aligned} \quad (5.2)$$

in which u is chosen from the interval $U = \{-\tan \phi_{max}, 0, \tan \phi_{max}\}$. For simplicity, assume that $\tan \phi = 1$. The following results also hold for any $\phi_{max} \in (0, \pi/2)$.

It was shown in [1] that between any two configurations, the shortest path for the Dubins car can always be expressed as a combination of no more than three motion primitives. Each motion primitive applies a constant action over an interval of time. Furthermore, the only actions that are needed to traverse the shortest paths are $u \in \{-1, 0, 1\}$. The primitives and their associated symbols are shown in 5.1. The S primitive drives the car straight ahead. The L and R primitives turn as sharply as possible to the left and right, respectively. Using these symbols, each possible kind of shortest path can be designated as a sequence of three symbols that corresponds to the

Table 5.1: The three motion primitives from which all optimal curves for the Dubins car can be constructed.

Symbol	Steering u
L	-1
S	0
R	1

order in which the primitives are applied. Let such a sequence be called a word . There is no need to have two consecutive primitives of the same kind because they can be merged into one. Under this observation, ten possible words of length three are possible. Dubins showed that only these six words are possibly optimal:

$$\{LRL, RLR, LSL, LSR, RSL, RSR\}. \quad (5.3)$$

The shortest path between any two configurations can always be characterized by one of these words. These are called the Dubins curves.

OPTIMIZATION USING DUBINS CURVES

The Dubins curves are used to optimize trajectory found with RRT-Path algorithm.

Whole optimization works as follows.

Two random states are chosen, which means one path as part of whole trajectory for each UAV will be optimized. Beginning and ending point of selected path are used as input to Dubins curve algorithm.

Dubins curve is then sampled with same frequency used in RRT-Path algorithm and is tested for feasibility. When the new path is feasible and shorter than original path, the original path is replaced. and distances of the original and new trajectory part are compared. When new trajectory part is shorter, it is used instead of original trajectory part.

This is repeated in cycle which stops when path is not shortened after many (eg. 150) iterations or optimization is too slow and path is shortened only by small distances (eg. shortening by 5% per 1000 iterations).

PATH RE-SAMPLING

Motion model in RRT-Path algorithm uses constant input in range from 0.5 to 1 second. Smaller interval for constant input causes RRT-Path algorithm to run for too long. When using too short constant input interval, the tree has too many nodes, grows slowly and runs out of memory much faster than longer interval. Interval longer than 1 second makes UAVs unable to manoeuvre between smaller obstacles. Thus range from 0.5 to 1 second was experimentally chosen as best interval. Using x seconds long constant input interval also means $\frac{1}{x}$ Hz frequency of points in resulting trajectory in output of the algorithm. So the range from 0.5 to 1 second implies resulting frequency is in range 1 Hz to 2 Hz.

Real UAVs in Multi-Robot Systems group at CTU use frequency 70 Hz and trajectories with lower frequency are linear interpolated to have frequency 70 Hz. That means frequency 2 Hz is too low for real usage because trajectory generated with this frequency would not be smooth enough.

Change of frequency before the RRT-Path algorithm makes the algorithm unable to run efficiently in bigger maps, so this does not solve the problem.

Another solution is to re-sample the path after Dubins curves. But this method failed because after Dubins optimization, the curves had different length and different constant input durations.

The best solution for this problem is re-sampling of trajectory generated by RRT-Path algorithm before it is optimized by Dubins curves. This solution also has big advantage in Dubins curves optimization because it results to shorter final path as will be shown in following experiment.

7.1 Influence of re-sampling on Dubins curves optimization

To demonstrate the optimization, I created map and let the RRT-Path algorithm find the trajectories for UAVs. The map with trajectories can be seen in 7.1. Obstacles are grey rectangles, AoI is green rectangle and each UAV has trajectory marked with different colour. For measuring of influence of re-sampling of path to Dubins curves optimization, I picked frequencies: 1 Hz (initial frequency used in RRT-Path algorithm), 2 Hz, 4 Hz, 6 Hz, 8 Hz, 10 Hz, 12 Hz, 14 Hz, 16 Hz, 18 Hz, 20 Hz.

As stated above, due to time and memory consumption, each optimization stopped after 150 iterations where optimization did not shorten the path or when speed of path shortening was slower than 5% of original path length per 1000 iterations.

Figure 7.1: Path before Dubins curves optimization

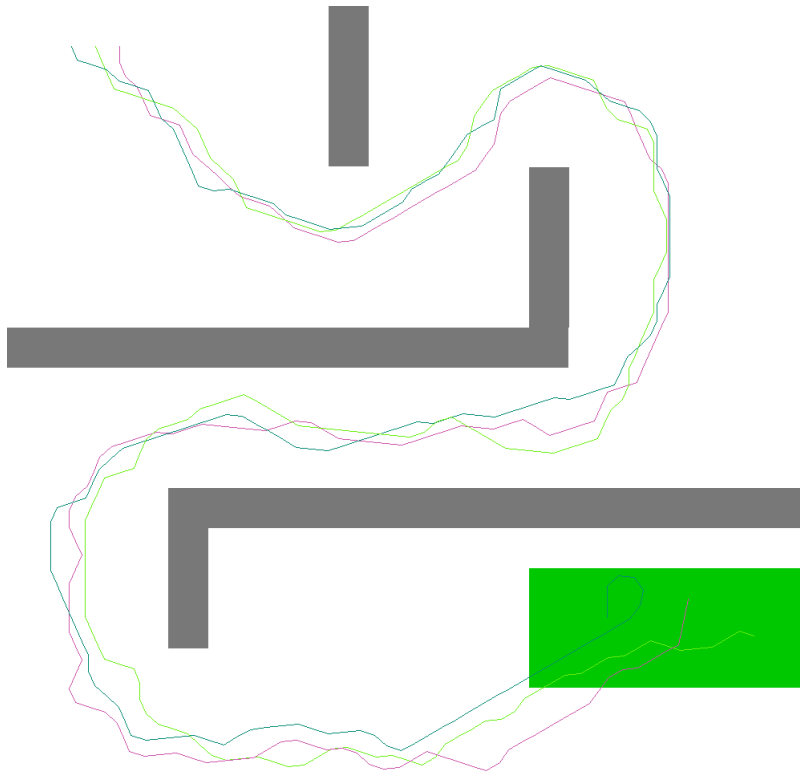


Table 7.1: Re-sampling and optimization results

Frequency of re-sampling [Hz]	1	2	4	6	8	10	12
Minimal total distance [m]	8582.18	8311.65	8366.88	8248.9	8249.88	8286.22	8302.51
Maximal total distance[m]	8849.7	8548.81	8393.09	8275.7	8378.51	8472.2	8309.2
Average total distance[m]	8721.2904	8430.23	8379.985	8262.3	8314.195	8379.21	8307.6613
Frequency of re-sampling [Hz]	Minimal total distance [m]		Maximal total distance[m]		Average total distance		
1	8582.18		8849.7		8721.2904		
2	8311.65		8548.81		8430.23		
4	8366.88		8393.09		8379.985		
6	8248.9		8275.7		8262.3		
8	8249.88		8378.51		8314.195		
10	8286.22		8472.2		8379.21		
12	8302.51		8309.2		8307.6613		
14	8303.18		8303.18		8303.18		
16	8363.92		8363.92		8363.92		
18	8510.32		8510.32		8510.32		
20	8194.22		8194.22		8194.22		

For each frequency, the optimization process was run 100 times to obtain relevant results because of using random numbers during the optimization.

Following table shows average total, minimal and maximal distance of all trajectories from 100 optimizations after the re-sampling and optimization.

The results are also shown in graph 7.2. On the graph we can see that initial frequency 1 Hz has worst results and the frequency 20 Hz has best results. We can also see that in frequency 14 Hz and higher, all 100 iterations had same results, the minimum, maximum and mean value are the same. But the second best frequency in terms of minimal, maximal and mean value is 6 Hz and even the worst optimization in 6 Hz has smaller total distance than 8 to 18 Hz.

Depending on re-sampling frequency, the courses of optimization are also different.

Here we can see mean values and standard deviations for different frequencies, divided to two graphs for better lucidity.

Figure 7.2: Re-sampling and optimization results graph

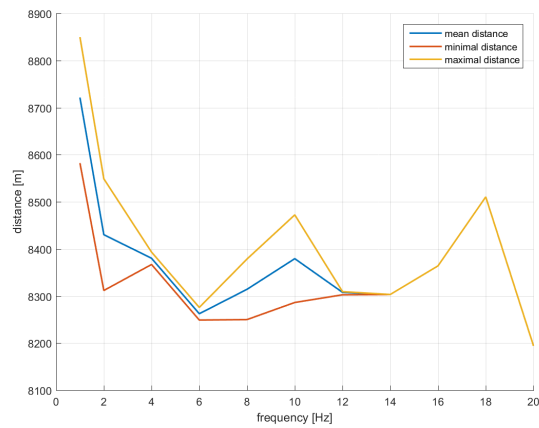
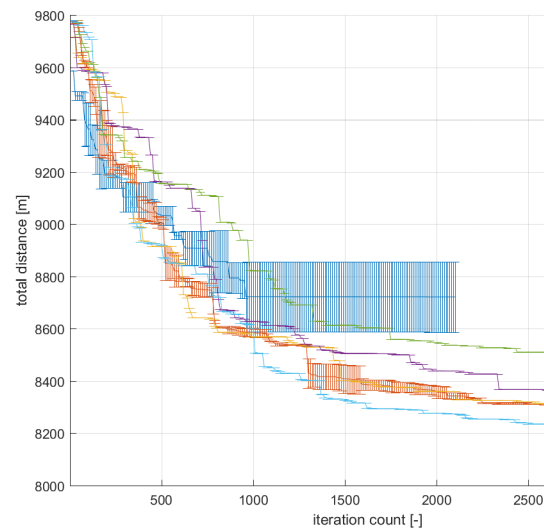
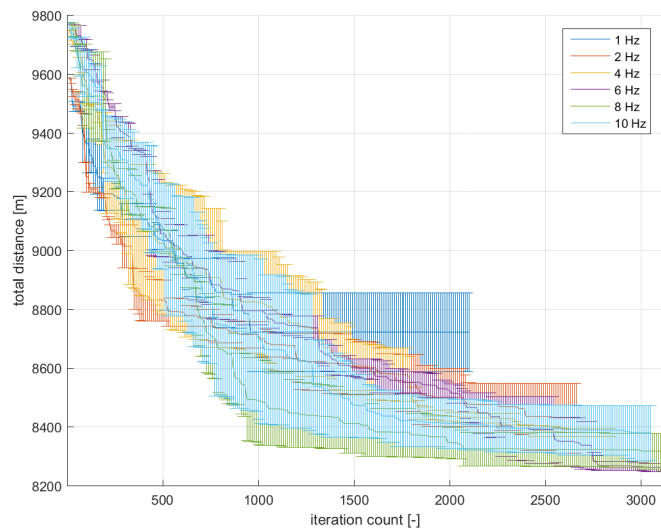


Figure 7.3: Time course of optimization



IMPLEMENTATION

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

8.1 External libraries

In implementation are used some external libraries. Every used library is mentioned here. Boost libraries is used for smart pointers, libraries for Dubins curves are from Master Thesis by Petr Váňa[4]. 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.

8.2 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, which 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 8.1.

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.

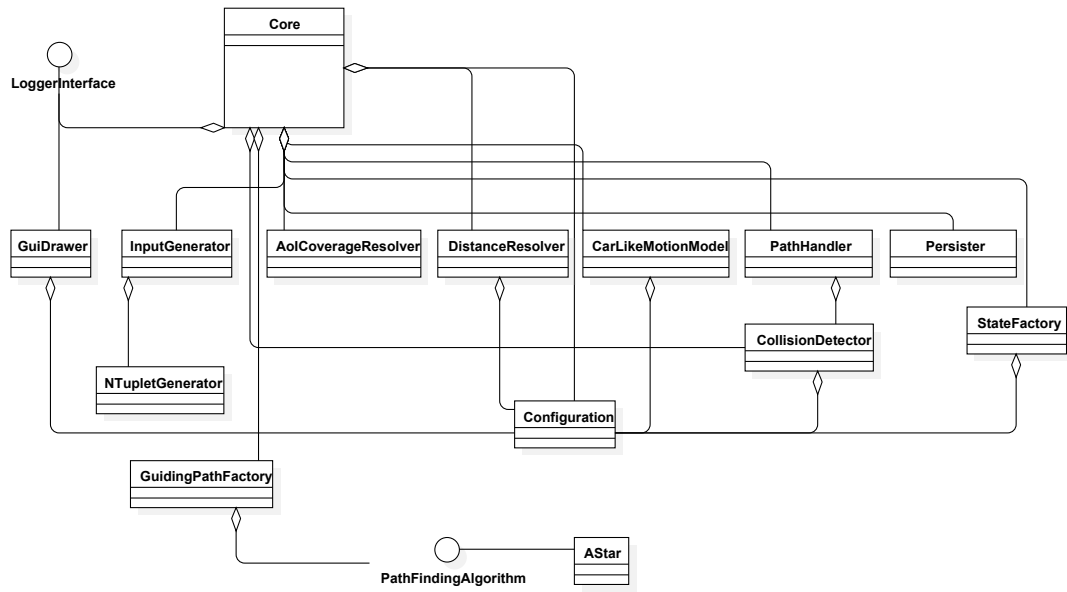
PathHandler serves as utils class for manipulations with path (vector of State classes).

CarLikeMotionModel holds motion model algorithm.

InputGenerator is used to generate inputs to motion model.

NTupletGenerator only generates variation with repeating for given input.

Figure 8.1: 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 AStar class.

BIBLIOGRAPHY

- [1] Lester Eli Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79(3):497–516, July 1957. URL http://www.jstor.org/stable/2372560?origin=crossref&seq=1#page_scan_tab_contents.
- [2] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006. URL <http://planning.cs.uiuc.edu/>.
- [3] Tomáš Krajník Martin Saska, Vojtěch Vonásek and Libor Přeučil. Coordination and navigation of heterogeneous mav–ugv formations localized by a ‘hawk-eye’-like approach under a model predictive control scheme. *The International Journal of Robotics Research*, 33, September 2014. doi: 10.1177/0278364914530482. URL <http://mrs.felk.cvut.cz/data/papers/IJRR2014.pdf>.
- [4] Petr Váňa. Path planning for non-holonomic vehicle in surveillance missions. Master’s thesis, Czech Technical University in Prague, 2015. URL <https://dspace.cvut.cz/bitstream/handle/10467/61814/F3-DP-2015-Vana-Petr-thesis.pdf>.

