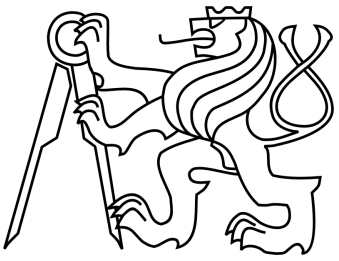
České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra kybernetiky



BAKALÁŘSKÁ/DIPLOMOVÁ PRÁCE

Název práce

Autor: Matěj Račinský

Vedoucí práce: Dr. Martin Saska

Praha, 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 30. prosince 2015 Jméno Příjmení + podpis

OBSAH

KAPITOLA
PRVNÍ

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.

Before converting map to nodes, optional modification of goal for guiding path can be turned on. I will cover the modification in next chapter.

todo: možná sem odkaz na kapitolu

Line 4 calculates the guiding paths for rrt path algorithm using the A* algorithm. Algorithm

Algoritmus 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(rrtPath, map);
```

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.

todo: možná sem odkaz na kapitolu

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

todo: možná sem odkaz na kapitolu

KAPITOLA
DRUHÁ

MODIFICATIONS OF GOAL FOR GUIDING PATH

During this preprocessing (method MapProcessor::getEndNodes in codebase) all AoIs are transformed 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.

KAPITOLA
TŘETÍ

PATHS NARROWING

KAPITOLA
ČTVRTÁ

IMPLEMENTATION

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

4.1 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.

4.2 Code structure

Here is shown brief UML scheme demonstrating dependency diagram of codebase. To keep diagram simple, only services are displayed, other classes were left out. 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 4.14.1.

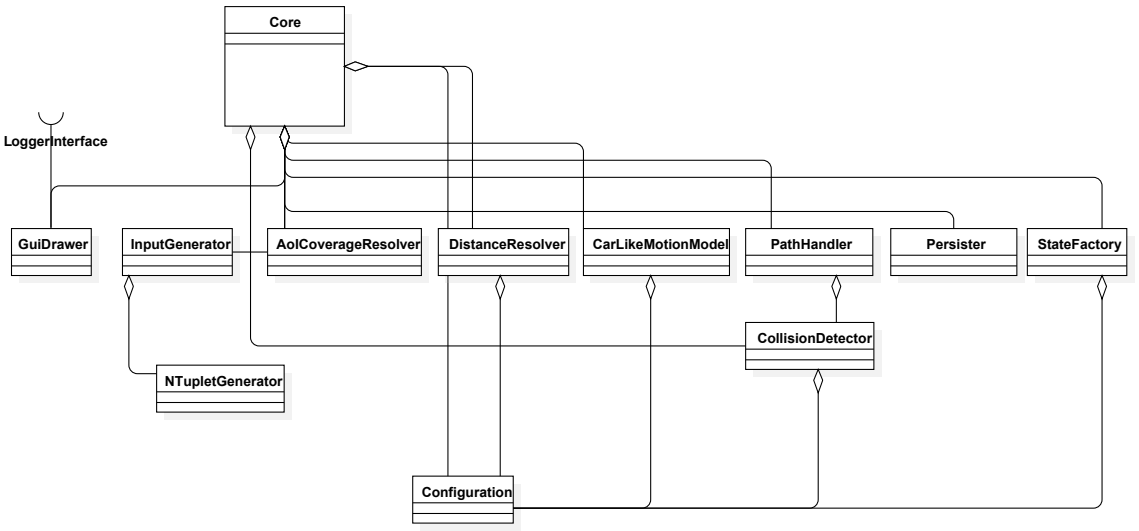
State factory creates State classes according to Factory pattern. State class represents state in RRT-Path algorithm. State has coordinates and rotations for all UAVs.

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.

Obrázek 4.1: Dependency diagram



LITERATURA

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