CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

Diploma Thesis

Robert Pěnička

Motion planning for seabed monitoring by autonomous underwater vehicles

Department of Cybernetics

Thesis supervisor: **Ing. Vojtěch Vonásek**

Prague, 2016

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

DIPLOMA THESIS ASSIGNMENT

Student:	Bc. Robert P ě n i č k a
Study programme:	Cybernetics and Robotics
Specialisation:	Robotics
Title of Diploma Thesis:	Motion Planning for Seabed Monitoring by Autonomous Underwater Vehicles

Guidelines:

1. Get familiar with motion planning [1] and with randomized motion planning methods like RRT [2] or PRM [3]. Implement selected motion planner suitable for motion planning of Autonomous Underwater Vehicle (AUV). Study principles of control of AUVs [4]. Consider motion model of AUV co-developed at Institute of Process control and Robotics, KIT, Germany.
2. Design motion planning for AUV based on Dubins curves.
3. Design motion planning for the docking the AUV to the recharge station.
4. Design mission planning system for up to 3 AUVs in the seabed monitoring task. The goal is to visit all places in a given area, so the seabed is measured by on-board sensors. AUVs can return to the recharging station if necessary.

Models of AUVs and environment will be provided by the advisor.

Bibliography/Sources:

[1] LaValle, Steven M.: Planning algorithms. Cambridge University Press, 2006.
[2] LaValle, Steven M., and James J. Kuffner Jr.: Rapidly-exploring random trees: Progress and prospects, (2000).
[3] Kavraki, Lydia E., et al.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. Robotics and Automation, IEEE Transactions on 12.4 (1996): 566-580.
[4] Fossen, Thor I.: Handbook of marine craft hydrodynamics and motion control. John Wiley & Sons, 2011.

Diploma Thesis Supervisor: Ing. Vojtěch Vonásek

Valid until: the end of the winter semester of academic year 2016/2017

L.S.

doc. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, September 23, 2015

Declaration

Acknowledgements

I would like to thank to my supervisor Vojtech Vonasek for his great guidance through the uneasy waters of writing this thesis. I also greatly appreciate help from David Oertel and Sergej Neumann from KIT regarding the technical details and data for the AUV.
Many thanks go to my family and my girlfriend for their patience and support during long hours spent working on this thesis.

Abstract

This thesis deals with mission planning for multi-robot seabed monitoring task with Autonomous Underwater Vehicles (AUV). The overall goal of this work is to design a mission planner that ensures visiting all places in a given area and keeps the AUVs in operational state by charging from the seabed station. Key part of designed mission planner is also the motion planning. We propose two different randomized motion planners for long distance motion planning and for planning of the docking motion. Presented experiments verify the capabilities of individual motion planners and also the ability to plan the whole seabed monitoring mission with the designed mission planner.

Keywords: autonomous underwater vehicles, motion planning, robot coverage, robot mission planning

Abstrakt

Tato práce se zabývá plánováním misí pro multirobotické monitorování podmořského dna s využitím autonomních ponorek. Cílem práce je návrh plánovače misí, jehož výsledkem je projetí zadané oblasti a nabíjení ponorek z podmořské nabíjecí stanice. Klíčovou součástí navrženého plánovače misí je také plánování pohybu. Byly vytvořeny dva různé pravděpodobnostní plánovače pohybu, první pro plánování na velké vzdálenosti a druhý pro plánování dokování do nabíjecí stanice. Provedené experimenty prokazují schopnosti jednotlivých plánovačů pohybu a také schopnosti plánovače misí najít plán, který pokryje zadanou oblast.

Klíčová slova: autonomní ponorky, plánování pohybu robotu, pokrytí prostoru robotem, plánování mise robotu

Contents

1 Introduction

Autonomous Underwater Vehicle (AUV) is a robotic platform that allows underwater operations without remote control. The AUV can be used for various applications such as deep sea mining, oceanographic research or surveillance of underwater constructions. All of these applications need mainly a precise localization system and a motion planner that can find feasible plans between two places under water. The design of motion planner for the AUV is one of the objectives of this work.

Very often requirement of the AUV applications is a monitoring or scanning of specified area with an on-board sensors. For the sea mining application we need to scan the seabed to find valuable metals. In oceanographic research, the AUVs can for example create a precise map of sea floor or search for marine organisms. For surveillance applications, the AUVs can inspect bases of power plants, submarine cables or any other underwater constructions. This requirement for scanning or monitoring the area leads to what is called a Robot Coverage Path Planning (CPP), which is a problem of determining a path for robot, that passes through all places in given area while avoiding obstacles. Implementation of the CPP for up to three AUVs is also one of the objectives of this thesis.

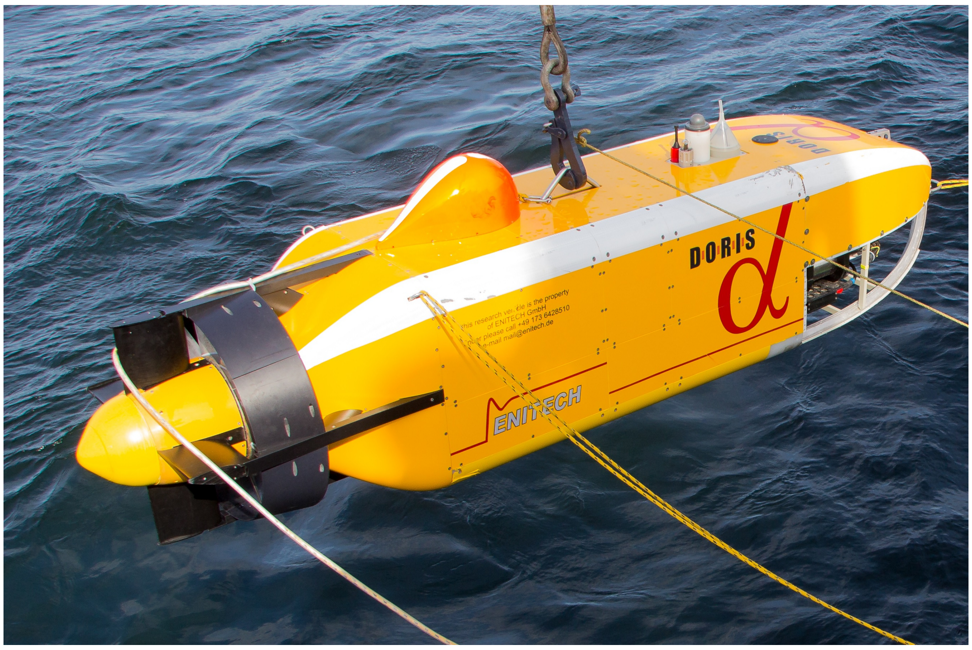


Figure 1: AUV co-developed at Karlsruhe Institute of Technology (KIT) whose model is used in all parts of motion planning for seabed monitoring (courtesy of David Oertel)

The overall goal of this thesis is to design a mission planner for multiple AUVs for seabed monitoring task. Such a task consists of visiting all places in a given area in order to measure the seabed with on-board sensors, with possible returns of AUVs to the recharging seabed station. All key parts of the mission planner for specified task are implemented and described through this thesis.

The first feature the mission planner has to posses is the ability to navigate the AUV through area without collisions. This feature is enabled by the long distance motion planner for AUV. The second necessary feature is the ability to dock the AUV to the recharging station. This feature is enabled by a docking motion planner that stops the AUV next to the docking device. Once we are able to both navigate between places and dock to the docking device for recharging, we can implement the mission planner for seabed monitoring task. In this thesis we designed a mission planner that combines the Robot Coverage Path Planning with a discrete task planning. The resulting mission plan for multiple AUVs contains sequence for visiting of all places in given area and also the necessary returns to the charging station.

Figure 2 shows the structure of the designed mission planner. It also explains why we need to design not only the mission planner, but also the motion planners for long distances and for the docking task.

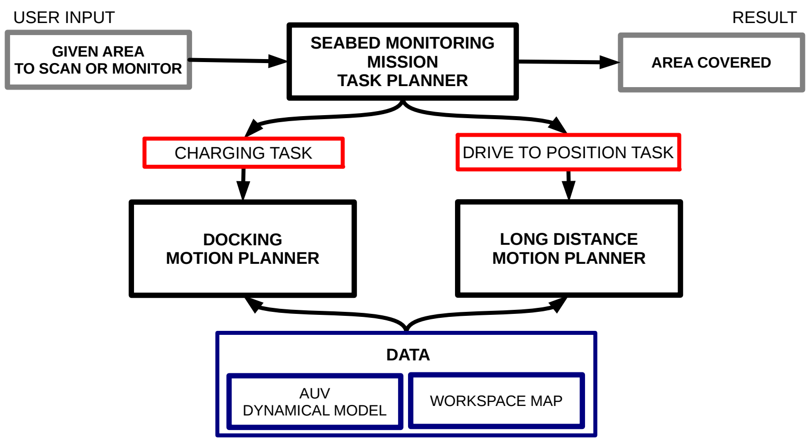


Figure 2: Diagram showing structure of seabed scanning mission planner that consists of task planning, long distance motion planner and docking motion planner

The contribution of this thesis is the novel system for the mission planning with multiple AUVs in seabed monitoring task. The mission planning system consists of three main parts. The first is the designed long distance motion planner for AUVs. Second is the proposed docking motion planner that solves the crucial task of docking to the seabed station. The last and the main contribution of this thesis is the seabed monitoring mission task planner, that utilizes both designed motion planners.

Later in this thesis we firstly describe and compare current approaches to the robot path and motion planning in Chapter 2. Then in Chapter 3, we describe current approaches for coverage path planning, because it is essential part of the designed mission planner. The necessary feature implemented for the overall goal are the AUV motion planning and docking. Chapter 4 describes designed AUV motion planners for both long distance travels and for AUV docking task. Finally, the mission planner for seabed monitoring by multiple AUVs is introduced in Chapter 5. Experimental verification of mission planner is described also in Chapter 5.

2 Robot Path and Motion Planning

Path planning and motion planning are one of the crucial tasks in robotics. The classical planning can be described as *Piano Mover's Problem*, where the goal is to find a path that moves the piano between two places inside house without hitting the walls. Other applications where planning has key role are for example autonomous cars [24], puzzle solving [28], protein molecule docking [26] and of course the Autonomous underwater vehicles [2].

Both planning problems require that robot shape and surrounding workspace are known. It means that we need a map of the environment where the robot moves, otherwise we are unable to determine how the robot is supposed to move without hitting the obstacles.

The path planning is simpler task than the motion planning. It does not require to consider the dynamics of transition between two positions. The path planning searches merely for sequence of positions that connects the start and goal positions.

The motion planning uses a motion model of the robot in order to to find a control actions that lead the model from start configuration to the goal configuration. For the *Piano Mover's Problem* one would seek how the movers have to use their arms to get the piano to its goal configuration. This also requires to consider the moments of the arms so that the movements can be executed for example on robots.

More analytically, the path and motion planning problems are defined as follows [28]. We consider robot to move inside either 2D or 3D space that we call *workspace* ($\mathcal{W}$), where $\mathcal{W} = \mathcal{R}^2$ or $\mathcal{W} = \mathcal{R}^3$. Inside the workspace one can find robot $\mathcal{A} \subset \mathcal{W}$ and j obstacles $\mathcal{B}_1 ... \mathcal{B}_j \subset \mathcal{W}$. The robot can be described with its configuration $\boldsymbol{q} \in \boldsymbol{C\text{-}space}$, which is a vector with k parameters whose number equals to degree of freedom (DOF) of such robot. Let p_A be position of the robot inside the workspace, then the position is function of actual configuration $p_A = f(q)$. All possible configuration of robot creates a *configuration space* $\boldsymbol{C\text{-}space}$. The configuration space could be divided in two subsets. The first subset $\boldsymbol{C_{obs} = \{q \in C\text{-}space | A(q) \cap B_i \neq \emptyset\}}$ consists of all configurations where robot collides with obstacles. The expression $A(q) \subset \mathcal{W}$ denotes space that is occupies by the robot in configuration q. The second is obstacle-free space $\boldsymbol{C_{free}}$ that is complementary to the first, $\boldsymbol{C_{free} = C\text{-}space \setminus C_{obs}}$. Equally we can also divide the workspace into parts that are collision-free and that are not.

The robot can change its configurations by using actions $\boldsymbol{a}$ such that $q_{i+1} = q_i + f_{mod}(q_i, a_i)$, where $f_{mod}(q, a)$ represents the motion model of the robot. For formulation of the path planning and motion planning tasks we also need a start configuration $\boldsymbol{q_{start}} \in \boldsymbol{C_{free}}$ and a goal configuration $\boldsymbol{q_{goal}} \in \boldsymbol{C_{free}}$.

The path planning is then formulated as task to find plan $\boldsymbol{P = \{p_{A1}, p_{A2}, ..., p_{An}\}}$ as sequence of collision-free positions of robot inside the workspace, where the positions $p_{A1} = f(q_{start})$ and $p_{An} = f(q_{goal})$. In other words, the path planning generates sequences of positions between start and goal configurations while avoiding collisions.

In the motion planning, we focuses more on the actions that changes the configuration. The plan can be represented as sequence of actions that lead the start configuration q_{start} to the goal configuration $q_{goal} = q_{start} + \sum_{i=1}^{n} f(g_i, a_i)$.

Both motion and path planning were introduced as discrete time, but both can be also performed in continuous time, where the plans are functions of positions and actions rather than the sequences as in the discrete time version.

Despite the fact that motion planning and path planning have different formulations, we can often switch between each other. With plans from path planner that obey the dynamical restrictions of robot, we can usually generate motion plans from dynamical model knowledge. On the other hand, we should be always able to calculate robot positions from it's configuration in order to generate path plan from the motion plan.

In next two subsections are presented overviews of current approaches to path planning and motion planning.

2.1 Basic Path Planning

Basic path planning algorithms are one of the oldest in field of robotics, but they are still very widely used because of their simplicity and robustness [28]. All presented methods have in common that they simplify the world description by using graphs. The task to find path between start and goal configurations is then solvable by using classical graph search methods like Depth First Search (DFS), Breadth First Search (BFS), Dijkstra and A*.

Based on their approaches to the path planning task, we can divide the basic path planning into three groups a **Cell Decomposition**, **Roadmap** and **Potential field**.

The **Cell Decomposition** approach splits the free configuration space into many simple regions. The decomposition could be either exact, where the free space is union of all regions, or approximate, where the regions have predefined shape and some regions can be partly covered by obstacles. The decomposed cells are then considered as graph vertices and each shared border of cell makes edge between those vertices. In such a graph, the path planning task is done by searching path in the graph between cells where are the start and goal positions.

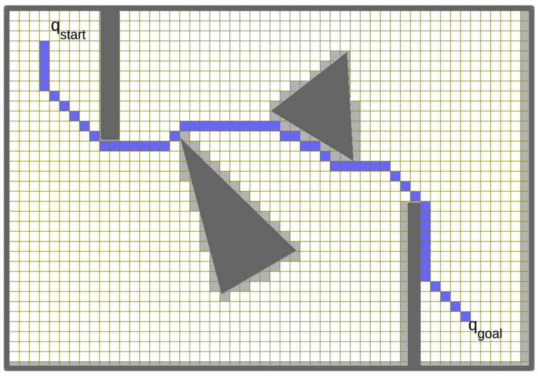
Triangulation of the free space [16,49] is one possible way of decomposition, that covers the space by different triangle shaped cells. Alternatively the free space can be decomposed into trapezoidal cells instead of triangular. For both Triangular and Trapezoidal decompositions a polygonal obstacles are required. This means that boundary of every obstacle $\mathcal{B}_j \subset \mathcal{W}$ must be describable as polygon.

Grid-based approximate decomposition is one of the easiest path planning techniques and is also very widely used [32, 45]. The main idea behind grid-based decomposition is covering the workspace by equal rectangular cells. The simplicity of this method is due to easy decomposition and representation of the grid by arrays. Also the planning could be applied directly to sensory based occupancy grid map without any additional computation of map that consists of polynomial obstacles. In Figure 3a we can see example path plan found using the grid-based decomposition and the graph search. In cases where obstacles are close to each other, we can also use Quadtree (2D) or Octrees (3D) decompositions. These methods further divide each partially occupied cell in order to get more precisions around obstacles.

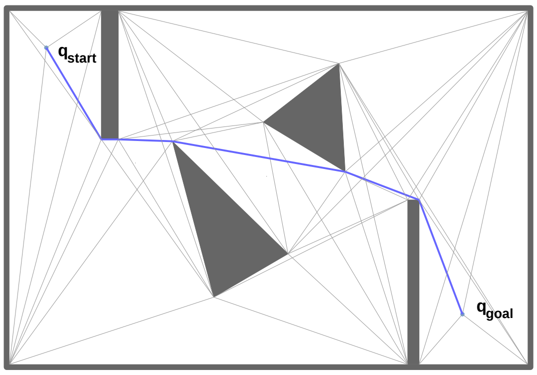
The **Roadmap** approach firstly finds curves that lies in the collision-free workspace and connects them together into the Roadmap. Then the robot moves only along the roads (found curves), which ensures that the generated path is without collision. To find path from the start configuration to the goal configuration we also use the graph search, where edges are the roads and vertices are the intersections between those roads.

Visibility graph (VG) [28, 34] uses polygons for describing the obstacles. The roadmap in VG consists of all collision-free lines between vertices of obstacles and vertices of start and goal positions. As we can see in Figure 3b, the VG approach computes the path as close to the obstacles as possible. In order to make the generated path collision-free, we must consider bigger obstacles for example by using Minkowski sum.

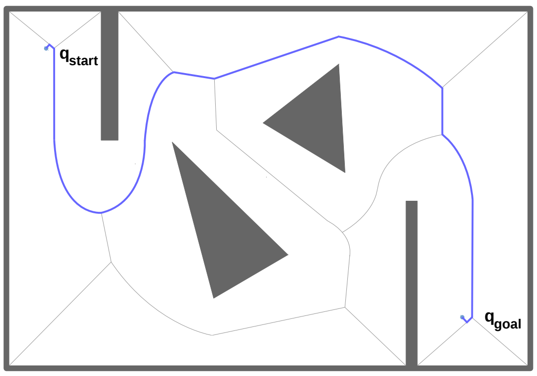
In contrast to the VG path planner, a *Voronoi diagram* approach, shown in Figure 3c, generates paths that are as far from the obstacles as possible. The Voronoi diagram is a set of line segments and polynomial curves that are made of points with same distance from at least two obstacles [4]. On the other hand the paths obtained from Voronoi diagrams are really not the optimal ones because of the maximal possible obstacle avoidance of the method.



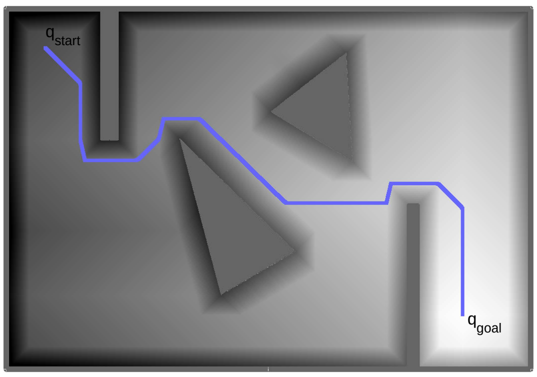
(a) Grid-based



(b) Visibility Graph



(c) Generalized Voronoi diagram



(d) Potential Field

Figure 3: Examples of basic Path Planners

The **Potential field** approach uses artificial potential field established in the configuration space, where the robot moves from start along a negative gradient to the goal with the minimal potential [43]. The potential field usually consists of two sources. The first is attractive potential that pulls the robot from start to its goal. The second potential is repulsive, which is used around obstacles in order to get the robot away from them. Example of path planning with potential field with both attractive and repulsive potential is shown in Figure 3d.

The main disadvantage of Potential field approaches is possible stuck in local minima. But even those problems can be overcome by some methods like introducing artificial obstacles in local minima [38] or by using navigation function that has only one global minima.

2.2 Randomized motion planning

In basic path planning task we searched for collision-free positions inside the workspace $\mathcal{W}$, which is usually $\mathcal{R}^2$ or $\mathcal{R}^3$. The motion planning that searches in configuration space tends to be a more difficult problem. The first reason is that the obstacles inside configurations space C_{obs} cannot be described as easily as in workspace. This is why we use a collision detection algorithms that decide whether specified configuration has collision position or not. The collision detection uses geometry of robot in specified configuration $A(q)$ and returns whether it has intersection with any obstacle B_i. The second reason why motion planning is usually more complex is that it needs to search in configuration space that has higher dimension than the workspace.

The most naive motion planner consists of grid-based planner that uses configuration space instead of workspace. Such approach would find motion plan, but is not applicable for most robotic systems, because the dimension of configuration space is so high that it is not possible to store the grid or compute the plan fast enough. Answer to this problem is randomness. By using random samples of configuration space we are able to generate motion plans without searching through the whole configuration space.

One of the first randomize motion planner was *Randomized potential field Path Planner (RPP)* [1]. The RPP uses potential field approach for motion planning. The main difference from classical potential field planner is usage of random motions (random walks) with increasing length in order to get from local minima.

Nowadays there are two mostly used randomized motion planners, Probabilistic Roadmaps (PRM) [29] and Rapidly Exploring Random Trees (RRT) [23]. Those two algorithms are discussed more deeply in next two sections.

2.2.1 Probabilistic Roadmaps

The Probabilistic Roadmaps (PRM) [23] is very popular motion planning algorithm for robots in static environment. It shares the idea of generating the roadmap as had the basic roadmap path planners. We can divide the algorithm in two phases.

The first *learning phase* creates the roadmap inside the configuration space by random sampling of the configuration space. Outcome of this phase is graph consisting of vertices as points in C_{free} and edges that represents collision-free path between configurations.

Algorithm 1: PRM learning phase

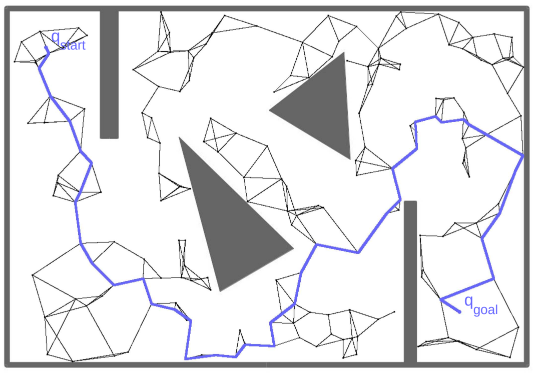
```
input : Configurations $q_{start}$ and $q_{goal}$ , number of nearest neighbours $k$ and
        number of generated nodes $N$.
output: Roadmap as graph G=(V,E)

1  $V = \emptyset$;
2  $E = \emptyset$;
3  while $|V| < N$ do
4      $q_{rand} = random\_configuration()$;
5      if $q_{rand}$ is collision free then
6          $V = V \cup \{q_{rand}\}$;
7      end
8  end
9  foreach $q \in V$ do
10     $S_q = k$ nearest neighbours from $V$ to $q$
11     foreach $q' \in S_q$ do
12         $e = (q, q')$;
13         if $e \notin E$ and $e$ is collision free then
14             $E = E \cup \{e\}$;
15         end
16     end
17 end
```

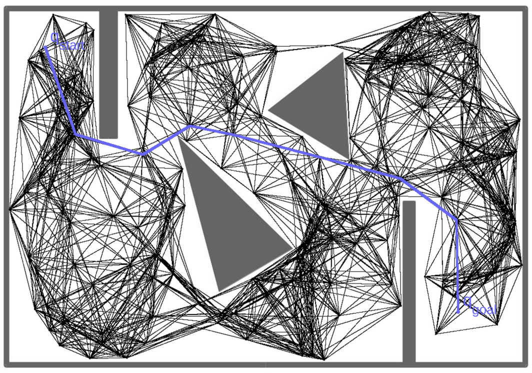
As we can see in the Algorithm 1, we firstly generate nodes of the roadmap until we have N vertices from C_{free} space. Then we find k nearest neighbours to all generated vertices and use a local planner between the vertex and it's neighbours. Such local planner generates path between two configurations in *C-space*. If there is a path that is collision free, then the path is added to the roadmap.

The main influence on the PRM planner performance has the two parameters k and N. The number of generated random points N usually determine whether the plan is found or not. By undersampling of the *C-space*, we may not be able to connect all parts of the space, for example because of some narrow passages. With higher number of generated vertices we may find shorter paths, as in Figure 4d, but with small number of nearest neighbours k we can also end up in situation where roadmap consists of many disjoined graphs like in Figure 4c.

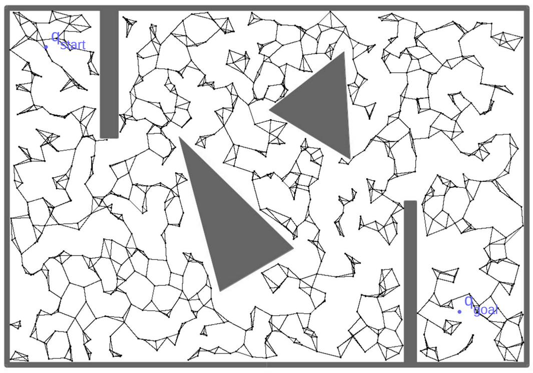
The PRM algorithm could be used both for motion planning and path planning, where instead of using configuration space, we use the workspace. For both purposes, the local planner is also very important. In Figure 4 are displayed roadmaps generated with PRM as path planner with straight-line local planner. In this type of local planner we use line segment that simply connects two configurations.
Dubins curves local planner is shown in Figure 5. In that case a Dubins curves (discussed in Chapter 4.2.2) are used to connect positions (x, y, φ) in 2D workspace.



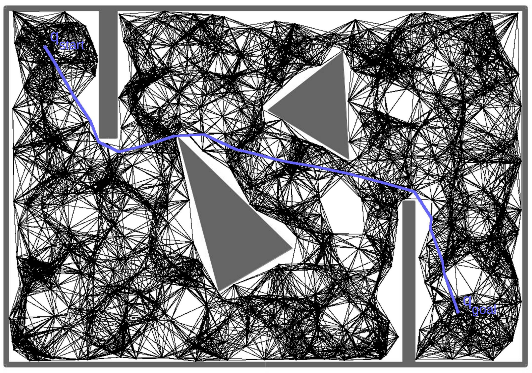
(a) Roadmap with lower density of points and small number of nearest neighbours ($N = 200$, $k = 3$)



(b) Roadmap with lower density of points but with more nearest neighbours($N = 200$, $k = 20$)



(c) Roadmap with many disjoined parts ($N = 1000$, $k = 3$)



(d) High density roadmap with high number of nearest neighbour ($N = 1000$, $k = 20$)

Figure 4: Influence of number of neighbours k and number of generated points N on the roadmap

After we have the roadmap from the first learning phase, the second *query phase* follows. During this phase we connect the start and goal configurations with the roadmap in same manner as the random points in previous learning phase. After connection of the start and goal to the roadmap we can search the graph using a Dijkstra's algorithm. The output is the shortest path in the roadmap between start and goal configurations. Possible failure of the Dijkstra's algorithm is because the start and goal configurations are in mutually separated graphs. The reason for that is either absence of any solution or inappropriate selection of parameters k and N.

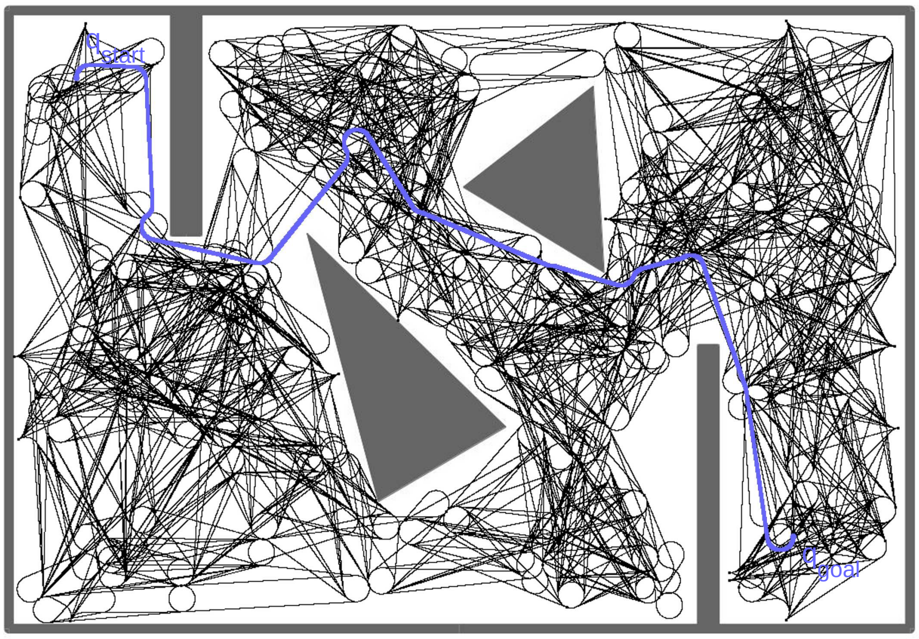


Figure 5: Path found by PRM with Dubins curves local planner

2.2.2 Rapidly Exploring Random Trees

The rapidly exploring random tree (RRT) [29] is also very popular algorithm for motion planning. The main idea behind the method is to grow a tree graph inside the C_{free}, which is rooted in the start configuration. The tree is expanded randomly until it reaches the goal configuration. In Algorithm 2 is shown the RRT without any modifications.

Algorithm 2: RRT

input : Configurations q_{start} and q_{goal} , maximal allowed final distance to goal d_{goal} and maximal number of iterations N.
output: Sequence P of actions(positions for path planning) or failure

```
1  $TREE.add(q_{start})$;
2  $iteration = 0$;
3  while $iteration < N$ do
4      $q_{rand} = random\_configuration()$;
5      $q_{near} = nearest\_neighbour(TREE, q_{rand})$;
6      $TREE.expand(q_{rand}, q_{near})$;
7      $d = distance(TREE, q_{goal})$;
8      if $d <= d_{goal}$ then
9          $P$ = backtrack from $q_{goal}$ to $q_{start}$
10         $returnP$
11     end
12     $iteration = iteration + 1$;
13 end
14 $returnfailure$
```

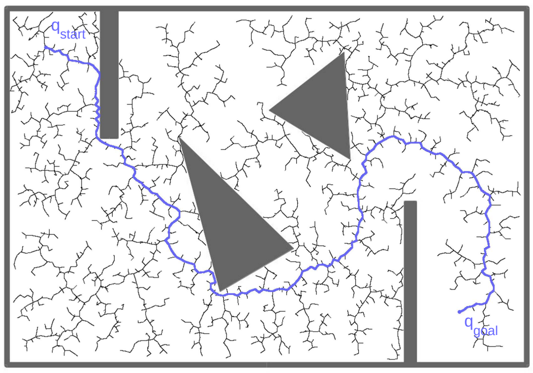
Besides the goal and start configuration, the algorithm uses two other input parameters. The maximal allowed final distance to goal d_{goal} defines when any tree node is considered as the goal configuration. It also depends on which metric we use in *C-space*, but when the distance of some tree node from the goal is lower or equal to d_{goal}, then the goal is considered as reached and the algorithm ends. The maximal number of iterations N in fact limits the maximal time we want to spend on search for the plan. When the plan is easy to find, the algorithm does not iterate after the goal is reached, but when the algorithm does not reach the goal configuration in N iterations it reports failure. The problem in such case, when the failure is reported, is that we can not decide whether the possible plan exists or the number of iteration N was not enough. Because of the probabilistic completeness of RRT we could decide about existence of plan only when the algorithm would iterate for infinite time.

After initializing the tree with start configuration as a root, we iteratively expand the tree. During each iteration we generate random configuration q_{rand} from *C-space*. To the generated configuration we find nearest neighbour q_{near} from the already created tree. During the expansion $expand(q_{rand}, q_{near})$ we find path between nearest neighbour and the random configuration. If the path length is more than d_{add}, then instead of random configuration we use configuration on the found path in d_{add} distance from the nearest neighbour. Afterwards we also check the path for collisions and if the path is collision-free then we add the random configuration as a child node of the nearest neighbour. By this manner, the tree is in each iteration expanded maximally to additional distance d_{add}. The algorithm terminates when the expanded configuration is in maximal distance d_{goal} to the goal or the failure is reported after N iterations.

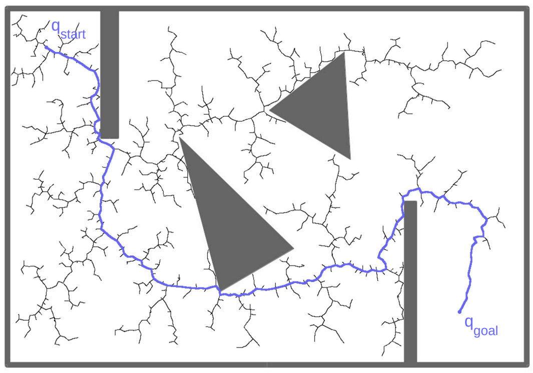
The RRT method could be used for both path planning and motion planning. The main difference is in expansion of the tree. In path planning we use for example straight-line or Dubins curves local planner between random and nearest neighbour positions. For motion planning we must take in account the motion model $\Delta q = f_{mod}(q_{nn}, a)$ and the expansion is done by using some action a from the nearest neighbour configuration q_{nn}.

Modifications of RRT

Very common modification of RRT to improve its speed is to attract the growth of the tree to direction of the goal. This modification called *goal-bias* [46] is done by changing the $random\_configuration()$ function to return also the goal configuration with probability p_{goal}. By this manner, the tree has tendency to grow in direction of the goal. In Figure 6 is shown how the usage of non-zero probability p_{goal} influences the planner performance. As we can see, the non-zero goal-bias decreases the iterations needed for finding the plan nearly by half. On the other hand by biasing the goal too much, e.g. $p_{goal} = 0.8$, it would take much more iterations to find the plan, because the tree would grow mainly around line between start and goal. Unfortunately this line is not collision free, so the tree would try to expand very frequently to the configurations with obstacle.



(a) RRT with no attraction to the goal, path found in 2933 iterations



(b) RRT with $p_{goal} = 0.1$, path found in 1462 iterations

Figure 6: Influence of goal-biasing to RRT algorithm

As for PRM planner we can use different types of local planners between randomly generated points and the nearest neighbours from the tree. In Figure 6 we used straight line planner as the simplest local planner. However the straight line planner is probably the most frequently used, there are also many systems with dynamical constraints where it can not be used. For example nonholonomic mobile robots has to use circular paths with no sharp turns. In Figure 7 we used the Dubins curves local planner.

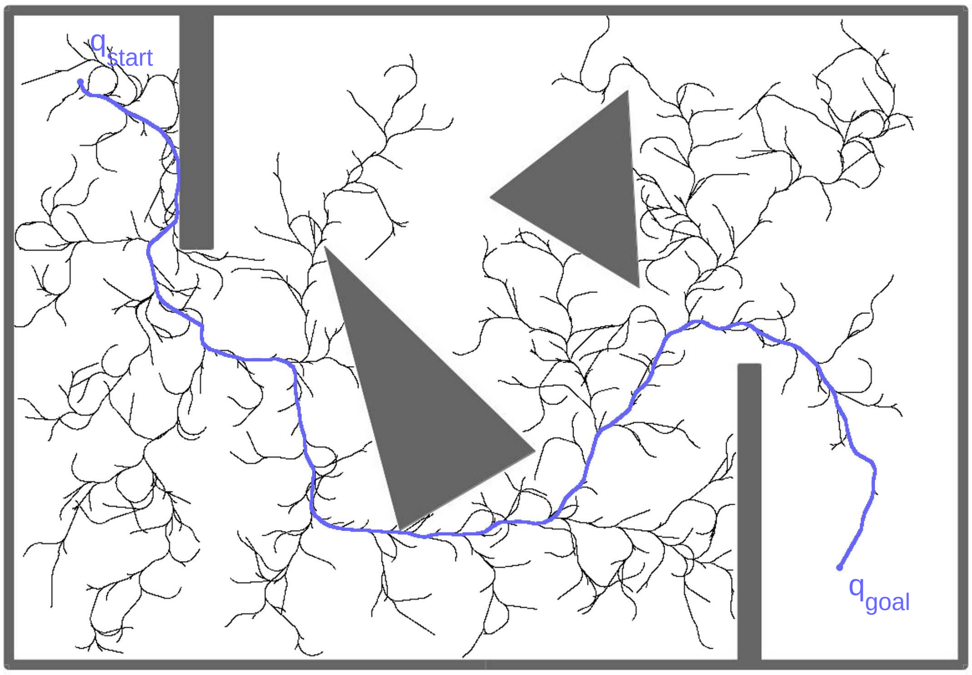


Figure 7: Example of RRT

Besides modifications of goal-bias and the local planner we can also find other interesting RRT approaches that modifies usually the expansion method.

In RRT-Bidirect [31] is firstly used idea of expanding two trees simultaneously one from q_{start} and the second from q_{goal}. The main change is in expansion step, where the same randomly generated configuration is used for both trees. Firstly the tree rooted in q_{start} is expanded by random configuration and if the expansion succeeds (is without collision), then also the second tree tries to expand to the same configuration. The growth of the trees is terminated if their distance is less than a threshold d_{goal}. Then, the trees are connected and the path/trajectory is found as in basic RRT. The connection of the trees is not a trivial task. While for path planning and holonomic mobile robots the interconnection is easy because the trees are connected simply by a straight line, for the robots with dynamical constraints the connection is nontrivial and requires further computation.

RRT-Connect [27] modifies the expansion step in such way, that the randomly generated configuration is used repeatedly until the configuration is reached or some obstacle is hit. By this manner the planner creates trees with long straight paths rather than tree with many tree branches as in classical RRT. This method could be also combined with using RRT-Bidirect approach with multiple trees.

Another very interesting idea is used in RRT-blossom [21]. In contrast to the original RRT, this method always tries to expands into multiple configurations. For motion planning it only requires to sample multiple possible actions a and use them in the motion model from selected nearest neighbour. Such generated configurations are added to the tree only if their distance to the nodes already present in the tree are lower than distance to their parent. This requirement called regression is crucial in order not to stuck in already explored space.

2.2.3 Pros and cons of sampling-based planners

The main advantage of randomized motion planning is that it can handle even very complicated planning problems in high dimensional spaces, that are not solvable by basic path planning methods. Both presented randomized motion planners share the basic idea of random generation of configurations $q \in C_{free}$ and connection of these configuration in order to get plan between start and goal configuration. The PRM differs from the RRT in precomputation done in the learning phase. Once we have the roadmap, we can use it repeatedly to find plans between different start and goal configurations. This advantage is mainly because the most time-consuming task in the whole algorithm is the search for the nearest neighbours and the collision detection. The problem arises when the query phase fails and we are forced to run the learning phase again with possible bigger k and N parameters. On the other hand, the RRT is only single query motion planner. The iterative growth of RRT planning tree could continue indefinitely until the plan is found and does not share the complication of rerunning the learning phase as PRM. The RRT is also probably more suitable for robots with dynamical constrains because when expanding the tree, the motion is done from a tree node to the new node. In the PRM, when connecting two configurations we can not determine which configuration of the motion is initial and which is terminal.

The downside of randomized motion planning is that the optimality of generated plans cannot be usually guaranteed. As the ***C-space*** is sampled randomly, the path between start and goal configurations are generated longer and their optimality is not guaranteed in both methods. PRM* and RRT* [22] are modifications of PRM and RRT planners that are asymptotically optimal. It is achieved by considering also cost of connection between points inside ***C-space***. RRT* addresses the important aspect of optimality in the RRT motion planning task. The optimality of RRT* is done by considering cost of expansion. When we generate new configuration q_{new} in the expansion step by steering the nearest neighbour configuration q_{near} in direction of q_{rand}, we also checks multiple nearest neighbours of the q_{new} in order to get the neighbour with the lowest cost of path to q_{new}. After connecting q_{new} to the node with lowest cost of path, all previously selected nearest neighbours could be reconnected in order to get smaller costs of connections.

Optimality of the plan could be also improved by post-processing [36] of the generated plan. The smoothing of the plan is the simplest improvement, yet it is very useful because the plans generated with probabilistic path or motion planning tends to be very jagged.

For the two already presented randomized planners, we have guarantees of probabilistic completeness, which means that they find the plan if one exists. In reality, the plan could be so difficult to find, that it takes infinite time to find it. This hardness of finding a plan is usually caused by what is called *narrow passage problem*. This problem is shared among all sampling-based methods. The narrow passages are parts of configuration space that are relatively small, the robot fits inside them very tightly, and their removal would change connectivity of the free space. With uniform sampling of the configuration space, those passages a usually not sampled enough to get through them. One possible technique to improve performance in narrow passages is base on their detection and adaptation of sampling inside them [41, 42]. Other possibility to overcome narrow passage problem is retraction-based planning [52], that uses not only samples generated in free configuration space, but also the samples inside $\boldsymbol{C_{obs}}$ and moves them to the closes point in $\boldsymbol{C_{free}}$ space.

The randomized motion planning is also suitable for the AUVs. As the AUV has nonholonomic drive and its dynamical model has its constrains, the basic path planning methods are not able to handle it. In Chapter 4 are presented several motion planners that use the RRT approach.

3 Robot Coverage Planning

Robot Coverage Path Planning (CPP), also known as *Robot Coverage Problem*, is task of finding a path through the free space that passes over all points in a given area [30]. Applications of CPP can be found for example in autonomous lawn mowing [48], vacuum cleaning [33], agriculture [18], area inspection or autonomous underwater vehicle scanning of the seabed [14]. The crucial part of any algorithm that solves the CPP is not only coverage of all selected points of space, but also optimality of the path. This optimality is mainly done by searching for a path without overlapping parts and repetition of paths. This optimality requirement is related to the well-known Travelling Salesman Problem (TSP) [47], whose goal is to visit each city exactly once, use the shortest possible route and return to the original city. As we know, the TSP is a NP-hard problem which means that also the coverage planning is NP-hard. This is why the approaches to solve the CPP problem usually prefer the requirement for covering all selected points before requirement for the optimality.

Generally we can divide the algorithms that solve the coverage problem to online and offline. The offline algorithms use static information about the environment and do not assume any change of the environment. The online coverage also utilizes sensors placed on the robot in order to cover the area. Further we can also distinguish between heuristic approaches, that do not guarantee the full coverage, and complete approaches that have proven coverage of the whole free space.

In following two sections, Single Robot Coverage and Multirobot Coverage, are presented selected methods for solving CPP. Methods are organized based on previous surveys in robot coverage planning [7, 15] and also based on their similarity in dealing with the coverage problem (especially how the workspace is partitioned).

3.1 Single Robot Coverage

Single robot coverage uses only one robot for covering the area. Advantages of using only one robot are lower costs and no need for cooperation of multiple robots. The algorithms presented bellow focus mainly on the completeness and optimality of the coverage plans. Differences between individual algorithms are mainly in way how they partition the workspace and also in the way how these parts of workspace are selected during the coverage. Disadvantages of using single robot for coverage are longer coverage time and vulnerability to robot failures.

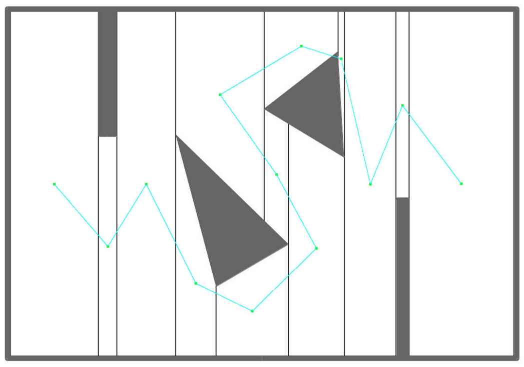
3.1.1 Exact Cellular Decomposition Methods

The exact cellular methods uses very similar idea as the exact decomposition path planners mentioned in section 2.1. The free area for coverage is decomposed into smaller cells that covers the whole area. From this decomposition we can create a graph, where nodes are the cells and the vertices between two cells represent their adjacency. Then each cell can be covered by simple plow-like sweeping algorithm that drives the cell using zigzag path as in Figure 9a.

Trapezoidal decomposition

This type of decomposition for coverage planning is the same one as for the path planning mentioned previously. Trapezoidal decomposition is capable to cover only workspaces with polygonal obstacles. Its outcome are trapezoidal or in some cases also triangular cells. The graph of adjacency is created from the cells. Then using exhaustive walk through the graph, e.g. with Depth-first search (DFS), we can determine in which order the cells are covered. In Figure 8a we can see trapezoidal decomposition of workspace together with lines between cell centres that indicate order in which the cells are covered. The final plow-like path that robot uses in order to cover the workspace is in Figure 8b.

Weak point of this method are connections of cells where we sometimes have to use already covered paths or backtrack in order to get from the end of plow-like path in first cell to the start of plow-like path in second cell.


(a) Trapezoidal decomposition coverage with order of cell coverage

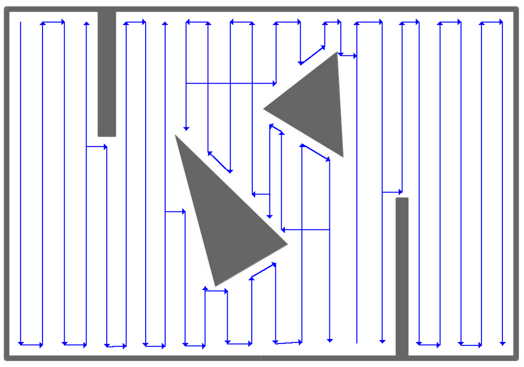
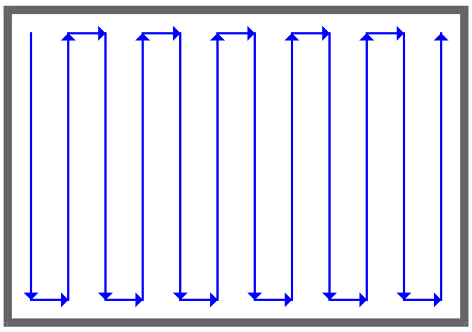

(b) Trapezoidal decomposition coverage robot path

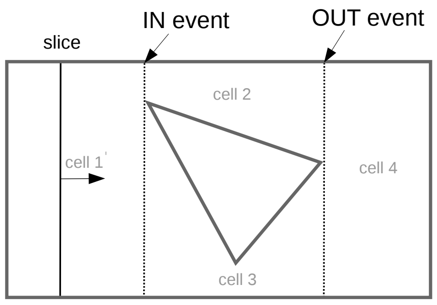
Figure 8: Trapezoidal decomposition coverage

Boustrophedon decomposition

The word boustrophedon comes from Greek and it means “the way of the ox”. This name refers to way how one cell is covered by plow-like sweep algorithm, exactly the same method as trapezoidal decomposition used 9a. The boustrophedon decomposition coverage [8,37] is quite similar to the trapezoidal decomposition, but it generates lower number of cells. The cells are determined by using a vertical line called *slice* that sweeps the workspace from left to right and detects events **IN** and **OUT**. The **IN** event is detected when the slice line is divided by obstacles into more parts than was before. This happens when new obstacle is reached. **OUT** event is when multiple parts of slice line joins together, which usually happens when the line gets out of the obstacle. On places where those evens occur we end previous cells and start new cells.



(a) Plow-like coverage of one cell



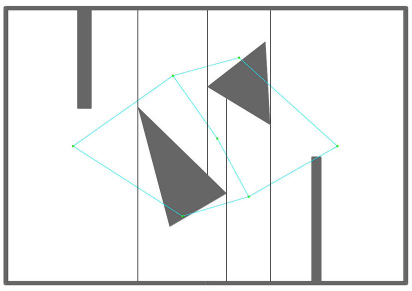
(b) Vertical line (*slice*) sweeps the workspace

Figure 9: Boustrophedon decomposition principles

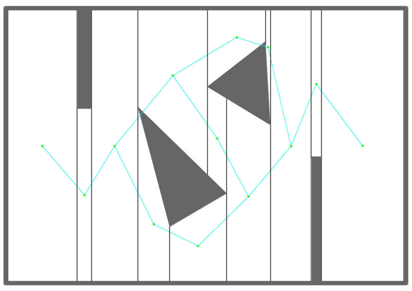
Finally, after the cells are created, their order is determined as in trapezoidal decomposition with an exhaustive walk through the graph.

The boustrophedon decomposition produces less number of cells than the trapezoidal decomposition, which means that search through graph is easier. Also problematic parts of plan where a robot has to traverse between two cells through already covered area are reduced.

In Figure 10 are shown difference in decompositions by using boustrophedon and trapezoidal decompositions. Even the cells are created little differently, the final robot path is in both cases almost the same.



(a) Boustrophedon decomposition



(b) Trapezoidal decomposition

Figure 10: Comparison of Trapezoidal and Boustrophedon Decompositions

Morse-based cellular decomposition

Both the Trapezoidal decomposition and Boustrophedon decomposition could be done by simple line sweep algorithm [20] as shown in Figure 9b. While the Boustrophedon decomposition only detects changes of connectivity of the line, in the Trapezoidal decomposition we also detect changes in shortening and lengthening of the parts of slice line that passes through the free space.

Last method presented among exact cellular decomposition methods is Morse-based cellular decomposition [6]. This method is in fact generalization of Boustrophedon decomposition. In previously mentioned Boustrophedon decomposition we used line slice to determine cell borders in places where the connectivity of the slice changes. The same approach is used in Morse-based decomposition with change that the slice could vary in its shape. We can define the slice as real valued function $h : R^m \to R$. In the Boustrophedon decomposition the slice was line defined as $h(x, y) = x$.

The Morse-based decomposition is named after morse functions that have in common that their critical points (places with zero differential) are isolated from each other. This feature is critical to find the decomposition, because the cell borders are placed where slice connectivity changes which is in critical points of the morse function that sweeps the workspace. It means that we can define the slice arbitrarily as long as it is morse function.

The advantage of Morse-based decomposition is that the obstacles could be of any shape and not only polynomial. We can also define the morse-based decomposition in 3D space.

In Figure 11 we can see morse-based decomposition using concentric circle slice defined as $h(x, y) = \sqrt{x^2 + y^2}$. This slice could be used for example for nonholonomic robots where sharp turns are not possible.

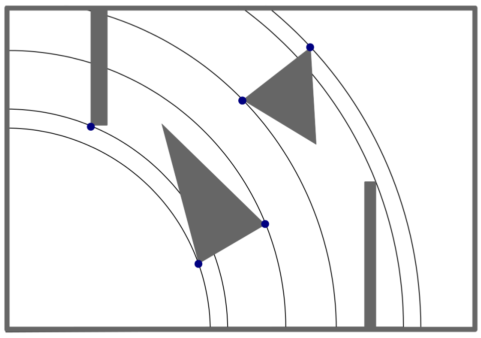


Figure 11: Concentric circles Morse decomposition with critical points

3.1.2 Grid-Based Decomposition Methods

The workspace that we want to cover by the robot can be also decomposed by using a grid. This means that we divide the workspace into square-shaped cells as we did in Grid-based approximate decomposition path planning. The advantage of this idea is simplicity of decomposition compared to previously described exact cellular decomposition methods. On the other hand there is also disadvantage in handling with grid cells that are partially occupied by obstacles. We have to consider such cells as obstacles which could lead to uncovered parts of workspace. Then the completeness of coverage depends mainly on grid resolution. Finally the coverage around obstacles, where such partially occupied cells occurs, can be also improved during execution of coverage plan by extending the path as close to the obstacles as possible.

For the exact cellular decomposition we used exhaustive walk through the graph in order to find order in which cells are covered. In grid-based decomposition we use different approach.

Spanning-Tree Coverage

In the Spanning-Tree Coverage (STC) method [13] is used search of minimal spanning tree of the grid adjacency graph. In such graph the grid cells represent vertices and the graph edges mean that the cells are next to each other. The minimal spanning tree (MST) is tree graph that has all vertices as original graph and has the minimal possible weight of edges. Finding of MST could be done by various algorithms, for example by using well known Prim's, Kruskal's or Borůvka's algorithms. As edge weights we use distance between vertices, which is in our case the grid spacing.

For the spanning tree algorithm we uses cells of size $4D$ called *mega cell*. Each mega cell contains four smaller cells of size D. The parameter D is very important, because it has to be of same size as robot or as the area that robot can cover by coverage sensor.

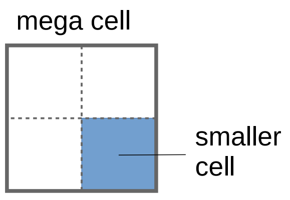
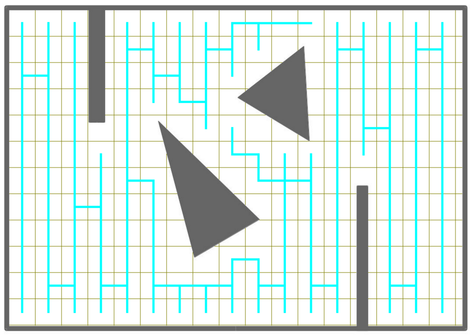
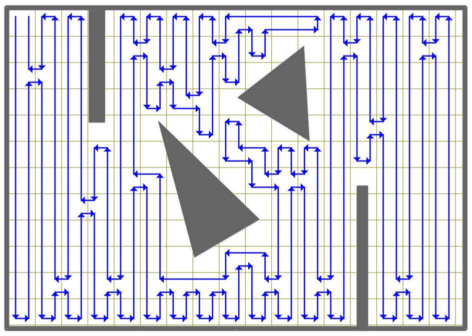


Figure 12: Spanning-Tree Coverage Mega Cell

When we have spanning tree that connects centres of mega cells, we can create coverage path for the robot by simply driving around spanning tree in counterclockwise direction. By this manner we create path that goes through all centres of small grid cells and the path is also smallest possible, because it was generated by using minimal spanning tree. In Figure 13 is shown spanning tree and generated coverage path that circumnavigates the spanning tree.



(a) Spanning tree of the grid



(b) Circumnavigate coverage path

Figure 13: Spanning tree coverage

As we can see, the off-line algorithm omits some partially occupied cells. In Spiral-STC [12], which is an on-line STC algorithm, are covered also such partially occupied cells. The Spiral-STC firstly generates the spanning tree also in partially occupied cells and then the final 'spiral' path is deformed in those cells in order to avoid the obstacles.

In Backtracking Spiral Algorithm (BSA) [17] the coverage of partially occupied cells is done by using wall-following procedure and by backtracking to cells where coverage is incomplete.

3.1.3 Alternative approaches

In alternative coverage approaches are mentioned some other robot coverage methods that are rather heuristic than algorithmic. Among such methods could be included bio-inspired or randomized methods.

Random coverage path planning is one possible heuristic method for robot coverage. Its implementation is very simple, because it only uses random motion patterns, for example turn randomly and go straight, to cover the workspace. This approach is widely used in cleaning robots [39]. Such generated paths are far from optimality in their length, but with increasing time the full coverage is ensured. Because of this fact, the applications of random coverage are limited to those where time and path costs are not important.

Neural networks approach is another approach that is provably capable of solving the CPP [50]. The approach uses grid of the workspace where in each cell is placed one neuron that has connection to its neighbours. From the input of actual robot's position, the neural network can determine move based on activity of neurons.

3.2 Multirobot Coverage

In multi-robot approaches to robot coverage path planning we consider using multiple robots mainly in order to decrease the coverage time. Despite the fact that multi-robot applications require increased costs, there are also other advantages like fault tolerance and capability of mutual localization. In this chapter are presented some basic algorithms with increased focus on the ones that are used further in this work.

3.2.1 Boustrophedon multi-robot coverage

The boustrophedon multi-robot coverage [40] uses exactly the same decomposition of workspace as the version of this algorithm for single robot. After decomposition we have graph representation of cells inside workspace. In order to utilize multiple robots, the Boustrophedon multi-robot Coverage consider two options.

The first option is to use a team-base multi-robot coverage. In this case, we find order in which the cells are covered as in single robot approach. Then we cover each cell by using simultaneously all robots as in Figure 14. The first two robot that enter the cell are explorers and cover boundaries of the cell and possible critical points where coverage is complicated due to for example narrow passages. Other robots are used as coverers that use plow-like path and cover the internal of the cell. By this manner all cells are covered using all robots. For case with only two robots, the both robots serves firstly as explorers and afterwards as coverers. The team-based approach is most useful in workspaces where cells are big enough to contain all robots, otherwise we can not utilize all robots during the coverage.

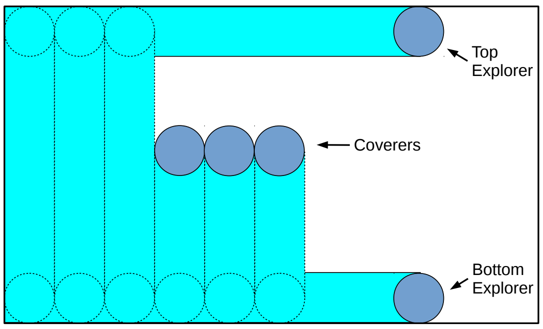


Figure 14: Team coverage of a single cell

The second option is a distributed approach. In this case the robots are spread out in the workspace. To each robot a part of workspace called *stripe* is assigned, which may or may not correspond with the cells created by previously presented Boustrophedon decomposition. Initially the robot tries to cover the outline of the stripe and then the robot performs single robot coverage of the interior of the stripe. Finally if some part of assigned stripe is not covered, for example because it is blocked by obstacle, the uncovered part is assigned to other robot by greedy auction mechanism.

3.2.2 Grid-Based Decomposition Methods

The grid-based method uses decomposition of workspace into square-shaped cells that form the grid. The two methods mentioned bellow are both base on minimal spanning trees search as we have seen in the single robot grid decomposition approaches.

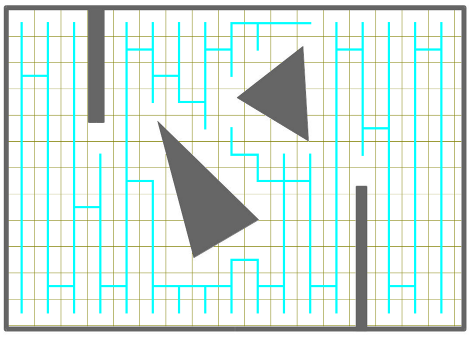
Multi-robot Spanning Tree Coverage

The Multi-robot Spanning Tree Coverage (MSTC) [19] is an extension of the single robot STC method for multiple robots. The algorithm is based on spanning tree of the grid adjacency graph. For the single robot version we let the robot to drive around the spanning tree in counterclockwise direction which creates shortest path possible while covers whole grid.

In *non-backtracking MSTC* all robots drive along the spanning tree in counterclockwise direction until they reach starting position of other robot. In this way the grid workspace is covered as with single robot, but the utilization of robots mainly depends on their starting positions. When the robots start near to each other, then there is one robot that covers almost whole workspace while the other robots only cover small space between each other. The best case scenario is when the robots are spread equidistantly in path around the spanning tree. In that case, with using n robots, the cover time is n times smaller than in single robot version.

The *backtracking MSTC* reduces the cover time in cases where robot starts close to each other. When there is robot that has to cover more than half of the whole path around the spanning tree, then this long part is covered by two robots. One of them is still covering the long part in counterclockwise direction and the other covers the long part in clockwise direction. By using this backtracking approach the the worst-case scenario for n cells is covered in $n/2 - 1$ steps for $k = 3$ robots or for $k = 2$ robots in $2n/3 - 1$ steps.

In following Figure 15b are shown paths for two robots that uses MSTC method. Robot's start positions are in left and right upper corners so the coverage algorithm does not have to use backtracking approach. We can also see, that some parts of the workspace are not covered because the cells there are partially occupied by obstacle.


(a) Spanning tree on the grid

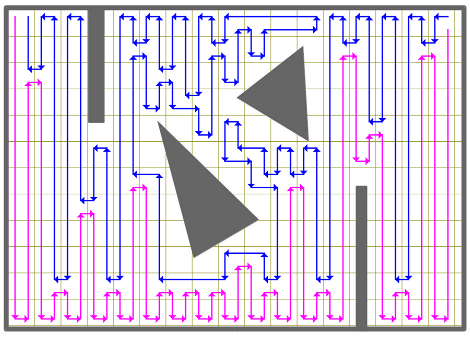

(b) Coverage path for two robots

Figure 15: Multi-robot Spanning Tree Coverage

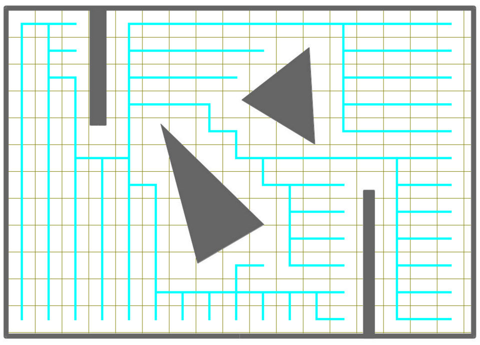
Multi-robot Forest Coverage

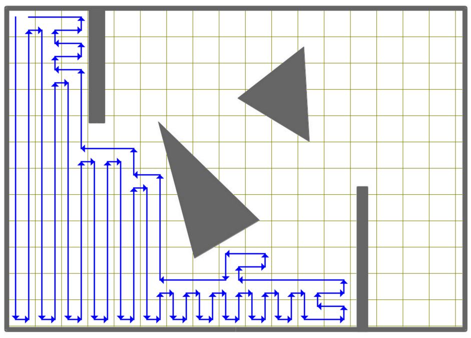
The second important method that uses grid-based decomposition of workspace in multi-robot coverage is the Multi-robot Forest Coverage (MFC) [53]. The main difference from the MSTC is that the MFC method finds a rooted tree cover, which is forest of k trees that covers the grid. All k robots then circumnavigate only their own tree.

The search for rooted tree cover is NP-complete because it can be reduced to bin-packing problem. The reason for this is that we need k trees that cover the space with minimum possible vertices and we also want the trees to be of nearly same length in order to utilize all robots equally. The presented MFC algorithm does not find the rooted tree coverage by hard, but instead it uses polynomial-time TREE COVER algorithm [11]. The TREE COVER does not find the minimal cover tree, but its worst case coverage distance is maximally four times larger than optimal.

The TREE COVER itself works as follows. Firstly we removes all edges of size larger than some value B. Then we find minimum spanning tree of this graph with contracted all roots into one. The minimum spanning tree is afterwards split to k trees. All trees are then decomposed into subtrees with weight in range $[B, 2B)$ and possible leftover subtrees with weight bellow B. The non-leftover trees are connected with their roots with edge of maximally B weight using maximum matching approach. If there are some non-leftover trees that cannot be connected using actual value of B, then we know that the rooted tree with maximal weight of B does not exists. In the end the algorithm returns trees that each has maximal weight of $4B$ and consists of root, one non-leftover tree and path to the leftover subtree. Key factor of algorithm described above is search of smallest possible value B in order to get minimal rooted tree cover.

As we can see in Figure 16, the algorithm finds the coverage for two robots that have their starting positions next to each other.


(a) Spanning Tree for MFC


(b) First Robot Coverage Path

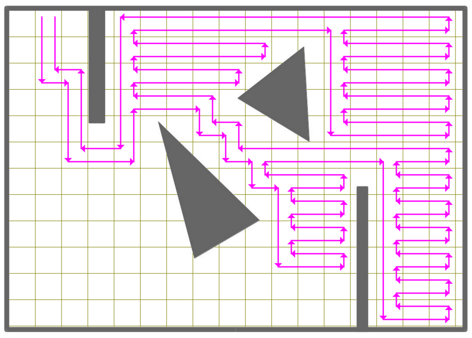

(c) Second Robot Coverage Path

Figure 16: Multi-robot Forest Coverage

3.2.3 Alternative approaches

Also in multi-robot coverage we can use heuristic approaches. As in single robot methods we can use the **random coverage path planning** that uses random motion patterns. The only difference from approach where only one robot is used, is requirement for avoiding other robots.

A bee pheromone signaling approach [3] is **biological inspired** method that tries to imitate the bees covering the area. Another bio-inspired approach uses ants alarm pheromones [35].

4 Motion Planner for AUV

The first goal of this thesis is to realize a motion planning for Autonomous Underwater Vehicle (AUV). Generally, the motion planning of AUVs in seabed monitoring task is used in two different scenarios. First, the motion planning is used for local (short-range) maneuver, e.g. during docking of AUV to the charging station. The second utilization of motion planning is designed for long distanced plans, where the vehicles have to travel several hundreds meters and more. For both scenarios we use the Rapidly Exploring Random Tree (RRT) method. The reason why we chose RRT rather than PRM is that the connection of nearest neighbour configurations in PRM's learning phase would require a special metric because of the dynamical constraints of the AUV (described in Chapter 4.1). In the RRT we could use the Euclidean metric for selecting nearest neighbour q_{near} and then expand this node by using a set of possible inputs to the AUV drive system. The second reason for choosing the RRT method is that the single query motion planning is sufficient for us.

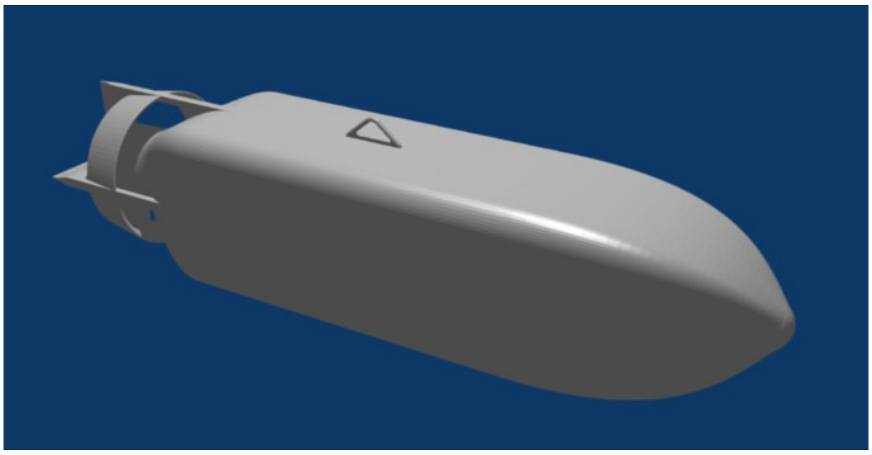


Figure 17: AUV 3D CAD model used by motion planners (courtesy of David Oertel)

Two different approaches were tested for solving the motion planning task for AUV. The first approach, described in Chapter 4.2.1, uses the RRT-blossom idea of expanding the selected nearest neighbour q_{near} into multiple nodes using set of possible inputs to the AUV drive system. In this approach we used the dynamical model of AUV inside the planner. By applying multiple inputs to thr dynamical model from nearest neighbour configuration q_{near} the RRT tree is expanded until the goal configuration is found.

The second approach described in Chapter 4.2.2 uses RRT path planning with the Dubins curves local planner. In this approach, the motion planner designs a motion plan as a sequence of curves that are obtained using Dubins-based local planner. The plan is represented by a set of 3D points. The AUV then navigates along the plan by using a simple Lookahead steering controller [5, 25].

4.1 AUV model

Motion planning for the AUVs is a challenging task due to the dynamics of the vehicles. To generate suitable motion plans, the dynamics needs to be considered in the motion planner. The chosen RRT method searches the configuration space by incrementally expanded tree. To ensure that the final plan is feasible, the tree has to be expanded using the motion model. In this work we used provided Matlab Simulink model of the AUV in order to guarantee the feasibility of designed motion plans. The AUV dynamical model was developed by David Oertel from Institute of Process control and Robotics, KIT, Germany.

The AUV model is governed internally by equations for 6-DOF marine craft [9, 10]:

$$\dot{\boldsymbol{\eta}} = \boldsymbol{J}(\boldsymbol{\eta})\boldsymbol{\nu}$$

$$\boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{g}(\boldsymbol{\eta}) = \boldsymbol{\tau}.$$

The generalized position $\boldsymbol{\eta} = [N, E, D, \phi, \theta, \psi]^T$ and velocity $\boldsymbol{\nu} = [u, v, w, p, q, r]^T$ represent actual state of the AUV. Figure 18 illustrates the North East Down (NED) coordinate system, that is used in marine craft equations. The velocity $\boldsymbol{\nu}$ represents speed in according NED position.

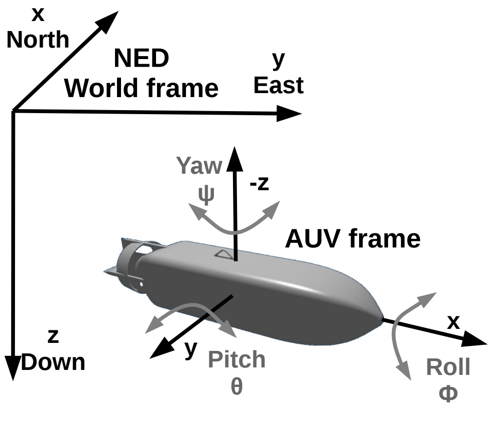


Figure 18: NED world frame system with AUV frame

Other parameters of the equations are Euler angle velocity transformation matrix $\boldsymbol{J}(\boldsymbol{\eta})$, System inertia matrix $\boldsymbol{M}$, Coriolis-centripetal and Damping matrices $\boldsymbol{C}(\boldsymbol{\nu})$, $\boldsymbol{D}(\boldsymbol{\nu})$, vector of gravitational/buoyancy forces $\boldsymbol{g}(\boldsymbol{\eta})$ and control inputs $\boldsymbol{\tau}$. We can control the AUV by three actuators: a vertical/horizontal rudder and a forward thrust.

The AUV is nonholonomic, which means that it can not move in any direction, but the direction of movement depends on its actual position. This property is caused by lower number of controllable degrees of freedom (DOF) than the total degrees of freedom. The controllable DOF is only three and consists of two rudders and one shaft inputs. On the other hand, as previously mentioned, the total degrees of freedom is six which represents actual AUV position inside the space.

To design a suitable motion planner, behaviour of the motion model needs to be investigated. The most important are responses to control inputs (height rudder, yaw rudder and thrust). Resulting behaviour of diving speed and turning diameter are then used to determine resolution of motion planner.

The height rudder is responsible for ascent and descent of the AUV depth. Its effects on the AUV is important, because it determines how fast the AUV can dive to a certain depth. This therefore determines time required to reach positions on different depths. The real AUV maximal diving speed determined from measured data is approximately 0.21 m/s. Using the motion model, the maximal diving speed was measured to be almost 0.2 m/s with forward speed of 1.5 m/s. The minimal speed of AUV is 0.51 m/s, because in smaller speeds the AUV is not able to dive dynamically and the height rudder has no control over the diving speed. As is shown in Figure 19, the diving speed is maximal for height rudder at angle -0.48 rad. The behaviour for going up is symmetrical with the diving behaviour, so further in the planners we consider the height rudder to use only angles in range $(-0.48, 0.48)$ rad and the diving speed to be maximally 0.2 m/s. From the maximal diving speed we can also derive that the maximal diving is approximately 0.13 m per one meter of forward motion.

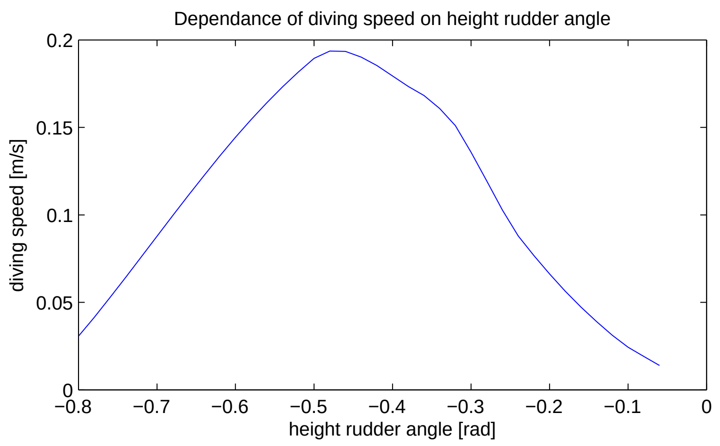


Figure 19: Dependence of diving speed on height rudder setting (with forward speed of 1.5 m/s)

The second (yaw) rudder is responsible for AUV turning around z axis. If the AUV would use only the yaw rudder without changing the depth of dive with the height rudder, then the behaviour of the AUV is similar to car-like or Dubins car mobile robot. In that case, AUV moves along circular trajectories. If the yaw rudder is set to zero angle then the AUV moves straight on circular trajectory with infinite diameter. The fact that AUV moving in same depth behaves like car-like robot motivates us to design a motion planner for long distances similarly as for car-like mobile robots. One of the possible solutions is to employ the Dubins planner [30]. For this planner, we need to investigate the minimal turning diameter. As shown in Figure 20, there exists angle of yaw rudder that generates trajectory with the minimal diameter.

Figure 20 also shows, that the provided model versions differ in their response to the yaw rudder setting. The first version used in the planners, denoted as 'older model version', has very similar behaviour to the height rudder and the minimal diameter of turning is 30 m with 0.48 rad angle of yaw rudder. In contrast to this, the actual AUV model continues to decrease the turning diameter even after angle of $\pi/2$, which is not what we would expect. In reality the angle over $\pi/2$ supposes to change direction of turning from right to left, which does not happen in the model. This is why we consider the yaw rudder angles to be in range $(-0.48, 0.48)$ rad. The minimal turning diameter used further in both planners and seabed monitoring is 25 m which is the turning diameter of newer model with 0.48 rad yaw rudder angle. In the motion planners we use the newer model, but with respect to the minimal turning diameter found in the older version.

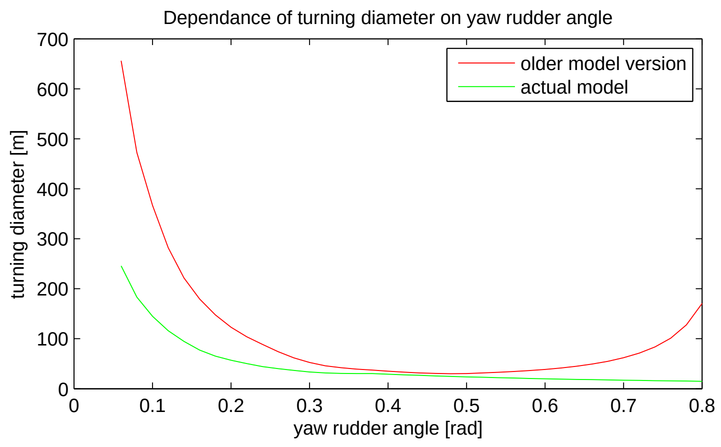


Figure 20: Dependence of turning diameter on yaw rudder setting (with forward speed of approximately 1.5 m/s)

The third AUV input, which is the shaft speed, that creates the forward thrust, is not so important because in planners described in next chapters the speed is considered to be constant around 1.5 m/s. This speed is used based on average forward speeds of the real AUV. For approximately constant speed of AUV we can use constant shaft speed input. The only case where we also consider non-constant shaft speeds is the docking task described in Chapter 4.3.

Utilization of Simulink model for RRT-based planner

In order to use the model inside the planner, the model was modified in a way that all internal states are both controllable and observable. It required integrators to be modified so that the internal states of them are both inputs and outputs of the model. As the AUV moves in 6-DOF space, both position and velocities, that are the internal states, consists of six parameters like in the equations for marine craft.

It means that the inputs of the AUV model are not only the drive control signals for height rudder, yaw rudder and shaft speed, but also the initial position and velocity. Also the outputs of AUV model had to be extended to consists of both position and velocity. This modification enables to save the whole description of state in which is the AUV model.

Finally, as shown in Figure 21, the model also allows to set external current velocities in order to simulate the deep water environment more realistically. Despite the fact that the influence of water currents was tested, it is not mentioned later because of lack of model behaviour of deep water currents and only static velocity currents could be simulated.

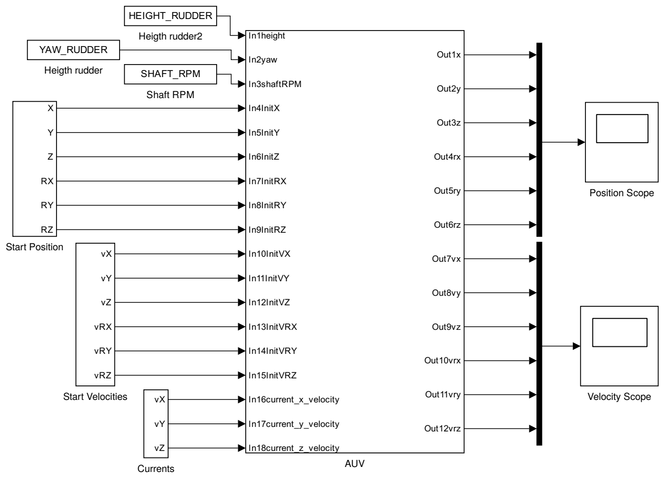


Figure 21: AUV Matlab Simulink model

Besides the motion (dynamical) model, the RRT planner also needs a 3D model of the vehicle in order to implement collision detection. The used 3D model is depicted in Figure 17. Both dynamical and 3D models of AUV are used with the permission of KIT.

4.2 Long distance planners

In this Chapter are presented two different approaches to AUV motion planning on long distances inside space with obstacles. The intended plan distances are in hundreds of meters or more. Both planners are capable of planning collision free motion plans that consist of both AUV state together with inputs to the AUV drive for transition to the next state. The first RRT motion planner in Chapter 4.2.1 uses multiple inputs to AUV model in order to grow the planning tree until the tree reaches the goal position. The second RRT planner 4.2.2 uses Dubins curves motion primitives to plan a path that is afterwards traversed by a steering controller that creates the final plan. Both planners are compared in Chapter 4.2.3.

4.2.1 Input-based RRT planner

The first implemented and tested RRT planner approach to solve the path planning problem for the AUV is planner that we call input-based. The reason for this name is that we use multiple different inputs to the AUV drive in order to grow the planning tree. This approach is similar to the RRT-blossom described in Chapter 2.2.2. The main difference is that we use Simulink AUV simulation in order to get AUV state using those different inputs.

As in the standard RRT, the planning tree starts with initial state s_{start} (configuration) of the AUV. Then in each step of the RRT algorithm, we find nearest neighbour state s_{near} in the tree to a randomly generated position p_{rand}. Then we use combination of control inputs to AUV (i.e. angles of rudders) and run the simulation of the AUV from nearest neighbour state for a constant time step t_{sim}. By this manner the planning tree expands throughout the space until it reaches the final required position. The Algorithm 3 summarizes how the input-based RRT planner works.

Algorithm 3: Input-based RRT planner

```
input : AUV start state $s_{start}$ and desired goal position $p_{goal}$ , maximal
        allowed final distance to goal $d_{goal}$, maximal number of iterations $N$,
        rudder step $r_{step}$, constant input AUV shaft speed $shaft\_RPM$ and
        simulation time $t_{sim}$.
output: Sequence $P$ of AUV states and inputs or failure
1  $TREE.add(s_{start})$;
2  $iteration = 0$;
3  while $iteration < N$ do
4      $p_{rand} = random\_position()$; // random $(N, E, D, \psi)$
5      $s_{near} = nearest\_neighbour(TREE, p_{rand})$; // nearest AUV state to $p_{rand}$
6      for $yaw\_rudder = -0.48$ to $yaw\_rudder = 0.48$ using $r_{step}$ do
7          for $height\_rudder = -0.48$ to $height\_rudder = 0.48$ using $r_{step}$ do
8              $\tau = (yaw\_rudder, height\_rudder, shaft\_RPM)$;
9              $s_{expand} = simulateAUV(s_{near}, \tau, t_{sim})$;
10             $newNode = \{s_{expand}, \tau\}$;
11             $TREE.expand(s_{near}, newNode)$;
12         end
13     end
14     $d = distance(TREE, p_{goal})$;
15     if $d \leq d_{goal}$ then
16         $s_{goal} = nearest\_neightbour(TREE, p_{goal})$;
17         $P$ = backtrack from $s_{goal}$ to $s_{start}$;
18         $return\ P$;
19     end
20     $iteration = iteration + 1$;
21 end
22 $return\ failure$
```

The nearest neighbour search algorithm($nearest\_neighbour(TREE, p_{rand})$) effectively uses only four coordinates of AUV, the three dimensional position of AUV centre of gravity and rotation of AUV around z axes (N, E, D, ψ). The other two position parameters, the rotations around x and y axes (roll ϕ and pitch θ), are neglected because their values are usually close to zero and they are not so important in term of distance between two positions of AUV. The state of AUV also contains another six parameters of AUV speed, but we also do not consider them in search for the nearest neighbour state because we use constant speeds, otherwise the planning would get even more complicated with higher dimension. For all four used parameters (N, E, D, ψ), we use Euclidean distance with higher scale for ψ and D parameters, because it is more difficult for the nonholonomic AUV drive to change them, so they have higher importance in terms of distance between two states.

In the algorithm, we also use the goal-bias method inside the $random\_position()$ function, that attracts the growth of tree in the direction of the goal position. The only required modification compared to the standard goal-bias is, that instead of random generation of the goal position, we also generate positions in cubic neighbourhood of the goal. The reason for this modification is that the AUV drive is nonholonomic, so the euclidean nearest neighbour does not have to be the closest state for the AUV drive. Size of the cubic neighbour is derived from the minimal turning diameter of the AUV. By using this type of goal-bias, we ensure a denser growth of planning tree around final position and also allows nonholonomic AUV drive to turn to right final ψ position.

As was shown in the previous Chapter 4.1 describing AUV model, the most effective inputs to yaw and height rudders are angles in range $(-0.48, 0.48)$ rad. For this reason we used them as boundary values in simulation of AUV from nearest neighbour state s_{near}. We used equidistantly distributed inputs to AUV rudders where the rudder step parameter r_{step} defines how much the planning tree branches out. The smaller the rudder step is used, the more expanded state we have and also the space is covered much denser. On the other hand, using too many rudder inputs leads to significantly slower search as more simulations must be computed. Because of these factors, we used $r_{step} = 0.24$ rad for states far from goal and smaller $r_{step} = 0.12$ rad for positions that are close to the goal. This increased number of simulated rudder inputs near to the goal positions allow closer approach of AUV to the final required position.

The designed input-based planner was verified in canyon-like environment. The size of the canyon is almost 1 km^2. Size of the AUV is approximately $3.5 \times 0.8 \times 0.8$ m. Plan found in the canyon-like environment by using described algorithm is shown in Figure 22. The plan was found in approximately ten minutes within 646 iterations and with 321025 nodes in planning tree. The threshold for reaching the goal position was set to $d_{goal} = 5$ mu, where mu is combination of four used position parameters (N, E, D, ψ) with increased scale for ψ and z parameters as was used in the $nearest\_neighbour(TREE, p_{rand})$ function. For each rudder input we used simulation time $t_{sim} = 15$ s, where every 0.5 s the position of AUV is tested for collision and the state is added to the planning tree. If the collision was detected, then the simulation for given rudder input was stopped and the AUV state was not added to the tree. The collision detection every 0.5 s is important, because of the AUV dimensions. With bigger collision detection period, the AUV could pass through obstacle without detected collision. The shaft rotation per minute was $shaft\_RPM = 250$ in order to get approximately 1.5 m/s forward speed of AUV.

The path needed between start position $(N, E, D, \psi) = (-260, 400, -90, -1.57)$ and goal position $(N, E, D, \psi) = (100, -400, -80, -1)$ has length of 1564 m.

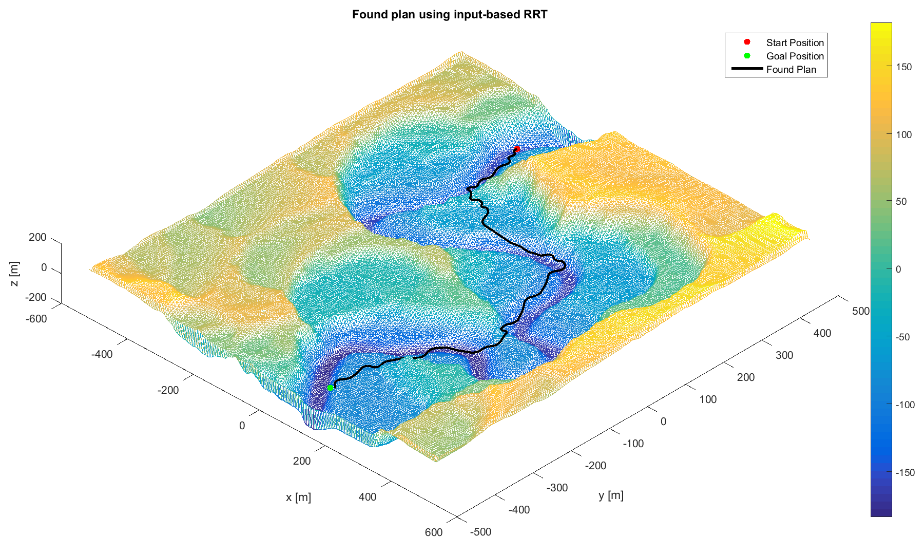


Figure 22: Plan found in the canyon-like environment using input-based planner

In Figure 23 we can see the planning tree used during the search for previously shown plan. We can see that the tree spans almost entire canyon. Also the input-base approach is clearly visible. As the single state branches into multiple states using different inputs to rudders, it creates a cluster of states that looks like a hand fan.

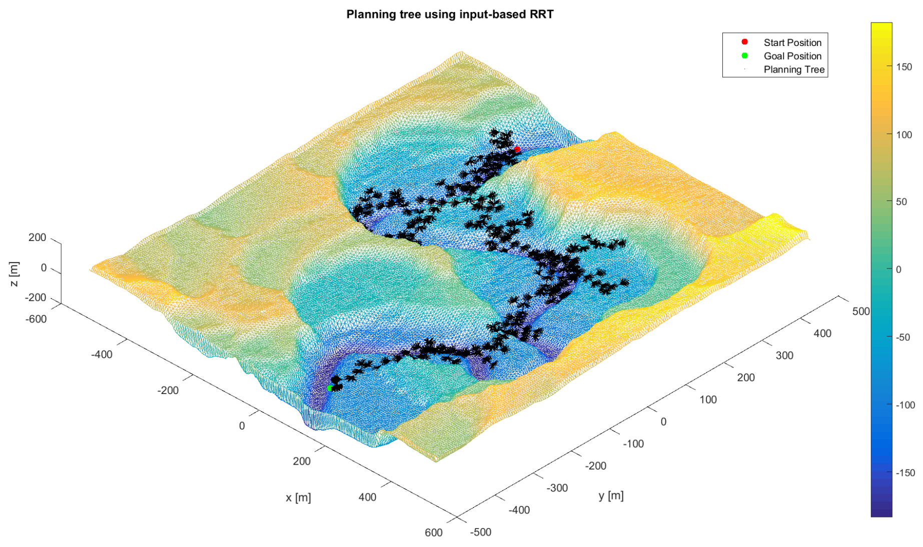


Figure 23: Planning tree in the canyon-like environment using input-based planner

The input-based method presented in this chapter is able to plan between two long distance positions. By using RRT-blossom approach and simple enhancements of goal-bias, we were able to find collision free motion plan in canyon-like environment with length of 1564 m and final position error bellow 5 m.

4.2.2 Dubins-based RRT planner

The second approach to RRT motion planner for AUV uses the Dubins curves [30]. These curves quite accurately describe the paths that creates the AUV when using the rudders and the shaft for control of forward speed and orientation. In fact these curves can be used as motion primitives that satisfies the dynamical constrains of the AUV drive and its nonholonomicity. The basic idea behind the planer is to firstly plan a path using Dubins curves and then use a Lookahead steering controller to traverse the path with the Simulink AUV simulation. The final plan consists of both AUV states and inputs from the steering controller and is qualitatively same as plan from the first input-based planner.

In following paragraph, the Dubins curves used for the path planning are firstly described. Then, the Lookahead steering controller used to drive the AUV along the planned path is described. Afterwards, the algorithm of Dubins-based RRT planner is presented together with an experiment.

Dubins curves

The Dubins curves [30] are the shortest curves that connect two position inside workspace $\mathcal{W} = \mathcal{R}^2$, where the robot position is described by three parameters $p = (x, y, \varphi)$. The (x, y) parameters are robot's position in workspace and the parameter φ is the heading angle of the robot.

Originally, the curves were used for nonholonomic car-like robot where the main dynamical constraint is a maximum steering angle which causes a minimum turning radius. This is why the AUV model was explored mainly in order to get the minimum turning diameter, because it is the only parameter of the Dubins curves.

There are a total number of six different Dubins curves types. Each type can be described by three motion primitives. The first primitive S is when the robot drives straight ahead. Primitives R and L stand for turning right and turning left, using the minimal turning radius. With these acronyms for motion primitives the six possible Dubins curves are LRL, RLR, LSL, LSR, RSL and RSR. In Figure 24 are shown all six Dubins curves between two positions. Important fact is that all six Dubins curves are not always possible especially for positions that are close to each other.

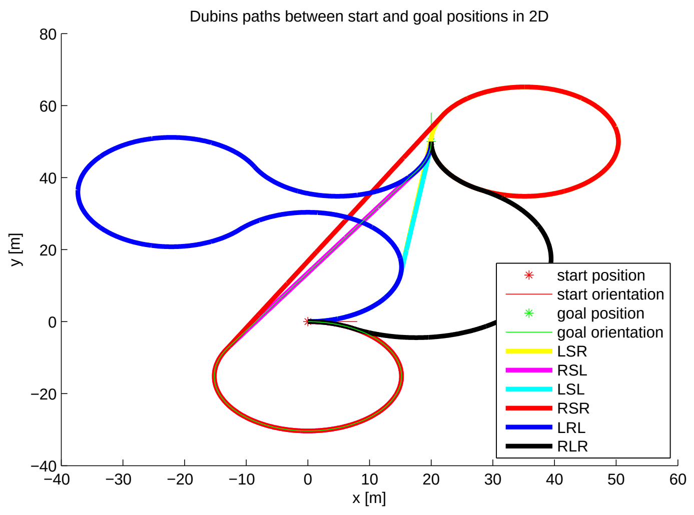


Figure 24: Types of Dubins curves

The Dubins curves can be easily extended for the workspace $\mathcal{W} = \mathcal{R}^3$ to enable their usage for AUV. For this extension we need the second important parameter, the maximal ascending/descending distance per forward driven distance, which corresponds to the maximal diving speed. The value of this parameter was also investigated in Chapter 4.1. Introduction of the depth position of AUV is then solved by using linear decrease of the depth on 2D Dubins curve. If the difference in depth is too high, so that the linear decrease has to be bigger than the allowed by maximal ascending/descending distance per forward driven distance, then we use following strategy. The AUV circles up or down using the maximal possible ascending/descending speed until it reaches depth, where the linear decrease of the depth for rest of Dubins curve is possible without violation of the maximal diving speed.

AUV steering controller

The Lookahead steering controller [25], also known as Follow-the-carrot, is a very simple controller. As the Dubins curves, we can find main usage of this controller in car-like robots with nonholonomic steering, but can be also used for AUV. Figure 25 ilustrates the idea behind the Lookahead controller.

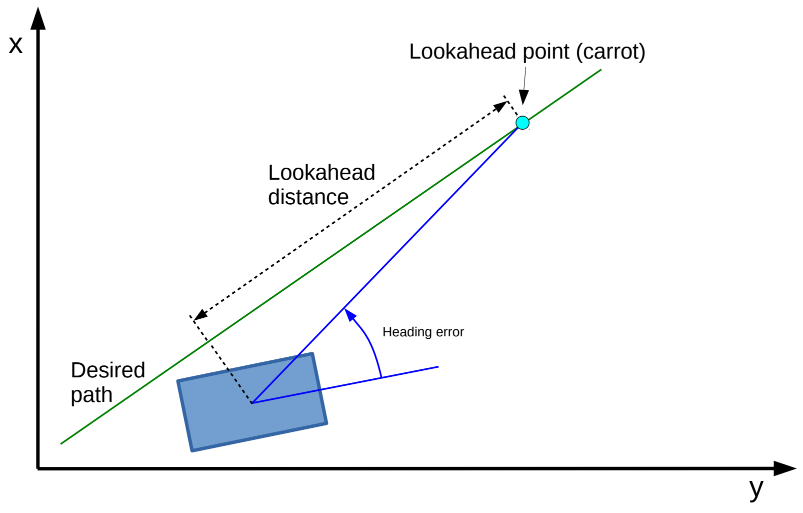


Figure 25: Lookahead steering diagram

In every step, the controller selects a lookahead point on the desired path. The point is found in a lookahead distance from the point that is nearest to the actual position of robot. Then a proportional control is used in order to minimize the Heading error. For the AUV the proportional control uses the yaw rudder to minimize the heading error. An extension of this controller to 3D space is simple and uses the same principle. In 2D we minimized only one heading error angle ψ in (x, y) reference frame. While in 3D, we try to minimize two heading error angles, the first ψ in (x, y) and the second θ that relates to z axis, for the AUV depth. The second vertical heading error supposes to be minimized by the height rudder. Because the fact, that the AUV θ oscillates while using the height rudder, we used a different method. Instead of controlling the pitch, we used a proportional-integral control to minimize a depth difference to lookahead point. By this change, the oscillations were minimized, which resulted in better precision in driving to the desired depth.

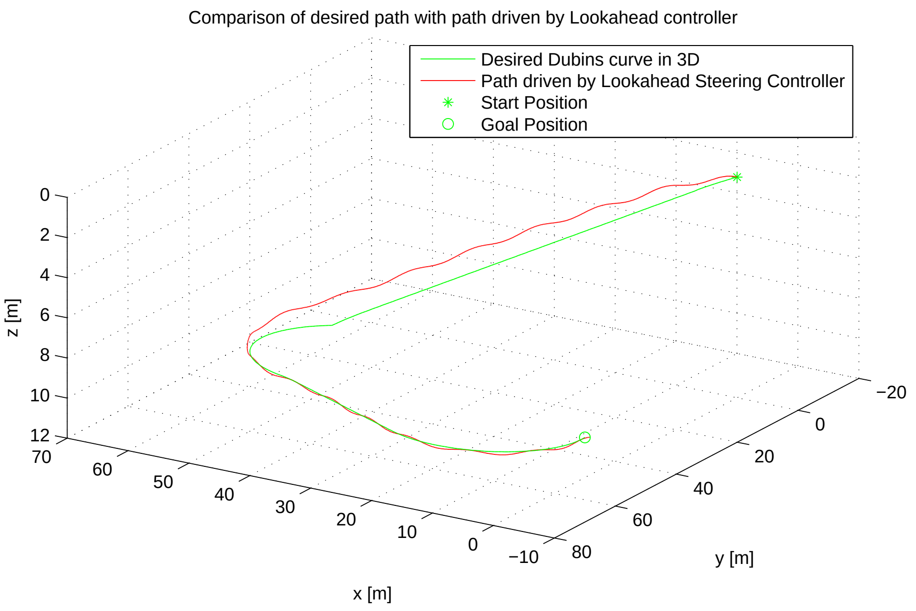


Figure 26: Lookahead steering controller used on two Dubins curves in 3D

A desired path composed of two 3D Dubins curves is shown in Figure 26. The figure also shows trajectory obtained by the Lookahead steering controller. The values from proportional and proportional-integral controls are limited in order to produce angle of rudders in range $(-0.48, 0.48)$ rad. The lookahead distance used in the experiments was $L = 5$ m. As we can see in the Figure 26, the controller is always a little delayed behind the desired path, which is caused by the fact that controller always aims to a point in front. Also the vertical control is more imprecise than the horizontal one because of the oscillations made by height rudder control.

Despite these inaccuracies cased by oscillations of the controls, the Lookahead steering is able to traverse the Dubins path. This allows us to simplify the motion planner, so it provides the result as a sequence of Dubins manoeuvres.

Dubins-based planner algorithm

The algorithm of Dubins-based RRT planner works similarly as the standard RRT planner with two main differences. The first difference is, that it uses Dubins curves local planner to create connection of two position p_{rand} and p_{near}. The second is, that the final motion plan is created from path plan P_{path} using the Lookahead steering controller.

Instead of using planning tree that consists of AUV states (both position and velocities), as we did in the input-based planner, in Dubins-based planner the planning tree consists of AUV position $\boldsymbol{\eta} = [N, E, D, \phi, \theta, \psi]^T$. The algorithm starts with adding of AUV start position p_{start} to the planning tree. Then in every step, the algorithm generates the random position p_{rand} using only four parameters of AUV position (N, E, D, ψ).

Using the random position p_{rand} we find a nearest neighbour position p_{near} out of the planning tree. The nearest neighbour function $nearest\_neighbour(TREE, p_{rand})$

works quite differently from the version we used in input-based planner. The function still uses Euclidean distance with higher scale for ψ and D parameters, but instead of searching for one nearest neighbour it searches for k nearest neighbours and afterwards it calculates Dubins curves to each of them and uses the one with shortest Dubins curve. By this method we overcome the fact that the nearest positions in Euclidean distance is usually not the nearest position using Dubins curves. The advantage of using k-Nearest Neighbours is that we do not have to calculate Dubins curves to all positions in planning tree, but only to k nearest. We also know that the nearest neighbour position using Dubins curve is probably among those k nearest neighbours using Euclidean distance. In the algorithm we used $k = 16$ which is the maximal value for used k-Nearest Neighbours implementation [51].

When the nearest neighbour position p_{near} is selected, the algorithm creates Dubins curve from the nearest neighbour position p_{near} to p_{rand}. The curve is returned as a sequence of positions that are in distance d_{sample} to each other. In order to limit growth of the planning tree, only d_{add} initial length of the curve is used. Then in expansion of the planning tree $TREE.expand(p_{near}, P_{dubins})$ in Algorithm 4, the positions in the Dubins curve are added to the tree consecutively until any position is in collision.

Finally when the planning tree reaches a position whose distance is less than d_{goal}, the Lookahead steering controller is used. Starting in AUV initial state s_{start}, the controller drives the previously created path plan P_{path} and in this way creates the motion plan.

Algorithm 4: Dubins-based RRT planner

input : AUV start state s_{start} and desired goal position p_{goal} , maximal allowed final distance to goal d_{goal}, maximal number of iterations N, maximal additional expand distance d_{add} and Dubins curves sampling distance d_{sample}.

output: Sequence P of AUV states and inputs or failure

```
1  $TREE.add(p_{start})$; //position part of AUV state $s_{start}$
2  $iteration = 0$;
3  while $iteration < N$ do
4      $p_{rand} = random\_position()$; // random $(N, E, D, \psi)$
5      $p_{near} = nearest\_neighbour(TREE, p_{rand})$; // nearest AUV position to $p_{rand}$
6      $P_{dubins} = dubinsLocalPlanner(p_{near}, p_{rand}, d_{add}, d_{sample})$; //returns Dubins
       curve between $p_{near}$ and $p_{rand}$ or its $d_{add}$ length from $p_{near}$
7      $TREE.expand(p_{near}, P_{dubins})$;
8      $d = distance(TREE, p_{goal})$;
9      if $d \leq d_{goal}$ then
10         $P_{path}$ = backtrack from $p_{goal}$ to $p_{start}$;
11         $P = LookaheadSteer(s_{start}, P_{path})$;
12         return $P$;
13     end
14     $iteration = iteration + 1$;
15 end
16 return $failure$;
```

Plan found in the canyon-like environment by using Dubins-based RRT planner is shown in Figure 27. The experiment was performed with the same starting state of AUV and goal position as we did for the input-based planner. The growth of RRT planning tree was limited by maximal additional expand distance $d_{add} = 10.0$ m and the Dubins curves used sampling distance $d_{sample} = 0.5$ m. The sampling distance was chosen with respect to the dimension of AUV in order to prevent passing through obstacles without collision detection. The plan was found between position $(N, E, D, \psi) = (-260, 400, -90, -1.57)$ and goal position $(N, E, D, \psi) = (100, -400, -80, -1)$ and its length is 1966.74 m. To the algorithm it took 62 s to find the plan with maximal allowed final distance to goal $d_{goal} = 5$ mu. The parameter d_{goal} is, as in the input-based method, the combination of four used position parameters (N, E, D, ψ).

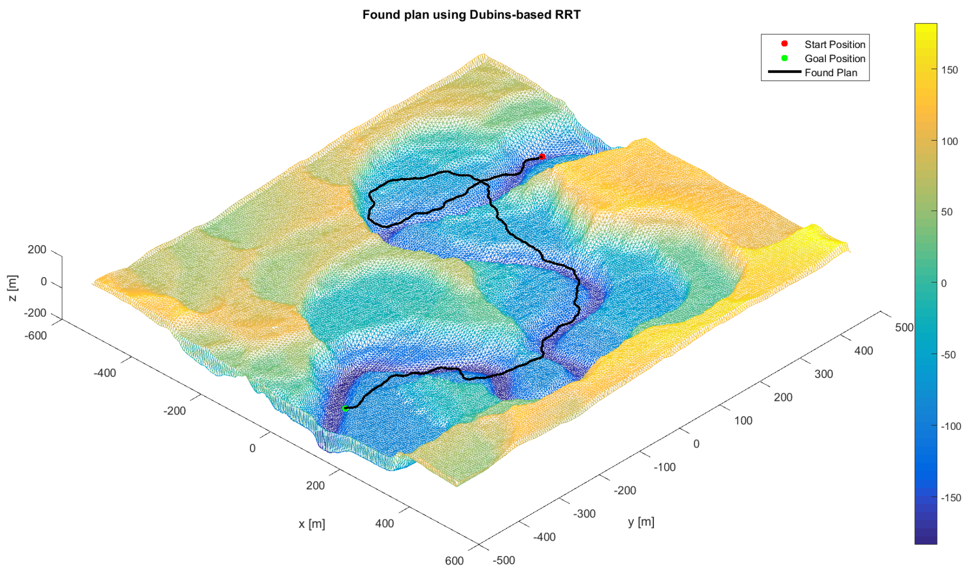


Figure 27: Plan found in the canyon-like environment using RRT Dubins-based planner for AUV

In the plan found in the canyon-like environment we can also see a loop that the AUV takes. What is not visible is, that it is taken mainly because the AUV has to go up by approximately 60 meters in a relatively small region, in order to continue towards goal position.

The Figure 28 shows the planning tree that expands through the whole canyon. The higher concentration of nodes around the goal position in the bottom left corner indicates that the algorithm spent longer time to connect the goal position.

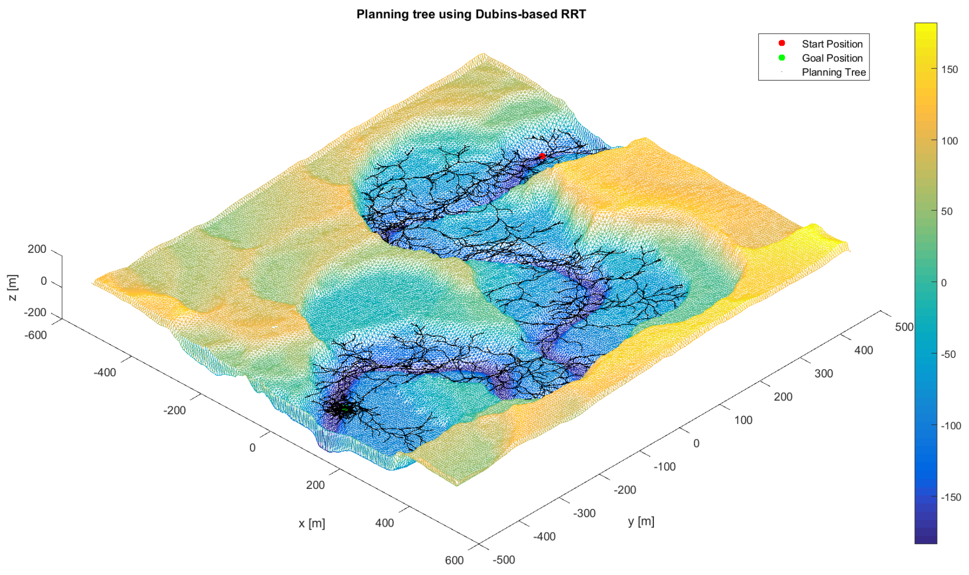


Figure 28: Planning tree in the canyon-like environment using RRT Dubins-based planner for AUV

4.2.3 Planners Comparison

The both proposed motion planners (input-based and Dubins-based) are compared using the canyon-like environment. As we used the RRT randomized motion planner, the plans found by both methods are different every time the algorithm is run, even when the same method is used. For this reason, we ran both methods repeatedly using the same start and goal configurations in order to properly compare the methods. Figure 29 shows results of this experiment. All the experiments have been performed on Notebook Intel Core i5 @ 1.7 GHz.

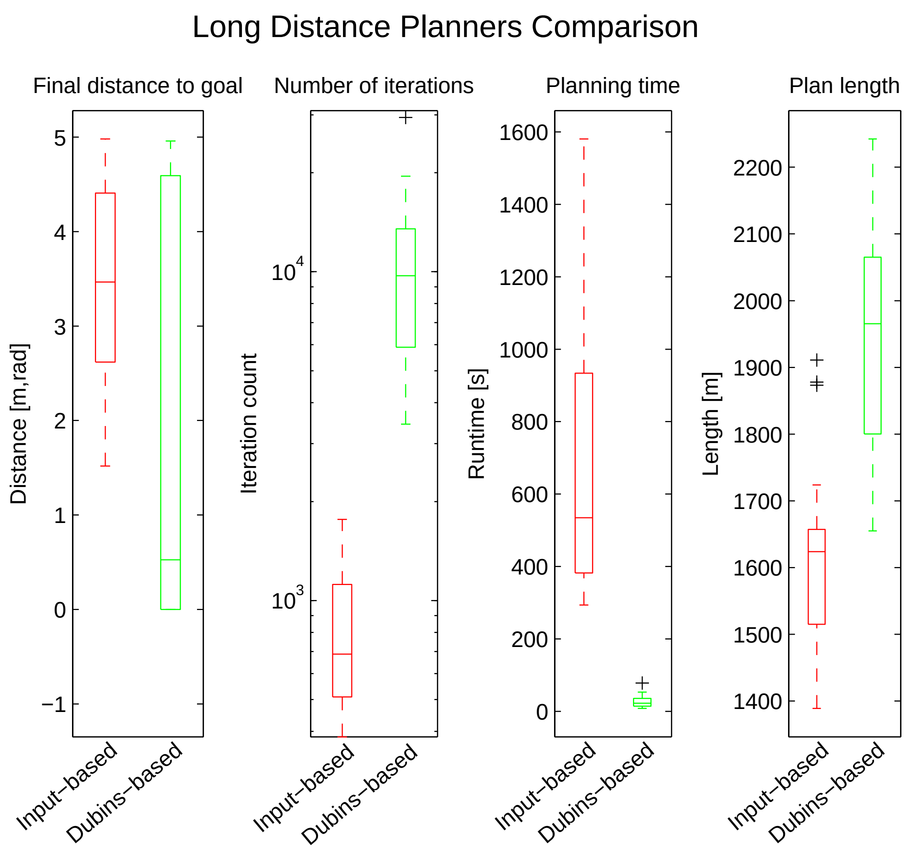


Figure 29: Comparison of Input-based and Dubins-based RRT planners

As the Input-based method uses simulation of AUV for many values of inputs to the yaw and height rudders, the planning time is significantly higher compared to the runtime of the Dubins-based method. The reason is that the AUV simulation is very complex and need time for computation. One second of simulation takes approximately 2 ms to compute, but to cover the whole configuration space with the planning tree it is time consuming. The Input-based method Runtime was in average around 8.9 min, while it takes in average only 23 s to the Dubins-based method.

Other very significant advantage of the Dubins-based method is the lower memory consumption. The Input-based method uses the simulation of AUV dynamics, where the whole state of AUV has to be saved. Every node in the planning tree has to hold the whole AUV state that consists of six parameters for position, six parameters for velocity and three input parameters. In contrast to this, the Dubins-based method effectively uses only four parameters for the position (N, E, D, ψ), which means that it is almost four times less memory consuming.

On the other hand the Input-based RRT planner has an advantage in producing shorter plans than the Dubins-based method. The Dubins-based method has in average plans with length of 1965 m while the Input-based has only 1624 m average plan length. This is caused by two facts. In the Input-based method, by using the the inputs to the AUV to expand the planning tree, we also minimize number of inputs needed to reach a certain positions in space. The second reason is that the Dubins-based method is able to connect any two positions by a Dubins curve, but using this, it also adds some curves that are unnecessary and in the end it enlarges the final plan. We can see this unnecessary enlargement of plan as the loop in plan depicted in Figure 27.

In the randomized motion planners we can sometimes observe a failure when the plan is not found within specified number of iterations N, which we chose for comparison experiments to be $N = 100000$. The Input-based approach found plan in 95 % cases for maximal allowed final distance from goal $d_{goal} = 5$ mu. On the other hand, the Dubins-based method succeeded in finding plan in the whole set of experiments.

From the comparison of final distance from goal, shown in Figure 29, we can see that the Input-based method achieves the higher average final distance of 3.5 m compared to 0.5 m average of Dubins-based method. The Dubins-based method is supposed to find a path between any two positions of AUV, but a problem arises when all the six Dubins curves are in collision. Then the method has to wait until such problematic area is sampled enough to expand using different collision free Dubins curve.

The second reason why the Dubins-based method does not always find a plan with zero final distance to goal, is that even by using $k = 16$ nearest neighbours, the algorithm can miss the position with the shortest Dubins curve. This miss can lead to a deadlock in region around goal, where many tree nodes has small Euclidean distance to the goal, but longer Dubins curve.

As we have shown above, the Input-based method needs longer time to find a plan, but in average the plans from this method are shorter. The Dubins-based method has the advantages in faster planning, smaller memory consumption and smaller final distance to goal. Using these observations, we can conclude that the Dubins-based method is more suitable for determination whether any plan between two positions exists. On the other hand the Input-based method is useful when we know that the plan exist and we want to find some shorter plan.

4.3 Short distance docking planner

The second goal of this thesis is to implement a docking task for the AUV in order to enable recharging from a seabed station. This task is crucial for the AUV to work autonomously without cable connection or necessary rising to the see surface to recharge. The seabed station (shown in Figure 30) will be placed on the sea floor in an accessible area and the AUV has to stop above the station and connect to charging device (module).

From the previously described motion planning task on long distances, the docking task is different in the fact that also the velocities $\boldsymbol{\nu} = [u, v, w, p, q, r]^T$ of AUV must be considered. Then the goal of the docking planner is to stop in a specified position, where the docking device is placed, without collisions with docking station or other obstacles.

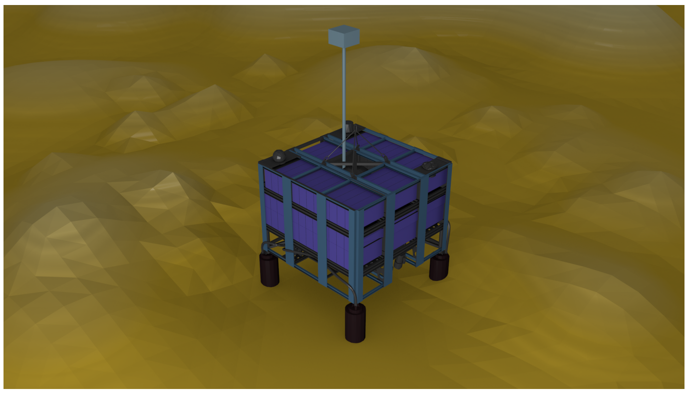


Figure 30: Seabed station with charging device (courtesy of David Oertel)

As we have described before in Chapter 4.1, the dynamical diving is possible only with forward speed above 0.51 m/s. This limitation is the main reason for selecting the following strategy for the docking task. In order to stop in a certain position, we expect that the planner firstly drives the AUV to desired depth and afterwards the final approach to docking device is ideally on a straight path with deceleration. Using this strategy we tested both previously introduced long distance planners with slight modifications that enable to plan with respect to AUV velocities.

4.3.1 Input-based docking

We firstly explored the possibility of using Input-based planner for solving the docking task. For using this planner, we needed to make some improvements, because the docking task differs in several key aspects. The first is that we need to control the AUV speeds in order to stop above the docking device. Secondly, we need a very precise motion plan that stops accurately where it is desired.

As the docking task needs to consider the AUV velocities, we introduced them to the planner. In the simulation function *simulateAUV* of the Input-based RRT Algorithm 3 we used a linear decrease of $shaft\_RPM$ based on distance to goal position of the docking device. In long distance Input-based method, we used a constant shaft speed, because it creates almost constant AUV forward speeds. In the Input-based docking task, we calculate the distance to the docking device in every step of the simulation and based on its value we set the $shaft\_RPM$.

To make the Input-based method work for the docking task, we also modified some parameters. The simplest modification was to significantly decrease the maximal allowed final distance d_{goal} to 5 cm. The function $random\_position()$ needs to generate only positions that are in the area for docking, otherwise the planning tree would grow unnecessarily to other parts of the workspace. We increased the simulation time t_{sim} that is used for each rudder input up to 30 s, but together with this modification we also added an interruption of the simulation when it drives more than four meters from the nearest neighbour state s_{near}. The increased simulation time and interruption of the simulation were used, because in positions close to the docking device the AUV is moving slowly and we need to run the simulation long enough to change the position. On the other hand, in positions far from docking device the AUV is moving fast and we need to limit the planning tree growth.

Using modification described above, we were able to plan the docking task so that the AUV starts in the same depth as the docking device and it stops over it. In Figure 31 is shown an example of computed plan.

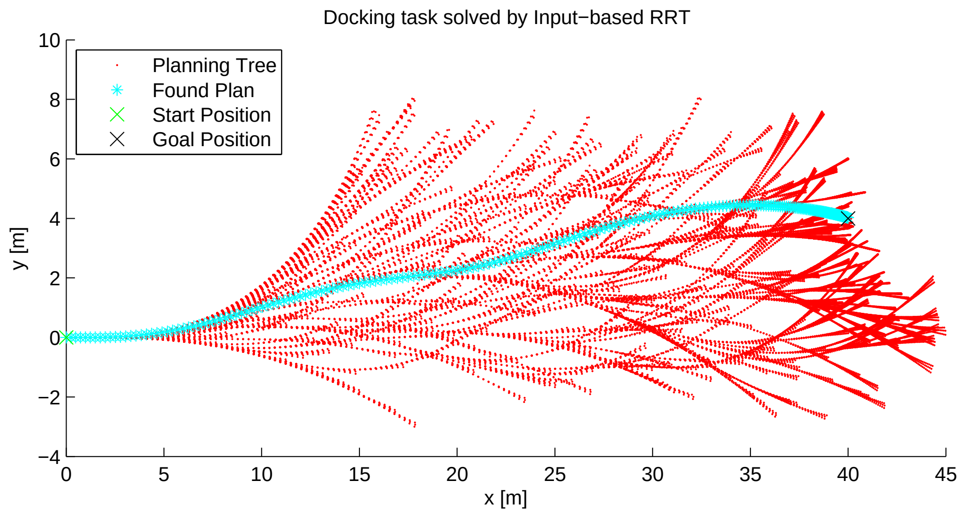


Figure 31: Path found for docking with Input-based planner

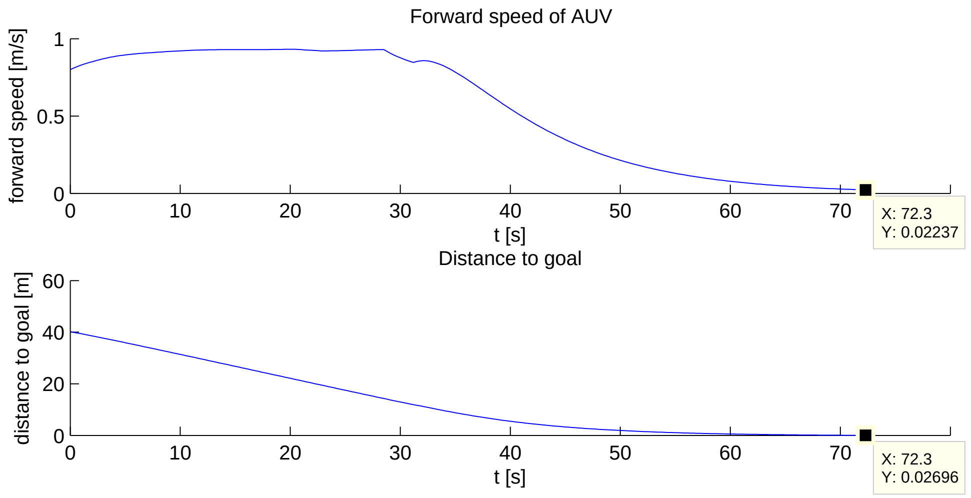


Figure 32: AUV forward speed and distance to docking device

The final speed and distance to the charging device are very important in the docking task. These values are supposed to be zero in order to enable successful connection to the docking station. Figure 32 shows progress of both speed and distance from goal for plan from Figure 31. The final distance and speeds achieved in shown experiment were 2.24 m/s and 2.7 cm. Because the docking device is still in the development and the maximal allowed error in position and speed is not specified, we assume that achieved accuracy is sufficient for the docking.

As the Input-based RRT planner is randomized algorithm, we ran the experiment multiple times and measured the results. The planner was able to find the plan in 70 % of trials, which is less than for the long distance planning, but for the required

$d_{goal} = 5$ cm maximal allowed final distance from goal is this success rate satisfactory. Figure 33 shows achieved results only for the successful runs, because the failed runs makes the other unreadable in the graph.

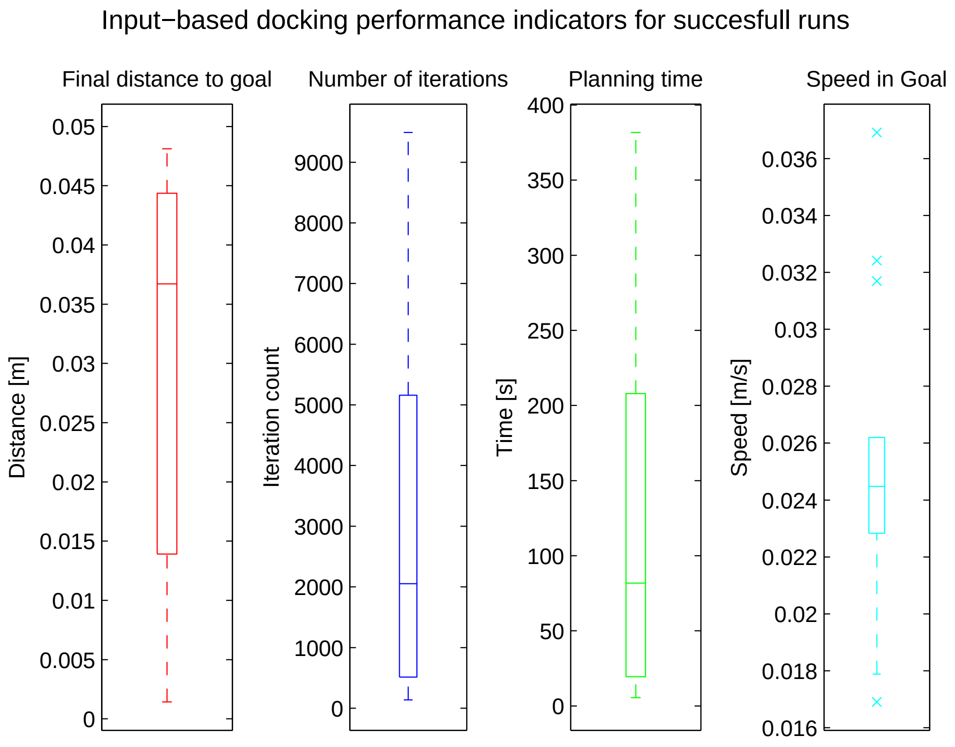


Figure 33: Performance of Input-based method in AUV docking task

As we can see, the average error in the final position is approximately 4 cm, but there were also some plans with error much lower. The final speed in the goal was in average 2.5 cm/s.

Based on the measured data, we can say, that the input-based planner is suitable not only for the long distance planning, but also for the short distance docking task. The method was able to almost stop the AUV in aproximately 70 s with final distance error of 4 cm.

4.3.2 Dubins-based docking

The second planner we tested for the docking task is the Dubins-based method described in Chapter 4.2.2. In order to get a precise final approach to the docking device, we modified both the Dubins curve local planner and the Lookahead steering controller. The Dubins curve generator was modified to create a path with bigger turning radius $r = 20$ m and also much slower diving speed per forward driven path. For the Lookahead steering control it is then easier to drive such a path, because it does not have to use the maximal rudder inputs in order to follow the path. The previously described Lookahead steering controller was adjusted for the docking task by reduction of proportional gains in order to get less greedy controller that is good for driving with minimal turning radius, but is not so good for precision of the docking task.

The Figure 34 shows docking with the Dubins-based method with Lookahead steering controller.

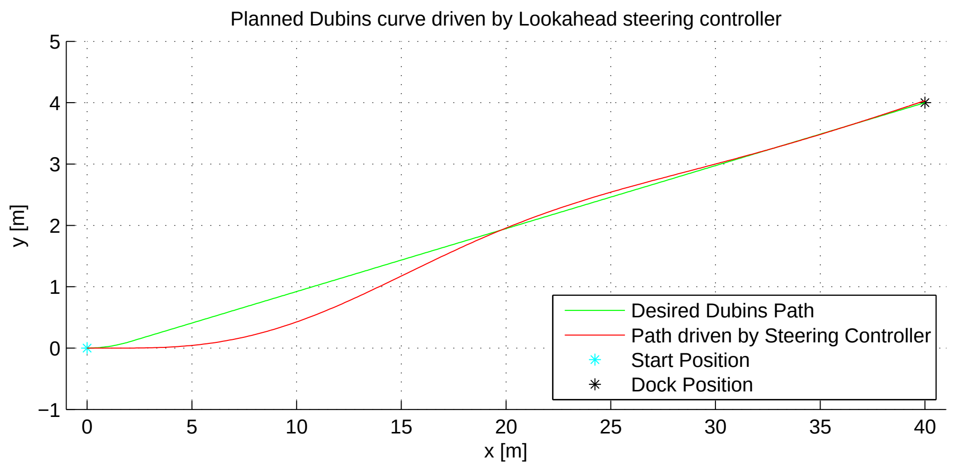


Figure 34: Docking with Dubins-based planner and Lookahead steering controller

The final speed and distance from docking device is shown in Figure 35. As we can see the Lookahead controller was able to slow down the AUV to final speed of 3 mm/s. The final error in distance to goal was approximately 3.2 cm.

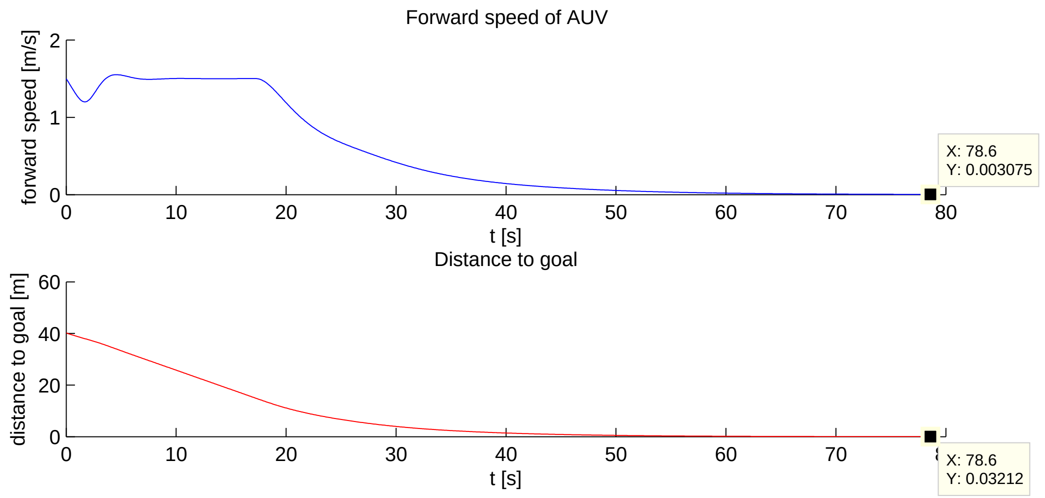


Figure 35: Docking with Dubins-based planner and Lookahead steering controller

The Dubins-based planner with Lookahead steering controller performed with same final error distance to the docking device as the input-based planner. The final speed was significantly smaller compared to the input-based planner which is probably caused by fact, that the input-based method uses only the proportional control of speed based on distance to the goal. The Lookahead steering controller uses the proportional-integral control which results in more precise speed control.

As we showed above, both long distance planners are useful for the AUV docking task and require only few modification to work properly. By introducing the AUV velocities to the planner, we were able to stop above the docking device with distance error less than 5 cm. To further decrease the final speed, the thrust motor could be driven in a reverse motion. This is however not considered in the employed motion model, so this option was not tested in this thesis.

5 Seabed monitoring

The last objective of this work is to design a mission planning system for up to 3 AUVs for the seabed monitoring task. The overall goal is to utilize multiple AUVs for scanning the whole seabed in a specified area with on-board sensors. For maintaining continuous operation of the AUVs, the mission planning system has to consider the AUV state of charge and drives the AUVs to the docking station for recharging if needed. Possible applications of described task are autonomous oceanographic imaging, deep sea mining or surveillance of underwater constructions such as power plant bases, offshore drilling and submarine cables.

In the designed mission planning system, we need to use the previously implemented motion planners. In order to plan the mission that contains visiting of specified positions, we need the long distance motion planner that we designed and described in Chapter 4.2. To enable the recharging from the seabed station, we previously implemented the docking motion planner in Chapter 4.3. The docking motion planner was designed by modifying the long distance motion planners to stop the AUV in desired position above charging device.

In Chapter 3.2 we showed several multi-robot approaches for the Robot Coverage Path Planning. We selected the Spanning-Tree Coverage as the main inspiration for the coverage part of the mission planning system. For simplification of the coverage planning for the AUV, we plan the coverage only in planar 2D space. The final coverage path in 3D can be then constructed from the 2D coverage plan by placing the 2D coverage plane to required distance above the seabed. This plane can be also modified to follow the seabed ascent and descent as long as the depth change is relatively smaller than the distance in 2D plane. All Spanning-Tree coverage approaches use the grid-based decomposition of the workspace for its planning. For the AUV the main restriction in the Spanning-Tree coverage is the nonholonomicity of the robot. We have to use a grid cells large enough, so that the AUV is able to turn inside the cell. From the grid cell size and the distance of the planar coverage above the seabed, we can derive which sensor we can use for the coverage in order to cover the entire cell with the sensor signal.

Although the Spanning-Tree approach would work in the coverage path planning for AUVs, we have to consider an additional methods to allow the AUVs to recharge during the mission, because the traditional coverage methods do not consider this option. The additional method is a discrete task planning, which is able to create a mission plan that consists of several different tasks such as the coverage and the recharging. However, using only the task planning approach is also not possible, because for the coverage planning it leads to the NP-hard Travelling Salesman Problem which is not solvable for grids with hundreds of cells.

As we can not use only the robot coverage path planning or only the task planning, we designed a mission planner for AUVs that combines both Spanning-Tree coverage planning and the task planning. In Chapter 5.1 we describe how the combined mission planner works and then we show the method performance in concrete scenario in Chapter 5.2.

5.1 Designed mission planning method

The task of the vehicles in the considered seabed monitoring scenario is to cover a predefined area. The vehicles start in a known depot (usually a docking station) and they have to cooperatively visit the area. As the considered missions can cover large area, it is required that the AUVs can automatically return to the docking station if necessary.

The novel method designed in this thesis for mission planning for up to 3 AUVs is combination of the multi-robot coverage and task planning approaches. As we discussed before, the multi-robot coverage methods do not deal with possible returns to the docking station in order to charge the robot. Also using only the task planning approach is not possible, because the number of possible solutions grows rapidly with the size of covering area.

We propose a new approach that is inspired by the Multi-robot Forest Coverage (MFC) and Spanning-Tree Coverage (STC) from Chapters 3.1 and 3.2. In the same time the approach also uses a task planning for search of feasible coverage mission plan that not only contains the coverage of entire area, but also charges the AUVs during the mission if it is necessary. To compute the mission plan the proposed method firstly decomposes the area into a grid and finds multiple tree graphs that cover the entire grid. Afterwards, we use the A* algorithm for discrete task planning that uses multiple possible actions like CHARGING, DRIVING_TREE(i) and WAITING to plan the mission. Finally the found mission plan, that consists of these actions, covers the entire area by circumnavigating the trees and also keeps the AUVs charged.

Our method starts with splitting the area using the Grid-Based Decomposition 3.1.2. The result is an adjacency graph of rectangular cells with side length of l_s. This parameter has to be chosen based on the AUV dynamic constrains, because we need the AUV to be able to move between centres of adjacent cells. We also have to establish a second grid that consists of mega cells, the same we introduced in STC method description in Chapter 3.1. Once we have the grid adjacency graph of the area, we can find the minimal spanning tree graph $G = (V, E)$ inside the graph of mega cells grid. The spanning tree graph contains the vertices V, which are the centres of mega cells, and the minimal number of edges E to connect these vertices.

As the key future of our mission planning is the ability to return to docking station for recharging, we choose the root vertex R_g of the spanning tree graph to be the cell where is placed the docking station. By using this spanning tree root, we know that by circumnavigating around the spanning tree we always return to the docking station. We use the Prim's algorithm for the spanning tree search. It uses the distance between cells as a weights of edge in the adjacency graph and finds the minimal spanning tree which then creates the minimal coverage of the area. By adjusting those weights a little bit, for example in hundredths, we can modify the resulting spanning tree, while the minimality stays unchanged. We increased each weight of edge based on its distance from the cell with docking station. Using higher weight increase for edges far from the docking station caused, that the Prim's algorithm always tries to add firstly the edges closes to the dock, because they have smaller weights. The resulting spanning tree is branching more around the root compared to the unchanged edges weights and also the maximal depth of the spanning tree is reduced.

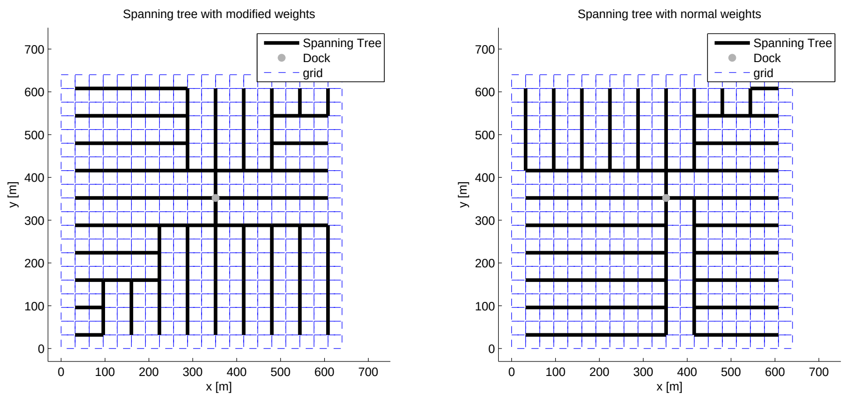


Figure 36: Comparison of spanning trees with modified and unmodified weights

Previous adjustments of the spanning tree search are essential, because after the spanning tree is found, we recursively go through the tree in order to decompose it into smaller trees. The inspiration by the Multi-robot Forest Coverage (MFC) is because the MFC approach also decomposes the spanning tree in order to get one tree for each robot (described in Chapter 3.2). Our method decomposes the spanning tree into unspecified number of trees to get multiple trees, where each tree has number of edges below specified value and is rooted in the original minimal spanning tree root. Starting from the original spanning tree root, we recursively go deeper into the spanning tree while controlling the number of edges rooted in actual node and the depth of actual node in order to get trees with specified number of edges.

Algorithm 5: Spanning tree decomposition

```
input : Spanning tree graph $G = (V, E)$ with root $R_g$ and maximal allowed
        number of edges $n_e$
output: Set of tree graphs $S$ rooted in $R_g$

decomposeSpanningTree($G$ , $n_e$)
    $S = \emptyset$;
    $S$ = decomposeNode($G, S, n_e, R_g, 0$) ;
    return $S$
decomposeNode($G, S, n_e, actualVertex, depth$)
    $numChildEdges$ = countSubtreeEdges($G, actualTreeVertex$);
    if $numChildEdges + depth \leq n_e$ then
        $newTree$ = vertexSubtree($G, actualVertex$);;
        $S = S \cup newTree$;
    else
        foreach $child$ vertex of $actualTreeVertex$ do
            $S = S \cup$ decomposeNode($G, S, n_e, child, depth + 1$);
        end
    end
    return $S$
```

As we can see in the Algorithm 5, the decomposition starts with empty set of trees and then go recursively through the spanning tree and creates trees with maximal number of edges n_e. Function countSubtreeEdges$(G, actualTreeVertex)$ at line 6 returns the number of edges in subtree rooted in vertex $actualTreeVertex$. Every time the node decomposition calculates that the $actualTreeVertex$ would create a tree with less of equal number of edges than required by n_e, then the tree is decomposed in this vertex. The Function vertexSubtree$(G, actualVertex)$ at line 8 returns such a decomposed tree consisting of both tree rooted in $actualVertex$ and edges between $actualVertex$ and the root of spanning tree R_g.

Figure 37 shows the decomposed spanning tree into multiple trees that covers the area. All trees obtained from the decomposition are rooted in the position of the docking station. In this example of tree decomposition, the maximal allowed number of edges was set to $n_e = 12$ and the algorithm produced trees with maximally ten edges (blue in the Figure). As the minimal turning diameter of AUV is below 30 m, we used cell decomposition with cell side length $l_s = 32$ m.

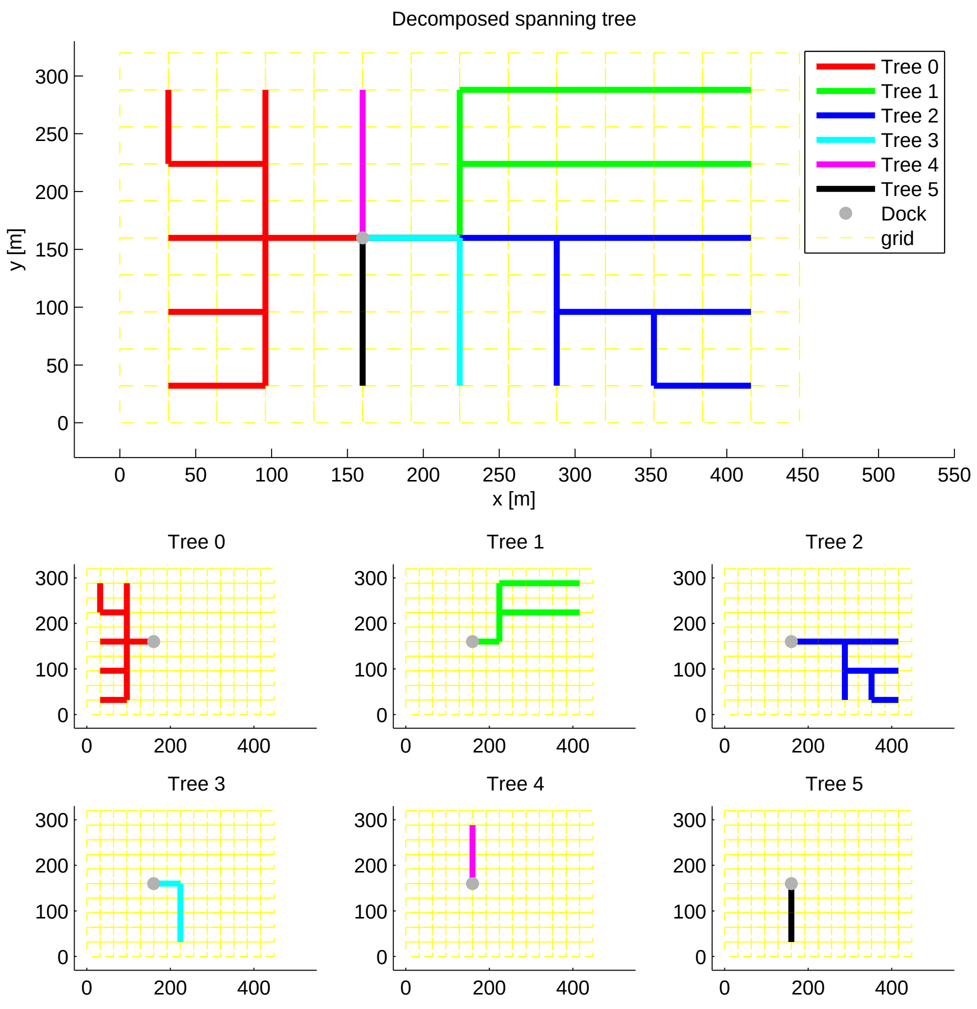


Figure 37: Decomposed spanning tree into multiple trees

Once we have the set of trees covering the area, we can use a task planning that creates the mission plan to cover the area while staying in the operational state by recharging. The parameter of maximal allowed number of edges in tree n_e used in spanning tree decomposition is very important and relates to the requirement to recharge the AUVs during the mission. As in classical STC, the path for robot is created by circumnavigation around the spanning tree in counterclockwise direction. We use the same strategy on the decomposed trees. When we specify the n_e, we also know that the path that circumnavigates the tree is approximately $2 \cdot n_e \cdot l_m$, where $l_m = 2 \cdot l_s = 64$ m is the length of the tree edge that connects the mega cells. This we know, because the AUV has to pass along one edge exactly twice while circumnavigating the tree. Then we need the maximal distance that AUV can drive during one charge cycle and we are able to determine the parameter n_e. It means that each decomposed tree can be circumnavigated by initially fully charged AUV without risk of getting discharged during the mission.

As we mentioned before, we use several possible actions in the discrete task planning to obtain the mission plan. The actions are following.

- **DRIVING_TREE(i)** - the action represents that the AUV is actually allocated for circumnavigation of tree with id i. This action can not be interrupted during execution and has to be completed before AUV can select next action. The AUV also needs to be charged enough to start the action for given tree.
- **WAITING** - action that is used in initial and final planning state. Beside this, it can be also used when one AUV is already charging and the other has to wait for recharging, because we expect only one concurrently charging AUV. The action can be interrupted in any time.
- **CHARGING** - represents the charging of the AUV. Can be interrupted any time.

Above mentioned actions are actually a part of the state in which are the AUVs. The state further contains a parameter that represents how much of the action is already competed (further denoted as completeness) and also the state of charge (SoC) of AUV. Then for each AUV we can describe its state as $(action, completeness, SoC)$ vector, where *action* is one of the three action types and both *completeness* and *Soc* are in range $\langle 0, 1\rangle$. The zero value of *completeness* means that the action has not started yet and the value of 1 means that the action is completed. The same applies to *SoC*, where zero value means fully discharged and one means fully charged. The action DRIVING_TREE(i) is in fact a representation of multiple actions, because it depends on which tree we use the action on. For WAITING action is the *completeness* and *SoC* unnecessary, because we always consider the action to be completed and for simplicity we do not apply any discharge when the AUV is WAITING.

Diagram in Figure 38 is a state transition diagram, that is used for each AUV inside the mission planner.

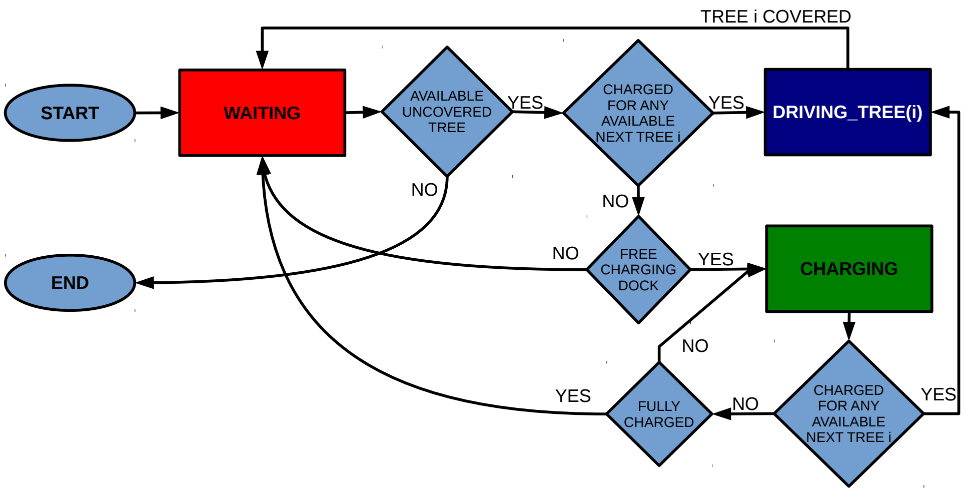


Figure 38: State transition diagram for a single AUV in task mission planner

For the mission planner, we also use a state of the mission m_n, that consists of all AUV states and also of tree ids i that are not yet covered. The initial state of mission is when all AUVs are WAITING close to the seabed station and no tree is covered. The goal state of the mission is when all AUVs are WAITING and all trees are covered, which means that also the whole area is covered. Using these initial and goal mission states, the discrete task planner implemented as the A* search has to find a sequence of mission states that leads from the initial to the goal state. For the A* algorithm we use the cost function $f(m_n) = g(m_n) + h(m_n)$, that represents a time that it would take to finish the mission in state m_n. The $g(n)$ is the time that it took to get from initial mission state to the actual mission state m_n. The heuristic $h(m_n)$ is estimated time to the goal state from state m_n. For our mission state the $g(m_n)$ is sum of times that the AUVs needed for already done actions. Time needed for DRIVING_TREE(i) action is always calculated based on the tree i length of circumnavigation path and used average speed. Both WAITING and CHARGING actions do not hold predefined time, because they can be interrupted in any time. Their time length is always calculated during expansion to the next state. The estimated time to goal state $h(m_n)$ can be calculated based on length of still uncovered trees and time we need to spend in CHARGING action in order to cover those uncovered trees. Using the described cost function $f(m_n)$, the task planning algorithm works as the normal A* search. It always expands the mission state from open set with the smallest cost and keeps doing so until the goal mission state is expanded.

Algorithm 6 shows the expansion of single mission state that is used inside the A* discrete task planning. It uses the state transition from Figure 38 for each AUV to obtain next possible AUV states ($getNextStates(m_n)$ at line 3 of Algorithm 6). Using Cartesian product of those next AUV states we create a set of next mission states. All states are also checked for feasibility, because the states with for example multiple AUVs in CHARGING action or DRIVING_TREE(i) same tree are not allowed. Finally we also has to recalculate the cost function $f(m_n)$ for each newly expanded state.

Algorithm 6: Mission state expansion

```
    input  : Mission state $m_n$, number of AUVs $k$
    output: Set $M$ of mission states expanded from input mission state $m_n$
 1  $M = \{\}$;
 2  foreach $AUV\ j = 1...k$ do
 3      $nextAUVStates_j = getNextAUVStates(m_n)$;
 4  end
 5  $S_{possible} = CartesianProduct(nextAUVStates)$;
 6  foreach $newstate\ in\ S_{possible}$ do
 7      if $isfeasible(newstate)$ then
 8          $recalculateCost(newstate)$;
 9          $M = M \cup newstate$;
10      end
11  end
12  return $M$;
```

Using previously described expansion of mission state in the A* task planning algorithm, our method is able to plan the coverage mission. Final task plan consist of action schedule for each AUV. The A* algorithm also minimizes the time needed to complete the mission while satisfying the mission requirements. All AUVs stay operational by recharging during the mission from the seabed station. Figure 39 shows possible mission plan, which is the outcome of the A* task planning.

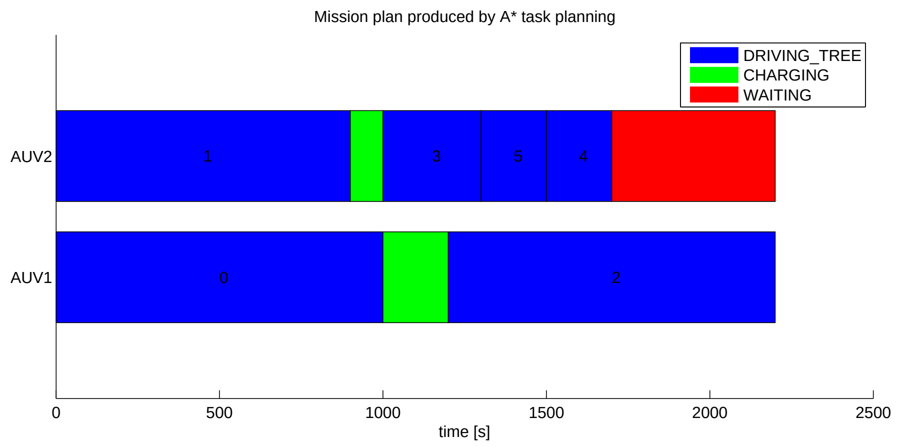


Figure 39: Example mission plan produced by A* task planning (the number in actions DRIVING_TREE(i) represents id of tree from Figure 37)

The final part of our approach is the path planning for the AUVs based on the mission plan. For the action DRIVING_TREE(i), we can find a path that circumnavigates the specified tree i by one of our long distance motion planning approaches from Chapter 4.2. Because of the faster performance, we used the Dubins-based method to get the path that circumnavigates the tree using Dubins curves between centres of gird cells. Figure 40 shows how a single decomposed tree is circumnavigated using the Dubins-based motion planner.

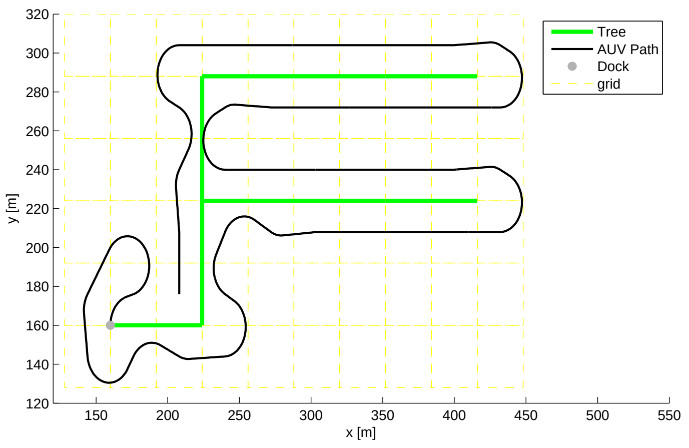


Figure 40: Circumnavigation path of a single decomposed tree

Finally the mission planner uses also the docking task from Chapter 4.3 to plan the connection to the docking device. The CHARGING action, we used in the task planner, consists of both docking task and then the actual charging of the AUV.

In next chapter, we present the experimental results obtained by using the designed mission planner.

5.2 Experimental results

In previous chapter, we proposed a novel mission planner for up to 3 AUVs in the seabed monitoring task. The key features of planned missions are coverage of the entire selected area and also recharging of the AUVs to keep them in operational state. In this section we show how the algorithm works in an experimental workspace with obstacles by simultaneously using 3 AUVs during the mission. For the experiment we use the same workspace as in Chapter 3 to allow comparison with other approaches for multi-robot coverage.

The algorithm starts with the grid-based decomposition of the desired area. For the experiment we used workspace with area of approximately 1 km^2, which resulted in a grid with almost one thousand cells. Using these cells we found the minimal spanning tree rooted in cell with docking station, which we placed to position $(x, y) = (160, 160)$ m. Then using the Algorithm 5 for decomposition of the spanning tree, we obtained 13 different trees that are all rooted in cell with the docking station. For the spanning tree decomposition algorithm we used maximal allowed number of edges $n_e = 93$. This number of edges was simply derived from maximal distance that AUV can drive when fully charged. For this value we used 6000 m although the real AUV can probably stay charged for longer paths. The resulting decomposed trees are shown in Figure 41.

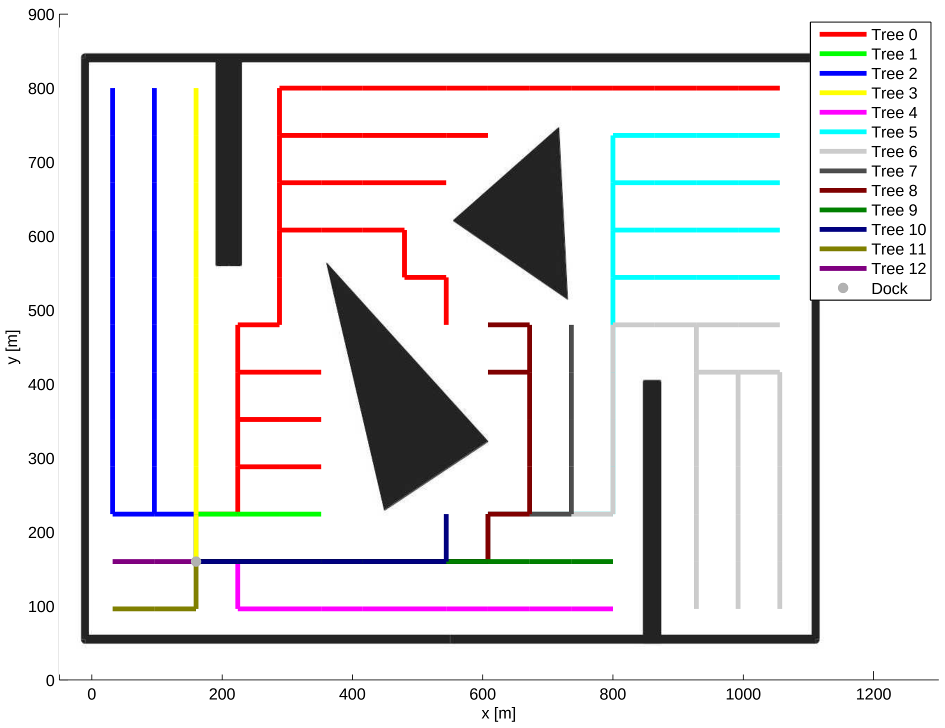


Figure 41: Example workspace covered by decomposed trees

With the decomposed trees, we were able to plan the whole mission to cover the entire workspace and in the same time recharge the AUVs if necessary. The resulting mission plan consist of action sequence for each AUV. The possible actions are DRIVING_TREE(i), CHARGING or WAITING, where the action for driving tree can be for one of the thirteen trees. The concrete plan obtained from the A* discrete task planner is shown in Figure 42. The types of actions in this figure are colour coded with number of driving tree for the action DRIVING_TREE(i). What is not visible in the mission plan is that in both initial and final states the AUVs have WAITING action.

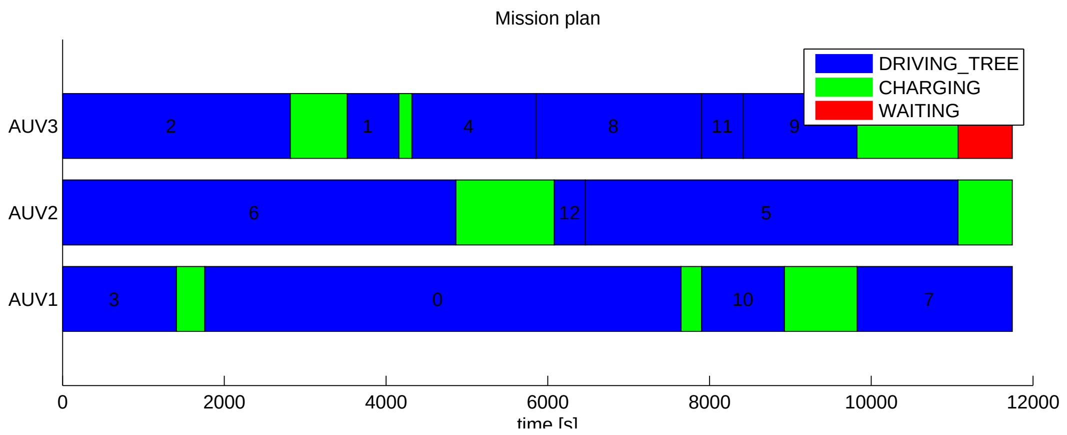


Figure 42: Found seabed monitoring mission plan

The main reason for using the task planning for the seabed monitoring is the ability to charge AUVs during the mission. It is important for the planner to keep the information about the state of charge of each AUV in order to plan the CHARING action before the AUV gets discharge. Additionally we also allow only one charging AUV a time. For the CHARGING action we used a charging speed four times larger than discharging speed. It means that when one AUV is circumnavigating the tree, it travels for one meter while in the same time the second AUV in CHARGING actions gets charged for additional four meters. The selection of charging speed is important for the algorithm, because it influences how often the AUVs have to expand to the CHARGING state. With too small charging speed, the mission plan would contain plans with AUVs WAITING for recharging. Figure 43 shows for how many meters are the AUVs charged during the found mission. We can clearly see, that the AUVs are discharging during the DRIVING_TREE(i) action. Also the higher charging than discharging speed can be observed from comparison of the graph inclination in CHARGING and DRIVING_TREE(i) actions. Finally we can also see, that the WAITING action has no effects on the state of charge, as we expect no discharge during this action.

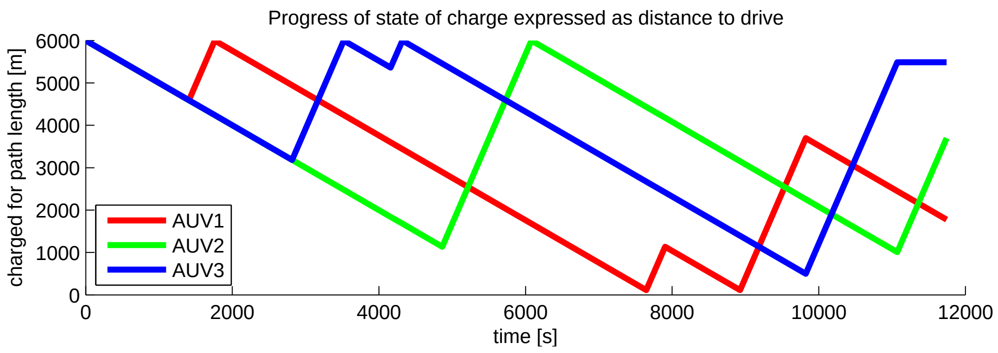


Figure 43: AUVs state of charge in mission plan

The overall goal of the mission was to cover the entire workspace. Once we have the task plan, we can find a path plan for each AUV. In the DRIVING_TREE(i) action we use the tree i for circumnavigation. The actual path that circumnavigates the tree is created from segments that are in fact the Dubins curves between centres of cells that surrounds the tree. For the action CHARGING, the AUV plans its path to the position of the seabed station, where the actual docking can be done by using one of the short distance docking planners from Chapter 4.3. Also the planning of the tree circumnavigation can be done not only by using Dubins-based planner, but also by the Input-based AUV planner from Chapter 4.2.1. Figure 44 shows the entire path for each AUV in the experiment. We can see that around the seabed station, the area is used many times by all AUVs. This is caused by the necessary charging from the seabed station. We can also see that the entire workspace is covered by the AUV paths.

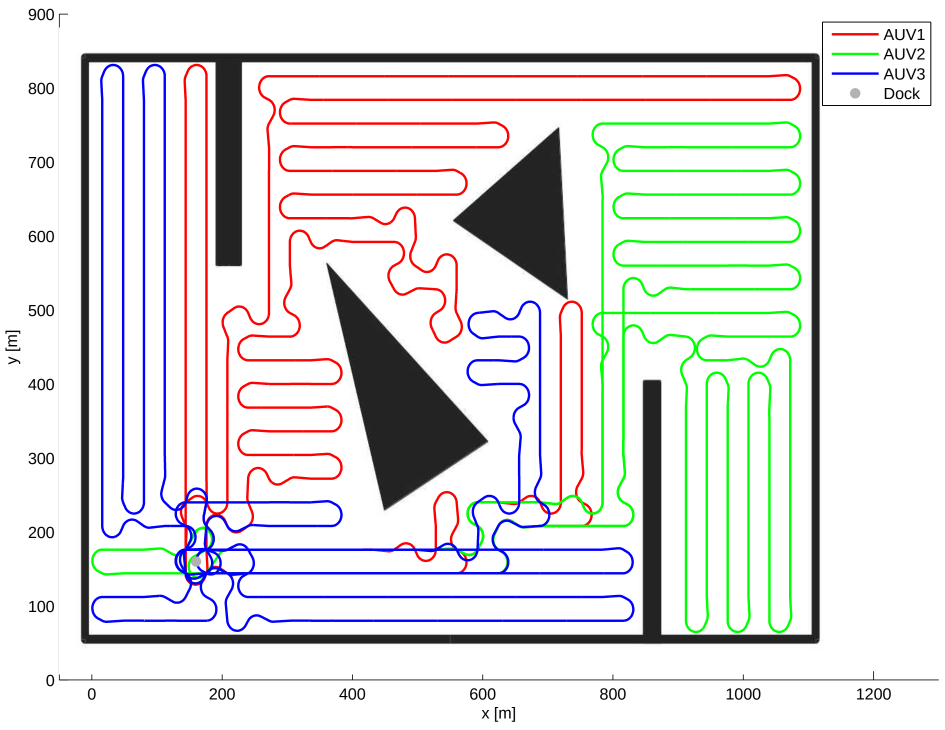


Figure 44: Paths taken by AUVs in coverage mission

The performed experiment showed, that the designed mission planner for AUVs fulfils the task. During the mission, all AUVs stayed in operation state by recharging from the seabed station and the AUVs covered the whole workspace.

6 Conclusion

In this thesis, we developed a solution for mission planning for up to three AUVs in the seabed monitoring task. The outcome plans consist of visiting all positions in given area with necessary returns to docking station for recharging.

The first key features of designed mission planner is the ability to navigate the AUV to desired position without collision. For this purpose, we designed two different long distance motion planners. The first Input-Based planner uses the RRT-blossom approach to find the motion plan for AUV. By applying multiple possible inputs to the AUV dynamical model from randomly selected AUV state, the RRT algorithm finds the plan between start and goal position. The second Dubins-based planner uses the Dubins curves as a motion primitives inside the RRT path planner to generate path plan. The final motion plan is then created by Lookahead steering controller, that traverse the previously found plan. By comparing the experimental results of both planners, we found out that both planners are suitable for AUV motion planning. The Input-Based planner generated shorter paths than the Dubins-based planner. On the other hand, the Dubins-base method is nearly four times less memory consuming and also nearly 23× faster than the Input-Based method.

The second key feature required in the missions is the ability to dock the AUV to a charging station. For the docking task, we modified previously used Input-Based and Dubins-based planners to create more precise plans for short distances. In the long distance motion plans we used average speeds of the AUV. For the docking task, we need to stop the AUV next to the docking device in order to start the charging. By introducing the AUV velocities to the planners, we were able to plan the docking motion with both planners, with approximately the same resulting accuracy.

Finally we introduced a novel mission planner together with experimental results. Current robot coverage path planners do not count with possible returns to the docking station for recharging. Also using only discrete task planning is not possible for robot coverage path planning, because it leads to NP-hard problem which is unsolvable for hundreds of cells. The proposed mission planner combines the Spanning-Tree coverage path planning with the discrete task planning to plan the seabed monitoring missions. Using the grid as a representation of given area, the mission planner firstly decomposes the minimal spanning tree of the grid into smaller trees. Then it uses the A* algorithm for search in state space of AUVs actions to find the mission plan. Possible actions for the AUVs are to cover one of decomposed trees, to charge from the seabed station or to wait. The final mission plan consists of sequence of these actions for each AUV. The coverage of given area is minimal possible with respect to the constraints, such as the maximal driven distance per one charging cycle or restriction to charge only on AUV in same time.

For the future work, we would like to introduce other types of actions to the mission planner, so that the mission plan can contain arbitrary actions such as driving to the sea level or even manipulation with underwater objects. Other challenging task we would like to try is implementation of designed coverage mission planner on different robotic platforms.

List of Figures

List of Algorithms

References

[1] Jérîme Barraquand and Jean-Claude Latombe. Robot motion planning: A distributed representation approach. *Int. J. Rob. Res.*, 10(6):628–649, December 1991.

[2] C.V. Caldwell, D.D. Dunlap, and E.G. Collins. Motion planning for an autonomous underwater vehicle via sampling based model predictive control. In *OCEANS 2010*, pages 1–6, Sept 2010.

[3] Ipek Caliskanelli, Bastian Broecker, and Karl Tuyls. Multi-robot coverage: A bee pheromone signalling approach. In Christopher J. Headleand, William J. Teahan, and Llyr Ap Cenydd, editors, *Artificial Life and Intelligent Agents*, volume 519 of *Communications in Computer and Information Science*, pages 124–140. Springer International Publishing, 2015.

[4] John Canny. A voronoi method for the piano-movers problem. In *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, volume 2, pages 530–535, Mar 1985.

[5] Chieh Chen and Han-Shue Tan. Steering control of high speed vehicles: dynamic look ahead and yaw rate feedback. In *Decision and Control, 1998. Proceedings of the 37th IEEE Conference on*, volume 1, pages 1025–1030 vol.1, 1998.

[6] H. Choset, E. Acar, A.A. Rizzi, and J. Luntz. Exact cellular decompositions in terms of critical points of morse functions. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 3, pages 2270–2277 vol.3, 2000.

[7] Howie Choset. Coverage for robotics – a survey of recent results. *Annals of Mathematics and Artificial Intelligence*, 31(1-4):113–126, 2001.

[8] Howie Choset and Philippe Pignon. Coverage path planning: The boustrophedon decomposition. In *International Conference on Field and Service Robotics*, 1997.

[9] Thor I. Fossen. *Handbook of marine craft hydrodynamics and motion control.* John Wiley & Sons, 2011.

[10] Thor I. FosseN. Mathematical models of ships and underwater vehicles. In John Baillieul and Tariq Samad, editors, *Encyclopedia of Systems and Control*, pages 1–9. Springer London, 2014.

[11] J. Könemann R. Ravi G. Even, N. Garg and A. Sinha. Min-max tree covers of graphs. In *Operations Research Letters*, pages 309–315, 2004.

[12] Y. Gabriely and E. Rimon. Spiral-stc: an on-line coverage algorithm of grid environments by a mobile robot. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, volume 1, pages 954–960 vol.1, 2002.

[13] Yoav Gabriely and Elon Rimon. Spanning-tree based coverage of continuous areas by a mobile robot. *Annals of Mathematics and Artificial Intelligence*, 31(1-4):77–98, 2001.

[14] E. Galceran and M. Carreras. Efficient seabed coverage path planning for asvs and auvs. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 88–93, Oct 2012.

[15] Enric Galceran and Marc Carreras. A survey on coverage path planning for robotics. *Robot. Auton. Syst.*, 61(12):1258–1276, December 2013.

[16] Faming Gong and Xin Wang. Robot path-planning based on triangulation tracing. In *Intelligent Information Technology Application Workshops, 2008. IITAW '08. International Symposium on*, pages 713–716, Dec 2008.

[17] E. Gonzalez, O. Alvarez, Y. Diaz, C. Parra, and C. Bustacara. Bsa: A complete coverage algorithm. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 2040–2044, April 2005.

[18] I.A. Hameed, A. la Cour-Harbo, and O.L. Osen. Side-to-side 3d coverage path planning approach for agricultural robots to minimize skip/overlap areas between swaths. *Robotics and Autonomous Systems*, 76:36 – 45, 2016.

[19] N. Hazon and G.A. Kaminka. Redundancy, efficiency and robustness in multi-robot coverage. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 735–741, April 2005.

[20] W.H. Huang. Optimal line-sweep-based decompositions for coverage algorithms. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 1, pages 27–32 vol.1, 2001.

[21] M. Kalisiak and M. van de Panne. Rrt-blossom: Rrt with a local flood-fill behavior. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1237–1242, May 2006.

[22] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. Journal of Robotics Research*, 30(7):846–894, June 2011.

[23] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, Aug 1996.

[24] D. Kiss, G. Csorvasi, and A. Nagy. A planning method to obtain good quality paths for autonomous cars. In *Engineering of Computer Based Systems (ECBS-EERC), 2015 4th Eastern European Regional Conference on the*, pages 104–110, Aug 2015.

[25] J. Kosecka, R. Blasi, C.J. Taylor, and J. Malik. Vision-based lateral control of vehicles. In *Intelligent Transportation System, 1997. ITSC '97., IEEE Conference on*, pages 900–905, Nov 1997.

[26] K.D. Kotay and D.L. Rus. Algorithms for self-reconfiguring molecule motion planning. In *Intelligent Robots and Systems, 2000. (IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, volume 3, pages 2184–2193 vol.3, 2000.

[27] J.J. Kuffner and S.M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 2, pages 995–1001 vol.2, 2000.

[28] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[29] Steven M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, 1998.

[30] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, New York, NY, USA, 2006.

[31] Steven M. Lavalle, James J. Kuffner, and Jr. Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions*, pages 293–308, 2000.

[32] Hyoung-Ki Lee, WooYeon Jeong, Sujin Lee, and Jonghwa Won. A hierarchical path planning of cleaning robot based on grid map. In *Consumer Electronics (ICCE), 2013 IEEE International Conference on*, pages 76–77, Jan 2013.

[33] Chaomin Luo, Simon X. Yang, and Xiaobu Yuan. Real-time area-covering operations with obstacle avoidance for cleaning robots. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 3, pages 2359–2364 vol.3, 2002.

[34] Yingchong Ma, Gang Zheng, and W. Perruquetti. Cooperative path planning for mobile robots based on visibility graph. In *Control Conference (CCC), 2013 32nd Chinese*, pages 4915–4920, July 2013.

[35] Ronaldo Menezes, Francisco Martins, Francisca Emanuelle Vieira, Rafael Silva, and Márcio Braga. A model for terrain coverage inspired by ant's alarm pheromones. In *Proceedings of the 2007 ACM Symposium on Applied Computing*, SAC '07, pages 728–732, New York, NY, USA, 2007. ACM.

[36] Ekmanis M. Liekna A. Nikitenko, A. Rrts postprocessing for uncertain environments. In *Proceedings of the 2013 International Conference on Systems, Control and Informatics (SCI 2013)*, pages 171–179, 2013.

[37] A. Ntawumenyikizaba, Hoang Huu Viet, and TaeChoong Chung. An online complete coverage algorithm for cleaning robots based on boustrophedon motions and a* search. In *Information Science and Digital Content Technology (ICIDT), 2012 8th International Conference on*, volume 2, pages 401–405, June 2012.

[38] MinGyu Park and MinCheol Lee. A new technique to escape local minimum in artificial potential field based path planning. *KSME International Journal*, 17(12):1876–1885, 2003.

[39] Erwin Prassler, Arno Ritter, Christoph Schaeffer, and Paolo Fiorini. A short history of cleaning robots. *Autonomous Robots*, 9(3):211–226, 2000.

[40] Ioannis Rekleitis, AiPeng New, EdwardSamuel Rankin, and Howie Choset. Efficient boustrophedon multi-robot coverage: an algorithmic approach. *Annals of Mathematics and Artificial Intelligence*, 52(2-4):109–142, 2008.

[41] Z. Sadeghi and H. Moradi. A new sample-based strategy for narrow passage detection. In *Intelligent Control and Automation (WCICA), 2011 9th World Congress on*, pages 1059–1064, June 2011.

[42] M. Saha and J.-C. Latombe. Finding narrow passages with probabilistic roadmaps: the small step retraction method. In *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 622–627, Aug 2005.

[43] Subramanian Saravanakumar and Thondiyath Asokan. Multipoint potential field method for path planning of autonomous underwater vehicles in 3d space. *Intelligent Service Robotics*, 6(4):211–224, 2013.

[44] K.S. Senthilkumar and K.K. Bharadwaj. Spanning tree based terrain coverage by multi robots in unknown environments. In *India Conference, 2008. INDICON 2008. Annual IEEE*, volume 1, pages 120–125, Dec 2008.

[45] K. Tanakitkorn, P.A. Wilson, S.R. Turnock, and A.B. Phillips. Grid-based ga path planning with improved cost function for an over-actuated hover-capable auv. In *Autonomous Underwater Vehicles (AUV), 2014 IEEE/OES*, pages 1–8, Oct 2014.

[46] C. Urmson and R. Simmons. Approaches for heuristically biasing rrt growth. In *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 2, pages 1178–1183 vol.2, Oct 2003.

[47] K. Vicencio, B. Davis, and I. Gentilini. Multi-goal path planning based on the generalized traveling salesman problem with neighborhoods. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 2985–2990, Sept 2014.

[48] M. Weiss-Cohen, I. Sirotin, and E. Rave. Lawn mowing system for known areas. In *Computational Intelligence for Modelling Control Automation, 2008 International Conference on*, pages 539–544, Dec 2008.

[49] Hongyang Yan, Huifang Wang, Yangzhou Chen, and G. Dai. Path planning based on constrained delaunay triangulation. In *Intelligent Control and Automation, 2008. WCICA 2008. 7th World Congress on*, pages 5168–5173, June 2008.

[50] Simon X. Yang and Chaomin Luo. A neural network approach to complete coverage path planning. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 34(1):718–724, Feb 2004.

[51] A. Yershova and S.M. LaValle. Improving motion-planning algorithms by efficient nearest-neighbor searching. *Robotics, IEEE Transactions on*, 23(1):151–157, Feb 2007.

[52] Liangjun Zhang and D. Manocha. An efficient retraction-based rrt planner. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 3743–3750, May 2008.

[53] Xiaoming Zheng, S. Jain, S. Koenig, and D. Kempe. Multi-robot forest coverage. In *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 3852–3857, Aug 2005.

Appendix CD Content

Directory or file name	**Description**
diplomaThesis.pfd	diploma thesis in pdf
matlab	dynamical model of AUVs, model analysis scripts, dubins curves scripts and others
src	source codes for motion and task planners
videos	simulation video