HTBLuVA Wiener Neustadt
Höhere Lehranstalt für Informatik

DIPLOMARBEIT

Autonomous universal Mapping and Navigation

Ausgeführt im Schuljahr 2020/21 von:

Themengebiet 1
Lukas Leskovar
5BHIF

Themengebiet 2
Fabian Kleinrad
5BHIF

Betreuer / Betreuerin:

MMag. Dr. Michael Stifter

Wiener Neustadt, am October 21, 2020/21

Abgabevermerk: Übernommen von:

Chapter 1

Eidestattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen als die im Literaturverzeichnis angegeben Quellen und Hilfsmittel verwendet habe. Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht habe.

Wiener Neustadt am October 21, 2020/21

Verfasser / Verfasserinnen:

Lukas Leskovar

Fabian Kleinrad

Contents

Chapter 2

Acknowledgement

The authors would like to thank ...

Chapter 3

Kurzfassung

asdf

Chapter 4

Abstract

asdf

Chapter 5

Introduction

Author: Lukas Leskovar

5.1 The Evolution of Robotics

Robotic research has always utilized concepts, processes, and methods of different scientific disciplines such as physics, mathematics, and biology to improve application and aid human needs. Because of this industrial, medical and even agricultural sectors have used technologies and products developed by researchers to improve workflows and alleviate employees from performing exhausting tasks. This relationship ranges back to the early ages of information technology in the 1950s and 1960s in which many developments on production robots and Artificial Intelligence (AI) have been made. Between 1970 and 1990 the public interest in automation and AI has decreased forcing the industry into the so-called AI winter. Despite this recession, research has been continued and the building blocks for another robot boom during the 1990s have been set. Since then the usage of robotic applications has broadened and the industry has proven itself to be a vital aspect of today's economy.

5.2 Robots with human interaction

Nowadays the utilization of robots in workplaces has broadened to almost every branch and is accepted by employees and workers. In countries like Japan, robots are no longer seen as a threat to jobs. Industrial robots are no longer used as simple construction tools, their safety and accuracy have improved so that collaborative robots (Cobots) are capable of working in close cooperation with humans. Surgery robots used in the medical sector not only allow for much more accurate procedures but also enable remote specialists to work on patients without having to be in the same hospital. In developed countries, educational robots are used at school or at home to teach children topics in a playful and interesting way.

[1] **malone2011unimate**

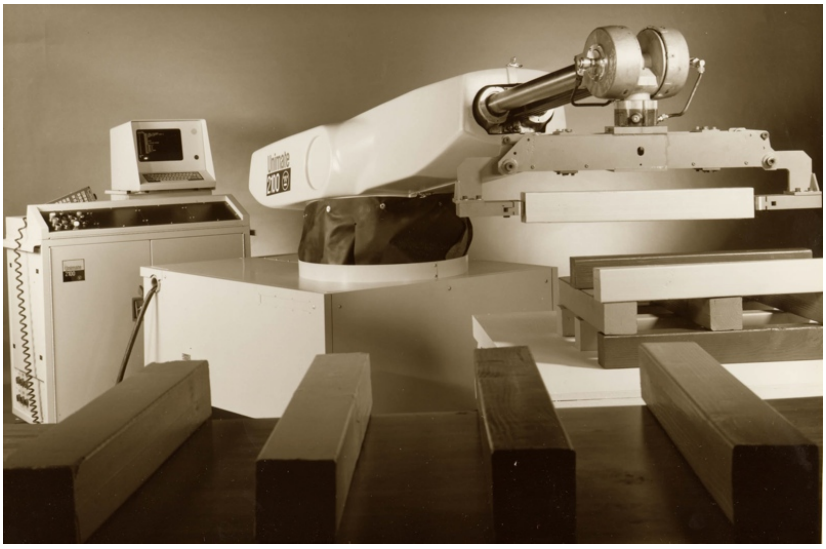


Figure 5.1: Picture of the first industrial robot. The Unimate developed by George Devol and Joseph Engelbert in 1961 was first used for hot die-casting and welding applications.[1]

5.3 Robots in hazardous environments

Robots do not only serve a purpose in a close-to-user work environment, they also ensure human safety by performing dangerous tasks in unsafe surroundings. Remotely controlled robots or drones can be used for inspecting mine shafts, collapsed buildings, pipelines, or overhead power poles. Other applications of such robots are bomb or mine defusion, fire extinction, or avalanche rescue.

5.4 Autonomous robots

Implementing an autonomous robot system is an intricate task that proposes many challenging problems for research or development teams. Autonomy requires a system to continuously work in a dynamic environment without external controlling inputs and utilize perceived information about its surroundings to adapt to environmental change.[2] Despite their complexity in development autonomous systems, mobile or stationary, immensely facilitate the execution of a job for the human user. Such robots can be used to navigate and organize warehouses, constructing parts in an assembly line, or map large areas for comprehensive calculations.

[2] **bekey2005autonomous**.

Figure 5.2: The Emesent Hovermap mapping a dangerous area inside a mineshaft.[5]

5.5 3D Mapping

5.6 Autonomous 3D Mapping

While 3D Mapping is a well-established and growing economy most applications require human interaction at some point during the mapping process. Products such as the Emesent Hovermap[3] or Exyn Aero[4] utilize the spatial flexibility of drones and high-end light detection and ranging (LiDAR) Sensors to facilitate autonomous mapping in GPS-denied areas without being within sight of the drone pilot. These solutions are used to autonomously map building or explore hazardous environments such as mineshafts in a human-safe manner as seen in Fig. 5.2.

5.7 Goals

The project associated with this thesis aims to implement a 3D Mapping system similar to the aforementioned solutions with limited financial, personnel as well as temporal resources. This goal forces the project team to primarily maintain an open-source approach during development.

The goal of this thesis is to document the challenges encountered and experiences gained by the project team during development to demonstrate how highly sophisticated industrial problems can be solved in a low-budget fashion.

[3] **hovermap2021**.
[4] **exynAero2021**.
[5] **hovermapIMG2021**

5.8 Requirements

In order to declare the project as successful the following criteria have to be implemented:

- The system is capable of generating a 3D Point-cloud of its surroundings
- All required hardware (e.g. sensors, computing boards, etc.) is mounted onto a drone-platform
- All calculations concerning the map-generation are run on a external server in direct communication with the drone
- The drone utilizes the Point-cloud to orientate in its surroundings and navigate one ore multiple waypoints
- The drone is capable of avoiding obstacles as it is moving through a unknown environment

The following criteria can be implemented but have no direct correlation to the success of the project:

- A Web-App facilitating the usage of the system and enabling the user to create waypoints, monitor the drone and mapping algorithm as well as evaluate the Point-cloud
- The system is replicated within the gazebo simulator to simplify future development without direct access to its hardware

Chapter 6

Study of Literature

Author:

Chapter 7

Robot Operating System

Author: Lukas Leskovar
This chapters objective is to describe the basic concepts of the Robot Operating System (ROS) utilized by Autumn. The ROS despite its name is a meta-operating system or middleware providing the utility and services often found in robotics frameworks. It enables the composition of distributed systems by utilizing publisher-subscriber communication between different programs of such systems. Furthermore ROS provides a comprehensive set of tools enabling the compilation, operation as well as testing, visualization and debugging of robotic systems. With its vast amount of libraries and huge open-source community providing useful functionality ROS facilitates the development of robotic applications without having to reimplement standardized technology.[1]

7.1 Conceptual Overview

The Robot Operating System can be divided into three conceptual levels each contributing a integral part to the utility of ROS. These different levels are described in the following sections.[2]

File System

The File System Level mainly provides constraints and best practices for creating and structuring packages and their components. ROS provides appropriate tools to facilitate file-system operations with and within packages.

Computational Graph

The Computational Graph provides crucial functionality to ROS as it refers to the peer-to-peer mesh network of processes (nodes) each providing data to be utilized within the graph by publishing and subscribing to topics. The concepts and technologies powering the computational graph are described in later in this chapter.

[1] **openSourceRoboticsFoundationDefinitionNodate**.
[2] **openSourceRoboticsFoundationConceptsNodate**.

Community

The Community preserves the usability of ROS as new and useful packages and tools are created as well as existing functionality is being maintained.

7.2 Naming

To aid the organization of programs, processes as well as resources ROS provides two naming schemes that are described in the following sections.[3]

Graph Resource Names

Graph Resource Names utilize a hierarchical structure to organize nodes, services, topics or anything else within the computational graph. ROS defines four different types of names:

Package Resource Names

Package Resource Names aim to facilitate the search process of resources at File System Level. These names usually consist of the packages name as well as the path to the desired resource within the package.

7.3 Packages

Software in ROS is organized in packages containing nodes, libraries or any other piece of software providing functionality.
Since packages are the atomic unit of build and release they aim to be a slim as possible my implementing only a limited set of features. In other words packages should be implemented to provide minimal usability without being too large-scaled.[4] This means that each package is developed to work together with other packages to deliver utility as a connected system.
At file-system level packages simply refer to directories. While most subfolders and files within a package depend on its purpose, every package has to contain a package.xml and a CMakeLists.txt providing meta and build information. Packages can be build by utilizing rosbuild or catkin.[5]

Metapackages

Metapackages are specialized packages only containing a package.xml that logically links multiple related packages.[6] They can be used to conveniently install a group of packages simultaneously.

[3] **openSourceRoboticsFoundationConceptsNodate**.
[4] **openSourceRoboticsFoundationPackageNodate**.
[5] **openSourceRoboticsFoundationBuildNodate**.
[6] **openSourceRoboticsFoundationMetapackageNodate**.

7.4 Nodes

The goal of ROS is to promote code reusability and decoupling of functionality to aid the versatility and usability of the system. Following this guideline every robotic system utilizing ROS consists of a fine-grained graph of processes called nodes. Each node provides computation on a single feature utilizing a ROS client library to communicate with others over a mesh-like peer-to-peer network.[7]
Exemplary for such as system would be one node running a LiDAR sensor, one responsible for localization, one performing motion planning, one controlling motor drivers and motors as well as one node running the robots main control loop.
This architecture allows for a much more fault safe and less complex applications in comparison to monolithic systems.[8] This means that development and debugging are facilitated since errors can be contained within a singular slim node rather than a larger program.
Each node has a node type consisting of the package name it is located and as well as the nodes executable.

7.5 Communication

7.5.1 Messages

Messages are the medium of communication used in topics or services to transport data between nodes.

Message Description

A message is a simple data structure consisting of multiple type fields. These fields can be primitives, arrays, custom types as well as other message types.[9]
The message description language can be used to structure custom messages in *.msg* files contained in the *msg* directory of a package.

Message Types

Message types refer to package resource names consisting of the packages name as well as the name of the messages *.msg* file.

7.5.2 Topics

The core component of communication in ROS are topics. They are unidirectional message streams enabling data transmission by utilizing the publisher-subscriber model Furthermore the decoupling of functionality is facilitated by anonymously connecting nodes as producer and consumer of data. This means neither publisher nor subscriber of the topic need to know each other. While ROS does not limit the amount of publishers and subscribers connected

[7] **openSourceRoboticsFoundationNodesNodate**.
[8] **stephensBeginning2015**.
[9] **openSourceRoboticsFoundationMessagesNodate**.

to a topic, it strictly enforces the usage of the exact message type specified for the topics communication to work properly.

7.5.3 Services

The communication architecture in ROS utilizing the publisher-subscriber model is advantageous in most use-cases, however most distributed systems require remote procedure calls (RPC) which are not supported by default.
With RPCs a client sends a request to a server specifying the procedure to be called and its parameters. While the server executes the procedure the client awaits a reply. Once the procedures results are computed and sent to the client its workflow can be resumed.[10]
Services enable communication over RPC by defining a pair of messages, one for requests and one for replies. Such service can then be attached to a node and called by a client using the service name.[11]

7.6 Master

One of the most important components of ROS is the Master. It tracks publishers and subscribers of topics as well as services and provides registration as well as name resolution to nodes. This means whenever a node wants to publish or subscribe to a specific topic or service it contacts the master first using XML-RPC. When a topic has at leat one subscriber and publisher the Master negotiates between the nodes so a peer-to-peer connection can be established using a Slave API provided by the nodes XML-RPC Server.[12] A simplified version of this procedure can be seen in Fig. 7.1
Besides registration and name resolution, the ROS Master also provides a Parameter Server used for globally storing static system parameters.[13]

7.7 Transform Library

A complex robotic system consists of multiple parts such as sensors, cameras, manipulators, etc. which are each represented as a coordinate frame, where each frame is connected to another frame using joints. When trying to move a specific part or coordinate frame, not only the transform of that single frame but the composite transform of each frame in relation to the target has to be calculated. This is especially important when moving a robotic arm based on sensor readings. In this example a transform between the position of the sensor and the arm needs to be calculated so the motion performed by the arm the motion perceived by the sensor match. Using the ROS Transform Library (tf) these complex calculations can be facilitated. To this end tf keeps track of each coordinate frame in a acyclic relationship tree where tf broadcasters then publish relative pose information and listeners query transforms between two coordinate frames. Because not all pose information in a robotic system is

[10] **rfc1831**.
[11] **openSourceRoboticsFoundationServicesNodate**.
[12] **openSourceRoboticsFoundationMasterNodate**.
[13] **openSourceRoboticsFoundationParameterServerNodate**.

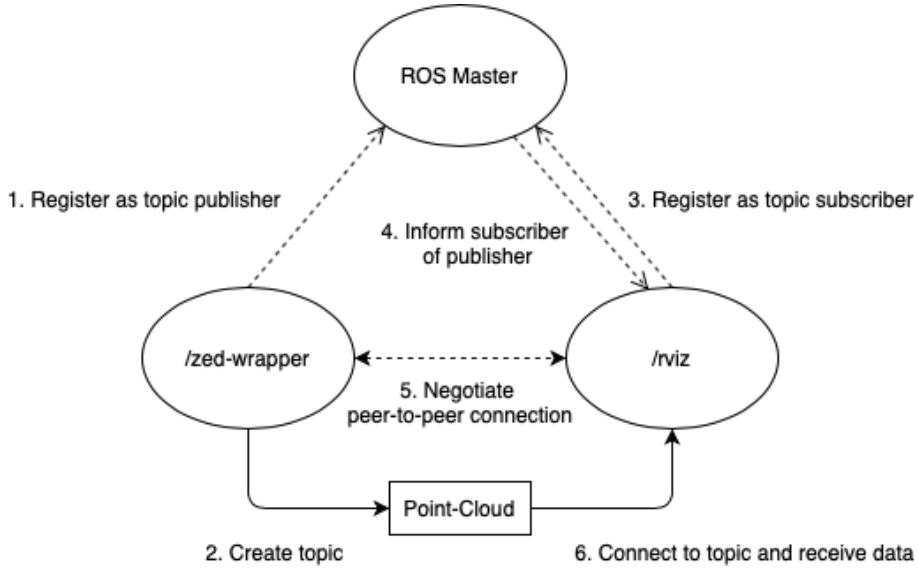


Figure 7.1: Diagram of the topic registration process in ROS at which the publishing node registers its topic before the subscriber tells the master its interest in the point-cloud topic. However a node can be registered as a subscriber to a specific topic without the topic existing yet.

instantly accessible, the tf saves this information for each frame over time. This means that transforms can be queried not just spacially but also temporally.

7.8 Simulation

Debugging and testing robot applications can be a repetitive and tedious task, especially when a test environment needs to be reset at every test cycle. In order to facilitate this part of development the Autumn robot utilises the representation and simulation technologies that are described in the following sections.

7.8.1 URDF

In order to perform simulations or compute coordinate frame transforms a robot needs to be described in some way. One of the more popular description formats is the Unified Robot Description Format (URDF) which provides a XML format for representing a robot and its components as well as a *c++* parser and tools to convert and verify and visualize these models.[14] The *check_urdf* tool parses a URDF-File and returns the robots kinematic chain if successful. To visualize the robots frames and joints in a graphviz[15] tree the *urdf_to_graphiz* tool can be used. The resulting tree corresponds to the relationship tree tf uses to calculate transforms.

[14] **openSourceRoboticsFoundationURDFNodate**.
[15] **graphvizAuthorsAboutNodate**.

7.8.2 Gazebo Simulator

Gazebo is a 3D physics simulator often used in close relation to ROS projects. Therefore it provides tooling for model and world design and generation as well as comprehensive interfaces for controlling a simulated robot through ROS. Further advantages of Gazebo are its accurate sensor and sensor noise generation as well as its large community providing countless models of robots and sensors.

Chapter 8

Path Planning

Author: Fabian Kleinrad
A crucial part of autonomy in robotics are the means for planning ahead movements in a cooperative manner with the environment. Means to accomplish this are so-called path planning algorithms. This chapter is going to focus on exploring different kinds of approaches to path planning and evaluate which approach is most fitting to be used in a real-time, high-dimensional use case present in the autumn project.

8.1 Types of Path Planning

The problem of finding an optimal path between two points is an old one. The first proposed solution was the Dijkstra's algorithm. However, with steadily evolving computer science the challenges to be mastered by such algorithms got harder and harder. That's the reason why over the last years the simple principle of the Dijkstra algorithm has branched out specializing and excelling in certain real-world applications.[1]

8.1.1 Sampling-based Algoritms

In motion planning, sampling-based algorithms can be differentiated to other kinds of approaches, by the way, they explore their environment. sampling-based algorithms such as the probabilistic road map algorithm or the rapidly exploring random tree use a random point in their reference space and expand in that direction. This random point is considered a sample.

8.1.2 Multiple-Query and Single-Query

The term multiple-query refers, in connection with path planning Algorithms, to the feasibility of deriving a variety of different paths, without the need of rerunning the algorithm. In contrast, single-query algorithms are only able to compute one path at a time.
Use cases for Multiple-Query Algorithms would be unchanging environments. The reason for that, by generating an extensive grid of connections to be able to calculate a multitude of

[1] **Pan2020**.

different start/goal combinations more computational time is needed.
Single-Query approaches focus on performance instead of reuse-ability, which makes them ideal for dynamic domains.[2][3]

8.2 PRM

Probabilistic RoadMap is a path planning algorithm tailored to multi-query applications. It is considered one of the most influential sampling-based path planning algorithms.
The algorithm can be broken down into two phases. The first phase, which is referred to as the pre-processing phase, starts with an empty graph. At first, it samples n random points and adds them to a set of vertices if they are located in space free from obstacles. After constructing a set of n vertices, it attempts connections between a random vertex and its neighboring nodes in a predefined radius. This connecting of to vertices is realized with a simple straight-line connection. All collision-free connections between vertices and their respective neighbors are added to a set of edges. The result of this pre-processing phase is a roadmap, with the number of sampled points determining the quality of to be calculated paths.[4]
Upon finishing the initial construction phase of a roadmap, like the one depicted in figure 8.1, start/goal combinations can be processed. The actual path finding in the generated graph is handled by other non-sampling-based path finding methods such as A*.
With the PRM focusing on a multi-query approach it is possible to calculate an arbitrary number of different paths without the need to construct a new graph.

8.3 RRT Algorithm

Rapidly exploring random tree is a path planning algorithm, that can be categorized as sampling-based and single-query. In the field of sampling-based path-finding algorithms is it considered together with the PRM the most influential. RRT works by constructing a tree of possible trajectories. Therefore it is first required to define an initial vertex, much like a root. After each following iteration, a random sample is taken from the space that is considered free from obstacles. After generating a random vertex, the nearest neighboring point in the tree is searched for. The closest node is then used as a pivot point and a new vertex is constructed a predefined distance away from the nearest node in the direction of the random sample. Thereafter it is attempted to connect the new vertex with the nearest. If this straight-line connection can exit without colliding with obstacles it is added to the set of edges. This procedure is depicted in figure 8.2. The algorithm ends when a new node is within the predefined distance away from the goal point.[5]
The name rapidly exploring random tree stems from the tree-like structure constructed, like the one you can see in figure 8.2 when exploring spaces This approach to path planning makes it possible to explore rapidly changing environments efficiently.

[2] **Bekris2003**.
[3] **stackexchangeMultiSingleQuery2019**.
[4] **Karaman2011**.
[5] **Karaman2011**.

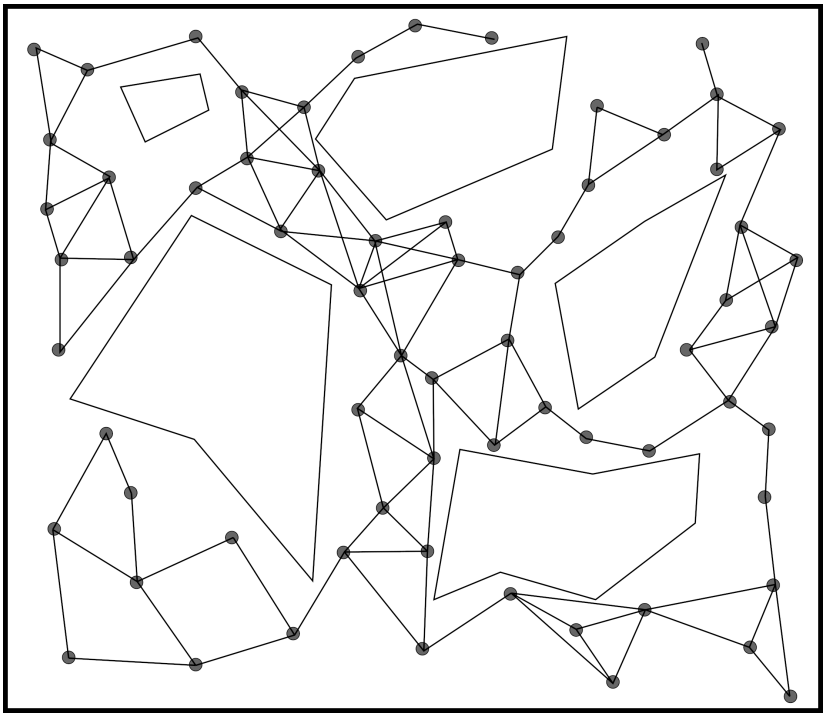


Figure 8.1: Example of a roadmap in two dimensional space constructed by the PRM algorithm.

8.4 A* Algorithm

The A* algorithm is a simplistic and reliable method compared to other path-planning techniques. Therefore it is nowadays prevalently used in a variety of different applications.[8]
It combines heuristic properties with formal virtues of Dijkstra's algorithm. Whereas a purely heuristic approach would not guarantee that a path can be found even if a possible solution existed, the A* guarantees that the shortest path is found.[9] A* works by trying all possible combinations like the Dijkstra algorithm in a grid or graph-based environment. With A* being based on the Dijkstra algorithm it favors nodes that are closer to the current position. Additionally, it follows heuristic principles by putting a higher priority on nodes closer to the goal. An example of what an A* search algorithm looks like in a grid-based environment is depicted in figure. 8.3.[10]
With these properties, it can be considered an informed Dijkstra search.

[8] **Zammit2018**.
[9] **Sathyaraj2008**.
[10] **standfordAStarComparison1997**.

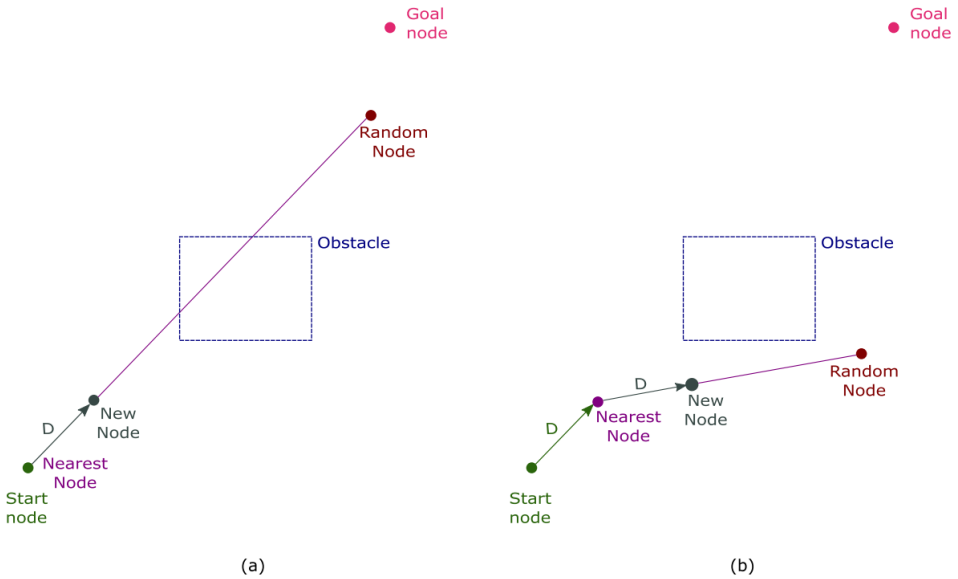


Figure 8.2: Visualization of the first two iterations (a), (b) when an rrt algorithm is exploring a simple 2d space wihttps://robotics.stackexchange.com/questions/18433/difference-between-single-query-and-multiple-query-algorithmsth an obstacle present. D referring to the predefined distance between vertices.[7]

8.5 Evaluation

This section is going to cover the reasons behind the selection of the path-planning algorithm used in autumn. Therefor we are going to theoretically evaluate the performance of each algorithm in relation to the environment provided by the project. Additionally, we are going to focus on properties essential for accomplishing the goal set in the autumn project.

8.5.1 Autumn Use-case

There are several criteria to be fulfilled, in order to be considered a working solution for the challenge of path planning faced by Autumn. As a means to use one of the aforementioned algorithms, they have to be suitable for a real-time, 3-dimensional application, like the one present in the project. When working with UAVs it is critical that an optimal path is calculated reliable and in the shortest amount of time possible.

8.5.2 Comparison

In the following comparison, we are going to look at the performance of the three different algorithms already mentioned in this chapter. Knowing that the PRM algorithm doesn't find a path on its own, it is implied from here on out that the PRM works in combination with the A* algorithm and the A* on its own is only applied with grid-based environments.

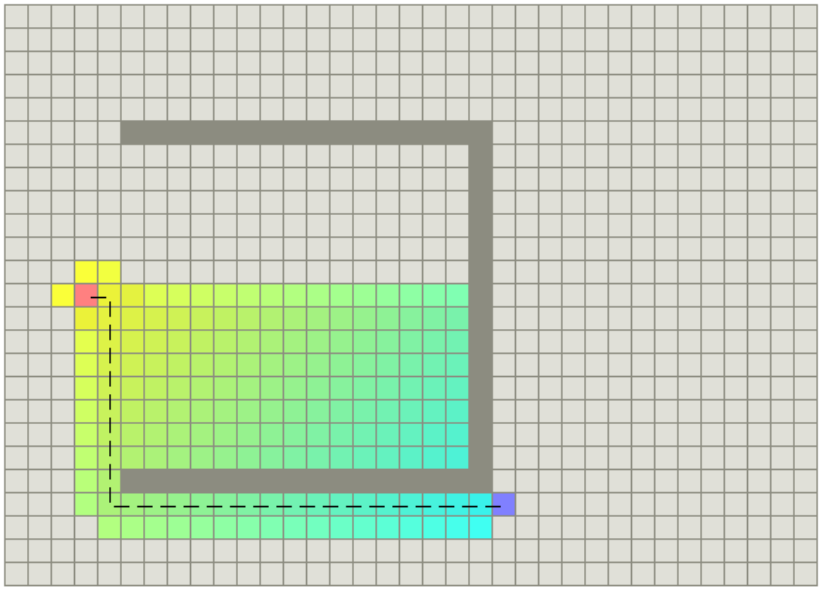


Figure 8.3: A* path finding around a concave obstacle in a grid-based environment.[11]

Computational time requirement

An important requirement to be fulfilled by the path planning algorithm of the Autumn project is to require as little computational time as possible. Taking a look at the PRM algorithm, with its principle of reusing the generated graph, as a road-map, resulting in a short runtime of the algorithm, when looking at the computation of multiple paths. The best-case scenario for the PRM in this configuration would be, an A* algorithm running in a graph-based environment, with the PRM only having to do generate the graph and improve it progressively. A problem that arises is that this best-case scenario is impossible to happen in the Autumn project. Starting uneducated about the environment, like in Autumn, is the worst case for the PRM. At first, the PRM algorithm would generate a road-map in an environment where most of the obstacles are still unknown. Thereafter when the drone gets its first path to its destination it detects obstacles and expands the known space for the algorithm. With the environment changing as much as in Autumn it would be far too expensive to rewire the existing road-map around new obstacles. A* is the best-case scenario in a graph-based environment for the PRM, but what about running the A* standalone now in a grid-based environment. Grid-based environments tend to be very resource-intensive, with full coverage of the entire space. To preclude the possibility of large portions of the search being dissipated, the A* is biased towards the goal. When comparing the A* to the PRM in terms of computational speed in Autumn, A* would be superior, because of the substantial amount of time the PRM takes to generate a road-map only to discard it a few iterations later. That being said, the A* still needs to cover a vast grid, depending on the granularity of the grid. Know knowing that both the A* and PRM algorithms aren't the most efficient in the matter of short runtime, let's cover the RRT algorithm. With RRT following a graph-based and sampling-based approach like the PRM, it covers the space to

be explored within a short time frame. The RRT, in contrast to the PRM, doesn't take time to construct the graph and then start to find a path. RRT searches for a path while constructing the tree-like structure. This results in the fastest computational time of all other mentioned algorithms when looking at one-time path planning. With the reference space of the algorithm in Autumn being drastically and continuously changing, it resembles a one-time use path perfectly fitted to be determined by a single-query path planning algorithm like the RRT.

Scalability for high-dimensional spaces

Thus far the application of proposed algorithms happened in a two-dimensional environment. When working with UAVs, there is the added challenge of a third dimension to be covered, in contrast to using ground-based vehicles. This added dimension imposes new challenges that the whole system has to handle, with the means to plan a path being one of the most affected. Additional time complexity being the most crucial change when working in a higher dimension, it is essential to be as efficient as possible. The last point covered the time required for each algorithm. With most path-planning algorithms having a time complexity of $O\left(b^d\right)$[12]. With A* being one of the most time-intensive. In this equation, the variable b is the branching factor and describes the number of descendants of a parent node. d refers to the depth of the goal node. The worst-case scenario would be $O\left(|V|\right)$, meaning that every vertex of the reference space has to be checked. Looking at the problem of three dimensions where the added dimension increases the number of vertices exponentially. A* being a simple approach with brute force characteristics isn't suitable for application in high dimensional spaces.
Comparing the RRT to the A* highlights the benefits of a sampling-based approach. While the A* looks at direct neighbours to continue the search, RRT has a step range as a predefined constant. Therefore the RRT explores with a faster pace due to a rougher coverage of the space. This means in relation to higher dimensional spaces it converges more efficiently than the A*.

8.5.3 Collusion

By evaluating the potential algorithms to be used in the autumn project, the RRT came out on top. With each algorithm providing unique features best fitted for the use-case they are designed for, the problems the RRT algorithm tries to solve are the same encountered in autumn. Properties like space complexity and path quality were neglected in this comparison, due to the low importance of the project. By shifting the computation away from the drone the importance of minimum time far outweighs the required space. Furthermore, the need for an optimal path isn't required on account of the goal being to cover the whole space eventually. That being said a big drawback of the base RRT algorithm is the incapability to provide the possibility to improve on an existing path by rewiring the graph. For that, we are going to take a look at a variant of RRT the RRT* that solves this exact problem.

[12] **stackexchangeAstarTimeComplexity2019**.

8.6 RRT* Algorithm

This section is going to cover the improved version of the RRT algorithm, mentioned above. The RRT* algorithm is being used in the Autumn project as the path planning algorithm. The rationale for this decision is covered in the evaluation section.

8.6.1 Concept behind RRT*

A problem that arises when using the RRT algorithm is that finding an optimal path is impossible due to the lack of rewiring options. Therefore the RRT* algorithm was invented. The improved algorithm works differently when connecting a newly sampled node into the existing tree structure, which in the RRT is done by connecting the new node to its nearest neighbour. In the case of the RRT*, it calculates the cost of other possible connections stemming from the root node. Hereby a set of nodes closest to the newly sampled vertex are used as possible connection candidates. By rewiring the tree after each new node is sampled it makes it possible for a near-optimal path to be found.

8.6.2 Algorithm

The RRT* works at first the same as the RRT algorithm, it first defines a set of vertices, V with the start node and an empty set of Edges, E. In contrast to the RRT algorithm is the runtime of the RRT* algorithm is not dependent on how fast the goal node is reached, but instead on the degree of optimization for the calculated path. Line four to six in Algorithm 1, are the same as the base algorithm. Hereby is a random node sampled, the nearest node in relation to the random node is calculated and a new node generated. After checking if a straight-line connection is possible between $x_{nearest}$ and x_{new}, a set of neighbouring vertices, X_{near} is initialized. X_{near} consist of nodes in the tree with a distance D, in relation to x_{new}. Thereafter x_{new} gets added to the V. In lines 9 to 13, the cost of paths connecting the nodes of X_{near} with x_{new} get calculated and an edge between the node with the cheapest path, x_{min} and x_{new} gets added to E. The final stage of the RRT* is to rewire all neighbouring nodes in X_{near}, if the path connecting to x_{new} is cheaper than the original path to the root

vertex.

Algorithm 1: RRT* 2011[a]

```
1  V ← {x_init};
2  E ← 0;
3  for i ← 1 to n do
4      x_rand ← SampleFree();
5      x_nearest ← Nearest(G = (V, E), x_rand);
6      x_new ← Steer(x_nearest, x_rand, D);
7      if ObstacleFree (x_nearest, x_new) then
8          X_near ← Near(G = (V, E), x_new, D);
9          V ← V ∪ {x_new};
10         x_min ← x_nearest;
11         c_min ← Cost(x_nearest) + c(Line(x_nearest, x_new));
12         foreach x_near ∈ X_near do
13             if CollisionFree (x_near, x_new) ∧ Cost (x_near) +
                c(Line(x_near, x_new)) < c_min then
14                 x_min ← x_near;
15                 c_min ← Cost(x_near) + c(Line(x_near, x_new));
16             end
17         end
18         E ← E ∪ {(x_min, x_new)};
19         foreach x_near ∈ X_near do
20             if CollisionFree (x_near, x_new) ∧ Cost (x_new) +
                c(Line(x_near, x_new)) < c_near then
21                 x_parent ← Parent(x_near);
22             end
23             E ← (E\{(x_parent, x_near)} ∪ {(x_new, x_near)});
24         end
25     end
26 end
27 return G = (V, E);
```

[a] **Karaman2011**.

8.7 RRT* Variants

8.7.1 RT-RRT*

The RT-RRT* takes focus on real-time path planning. RT-RRT* makes it possible to reposition the root node. Using this online tree rewiring strategy makes it possible to keep the previously sampled tree. The two core principles introduced with the RT-RRT*, are tree expansion and tree rewiring. At first, the Algorithm starts by initializing a root node. After each following iterations, the tree expands and rewires. This sampling lasts for a user-defined

time and is followed by planning a path from the root. The length of this path is controlled by the user, by defining the number of steps the path consists of. In this phase, the agent is moved gradually towards the tree root. Thereafter, when path planning is finished and the agent is located at the position of the tree root, the position of the root is changed to the next node in the generated path. This approach enables the agent to move without the time needed to compute the whole path. Expansion in the RT-RRT* is done similar to the base algorithm, by sampling a random node, finding its neighbor with the minimum cost-to-reach and connecting them. The rewiring is done in the default scenario done because newly added nodes have a smaller cost-to-reach than the parent of a certain node. In the second case, rewiring is needed if the environment changes. Therefore a larger portion of the tree has to change. Hereby two different modes are utilized. The first one consists of rewiring the tree in a circle centered around the tree node. For the second option, both focused and uniform sampling is used, with the difference that it is not done for one node, but instead concentrates on patches.[13]

RT-RRT* in Autumn

When applying the concept of the RT-RRT* to Autumn it would seem a good idea. Taking a closer look uncovers that the proposed RT-RRT* only works efficiently in environments with limited changes. Rewiring the tree, done in the case of a dynamic obstacle changing, doesn't make sense when the obstacle referred to covers a large portion of the space. This being the case in Autumn, because contrary to the examples in the paper covering the RT-RRT* is the algorithm not educated about the environment. Only while moving are obstacles and boundaries discovered In general the RT-RRT* is using concepts of road-maps like the one generated in the PRM algorithm and applying it to the RRT* algorithm.

8.7.2 Smart-RRT*

Smart-RRT* focuses on solving the problem imposed when using the RRT* algorithm. With the slow convergence rate of the RRT* towards an ideal path, it takes up to an infinite amount of time to reach the most optimal path. Therefore Smart-RRT* utilizes two techniques, path optimization and intelligent sampling, in an effort of accelerating the rate of convergence. It works the same as the base algorithm till the first path is found. From there on out it applies path optimization and intelligent sampling. Path optimization works by interconnecting nodes in the found path that are directly visible. This leads based on triangular inequality to a shorter path. The reconnected nodes are then used as biased points for intelligent sampling. Intelligent sampling samples new points based on a bias generated from finding a shorter path. The result is a greater node density in the area of critical points on the path, mostly located around obstacles.[14]

[13] **Naderi2015**.
[14] **Islam2012**.

Smart-RRT* in Autumn

Using Smart-RRT* improves the rate of convergence and time to reach an optimal path drastically. This makes one wonder why not use this improved version of the implemented algorithm in the project. When looking at this it is important to keep the goal in mind the autumn project aims to reach. Which is, to capture an environment in form of a 3-dimensional model. Therefore it is unnecessary to compute an optimal path. The whole space is going to be covered eventually which would only be a waste of computational power to invest in additional path optimization.

Chapter 9

Methodology

Author:

Chapter 10

Implementation

Author:

Chapter 11

Experiment 1

Author:

Chapter 12

Lessons learned

Author:

Chapter 13

Experiment 2

Author:

Chapter 14

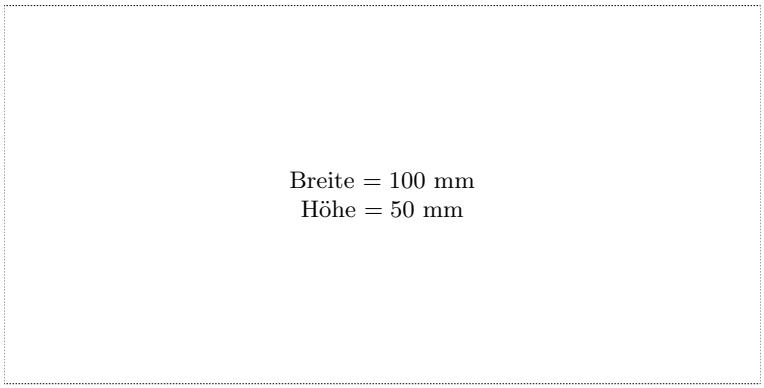
Conclusion

Author:

Index

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —