CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

BACHELOR'S THESIS

Michal Werner

Control System for a Tracked Mobile Robot

Department of Cybernetics

Thesis supervisor: **Ing. Tomáš Báča**

MAY 2020

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Werner Michal**	Personal ID number:	**474638**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Cybernetics and Robotics**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Control System for a Tracked Mobile Robot

Bachelor’s thesis title in Czech:

Realizace autonomního systému pro pásový mobilní robot

Guidelines:

The student will familiarize himself with an existing platform of a small, tracked mobile robot. He will then equip the robot with an onboard computer and sensors to allow autonomous motion. One of the sensors will be a lidar rangefinder, which will allow the use of an algorithm for simultaneous localization and mapping (SLAM). The student will then study existing solutions of SLAM and will choose one to be applied to the robot. Finally, the student will implement routines for control and planning, which will allow the robot to drive to a chosen spot in a map generated by the onboard SLAM. The thesis will consist of the following milestones:

- Study the Robot Operating System (ROS) and prepare an onboard computer for its installation on the provided robotic platform.
- Study the SLAM algorithm and choose an existing solution, which will be used on the robot. Then install it on the platform.
- Equip the robot with necessary sensors and make them available within ROS.
- Implement hardware and software interface for controlling the drives of the robot.
- Implement a planning and control routine, which will allow autonomous travel using the data provided by the onboard SLAM algorithm.

Bibliography / sources:

[1] Kohlbrecher, Stefan, et al. "A flexible and scalable slam system with full 3d motion estimation." 2011 IEEE International Symposium on Safety, Security, and Rescue Robotics. IEEE, 2011.
[2] Santos, Joao Machado, David Portugal, and Rui P. Rocha. "An evaluation of 2D SLAM techniques available in robot operating system." 2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR). IEEE, 2013.
[3] Quigley, Morgan, et al. "ROS: an open-source Robot Operating System." ICRA workshop on open source software. Vol. 3. No. 3.2. 2009.
[4] Dissanayake, MWM Gamini, et al. "A solution to the simultaneous localization and map building (SLAM) problem." IEEE Transactions on robotics and automation 17.3 (2001): 229-241.

Name and workplace of bachelor’s thesis supervisor:

Ing. Tomáš Báča, Multi-robot Systems, FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **10.01.2020** Deadline for bachelor thesis submission: **22.05.2020**

Assignment valid until: **30.09.2021**

Ing. Tomáš Báča	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Author statement

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date ....

Acknowledgements

I would like to thank my supervisor Tomáš Báča for all his advice and great support during this project. Many thanks also to my family and friends who supported me during my whole studies.

Abstract

This thesis deals with development of a control system for a tracked mobile robot. Firstly, the task of Simultaneous Localization And Mapping (SLAM) is briefly introduced and main SLAM approaches are described. As the main part of the thesis, the control system is designed and implemented in *Robot Operating System* (ROS). It consists of path planning, motion control and exploration. As a result, the developed robotic system is able to semi-autonomously explore and map the unknown environment. Finally, the presented solution was tested in real world experiments.

Keywords: simultaneous localization and mapping, path planning, control, exploration, unmanned ground vehicle, ROS

Abstrakt

Tato práce se zabývá návrhem řídícího systému pro pásový robot. Nejprve stručně představuje úlohu Simultánní Lokalizace A Mapování (SLAM) a popisuje hlavní přístupy k jejímu řešení. Hlavní částí této práce je pak návrh řídícího systému vytvořeného v prostředí ROS. Jeho součástí je plánování trasy, řízení pohybu a prohledávání prostoru. Výsledný systém je schopen prohledávat a mapovat neznámé prostředí v režimu částečné autonomie. Nakonec bylo prezentované řešení otestováno v reálném prostředí.

Klíčová slova: simultánní lokalizace a mapování, plánování trasy, řízení, explorace, bezosádkové pozemní vozidlo, ROS

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Autonomous robot systems have undergone fast development in recent decades. Platforms are being developed to be able to undertake assigned tasks without direct external commands. This approach is present in many fields of human work, such as automotive industry, agriculture, aircraft, military, rescue operations and many others. In general, the importance of autonomous robotics is growing in time.

One of the main categories of autonomous systems is unmanned ground vehicles (UGV). They can be used for the situations when a human presence is not desirable, such as hazardous environments, rescue operations indoors and underground and many others. These environments are often unknown or changed. For this reason, exploration and mapping can be one of the tasks for autonomous vehicles.

The prerequisite for autonomous vehicles is the ability to perceive the outside world. This sensor information must be processed so that the robots are able to move in the environment, avoid obstacles and fulfill assigned tasks. This addresses the problem of Simultaneous Localization And Mapping (SLAM). It is one of the most widely researched topics in robotics in past decades. The idea is to place the robot equipped with sensors into an unknown environment. As it moves and observes the surroundings, it incrementally builds the map of the environment and simultaneously estimates its position.

This thesis focuses on the SLAM problem from the theoretical and practical point of view. The problem definition and overview of main SLAM approaches is given in the chapter 2. As the main part, the control system for partly autonomous unmanned ground vehicle is developed. The control algorithm is implemented in Robot Operation System (ROS) and consists of chosen SLAM algorithm, map preprocessing and path planning. Together with hardware interface, this control system is designed for an exploration of indoor environments. Description of the used hardware platform, sensors, onboard computer and other hardware and software is given in the chapter 3. Used solution is described in the chapter 4. Finally, the developed platform is tested, and these experimental results are presented in the chapter 5.

1.1 State of the art

The topic of autonomous ground vehicles used for exploration is extensive and was widely researched in past decades. It consists of all the subtopics such as path planning, control, sensors, localization and mapping, hardware design and others. These tasks were solved in different ways, and detailed overview of each subtopic is beyond the scope of this thesis. There are many different types of platforms (for example: four-legged robots [4], wheeled, tracked and crawling platforms [5]) used for exploration, and each of them requires different path planning and control approach, depending on its hardware design and given environment and application.

Open research area is the problem of Simultaneous Localization And Mapping in real-world environments. A brief overview of mainly used sensors and state-of-the-art SLAM methods is given in the next chapter. Contemporary research is focusing on the multi-robot systems and cooperation between ground and aerial unmanned vehicles [6], [7]. To give an example of such research, the paper [5] describes the autonomous exploration in the subterranean environment. The research team from FEE CTU [1] in Prague used wheeled, tracked and crawling unmanned ground vehicles together with drones. Each of the platforms is carrying different type of sensors (2D or 3D high-range lidars, cameras). The robots are sharing information with each other and are able to autonomously build the 3D spatial map of the environment and localize some objects of interest without the need of GPS signal.

In general, the aim of this work is not to push the boundaries of human knowledge, but to demonstrate the basic principles and develop a control system for the specific robotic platform.

[1]Faculty of Electrical Engineering, Czech Technical University

Chapter 2

SLAM

Simultaneous Localization And Mapping (SLAM) is considered to be a fundamental problem for truly autonomous robots. As mentioned in [8], the popularity of the SLAM problem is connected with the emergence of indoor applications of mobile robotics. In these environments, typical localization methods (GPS) are not available. The localization systems based only on the robot's odometry are not accurate enough and the accumulated error diverges.

The SLAM is sometimes referred as the "chicken-and-egg" problem. The mapping could be done easily if the exact position of the robot is known. Also, the localization in the known environment is easy to solve. But both processes are taken simultaneously and depend on each other, which makes this problem challenging.

2.1 SLAM problem definition

SLAM problem has been formulated and solved as a theoretical problem in a number of different ways. Following definition is based on the broadly used probabilistic approach described in [9] or [10]. Assume a robot equipped with sensors in the unknown environment. The robot position is described as x_t at the given time t. The path of the robot is then:

$$X_{0:t} = \{x_0, x_1, ..., x_t\}, \tag{2.1}$$

where x_0 is the known starting position. The map of the environment m describes the locations of all landmarks and obstacles. The environment is mostly assumed as static (not changing in time).

The robot moves according to the control inputs u_t. These controls can be sensed by an odometry (for example wheel encoders), which characterized the motion of the robot between the time t and $t-1$. The sequence

$$U_{1:t} = \{u_1, u_2, ..., u_t\} \tag{2.2}$$

describes the control inputs or the data from motor encoders from start to time t. As the robot moves through this environment, it measures the relative positions of obstacles around (landmarks). There are many different types of sensors, such as laser scanners, cameras, sonars and others). These observations are described as

$$Z_{1:t} = \{z_1, z_2, ..., z_t\}, \tag{2.3}$$

where z_t is the measurement done at time t. In the real world applications, the odometry data are not perfectly accurate to reconstruct the robot motion precisely. The uncertainty is growing with the motion. Also the observations are affected by noise and inaccuracy of the sensors. Because of this, the goal is to derive the “most probable” position of the robot and landmarks in the environment. In that point of view, we are searching for probability distribution

$$p(x_t, m|Z_{1:t}, U_{1:t}). \tag{2.4}$$

The equation 2.4 defines the whole SLAM problem. The goal is to estimate the “most probable” robot position x_t and map state m, depending on the given measurements and odometry data (both affected by some error) recorded until time t.

The structure of the SLAM problem and involved variables are described in the figure 2.1. The map m and positions x_i (white) are unknown and should be estimated, the observations z_i and odometry u_i (brown) are directly observable to the robot. The arrows represent the dependency.

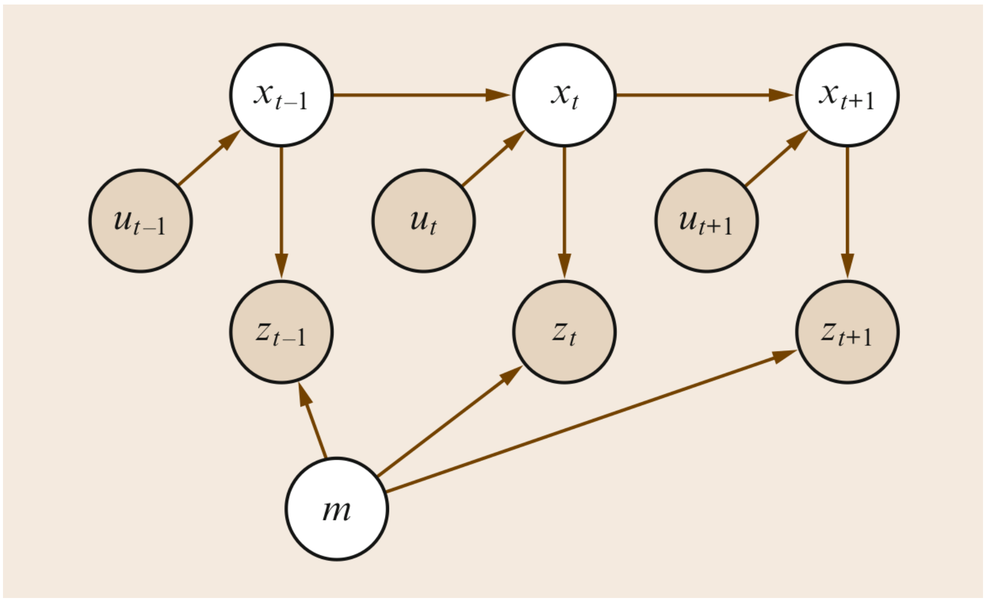


Figure 2.1: Graph visualisation of the SLAM problem. Source: [1]

2.2 Types of SLAM

The equation 2.4 refers to the *Online SLAM* problem. This approach is recovering only the most recent pose of the robot. This is used mostly in applications, where the robot moves autonomously and must interact with the environment. On the other hand, we can also define *Full SLAM* as probabilistic distribution

$$p(X_{0:t}, m | Z_{1:t}, U_{1:t}). \tag{2.5}$$

In that case, we are estimating the whole path of the robot, not only the recent position. The data are often processed at one time. This process requires more computing power and is not suitable for autonomous control and planning in real-time [11].

There is no single best solution to the SLAM problem. The chosen solution for a given application will depend on many factors, such as type of used sensor, desired map resolution, the nature of the environment and others [1].

By the years, many different solutions addressing the SLAM problem were developed. One significant approach is based on probabilistic theory, more precisely on Extended Kalman filters and Particle filters. Another important group of algorithms belongs to the category of graph-based methods. These three approaches are referred as main paradigms of SLAM problem ([1]) and are briefly described in the following sections and some remarkable algorithms are mentioned.

2.3 Filter based methods

One of the classical approaches is based on theory of Bayes filters. These techniques provide a recursive solution using motion and observation model of the system. Then the observation model is formulated as

$$p(z_t | x_t, m), \tag{2.6}$$

and describes the probability of measuring z_t, when the position of the robot and landmarks are given. The motion model describes the robot state x_t at time t based on the given robot state x_{t-1} and control input u_t:

$$p(x_t | x_{t-1}, u_t). \tag{2.7}$$

We assume that the next state of the robot depends only on the previous state and the control input and is independent of both the observations and the map.

2.3.1 Extended Kalman filters

This method described in [10] is based on one of the most popular implementation of Bayes filters: Kalman filters. The algorithm itself consists of two steps: prediction and correction.

In the prediction step, the next state of the system (position of the robot and landmarks) is predicted. This is done based on the previous estimate of the state of the system and motion model. After that, the expected "observation data" are calculated by the observation model (what sensors expect to measure in the predicted state). Both these predictions and real measurement obtained by the sensor contain some level of uncertainty. As the result, the prediction and measured data are combined and weighted mean is calculated. This result is taken as new estimate of the system. Then the prediction and correction is repeated in next iteration.

The main idea of this filter based approach is to reduce the Gaussian noise in the motion and measurement combining these two inputs (prediction and correction) together. For example, if the precision of the measurement is poor, the Kalman filter estimate depends more on the motion prediction (odometry). On the other hand: if the measurement precision is high, the estimate is based more on the actual sensor data.

The key assumption of this algorithm is that the uncertainty present in the system (noise in the measurement and motion of the robot) is Gaussian. This approach requires the linear motion model of the system, hence called Linear Kalman Filter (LKF). The possible nonlinearity is solved using Extended Kalman filter, which takes the local linearization using Taylor expansion.

The implementation of the EKF SLAM algorithm requires the covariance matrix representing all correlations between landmarks and robot position, which is the key limiting factor, as stated in [1]. The algorithm complexity (in the naive implementation) increases quadratically with the number of landmarks. Because of that, this approach is not suitable for larger environments, as written in [9].

2.3.2 Particle filters

Another probabilistic approach based on the different type of Bayes filters are particle filters. The idea of particle filters is following. The probability density function (estimating, for example, the robot position) is represented with weighted discrete samples called particles. As the first step, the particles are sampled from proposal (at the start of the algorithm, the particles are distributed equally, because each position has the same probability). As the data from sensors arrive, the algorithm increases the weights of particles representing the probable robot positions (based on the observation and motion model). In other words, the particles on the poses that fit the measurements and odometry represent more probably the real robot position. After that, the discrete distribution is resampled and least-important particles are thrown away.

The particle filter approach (known as Monte Carlo sampling) allows representing also the non-gaussian distributions [9]. It works well in low dimensional problems, such as localization. On the other hand, the SLAM problem is high-dimensional (we need to estimate the position of the robot and each landmark).

This problem was solved by the *FastSLAM* algorithm [12]. In combines the EKF and particle filter methods. It makes use of a modified particle filter to estimate the probability distribution. Afterwards, each particle possesses K Kalman filters that estimate the K landmark locations [11]. This reduced the computing complexity compared to EKF methods and is also usable in environments with a higher number of landmarks.

This problem is solved in another worth-mentioning SLAM implementation: *FastSLAM* algorithm. This algorithm combines the particle filters and extended Kalman filter approaches. Compared to the basic EKF approach, *FastSLAM* works efficiently even in the high-dimensional spaces and decreases the SLAM problem complexity. There is also no problem with nonlinearities. Another example of this method is *Gmapping* algorithm described in [13], very popular in the robotic community.

2.4 Graph-based methods

Another popular approach addressing SLAM problem are the graph-based methods. As the name suggests, the SLAM problem is represented with a graph data structure. Every node of that graph corresponds with a pose of the robot during mapping (the corresponding sensor data measured at this position are stored in the memory). Every edge between two nodes corresponds to spatial constraints between them.

Generally, we can divide the solution into two parts: front-end and back-end. In the front-end, the goal is to find the correspondence between nodes of the graph. This process is called as *data association.* In other words, we need to find the relation between observed data and landmarks discovered so far.

Then, in the back-end part, the optimization techniques are used to minimize the error introduced by constraints. The minimization is commonly solved via Gauss-Newton or the Levenberq-Marquart methods [8]. In other words, the goal is to find the transformation that places all corresponding measurements to the correct place on the map. These two steps go recursively (the edges are updated based on the estimated transformation). The *GraphSLAM* algorithm [14] is application of this technique.

These algorithms generally reduce the dimensionality of the SLAM problem and allow to use of modern optimization techniques. They focus on reconstruction of the robot's path $X_{0:t}$, but not all the landmarks positions. The idea behind is: when the robot path $X_{0:t}$ is known, it is easy to reconstruct the map with the observations measured in every position.

2.4.1 Scan-matching

The graph based approach in simplified in the scan-matching SLAM algorithms. The main idea is based on *data association* and reducing the computational complexity. The main difference is that the data are not localized relative to the whole map (path), but only to slightly older scans. Once the localization (using some optimization techniques) is done, it is assumed as correct. This assumption makes the algorithm much faster. On the other hand, it is not a solution to the *full SLAM* and can only accommodate very small amounts of location uncertainties, as stated in [1]. Also the *loop closing* and wrong *data association* might be an issue. The *loop closing* is the task of deciding whether the robot returned to a previously visited area. In the “full SLAM” solutions, the correct *loop closing* reduces the uncertainty of the pose estimate and provides the understanding of the real topology of the environment [8]. This is not guaranteed in *scan-matching* methods.

To give an example of *scan matching* methods, the ICP (Iterative Closest Point) algorithm consists of three steps: association, transformation and error evaluation. In the association step, each landmark in new measurement is associated with the nearest landmark in the previous measurements. Then, the optimization process minimizes the mean-squared distance between the associated points. As a result, the spatial transformation (rotation and translation) is applied to the new scan. In the third step, the mean-squared error is evaluated. If the error is over some limit, the process is repeated. Otherwise the process is terminated and the map is updated.

2.5 Other types of SLAM

Various methods were used to solve the SLAM and there is still intensive research in this field. Most of the methods mentioned above focus on the 2D SLAM, but also 3D SLAM algorithms were developed. Each application requires a different approach. Also, the range of used sensors is wide. The main group of sensors are laser rangefinders. They provide relatively accurate scanning in 2D or 3D. Another family of SLAM algorithms are called *Visual SLAM*. These algorithms are using data from a monocular or stereo camera. We can also use other types of sensors, such as sonar, radar, acoustic waves or wifi signal. The contemporary research focuses on the visual SLAM methods and the use of *deep learning* methods [8].

2.6 Opened issues

The SLAM is considered to be a solved problem on the theoretical and conceptual level. On the other hand, the research is still in progress and some open issues remain [8].

Major challenge is the development of algorithms for long term applications. In that case, the robotic system should be able to robustly perform SLAM for a longer period of time with minimal human supervision. Research is needed to develop algorithms that don’t fail after limited time, recover from mistakes and deal with the dynamically changing environments [1]. Most of the methods mentioned before work only in the largely static environments. Unfortunately, real-world scenarios are mostly dynamic. Also the sensors might be affected by the changing outdoor conditions (light, weather...).

Another direction of research is to not create the map always from scratch, but partly exploit the prior knowledge of the environment or data from other robots in multi-robot applications. Another research topic is the complexity of the SLAM algorithms. Although the algorithms have undergone significant development in this area, the efficient mapping of large areas is still challenging. Also the effort of minimizing the robotic platforms requires more efficient algorithms that would be applicable to on-board computers [8].

2.7 The SLAM algorithm for this project

The goal of this thesis is to create partly autonomous robotic system, that localizes itself and incrementally builds the map without the need of GPS signal and wheel odometry. Depending on these comparison [11] of different SLAM implementations, the *Hector slam* algorithm was selected.

2.7.1 Hector SLAM

This SLAM algorithm described in [15] is based on scan-matching principle. It provides online localization and mapping. The computing requirements are relatively small and allow the usage on the small onboard computer. Hector SLAM also does not require the robot's odometry, which is not available on the used robotic platform. It takes two-dimensional laser scan data as the input and produces two-dimensional map of the environment.

It is worth mentioning that this algorithm is not the *full SLAM* solution and we can classify it into a group of *Scan matching* methods, lacking the *loop closing* mentioned in 2.4.1. Despite this fact, it works sufficiently good in the real world scenarios [15].

Map representation

The map is represented as a grid of cells. These grid map cell values can be viewed as samples of the continuous probability distribution, representing the probability of landmark presence in the given point. The discrete representation limits the precision and does not allow the direct computation of derivatives. Because of this, there is a need for interpolation of values between "discrete samples" (cells). This approximation is managed by bilinear filtering and depends on the nearest grid values.

Working principle

The data from the sweep scanner are preprocessed (downsampling, outliers removal). The laser scan data are converted into a point cloud, where each point represents the laser beam endpoint (where the laser beam was reflected from the obstacle). The main idea of this approach is scan matching. New measurement (point cloud) is aligned with the existing map. Because of the inaccuracy of the sensor, the points from new measurement does not ideally fit the previous point clouds. The matching is done using gradient-optimization based on the Gauss-Newton approach.

We need to get transformation $\xi = (p_x, p_y, \psi)$, which minimizes the criterion

$$\xi^* = \text{argmin} \sum_{i=1}^{n} [1 - M(\mathbf{S_i}(\xi))]^{\mathbf{2}}, \tag{2.8}$$

where $\mathbf{S}_i(\xi)$ is the position of the measured landmark in the world coordinates $s_i = (s_{i_x}, s_{i_y})^T$:

$$\mathbf{S_i}(\xi) = \begin{bmatrix} \cos\psi & -\sin\psi \\ \sin\psi & \cos\psi \end{bmatrix} \begin{bmatrix} s_{i_x} \\ s_{i_y} \end{bmatrix} + \begin{bmatrix} p_x \\ p_y \end{bmatrix}. \tag{2.9}$$

$\mathbf{S}_i$ is a function of ξ, because measurements are relative to the robot position. $M(\mathbf{S_i}(\xi))$ returns the map value on the given position $\mathbf{S}_i(\xi)$. In other words, we are looking for transformation that minimizes the error between measurement and actual version of the map (landmarks present there). Also, the robot's position is represented by this transformation.

The equation 2.8 is non-linear least squares problem and can be transformed into Gauss-Newton equation. As a result, one step $\Delta\xi$ towards the minimum is evaluated, and transformation ξ is updated. This "descent method" consists of the risk of getting into local minima, because the optimized function is not convex. Despite this fact, this algorithm works sufficiently good in real applications, as stated in [15]. This risk is also minimized by multi-resolution map representation.

Multiresolution map representation

The maps with the different resolution are stored in the memory in a "pyramid" pattern. Multiple occupancy grids are used with each coarser map having half the resolution of the preceding one. When new sensor data are available, the optimization process is performed on the grid with the lowest resolution. The computed suboptimal scan matching transformation is used as a starting estimate to the following grid. This process is repeated from the map with the lowest resolution to the map with the highest resolution. All the grids with the different resolution are simultaneously updated and stored in memory. In spite of all these features, computational requirements are relatively low. Moreover, the accuracy of the created map and localization is sufficient in practice, according to [15].

Chapter 3

Preliminary

3.1 Software

3.1.1 ROS

The Robot Operating System (ROS) is a flexible open-source framework for writing robot software. ROS was originally developed by researchers from Standford University, Willow Garage and the University of Southern California [16]. It runs on Unix-based platforms and can be controlled using a command-line. It supports different programming languages, such as C++, Python, Octave and others. The code is organised into packages, which can be easily added and updated. The compilation is provided using CMAKE.

A deployed system is composed of small computing units called nodes. The structure is illustrated in figure 3.1. Nodes communicate via messages with a strictly defined structure. These messages are published into *topics*, which can be subscribed by other nodes interested in this particular message type and information included in that message. ROS also provides communication using request and response messages. This functionality is called *service*.

The communication between multiple nodes is directed by a *master*. ROS supports peer-to-peer communication, so that nodes can communicate directly with each other, while the communication is first established by a master coordinator node. ROS contains many useful existing tools. Multiple nodes can be packaged and ran together using roslaunch command. Logging and replaying measured data is provided by rosbag functionality. System structure and nodes communication could be plotted using *rqt tool*, which is useful for debugging. The *transformation library* builds transformation tree and computes transformation between different reference systems.

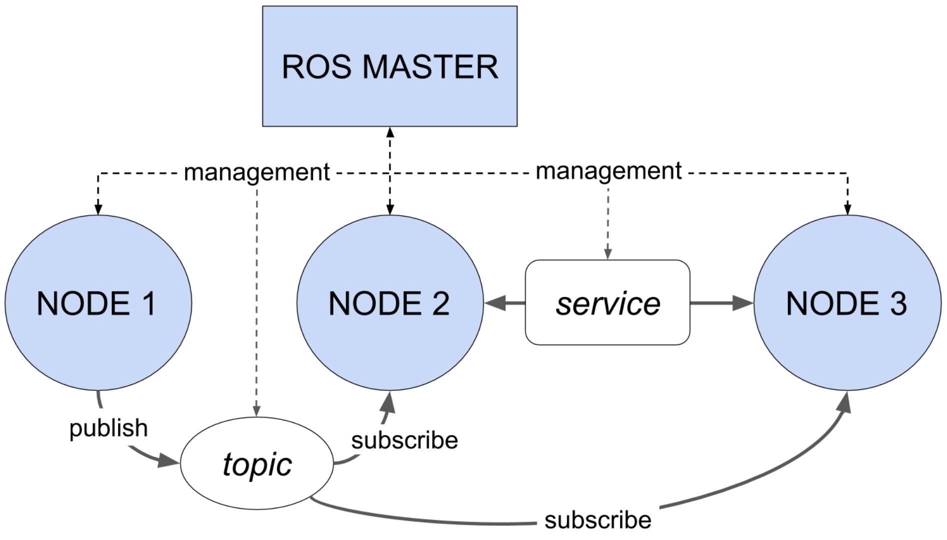
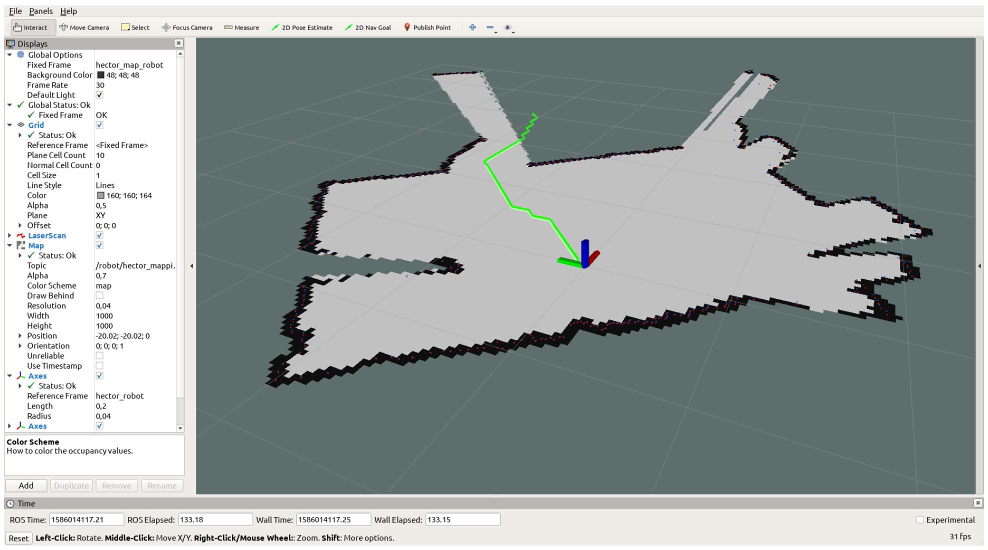


Figure 3.1: ROS node graph — vizualization of communication between nodes in a deployed system.

3.1.2 RVIZ

This program provides 3D visualisation for ROS framework. It supports all ROS standard message types. RVIZ also allows publishing into given topics which can be used for commanding the robot. It is also a powerfull tool for debugging and data vizualisation. An example of RVIZ environment if given in figure 3.2.



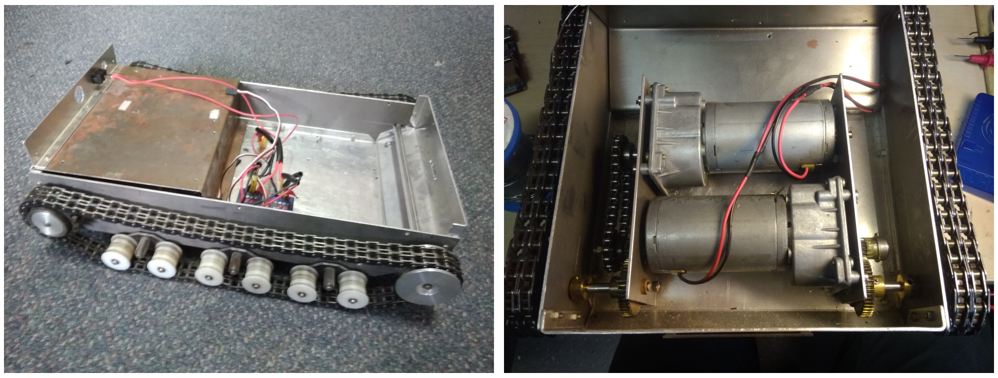
```
Global Options
  Fixed Frame          hector_map_robot
  Background Color     ■ 48; 48; 48
  Frame Rate           30
  Default Light        ✓
✓ Global Status: Ok
  ✓ Fixed Frame        OK
Grid                   ✓
  ✓ Status: Ok
  Reference Frame      <Fixed Frame>
  Plane Cell Count     10
  Normal Cell Count    0
  Cell Size            1
  Line Style           Lines
  Color                ▨ 160; 160; 164
  Alpha                0,5
  Plane                XY
  Offset               0; 0; 0
LaserScan              ✓
Map                    ✓
  ✓ Status: Ok
  Topic                /robot/hector_mappi.
  Alpha                0,7
  Color Scheme         map
  Draw Behind          ☐
  Resolution           0,04
  Width                1000
  Height               1000
  Position             -20.02; -20.02; 0
  Orientation          0; 0; 0; 1
  Unreliable           ☐
  Use Timestamp        ☐
Axes                   ✓
  ✓ Status: Ok
  Reference Frame      hector_robot
  Length               0,2
  Radius               0,04
Axes                   ✓

Color Scheme
How to color the occupancy values.

ROS Time: 1586014117.21  ROS Elapsed: 133.18  Wall Time: 1586014117.25  Wall Elapsed: 133.15   ☐ Experimental
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.   31 fps
```

Figure 3.2: A 2D map of an environment vizualized by the RVIZ tool.

3.2 Hardware

3.2.1 Hardware platform

The robot platform “Saladin” was originally created by Tomáš Báča and Antonín Novák and is described in [17]. The body of the robot is made of 1.5 mm thick aluminium plate. The dimensions of the robot with tracks are 28 × 41 cm. The undercarriage consists of two iron tracks and 16 wheels. Each track is driven by DC motor with planetary gearbox. The tracks are able to rotate in both directions, which allows the robot to rotate around a vertical axis. The motors are controlled using ZD ESC-30A motor controllers.



(a) The tracks and wheels. (b) The robot drive unit.

Figure 3.3: Hardware platform in its original state (before the work on this thesis started).

3.2.2 Onboard computer

The Odroid XU-4 is used as the computing unit. It is equipped with Samsung Exynos5422 Cortex-A15 2 GHz and Cortex-A7 Octa core CPUs with Mali-T628 MP6 GPU. The eMMC module is used as a storage. There computer has 3 USB ports, ethernet and HDMI output. It is supplied with 5 V (4 A) power supply. The system is running on a GNU/LINUX Ubuntu 18.04 operating system.

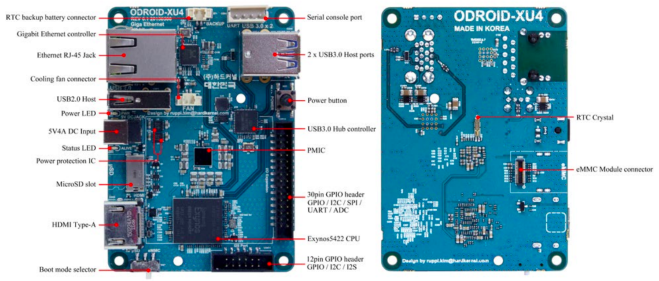


Figure 3.4: Odroid XU4. Source: [2].

3.2.3 Used sensor

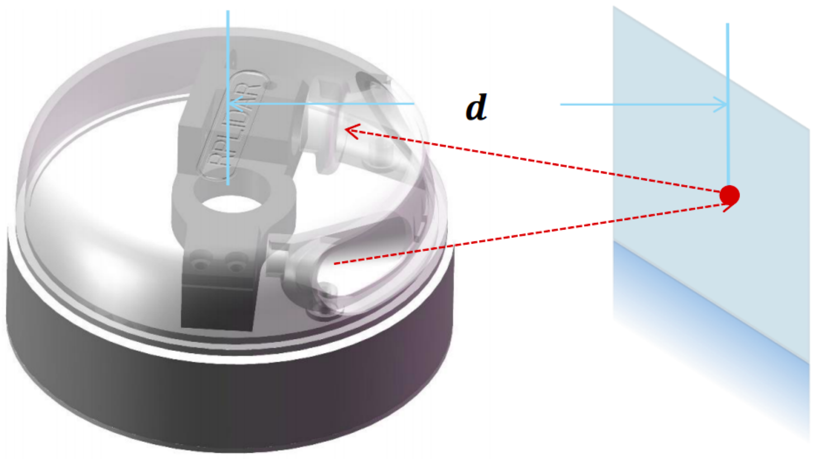


Figure 3.5: Working principle of *RPLIDAR* sensor. Source: [3]

The equipped sensor described in [3] provides distance measurement in a horizontal plane in all directions. The distance measurement is based on the "triangulation-ranging" principle. The sensor is transmitting modulated infrared laser signal. The system ranges more than 16000 times per second. When the laser signal encounters some object, it returns back to the sensor. The distance is calculated using the measured angle of reflected laser signal. The sensor uses the optical encoder to measure the actual angle of the rotating head. This is done every time the light beam is detected by the sensor. These values (measured distance and corresponding angle) are sent to the ROS topic.

3.2.4 Other hardware

These are other hardware components used in this project:

- Microcontroller: Arduino Nano with ATmega328P chip
- Wireless access point
- Switching voltage regulator Foxy UBEC 6A
- Power source: 14.8 V, 6750 mAh, Li-Po battery

Chapter 4

Control system design

The control system of the unmanned ground vehicle consists of several subsystems. First of all, the measured sensor data should be processed by the SLAM algorithm. This provides the localization and mapping of the environment. Next key component is the path planning algorithm, which finds the best way from the current position to the goal. The execution of the planned trajectory is done using waypoint tracking subsystem. It calculates the motor commands based on the planned path and current position. These commands are then sent to the motors via hardware interface: serial line and Arduino microcontroller. There is also a need of communication between UGV and operator. In the applications, such as *Urban search and rescue* (USAR) tasks at dangerous environments, the sensored data (created map and estimated position) should be presented to the operator in real-time. On the other hand, the operator should be able to give the commands to the platform, which are executed with some level of autonomy. This is managed by the user interface.

The developed control system involves all the subsystems described above: sensors, SLAM, map processing, path planning, exploration, waypoint tracking, hardware and user interface. The software is organized into *ROS packages* and implemented in `C++` and `Python`. The developed platform is able to operate in three different modes: autonomous, semi-autonomous and manual. All these parts of the system are described in this chapter.

4.1 Control system structure

The structure and all dependencies are illustrated in the figure 4.1. The subsystems are shown as rectangles (*ROS nodes* are bold). These nodes are sharing data via *ROS topics*. These *topics* are shown as ellipses. The relations between nodes (publishers and subscribers) are illustrated with arrows.

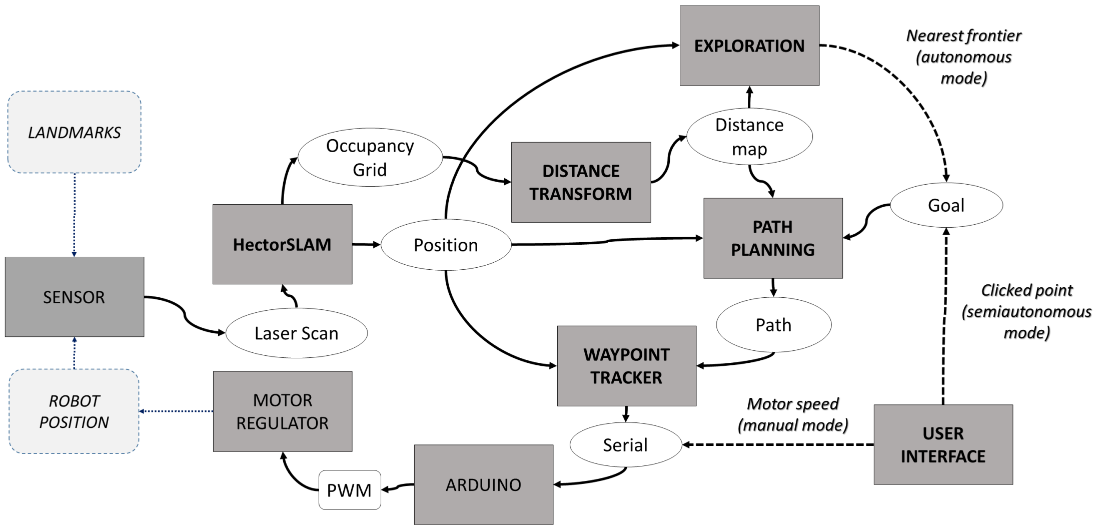


Figure 4.1: The structure of the control system. Each subsystem is square (*ROS nodes* are bold), *ROS topics* are elliptical. The dependencies are illustrated with arrows.

4.2 Coordinate system

The coordinate system is described as follows. The origin is located on the starting position of the robot (where mapping procedure started). We assume the robot is moving on flat ground. Because of this, the robot state vector $\mathbf{x}_r$ is

$$\mathbf{x}_r = \begin{bmatrix} x_r & y_r & \phi_r \end{bmatrix}^T, \tag{4.1}$$

where x_r and y_r are planar coordinates and ϕ_r describes heading of the robot. The origin of the coordinate system is placed at the spot where HectorSLAM algorithm started the mapping procedure.

4.3 SLAM

The simultaneous localization and mapping is provided by HectorSLAM[1] algorithm described in the section 2.7. The map resolution is set to 4 cm, the map size is set to 40 × 40 m. The input is data from *RPLIDAR* laser scanner in the `sensor_msgs/LaserScan` message type. The outputs are robot position $\mathbf{x}_r$ in the world coordinates and a map of the environment represented as ROS `nav_msgs/OccupancyGrid` message type. It is an array of `int8_t` data type, where an unexplored area is -1, free space is 0 and obstacle is 100.

[1]Available at:https://github.com/tu-darmstadt-ros-pkg/hector_slam

4.4 Distance transform

The goal is to efficiently calculate the distance from each cell to the nearest obstacle. This is done using the “convolutional” approach widely used in image processing algorithms. The whole procedure uses 4 sliding windows (convolution kernels) going through the map. These windows are described in 4.1.

cell*	1
1	$\sqrt{2}$

(a)→↓

$\sqrt{2}$	1
1	cell*

(b)←↑

1	cell*
$\sqrt{2}$	1

(c)←↓

1	$\sqrt{2}$
cell*	1

(d) →↑

Table 4.1: Four sliding windows used for distance transform. The cell* represents the actual position in the map, the direction of movement through the table is illustrated with arrows.

The whole distance transform procedure is described in the algorithm 1. As the first step of the distance transform algorithm, the cell values are initialized. The initial value is 0 for obstacles and $+\infty$ for free cells. Each of the four windows starts in a different corner and goes through the grid map. For example, the sliding window (a) in 4.1 starts in the top left corner, goes through the first row, second row and rest of the map. As a result, the approximate euclidean distance to the nearest obstacle is stored in each yet observed map cell. The obstacle values are set to 0, unexplored area is marked with -1 and detected frontiers are 100 in the output map.

Algorithm 1 Distance transform algorithm pseudocode

```
procedure DISTANCE TRANSFORM
    Input:
        map                                   ▷ Output of HectorSLAM: nav_msgs/OcuupancyGrid map
    for each cell in map do
        if map_value(cell) is obstacle then
            map_value(cell) ← 0
        else
            map_value(cell) ← $\infty$
        end if
    end for
    for each window in sliding windows do                ▷ Four sliding windows described in 4.1
        for each cell* in map do          ▷ cell* refers to the actual window position, as shown in 4.1
            for each neighbour in window do      ▷ neighbour refers to the other window cells, as in 4.1
                if map_value(neighbour) > map_value(cell*)+window_value(neighbour) then
                    map_value(neighbour) ← map_value(cell*)+window_value(neighbour)
                end if
            end for
        end for
    end for
    detect_frontiers()                                                  ▷ described in 4.6
    Output:
        map                                                             ▷ updated map
end procedure
```

4.5 Path planning

The goal is to find path towards the goal, which is:

- **optimal**: the shortest path to the goal
- **safe**: the robot avoids collision with obstacles

The safety could be ensured if the robot drives in the safe distance from obstacles. The exact safe distance for this type of robot is difficult to define. The robot is able to drive forward or rotate on the spot. The rotation requires much bigger "safe space" than forward motion. If we define this limit arbitrary, the robot will not plan the path to the spots, that are reachable by some type of motion. Another approach is to drive "as far as possible" from the obstacles. Finally, the "safety" is in contrast to optimality in some way and both requests should be balanced.

Path planning is provided by modified A* algorithm. The distance map (transformed output of HectorSLAM) is taken as a graph with nodes n. The goal is to find optimal way from the robot actual position n_{start} to the goal n_{goal}. The motion in the graph is allowed in 8-neighbourhood. Because of that, the grid distance $dist_{grid}$ (defined in pseudocode 2) is used in the cost and heuristic function. Moreover, it is necessary to avoid obstacles along the route and drive around all obstacles in the safe distance. The expanding function omits the cells closer than 20 cm from obstacles as the minimal safe distance for the motion. It also penalizes the positions that are closer than 1 m to the obstacles.

The algorithm is based on an evaluation function

$$f(n_i) = h(n_i) + g(n_i), \tag{4.2}$$

where $h(n_i)$ is the heuristic function, $g(n_i)$ is the cost function and n_i is the current node in the algorithm. The heuristic function is calculated as:

$$h(n_i) = \text{dist}_{grid}(n_i, n_{goal}) + \alpha p(n_i). \tag{4.3}$$

The $p(n_i)$ is the penalty function, which penalizes the nodes that are closer than 1 m to the nearest obstacle as follows:

$$p(n_i) = \begin{cases} 1 - m(n_i) & \text{if } (1 - m(n_i)) > 0 \\ 0 & \text{otherwise} \end{cases},$$

where $m()$ is the recalculated diagonal distance to the nearest obstacle from the distance-trasformed map in meters. It is weighted by the parameter α. This parameter sets how much the path planning algorithm is avoiding obstacles. The cost function is defined as

$$g(n_i) = \sum_{a=1}^{i} \text{dist}_{grid}(n_{a-1}, n_a). \tag{4.5}$$

In other words, the cost function $g(n_i)$ represents the real distance from starting node to the current node n_i.

Algorithm description

The planning process starts at the starting node n_{start}. The current node is expanded. Its neighbours are added to the `open_nodes` list (list of explored notes) and their $f()$ value is calculated (nodes closer than 20 cm to the nearest obstacle are omitted to avoid collision with the obstacle). After that, the nodes in `open_nodes` list are sorted depending on their $f()$ value (this is implemented using heap data structure). The node with the lowest $f()$ value is popped and taken as the new current node. The previous node is added to the `closed_nodes` list (list of visited nodes). When the goal node is reached, the path is reconstructed. If the `open_nodes` list is empty and the goal was not reached yet, there is no path to the goal.

Algorithm 2 modified A* algorithm

```
procedure FIND PATH($map, n_{start}, n_{goal}$)
    $open\_nodes$                                   ▷ heap data structure sorted by $f()$ value
    $closed\_nodes$
    $open\_nodes \leftarrow n_{start}$              ▷ add initial position
    while $open\_nodes$ not empty do
        $n_{current} \leftarrow open\_list$.pop()   ▷ Get node with lowest $f()$ value from heap
        $new\_nodes$ = expand($n_{current}$)        ▷ find all accessible neighbours
        $closed\_nodes$.add($n_{current}$)
        if $n_{current}$ is $n_{goal}$ then
            return reconstruct_path()               ▷ return path
        end if
        for each $n_i$ in $new\_nodes$ do
            if $n_i$ in $closed\_nodes$ then
                continue
            end if
            $h(n_i) \leftarrow$ GRID_DISTANCE($n_i, n_{goal}$) $+ \alpha p(n_i)$
            $f(n_i) \leftarrow h(n_i) + g(n_i)$
            if $n_i$ in $open\_nodes$ then
                if $g(n_i) < g(open\_nodes[n_i])$ then   ▷ new $g()$ value is better than previous for the given node
                    update $n_i$
                end if
            else
                $open\_nodes$.add($n_i$)
            end if
        end for
    end while
    return $None$                                   ▷ no path to the goal
end procedure
procedure GRID_DISTANCE($n_a, n_b$)
    $dx \leftarrow |x_a - x_b|$
    $dy \leftarrow |y_a - y_b|$
    return $|dx - dy| + \sqrt{2(\min(dx, dy))^2}$
end procedure
```

Optimality and variability

According to the theory, the A* algorithm finds the optimal path if the heuristic is consistent and admissible. The heuristic $h()$ is admissible, if

$$0 \leq h(n) \leq h^*(n)\ \forall n, \tag{4.6}$$

where $h^*(n)$ is the true cost of going from n to the nearest goal). In this case, the $h()$ is not admissible heuristic, because $p()$ is not calculated in the cost function $g()$. Despite this fact, this algorithm provides sufficient path planning with avoiding obstacles. The results are presented in the next chapter.

The parameter α could be used for changing the behaviour of the algorithm. If $\alpha = 0$, the algorithm is equivalent to the standard A* (with admissible and consistent heuristics). The greater the α is, the more the algorithm prioritizes the obstacle avoidance along the planned path and minimize the risk of collision with an obstacle. On the other hand, a big value of α may increase the time of path planning. If the $p()$ is much greater than the calculated distance to the goal, the algorithm behaves more like a breadth-first-search algorithm and expands more nodes.

4.6 Exploration

In the autonomous mode, the robot is able to find the nearest reachable unexplored area and pass this goal to the path planning algorithm. The exploration is based on frontier detection. This approach was described in [18]. As frontier is taken the discovered field on the grid map that is adjacent to at least 2 undiscovered fields. These frontiers are detected in the distance transform node. The exploration process is based on the Breadth-first search (BFS) algorithm. The exploration starts at the actual robot position and expands all discovered positions, that are at least 20 cm from the nearest obstacle. This ensures that non-reachable positions are ommited. After the frontier is found, it is sent to the planning subsystem as the new goal for the robot.

4.7 Waypoint tracking

The waypoint tracking subsystem should control the robot motion in order to follow the planned trajectory. The first input of this algorithm is the estimated robot position from SLAM algorithm $\mathbf{x}_r$. The second input is the planned path of the robot, described as

$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1 & ... & \mathbf{w}_n \end{bmatrix}, \tag{4.7}$$

where $\mathbf{w}_i = \begin{bmatrix} x_{w_i} \\ y_{w_i} \end{bmatrix}$ is the coordinate of the waypoint.

The robot as always navigating to the one selected current waypoint $\mathbf{w}_c$. In the beginning, the robot is navigating to the first waypoint $\mathbf{w}_c := \mathbf{w}_0$. If the distance between robot position and current waypoint is below the setted limit $dist_{min}$, the current waypoint is updated $\mathbf{w}_c := \mathbf{w}_{c+1}$. This condition is checked by the internal ROS timer at 100 Hz frequency. This process continues until the last waypoint of the path is reached.

The motion towards the actual waypoint $\mathbf{w}_c$ is calculated every time the update of robot position comes from SLAM algorithm. First of all, the angle to the next waypoint is calculated as:

$$\phi_{\mathbf{w}_c} = \operatorname{atan2}(y_{w_c} - y_r, x_{w_c} - x_r). \tag{4.8}$$

The angles ϕ_{w_i} and ϕ_r are in range $[-\pi, \pi]$ (in the world coordinates). Because of this, the difference $\Delta\phi$ between robot orientation ϕ_r and angle to the next waypoint ϕ_{w_i} is calculated as:

$$\Delta\phi = \begin{cases} (\phi_{\mathbf{w}_c} - \phi_r) + 2\pi & \text{if } (\phi_{\mathbf{w}_c} - \phi_r) < -\pi \\ (\phi_{\mathbf{w}_c} - \phi_r) - 2\pi & \text{if } (\phi_{\mathbf{w}_c} - \phi_r) > \pi \\ \phi_{\mathbf{w}_c} - \phi_r & \text{otherwise} \end{cases} .$$

Based on control error $\Delta\phi$, the controller action is calculated. The chassis track construction allows rotation around the vertical axis. Expected operating space of the robot is an indoor environment. To ensure the ability of movement in the confined spaces and also the continuity of the motion in open areas, the robot uses the rotation only when $\Delta\phi$ is greater the setted limit ϕ_{rotate} =0.8 rad. When the control error is small, the robot drives towards the next waypoint. This movement forward is controlled by the proportional regulator. The whole control system is described in the algorithm 3:

Algorithm 3 The control system

```
procedure POSITION CALLBACK
    Input:
        $\mathbf{w}_c$                                   ▷ current waypoint
        $\mathbf{x}_r$                                   ▷ position of the robot
    Output:
        $right\_motor$                           ▷ command to left motor
        $left\_motor$                            ▷ command to right motor
    calculate $\phi_{w_c}$          ▷ calculate angle to the actual waypoint, via eq. 4.8
    calculate $\Delta\phi$                  ▷ calculate control error, via eq. 4.9
    if $-\phi_{rot} < \Delta\phi < \phi_{rot}$ then                  ▷ rotate on the spot
        if $\Delta\phi < 0$ then
            $left\_motor \leftarrow backward$
            $right\_motor \leftarrow forward$
        else
            $left\_motor \leftarrow forward$
            $right\_motor \leftarrow backward$
        end if
    else                                   ▷ drive towards the actual waypoint
        $left\_motor \leftarrow (forward - P\Delta\phi)$
        $right\_motor \leftarrow (forward + P\Delta\phi)$
    end if
end procedure

procedure TIMER CALLBACK                          ▷ 100 Hz timer callback
    $dist \leftarrow distance(\mathbf{x}_r, \mathbf{w}_c)$
    if $dist < dist_{min}$ then
        $\mathbf{w}_c \leftarrow \mathbf{w}_{c+1}$                      ▷ update current waypoint
    end if
end procedure
```

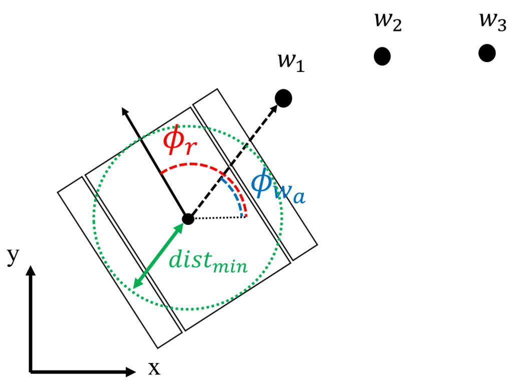


Figure 4.2: Waypoint tracking procedure vizualization. ϕ_r (red) is the heading of the robot, ϕ_{w_c} (blue) is the angle to the current waypoint, $dist_{min}$ (green) is the distance when the current waypoint is updated.

4.8 Hardware interface

The commands calculated in the waypoint tracking algorithm should be sent to the hardware platform and transformed into PWM signal commanding the motors. This is managed by the serial line protocol and Arduino microcontroller.

4.8.1 Serial line protocol

After the control command is calculated, it is sent to the Arduino via serial line. The communication between ROS and Arduino microcontroller is provided using the ROS package `mrs_serial`[2] . One message consists of 6 bytes of data, where each byte is represented as 8 bits. The message is defined as

```
[b][payload_size][ID][left_motor_command][right_motor_command][checksum]
```

where byte `'b'` represents the beginning of the message, `payload_size` defines length of the message, `ID` defines message type. Bytes `left_motor_command` and `right_motor_command` of type `uint8_t` represent desired motor speed. This value is in range 0–200, where 0–99 represents backward motion, 100 stop command, 101–200 forward motion. After all, the checksum is calculated and send via serial line. This ensures the validity of data received on the Arduino.

[2]Available at:https://github.com/ctu-mrs/mrs_serial

4.8.2 Arduino

The code is written in the programming language `wiring` based on C++. The Arduino receives a message via serial line. According to received commands, Arduino generates PWM(pulse width modulation) signal. Pulse width modulation is based on generating a square wave oscillating between 0 and 5 V. Depending on the pulse width, the motor controller sets the speed on the motor.

Another device connected to the Arduino is the sound buzzer. It indicates the source of command messages (manual/automatic control). In manual mode, the signal 30 ms long is heard every 5 seconds. The autonomous and semi-autonomous mode is indicated every second by signal of the same length. The error state (no messages coming from ROS) is indicated with 300 ms signal repeated every second.

4.9 Modes of operation

The communication between operator and robot is provided using the *Wifi* access point. The robotic platform is able to operate in 3 different modes: autonomous, semi-autonomous and manual:

- **autonomous mode (exploration)**: the robot is searching for the nearest reachable frontier (border point between explored and unexplored area) and drives there autonomously without any human input
- **semi-autonomous mode**: human operator clicks to the map and robot drives autonomously to that location
- **manual mode**: human operator controls the motion of the tracks directly

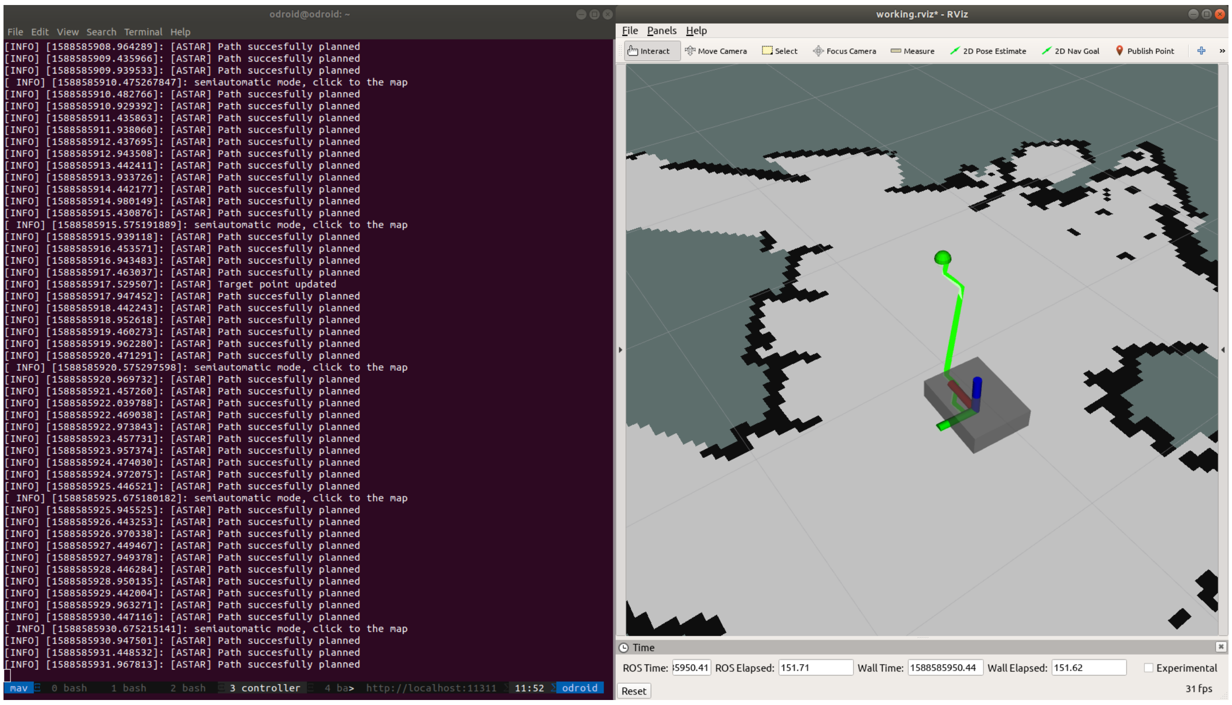
4.10 User interface

The estimated map and position of the robot is presented to the operator in the RVIZ tool, as illustrated on the right side of the figure 4.3. The planned path is marked with the green line, to robot position is shown as the grey cube and axis determining the robot orientation. In the semi-autonomous mode, the desired goal position is clicked to the map in RVIZ. The mode of operation is switched using the command line control interface illustrated in 4.3. The terminal interface is commanded by following commands:

Keyboard input	**command**
1	switch to manual mode
2	switch to semi-autonomous mode
3	switch to autonomous mode

Manual mode	**command**
Q	left motor forward
A	left motor back
E	right motor forward
D	right motor back
W	drive forward
S	drive backward
X	stop

Table 4.2: Keyboard commands for human operator



```
[INFO] [1588585908.964289]: [ASTAR] Path succesfully planned
[INFO] [1588585909.435966]: [ASTAR] Path succesfully planned
[INFO] [1588585909.939533]: [ASTAR] Path succesfully planned
[ INFO] [1588585910.475267847]: semiautomatic mode, click to the map
[INFO] [1588585910.482766]: [ASTAR] Path succesfully planned
[INFO] [1588585910.929392]: [ASTAR] Path succesfully planned
[INFO] [1588585911.435863]: [ASTAR] Path succesfully planned
[INFO] [1588585911.938060]: [ASTAR] Path succesfully planned
[INFO] [1588585912.437695]: [ASTAR] Path succesfully planned
[INFO] [1588585912.943508]: [ASTAR] Path succesfully planned
[INFO] [1588585913.442411]: [ASTAR] Path succesfully planned
[INFO] [1588585913.933726]: [ASTAR] Path succesfully planned
[INFO] [1588585914.442177]: [ASTAR] Path succesfully planned
[INFO] [1588585914.980149]: [ASTAR] Path succesfully planned
[INFO] [1588585915.430876]: [ASTAR] Path succesfully planned
[ INFO] [1588585915.575191889]: semiautomatic mode, click to the map
[INFO] [1588585915.939118]: [ASTAR] Path succesfully planned
[INFO] [1588585916.453571]: [ASTAR] Path succesfully planned
[INFO] [1588585916.943483]: [ASTAR] Path succesfully planned
[INFO] [1588585917.463037]: [ASTAR] Path succesfully planned
[INFO] [1588585917.529507]: [ASTAR] Target point updated
[INFO] [1588585917.947452]: [ASTAR] Path succesfully planned
[INFO] [1588585918.442243]: [ASTAR] Path succesfully planned
[INFO] [1588585918.952618]: [ASTAR] Path succesfully planned
[INFO] [1588585919.460273]: [ASTAR] Path succesfully planned
[INFO] [1588585919.962280]: [ASTAR] Path succesfully planned
[INFO] [1588585920.471291]: [ASTAR] Path succesfully planned
[ INFO] [1588585920.575297598]: semiautomatic mode, click to the map
[INFO] [1588585920.969732]: [ASTAR] Path succesfully planned
[INFO] [1588585921.457260]: [ASTAR] Path succesfully planned
[INFO] [1588585922.039788]: [ASTAR] Path succesfully planned
[INFO] [1588585922.469038]: [ASTAR] Path succesfully planned
[INFO] [1588585922.973843]: [ASTAR] Path succesfully planned
[INFO] [1588585923.457731]: [ASTAR] Path succesfully planned
[INFO] [1588585923.957374]: [ASTAR] Path succesfully planned
[INFO] [1588585924.474030]: [ASTAR] Path succesfully planned
[INFO] [1588585924.972075]: [ASTAR] Path succesfully planned
[INFO] [1588585925.446521]: [ASTAR] Path succesfully planned
[ INFO] [1588585925.675180182]: semiautomatic mode, click to the map
[INFO] [1588585925.945525]: [ASTAR] Path succesfully planned
[INFO] [1588585926.443253]: [ASTAR] Path succesfully planned
[INFO] [1588585926.970338]: [ASTAR] Path succesfully planned
[INFO] [1588585927.449467]: [ASTAR] Path succesfully planned
[INFO] [1588585927.949378]: [ASTAR] Path succesfully planned
[INFO] [1588585928.446284]: [ASTAR] Path succesfully planned
[INFO] [1588585928.950135]: [ASTAR] Path succesfully planned
[INFO] [1588585929.442004]: [ASTAR] Path succesfully planned
[INFO] [1588585929.963271]: [ASTAR] Path succesfully planned
[INFO] [1588585930.447116]: [ASTAR] Path succesfully planned
[ INFO] [1588585930.675215141]: semiautomatic mode, click to the map
[INFO] [1588585930.947501]: [ASTAR] Path succesfully planned
[INFO] [1588585931.448532]: [ASTAR] Path succesfully planned
[INFO] [1588585931.967813]: [ASTAR] Path succesfully planned

 mav   0 bash   1 bash   2 bash   3 controller   4 ba>  http://localhost:11311  11:52  odroid
```

ROS Time: 5950.41 ROS Elapsed: 151.71 Wall Time: 1588585950.44 Wall Elapsed: 151.62 Experimental

31 fps

Figure 4.3: Control interface for human operator. On the left side, the operator can give commands (mode switching/manual control) via command line and see what is happening in the control system. On the right side, the map, current path and robot position are presented in RVIZ tool.

Chapter 5

Testing and results

The developed UGV system was tested in the real-world environments. It is important to mention that objective testing of such a complex system is challenging. The used platform is unique and there is no second control system for comparison available. The testing is divided into two parts. Firstly, the main subsystems (path planning, waypoint tracking) are evaluated separately. Secondly, the created system is tested as a whole in outdoor and indoor conditions. These experiments are also used for evaluation of autonomous and semi-autonomous mode. All these results are presented and commented in this chapter. The scale of each map presented below is represented with a grid with 1 m per cell.

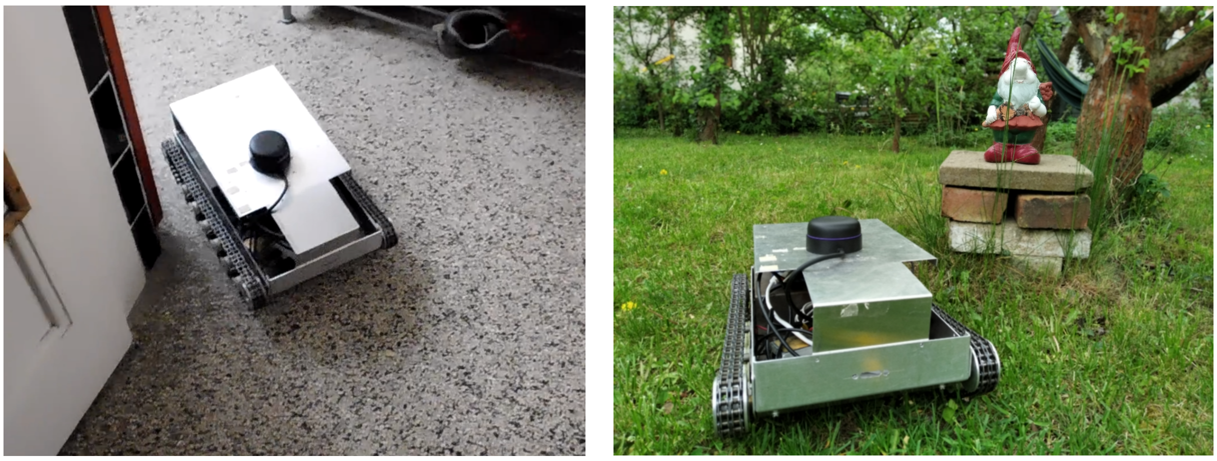


Figure 5.1: Testing indoors and outdoors.

5.1 Planning

The objective evaluation of the planned trajectory is challenging. As mentioned in chapter 4.5, the planned path should be safe (avoiding obstacles) and optimal (shortest), which is contrary in some way. The path planning was tested in several experiments. An example of planned trajectory is shown in 5.2. As we can see, the path is avoiding obstacles in a safe distance, and the shorter path (from two options) was chosen by the algorithm. It is planned only in the 8-neighbourhood and the green curve connecting all planned waypoints is not "smooth".

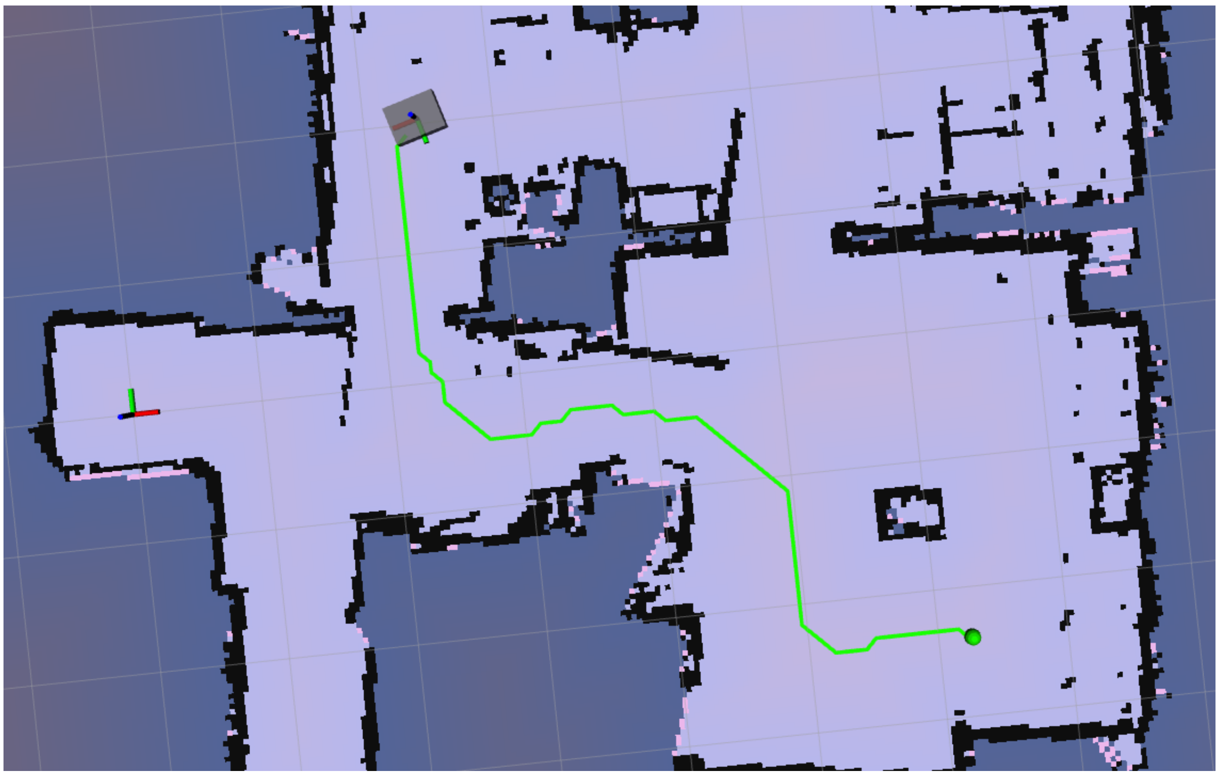


Figure 5.2: An example of the planned path (green).

5.2 Waypoint tracking

The waypoint tracking subsystem should control the motion according to the planned path. The comparison of planned (green) and executed (red) trajectory is shown in figure 5.3. We can see that the biggest control deviation is caused by the shape of the planned path, which is planned only in 8-neighbourhood. The quantitative evaluation of path following precision is difficult to calculate, because of the used waypoint tracking and path planning method. It is important to mention that the precise following of the path planned in 8-neighbourhood is not desired behaviour. The shape of the executed path (if the motion is "smooth" and not oscillating) is more important. The maximal distance between the planned and executed path (between the nearest points of both) is below 7 cm. In general, the planned path is smooth and follows the executed path in the safe distance from obstacles.

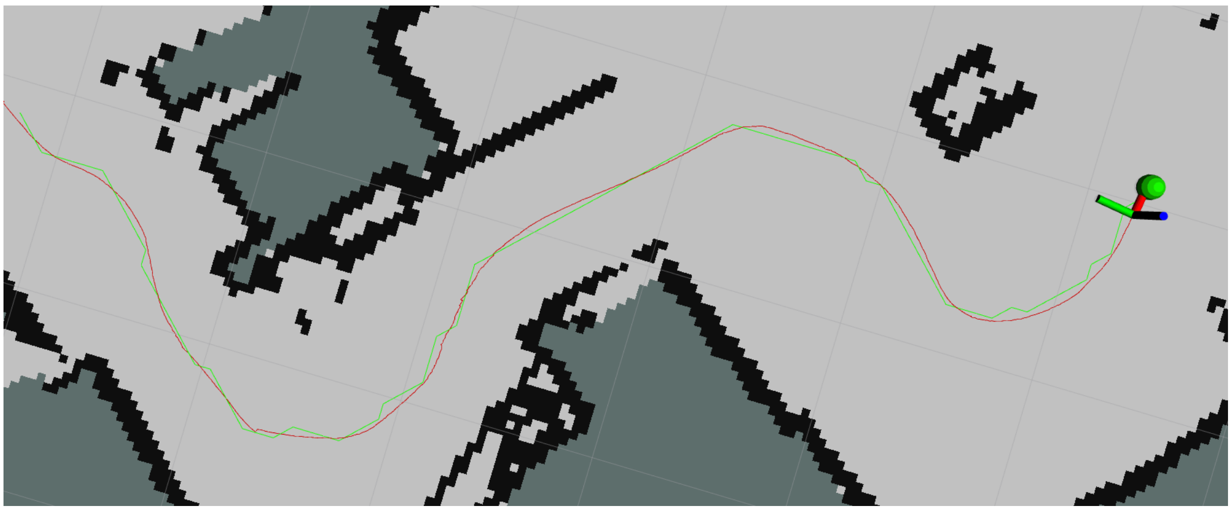


Figure 5.3: Comparison of the planned path (green) and the real path of the robot (red)

5.3 Experiment 1: Indoor autonomous mapping

The autonomous mode was tested in the indoor environment of a small house. The created map is presented in the figure 5.4, video is available at http://mrs.felk.cvut.cz/saladin. The robot was placed in the starting position (1). After the autonomous mode was enabled, it explored the reachable area without any commands from the human operator. The path of the robot is marked with red line.

The comparison between created map and precise construction plan of the building is presented in figure 5.5. The sensors were able to spot all the main obstacles in the indoor environment. The table legs and chairs are visible at position (2). The robotic platform also overcame the door sill during the motion through the house position (3). In that situation, the sensor was not in the horizontal position for a few seconds. Despite this fact, the used SLAM algorithm tackled with that situation and mapping procedure was not broken. We can state that the chosen sensor and SLAM algorithm are sufficient for mapping and path planning indoors.

As we can see from the marked path (red) in figure 5.4, the robot avoided all the obstacles safely. The exploration subsystem worked — no unexplored reachable areas left. On the other hand, the BFS exploration is computationally expensive and takes significant time to compute. For that reason, the robot sometimes stopped for few seconds until new frontier was found. Another drawback of the used exploration method (revealed during testing) is zigzag behavior in some situations. The system is always navigating to the nearest frontier and sometimes changes the direction of the motion several times in the row. This problem could be solved by planning to the farthest frontier, which would require a different exploration approach.

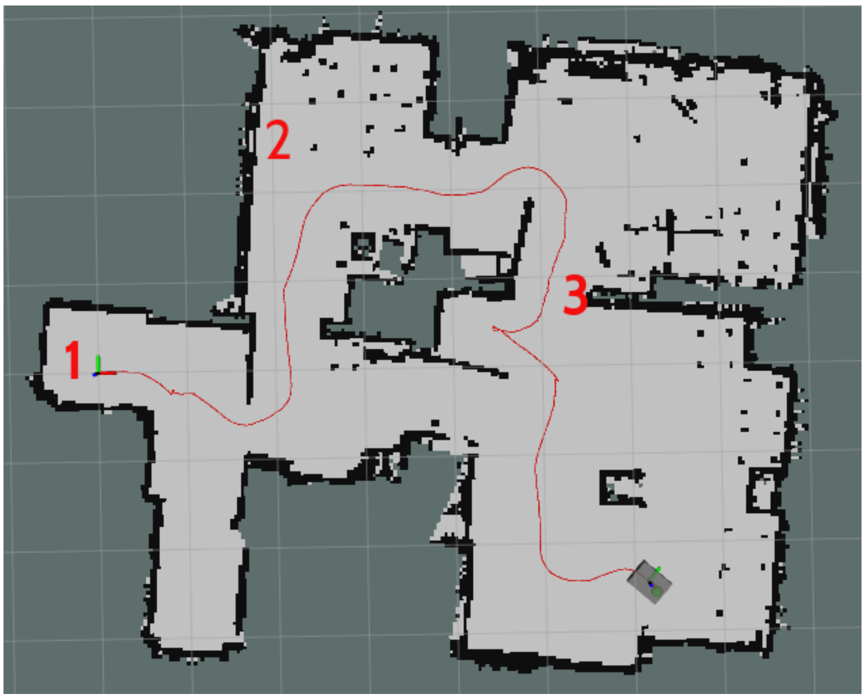


Figure 5.4: Experiment 1: Final map created in autonomous mode. Executed path is red.

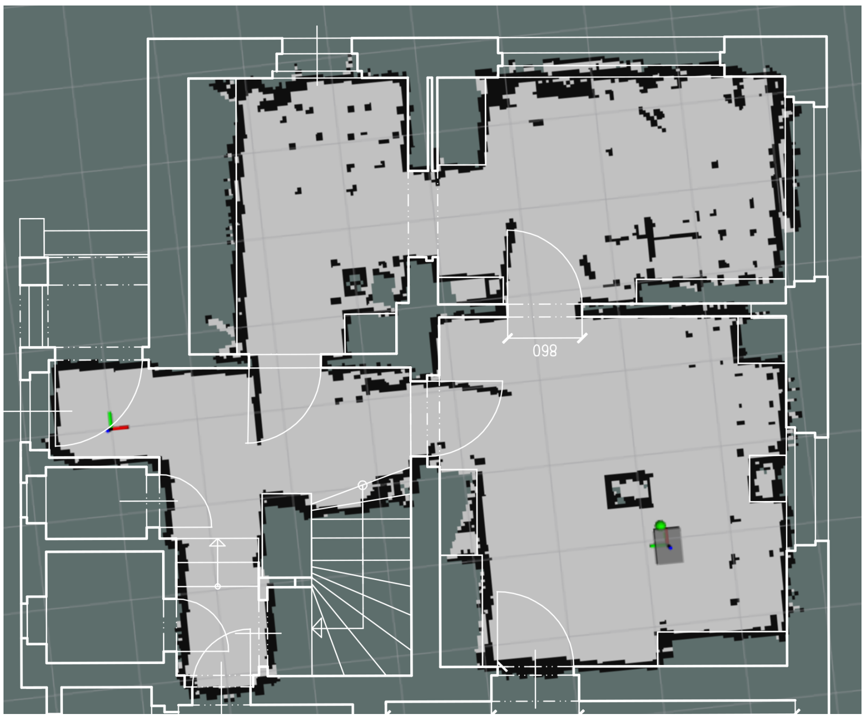


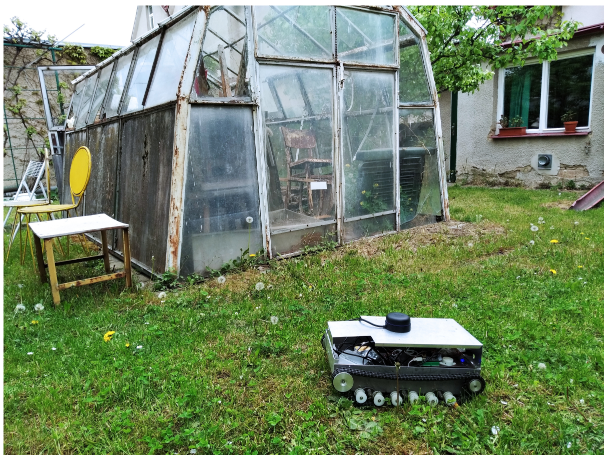
Figure 5.5: Experiment 1: Comparison of the created map and construction plan.

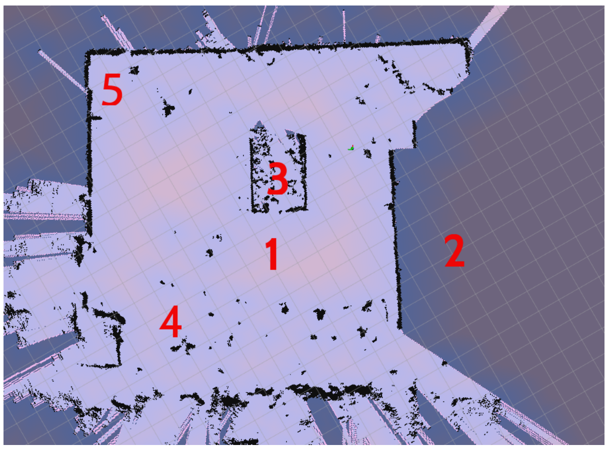
5.4 Experiment 2: semi-autonomous outdoor mapping

The robotic platform was tested in the garden around a family house. The movement of the robot was controlled in semi-autonomous mode. In other words, the operator controlled the motion via setting the goal in the map. The experimental results are shown in 5.6 and 5.7. The robot started at the position (1), then drove clockwise around family house (2) and then returned back to the starting position. The greenhouse is located at (3).

As we can see from the maps, the mapping algorithm is able to spot all the obstacles in the horizontal level, such as trees (4), walls and greenhouse. The biggest issues were the lack of 3D spatial information and hardware limitations of the robotic platform. The used 2D sensor is not able to spot some spatial conditions, such as terrain unevenness, ground steps and others. As a result, the path planning subsystem is not avoiding these obstacles.

Also the *scan matching* SLAM technique limitations are shown in 5.7. After the robot drove around the house, the sensor was not in the horizontal position for the moment and the sensor data were wrongly associated. The left side of the map 5.7 is shifted, which is visible in the corner of the garden (5). After this wrong data association, the simultaneous localization and mapping procedure was broken and not corresponding with the real environment.


(a) Starting position (1) and greenhouse (3)



(b) Created map in the middle of experiment

Figure 5.6: Experiment 2: Mapping in the garden. The environment is illustrated on left, the map in the middle of the experiment is on right side.

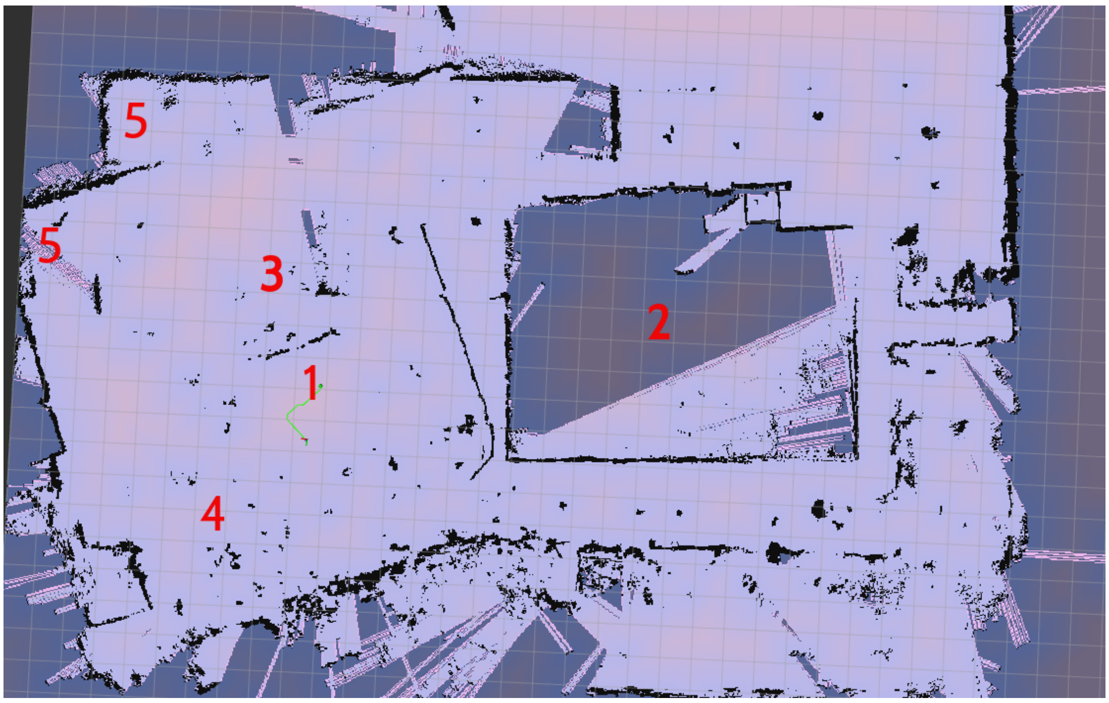


Figure 5.7: Experiment 2: Final map of the garden. The robot started at position (1) and drove around the house (2) clockwise and returned back to starting position. The mapping procedure was broken in the last part of experiment, which is obvious in corner of the garden (5).

5.5 Summary

According to the experiments, the developed platform is able to fulfill the main task: safely drive towards the given goal. The used methods of path planning and waypoint tracking are relatively simple (the planning is provided only in 8-neighbourhood, the path is not “smooth”, the robot is navigating to the waypoint which is in front of the robot). On the other hand, the combination of these methods is sufficient for the given application.

The used combination of sensor and SLAM algorithm works well in the indoor environment. The outdoor application is possible only if the terrain is not significantly uneven. The sensor cannot spot obstacles which are too thin or under the level of measurement. This is visible in figure 5.5, where the downstairs were not detected by the control system.

The semi-autonomous mode is working and control of the robot is simple. The option of commanding the robot manually is helpful in some situations when the robot encounters some obstacles, which are not detected by the sensor. The autonomous mode is applicable and the platform is able to explore the area without direct commands.

Chapter 6

Conclusion

This thesis presented a solution to the feedback control system of the tracked robot. The brief overview of key principle for autonomous robots — SLAM — was given in chapter 2. The robotic platform was equipped with sensor, onboard computer, microcontroller, power source and other hardware providing the communication and motor control. Used hardware was described in chapter 3.

As the main part of the thesis, the autonomous control system was developed in the ROS environment. The HectorSLAM algorithm was used for simultaneous localization and mapping. The produced map is then transformed using the distance transform algorithm. These data are used for path planning. Finally, the waypoint tracking system together with the developed hardware interface control the motion towards the goal.

The developed system is able to work in three different modes. In the autonomous mode, the platform explores the reachable unexplored areas without direct human control. In the semi-autonomous mode, the robot navigates itself to the given goal. Lastly, the robot could be controlled directly by the operator.

The developed system was tested in both indoor and outdoor conditions. According to this testing, the robot could be used for mapping of unknown indoor environment. The developed control system is able to safely navigate towards the given goal. In conclusion, it is possible to state that assignment has been fulfilled with some improvement (autonomous exploration).

6.1 Future work

Work presented in this thesis provides an introduction to the topic of unmanned ground vehicles. There presented solution could be improved in different ways. First of all, the used sensor and SLAM algorithm provides only 2D mapping, which is not sufficient in some real-world scenarios. The robotic platform could be equipped with some extra sensors, such as inertial measurement unit, 3D lidar, camera or depth sensor. This could improve the precision and reliability of mapping and localization procedure in environments with a significantly uneven surface. The motion planning procedure could be improved by modelling and planning the whole motion of the robot, not only the path of the central point of the robot. Also, the goal-setting in the autonomous mode could be improved to higher efficiency with some more advanced methods.

Bibliography

[1] B. Siciliano and O. Khatib, “Simultaneous localization and mapping,” in *Springer Handbook of Robotics*. Springer Publishing Company, Incorporated, 2008, pp. 871–875.

[2] R. Roy and V. Bommakanti, “Odroid xu-4 user manual.” [Online]. Available: https://magazine.odroid.com/wp-content/uploads/odroid-xu4-user-manual.pdf

[3] SLAMTEC, “Rplidar a3 introduction and datasheet.” [Online]. Available: https://www.slamtec.com/en/Lidar/A3

[4] BOSTON DYNAMICS, “Spot: compact, four-legged robot.” [Online]. Available: https://www.bostondynamics.com/spot

[5] T. Rouček, M. Pecka, P. Čížek, T. Petříček, J. Bayer, V. Šalanský, D. Heřt, M. Petrlík, T. Báča, V. Spurný, F. Pomerleau, V. Kubelka, J. Faigl, K. Zimmermann, M. Saska, T. Svoboda, and T. Krajník, “Darpa subterranean challenge: Multi-robotic exploration of underground environments,” in *Modelling and Simulation for Autonomous Systems*, J. Mazal, A. Fagiolini, and P. Vasik, Eds. Cham: Springer International Publishing, 2020, pp. 274–290.

[6] H. Qin, Z. Meng, W. Meng, X. Chen, H. Sun, F. Lin, and M. H. Ang, “Autonomous exploration and mapping system using heterogeneous uavs and ugvs in gps-denied environments,” *IEEE Transactions on Vehicular Technology*, vol. 68, no. 2, pp. 1339–1350, 2019.

[7] Z. Kashino, G. Nejat, and B. Benhabib, “Aerial wilderness search and rescue with ground support,” *Journal of Intelligent Robotic Systems*, 11 2019.

[8] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. Leonard, “Past, present, and future of simultaneous localization and mapping: Towards the robust-perception age,” *IEEE Transactions on Robotics*, vol. 32, no. 6, p. 1309–1332, 2016.

[9] H. D. Whyte and T.Bailey, “Simultaneous localization and mapping: part i,” *IEEE Robotics Journal Automation Magazine*, vol. 13, no. 2, pp. 90–110, 2006.

[10] M. W. M. G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba, “A solution to the simultaneous localization and map building (slam) problem,” *IEEE Transactions on Robotics and Automation*, vol. 17, no. 3, pp. 229–241, 2001.

[11] J. M. Santos, D. Portugal, and R. P. Rocha, “An evaluation of 2d slam techniques available in robot operating system,” *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pp. 1–6, 2013.

[12] S. Thrun and M. Montemerlo, “The graph slam algorithm with applications to large-scale mapping of urban structures,” *I. J. Robotic Res.*, vol. 25, pp. 403–429, 05 2006.

[13] G. Grisetti, C. Stachniss, and W. Burgard, “Improved techniques for grid mapping with rao-blackwellized particle filters,” *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007.

[14] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, “A tutorial on graph-based SLAM,” *IEEE Transactions on Intelligent Transportation Systems Magazine*, vol. 2, pp. 31–43, 2010.

[15] S. Kohlbrecher, J. Meyer, T. Graber, K. Petersen, U. Klingauf, and O. Von Stryk, “Hector open source modules for autonomous mapping and navigation with rescue robots,” 01 2014.

[16] M. e. a. Quigley, “Ros: an open-source robot operating system,” *ICRA workshop on open source software*, vol. 3, no. 3.

[17] T. Báča and A. Novák, “Autonomní robotické vozidlo saladin (autonomous robotic vehicle saladin),” *Středoškolská odborná činnost*, vol. 32, 2010. [Online]. Available: http://soc.nidv.cz/archiv/rocnik32/obor/10

[18] B. Yamauchi, “A frontier-based approach for autonomous exploration,” *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA’97. ’Towards New Computational Principles for Robotics and Automation’*, pp. 146–151, 1997.

Appendices

CD Content

In Table 1 are listed names of all root directories on CD.

Directory name	**Description**
thesis	the thesis in pdf format
ros_packages	implementation of path planning, waypoint tracking, distance transform and exploration
arduino_code	code for Arduino

Table 1: CD Content

List of abbreviations

In Table 2 are listed abbreviations used in this thesis.

Abbreviation	**Meaning**
SLAM	Simultaneous Localization And Mapping
PWM	Pulse Width Modulation
UGV	Unmanned Ground Vehicle
USAR	Urban Search And Rescue
ROS	Robot Operating System
BFS	Breadth-first search
LKF	Linear Kalman Filter
EKF	Extended Kalman Filter
GPS	Global Positioning System

Table 2: Lists of abbreviations