TALLINN UNIVERSITY OF TECHNOLOGY
SCHOOL OF ENGINEERING

Department of Electrical Power Engineering and Mechatronics

The Implementation of the Algorithm for Constructing Maps and Automatic Navigation Using an Autonomous Mobile Wheeled Platform

MASTER THESIS

MECHATRONICS PROGRAM

Student	Anton Isakov
Student code	163841
Supervisor	Robert Hudjakov
	Svetlana Perepelkina

Tallinn, 2017

AUTHOR’S DECLARATION

Hereby I declare, that I have written this thesis independently.

No academic degree has been applied for based on this material.

All works, major viewpoints and data of the other authors used in this thesis have been referenced.

Thesis is completed under the supervision of Robert Hudjakov and Svetlana Perepelkina

“.......” .... 201…..

Author: ....

/signature /

Thesis is in accordance with terms and requirements

“.......” .... 201….

Supervisor: …..........................

/signature/

Accepted for defence

“.......”....................201… .

Chairman of .... theses’ defence commission: ....

/signature/

TTU

School of Engineering

THESIS TASK

Student	Anton Isakov, 163841MV
Study programme, main speciality:	MAHM Mechatronics
Supervisor:	PhD, Robert Hudjakov (TUT)
Supervisor:	PhD, Svetlana Perepelkina (ITMO)

Thesis topic:

(in English) The implementation of the algorithm for constructing maps and automatic navigation using an autonomous mobile wheeled platform.

(in Estonian) Kaardi loomise ja autonoomse navigeerimise algoritmi teostus mobiilsele ratastega platformile.

Thesis proposal

1. Introduction

Nowadays the technical progress in microcontrollers, cameras and algorithms widely opened the way for multiple devices to be able to calculate the data coming from the sensors allowing them to create maps of the surrounding environment and to detect their own position on them. Lots of technologies were created to solve the problem called “SLAM”, which stands for “Simultaneous Localization and Mapping” – the computational problem of constructing or updating a map of the unknown environment while simultaneously keeping track of an device’s location within it. Needless to say that those technologies are widely used in the world, for example, in small vacuum-cleaner robots, hospital and guard robots, even autonomous cars – lots of major automotive manufactures like Volvo, Ford and Tesla are testing their driverless car systems – autonomous ships and aircraft. [1][2][3]. Even the rovers like Spirit and Opportunity are able to create a map of the unknown planet and plan their way to the point that the developers told them to go. The last, but not the least important thing – the robots using SLAM technologies are able to successfully operate in the areas that are dangerous for the humans – in flooded areas, irradiated and burning buildings, under collapsed roofs of the caves and in places, which humans still cannot reach, like other planets and satellites.

My inspiration is that with current technologies and rapid development in microcontrollers and sensors almost anyone is able to find the solution to the problem of SLAM and create the platform that is cheap, mobile, powerful, and is able to monitor the environment around it and calculate its position based on the obtained data from multiple non-industrial grade sensors like RGB-D (Red-Green-Blue-Depth) cameras, sonars, infrared detectors etc. with reasonable quality and cost. Therefore, in this master thesis I am going to design and build a mobile platform, equip it with the

Microsoft Kinect camera and use it to implement and compare a couple of SLAM algorithms. The thesis will present the modifications to the existing open source algorithms so they could be able to make use of the measurements made by the compass and gyroscope module, along with tactile feedback sensors and sonar sensors. Both the SLAM algorithms and the control program will be implemented in C++, using Robot Operating System framework (ROS) to handle compiling, installing, saving and comparing sets of data gained from all the algorithms.

2. **Background**

The current understanding of the problem is that most algorithms require lots of processing power and various high-end sensors like LIDAR (Light Identification Detection and Ranging), which are very precise but expensive to the point that only big groups and companies could afford to use them. Less algorithms tend to use only cameras, both standard cameras we have in our smartphones and RGB-D cameras, because they offer the most flexible and accurate sensing for localization and mapping while still being low-cost and compact. Even less algorithms combine data from different kinds of sensors to get a better result. [4][5][6][7]

While most of the works are explaining their observations and solutions on the SLAM problem, they often do not create any kind of platform for the camera and use their own hands to move it around the building. From my opinion these solutions are very ineffective, because for obtaining more observations of the environment around the camera it should be able to move around freely, exploring the unknown areas and map them and update information about the already mapped ones.

3. **Methodology**

First, I need to check and compare the hardware and software I will need to use in my project and check them for compatibility issues, and also research the current algorithms, their possibilities and drawbacks, and choose a few of them for implementation. Second, I will design, test, print and build a platform keeping in mind the motors, microcontrollers, sensors and battery supply and their positions on the platform. Third part is setting up the algorithms on the platform and making the tests to see which one works better in which conditions. All the algorithms would be tested in two different environments – the standard room full of furniture and the corridor with opened and unopened doors and occasional passers-by. The success of the research would be the comparison between the maps different algorithms provide, their computation speed and hardware usage, and the differences in setup difficulties.

4. **Research Schedule**

№	Description	Completion date
1	Make a research on current SLAM solutions and sensors; Choose sensors and algorithms for implementation	27.12.2016
2	Select the hardware for the mobile platform	30.01.2017

3	Design the mobile platform based on the selected hardware; Print platform	28.02.2017
4	Set up the hardware on the platform; Install all the recommended software; Check that the setup is correct.	14.03.2017
5	Implement and test selected SLAM algorithms on the chosen scenarios; Make comparisons; Choose the best algorithm in every scenario	01.05.2017
6	Finish composing the thesis and presentation	15.05.2017

5. **References**

1. “Autonomous Driving | Intellisafe | Volvo Cars” [Online]. Available: http://www.volvocars.com/intl/about/our-innovation-brands/intellisafe/autonomous-driving [Accessed: March 2017]
2. “FORD TARGETS FULLY AUTONOMOUS VEHICLE FOR RIDE SHARING IN 2021” [Online]. Available: https://media.ford.com/content/fordmedia/fna/us/en/news/2016/08/16/ford-targets-fully-autonomous-vehicle-for-ride-sharing-in-2021.html [Accessed: March 2017]
3. “Autopilot | Tesla” [Online]. Available: https://www.tesla.com/autopilot [Accessed: March 2017]
4. Robert Tubman, Johan Potgieter, Khalid Mahmood Arif, 2016, “Efficient Robotic SLAM by Fusion of RatSLAM and RGBD-SLAM”, School of Engineering and Advanced Technology, Massey University, Auckland, New Zealand
5. Andrew Davison, “The History and Future of Visual SLAM - Keynotes”, Imperial College, London
6. “Google Releases LiDAR SLAM Algorithms, Teases Innovative Mapping Solution” [Online]. Available: http://www.spar3d.com/news/software/google-releases-lidar-slam-algorithms-teases-innovative-mapping-solution/ [Accessed: March 2017]
7. Felix Endres, Jürgen Hess, Nikolas Engelhard, Jürgen Sturm, Daniel Cremers, Wolfram Burgard, 2012, “An Evaluation of the RGB-D SLAM System”, 2012 IEEE International Conference on Robotics and Automation, RiverCentre, Saint Paul, Minnesota, USA, May 14-18, 2012

Student: ……………………...... .... “.......”.................... 201….
/signature/

Supervisor: ……………………. .... “.......”. .... 201….
/signature/

TABLE OF CONTENTS

FOREWORD

The main idea of building a platform and make it run by itself was following me for a long period. Ever since I was a kid I always liked to play and work with mechanisms and technical equipment – building robots and cars from LEGO, helping my father in repairing home devices and ever upgrading some of the toys I had to be better at some characteristics, like making the RC car accelerate much faster by short-circuiting some specific pins on the motherboard or designing and creating the RC Boat for participating in competition. Those tendencies grew in me, and when it was the time to attend university, I already knew that the Mechatronics is the only way that will help me in my ambitions.

The topic is the continuation of my Bachelor degree thesis, in which I was modifying my old RC off-road truck to get the hold of the programming, wiring and control techniques, and to implement a basic Visual SLAM algorithm utilizing one small camera to get information of the environment. The new solution is way faster and accurate and is designed and built from scratch based on the problems I had experienced before.

I want to express my gratitude to the Mechatronics and Robotics Department of ITMO for teaching me and helping me through the studying years, MT.Lab for providing me with necessary equipment and personally to Dmitry Kupriyanov, who opened the world of microcontrollers for me and lit my curiosity for studying them. I would also like to thank the head of TUT Mechatronics Department Professor Mart Tamre and the head of ITMO Mechatronics Department Ph.D. Yuri Monakhov for giving me the opportunity to study in Mechatronics Double Degree program and Robert Hudjakov and Svetlana Perepelkina for helping me with writing this thesis.

EESSÕNA

Põhieesmärk platvormi ehitamisel ja selle autonoomseks tegemisel oli panna ta mind pikaks ajaks järgima. Ma olen lapsest saati armastanud mängida ja töötada mehhanismide ja tehniliste seadmetega - ehitada roboteid ja autosid LEGO klotsidest, aidata isal parandada koduseadmeid ja isegi arendada oma mänguasju paremaks mõnes aspektis nagu näiteks parandades puldiga auto kiirendust lühistades emaplaadil mõned teatud jalad või konstrueerides puldiga paate võistluseesmärgil. Need huvid on minus aja jooksul tugevnenud ning kui aeg oli minna ülikooli, siis ma juba teadsin, et mehhatroonika on ainus viis mu ambitsioone tõeks teha. Käesolev teema on jätk minu bakalaureusetööle, milles ma kohendasin oma vana puldiga maastikuautod, õppimaks programmeerimist, kaabeldamist ja juhtimistehnikaid, eesmärgiga korjata ümbruskonnast kaamera vahendusel informatsiooni loomaks Visual SLAM algoritmi abil kaart. Käesolevas töös loodud lahendus on kiirem ja täpsem ning on loodud alustades puhtalt lehelt lahedamaks probleeme, mida kohtasin varem.

Ma soovin tänada ITMO Mehhatroonika ja Robootika instituuti minu õpetamise ja abistamise eest möödunud õppeaastate jooksul. MT laborit minu varustamise eest ning personaalselt Dmitry Kupriyanovit, kes avas mulle mikrokontrollerite maailma ning süütas minu uudishimu nende vastu. Lisaks soovin tänada TTÜ Mehhantroonika instituudi direktorit Professor Mart Tamret ja ITMO Mehhatroonika instituudi direktorit Ph.D. Yuri Monakhovit võimaluse eest õppida Mehhatroonika topelkraadiõppe programmis ning Robert Hudjakovi ja Svetlana Perepelkinat abi eest selle lõputöö kirjutamisel.

LIST OF ABBREVIATIONS

ROS – Robot Operating System

SLAM – Simultaneous localization and mapping

GPIO – General Purpose Input and Output

SSD – Sum of Squared Differences

SIFT - Scale-Invariant Feature Transform

SURF - Speeded Up Robust Feature

NARF - Normal Aligned Radial Feature

BRIEF - Binary Robust Independent Elementary Feature

FAST - Features from Accelerated Segment Test

CHAPTER 1

1. INTRODUCTION

This thesis was created as a part of MSc Mechatronics Double Degree program between the Tallinn Technical University and Saint-Petersburg ITMO University. The topic is a continuation of the Bachelor thesis, and the main objective is to create a robust solution to the SLAM problem, which should be cheaper, easier to set up and use than existing solutions.

This work aims for completion of the following tasks:

1. Design and build a mobile platform
2. Create a control program
3. Implement four SLAM algorithms and compare them

The second chapter explains the primary technology behind the solution, describes and justifies the choices in hardware and software.

The third chapter gives an overview on the development stage of the project, showing how to install, setup and utilize the ROS framework and the SLAM algorithms, provides information about the control programs and algorithms behind the operation of the platform and shows the steps of designing the prototype of the platform.

The fourth chapter shows the results of the work –the comparison between the selected algorithms.

CHAPTER 2

2. DESIGN

2.1. SLAM and Visual SLAM

One of the most basic yet important features of intelligent mobile robots is the ability to navigate autonomously. Autonomous robots are capable of safely exploring their surroundings without colliding with obstacles. For navigation in the unknown environment, the robot should build a map of its surroundings and be able to know its position on the generated map at the same time. [1]

SLAM – Simultaneous Localization and Mapping – is a term for algorithms that build maps of an existing unknown environment while being able to perform localization in that area. This method is widely used in robotics and often plays the main role in creation of the autonomous robot. It allows robots and autonomous vehicles to build a map of the environment or to update a preexisting map with new information while constantly keeping track of their position on such map. The method of SLAM allows to combine two independent processes – navigation and mapping – into a continuous cycle of consistent calculations, in which the result of one process is an input information for the other. This way the trajectory of the robotic platform and the mapping information about the surrounding area is estimated in real time without the need for any present knowledge of the location. [2] [3] A solution to a SLAM problem would allow robots to make maps without any human assistance.

2.1.1. History of SLAM

The researches in SLAM area were conducted in multiple universities and other institutions since 1986, where the origins of SLAM problem were presented at the 1986 IEEE Robotics and Automation Conference that took place in San Francisco. The work that started the researches on SLAM technology was the "On the Representation and Estimation of Spatial Uncertainty" made in 1986 by R.C. Smith and P. Cheeseman [4]. In this paper the method for estimating the nominal relationship and expecting error between coordinate frames that represent the relative locations of objects was described. The method could be used to answer the such questions as whether a camera attached to a robot platform is likely to have a particular reference object in its field of view. The method made it possible to decide in advance whether an uncertain relationship is known accurately enough for a required task and if not – how much of an improvement in locational knowledge a proposed sensor will provide.

Next work in this field of research was "Estimating Uncertain Spatial Relationships in Robotics" by R.C. Smith and P. Cheeseman in 1990 [5]. It described the representation for spatial information, called “stochastic map”, and included the methods of creating a map, extracting information from sensors and revising it as new information is collected. The map they presented contained the estimates of the relationship between the objects on the map and their uncertainties, given all the available information. As an advance over the previous approaches on the problem was the fact that the estimates were probabilistic by the nature, and the development of the procedures in the state-estimation and filtering theory provided a solid basis for the followed extensions.

The “Simultaneous Map Building and Localization for an Autonomous Mobile Robot” (1991) by J.J. Leonard and H.F. Durrant-Whyte [6] discussed the problem of estimation of the robot position without a priori information. That problem was difficult because of the conflict between the main processes of SLAM – localization and mapping. For precision moving the mobile robot should have an accurate representation of the environment, but for building an accurate map the robot’s location should be known precisely. In this paper the SLAM problem was presented as “chicken-egg” problem, means that the two processes are working recursively, and the first one’s output is a next one’s input. For overcoming the issue the authors used an array of ultrasonic sensors mounted on the servo motors of the robot, which gave them an ability to sense the accurate information about the robot’s surroundings from the very beginning and to have a correct tracking of the extracted features to provide precise positioning.

A conceptual break-through came when the problem of simultaneous navigation and localization, which was first described as estimated, was found to be convergent. The relations between the points on the map, which a lot of researches tried to decrease, played a major role in the solution of the problem, and the more relations there were the more accurate solution was generated. Most of the theory of convergence and multiple results were achieved by M. Csorba in the works “A new approach to simultaneous localization and map building” [7] and “Simultaneous Localization and Map Building” [8], which were published in 1996 and 1997 respectively. It was shown that correlations arise from the errors in the vehicle and the map estimates; these correlations were identified as fundamentally important to the solution of the SLAM problem, and ignoring these correlations lead to inconsistencies in map generation and position estimation. The provided results show that it is possible to start the robot in the unknown location in an unknown environment and be able to build a map by which to navigate.

2.1.2. Visual SLAM

In the last couple of decades the area of the mobile robotics and autonomous platforms has attracted significant attention from researchers all around the world, which resulted in multiple breakthroughs and technological advances. Mobile robots are able to perform complicated tasks autonomously, while in the past they required assistance from human personnel. The range of applications contains various fields, such as medical, military and domestic. In those applications mobile robots are required to accomplish complex tasks requiring navigation in an unknown, complex indoor and outdoor environments without any human input. As a result, the SLAM problem was studied in detail and various techniques have been proposed to solve the localization problem.

The research on SLAM grew heavily for the last decade, with multiple institutions developing new algorithms, sensors and solutions. The continuous improvement of the computation capabilities of microcontrollers and the quality of sensors have made it possible for SLAM applications to be used outside of the pure test environments, and nowadays they are able to work even outdoors. With the major progress in algorithms, multiple sensors could be used to solve the SLAM problem.

1. Mono-camera. The algorithms like PTAM, VSLAM and others work with the video stream from the single camera that moves relative to the environment, detecting keypoints between the video frames and trying to link them one to another. The performance and the quality of the obtained map is not so good, but it depends on the algorithm.

2. Stereo camera. The algorithms work with a pair of cameras with known parameters, good calibration and the distance between them. Other work is similar to the mono-camera algorithms – they detect keypoints on the images and link them between cameras and between consecutive frames. Usage of two cameras provides them with better results at detecting and tracking keypoints.

3. Multi-camera. Same as #2, but they use a matrix of cameras for detecting points. Very computer performance-hungry, but good results.

4. RGBD cameras and LIDARs. They provide the system with the point clouds eliminating the need for detecting them on the frames. RGBD (Red-Green-Blue-Depth) cameras give out the video stream and the point cloud in front of the camera and cost not so much, while LIDARs give out a point cloud of the full terrain around the robot, work much faster and deliver much more information than any of the above, but have high prices for a middle-end solution.

In every implementation of SLAM the first thing is to create a map of the environment around the robot. The most easiest and cheapest way to do so is to use optic sensors.

Visual SLAM implies the usage of an array of cameras or a LIDAR to detect the robot’s location with help from the odometry and accelerometers installed. LIDARs are more preferable due to their rapid data obtainment, precise distance information and very high angle of sight (typically full 360 degree view), but the severe con of their typical price makes them less preferable for most of the projects.

On the other side, using cameras we can obtain much more information about the robot’s surroundings than any other type of sensor. With RGB-D cameras it is possible to receive data about thousands points around the robot and with the usage of specialized algorithms to compare these points to the database, understanding the camera’s location, the size and the shape of the object in front of it, and also the material and color.

There are also the combined systems that unite best parameters of cameras and LIDARs, but due to their complexity and cost, using these systems is only practical in high-quality projects, where the extreme precision is most necessary. In most cases the prototypes of such systems could not be moved to mass production, which forces the developers into searching for different variants of the optical systems.

One of the possible solutions includes using monocular vision, because it saves most of the advantages of the cameras while keeping the production at a satisfactory cost. But to the problem of analyzing the video stream adds another, most severe disadvantage of Visual SLAM – necessity of searching for correlations in obtained images. In most of the cases modern systems collect data using multiple algorithms and present the maps as a set of points, lines or simple geometrical objects. This data could help in viewing the surface around the robot as a coarse sketch.

Techniques that use stereo vision could deliver the high-quality information due to the accurate measurements and precise calibration of the cameras. But despite all of the benefits of this solution it did not got wide spread in the Visual SLAM projects. The complexity of calculations required for computation of geometric parameters of the environment become the practical limit of such technics for obtaining real-time models in SLAM.

In my thesis I decided to use the Visual SLAM solution that implies the usage of RGB-D camera to get the information about the robot platform’s environment.

2.1.3. SLAM Problem definition

The problem of consistently matching (aligning) various 3D point clouds taken from different point of views into a complete model is known as registration. The goal of this process is to find the relative positions and orientations of the separately acquired point clouds in a global coordinate framework. The key idea to perform this task is to identify corresponding points between the data sets then find a transformation that minimizes the distance (alignment error) between the corresponding points. The process is repeated until the alignment errors falls below a given threshold: at this point the registration is said to be complete.

To perform point clouds matching, usually the following steps are used:

1. From the two consecutive noisy point clouds that we want to match, keypoints that best represent the scene in both set of points are extracted.

2. For each keypoint, a feature descriptor is computed.

3. Correspondences between the extracted features in both point clouds are estimated using the feature descriptors and their XYZ positions in the datasets.

4. The point clouds are assumed to be noisy and not all correspondences are valid, so bad correspondences that contribute negatively to the registration process are rejected.

5. From the remaining set of good correspondences, an initial rough transformation between the two point clouds is estimated.

6. Refinement of the matching between the point clouds is performed using the ICP (Iterative Closest Point) or the NDT (Normal Distribution Transform) algorithms.

Generally speaking, a keypoint (also known as “interest point”) is simply a point that has been identified as relevant in some way. Whether any point of the point cloud is considered as keypoint or not depends on the used “keypoint detector”. Raw point clouds extracted from a stereo camera system are usually noisy and the feature descriptors needed for the registration process are expensive to compute at every point. This is why keypoints are usually used to identify a small number of locations where computing feature descriptors is likely to be most effective.

There is no strict definition for what constitutes a keypoint detector, but a good keypoint detector will find points which have the following properties:

1. Sparseness: Typically, only a small subset of points in the scene are keypoints
2. Repeatability: If a point is determined to be a keypoint in one point cloud, a keypoint should also be found in a second point cloud taken from a different view point. Such keypoint will be called “Stable”.
3. Distinctiveness: The area surrounding each keypoint should have a unique shape or appearance that can be captured by some feature descriptor.

Interest points are usually placed at corners of shapes and where the color/brightness gradient is the highest. There are various methods to find keypoints, and each technique has its specific output. [9]

2.2. Algorithms

2.2.1. SLAM Process

A SLAM algorithm essentially consists of the following steps:

1. Data acquisition; in this step measurements from the sensors, e.g. laser scanner or video camera, are gathered.
2. Feature extraction; a number of characteristic, and thereby easily recognizable, landmarks are selected from the data set.
3. Feature association; landmarks from previous measurements are associated with landmarks from the most recent measurement.
4. Pose estimation; the relative change between the landmarks and the position of the vehicle is used to estimate the new pose of the vehicle.

5. Map adjustment; the map is updated according to the new pose and the corresponding measurements.

The five tasks are continuously repeated and a trajectory of position estimates and a map is built up. In the case of visual SLAM, a landmark can be anything that is easily recognizable by a visual sensor, e.g.

- a corner,
- an edge,
- a dot in a protruding color.

The robot extracts landmarks from the data and searches through its database to see if there are any matches with old landmarks. Extracted landmarks that are not found in the database are added and landmarks giving a match in the database are used to estimate the change of the robot's pose. This is done by measuring the change in distance and angle to the old landmarks. When the new pose is estimated the robot uses this estimate and the measurements to adjust the positions of the landmarks. SLAM can be regarded as a hen and egg problem. A proper map is needed to get a proper pose estimate and a proper pose estimate is needed to get a proper map. [10]

2.2.2. Feature detection

In computer vision, and more specifically in object recognition, many techniques are based on the detection of points of interests on object or surfaces. This is done through the extraction of features. In order to track these points of interests during a motion of the camera and/or the robot, a reliable feature has to be invariant to image location, scale and rotation. A few methods are briefly presented here:

- Moravec Corner detection algorithm [11]
- Harris and Stephens Corner detection algorithm [12]
- SIFT - Scalar Invariant Feature Transform, by David Lowe [13]
- SURF - Speeded Up Robust Feature [14]
- NARF - Normal Aligned Radial Feature [15]
- BRIEF - Binary Robust Independent Elementary Feature [16]
- FAST - Features from accelerated segment test [17]

There are two aspects concerning a feature: the detection of a keypoint, which identifies an area of interest, and its descriptor, which characterizes its region. Typically, the detector identifies a region containing a strong variation of intensity such as an edge or a corner, and its center is designed as a keypoint. The descriptor is generally computed by measuring the main orientations of the surrounding points, leading to a multidimensional feature vector which identifies the given keypoint.

Given a set of features, a matching can then be performed in order to associate some pairs of keypoints between a couple of frames.

Moravec Corner detection algorithm

One of the earliest corner detection algorithm. It defines a corner as a point with low self-similarity. The algorithm tests each pixel in the image to see if a corner is present, by considering how similar a patch centered on the pixel is to nearby, largely overlapping patches. The similarity is measured by taking the sum of squared differences between the corresponding pixels of two patches. A lower number indicates more similarity.

If the pixel is in a region of uniform intensity, then the nearby patches will look similar. If the pixel is on an edge, then nearby patches in a direction perpendicular to the edge will look quite different, but nearby patches in a direction parallel to the edge will result in only in a small change. If the pixel is on a feature with variation in all directions, then none of the nearby patches will look similar.

The corner strength is defined as the smallest SSD (Sum of squared differences) between the patch and its neighbours (horizontal, vertical and on the two diagonals). The reason is that if this number is high, then the variation along all shifts is either equal to it or larger than it, so capturing that all nearby patches look different.

If the corner strength number is computed for all locations, that it is locally maximal for one location indicates that a feature of interest is present in it.

As pointed out by Moravec, one of the main problems with this operator is that it is not isotropic: if an edge is present that is not in the direction of the neighbours (horizontal, vertical, or diagonal), then the smallest SSD will be large and the edge will be incorrectly chosen as an interest point. [11]

Harris Corner

Known as the Harris corner operator, this is one of the earliest detector, as it was proposed in 1988 by Harris and Stephens [12] as an improved version over Moravec's. It considers the differential of the corner score with respect to direction directly, instead of using shifted patches. The notion of corner should be taken in a wide sense as it allows to detect not only corners, but edges and more generally, keypoints. It is done by computing the second moment matrix (or auto-correlation matrix) of the image intensities, describing its local variations. One of the main limitation with the Harris operator, at least in its original version, concerns the scale invariance as the matrix should be recomputed for a different scale. [18]

Without loss of generality, we will assume a grayscale 2-dimensional image is used. Let this image be given by I. Consider taking an image patch over the area (u, v) and shifting it by (x, y). The weighted sum of squared differences (SSD) between these two patches, denoted S, is given by:

$$S(x,y) = \sum_u \sum_v w(u,v)\big(I(u+x, v+y) - I(u,v)\big)^2 \tag{1.1}$$

$I(u+x, v+y)$ can be approximated by a Taylor expansion [19]. Let I_x and I_y be the partial derivatives of I, such that

$$I(u+x, v+y) \approx I(u,v) + I_x(u,v)x + I_y(u,v)y \tag{1.2}$$

This produces the approximation

$$S(x,y) \approx \sum_u \sum_v w(u,v)\big(I_x(u,v)x + I_y(u,v)y\big)^2 \tag{1.3}$$

Which can be written in matrix form

$$S(x,y) \approx (x \quad y)A\binom{x}{y} \tag{1.4}$$

Where A is the structure tensor,

$$A = \sum_u \sum_v w(u,v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = \begin{bmatrix} \langle I_x^2 \rangle & \langle I_x I_y \rangle \\ \langle I_x I_y \rangle & \langle I_y^2 \rangle \end{bmatrix} \tag{1.5}$$

This matrix is a Harris matrix, and angle brackets denote summation over (u, v). If a circular window $w(u, v)$ is used, then the response will be isotropic.

A corner (or in general an interest point) is characterized by a large variation of S in all directions of the vector $(x \quad y)$. By analyzing the eigenvalues of A, this characterization can be expressed in the following way: A should have two "large" eigenvalues for an interest point. Based on the magnitudes of the eigenvalues, the following inferences can be made based on this argument:

1. If $\lambda_1 \approx 0$ and $\lambda_2 \approx 0$ then this pixel (x, y) has no features of interest.
2. If $\lambda_1 \approx 0$ and λ_2 has some large positive value, then an edge is found.
3. If λ_1 and λ_2 have large positive values, then a corner is found.

Harris and Stephens note that exact computation of the eigenvalues is computationally expensive, since it requires the computation of a square root, and instead suggest the following function M_c, where κ is a tunable sensitivity parameter:

$$M_c = \lambda_1 \lambda_2 - \kappa(\lambda_1 + \lambda_2)^2 = \det(A) - \kappa \, trace^2(A) \tag{1.6}$$

Therefore, the algorithm does not have to actually compute the eigenvalue decomposition of the matrix A and instead it is sufficient to evaluate the determinant and trace of A to find corners, or rather interest points in general.

The Shi–Tomasi [20] [21] corner detector directly computes $\min(\lambda_1, \lambda_2)$ because under certain assumptions, the corners are more stable for tracking. This method is also sometimes referred to as the Kanade-Tomasi corner detector.

The value of κ has to be determined empirically, and in the literature values in the range 0.04–0.15 have been reported as feasible. There is a possibility of avoiding setting the parameter κ by using Noble's [22] corner measure M_c' which amounts to the harmonic mean of the eigenvalues:

$$M_c' = 2\frac{\det(A)}{trace(A) + \epsilon'} \tag{1.7}$$

ϵ being a small positive constant,

The covariance matrix for the corner position is A^{-1}, i.e.

$$\frac{1}{\langle I_x^2\rangle\langle I_y^2\rangle - \langle I_x I_y\rangle^2}\begin{bmatrix} \langle I_y^2\rangle & -\langle I_x I_y\rangle \\ -\langle I_x I_y\rangle & \langle I_x^2\rangle \end{bmatrix} \tag{1.8}$$

SIFT

The Scalar Invariant Feature Transform (SIFT) is a method presented by David Lowe [13], now widely used in robotics and computer vision. This is a method to detect distinctive, invariant image feature points, which easily can be matched between images to perform tasks such as object detection and recognition, or to compute geometrical transformations between images.

The main idea of the SIFT method is to define a cascade of operations following an increasing complexity, so that the most expensive operations are only performed to the most probable candidates.

1. Scale-invariant feature detection.

The first step relies on a pyramid of Difference-of-Gaussian (DoG) in order to be invariant to scale and orientation. Lowe's method for image feature generation transforms an image into a large collection of feature vectors, each of which is invariant to image translation, scaling, and rotation, partially invariant to illumination changes and robust to local geometric distortion. These features share similar properties with neurons in primary Visual cortex that are encoding basic forms, color and movement for object detection in primate vision. [23] Key locations are defined as maxima and minima of the result of difference of Gaussians function applied in scale space to a series of smoothed and resampled images. Low contrast candidate points and edge response points along an edge are discarded. Dominant orientations are assigned to localized keypoints. These steps ensure that the keypoints are more stable for matching and recognition. SIFT descriptors robust to local affine distortion are then obtained by considering pixels around a radius of the key location, blurring and resampling of local image orientation planes.

2. Feature matching and indexing

Indexing consists of storing SIFT keys and identifying matching keys from the new image. Lowe used a modification of the k-d tree algorithm called the Best-bin-first search method [24] that can identify the nearest neighbors with high probability using only a limited amount of computation. The BBF algorithm uses a modified search ordering for the k-d tree algorithm so that bins in feature space are searched in the order of their closest distance from the query location. This search order requires the use of a heap-based priority queue for efficient determination of the search order. The best candidate match for each keypoint is found by identifying its nearest neighbor in the database of keypoints from training images. The nearest neighbors are defined as the keypoints with minimum Euclidean distance from the given descriptor vector. The probability that a match is correct can be determined by taking the ratio of distance from the closest neighbor to the distance of the second closest.

Lowe rejected all matches in which the distance ratio is greater than 0.8, which eliminates 90% of the false matches while discarding less than 5% of the correct matches. To further improve the efficiency of the best-bin-first algorithm search was cut off after checking the first 200 nearest neighbor candidates. For a database of 100,000 keypoints, this provides a speedup over exact nearest neighbor search by about 2 orders of magnitude, yet results in less than a 5% loss in the number of correct matches. [25]

3. Cluster identification by Hough transform voting

Hough Transform is used to cluster reliable model hypotheses to search for keys that agree upon a particular model pose. Hough transform identifies clusters of features with a consistent interpretation by using each feature to vote for all object poses that are consistent with the feature. When clusters of features are found to vote for the same pose of an object, the probability of the interpretation being correct is much higher than for any single feature. An entry in a hash table is created predicting the model location, orientation, and scale from the match hypothesis. The hash table is searched to identify all clusters of at least 3 entries in a bin, and the bins are sorted into decreasing order of size.

Each of the SIFT keypoints specifies 2D location, scale, and orientation, and each matched keypoint in the database has a record of its parameters relative to the training image in which it was found. The similarity transform implied by these 4 parameters is only an approximation to the full 6 degree-of-freedom pose space for a 3D object and also does not account for any non-rigid deformations. Therefore, Lowe used broad bin sizes of 30 degrees for orientation, a factor of 2 for scale, and 0.25 times the maximum projected training image dimension (using the predicted scale) for location. The SIFT key samples generated at the larger scale are given twice the weight of those at the smaller scale. This means that the larger scale is in effect able to filter the most likely neighbours for checking at the smaller scale. This also improves recognition performance by giving more weight to the least-noisy scale. To avoid the problem of boundary effects in bin assignment, each keypoint match votes for the 2 closest bins in each dimension, giving a total of 16 entries for each hypothesis and further broadening the pose range.

4. Model verification by linear least squares

Each identified cluster is then subject to a verification procedure in which a linear least squares solution is performed for the parameters of the affine transformation relating the model to the image. The affine transformation of a model point $[x \quad y]^T$ to an image point $[u \quad v]^T$ can be written as below:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} m1 & m2 \\ m3 & m4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} tx \\ ty \end{bmatrix} \tag{1.9}$$

Where the model translation is $[tx \quad ty]^T$ and the affine rotation, scale, and stretch are represented by the parameters m1, m2, m3 and m4. To solve for the transformation parameters the equation above can be rewritten to gather the unknowns into a column vector.

$$\begin{bmatrix} x & y & 0 & 0 & 1 & 0 \\ 0 & 0 & x & y & 0 & 1 \\ \ldots & & & & & \end{bmatrix} \begin{bmatrix} m1 \\ m2 \\ m3 \\ m4 \\ tx \\ ty \end{bmatrix} = \begin{bmatrix} u \\ v \\ . \\ . \end{bmatrix} \tag{1.10}$$

This equation shows a single match, but any number of further matches can be added, with each match contributing two more rows to the first and last matrix. At least 3 matches are needed to provide a solution. This linear system could be written as

$$A\hat{x} \approx b \tag{1.11}$$

Where A is a known m-by-n matrix (usually with m > n), x is an unknown n-dimensional parameter vector, and b is a known m-dimensional measurement vector.

Therefore, the minimizing vector $\hat{x}$ is a solution of the normal equation

$$A^T A\hat{x} = A^T b \tag{1.12}$$

The solution of the system of linear equations is given in terms of the matrix $(A^T A)^{-1} A^T$, called the pseudoinverse of A, by

$$\hat{x} = (A^T A)^{-1} A^T b \tag{1.13}$$

which minimizes the sum of the squares of the distances from the projected model locations to the corresponding image locations.

5. Outlier detection

Outliers can now be removed by checking for agreement between each image feature and the model, given the parameter solution. Given the linear least squares solution, each match is required to agree within half the error range that was used for the parameters in the Hough transform bins. As outliers are discarded, the linear least squares solution is re-solved with the remaining points, and the process iterated. If fewer than 3 points remain after discarding outliers, then the match is rejected. In addition, a top-down matching phase is used to add any further matches that agree with the projected model position, which may have been missed from the Hough transform bin due to the similarity transform approximation or other errors.

The final decision to accept or reject a model hypothesis is based on a detailed probabilistic model. [26] This method first computes the expected number of false matches to the model pose, given the projected size of the model, the number of features within the region, and the accuracy of the fit. A Bayesian probability analysis then gives the probability that the object is present based on the actual number of matching features found. A model is accepted if the final probability for a correct interpretation is greater than 0.98. Lowe's SIFT based object recognition gives excellent results except under wide illumination variations and under non-rigid transformations. [27]

SURF

The Speeded Up Robust Feature (SURF) provides a robust detector and descriptor [14], that can be used in computer vision tasks like object recognition or 3D reconstruction. It is partly inspired by the SIFT descriptor, both are using local gradient histograms. The main difference concerns the performance, lowering the computational time through an efficient use of integral images for the image convolutions, Hessian matrix-based detector (optimized through approximations of the second order Gaussian partial derivatives), and sums of approximated 2D Haar wavelet responses for the descriptor. The standard version of SURF is several times faster than SIFT and claimed by its authors to be more robust against different image transformations than SIFT. [28]

The algorithm has three main parts: interest point detection, local neighborhood description and matching.

1. Detection.

SURF uses square-shaped filters as an approximation of Gaussian smoothing. (The SIFT approach uses cascaded filters to detect scale-invariant characteristic points, where the difference of Gaussians (DoG) is calculated on rescaled images progressively.) Filtering the image with a square is much faster if the integral image is used:

$$S(x,y)=\sum_{i=0}^{x}\sum_{j=0}^{y}I(i,j) \tag{1.14}$$

The sum of the original image within a rectangle can be evaluated quickly using the integral image, requiring evaluations at the rectangle's four corners.

SURF uses a blob detector based on the Hessian matrix to find points of interest. The determinant of the Hessian matrix is used as a measure of local change around the point and points are chosen where this determinant is maximal. In contrast to the Hessian-Laplacian detector by Mikolajczyk and Schmid [28], SURF also uses the determinant of the Hessian for selecting the scale, as is also done by Lindeberg [29]. Given a point $p=(x,y)$ in an image I, the Hessian matrix $H(p,\sigma)$ at point p and scale σ, is:

$$H(p,\sigma)=\begin{pmatrix} L_{xx}(p,\sigma) & L_{xy}(p,\sigma) \\ L_{yx}(p,\sigma) & L_{yy}(p,\sigma) \end{pmatrix} \tag{1.15}$$

Where $L_{xx}(p,\sigma)$ etc. is the convolution of the second-order derivative of gaussian with the image $I(x,y)$ at the point x.

The box filter of size 9×9 is an approximation of a Gaussian with σ=1.2 and represents the lowest level (highest spatial resolution) for blob-response maps. [30]

2. Scale-space representation and location of points of interest

Interest points can be found at different scales, partly because the search for correspondences often requires comparison images where they are seen at different scales. In other feature detection algorithms, the scale space is usually realized as an image pyramid. Images are repeatedly smoothed with a Gaussian filter, then they are subsampled to get the next higher level of the pyramid. Therefore, several floors or stairs with various measures of the masks are calculated:

$$\sigma_{approx}=Current\ filter\ size * \left(\frac{Base\ Filter\ Scale}{Base\ Filter\ Size}\right) \tag{1.16}$$

The scale space is divided into a number of octaves, where an octave refers to a series of response maps of covering a doubling of scale. In SURF, the lowest level of the scale space is obtained from the output of the 9×9 filters.

Hence, unlike previous methods, scale spaces in SURF are implemented by applying box filters of different sizes. Accordingly, the scale space is analyzed by up-scaling the filter size rather than iteratively reducing the image size. The output of the above 9×9 filter is considered as the initial scale layer at scale s=1.2 (corresponding to Gaussian derivatives with $\sigma=1.2$). The following layers are obtained by filtering the image with gradually bigger masks, taking into account the discrete nature of integral images and the specific filter structure. This results in filters of size 9×9, 15×15, 21×21, 27×27... Non-maximum suppression in a 3×3×3 neighborhood is applied to localize interest points in the image and over scales. The maxima of the determinant of the Hessian matrix are then interpolated in scale and image space with the method proposed by Brown, et al. [31]. Scale space interpolation is especially important in this case, as the difference in scale between the first layers of every octave is relatively large. [30]

3. Descriptor

The goal of a descriptor is to provide a unique and robust description of an image feature, e.g., by describing the intensity distribution of the pixels within the neighbourhood of the point of interest. Most descriptors are thus computed in a local manner, hence a description is obtained for every point of interest identified previously.

The dimensionality of the descriptor has direct impact on both its computational complexity and point-matching robustness/accuracy. A short descriptor may be more robust against appearance variations, but may not offer sufficient discrimination and thus give too many false positives.

The first step consists of fixing a reproducible orientation based on information from a circular region around the interest point. Then we construct a square region aligned to the selected orientation, and extract the SURF descriptor from it. [30]

3.1. Orientation assignment

In order to achieve rotational invariance, the orientation of the point of interest needs to be found. The Haar wavelet responses in both x- and y-directions within a circular neighbourhood of radius $6s$ around the point of interest are computed, where s is the scale at which the point of interest was detected. The obtained responses are weighted by a Gaussian function centered at the point of interest, then plotted as points in a two-dimensional space, with the horizontal response in the abscissa and the vertical response in the ordinate. The dominant orientation is estimated by calculating the sum of all responses within a sliding orientation window of size $\pi/3$. The horizontal and vertical responses within the window are summed. The two summed responses then yield a local orientation vector. The longest such vector overall defines the orientation of the point of interest. The size of the sliding window is a parameter that has to be chosen carefully to achieve a desired balance between robustness and angular resolution. [30]

3.2. Descriptor based on the sum of Haar wavelet responses

To describe the region around the point, a square region is extracted, centered on the interest point and oriented along the orientation as selected above. The size of this window is $20s$.

The interest region is split into smaller 4x4 square sub-regions, and for each one, the Haar wavelet responses are extracted at 5x5 regularly spaced sample points. The responses are weighted with a Gaussian (to offer more robustness for deformations, noise and translation).

4. Matching

By comparing the descriptors obtained from different images, matching pairs can be found.

NARF

The Normal Aligned Radial Feature (NARF) is presented by Bastian Steder in [32]. It is meant to be used on single range scan obtained with 3D laser range finders or stereo camera. This feature is available in the Point Cloud Library (PCL) [33], which is part of the Robot Operating System (ROS), but also released as a standalone library. Its detector looks for stable areas with significant change in vicinity, which can be identified from different viewpoints. The descriptor characterizes the area around the keypoint by calculating a normal aligned range value patch and finding the dominant orientation of the neighboring pixels.

This algorithm extends SIFT's concepts. It works by iterating over all interest points in the range image RI, for every point Pi it creates a small image patch by looking at it along it's normal. The normal is the Z-axis of the image patch's local coordinate system where Pi is at (0,0). The Y-axis is the world coordinate system Y-Axis. The X-axis aligns accordingly. All neighbours within the radius r around Pi are transfered into this local coordinate system.

A star pattern with n beams is projected on the image patch. For each beam a score in [-0.5,0.5] is calculated. Beams have a high score if there are lots of intensity changes in the cells lying under the beam. This is calculated by comparing each cell with the next adjacent one. Additionally cells closer to the center contribute to the score with a higher weight (2 in the middle, 1 at the edge).

Finally the dominant orientation of the patch is calculated to make it invariant against rotations around the normal. [34]

BRIEF

The Binary Robust Independent Elementary Feature (BRIEF) presented in [16] is an efficient alternative for the descriptor, based on binary strings computed directly from image patches, and measures the Hamming distance instead of the L^2 norm commonly used for high dimension descriptors. As the binary comparison can be performed very efficiently, the matching between several candidates can be done much faster. The use of a BRIEF descriptor supposes the keypoints are already known, this can be done with a detector such as SIFT or SURF. A deeper study is available in the PhD thesis of Calonder [32], who is the main author of the BRIEF features. The main interest of this descriptor resides in its performance.

FAST

Features from accelerated segment test (FAST) is a corner detection method, which could be used to extract feature points and later used to track and map objects in many computer vision tasks. FAST corner detector was originally developed by Edward Rosten and Tom Drummond, and published in 2005 [17]. The most promising advantage of the FAST corner detector is its computational efficiency. Referring to its name, it is fast and indeed it is faster than many other well-known feature extraction methods, such as difference of Gaussians (DoG) used by the SIFT, SUSAN and Harris detectors. Moreover, when machine learning techniques are applied, superior performance in terms of computation time and resources can be realised. The FAST corner detector is very suitable for real-time video processing application because of this high-speed performance.

FAST corner detector uses a circle of 16 pixels (a Bresenham circle of radius 3) to classify whether a candidate point p is actually a corner. Each pixel in the circle is labeled from integer number 1 to 16 clockwise. If a set of N contiguous pixels in the circle are all brighter than the intensity of candidate pixel p (denoted by Ip) plus a threshold value t or all darker than the intensity of candidate pixel p minus threshold value t, then p is classified as corner. The conditions can be written as:

Condition 1: A set of N contiguous pixels S, $\forall\, x \in S$, the intensity of x $(Ix) > Ip$ + threshold t

Condition 2: A set of N contiguous pixels S, $\forall\, x \in S, Ix < Ip - t$

So when either of the two conditions is met, candidate p can be classified as a corner. There is a tradeoff of choosing N, the number of contiguous pixels and the threshold value t. On one hand the number of detected corner points should not be too many, on the other hand, the high performance should not be achieved by sacrificing computational efficiency. Without the improvement of machine learning, N is usually chosen as 12. A high-speed test method could be applied to exclude non-corner points. [35]

2.3. SLAM Algorithms

2.3.1. Hector SLAM

Hector_mapping is a SLAM approach that can be used without odometry as well as on platforms that exhibit roll/pitch motion (of the sensor, the platform or both). It leverages the high update rate of modern LIDAR systems like the Hokuyo UTM-30LX and provides 2D pose estimates at scan rate of the sensors (40Hz for the UTM-30LX). While the system does not provide explicit loop closing ability, it is sufficiently accurate for many real world scenarios. The system has successfully been used on Unmanned Ground Robots, Unmanned Surface Vehicles, Handheld Mapping Devices and logged data from quadrotor UAVs. [36]

To be able to represent arbitrary environments an occupancy grid map is used, which is a proven approach for mobile robot localization using LIDARs in real-world environments [37]. As the LIDAR platform might exhibit 6DOF motion, the scan has to be transformed into a local stabilized coordinate frame using the estimated attitude of the LIDAR system. Using the estimated platform orientation and joint values, the scan is converted into a point cloud of scan endpoints. Depending on the scenario, this point cloud can be preprocessed, for example by downsampling the number of points or removal of outliers. For the presented approach, only filtering based on the endpoint z coordinate is used, so that only endpoints within a threshold of the intended scan plane are used in the scan matching process.

1. Map Access

The occupancy grid maps discrete nature limits the precision that could be achieved and also does not allow the direct computation of interpolated values or derivatives. For this reason an interpolation scheme allowing sub-grid cell accuracy through bilinear filtering is employed for both estimating occupancy probabilities and derivatives. Intuitively, the grid map cell values can be viewed as samples of an underlying continuous probability distribution.

Given a continuous map coordinate Pm, the occupancy value $M(Pm)$ as well as the gradient $\Delta M(Pm) = (\frac{\partial M}{\partial x}(Pm), \frac{\partial M}{\partial y}(Pm))$ can be approximated by using the four closest integer coordinates $P_{00.\ .11}$. Linear interpolation along the x- and y-axis then yields

$$\begin{aligned} M(Pm) \approx & \frac{y - y_0}{y_1 - y_0}\left(\frac{x - x_0}{x_1 - x_0} M(P_{11}) + \frac{x_1 - x}{x_1 - x_0} M(P_{01})\right) \\ & + \frac{y_1 - y}{y_1 - y_0}\left(\frac{x - x_0}{x_1 - x_0} M(P_{10}) + \frac{x_1 - x}{x_1 - x_0} M(P_{00})\right) \end{aligned} \tag{1.17}$$

The derivatives can be approximated by:

$$\frac{\partial M}{\partial x}(Pm) \approx \frac{y - y_0}{y_1 - y_0}\big(M(P_{11}) - M(P_{01})\big) + \frac{y_1 - y}{y_1 - y_0}\big(M(P_{10}) - M(P_{00})\big) \tag{1.18}$$

$$\frac{\partial M}{\partial y}(Pm) \approx \frac{x - x_0}{x_1 - x_0}\big(M(P_{11}) - M(P_{01})\big) + \frac{x_1 - x}{x_1 - x_0}\big(M(P_{10}) - M(P_{00})\big) \tag{1.19}$$

It should be noted that the sample points/grid cells of the map are situated on a regular grid with distance 1 (in map coordinates) from each other, which simplifies the presented equations for the gradient approximation.

2. Scan Matching

Scan matching is the process of aligning laser scans with each other or with an existing map. Modern laser scanners have low distance measurement noise and high scan rates. A method for registering scans might yield very accurate results for this reason. For many robot systems the accuracy and precision of the laser scanner is much higher than that of odometry data, if available at all.

The approach is based on optimization of the alignment of beam endpoints with the map learnt so far. The basic idea using a Gauss-Newton approach is inspired by work in computer vision [34]. Using this approach, there is no need for a data association search between beam endpoints or an exhaustive pose search. As scans get aligned with the existing map, the matching is implicitly performed with all preceding scans.

3. Multi-Resolution Map Representation

Any hill climbing/gradient based approach has the inherent risk of getting stuck in local minima. As the presented approach is based on gradient ascent, it also is potentially prone to get stuck in local minima. The problem is mitigated by using a multi-resolution map representation similar to image pyramid approaches used in computer vision. In our approach, we optionally use multiple occupancy grid maps with each coarser map having half the resolution of the preceding one. However, the multiple map levels are not generated from a single high resolution map by applying Gaussian filtering and downsampling as is commonly done in image processing. Instead, different maps are kept in memory and simultaneously updated using the pose estimates generated by the alignment process. This generative approach ensures that maps are consistent across scales while at the same time avoiding costly downsampling operations. The scan alignment process is started at the coarsest map level, with the resulting estimated pose getting used as the start estimate for the next level, similar to the approach presented in [32]. A positive side-effect is the immediate availability of coarse grained maps which can for example be used for path planning [36]

2.3.2. Gmapping

According to the findings of J. M. Santos, D. Portugal, and R. P. Rocha [38], KartoSLAM, HectorSLAM and Gmapping produce the most accurate maps. These algorithms despite having quite similar performance from map accuracy point of view, are actually conceptually different. That's, HectorSLAM is based on EKF (extended Kalman filter), while Gmapping is based on Rao-Blackwellized particle filtering (RBPF) occupancy grid mapping. Based on the results presented in [39], gmapping can potentially perform well on a limited processing power robotic system such as ours. The algorithm uses a relatively small number of particles to represent the SLAM posterior and reduces the computational effort required to perform resampling to successfully build very accurate maps.

The gmapping offers a flexible way to optimizing the mapping process to fit the application specific needs by tuning some mapping parameters such as number of particles used by RBPF, the displacement step to process new scan and the resampling threshold. [1]

This approach applies two concepts that have previously been identified as key pre-requisites for efficient particle filter implementations (see Doucet et al. [40]), namely the computation of an improved proposal distribution and an adaptive resampling technique.

The algorithms need to draw samples from a proposal distribution π in the prediction step in order to obtain the next generation of particles. Intuitively, the better the proposal distribution approximates the target distribution, the better is the performance of the filter. For instance, if we were able to directly draw samples from the target distribution, the importance weights would become equal for all particles and the resampling step would no longer be needed. Unfortunately, in the context of SLAM a closed form of this posterior is not available in general. As a result, typical particle filter applications [3, 18] use the odometry motion model as the proposal distribution. This motion model has the advantage that it is easy to compute for most types of robots.

This proposal distribution, however, is suboptimal especially when the sensor information is significantly more precise than the motion estimate of the robot based on the odometry, which is typically the case if a robot equipped with a laser range finder (e.g., with a SICK LMS). Imagine a situation in which the meaningful area of the observation likelihood is substantially smaller than the meaningful area of the motion model. When using the odometry model as the proposal distribution in such a case, the importance weights of the individual samples can differ significantly from each other since only a fraction of the drawn samples cover the regions of state space that have a high likelihood under the observation model. As a result, one needs a comparably high number of samples to sufficiently cover the regions with high observation likelihood.

A common approach – especially in localization – is to use a smoothed likelihood function, which avoids that particles close to the meaningful area get a too low importance weight. However, this approach discards useful information gathered by the sensor and, at least to our experience, often leads to less accurate maps in the SLAM context. To overcome this problem, one can consider the most recent sensor observation z_t when generating the next generation of samples. By integrating z_t into the proposal one can focus the sampling on the meaningful regions of the observation likelihood. According to Doucet [41], the distribution

$$p\left(x_t \middle| m_{t-1}^{(i)}, x_{t-1}^{(i)}, z_t, u_{t-1}\right) = \frac{p\left(z_t \middle| m_{t-1}^{(i)}, x_t\right) p\left(x_t \middle| x_{t-1}^{(i)}, u_{t-1}\right)}{p\left(x_t \middle| m_{t-1}^{(i)}, x_{t-1}^{(i)}, u_{t-1}\right)} \qquad (1.20)$$

is the optimal proposal distribution with respect to the variance of the particle weights. [39] The variables would be explained later in the chapter.

When modeling a mobile robot equipped with an accurate sensor, e.g. a laser range finder, it is convenient to use such an improved proposal since the accuracy of the laser range finder leads to extremely peaked likelihood functions. In the context of landmark-based SLAM, Montemerlo et al. [42] presented a Rao-Blackwellized particle filter that uses a Gaussian approximation of the improved proposal. This Gaussian is computed for each particle using a Kalman filter that estimates the pose of the robot. This approach can be used when the map is represented by a set of features and if the error affecting the feature detection is assumed to be Gaussian. With this algorithm the idea of computing an improved proposal is transferred to the situation in which dense grid maps are used instead of landmark-based representations.

Each time a new measurement tuple (u_{t-1}, z_t) is available, the proposal is computed for each particle individually and is then used to update that particle. This results in the following steps:

1. An initial guess $x_t^{\prime(i)} = x_{t-1}^{(i)} \oplus u_{t-1}$ for the robot's pose represented by the particle i is obtained from the previous pose $x_{t-1}^{(i)}$ of that particle and the odometry measurements u_{t-1} collected since the last filter update. Here, the operator $\oplus$ corresponds to the standard pose compounding operator, as in [47].

2. A scan-matching algorithm is executed based on the map $m_{t-1}^{(i)}$ starting from the initial guess $x_t^{\prime(i)}$. The search performed by the scan-matcher is bounded to a limited region around $x_t^{\prime(i)}$. If the scan-matching reports a failure, the pose and the weights are computed according to the motion model (and the steps 3 and 4 are ignored).

3. A set of sampling points is selected in an interval around the pose $\hat{x}_t^{\prime(i)}$ reported scan-matcher. Based on this points, the mean and the covariance matrix of the proposal are computed by pointwise evaluating the target distribution $p\left(z_t \middle| m_{t-1}^{(i)}, x_t\right) p\left(x_t \middle| x_{t-1}^{(i)}, u_{t-1}\right)$ in the sampled positions x_j. During this phase, also the weighting factor $\eta^{(i)}$ is computed.

4. The new pose $x_t^{(i)}$ of the particle i is drawn from the Gaussian approximation $N\left(\mu_t^{(i)}, \Sigma_t^{(i)}\right)$ of the improved proposal distribution.

5. Update of the importance weights.

6. The map $m^{(i)}$ of particle i is updated according to the drawn pose $x_t^{(i)}$ and the observation z_t.

After computing the next generation of samples, a resampling step is carried out depending on the value of N_{eff}. [39]

2.3.3. ORB-SLAM

ORB-SLAM is a versatile and accurate SLAM solution for Monocular, Stereo and RGB-D cameras. It is able to compute in real-time the camera trajectory and a sparse 3D reconstruction of the scene in a wide variety of environments, ranging from small hand-held sequences of a desk to a car driven around several city blocks. It is able to close large loops and perform global relocalisation in real-time and from wide baselines. It includes an automatic and robust initialization from planar and non-planar scenes. [43]

This is the first open-source SLAM system for monocular, stereo and RGB-D cameras, including loop closing, relocalization and map reuse. The RGB-D results shows that by using Bundle Adjustment (BA) it achieves more accuracy than state-of-the-art methods based on ICP or photometric and depth error minimization. By using close and far stereo points and monocular observations the stereo results are more accurate than the state-of-the-art direct stereo SLAM. A lightweight localization mode that can effectively reuse the map with mapping disabled.

ORB-SLAM2 for stereo and RGB-D cameras is built on the monocular feature-based ORB-SLAM [44], whose main components are summarized here for reader convenience. The system has three main parallel threads: 1) the Tracking to localize the camera with every frame by finding feature matches to the local map and minimizing the reprojection error applying motion-only BA, 2) the Local Mapping to manage the local map and optimize it, performing local BA, 3) the Loop Closing to detect large loops and correct the accumulated drift by performing a pose-graph optimization. This thread launches a fourth thread to perform full BA after the pose-graph optimization, to compute the optimal structure and motion solution.

ORB-SLAM2 as a feature-based method preprocess the input to extract features at salient keypoint locations. The input images are then discarded and all system operations are based on these features, so that the system is independent on the sensor being stereo or RGB-D. This system handles monocular and stereo keypoints, which are further classified as close or far.

Stereo keypoints are defined by three coordinates xs = (uL; vL; uR), being (uL; vL) the coordinates on the left image and uR the horizontal coordinate in the right image. For stereo cameras, the algorithm extract ORB in both images and for every left ORB it searches for a match in the right image. This can be done very efficiently assuming stereo rectified images, so that epipolar lines are horizontal. Then it generates the stereo keypoint with the coordinates of the left ORB and the horizontal coordinate of the right match, which is subpixel refined by patch correlation. For RGB-D cameras, it extracts ORB features on the image channel and, as proposed by Strasdat et al. [8], it synthesizes a right coordinate for each feature, using the associated depth value in the registered depth map channel, and the baseline between the structured light projector and the infrared camera, which for Kinect and Asus Xtion cameras we approximate to 8 cm.

A stereo keypoint is classified as close if its associated depth is less than 40 times the stereo/RGB-D baseline, as suggested in [44], otherwise it is classified as far. Close keypoints can be safely triangulated from one frame as depth is accurately estimated and provide scale, translation and rotation information. On the other hand, far points provide accurate rotation information but weaker scale and translation information. Far points are triangulated when they are supported by multiple views.

Monocular keypoints are defined by two coordinates $x_m = (u_L;\ v_L)$ on the left image and correspond to all those ORB for which a stereo match could not be found or that have an invalid depth value in the RGB-D case. These points are only triangulated from multiple views and do not provide scale information, but contribute to the rotation and translation estimation.

One of the main benefits of using stereo or RGB cameras is that, by having depth information from just one frame, a specific structure from motion initialization is not needed as in the monocular case. At system startup a keyframe is created with the first frame, set its pose to the origin, and create an initial map from all stereo keypoints.

The system performs bundle adjustment to optimize the camera pose in the Tracking (motion-only BA), to optimize a local window of keyframes and points in the Local Mapping (local BA), and after a loop closure to optimize all keyframes and points (full BA).

Loop closing is performed in two steps, firstly a loop has to be detected and validated, and secondly the loop is corrected optimizing a pose-graph. In contrast to monocular ORBSLAM, where scale drift may occur [44], the stereo/depth information makes scale observable and the geometric validation and pose-graph optimization no longer require dealing with scale drift and are based on rigid body transformations instead of similarities.

In ORB-SLAM2 a full BA optimization is incorporated after the pose-graph to achieve the optimal solution. This optimization might be very costly and therefore it performs in a separate thread, allowing the system to continue creating map and detecting loops. However, this brings the challenge of merging the bundle adjustment output with the current state of the map. If a new loop is detected while the optimization is running, the optimization is aborted and proceed to close the loop, which will launch the full BA optimization again. When the full BA finishes, the updated subset of keyframes and points optimized by the full BA are merged, with the non-updated keyframes and points that where inserted while the optimization was running. This is done by propagating the correction of updated keyframes (i.e. the transformation from the non-optimized to the optimized pose) to non-updated keyframes through the spanning tree. Non-updated points are transformed according to the correction applied to their reference keyframe.

ORB-SLAM2 follows the policy introduced in monocular ORB-SLAM of inserting keyframes very often and culling redundant ones afterwards. The distinction between close and far stereo points allows to introduce a new condition for keyframe insertion, which can be critical in challenging environments where a big part of the scene is far from the stereo sensor. In such environment a sufficient amount of close points is required to accurately estimate translation, therefore if the number of tracked close points drops below τ_t and the frame could create at least τ_c new close stereo points, the system will insert a new keyframe. It was found empirically that $\tau_t = 100$ and $\tau_c = 70$ works well in all experiments.

A Localization Mode is also incorporated, which can be useful for lightweight long-term localization in well mapped areas, as long as there are not significant changes in the environment. In this mode the Local Mapping and Loop Closing threads are deactivated and the camera is continuously localized by the Tracking using relocalization if needed. In this mode the tracking leverages visual odometry matches and matches to map points. Visual odometry matches are matches between ORB in the current frame and 3D points created in the previous frame from the stereo/depth information. These matches make the localization robust to unmapped regions, but drift can be accumulated. Map point matches ensure drift-free localization to the existing map. [44] [46]

2.3.4. RGBDSLAM

RGBDSLAMv2 is a SLAM solution for RGB-D cameras. It provides the current pose of the camera and allows to create a registered point cloud or an octomap. It features a GUI interface for easy usage, but can also be controlled by ROS service calls, e.g., when running on a robot.

This is one of the first RGB-D SLAM systems that took advantage of the dense color and depth images provided by RGB-D cameras. Compared to the first version, several extensions were introduced that aim at further increasing the robustness and accuracy. In particular, the use of an environment measurement model (EMM) is proposed to validate the transformations estimated by feature correspondences and the iterative closest point (ICP) algorithm. Extensive experiments show that this RGB-D SLAM system allows to accurately track the robot pose over long trajectories and under challenging circumstances. To allow other researchers to use the software, reproduce the results, and improve on them, the system is released under an open-source license. [47]

As a first step the depth and RGB-images are collected with synchronized timestamps. Then features are extracted from the RGB-image by a feature extraction algorithm. RGBDSLAM has multiple feature extraction algorithms implemented. The implementations have different pros and cons in different environments and they differ in computation time. The algorithms implemented are SURF, SIFT and ORB. In the next step of the algorithm extracted features are projected to the depth image. This step introduces some uncertainty into the chain of operations. Mainly due to the synchronization mismatch between depth and RGB-images, but also because of interpolation between points with large differences in depth. The fact that a minor misprojection of a feature lying on an object border on to the depth image can result in a big depth error makes features picked at object borders unreliable. [47]

To find a 6D transform for the camera position in this noise the RANSAC algorithm is used. Features are matched with earlier extracted features from a set of 20 images in the standard configuration. The set consists of a subset including some of the most recent captured images and another subset including images randomly selected from the set of all formerly captured images. Three matched feature pairs are randomly selected and are used to calculate a 6D transform. All feature pairs are then evaluated by their Euclidian distance to each other. Pairs whose Euclidian distance is below a certain threshold are counted as inliers. From these inliers a refined 6D transform is calculated using GICP.

RANdom SAmple Consensus, or RANSAC for short, is an iterative algorithm used to adapt the parameters of a mathematical model to experimental data. RANSAC is a suitable method when a data set contains a high percentage of outliers, i.e. measurements that suffer from measurement errors so large that the validity of the measurements is low. The method was first presented in the beginning of the eighties in Fischler and Bolles [48] and was suggested to be a suitable method for automated image analysis.

Assume a mathematical model that has n free parameters which can be estimated given a set of measurements, P. The number of measurements in P has to be greater than n, #P > n. Let S and T be two different varying subsets of P. Given the assumptions the RANSAC algorithm works as follows:

1. Randomly select a subset of the measurements in P and call it S. Use S to make a first estimate of the n free parameters of the model.
2. Use the current estimate of the model to select a new subset of points, T , from the measurements that are within some error tolerance from the model.
3. If T contains more measurements than some given limit then re-estimate the free parameters of the model according to this new subset. Calculate a measure of how well T and the model coincide, store that value and select a new subset S.
4. If T does not contain more measurements than the given limit, randomly select a new subset S from P and start all over again.
5. If none of the selected subsets hold more measurements than the limit, exit in failure, or if the maximum number of iterations has been reached, exit.

The method is characterized by three parameters:

1. The error tolerance used to determine when data is not part of the model.
2. The maximum number of iterations in the algorithm.
3. The minimum value on the number of measurements in a subset to be used for parameter estimation.

The big advantage with RANSAC is the robust way it handles outliers in the data set. The drawbacks with RANSAC is that it does not guarantee any solution, nor that a given solution is optimal. Furthermore the three parameters mentioned above are to a large extent problem specific, which means that experimental adjustment to the specific case treated is required. [47]

2.4. Robot Operating System (ROS)

ROS is an open-source meta-operating system for robots. It provides the services that are expected from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.

The ROS runtime "graph" is a peer-to-peer network of processes (potentially distributed across machines) that are loosely coupled using the ROS communication infrastructure. ROS implements several different styles of communication, including synchronous RPC-style communication over services, asynchronous streaming of data over topics, and storage of data on a Parameter Server.

ROS is a distributed framework of processes (aka Nodes) that enables executables to be individually designed and loosely coupled at runtime. These processes can be grouped into Packages and Stacks, which can be easily shared and distributed. ROS also supports a federated system of code Repositories that enable collaboration to be distributed as well. This design, from the filesystem level to the community level, enables independent decisions about development and implementation, but all can be brought together with ROS infrastructure tools.

ROS is divided into three conceptual levels: the filesystem level, the computation graph level, and the community level. [49]

2.4.1. Filesystem Level

The filesystem level is the organization of the ROS framework on a machine (see Fig.2.1). At the heart of the ROS's organization of software is the package. A package may contain ROS runtime execution programs, which are called nodes, a ROS-independent library, datasets, configuration files, third-party software, or any software that should be organized together [48]. The goal of the packages is to provide easy to use functionality in a well-organized manner so that software may be reused for many different projects. This organization, along with object-oriented programming, allows packages to act as modular building blocks, working harmoniously together to accomplish the desired end-state. Packages typically follow a common structure and usually contain the following elements: package manifests, message types, service types, headers, executable scripts, a build file, and runtime processes [48]. Package manifests provide metadata about a package, such as the name, author, version, description, license information, and dependencies. Packages may also contain message types, which define the structure of data for messages sent within ROS, and service types, which define the request and response data structures for services. Also within the filesystem level are repositories, which are a collection of packages sharing a common version control system. Both packages and repositories help make ROS a modular system. [47]

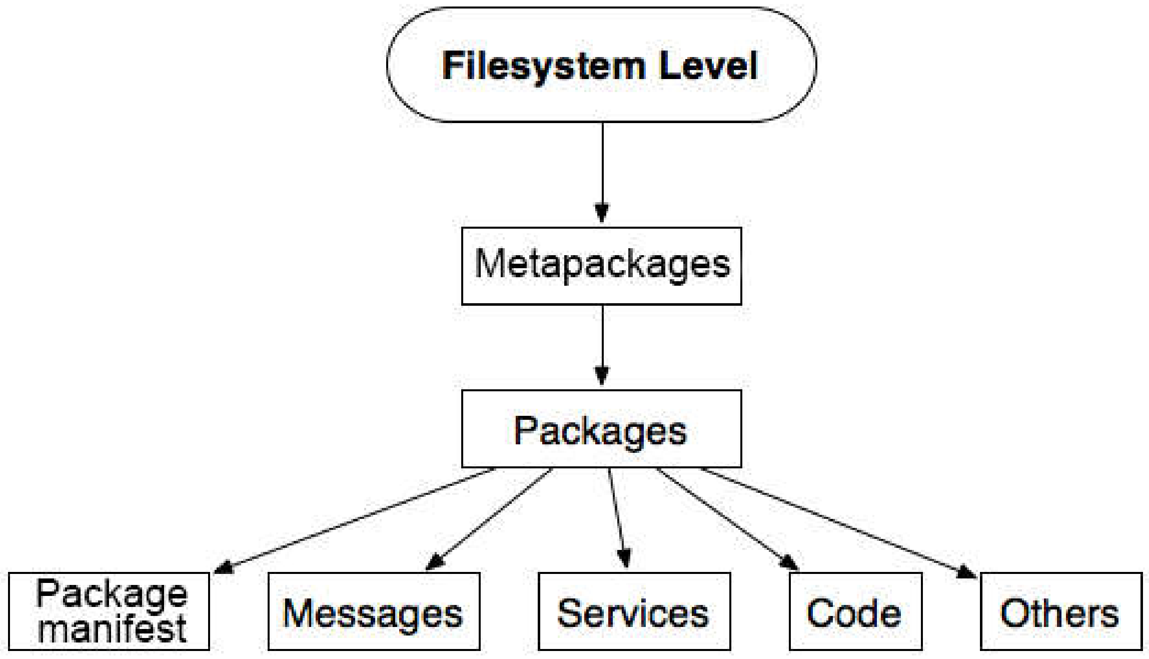


Figure 2.1. ROS Filesystem level scheme

2.4.2. Computation Graph Level

The computation graph level is where ROS processes data within a peer-to-peer network (See Fig.2.2). The basic elements of ROS's computation graph level are nodes, messages, topics, services, bags, Master, and Parameter Server. Nodes are the small-scale workhorses of ROS, subscribing to topics to receive information, performing computations, controlling sensors and actuators, and publishing data to topics for other nodes to use [50]. The rosnode tool is a useful command-line tool for displaying information about ROS nodes. The command, rosnode list, displays all active nodes running on the ROS Master. A package may have many nodes within it to accomplish a group of computation and tasks, in which they all communicate with each other through topics and services via messages.

The primary method in which nodes pass data to each other is by publishing messages to topics. A message is simply a structure of data so it is in a useful, standard format for other nodes to use. Standard types, such as integer, floating point, and Boolean, are supported as well as arrays. The command rosmsg list prints all messages available to the ROS Master. The key to the modularity of ROS is the method in which nodes typically communicate with each other through topics.

Rather than communicating directly with each other, nodes usually communicate through topics. Topics are named hubs in which nodes can publish and subscribe and are the crux of what makes ROS an object-oriented and modular environment. Nodes that generate data are only interested in publishing that data, in the correct message format, to the correct topic. Nodes that require data simply subscribe to the topics of interest to pull the required information. This method of publishing and subscribing to topics decouples the production of information from the consumption of information. It allows nodes within different packages to work harmoniously with each other even though they may have different origins and functions. The rostopic command-line tool is useful for displaying debugging information about ROS topics. To display all active topics, the command rostopic list is utilized. The command rostopic info <topic_name> prints the message type accepted by the topic and publishing and subscribing nodes. Another useful command-line tool is rostopic echo <topic_name>, which prints messages published to a topic. The commands rostopic hz <topic_name> and rostopic bw <topic_name> displays the publishing rate and the bandwidth used by a topic, respectively. Additionally, data can be manually published to a topic by using the rostopic pub <topic_name> command.

In addition to publishing messages to topics, nodes can also exchange a request and response message as part of a ROS service. This is useful if the publish and subscribe (many-to-many) communication method is not appropriate, such as a remote procedure call. A ROS node that provides data offers a service under a string name, and a client node that requires data calls the service by sending the request message and awaiting the response. Active services can be displayed by utilizing the command rosservice list, and information about a service can be found by using rosservice info <service_name>. [50]

Bags are a method for recording and storing ROS message data. This is a powerful tool that allows users to store, process, analyze, and visualize the flow of messages. Bags are created utilizing the rosbag tool, which subscribes to one or more ROS topics and stores message data as they are received. This stored data can be replayed in ROS to the same topics, as if the original nodes were sending the messages. This tool is useful for conducting experiments using a controlled set of data streams to test different algorithms, sensors, actuators, and controllers. To record data, the command rosbag record <topic_names> should be used. To view information about a bagfile already created, the command rosbag info <bag_file> should be utilized. The command rosbag play <bag_file> can be used to publish messages from topics just as if they were being played for the first time. When rosbag is utilized to play data, the time synchronization is based on the global timestamp when the bagfile was recorded. It is recommended that when playing back data using rosbag play to use rosparam set sim_time true and rosbag play <bag_file> --clock in order to run the recorded system with simulated timestamps.

A launch file is method of launching multiple ROS nodes, either locally or remotely, as well as establishing parameters on the ROS Parameter Server. It is useful for running large projects, which may have many packages, nodes, libraries, parameters, and even other launch files, which all can be started via one launch file rather than individually running each node separately. The roslaunch tool uses extensible markup language (XML) files that describe the nodes that should be run, parameters that should be set, and other attributes of launching a collection of ROS nodes. The roslaunch tool is utilized by using the command roslaunch <package_name> <file.launch>.

The ROS Master acts as a domain name system server, storing topics and services registration information for ROS nodes. ROS Master provides an application program interface (API), a set of routines and protocols, tracking services and publishers and subscribers to topics. A node notifies ROS Master if it wants to publish a message to a topic. When another node notifies the master that it wants to subscribe to the same topic, the master notifies both nodes that the topic is ready for publishing and subscribing. The master also makes callbacks to nodes already online, which allows nodes to dynamically create connections as new nodes are run. The ROS Master is started with the command roscore and must be used to run nodes in ROS. The ROS Master also provides the Parameter Server. The ROS Parameter Server can store integers, floats, Boolean, dictionaries, and lists and is meant to be globally viewable for non-binary data. The parameter server is useful for storing global variables such as the configuration parameters of the physical characteristics of a robot. ROS parameters can be displayed by utilizing the command rosparam list. A user can also set a parameter from the command line by using rosparam set <parameter_name> <parameter_value>. Parameters can also be loaded from a .yaml file by using the command rosparam load <parameters.yaml>.

Names have an important role within ROS. Every node, topic, service, and parameter has a unique name. This architecture allows for decoupled operation that allows large, complex systems to be built. ROS supports command-line remapping of names, which means a compiled program may be reconfigured at runtime to operate in a different computation graph topology. This means that the same node can be run multiple times, publishing difference messages to separate topics. [49]

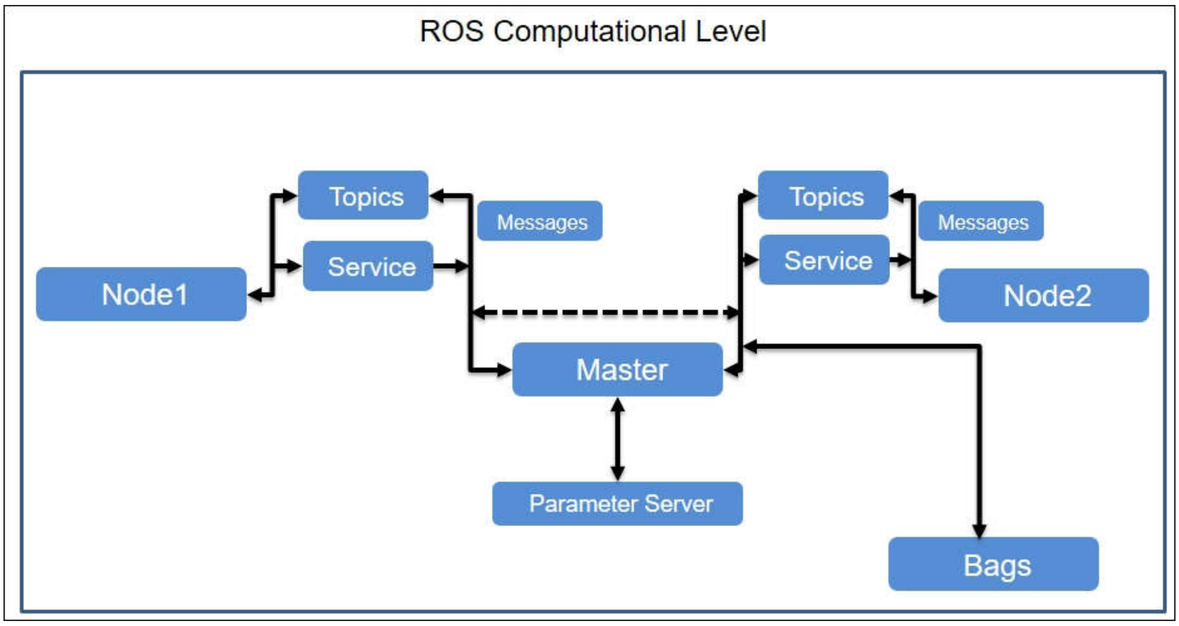


Figure 2.2. ROS Computational level

2.4.3. Community Level

The ROS Community Level consists of ROS distributions, repositories, the ROS Wiki, and ROS Answers (See Figure 2.3), which enable researchers, hobbyists, and industries to exchange software, ideas, and knowledge in order to progress robotics communities worldwide. ROS distributions are similar to the roles that Linux distributions play. They are a collection of versioned ROS stacks, which allow users to utilize different versions of ROS software frameworks. Even while ROS continues to be updated, users can maintain their projects with older more stable versions and can easily switch between versions at any time.

ROS does not maintain a single repository for ROS packages; rather, ROS encourages users and developers to host their own repositories for packages that they have used or created. ROS simply provides an index of packages, allowing developers to maintain ownership and control over their software. Developers can then utilize the ROS Wiki to advertise and create tutorials to demonstrate the use and functionality of their packages. The ROS Wiki is the forum for documenting information about ROS, where researchers and developers contribute documentation, updates, links to their repositories, and tutorials for any open-sourced software they have produced. ROS Answers is a community-oriented site to help answer ROS-related questions that users may have. [51]

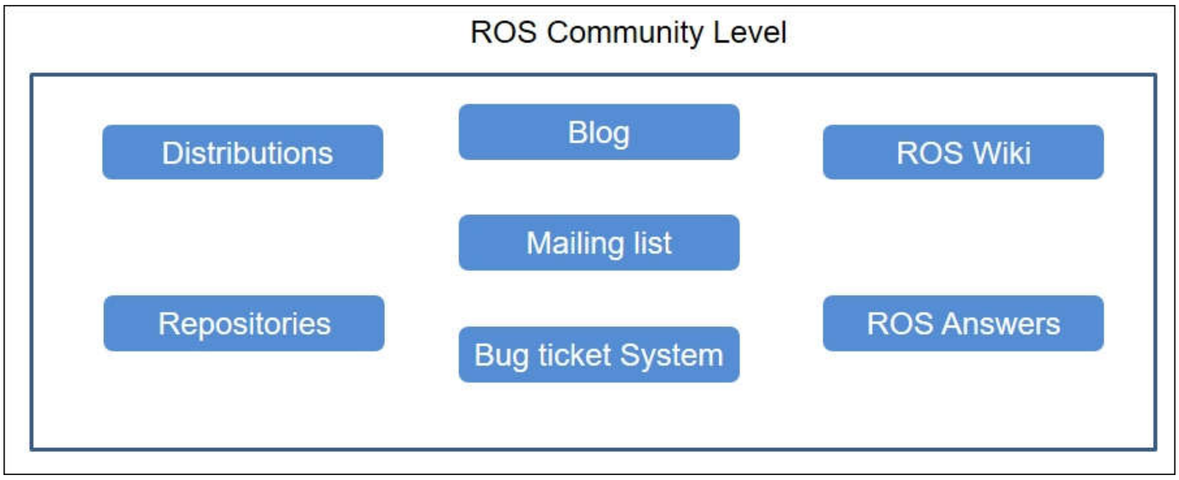


Figure 2.3. ROS Community level

2.4.4. Other ROS Concepts

Unified Robot Description Format

The unified robot description format (URDF) package contains an XML file that represents a robot model. The URDF is another tool within ROS that makes it a modular system. Rather than creating a unique process for different styles of robots, nodes are created without regard for the robot that will utilize them. The URDF file provides the necessary, robot-specific, information so nodes may conduct their procedures. A URDF file is written so that each link of the robot is the child of a parent link, with joints connecting each link, and joints are defined with their offset from the reference frame of the parent link and their axis of rotation [51]. In this way, a complete kinematic model of the robot is created. A tree diagram can be visualized utilizing the urdf_to_graphiz tool as shown on Figure 2.4.

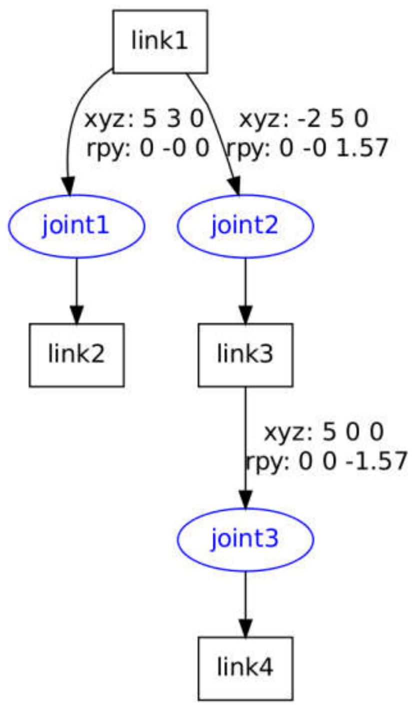


Figure 2.4. URDF tree diagram example

Coordinate and Transform frames

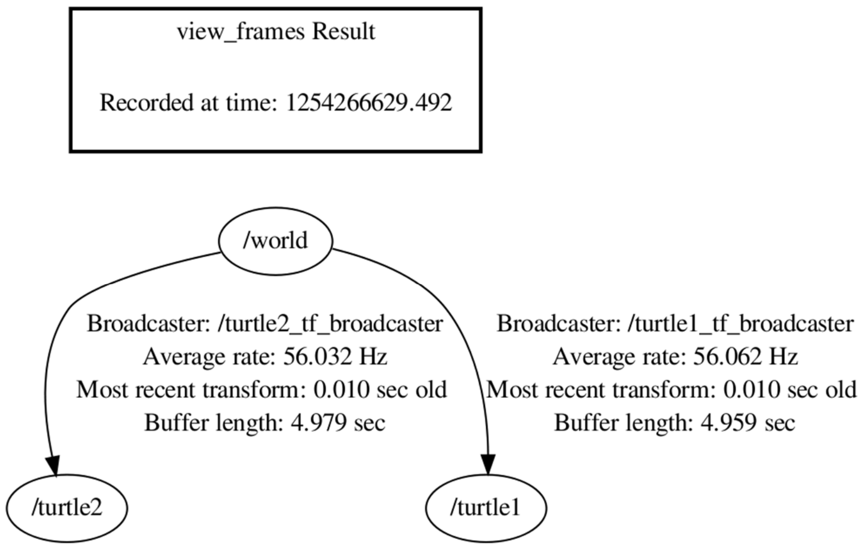

view_frames Result
Recorded at time: 1254266629.492
/world
Broadcaster: /turtle2_tf_broadcaster
Average rate: 56.032 Hz
Most recent transform: 0.010 sec old
Buffer length: 4.979 sec
Broadcaster: /turtle1_tf_broadcaster
Average rate: 56.062 Hz
Most recent transform: 0.010 sec old
Buffer length: 4.959 sec
/turtle2
/turtle1

Figure 2.5. ROS view_frames command example

A robotic system typically has many three-dimensional coordinate frames that change over time. The tf ROS package keeps track of multiple coordinate frames in the form of a tree structure (See Fig.1.5). Just as the URDF manages joints and links, the tf package maintains the relationships between coordinate frames of points, vectors, and poses, and computes the transforms between them. The tf package operates in a distributed system; all ROS components within the system have access to information about the coordinate frames. The transform tree can also be viewed by developers for debugging by utilizing the view_frames tool as shown in Figure 2.5. Additional command-line tools for the tf package are rosrun tf tf_monitor, rosrun tf tf_echo <source_frame> <target_frame>, and roswtf, which, respectively, monitors delays between transforms of coordinate frames, prints transforms between coordinate frames, and aids in debugging. [51]

CHAPTER 3

3. DEVELOPMENT

3.1. ROS Installation

Both the Raspberry Pi 3 and the control PC have Ubuntu 16.04 LTS as the main operating system, so the installation, as described in [59]:

First, we need to set up the ROS repository and add the required keys to our system to gain access to the packages.

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/source
s.list.d/ros-latest.list'

sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F71
7815A3895523BAEEB01FA116
```

Then, after updating the Ubuntu repository list, the desktop version of ROS could be installed.

```
sudo apt-get update

sudo apt-get install ros-kinetic-desktop-full
```

ROS provides 4 different configurations, the one I chose has all the required tools for building and evaluating the platform.

After the successful installation, a rosdep tool should be initialized. Rosdep handles the installation of all of the dependencies of the packages and is required to run ROS system components.

```
sudo rosdep init

rosdep update
```

Then, after adding the ROS environment variables to the bashrc script, the ROS framework is installed.

```
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc

source ~/.bashrc
```

For the usage of the ROS packages the Workspace should be created. Workspace will contain all the tools and algorithms provided, and is the correct way of handling ROS packages.

```
mkdir -p ~/catkin_ws/src

cd ~/catkin_ws/src

catkin_init_workspace
```

```
cd ~/catkin_ws/
catkin_make
echo "source ~/devel/setup.bash " >> ~/.bashrc
```

After these steps, the ROS Workspace under the name of “catkin_ws” is now created.

3.2. Prerequisites

For the SLAM algorithms to work we are required to install a couple of packages, which the algorithms depend on.

3.2.1. OpenNI

The OpenNI 2.0 API provides access to PrimeSense compatible depth sensors. It allows an application to initialize a sensor and receive depth, RGB, and IR video streams from the device. It provides a single unified interface to sensors and .ONI recordings created with depth sensors. OpenNI also provides a uniform interface that third party middleware developers can use to interact with depth sensors. Applications are then able to make use of both the third party middleware, as well as underlying basic depth and video data provided directly by OpenNI. [65]

Installation requires the Rgbd_Launch package to be installed too.

```
cd ~/catkin_ws/src
git clone https://github.com/ros-drivers/rgbd_launch
git clone https://github.com/ros-drivers/openni_launch.git
rosdep install rgbd_launch
rosdep install openni_launch
cd ~/catkin_ws
catkin_make
```

After the successful compilation the OpenNI node could be launched. There is no need for special configuration, because the OpenNI comes with the optimized set of the variables.

3.2.2. Depthimage_to_laserscan

The depthimage_to_laserscan package is essential when working with low cost setups like depth sensors or the Microsoft Kinect. It furthermore is essential when interfacing SLAM algorithms.

It subscribes to the PointCloud2 topic that is provided by Kinect (Figure 3.1), selects a few rows of information in the center of it, approximates the parameters and outputs the LaserScan topic with the information look-alike as what the Laser scanner would provide (Figure 3.2). [66]

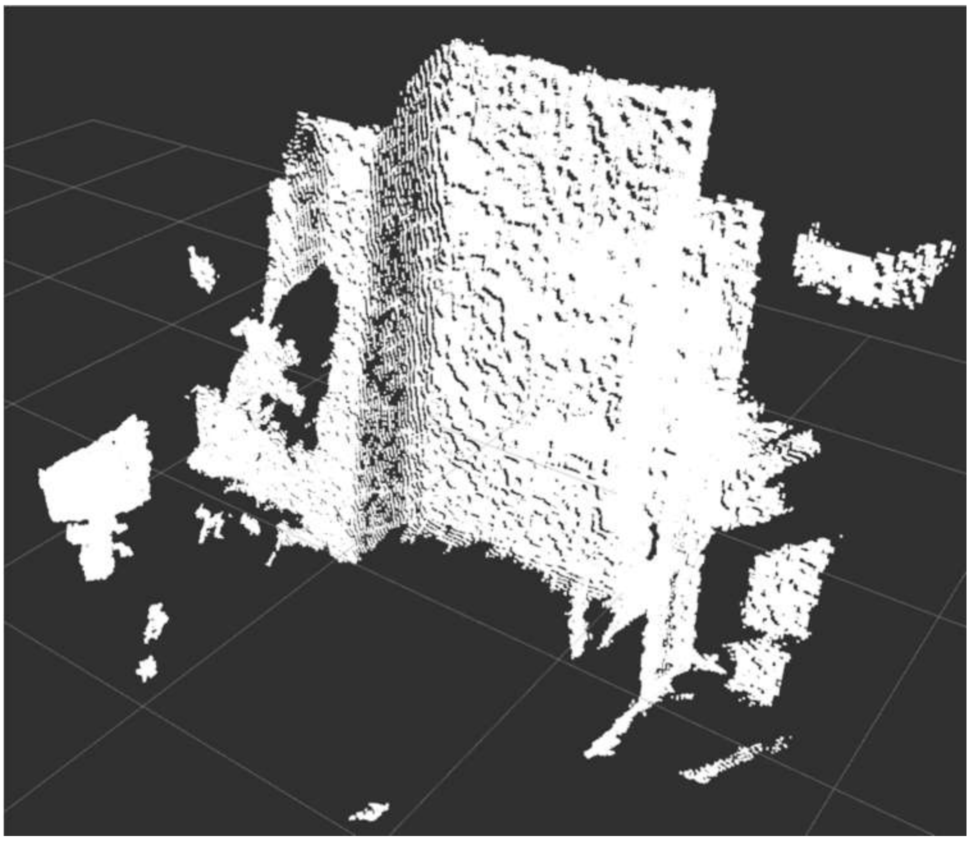


Figure 3.1. The point cloud obtained from Kinect

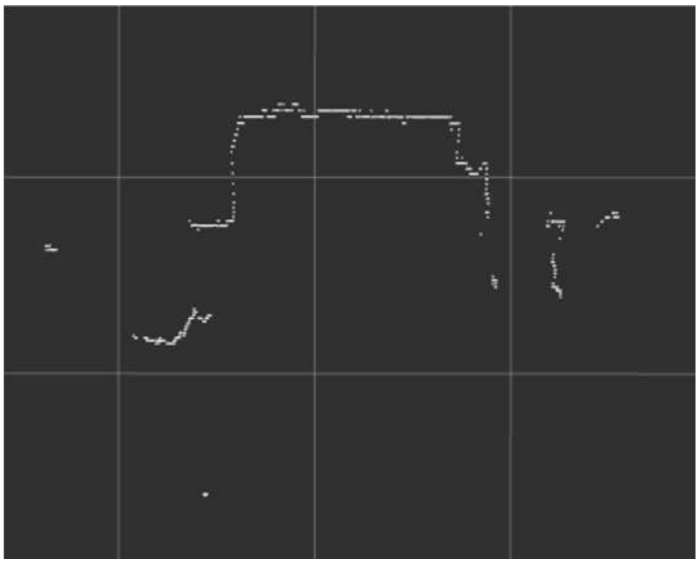


Figure 3.2. The laser scan that was provided from the package

The package should be cloned from Github.com into the workspace and built.

```
cd ~/catkin_ws/src
git clone https://github.com/ros-perception/depthimage_to_laserscan.git
rosdep install depthimage_to_laserscan
cd ~/catkin_ws/
catkin_make
```

After the successful compilation of the project it could be configured and launched.

It is a known practice of launching packages through prepared launch files, providing the nodes the required configurations and variables:

1. Scan_height parameter, sets the height of the pixel row in the center that would be transformed into the laser scan;
2. Scan_time parameter, sets the rate of which the Laserscan messages would be published;
3. Output frame id, sets the name of the node;
4. Range_min, sets the minimal range of the pointcloud data to be processed in meters;
5. Range_max, sets the maximal range;
6. Remapping the Image topic to the one the Kinect is providing;
7. Remapping the Scan topic to the one which we would be using.

The following code was used in the Launch file.

```
<node pkg="nodelet" type="nodelet" name="depthimage_to_laserscan" args="load depthimage_to
_laserscan/DepthImageToLaserScanNodelet camera/camera_nodelet_manager">
    <param name="scan_height" value="10"/>
    <param name="scan_time" value="0.033"/>
    <param name="output_frame_id" value="/$(arg camera)_depth_frame"/>
    <param name="range_min" value="0.45"/>
    <param name="range_max" value="5.00"/>
    <remap from="image" to="$(arg camera)/$(arg depth)/image_raw"/>
    <remap from="scan" to="$(arg scan_topic)"/>
</node>
```

3.2.3. Laser_Scan_Matcher

The laser_scan_matcher package [67] is an incremental laser scan registration tool. The package allows to scan match between consecutive sensor_msgs/LaserScan messages, and publish the estimated position of the laser as a geometry_msgs/Pose2D or a tf transform.

The package can be used without any odometry estimation provided by other sensors. Thus, it can serve as a stand-alone odometry estimator. Alternatively, several types of odometry input could be provided to improve the registration speed and accuracy.

The package should be cloned to the workspace folder and built.

```
cd ~/catkin_ws/src
git clone https://github.com/ccny-ros-pkg/scan_tools.git
rosdep install scan_tools
cd ~/catkin_ws/
catkin_make
```

To ensure the odometry frame is calculated and received correctly, we need to provide the package with the next variables:

1. fixed_frame – set to “map”, this is the frame that is fixed and is not moving;
2. base_frame – set to “base_link”, this is the frame of the robot platform;
3. use_odom – set to “false” due to our platform not having any odometry sensor.

Other parameters should be configured already for the Kinect.

3.2.4. Camera_calibration

This is the package for calibrating the Kinect’s Depth and RGB cameras for achieving better quality of the data obtained from them. [68]

The calibration is not necessarily, because the openni_camera driver provides default camera models out-of-the-box with reasonably accurate focal lengths (relating 3D points to 2D image coordinates). They do not model lens distortion, but fortunately the Kinect uses low-distortion lenses (|k1| ~= 0.1), so even the edges of the image are not displaced by more than a few pixels.

The camera_calibration package is a part of the image_pipeline metapackage and could not be installed separately.

```
cd ~/catkin_ws/src
git clone https://github.com/ros-perception/image_pipeline.git
```

```
rosdep install image_pipeline

cd ~/catkin_ws/

catkin_make
```

For the calibration the special chessboard-like file should be printed and the size of the squares on it should be measured.

Calibrating the RGB camera (Figure 3.3):

```
rosrun camera_calibration cameracalibrator.py image:=/camera/rgb/image_raw camera:=/camera/r
gb --size 8x6 --square 0.0245
```

Calibrating the Depth camera (Figure 3.4):

```
rosrun camera_calibration cameracalibrator.py image:=/camera/ir/image_raw camera:=/camera/ir -
-size 8x6 --square 0.0245
```

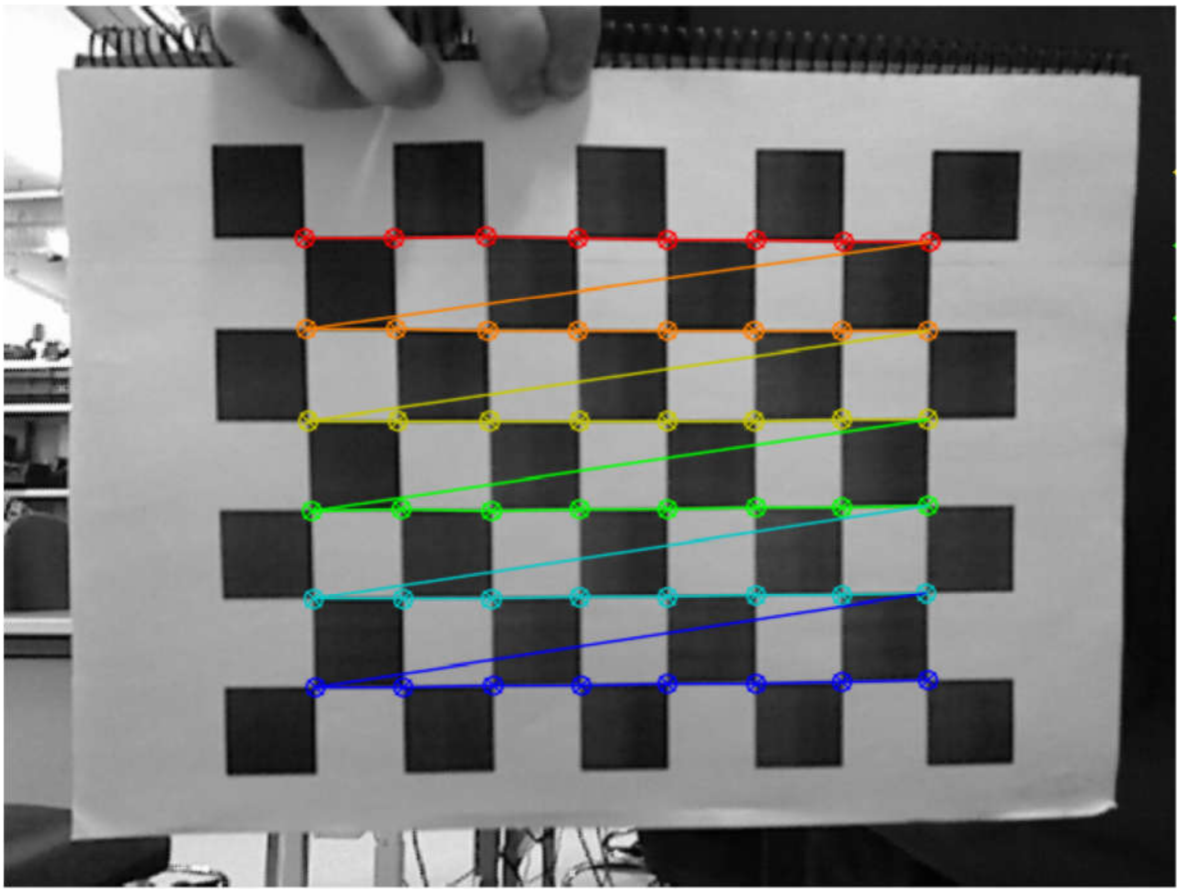


Figure 3.3. Calibration of the RGB camera of the Kinect.

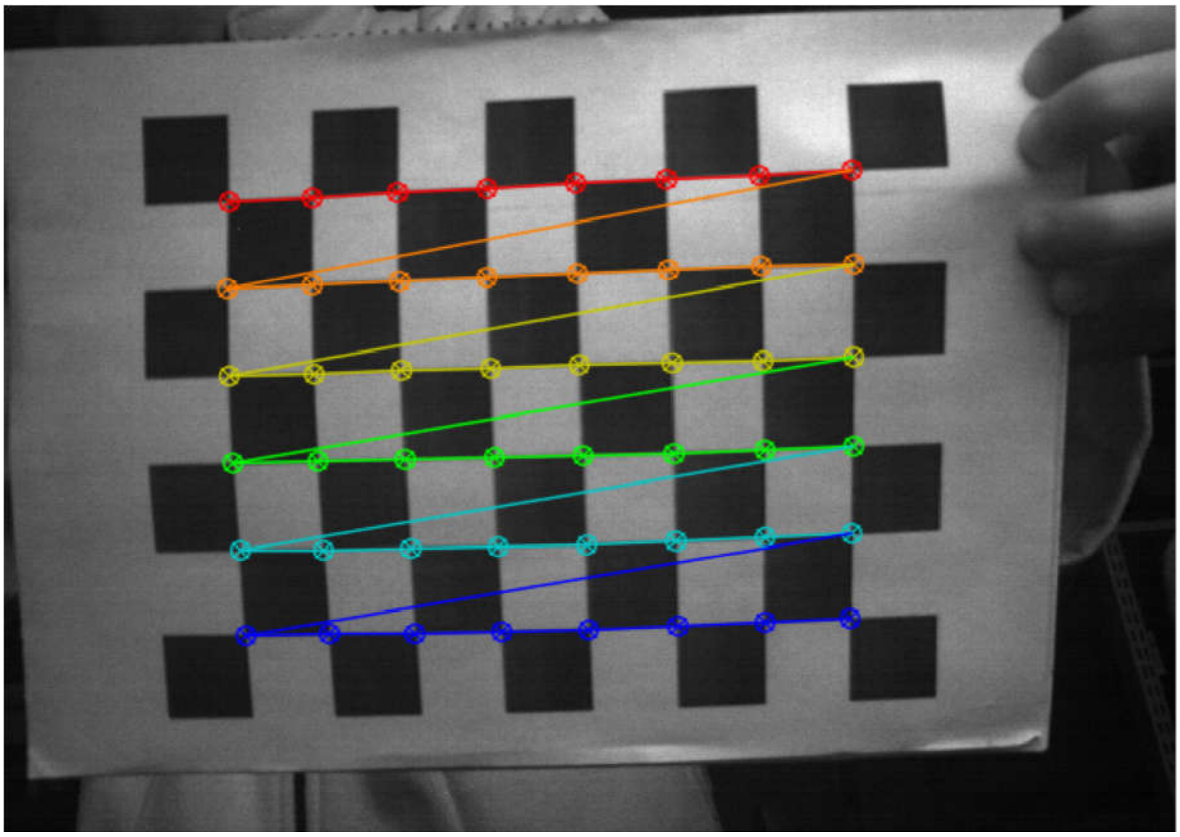


Figure 3.4. Calibration of the Depth camera.

Chess pattern that is printed on paper is detected by the package and the lens distortion parameters are written into the ~/.ros/camera_info/ folder, from where it is detected by any package that requires connection to the Kinect.

3.3. Algorithms

I chose 4 algorithms of SLAM, which belong to the two groups:

1. Laser-based SLAM
2. 3D occupancy grid SLAM

For the Laser-based SLAM I selected Hector SLAM and Gmapping, and for 3D-based SLAM I chose RGBDSLAMv2 and ORB-SLAM2.

3.3.1. Hector SLAM

For the usage of hector_mapping in ROS, a source of “sensor_msgs/LaserScan” data is required. The node uses tf for transformation of scan data, so the LIDAR does not have to be fixed related to the specified base frame. Odometry data is not needed.

Kinect provides us with 3D pointcloud, and we are required to transform it to the laserscan. ROS has a package “Depthimage_to_Laserscan”, which mimics the laser scanner by processing the Depth images from Kinect and turning them to the format the laser scanners are delivering.

Installation with all of the dependencies is simple:

```
cd ~/catkin_ws/src
git clone https://github.com/tu-darmstadt-ros-pkg/hector_slam.git
rosdep install hector_slam
cd ~/catkin_ws/
catkin_make
```

After performing these commands the package “hector_slam” and all of its dependencies are built and installed.

Subscribed topics:

1. /scan (sensor_msgs/LaserScan)

This is the topic delivered by “depthimage_to_laserscan” package used by the SLAM system.

2. /syscommand (std_msgs/String)

System command. If the string equals "reset" the map and robot pose are reset to their initial state.

Published topics:

1. /map_metadata (nav_msgs/MapMetaData)
2. /map (nav_msgs/OccupancyGrid)

These topics contain the map data, which is latched, and updated periodically.

3. /slam_out_pose (geometry_msgs/PoseStamped)

The estimated robot pose without covariance.

4. /poseupdate (geometry_msgs/PoseWithCovarianceStamped)

The estimated robot pose with an gaussian estimate of uncertainty.

3.3.2. Gmapping

Gmapping also requires a source of “sensor_msgs/LaserScan” data and is launched with “depthimage_to_laserscan” package.

Gmapping requires an odometry source from the robot to understand correctly where the camera is pointed at and where it moves. For achieve that goal the ROS provided “laser_scan_matcher” package.

The installation is as follow:

```
cd ~/catkin_ws/src
git clone https://github.com/ros-perception/slam_gmapping.git
git clone https://github.com/ros-perception/openslam_gmapping.git
rosdep install slam_gmapping
rosdep install openslam_gmapping
cd ~/catkin_ws/
catkin_make
```

Gmapping requires the following variables to be set:

1. Scan – set to “/scan”, this is the topic from where Gmapping takes the laser scan information;
2. Base_frame – set to “base_link”, the frame connected to the robot platform;
3. Odom_frame – set to “odom”, the frame of the estimated odometry provided by laser_scan_matcher node.

All other variables are good optimized.

3.3.3. RGBDSLAM-v2

RGBDSLAM-v2 takes Pointcloud2 messages and builds a map from them. It does not require odometry information from the platform.

Installation is harder than any of the previous algorithms, because it requires a specially optimized program g2o to be built before installing theh package itself.

```
git clone -b c++03 https://github.com/felixendres/g2o.git g2ofork
mkdir g2ofork/build
cd g2ofork/build
cmake .. -DCMAKE_INSTALL_PREFIX=/opt/g2ofork -DG2O_BUILD_EXAMPLES=OFF
nice make -j2 install
```

```
export G2O_DIR=/opt/g2ofork
cd ~/catkin_ws/src
git clone -b kinetic https://github.com/felixendres/rgbdslam_v2.git $WORKSPACE/src/rgbdslam
rosdep install rgbdslam
rosdep install rgbdslam_v2
cd ~/catkin_ws/
catkin_make
```

The compilation and installation of this package took two hours on the notebook.

It requires the following variables to be set:

1. config/topic_image_mono – set to "/camera/rgb/image_color", the image from the camera on the Kinect;
2. config/topic_image_depth – set to "/camera/depth_registered/sw_registered/image_rect_raw", the depth image from Kinect.

3.3.4. ORB_SLAM2

The main difference between this algorithm and the others that it does not require ROS to be installed to work. Also, it installs as the separate program and should not be put into the ROS workspace.

The installation is simple, because the developers provided the special installation script that builds everything needed and compiles the package itself.

```
cd ~/
git clone https://github.com/raulmur/ORB_SLAM2.git ORB_SLAM2
cd ORB_SLAM2
chmod +x build.sh
./build.sh
chmod +x build_ros.sh
./build_ros.sh
```

After the installation the path to the built ROS files should be set up for ROS to find the package files.

```
export ROS_PACKAGE_PATH=${ROS_PACKAGE_PATH}:~/ORB_SLAM2/Examples/ROS
```

The installation and compilation of this algorithm took 40 minutes on the notebook and slightly less than 4 hours on the Raspberry Pi 3.

For an RGB-D input from Kinect's topics /camera/rgb/image_raw and /camera/depth_registered/image_raw the node ORB_SLAM2/RGBD should be started and provided with the Vocabulary file and with the camera calibration file.

3.4. Mobile platform

The aim of this thesis was to build a mobile platform capable of navigate itself through an unknown environment while being constantly building and updating the map of such environment.

3.4.1. Requirements

The platform should be compact and lightweight, easy to modify when required and should have enough sensors and computational capacity to perform the calculations in real time. Nevertheless, the components of the platform should be relatively inexpensive and be able to provide comparable results.

To sum up, the main requirements for the platform:

1. Ability to be easily modified and disassembled
2. Ability to rotate while staying on one place
3. Be lightweight
4. Have low costs for production

To achieve the first requirement the platform should consist of multiple blocks that could be replaced with ease, and also the use of the screws or any other holding materials should be minimized.

Second requirement is achieved by placing the engines in the corners of the rectangle shape, so the centers of the wheels would form a square, with its center aligned with the center of the platform. This way if the engines on the opposite sides would receive contrary control signals, the platform would rotate around its center.

The platform was planned to be printed on the 3D printer, so the usage of ABS plastic was the best solution to achieve relatively low weight of the platform while keeping the production costs at a minimum.

3.4.2. Microsoft Kinect

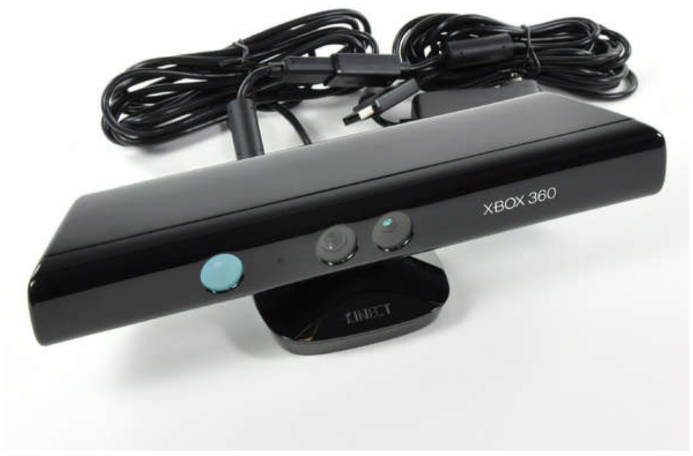


Figure 3.5 Microsoft Kinect for Xbox 360

For SLAM purposes, the best and most accurate type of sensor would be LIDAR, because of the wide range, speed and quality that they could provide, but their price blocks most of the low- and medium-range applications. A cheaper alternative to LIDAR would be RGB-D (Red-Green-Blue-Depth) cameras, which a capable of providing color and depth images simultaneously.

As a consumer-grade RGB-D camera, I pre-selected a few variants – Microsoft Kinect for Xbox 360 (See Fig.3.5), Microsoft Kinect for Xbox One and Asus Xtion Live – but after searching the market for the availability the Asus Xtion Live was proved to be much harder to find because the production of it has been stopped. Microsoft Kinect, as a sensor developed for the gaming consoles in the first place and shipped most of the time in a single package with them, is still widely used by gamers and researchers and still has a big share in the market as an easy-to-use RGB-D sensor.

Table 3.1 Comparison of Microsoft Kinect for Xbox 360 and Xbox One [52]

Feature	Kinect for Xbox 360	Kinect for Xbox One
Color camera	640 x 480 @30 fps	1920 x 1080 @30 fps
Depth Camera	320 x 240	512 x 424
Working range (m)	0.4 - 4.5	0.5 - 8
Horizontal Field of View	57 degrees	70 degrees
Vertical Field of View	43 degrees	60 degrees
USB Standard	2.0	3.0
Market price	From $20 for a used one	$199 for a new, from $50 for a used

According to Table 3.1, pros for using Kinect for Xbox One would be better color and depth cameras specifications, bigger Field of View and working range, but the requirement for using only USB 3.0 standard for communication and way bigger price makes a Kinect for Xbox 360 a better option. Due to its longer life span, there were a lot of projects including it in multiple applications, which results in much bigger support by the community.

3.4.3. Raspberry Pi 3

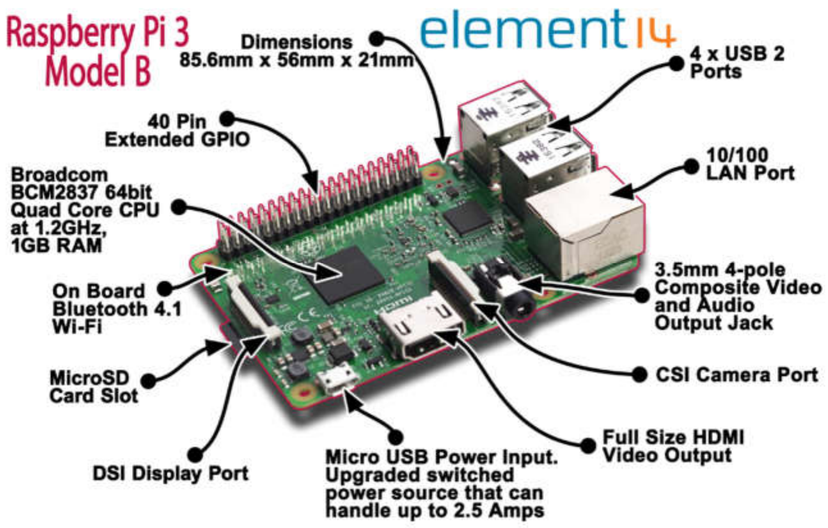

Raspberry Pi 3 Model B
element14
Dimensions 85.6mm x 56mm x 21mm
4 x USB 2 Ports
40 Pin Extended GPIO
10/100 LAN Port
Broadcom BCM2837 64bit Quad Core CPU at 1.2GHz, 1GB RAM
3.5mm 4-pole Composite Video and Audio Output Jack
On Board Bluetooth 4.1 Wi-Fi
MicroSD Card Slot
CSI Camera Port
Full Size HDMI Video Output
DSI Display Port
Micro USB Power Input. Upgraded switched power source that can handle up to 2.5 Amps

Figure 3.6 Raspberry Pi 3 [53]

For the SLAM purposes the microcontroller should be able to run a Linux operating system, should have an array of GPIO (General Purpose Input-Output) pins and a few USB 2.0 ports. A great addition would be having a built-in Wi-Fi adapter for easier communication and troubleshooting the board and ability to work from the external power supply. Most important – the CPU and GPU of the microcontroller should have decent computing capabilities required for running SLAM algorithms in real time.

After searching for the microcontroller, that fits the requirements, I pre-selected three models from different manufacturers – Raspberry Pi 3 (See Fig.3.6), Orange Pi Plus 2 and ODROID XU4.

Table 3.2 Comparison of Raspberry Pi 3, Orange Pi Plus 2 and ODROID XU4 [54]

Feature	Raspberry Pi 3	Orange Pi Plus 2	ODROID XU4
SoC Vendor and Chip	Broadcom BCM2837	Allwinner H3	Samsung Exynos 5422
CPU Instructions	ARMv8	ARMv7	ARMv7
CPU Frequency	1.2GHz	1.5GHz	1.5GHz

Memory	1GB DDR2	2GB DDR3	2GB DDR3
Wi-Fi module installed	Yes	No	Yes
Power requirements	5V 2.5A	5V 2A	5V 4A
GPIO support by OS	Excellent	Medium	Medium
Community support	Excellent	Good at the launch	Good at the launch

Comparing the specifications in Table 3.2, the ODROID X4 seems to be the best solution, followed by Raspberry Pi 3. But after looking at forums dedicated to each of the three microcontrollers I noticed multiple threads filled with complains about Orange Pi and ODROID's flaws compared to the Raspberry Pi. They included worse support from the manufacturer, hardware and software faults due to the System-On-Chipset used, poor driver support and requirement to compile every tool needed from scratch to avoid the typical problems. Raspberry Pi, on the other hand, has a wide community dedicated to the software development and hardware optimization. The market share of Raspberry Pi is also much wider than any competitors. Due to that, to low price of Raspberry board and the better support from community and software developers I selected Raspberry Pi 3 as a main board for my platform.

3.4.4. Motors and Motor Shield

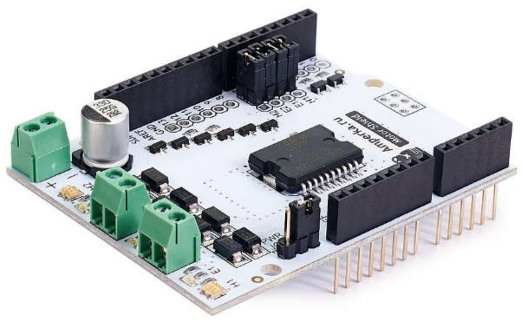


Figure 3.7 Micro DC Gear motor and Amperka Motor Shield

The motors of the platform could not be connected to the microcontroller directly because of two reasons – they often require much more current that the board could provide (up to 2A, while the board's maximum pin current is around 50-100mA), and the board is not protected from the induced current from the motors and could be easily damaged.

The motors I selected for this project are Micro DC Gear motors with 1:48 Gear ratio (Figure 3.7, left). The specification of the motors is shown in Table 3.3. The motors came with plastic wheels with rubber tires, outer diameter 65mm and width of 26mm.

Table 3.3 Micro DC Gear motor specification

Feature	Micro DC Gear motor
Gear Ratio	1:48
No-load speed (5V)	About 208 RPM
Rated Torque (5V)	0.8 Kg*cm
No-load current (5V)	<350 mA

Amperka Motor Shield (Figure 3.7, right) [55] – is an expansion shield capable of controlling motors with voltages of 5-24V in the separate power supply mode and 7-12V in combined power supply mode. The board is based on L298P chip, which has two separate channels, and could control:

1. A pair of DC motors
2. One two-phase step motor
3. One DC motor with current of 4A, if the channels are combined.

Moreover, the board has built-in protection for the inductive current from the motors, which is done by placing diodes in the output line.

The motors are connected by pairs into two separate channels, which makes the control scheme "tank-alike" – to rotate the platform the motors should spin in opposite directions. To achieve maximum performance, the motors wheels should be placed in the corners of the square, this way the platform' rotation would be done around the center axis.

The Motor Shield requires 6-pin connection to the Raspberry Pi 3 board and a separate connection to the power supply. Two of the Raspberry Pi 3 pins should support PWM (Pulse-Width modulation) for controlling the rotation speed of the motors precisely, and another two are used for choosing the direction of the motors.

As Fig.3.8 shows, the connection scheme is as follows:

Motor Shield's pins 4 and 7 are setting the direction of the motors, and are connected to Raspberry Pi 3's pins 29 and 31. Pins 5 and 6 are responsible for motors' speed, and are connected to the pins 32 and 33, which are the two separate PWM channels of Raspberry Pi 3. Ground and +5V pins are connected to each other to provide power to Motor Shield's logic.

Motors 1 and 2, located on the left side of the platform, are connected to the Channel A of the Motor Shield, and motors 3 and 4 connected to the Channel B.

Motor Shield is also connected to the Power supply, which provides voltage and current to the motors.

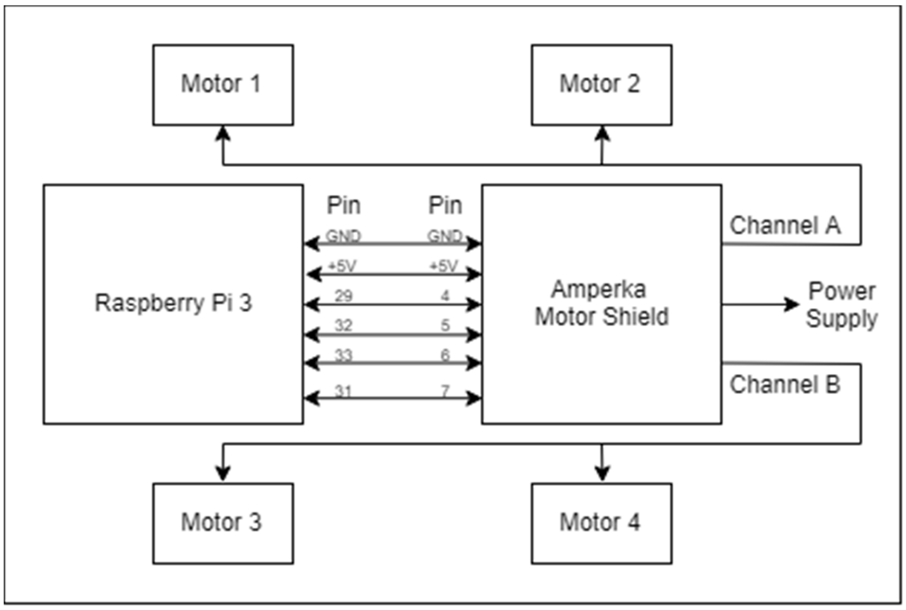


Figure 3.8 Motor Shield, Raspberry Pi 3 and Motors connection scheme

3.4.5. Power Supply

For the robot platform to be truly autonomous it should have an expedient power supply that is capable of handling every component that it is connected to.

As a battery, I chose Lithium-Polymer battery Zippy 30C series 3S1P (Figure 3.9). The specifications contain 11.1V output voltage, a decent capacity of 8000mAh, ability to discharge at 30C (30 * 8000mAh = 240A maximum), low weight of 644 grams and relatively light dimensions of 169x69x27mm. [56]

Figure 3.9. Zippy 30C 8000mAh battery

For the other components of the board the voltage requirements vary, and the Raspberry Pi 3 requires 5V 2.5A, the Microsoft Kinect for Xbox 360 requires 12V 1A and the motors are capable of using from 4V to 9V.

To get the required voltage I used two DC-DC voltage converters, the first one is CPT C240505 Step-Down DC/DC converter (Figure 3.10, right), and the second one is Amperka Troyka-Module Step-Up DC/DC Converter (Figure 3.10, left). The specifications for the both converters are shown in Table 3.4.

Table 3.4 The specifications of the DC/DC converters [57] [58]

Feature	CPT C240505	Amperka Troyka-Module
Integrated Circuit	CPT C240505	Texas Instruments LM27313XMF
Input Voltage	9V-35V	2.7-14V
Output Voltage and Current	5V 5A	5-28V (Fixed at 12V), 1A
Full-load efficiency	>85%	>85%

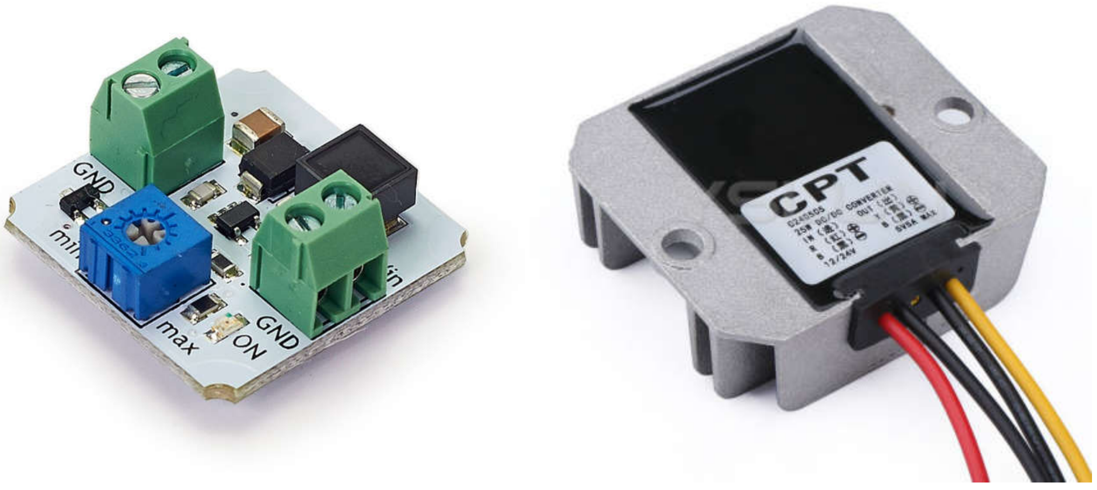


Figure 3.10. Amperka DC/DC converter and CPT DC/DC converter

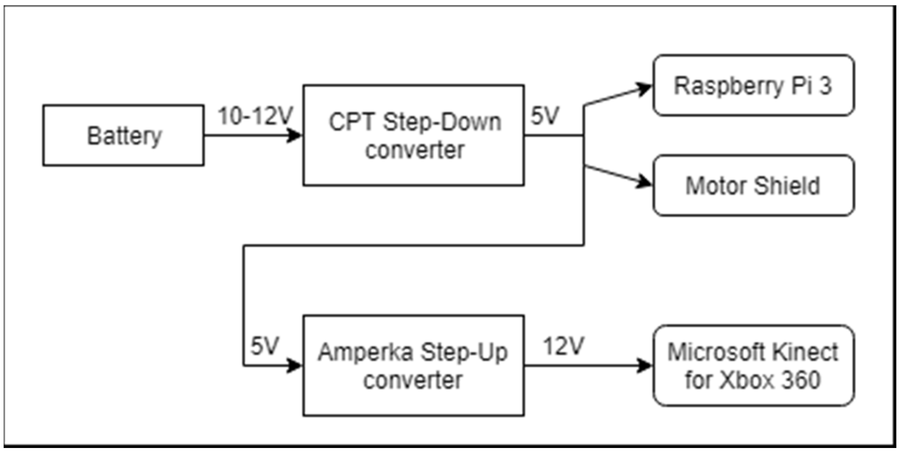


Figure 3.11 Power Supply schematic

On Figure 3.11 shown the electric connection scheme between the components of the platform. The battery is connected to the CPT Step-Down converter, providing stable 5V to the Raspberry Pi 3 microcontroller and to the Motor Shield board, and the CPT Step-Down converter is connected to the Amperka Step-Up converter that outputs the stable 12V to the Microsoft Kinect for Xbox 360.

The connection is done that way due to two reasons: first, the battery will not provide stable voltage due to the basic principles of batteries – the bigger the charge is, the bigger the voltage. The Zippy battery is built from 3 separate power cells, each working in 3.3-4.2 voltage range, and the resulting voltage from the battery could be from 9.9V to 12.4V. Second, the Step-Up converters require the input voltage to be stable to have consistency in output voltage, and also need the input to be at least 1.5V less than output.

3.4.6. Solidworks Model

After measuring the components and analyzing the requirements the prototype was modelled and printed (Figure 3.12).

It could be seen that the engines are located in the corners of the platform and are held by the separate parts, and the battery is placed on top of them. The top surface is made flat for easier placement of any components that should be mounted.

The assembled prototype of the platform could be seen on Figure 3.13. This platform was used in the tests described in the Chapter 4.

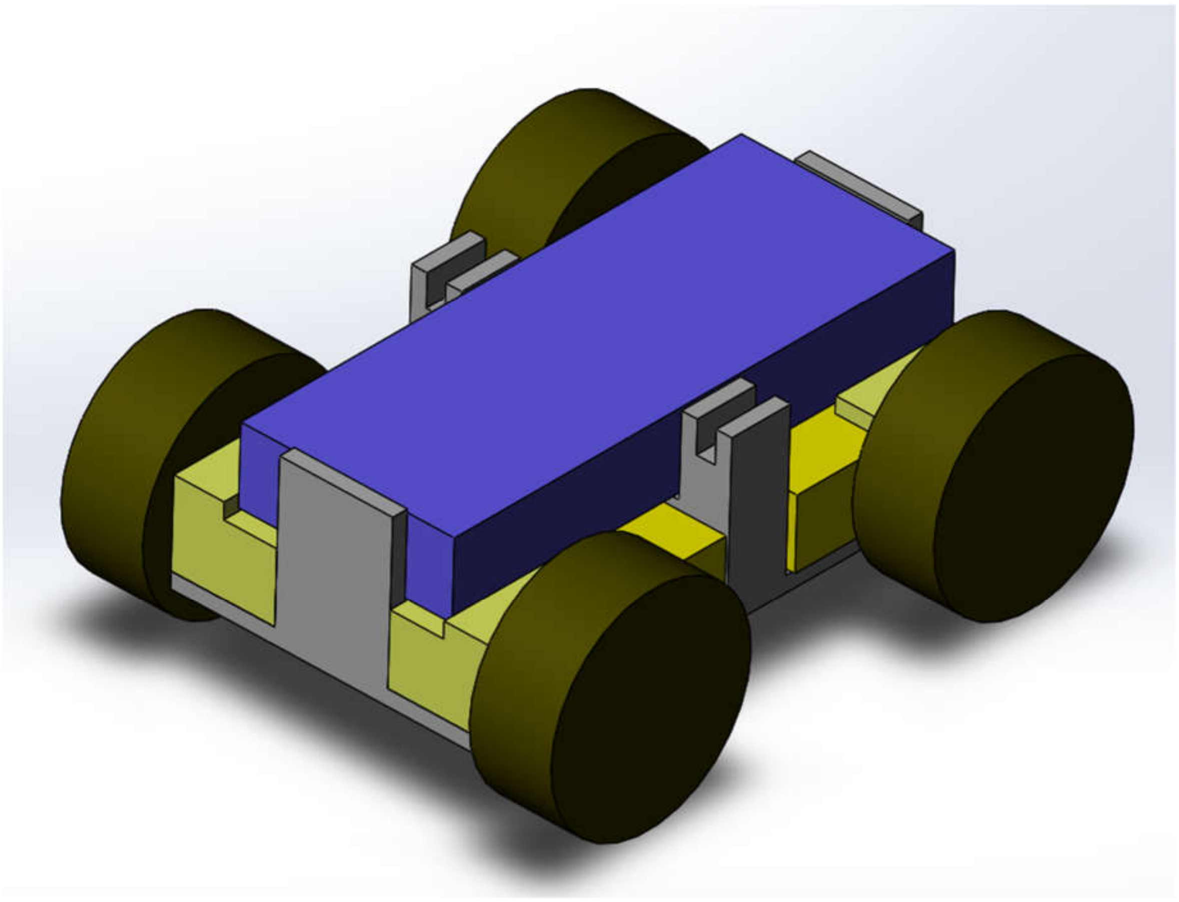


Figure 3.12. The prototype of the platform.

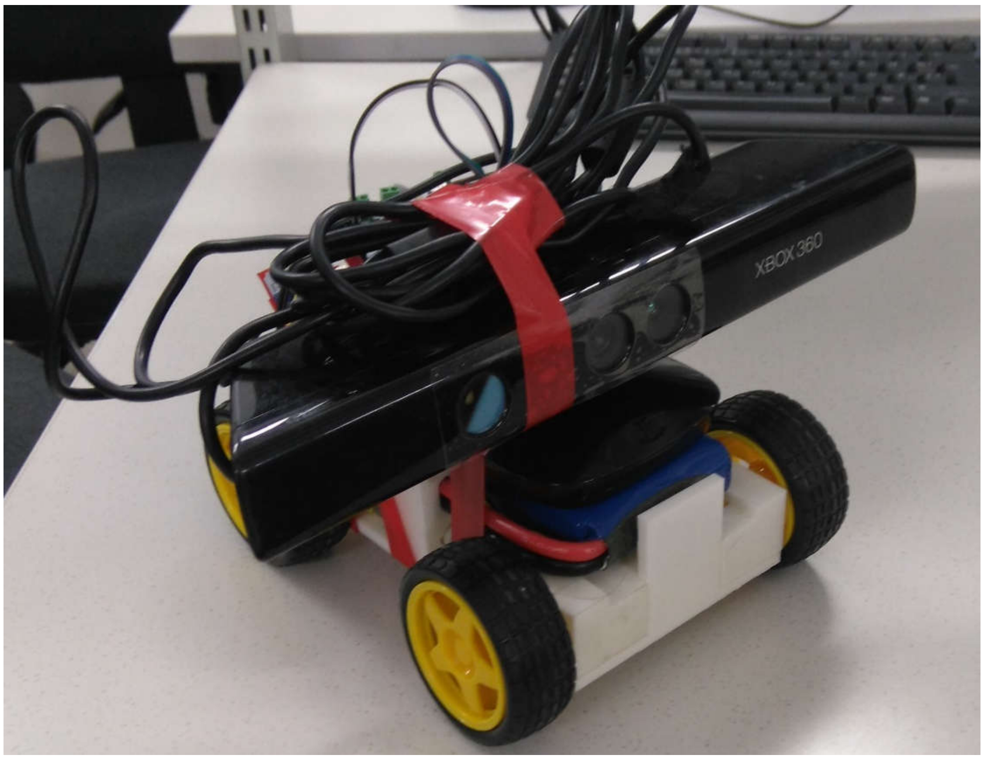


Figure 3.13. The fully assembled prototype of the platform.

3.5. Control programs

I have created two programs for controlling the platform. First one (“Joystick”) is dedicated to establish the connection between the Dualshock 4 gamepad [69] and the laptop PC or Raspberry Pi 3, and the second one (“Control”) for operating the motors of the platform. The programs were separated to achieve the goal of easier modification and upgradeability – the “Control” process could receive any operation commands, if they are correctly formulated and posted to the specific topic, and is independent from the source of these commands, and the “Joystick” process is sending the commands to the topic without knowing, what program is going to receive it and what would it do with these commands. This way, the platform could be controlled from various sources. The communication between the programs is handled by the ROS framework. Both of the programs were written in C++ language.

The “Joystick” program could be launched from both the laptop PC and the Raspberry Pi 3. It utilizes the “joystick” library, explained in [70]. The program connects to the Dualshock 4 gamepad via USB cable, detects any operation with it (e.g. button press, analog stick move, internal accelerator or gyroscope data) and posts the information about it in ROS network in topic “/Joy” with the message type “joy::joy”, which is explained in the following paragraph.

The “joy::joy” message type contains the information about all the buttons pressed and axes moved. It was written to ease the ability of incorporating any specific buttons and axes in any program. Every activity from the Dualshock 4 is posted into the separate stream, e.g. the event of pressing X could be received from “message.btn1”, and the movement of the right stick is obtainable from “message.axis1” and “message.axis0” for X-axis and Y-axis accordingly.

The “Control” program is launched on the Raspberry Pi 3. It utilizes the “wiringPi” library for connecting to the Motor Shield and controlling the motors. Firstly, it establishes the connection to the Motor Shield utilizing the GPIO pins, which were shown in Figure 3.8. Then, it starts receiving data from the “/Joy” topic in “joy::joy” format and filters from it the information about left D-Pad and left Analog stick. After that the control data is converted to the type which is suitable for the Motor Shield and is sent to it.

This program supports both so-called “Digital” and “Analog” movement. In “Digital” mode the motors are always working on the constant rate, when the buttons are pressed, and provide full throttle. In “Analog” mode the motors could be handled more precisely, allowing the 1024 positions between the full stop and full throttle.

CHAPTER 4

4. RESULTS

4.1. Evaluated characteristics

The comparison of the algorithms was made by the following criteria:

1. Time required to set up; The time required to compile, install and configure the algorithm.
2. Effective framerate of the algorithm; How fast the algorithm updates the map, the time between the updates.
3. Overall map quality; How accurate the map compared to the test location.
4. Computation requirements; How computation-hungry the algorithm is, in terms of the CPU usage on the platform.

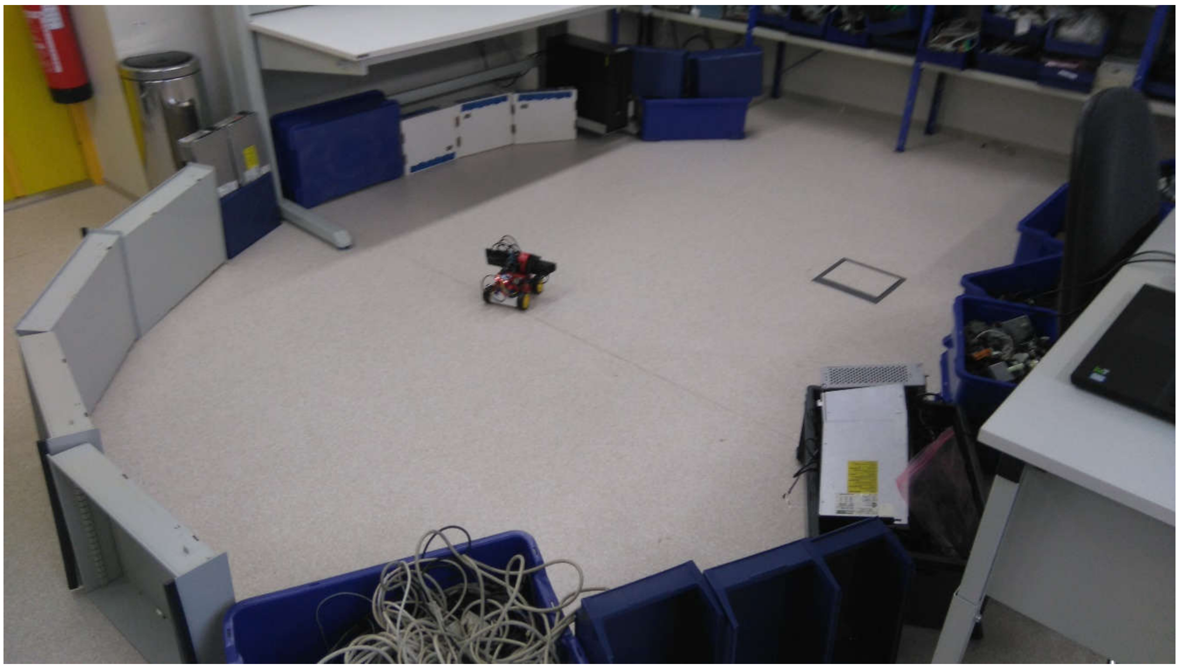


Figure 4.1. The photograph of the testing location

On the Figure 4.1 could be seen the test location that was used to evaluate the results.

4.2. Analysis of the results

Table 4.1. Table of the results

Parameter	Hector_slam	Gmapping	ORB_SLAM2	RGBDSLAMv2
Time required	2 hours	3 hours	4 hours	Not compiled
Framerate	3 frames per second	6 frames per second	1 frame per 3 seconds	N/A
Map quality	Fine	Average	Bad	N/A
Computation	60% of the CPU	70% of the CPU	100% of the CPU	N/A

The Table 4.1 contains the results of the algorithms, which would be explained in this chapter.

1. **Time required to setup.**

Hector_slam package was relatively easy to compile and build and it took 2 hours from the start.

Gmapping, on the other hand, required a few libraries to be built before itself, and there were troubles in compilation, and after all the dependencies were satisfied and the compilation took around 3 hours.

The ORB_SLAM2 package was the easiest to compile, as it comes with all the dependencies and the installation script that checkes everything and prompts messages if something is wrong. The compilation of all of the dependencies and the package itself took nearly 4 hours on the Raspberry Pi 3.

The RGBDSLAMv2 package showed the worst results. For the successful build it requires a specially-configured library to be installed first, and that library requires compilation and installation of system-based packages like QT5, Python3 and GPP. Sadly, QT5 is not supported by the ARM CPU of the Raspberry Pi 3, and that made the package impossible to install and use at all.

2. **Effective framerate.**

Hector_slam package updated the map on an average of 3 frames per second with the minimal framerates of 1 frame per 10 seconds.

Gmapping package was providing from 3 to 10 frames per second in map-building mode, with the average framerate of 6 frames per second. There were frame drops for the maximum of 5 seconds waiting for the next frame, but they were happening rarely.

ORB_SLAM2 package provided one frame in roughly 3 seconds, and never achieved more than 1 frame per second.

3. Map quality.

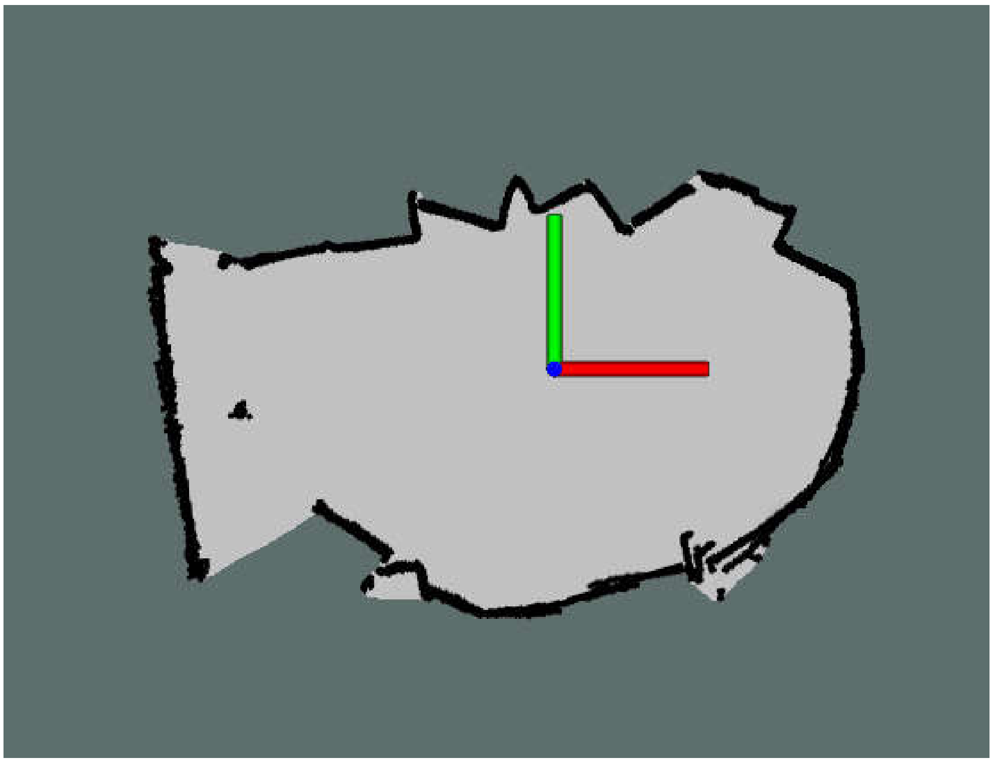


Figure 4.2. The map built by hector_slam package

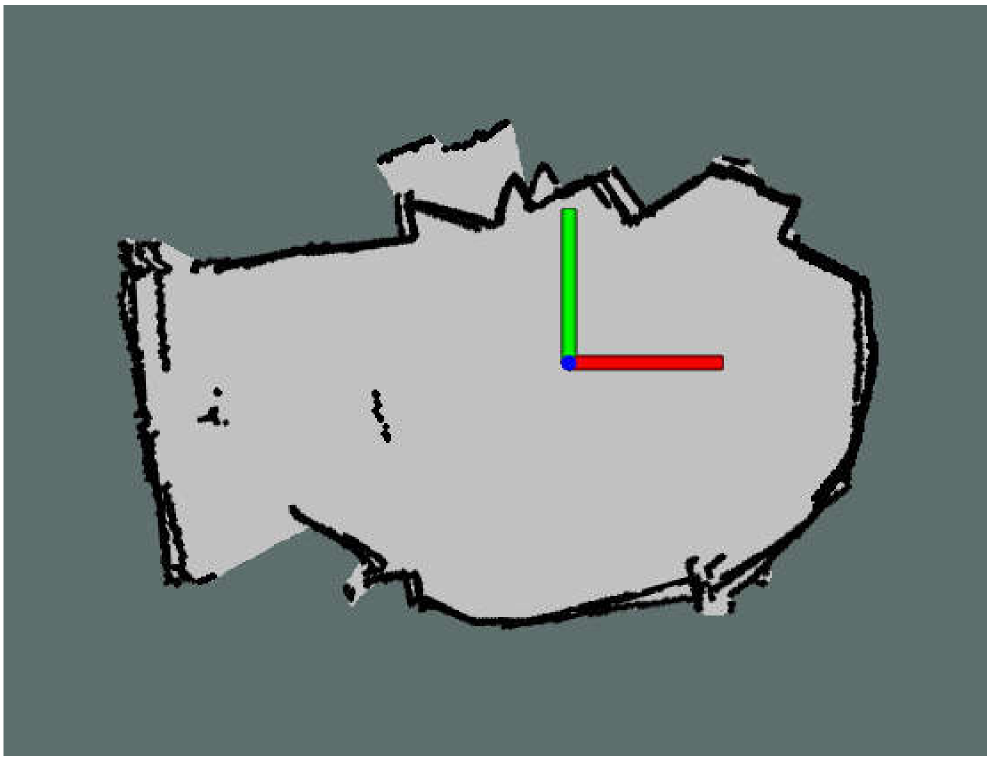
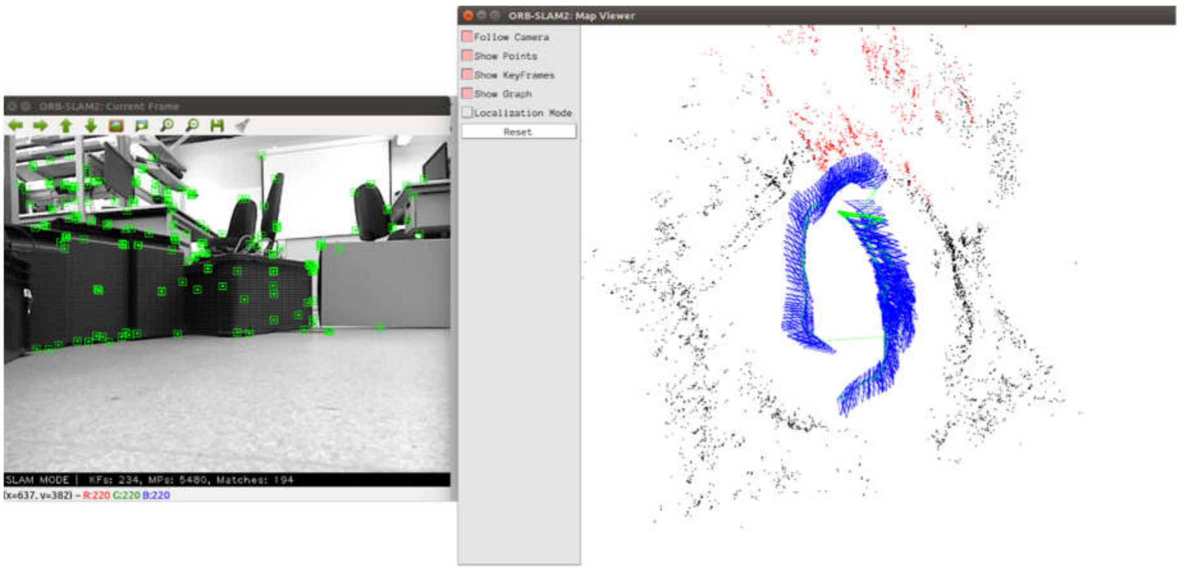
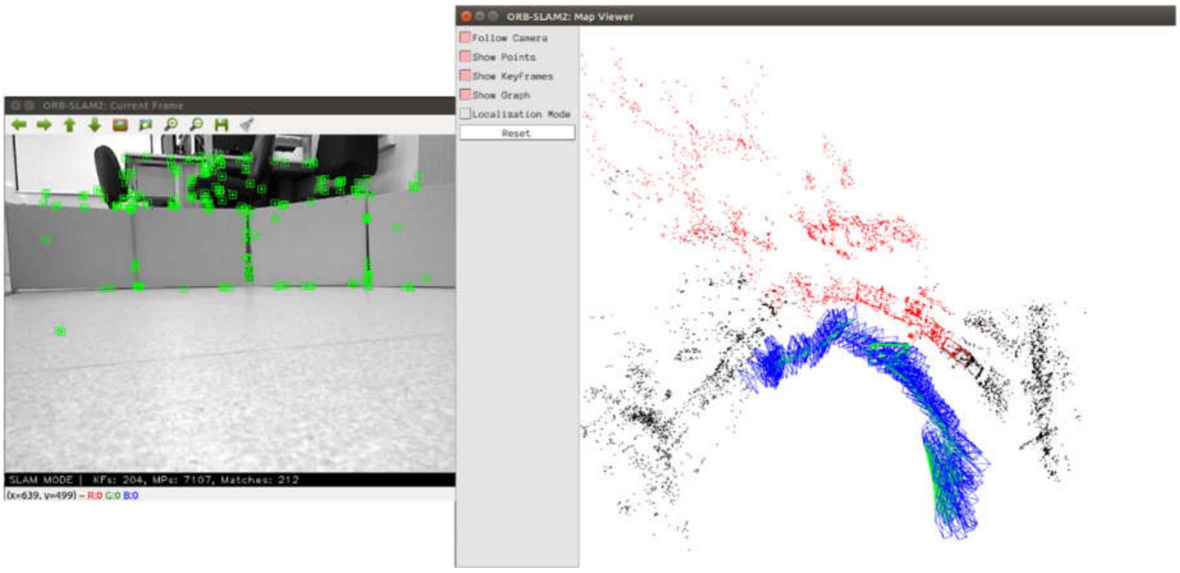


Figure 4.3. The map built by Gmapping package



```
SLAM MODE |  KFs: 234, MPs: 5480, Matches: 194
(x=637, y=382) ~ R:220 G:220 B:220
```

Figure 4.4. The map built by the ORBSLAM2 package, result 1



```
SLAM MODE |  KFs: 204, MPs: 7107, Matches: 212
(x=639, y=499) ~ R:0 G:0 B:0
```

Figure 4.5. The map built by the ORBSLAM2 package, result 2

As seen on Figure 4.2, the hector_slam package has created the best map of the three algorithms. The map has a few misaligned walls, but the overall quality is very good, the shape of the test location could be easily described and there are minimum of the false points on the map.

The Gmapping package (Figure 4.3) showed worse results, but the shape of the test location could be described as well. There are a lot of misaligned walls and false points, but they are mostly located at the edges of the map.

The ORBSLAM2 package showed the worst results due to the computation constraints of the Raspberry Pi 3. While the algorithm was capable of finding keypoints and detecting features, the extreme low framerates led to the very bad quality of the map and incorrect estimated camera positions. (Figures 4.4 and 4.5)

4. Computation requirements

Both hector_slam and Gmapping packages showed the CPU load around the 60-70%, with Gmapping using slightly more due to the laser_scan_matcher process.

The ORBSLAM2 reached 100% of the CPU usage almost instantly, showing that it requires much more performance that the Raspberry Pi 3 could provide.

4.3. Conclusion

After comparing and analyzing the results the following conclusions could be made:

1. The hector_slam provided the best results in the comparison. It was easy to set up, the performance requirements are average and the map built by this algorithm was good.
2. The Gmapping package was faster at map building, but the resulting map contained errors and misaligns.
3. ORBSLAM2 requires much more computation capabilities than the Raspberry Pi 3 has, and due to that the resulting maps were poor quality. It is not recommended to use this algorithm on the mobile robot.

CHAPTER 5

5. SUMMARY

5.1. Summary

In this thesis were described the steps required for implementation of SLAM algorithms from choosing the sensors to designing the platform, setting up the algorithms and analyzing the results. During the work the following goals were obtained:

1. The overview and comparison of the sensors were made;

   An overview of the types of sensors, suitable for Visual SLAM, was presented. The RGBD cameras were chosen as the best solution, because compared to the mono and stereo camera setups, they could provide the depth information about the environment of the platform without the need for detecting and comparing keypoints on consecutive frames. The LIDAR sensors were considered as the possible solution, but were rejected due to the bigger expenses for comparable results. As the RGBD sensor the Microsoft Kinect 360 camera was selected, as it provides sufficient accuracy of depth data, has a lot of support in the community and is easy to obtain, implement and use.

2. The prototype of the platform was created and assembled;

   The platform was designed in Solidworks software and was printed on a 3D printer. It has enough surface for mounting the hardware and is able to move around without wired connections.

3. The control program for the platform was created;

   The program is capable of receiving control commands from any source in the specific format, and is able to control the platform's motors precisely in analog and digital modes.

4. The overview and comparison of the SLAM algorithms were made and analyzed.

   The algorithms "hector_slam", "Gmapping" and "ORBSLAM2" were implemented on the platform, and the comparison between them was made. The criteria contained the time required to set up the algorithms, the performance characteristics and computation usage, and the built map quality. The "hector_slam" algorithm was chosen as a preferred one, because it created a map with the minimum amount of the false points, used less CPU time than the other algorithms and was the easiest to set up and configure.

It was shown, that the usage of Raspberry Pi 3 and the Microsoft Kinect 360 camera is a viable solution for the implementation of 2D SLAM algorithms. The 3D SLAM algorithms require much more computation capacities and are not recommended for the mobile platforms based on the low-performance microcontrollers.

5.2. Kokkuvõte

See lõputöö kirjeldab samme, mis tuleb läbida autonoomse robotplatvormi ehitamisel, alates andurite valikust kuni platvormi konstruktsioonini, algoritmide valikuni ja tulemuste analüüsini. Töö käigus jõuti järgmiste tulemusteni:

1. Töös esitati erinevate andurite ülevaade ja võrdlus;
   Anti ülevaade erinevates anduritüüpidest, mis sobivad „Visual SLAM“ meetodile. Parimaks valikuks osutus RGBD tüüpi kaamera, sest erinevalt mono- ja stereokaameratest võimaldab see edastada sügavusinformatsiooni ilma kuluka võtmepunktide kaevandamise ning kaadrist kaadrisse erinevuste võrdlemise. Kaaluti ka LIDARi kasutamist, kuid sellest loobuti suure hinna tõttu võrreldava tulemuse saamiseks. Konkreetselt kasutati töös Microsoft Kinect 360 RGBD andurit, kuna see edastab piisava täpsusega sügavusinformatsiooni, on kergesti kättesaadav, on laialdaselt toetatud ja kergesti integreeritav.
2. Loodi platvormi prototüüp;
   Platvorm konstrueeriti Solidworks keskkonnas ja valmistati 3D printeri abil. Sellel on piisavalt ühenduspunkte andurite, aku ja kontrolleri kinnitamiseks ning on võimeline juhtmevabalt ringi liikuma.
3. Loodi platvormi juhtprogramm;
   Programm võtab ettemääratud formaadis liikumiskäske suvalisest allikast ning suudab mootoreid juhtida nii proportsionaalses kui ka lülitirežiimis.
4. Sai tehtud ülevaade ja võrdlus erinevatest SLAM algoritmidest.
   Algoritmid „Hector SLAM“, „Gmapping“ ja „ORBSLAM2“ said rakendatud ka nimetatud platvormil ja tehtud sai ka võrdlev analüüs. Võrreldi algoritmi ülesseadmiseks kuluvat aega, algoritmide jõudluskarakteristikuid, arvutusressursside kasutust ja loodud kaardi kvaliteeti. Analüüsi tulemusel osutus valituks „Hector SLAM“ algoritm, kuna see lõi vähima müraga kaardi kasutades vähem arvutusvõimsust ning oli lihtne ülesseada.

Näidati, et Raspberry Pi 3 avuti ja Microsoft Kinect 360 kaamera on sobilik valik 2D SLAM algoritmi teostuseks. 3D SLAM algoritmid nõuavad oluliselt rohkem arvutusvõimsust ning autor ei soovita nende kasutamist mobiilsetel platvormidel, mis utiliseerivad madalavõimsuselist mikrokontrollerit.

LIST OF REFERENCES

1. Y. Abdelrasoul, A. B. Sayuti HM Saman, P. Sebastian, "A Quantitative Study of Tuning ROS Gmapping Parameters and Their Effect on Performing Indoor 2D SLAM" Electrical and Electronic Engineering department, Universiti Teknologi PETRONAS
2. Leonard, J.J.; Durrant-whyte, H.F. (1991). "Simultaneous map building and localization for an autonomous mobile robot", 'Intelligent Robots and Systems' 91. 'Intelligence for Mechanical Systems, Proceedings IROS' 91. IEEE/RSJ International Workshop on: 1442–1447
3. B. Hiebert-Treuer, "An Introduction to Robot SLAM (Simultaneous Localization And Mapping)"
4. Smith, R.C.; Cheeseman, P. (1986). «On the Representation and Estimation of Spatial Uncertainty». The International Journal of Robotics Research 5 (4): 56–68.
5. Smith, R.C.; Self, M.; Cheeseman, P. (1990). «Estimating Uncertain Spatial Relationships in Robotics», Proceedings of the Second Annual Conference on Uncertainty in Artificial Intelligence. UAI '86. University of Pennsylvania, Philadelphia, PA, USA: Elsevier. pp. 435–461.
6. Leonard, J.J.; Durrant-whyte, H.F. (1991). "Simultaneous map building and localization for an autonomous mobile robot", 'Intelligent Robots and Systems' 91. 'Intelligence for Mechanical Systems, Proceedings IROS' 91. IEEE/RSJ International Workshop on: 1442–1447
7. M. Csorba and H.F. Durrant-Whyte. "A new approach to simultaneous localization and map building". In Proceedings of SPIE Aerosense, Orlando, 1996.
8. M. Csorba. "Simultaneous Localization and Map Building". PhD thesis, University of Oxford, 1997.
9. A. Meguenani, "Onboard Vision based SLAM on a Quadruped Robot", Istituto Italiano di Tecnologia, Genova, Italy
10. F. Hjelmare, J. Rangsjö, "Simultaneous Localization And Mapping Using a Kinect™ In a Sparse Feature Indoor Environment", Linköping 2012
11. H. Moravec (1980). "Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover". Tech Report CMU-RI-TR-3 Carnegie-Mellon University, Robotics Institute.
12. C. Harris and M. Stephens. "A Combined Corner and Edge Detector." In Proceedings of the 4th Alvey Vision Conference, pages 147–151, 1988
13. Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus H. Gross. "Surfels: surface elements as rendering primitives". In SIGGRAPH, pages 335–342, 2000.
14. Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. "Speeded-Up Robust Features (SURF)". Comput. Vis. Image Underst., 110:346–359, June 2008.
15. B. Steder, R. B. Rusu, K. Konolige, and W. Burgard. "NARF: 3D Range Image Features for Object Recognition". In Workshop on Defining and Solving Realistic Perception Problems in

Personal Robotics at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), Taipei, Taiwan, 2010.

16. M. Calonder, V. Lepetit, C. Strecha, and P. Fua. "BRIEF: Binary Robust Independent Elementary Features". In European Conference on Computer Vision, September 2010.
17. Rosten, Edward; Tom Drummond (2005). "Fusing points and lines for high performance tracking" (PDF). IEEE International Conference on Computer Vision. 2: 1508–1511. doi:10.1109/ICCV.2005.104.
18. Corner Detection (Online) https://en.wikipedia.org/wiki/Corner_detection
19. Taylor, Brook (1715). "Methodus Incrementorum Directa et Inversa [Direct and Reverse Methods of Incrementation]" (in Latin). London. p. 21–23 (Prop. VII, Thm. 3, Cor. 2). Translated into English in Struik, D. J. (1969). A Source Book in Mathematics 1200–1800. Cambridge, Massachusetts: Harvard University Press. pp. 329–332.
20. J. Shi and C. Tomasi (June 1994). "Good Features to Track,". 9th IEEE Conference on Computer Vision and Pattern Recognition. Springer.
21. C. Tomasi and T. Kanade (2004). "Detection and Tracking of Point Features". Pattern Recognition. 37: 165–168. doi:10.1016/S0031-3203(03)00234-6
22. A. Noble (1989). "Descriptions of Image Surfaces (Ph.D.)". Department of Engineering Science, Oxford University. p. 45.
23. Serre, T., Kouh, M., Cadieu, C., Knoblich, U., Kreiman, G., Poggio, T., "A Theory of Object Recognition: Computations and Circuits in the Feedforward Path of the Ventral Stream in Primate Visual Cortex", Computer Science and Artificial Intelligence Laboratory Technical Report, December 19, 2005 MIT-CSAIL-TR-2005-082.
24. Beis, J.; Lowe, David G. (1997). "Shape indexing using approximate nearest-neighbour search in high-dimensional spaces". Conference on Computer Vision and Pattern Recognition, Puerto Rico: sn. pp. 1000–1006. doi:10.1109/CVPR.1997.609451.
25. Lowe, David G. (2004). "Distinctive Image Features from Scale-Invariant Keypoints". International Journal of Computer Vision. 60 (2): 91–110. doi:10.1023/B:VISI.0000029664.99615.94.
26. Lowe, D.G., "Local feature view clustering for 3D object recognition". IEEE Conference on Computer Vision and Pattern Recognition,Kauai, Hawaii, 2001, pp. 682-688.
27. Scale-invariant feature transform (Online) https://en.wikipedia.org/wiki/Scale-invariant_feature_transform
28. Mikolajczyk, K. and Schmid, C. 2002. An affine invariant interest point detector. In Proceedings of the 8th International Conference on Computer Vision, Vancouver, Canada.
29. Lindeberg, Tony. "Feature detection with automatic scale selection", International Journal of Computer Vision, 30, 2, pp. 77-116, 1998.
30. Speeded up robust feature (Online) https://en.wikipedia.org/wiki/Speeded_up_robust_features
31. Brown, M. and Lowe, D., 2002. Invariant Features from Interest Point Groups. In: BMVC 2002: 13th British Machine Vision Conference, 2002-09-02 - 2002-09-05.

32. Michael Calonder. Robust, "High-Speed Interest Point Matching for Real-Time Applications" (PhD Thesis), 2010.
33. Tony Lindeberg. "Scale-Space Theory in Computer Vision". Kluwer Academic Publishers, Norwell, MA, USA, 1994.
34. B. Steder, R. B. Rusu, K. Konolige, W. Burgard. "Point Feature Extraction on 3D Range Scans Taking into Account Object Boundaries"
35. Features from accelerated segment test (online) https://en.wikipedia.org/wiki/Features_from_accelerated_segment_test
36. S. Thrun, W. Burgard, and D. Fox, "Probabilistic robotics". MIT Press, 2008.
37. B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision (darpa)" in DARPA Image Understanding Workshop, Apr 1981, pp. 121–130.
38. J. M. Santos, D. Portugal, and R. P. Rocha, "An evaluation of 2D SLAM techniques available in Robot Operating System," in 2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), 2013, pp. 1–6
39. G. Grisetti, C. Stachniss, and W. Burgard, "Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters" IEEE Trans. Robot., vol. 23, no. 1, pp. 34–46, Feb. 2007.
40. A. Doucet, N. de Freitas, and N. Gordan, editors. "Sequential MonteCarlo Methods in Practice". Springer Verlag, 2001
41. A. Doucet. "On sequential simulation-based methods for bayesian filtering". Technical report, Signal Processing Group, Dept. of Engeneering, University of Cambridge, 1998
42. M. Montemerlo, S. Thrun D. Koller, and B. Wegbreit. "FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges". In Proc. of the Int. Conf. on Artificial Intelligence (IJCAI), pages 1151–1156, Acapulco, Mexico, 2003
43. ORB-SLAM Project Webpage (Online) http://webdiis.unizar.es/~raulmur/orbslam/
44. R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: a versatile and accurate monocular SLAM system" IEEE Transactions on Robotics, vol. 31, no. 5, pp. 1147–1163, 2015
45. L. M. Paz, P. Pinies, J. D. Tardos, and J. Neira, "Large-scale 6-DOF SLAM with stereo-in-hand," IEEE Transactions on Robotics, vol. 24, no. 5, pp. 946–957, 2008.
46. Raúl Mur-Artal, and Juan D. Tardós. "ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras". ArXiv preprint arXiv:1610.06475, 2016
47. RGBDSLAM (Online) http://wiki.ros.org/rgbdslam
48. Martin A. Fischler and Robert C. Bolles (1981), "Random Sample Consensus: A Paradigm for Model Fitting with Apphcatlons to Image Analysis and Automated Cartography", SRI International
49. Joshua S., "Utilizing Robot Operating System (ROS) in robot vision and control", Monterey, California
50. ROS Packages (Online) http://wiki.ros.org/Packages
51. ROS Concepts (Online) http://wiki.ros.org/ROS/Concepts

52. QUICK REFERENCE: KINECT 1 VS KINECT 2 (Online) http://www.imaginativeuniversal.com/blog/2014/03/05/Quick-Reference-Kinect-1-vs-Kinect-2/
53. Raspberry Pi 3 Model B Technical Specifications (Online) https://www.element14.com/community/docs/DOC-80899/l/raspberry-pi-3-model-b-technical-specifications#
54. Raspberry Pi 3, Banana Pi M3, Orange Pi Plus 2, ODROID C2 Spec Comparison (Online) https://www.loverpi.com/blogs/news/94801153-raspberry-pi-3-banana-pi-m3-orange-pi-plus-2-odroid-c2-spec-comparison
55. Motor Shield [Amperka / Wiki] (Online) http://wiki.amperka.ru/продукты:motor-shield
56. ZIPPY Flightmax 8000mAh 3S1P 30C Lipo Pack (Online) https://hobbyking.com/en_us/zippy-flightmax-8000mah-3s1p-30c-lipo-pack.html
57. CPT / Fulree DC-DC converters (Online) http://www.current-logic.com/dcdc_converter_cpt_fulree.php
58. Повышающий стабилизатор напряжения (Troyka-модуль) (Online) http://amperka.ru/product/troyka-dc-dc-booster
59. ROS Kinetic Installation (Online) http://wiki.ros.org/kinetic/Installation/Ubuntu
60. S. Kohlbrecher and J. Meyer and O. von Stryk and U. Klingauf (2011), "A Flexible and Scalable SLAM System with Full 3D Motion Estimation", Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)
61. F. Lu and E. Milios. "Globally consistent range scan alignment for environment mapping". Journal of Autonomous Robots, 4:333–349, 1997.
62. F. Endres, J. Hess, J. Sturm, D. Cremers, W. Burgard, "3D Mapping with an RGB-D Camera", IEEE Transactions on Robotics, 2014.
63. L. Joseph, "Learning Robotics Using Python", Birmingham, UK: Packt Publishing, 2015, pp. 56-57.
64. J.J. Leonard and H.J.S. Feder, "A computational efficient method for large-scale concurrent mapping and localisation," in Robotics Research, The Ninth International Symposium (ISRR'99), J. Hollerbach and D. Koditscheck, Eds. New York: Springer-Verlag, pp. 169–176, 2000.
65. OpenNI Programmers Guide (pdf) https://s3.amazonaws.com/com.occipital.openni/OpenNI_Programmers_Guide.pdf
66. Depthimage to laserscan package (Online) http://wiki.ros.org/depthimage_to_laserscan
67. Laser Scan Matcher package (Online) http://wiki.ros.org/laser_scan_matcher
68. Camera Calibration package (Online) http://wiki.ros.org/camera_calibration
69. Sony Dualshock 4 Wireless Controller (Online) https://www.playstation.com/ru-ru/explore/accessories/dualshock-4-wireless-controller/
70. Joystick API by R.H. Espinosa, 1998 (Online) https://www.kernel.org/doc/Documentation/input/joystick-api.txt