BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

AUTONOMOUS ROVER NAVIGATION ON PLANETARY SURFACE

AUTONOMNÍ NAVIGACE VOZIDLA NA POVRCHU PLANETY

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **MAREK VAŠKO**
AUTOR PRÁCE

SUPERVISOR **doc. Ing. PETER CHUDÝ, Ph.D. MBA**
VEDOUCÍ PRÁCE

BRNO 2020

Brno University of Technology
Faculty of Information Technology

Department of Computer Graphics and Multimedia (DCGM) Academic year 2019/2020

Bachelor's Thesis Specification

Student: **Vaško Marek**
Programme: Information Technology
Title: **Autonomous Rover Navigation on Planetary Surface**
Category: Modelling and Simulation

Assignment:

1. Research historical evolution of planetary exploration rovers and get familiar with their autonomous navigation principles.
2. Research a suitable algorithm for rover navigation using sensors available to a planetary rover.
3. Perform Matlab implementation of the researched rover navigation solution.
4. Implement the navigation solution into a suitable planetary simulator.
5. Evaluate achieved results and discuss potential further improvements.

Recommended literature:

- According to supervisor's recommendations.

Requirements for the first semester:

- Items No. 1, 2 and partially item No. 3.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor: **Chudý Peter, doc. Ing., Ph.D. MBA**
Head of Department: Černocký Jan, doc. Dr. Ing.
Beginning of work: November 1, 2019
Submission deadline: July 31, 2020
Approval date: July 23, 2020

Abstract

Exploration of the depths of the space has led to the development of technology in various fields. One of these areas is the exploration of the surface of extraterrestrial planets. An unmanned ground vehicle is an effective way of exploration. This thesis deals with one of the most important systems of unmanned vehicles, which is autonomous navigation. The vehicle must be able to navigate in the environment and map potential obstacles. The thesis will examine the navigation principles that have been used by existing vehicles. It will then research the use of the algorithm based on the principle of simultaneous localization and mapping and its implementation in MATLAB. This algorithm will be integrated into the simulator, which will allow later integration into the real environment using the Robot Operating System. A vehicle platform with simulated sensors and a six-wheel chassis, which will be used for an integrated algorithm, will be designed in the simulator. Finally, the quality of the proposed algorithm is evaluated and a discussion about future improvements is initiated.

Abstrakt

Výskum hlbín vesmíru doviedol k vývoju technológii v rôznych oblastiach. Jednou z týchto oblastí je prieskum povrchu mimozemských planét. Efektívny spôsob skúmania je bezpilotné pozemné vozidlo. Práca sa zaoberá jedným z najdôležitejších systémov bezpilotných vozidiel, čím je autonómna navigácia. Vozidlo sa musí vedieť orientovať v priestore a zmapovať potenciálne prekážky. Práca v úvode preskúma navigačné princípy, ktoré boli využívané existujúcimi vozidlami. Neskôr preskúma využitie algoritmu na princípe súčasnej lokalizácie a mapovania a jeho implementáciu v MATLAB-e. Tento algoritmus bude integrovaný do simulátora, ktorý umožní neskoršiu integráciu do reálneho prostredia pomocou Robot Operating System. V simulátore bude navrhnutá platforma vozidla so simulovanými senzormi a šesť-kolesovým podvozkom, ktorá bude slúžiť na testovanie integrovaného algoritmu. V závere sa vyhodnotí kvalita navrhnutého algoritmu a zaháji sa diskusia o budúcich vylepšeniach.

Keywords

Autonomous navigation, Gazebo, localization, LiDAR, Mars, mapping, NASA, RGB-D, ROS, simulation, SLAM, unmanned rover

Klíčová slova

Autonómna navigácia, bezpilotné vozidlo, Gazebo, lokalizácia, LiDAR, Mars, mapovanie, NASA, RGB-D, ROS, simulácia, SLAM

Reference

VAŠKO, Marek. *Autonomous Rover Navigation on Planetary Surface.* Brno, 2020. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor doc. Ing. Peter Chudý, Ph.D. MBA

Rozšířený abstrakt

Misie mimo dosah orbity Zeme určené pre hľadanie nových potenciálnych miest pre život boli hlavným cieľom prieskumu vesmíru. Prvá medziplanetárna misia smerom k Venuši bola vyslaná už v šesťdesiatych rokoch minulého storočia. Toto bol zlomový bod prieskumných misií, ktoré začali minulým storočím a pokračujú do súčasnosti. Náročné prostredie mimozemských planét sťažuje prieskum s posádkou. Bezpilotné misie poskytujú kompromis v potenciálnych ziskoch a rizikách. Cieľom každej z týchto misií je dosiahnuť čo najviac vedeckých pokrokov.

Jedným zo spôsobov prieskumu povrchu planét je použitie pristávacích modulov. Platformou, ktorá poskytuje väčšiu flexibilitu ako pristávací modul, je bezpilotné vozidlo. Skúmanie pomocou bezpilotných vozidiel prináša svoje vlastné výzvy. Jedna z najväčších je lokalizácia a vyhýbanie sa potenciálnym prekážkam.

Planéta Zem má vybudovanú infraštruktúru určenú pre navigáciu. Služby, ako GPS, poskytujú pokrytie takmer všetkých miest na povrchu. Vďaka tomu je možné získať informáciu o presnej polohe v reálnom čase. Presné mapy poskytujú podrobný popis každého možného terénu. Misia skúmajúca inú planétu môže byť prvou, ktorá kedy pristála na povrchu vzdialenom milióny kilometrov od Zeme.

Dôležitosť riešenia tohto problému navigácie narastá, v prípade misií s časovo limitovanou dobou trvania. V týchto prípadoch je dôležité preskúmať čo najväčšie územie v čo najkratšom čase. Vozidlo je závislé od informácii, ktoré je schopné samostatne získať a spracovať. Operátor nemôže zasiahnuť do riadenia vozidla v reálnom čase, tým pádom je potrebné vytvárať autonómne mapu okolitého prostredia a navigovať sa na nej.

V posledných rokoch došlo k veľkému vývoju smerom k rôznym riešeniam navigácie. Tieto riešenia sú navrhnuté tak, aby neboli potrebné žiadne ďalšie navigačné údaje. Využívajú sa hlavne komerčne dostupné senzory, čím táto forma navigácie nadobúda širšie uplatnenie. Misie prieskumných vozidiel môžu tieto algoritmy taktiež efektívne využívať.

Existujúce misie používali vlastné riešenia, ktorým sa v práci budem podrobnejšie venovať. Prehľad riešení sa zamerá na históriu prieskumných vozidiel a vysvetlí použité navigačné princípy ako sú vizuálna odometria a odometer. Senzory ktoré boli využívané pre navigáciu a princípy ich fungovania budú taktiež detailnejšie popísané.

Algoritmus, ktorý využíva existujúce komerčne dostupné snímače, je hlavnou časťou tejto práce. Tento algoritmus využije existujúce riešenie pre súčasnú lokalizáciu a mapovanie. Jeho výsledkom bude mapa a poloha vozidla. Poloha a mapa sa získa z viacerých senzorov ako sú odometer, stereoskopická kamera, hĺbková kamera a LiDAR. Algoritmus bude implementovaný v MATLAB-e.

V práci bude navrhnutá simulovaná platforma vozidla, využívaná pre testovanie a integráciu algoritmu. Hlavnou inšpiráciou pri návrhu tejto platformy je prvá misa na povrchu Marsu s bezpilotným vozidlom Sojourner. Vozidlo bude mať šesť-kolesový podvozok, modely senzorov so simulovanými výstupnými dátami a 3D modely iných častí ako je solárny panel a rám. Ovládanie vozidla bude naprogramované v Python-e a princíp ovládania bude v práci vysvetlený.

Integrácia skúmaného algoritmu do simulátoru bude pomocou Robot Operating System, ktorý umožní neskoršiu integráciu do reálneho prostredia. V simulačnom prostredí Gazebo bude navrhnutá testovacia scéna simulujúca podmienky na povrchu Marsu. Táto scéna bude obsahovať rôzne prekážky. Navrhnuté vozidlo sa bude pohybovať po zvolenej trase pomedzi prekážky a získa dáta o svojom okolí, ktoré budú spracované do mapy. Výsledky práce budú štatisticky zhodnotené a určí sa najlepšia kombinácia využitých senzorov.

Autonomous Rover Navigation on Planetary Surface

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of doc. Ing. Peter Chudý Ph.D. MBA. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Marek Vaško
July 30, 2020

Acknowledgements

I would like to thank my supervisor doc. Ing. Peter Chudý Ph.D. MBA. for guidance, professional help and comments on the researched subject which helped to reach desired goals of this thesis. A special thanks comes to Alfredo C. Plascencia Ph.D. for providing a professional help with the Robot Operating System.

Contents

List of Figures

List of Tables

Abbreviations

CAD	Computer Aided Design
CCD	Charge-coupled device
ECEF	earth-centered, earth-fixed
ENU	East North Up
FOV	Field of View
GNC	Guidance Navigation Control
GPS	Global Positioning System
GT	Ground-truth
IMU	Inertial Measurement Unit
IR	Infra-Red
LiDAR	Light Detection and Ranging
MAHLI	Mars Hand Lens Imager
MARDI	Mars Descent Imager
MER	Mars Exploration Rovers
MFEX	Microrover Flight Experiment
MSL	Mars Science Laboratory
NASA	National Aeronautics and Space Administration
NED	North East Down
PCL	Point Cloud Library
PST	Pacific Standard Time
RAT	Rock Abrasion Tool
RGB	Red Green Blue
RGB-D	Red Green Blue Depth
ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping
SUV	Sport utility vehicle
UAV	Unmanned Areal Vehicle
UGV	Unmanned Ground Vehicle
URDF	Unified Robot Description Format
USA	United States of America
USSR	Union of Soviet Socialist Republics
UTC	Coordinated Universal Time
V-SLAM	Visual Simultaneous Localization and Mapping
VCS	Version Control System

Symbols

α	Steering angle
β	Angle between local x-axis and a vector towards the center of an arc
Δ	Difference
ω	Angular velocity
ψ	Yaw angle
σ	Standard deviation
$\overline{x}$	Arithmetic mean
C	Center point of an arc
d	Distance
l_1	Distance between front and middle axle
l_2	Distance between middle and rear axle
l_3	Distance between right and left wheel, measured from the center of wheel
M	Rotational matrix
P	Position
$\vec{p}$	Vector from previous to current position
P_{err}	Position error
r	Radius
t	Time
v	Velocity
$\hat{}$	Estimate
$_{cmd}$	Command
$_{i}$	Current state
$_{i-1}$	Previous state
$_{left}$	Value for a left wheel
$_{rel}$	Relative value
$_{right}$	Value for a right wheel
$_{wheel}$	Value for a wheel
$_{x}$	Value in the x-axis
$_{y}$	Value in the y-axis

Chapter 1

Introduction

Curiosity of mankind to search for resources and potential life on planets of our solar system lead to missions beyond reach of Earth's orbit. As early as 1960s, the first interplanetary mission toward Venus was sent. This was a starting point of exploration missions going thought the last century well into present time. The challenging environment of extraterrestrial planets makes a manned exploration hard to undergo. Unmanned exploration provides a compromise in potential gains and risks.

The goal of any exploration mission is to make as many scientific advancements as possible. One way of planetary exploration is through use of landers. A lander provides stationary base which can be insufficient for instruments on-board. A platform which provides flexibility is an unmanned rover. Agencies such as National Aeronautics and Space Administration (NASA) have been flying rover missions towards other planets since 1990s. The 2020 Rover Perseverance being latest addition to the NASA rover family [31] and an earlier launched Chinese Mars rover Tianwen-1 [1].

Unmanned rover exploration brings challenges of it's own. One of the biggest is the knowledge of location and avoidance of potential obstacles. Earth has multiple infrastructures which are designed for navigation. Services such as GPS provide coverage on nearly all places on the Earth surface. This makes knowledge of exact location in real-time possible. Accurate maps provide a detailed description of terrain however exploration rover mission can be the first to ever land on a planetary surface distanced millions of kilometers away form the Earth. The simple question is, how can the navigation information become available?

The importance of this problem is most notable if a rover mission explores many places in as a narrow time frame. The rover depends on its own since no navigation infrastructure is available. To make this task even more challenging, an operator who may have access to available rover data is not able to find obstacles or create maps in time for the rover to avoid them or navigate around them. Thus, and autonomous

Recent years have seen a great amount of development towards different solution for navigation. These solution are designed so no additional navigation data is needed. These solutions take advantage of commercially available sensors and this makes this form of navigation accessible. Exploration rovers can take the advantage of using these algorithms. Existing missions used solutions of their own. The Chapter 2 takes a deeper look into the history of exploration rovers. Navigation principles of unmanned rovers are explored in the Chapter 3. Our approach, presented in the Chapter 4, will combine data provided by multiple commercially available sensors such as stereoscopic camera, depth camera and LiDAR and use them for navigation. The solution will provide both a map and position

without the use of additional navigation infrastructure. Inner working of implementation is described in the Chapter 5. The designed algorithm is to made be used with a exploration rover equipped these sensors. A simulation model of rover platform and its environment is described in the Chapter 6. The full implementation of the navigation algorithm and its performance is evaluated in the Chapter 7.

Chapter 2

Historical evolution of planetary exploration rovers

Mankind has been looking towards space exploration form early ages. This can be seen in early civilizations drawing star maps and maps of the Moon. The exploration of space has become reality in the last century . It is ranging from manned mission to the Moon, probes beyond edges of our solar system to more thorough exploration of planets, comets asteroids and moons. Unmanned rovers being one way of planetary surface exploration.

First unmanned space exploration rovers date back to the second half of 20^{th} century to back then Union of Soviet Socialist Republics (USSR). Lunokhod 1, model of which is shown in the Figure 2.1, has been a part of Luna 17 mission. Lunokhod 1 is first successful unmanned rover mission followed by the second Lunokhod 2 mission. The purpose of this mission was to explore Lunar surface, making this important step towards future planetary exploration missions. Lunokhod rovers were at first meant to pave a way for manned missions to the Moon. It would provide guidance beacon for manned lander and afterwards cosmonauts would use rover to drive across Lunar surface. This path of development was not pursued. The Lunokhod is eight wheeled exploration vehicle containing scientific instruments such as X-Ray detector, photodetector, magnetometer, soil mechanics tester and laser reflector. In addition to these instruments Lunokhod 1 and 2 contained a set of TV cameras used for surface observation and navigation [10]. The navigation of these rovers did not include any autonomy. The rover was controlled by a crew of 5 people including commander, navigator, flight engineer, communications operator and driver. The crew was controlling rover with Lunokhod Control Center which allowed each of the crew members to control their part of rovers instrumentation. Images form Lunokhod navigation cameras were received every twenty seconds. Single autonomous principle was fail-safe in form of tilt detection where if over tilt was detected, rover would stop [10, 43].

Rovers missions continued through the rest of the century and are still pursued to this day. The rover exploration effort was pursued mainly by three countries United States of America (USA), USSR/Russia and China. Each of them having their own missions, Lunokhod 1 and 2 from USSR/Russia, Yutu [40] and Yutu 2 [3] from China and Sojourner, Opportunity, Spirit and Curiosity from the USA (NASA). The USA being the only country with rovers on Mars surface, thus missions performed by NASA being most interesting for this thesis.

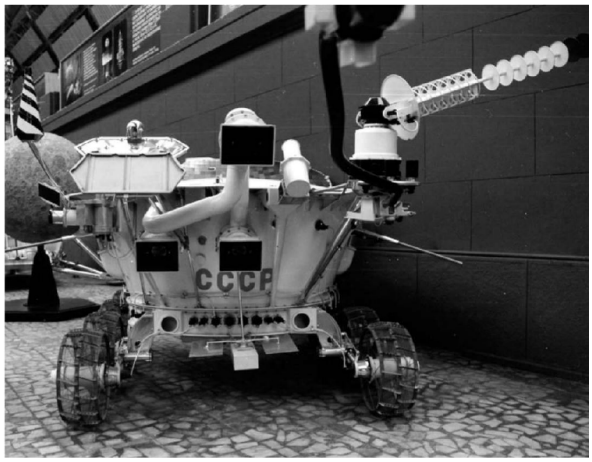


Figure 2.1: Lunokhod rover in museum.
Source: [10].

2.1 Sojourner rover

Sojourner rover, the first rover to drive on a surface of another planet, was launched as part of NASA's Pathfinder mission. The rover experiment with code-name Microrover Flight Experiment (MFEX), was small, low cost exploration platform with weight of approximately 10 kilograms [28]. It was launched in year 1996 and landed in year 1997 [42]. The rover platform was limited mainly by computational power of radiation hardened processors of that time. This ended up being one of the key limitations of design of appropriate navigation and collision avoidance systems.

Sensors and instrumentation

The rover is equipped with a selection of sensors. Some of these sensors have multiple purposes such as cameras [44]. Cameras are being used for both imaging and navigation. List of sensors and their purposes can be seen in the Table 2.1. Some of these sensors such as accelerometers are used for excessive tilt detection, the gyros are used to compute current heading of the rover. The camera system present on this rover uses a pair of stereo cameras. Stereo cameras produce image with a three dimensional perspective.

Stereoscopic cameras are extended with a set of five stripe laser projectors as shown in the Figure 2.2. Laser stripe projectors project vertical stripes on the surface of Mars. The stripe projectors and cameras are mounted to the front of the vehicle just below the solar panel. Stripe projectors generate vertical beams of light which create stripes on the surface of obstacles and the terrain in front of the rover.

Odometry was preformed using dead reckoning with data acquired by rotary encoders on wheels. The direction tracking was done using gyro and a potentiometer for the steering angle measurement. This approach does not result in high precision measurement when used in environment such as Mars, since Mars has mainly dusty surface with slopes and stones resulting in some amount of slip.

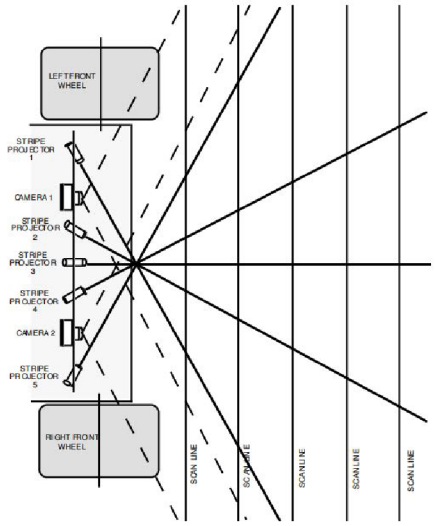


Figure 2.2: MFEX Stripe projector.
Source: [44].

Table 2.1: MFEX on-board sensors.
Source: [44].

Qty	Sensor	Primary Function(s)
3	Accelerometers	Hazard Detection
1	Rate Gryro	Dead Reckoning
2	Bumper (chassis)	Collision Detection
4	Bumper (solar)	Collision Detection
2	CCDs (front)	Imaging; Proximity
1	CCDs (rear)	Imaging; Target Validation
11	Current (Motor)	Torque Monitoring; Fault Protection
2	Bogie Position	Hazard Detection; Mobility
1	Differential Pos.	Hazard Detection; Mobility
6	Wheel Position	Dead Reckoning
4	Steering Angle	Direction Control

Obstacle detection and avoidance

The obstacle detection system makes use of a pair stereo cameras rigidly mounted to the rover front panel. The rover navigation system uses images from both right and left cameras. The images taken do not use the whole sensor area but a limited set of five horizontal scan lines. The process is repeated approximately every wheel radius worth of distance [44].

Image capture consists of five different images. One of the five laser stripe projectors is turned on per image. The image will capture stripe projected by selected projector. The lines captured by camera are then fed into a peak detector. The location of peaks in captured line signify laser stripe is projected on the surface. Since rover has available pair of stereo camera and the process of image capture is done for each of them, it is possible to calculate depth for each captured point. The result for images combined will give five

by five range measurements array. The values of positions are then evaluated to determine presence of obstacles [44].

The process of detection is based on comparison of measured values to values which would be given by a flat surface. The values of range high above surface are considered an obstacle. Ranges below flat surface level signify presence of trench or a cliff. The thresholds which determine safety of traversed terrain are determined by ground-based testing. Combination of this information and odometry provides a simple map of obstacles [44].

2.2 Spirit and Opportunity rovers

Year 2004 brought two new rover exploration missions in the form of twin rovers of Mars Exploration Rovers (MER) program.

> *„Spirit and Opportunity landed on Mars January 3 and January 24, 2004 PST (Jan. 4 and Jan. 25 UTC). Both rovers lived well beyond their planned 90-day missions. Opportunity worked nearly 15 years on Mars and broke the driving record for putting the most miles on the odometer [38].“*

Both rovers being much larger platforms than their predecessor Sojourner. Rover weigh around 180 kilogram and are the size of golf cart [38]. Rovers included more instrumentation, ranging from different scientific experiments such as Rock Abrasion Tool (RAT), spectrometers and a microscopic imager. The camera system has been improved by use of Hazcams (similar to stereo cameras used by Sojourner) rigidly mounted to the front and back of the rover [48], a pair of Pancams (panoramic cameras) mounted on a „head“ (rotating mast at the height of human eyes) [47] and a pair of Navcams (engineering cameras used for three-dimensional imagery of surroundings) located on the same mast as Pancams [48]. Better camera equipment along with performance improvements of processor power resulted in different navigation method.

2.3 Curiosity rover

Curiosity rover was launched in year 2011 and landed on the surface of Mars on 6th August 2012. Mission known by its name Mars Science Laboratory (MSL) was determined to search for mainly biological compounds. The rover platform is biggest launched so far with weight of approximately 900 kilograms and size of a small SUV [46]. This rover is a newer generation of Mars Exploration Rovers with improved on-board computational power. The MSL has radiation-hardened 200 MHz processor based on PowerPC 750 architecture which is ten times faster than previous to twin rovers of MER [5]. This mission has seen large reuse of navigation software already developed for MER mission including local and global navigation algorithms. The notable changes for this mission was the usage of new sensors some of which are described in this section. These sensors include cameras and better science instruments. All sensors and instruments on-board can be seen in the Figure 2.3.

Camera systems

Curiosity camera system consists more or less of same types of cameras as previous MER missions. The camera systems are hazard avoidance cameras „HazCams“, navigation cameras „NavCams“ [6] and cameras located on a rotating mast „MastCams“ [19]. HazCams

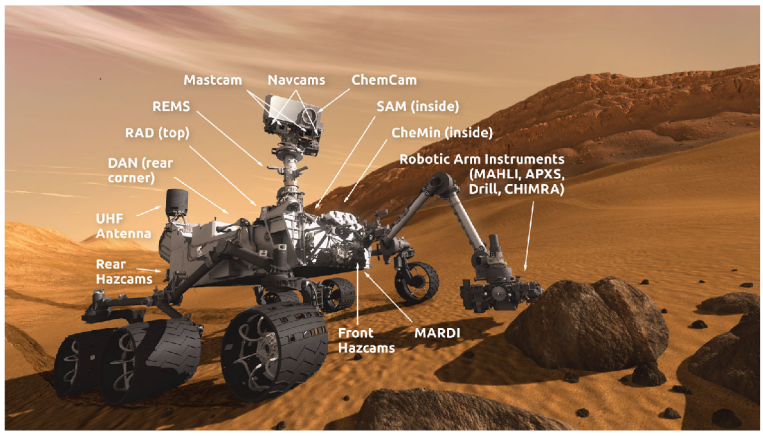


Figure 2.3: MSL instruments
Source: [45].

are rigidly mounted to the body of rover in the front and rear. Stereo imaging capability allows them to take pictures with three-dimensional perspective. HazCams take pictures with Field of View (FOV) of 124 degrees, in gray-scale format with filter for red wavelength of 650 nanometers. HazCams and NavCams are engineering stereoscopic cameras used for autonomous navigation and collision avoidance. NavCams unlike HazCams are located on the rotating mast so called „head“, pointing down towards ground providing field of view of 45 degrees. These cameras have also the capability of stereoscopic images in the same wavelength as HazCams. HazCams and NavCams both take images with resolution of 1024 by 1024 pixels [6].

Engineering cameras are accompanied by a set of science cameras. These cameras include MastCams, ChemCam and Mars Hand Lens Imager (MAHLI) [45]. Scientific cameras are not necessary for navigation but their use will be mentioned just for completion and to highlight some of the science experiments on-board. The MastCams are pair of stereoscopic color imaging cameras used for imaging of mars surface in three-dimensions. These cameras are located on the rotating mast and are equipped with a zoom lens [19]. ChemCam analyzes elemental composition of Martian rocks after a powerful laser beam has been fired at them. The last scientific camera described is MAHLI. This camera provides closeup view of minerals. Geologist back on Earth can then examine features on Mars rock which are smaller than width of human hair [17]. A less profound imaging system is Mars Descent Imager (MARDI). This system was not used after landing [18].

Chapter 3

Rover navigation principles

The MER and MSL missions use their cameras as main source for navigation information [2]. These rovers have the ability to track as much as 16,000 point per single image. The computing performance improvements being the main reason for such improvement. MER is able to travel autonomously up to ten times the speed of Sojourner, which results in larger area rover is able explore.

One of the greatest improvements allowing this level of speed was the use of visual odometry (see Section 3.2). The wheel odometry used on the previous generation of rover was suffering from inaccuracies accumulated over time. This is bound to be solved by visual odometry.

The usage of stereo cameras allows rover co create three dimensional representation of terrain. The rover is then able to create a map of environment in front of it's sensors and along path as it moves. This is done together with use of visual odometry. The autonomous navigation system called AutoNav [2] uses these points to determine the height of the terrain, density of rocks, excessive tilts and roughness of surface as can be seen in the Figure 3.1. The surrounding environment is mapped on a grid map with grid size of diameter of the wheel. Different parts of terrain are then marked with a goodness value.

Goodness values indicate traversability of terrain. High value is traverseble and low values indicate hazardous areas. The obstacles are inflated by the diameter of circle encapsulating rover. Map is updated every step. Single step is between 0.5 and 2 meters depending on how many obstacles are nearby.

A more recent addition is use of global cost-map. The cost map is a uniform grid. Each cell has inverse value from values in goodness map. The cost map classifies cells by their occupancy. Cells with lower cost are considered to be traversable where as high cost value is given to cells containing obstacles or more hard to traverse terrain. A combination of visual odometry and feature detected in images is used to create this map. This system allow a use of two dimensional planning algorithm called Field D* [7].

This chapter will explore in more detail different approaches which where described in previous paragraphs. The Section 3.1 will describe one approach to odometer which is similar to ones used on planetary rovers. The Section 3.2 will describe a visual odometry approach. The last Section 3.3 will explore a different approach which will take advantage of map to localize a rover. This approach will then be used in later chapters.

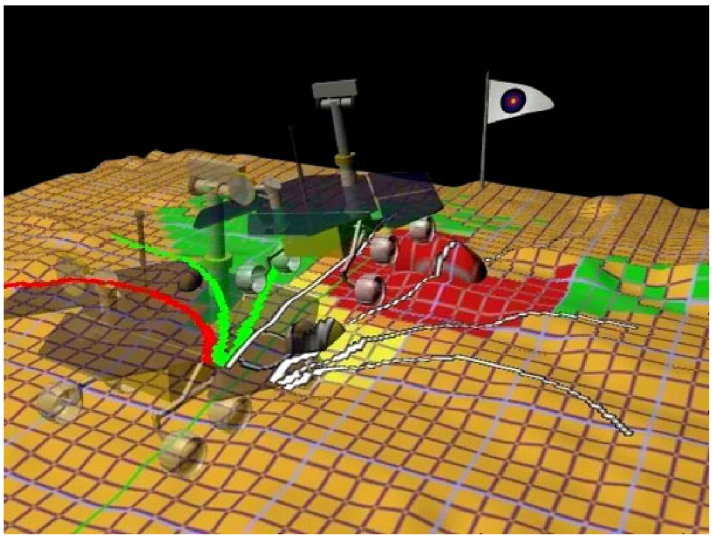


Figure 3.1: Showcase of AutoNav.
Source: [2].

3.1 Odometer

Earthbound vehicles of the past or planetary rovers such as Sojourner use localization data based on encoders. This approach takes advantage of measurement of number of wheel rotations. The accuracy of this approach is compromised once mechanical properties of the surface are unpredictable or unstable.

Slip should be considered by making this data less relied upon or by determining probabilistic nature of position information given by this source. Error created by slippage of wheel is not immediate but accumulates over time.

The calculations presented will use encoder data of two non-steerable wheels of the rover to determine a dead-reckoning odometry sample for single step. Accuracy can be improved by averaging and calculating odometry of each wheel present on the vehicle. Odometry for one set of wheels is sufficient for our use-case. The only condition for is that wheels are in contact with ground at all time.

The dead-reckoning approach uses model of robot equivalent to differential drive robot and is based on code provided in an open-source differential drive solution [4]. This approach may be used for rover types such as differential drive or for rear wheels of Ackermann steering geometry [8]. The later described rover model will be using six wheels with fixed middle axle.

The assumption is made that a path raveled by the rover can be approximated using multiple arc segments. Encoders present on motors or software controlling position of wheels can present data about position of wheels. Data about position of the wheel is expected to reflect real world orientation of the wheel. A distance traveled by the wheel can be calculated if previous and current position of the wheel are know. The difference in distances traveled by right and left wheels allows a calculation of relative yaw angle. Distance traveled by the rover is determined as average of distances traveled by left and right wheels. A relative position is estimated as a position on arc segment with arc length of measured distance and center angle of relative yaw. The new position and new yaw angle are then calculated form the previous position and the previous yaw angle.

The overall position will be in a coordinate frame of odometry. This frame can be local or represent other coordinate system such as North East Down (NED) or East North Up (ENU). The Figure 3.2 shows estimation using arc segments in ENU with x-axis in the direction of East and y-axis in the direction of North. Value of $\Delta\hat{\psi}$ is estimated relative yaw angle measured in radians, $\hat{P}_i$ is current position estimate, $\hat{P}_{i-1}$ is previous position estimate, $\hat{C}$ is estimated center point of the turn and $\hat{r}$ is estimated radius of the turn. The estimate of distance traveled by vehicle $\Delta\hat{d}$ in $\Delta t = t_i - t_{i-1}$ will equal to average of both distances traveled by left Δd_{left} and right wheel Δd_{right}. Estimated paths traveled by right and left wheels can be seen in this figure. All position are expected to be in two-dimensions and measured in meters, distances are expected to be measured in meters, time is measured in seconds.

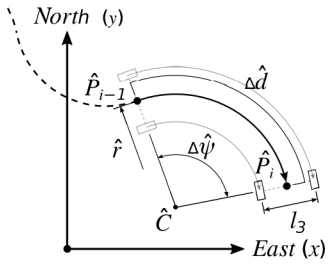


Figure 3.2: Previous and current pose estimates

$$\Delta\hat{d} = \frac{\Delta d_{left} + \Delta d_{right}}{2} \tag{3.1}$$

The arcs traveled by both wheels and rover will have the same center point. If a distance of both wheels, labeled l_3 (see section 6.2), is known the $\Delta\hat{\psi}$ can be derived from equations of arc length. A positive value of $\hat{r}$ is considered to be a left turn.

$$\Delta d_{left} = \left(\hat{r} - \frac{l_3}{2}\right)\Delta\hat{\psi} \tag{3.2}$$

$$\hat{r} = \frac{\Delta d_{left}}{\Delta\hat{\psi}} + \frac{l_3}{2} \tag{3.3}$$

$$\Delta d_{right} = \left(\hat{r} + \frac{l_3}{2}\right)\Delta\hat{\psi} \tag{3.4}$$

$$\hat{r} = \frac{\Delta d_{right}}{\Delta\hat{\psi}} - \frac{l_3}{2} \tag{3.5}$$

$$\frac{\Delta d_{left}}{\Delta\hat{\psi}} + \frac{l_3}{2} = \frac{\Delta d_{right}}{\Delta\hat{\psi}} - \frac{l_3}{2} \tag{3.6}$$

$$\Delta\hat{\psi} = \frac{\Delta d_{right} - \Delta d_{left}}{l_3} \tag{3.7}$$

The value of $\hat{r}$ can be now derived from equation for angular distance of an arc or from other previously defined equations.

$$\Delta\hat{d} = \Delta\hat{\psi}\hat{r} \tag{3.8}$$

$$\hat{r} = \frac{\Delta\hat{d}}{\Delta\hat{\psi}} \tag{3.9}$$

$$\hat{r} = \frac{l_3(\Delta d_{left} + \Delta d_{right})}{2(\Delta d_{right} - \Delta d_{left})} \tag{3.10}$$

If the measurement of d_l and Δd_{right} was made in time Δt the estimated velocity $\hat{v}$ and estimated angular velocity $\hat{\omega}$ will equal to

$$\hat{v} = \frac{\Delta\hat{d}}{\Delta t} \tag{3.11}$$

$$\hat{\omega} = \frac{\Delta\hat{\psi}}{\Delta t} \tag{3.12}$$

The new position is then found by first searching for a point on a arc defined by center point $\hat{C}$ and radius $\hat{r}$ at which rover would have same yaw angle as previous yaw estimate $\hat{\psi}_{i-1}$. This position is equivalent to last position estimate $\hat{P}_{i-1}$. A position on a arc where rover has current yaw angle $\hat{\psi}_i$ is found next. This position is equivalent to current position estimate $\hat{P}_i$.

3.2 Visual odometry

Visual odometry unlike wheel odometry does not measure distance traveled by wheel rotation but rather by detecting specific features between two or more consecutive images. The simple explanation of this approach is to take multiple images in different time and compare them to each other. The movement of camera between these images will result in movement of features present in images. Resulting features can be later used for purposes other than odometry. A three-dimensional map can be created since a known three-dimensional positions of features is available. This form of map creation principle is being used by planetary rovers such as MER or MSL. Rovers use this approach to detect and later evaluate obstacles near the vehicle.

Visual odometry approaches may use different kinds of image sensors. This includes single camera, stereo camera pair or even a depth camera. Images used as source for visual odometry needed to be rectified. The approach used for calibration will be introduced in the Section 4.2.1. Available algorithms provide motion and position error estimate. Depth cameras and stereo cameras provide higher accuracy of tracked features. Packages such as viso2, ORB-SLAM2 [29] and others provide feature complete visual odometry solutions. The package RTAB-Map [15] provides multiple visual odometry approaches. It implements two before mentioned approaches and allows the use of stereo cameras. The common principle used by RTAB-Map splits process of visual odometry into five different parts listed below.

- Feature detection.
- Feature matching or optical flow.
- Motion estimation.

- Pose update.
- Feature map or key frame update.

A different kind of feature detection algorithm or approach can be configured in a initialization phase. This is used to fine-tune performance of this algorithm which may be needed depending on an exact environment configuration. The specific implementation considered for purposes of this work is previously mentioned RTAB-Map. A stereo camera configuration is used.

3.3 Simultaneous Localization and Mapping

The main goal of the navigation algorithm presented in this thesis is to create a map and to know a position on the map. This approach should take a limited amount of available data and process it in a way which is suitable for navigation. There exist multiple approaches to do so. Approaches used by planetary rovers which have been explained at at the start of this Chapter do not take the advantage of previously created maps. Knowledge of previously visited areas can be an advantage if a location on the map is re-visited multiple times. This approach can than re-evaluate its position relative to map, which can improve overall accuracy of produced map and tracked location.

Problem of Simultaneous Localization and Mapping (SLAM) also known as Concurrent Mapping and Localization (CML), as defined by book of *Probabilistic Robotics* [49], arises when robot does not have access to a map of the environment nor does it have access to its own poses. The only information know are measurements and controls. Robot needs to acquire map of the environment whilst also localizing it on it. Online SLAM problem involves estimating these parameters for a current moment in time. Many algorithm for online SLAM problem discard past measurements and controls once they are processed. A full SLAM aims to estimate these parameters over full time.

Existing algorithms provide more optimal solutions to the problem of SLAM. Choice for different approaches was settled to be on solution provided by *MATLAB Navigation Toolbox* with name of publication *Real-time loop closure in 2D LIDAR SLAM* [11]. This approach uses data from multiple sources including laser scans and initial pose estimate to create accurate map of the environment. Laser scans are in two-dimensional form. The algorithm uses loop closure detection and pose graph optimization to make more accurate estimations. The full implementation exceed reach of this work but simpler concepts and insight into working of this approach need to be discussed. Knowledge of these principles is necessary for proper integration of algorithm into designed navigation solution.

3.3.1 Occupancy grid-map

Map provided by the SLAM is in a grid-map form. This representation is more equivalent to a raster image where each pixel represents single cell. Information stored in the cell is in a form of probability of cell being occupied. Grid-map processing then takes all cells with their corresponding probabilities and creates map containing only binary values. Information about cell being occupied or free is saved. The occupancy threshold is a value at which cell is considered to be occupied. Other threshold can determine probability at which cell is considered to be free. Single or both of these thresholds can be used to create binary map form of the occupancy map [21].

3.3.2 Scan matching

Pose changes between two different scans can be computed from scan-to-scan matching algorithm. This can adjust for an error generated by initial estimate using odometry source. The most efficient approach to scan matching uses only current and previous scan match, which is called scan-to-scan matching. On its own scan-to-scan matching accumulates error over time. Scan-to-map matching matches new scans against global map and tries to find optimal match, this improves position especially for use with pixel-accurate methods. The scan-to-map matching is however more resource expensive, which manifests itself especially on larger maps. Approach used in this case splits map into sub-map containing most recent scans to make it more efficient. Scan matching is done against recent sub-map. After amount of new scans is inserted into sub-map, the sub-map is finished and no new scans are added to it. Poses of matched scans are saved to pose graph [11].

3.3.3 Pose graph optimization

Further improvements to reduction accumulated error is through optimization of pose graph. Pose graph is structure witch attaches each scan to its pose in map. Further reduction of pose error is possible, if graph is periodically optimized as new sub maps are added. The start of this optimization can be triggered when loop closure is detected [11]. The result of this optimization should improve overall pose in a global map.

3.3.4 Loop closure detection

Finished sub-maps are added as candidates for loop closure detection. The process of loop closure detection tries to scan-match newly scanned areas to already explored parts of the map. Match will occur if the scan with greater correlation than set threshold is found. The search is preformed in a selected radius. Adjustment to pose and optimization to pose-graph is made if a match happens [26].

Chapter 4

Research of used navigation algorithm

Rover navigation on a planetary surface is a challenging task when a limited set of sensors and available navigation data is considered. Earthbound vehicles are equipped with sensors capable of accurate pose estimate in real-time. Systems such as GPS or European Galileo provide exact navigation information in virtually all places on the Earth's surface. These systems work on the basis of multiple satellites providing constant signal to the vehicle. The navigation on the surface of planet other than Earth's cannot be based on expectation of constant position updates from such systems. Planets such as Mars do not have navigation infrastructure capable of providing this kind of service. The accurate estimate of pose may not be possible or available for long duration of time. Rover is in this case depending on on-board sensors. Overall overview of sensors data used and outputs of the navigation algorithm can be seen in the Figure 4.1.

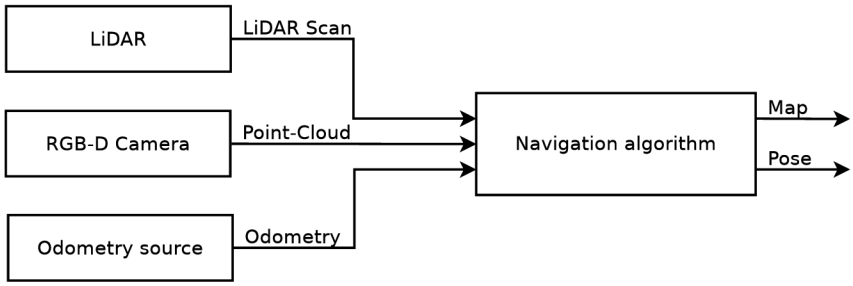


Figure 4.1: Navigation algorithm architecture

Navigation algorithm used for problem of localization must be able to identify motion and pose relative to the surrounding environment of the rover. A map of surrounding environment is requirement for autonomous and safe navigation and path planning. The map may already be available or unknown. The navigation in already known environment is commonly used for in non-changing environment such as household. The localization in this environment may be provided using Monte Carlo localization [49] based on reference grid map and sensor data.

The case for planetary exploration has an unknown map. This requires the use of previously mentioned SLAM. SLAM algorithm may be used to map two-dimensional or three-dimensional environment depending on data provided. This form of mapping gen-

erates maps in form of two-dimensional grid or three-dimensional octree structures [12]. Sensors providing information about depth may use two-dimensional SLAM. Image-only based approach, Visual Simultaneous Localization and Mapping (V-SLAM) [15], uses data provided by single or stereo pair of cameras to create three-dimensional map of the environment. Both SLAM approaches may take advantage of initial pose estimate.

The navigation algorithm will take advantage of selection sensors based of sensors used on existing Mars rovers. It will process available data into form of two dimensional scan. Existing two-dimensional SLAM algorithm will be used to create a map of the environment and track rover's location. More detailed flowchart presented in the Figure 4.2 provides a overview of different parts of navigation algorithm.

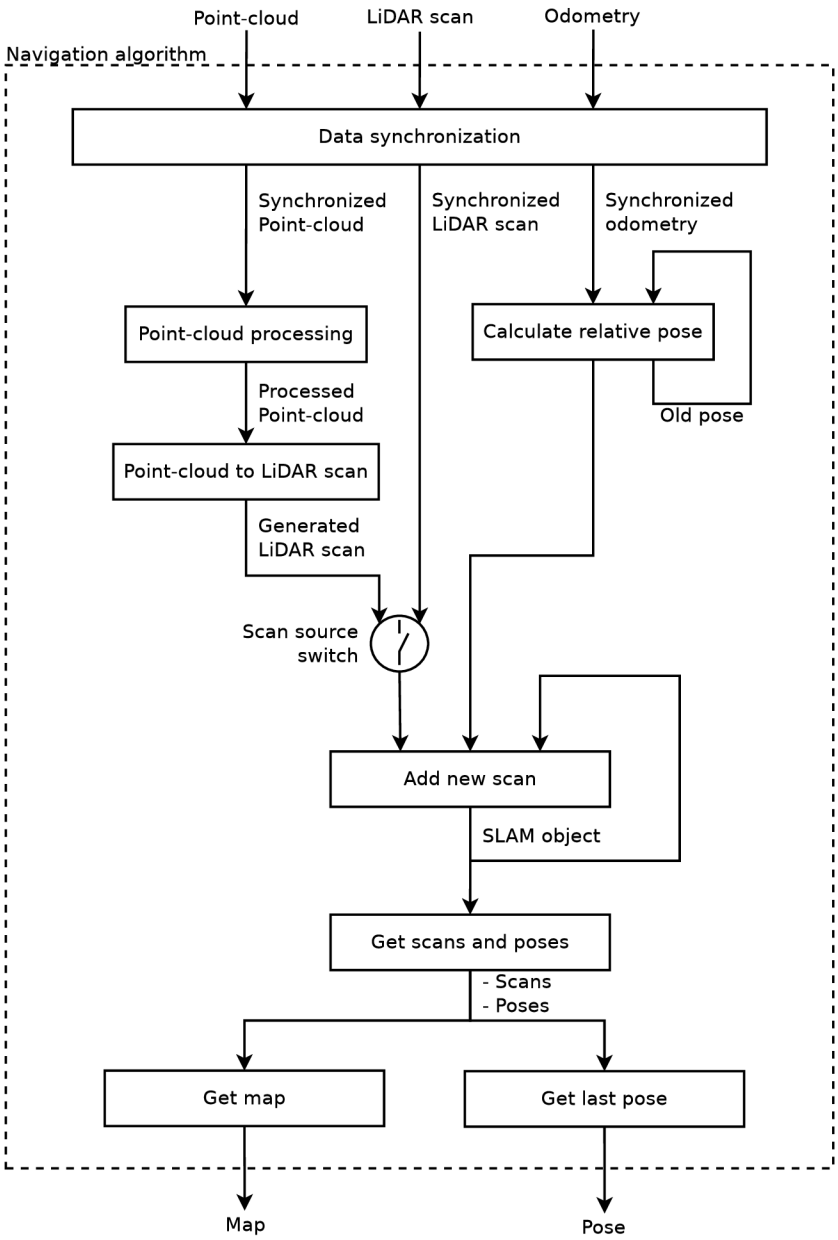


Figure 4.2: Overview of the navigation algorithm

4.1 Sensors

Planetary roves sent throughout the years have been equipped with different variety of sensors. Modern and commercially available counterparts to these sensors are considered as it is more desirable for future work. Today's market provides selection of sensors which can be used for navigation. A number of different cameras and depth sensors which can be considered today's version of those used on Mars missions will be described in this section.

4.1.1 Stereoscopic camera

One kind of imaging sensor commonly used for planetary rover navigation is the stereoscopic camera. Camera consists of two separate sensors distanced from each other by a known length. Two sensors are imaging same space from different angles or positions. If a feature is detected by both cameras at the same time (considering cameras are calibrated) the triangulation can be used to determine distance of it from these sensors [36]. Stereoscopic camera provides a more accurate pose estimate when used for visual odometry described in the Section 3.2.

4.1.2 Depth camera

One approach used for range imaging is a depth camera. Depth cameras, considered for this algorithm, provide image in the form of RGB-D [16]. As a name may suggest the output image consists of four different channels, Red (R), Green (G), Blue (B) and Depth (D). Depth perception is achieved in similar way to stereoscopic imaging. Most depth cameras use 3 different imaging sensors. Infrared projector is used in addition to project points on the surface in front of sensors. Two infrared sensors are detecting projected points. A method is used to determine three-dimensional position of projected points. This directly results in depth perception without need for additional processing on the receiving end. Third camera is used for RGB observation of environment and it can be used for later processing.

4.1.3 Light Detection and Ranging

A different approach to depth perception is through Light Detection and Ranging (LiDAR). The LiDAR uses laser beam range finder to determine distance information [32]. The distance is determined by measuring time it takes for the sent signal to return. This approach is similar to RADAR systems, hence the name. Commercial LiDAR sensors provide measurement of 360 degrees around one point. A form of scan can be a one-dimensional line or two-dimensional image. The lidar will sent beams of light in multiple direction around the point of rotation. Sending rate of these beams can be changed on-the-go which allows adjustable resolution of measurements.

4.2 Image processing

Raw data acquired by a camera is not a suitable input for processing algorithm such as visual odometry. This section will describe camera calibration method used to process raw image data in way such that they can be later used in the researched navigation algorithm.

4.2.1 Camera calibration

Most of feature detection algorithms expect input images to be in a form equivalent to a pinhole camera [36]. The image generated by real-world camera is distorted due to usage of lenses.

Image calibration is done using software capable of determining distortion parameters. Software such as MATLAB's Single Camera Calibrator App, MATLAB's Stereo Camera Calibrator or open-source library OpenCV provide this functionality. The calibration process uses a physical calibration pattern in a form of checkerboard, an asymmetrical circle pattern or a symmetrical circle pattern. The result of calibration will be a list of distortion coefficients.

OpenCV documentation [37] describes calibration process as estimation or radial and tangential distortions. Distortion coefficients are described by coefficients $(k_1, k_2, p_1, p_2, k_3)$ where k_1, k_2, k_3 are used for radial correction and p_1, p_2 are coefficients for tangential correction. Radial correction is done to remove fish-eye lens effect. This is mainly due to geometry of selected lenses and is more profound on lenses with high field of view. Tangential correction is done to remove orientation distortion. This distortion is result of imaging plane not being parallel to lenses when taking an image.

Reverse transformation is then applied on every image taken by calibrated camera, resulting in pinhole camera-like image. This image is considered to be rectified. Unrectified images may cause problems later down the line, especially when exact real-world position of object has to be estimated.

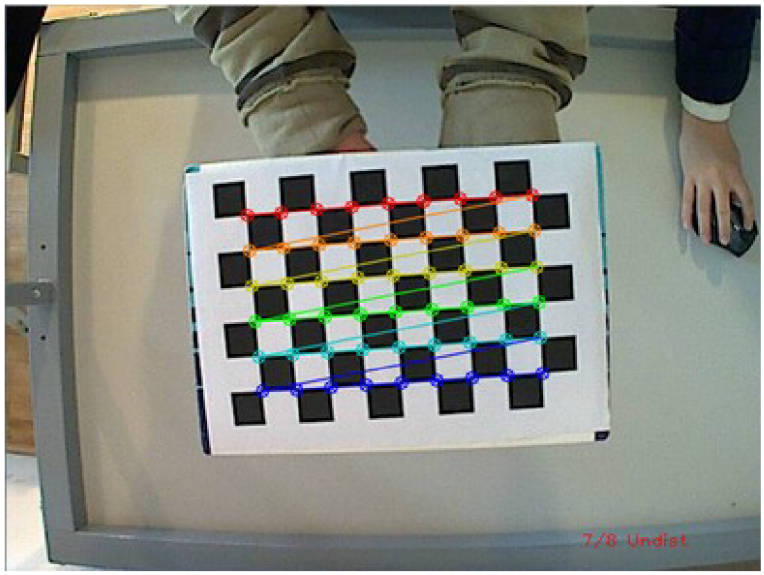


Figure 4.3: Camera calibration with OpenCV.
Source: [37].

4.3 Relative pose

A odometry estimate will give an absolute position in defined coordinate frame. The navigation algorithm requires a position relative to coordinate frame of last scan. This is calculated form previous odometry position from as described in the Section 3.1. Information needed is current position P_i and a previous position P_{i-1}, both position are represented as a point in two dimensions measured in meters. Yaw angles ψ_{i-1} and ψ_i are required as well and are expected to be in radians. The communication interface between navigation algorithm and odometry block uses quaternion for orientation. The yaw angle needs to be acquired by converting quaternion to Euler angles. A difference of positions in a form of vector $\vec{p}$ is then calculated as $\vec{p} = P_i - P_{i-1}$. This will be a two-dimensional row vector. Relative yaw angle is calculated as difference of $\Delta\psi = \psi_i - \psi_{i-1}$. The vector $\vec{p}$ does not represent position in coordinate frame of previous scan. In order to transform it a proper coordinate frame an inverse rotation around origin by angle of ψ_{i-1} is needed.

$$M^{-1}_{\psi_{i-1}} = \begin{bmatrix} \cos(-\psi_{i-1}) & \sin(-\psi_{i-1}) \\ -\sin(-\psi_{i-1}) & \cos(-\psi_{i-1}) \end{bmatrix} \tag{4.1}$$

$$\tag{4.2}$$

The relative position in coordinate frame of previous scan P_{rel} is then calculated as multiplication of a vector by a matrix.

$$P_{rel} = \vec{p} M^{-1}_{\psi_{i-1}} \tag{4.3}$$

4.4 Data synchronization

Selected SLAM algorithm expect data form multiple sensors to be available. Data acquired from sensors may not be synchronized. This causes two contradictory information to be received at the same time. In an ideal situation, the data provided by both sensors would the same be at the moment in time. This would be true for received data. Different real-world sensors and networking solutions provide different data-rates and data delays. This requires usage of time stamps. Time stamp provide a simple description about when the data was created or measured.

Synchronization process used in this algorithm expects different rates for data publishing among sensors. At first, current time stamp is established and only data with newer timestamp is received. Second expectation is about sensors publishing data in different rates. The odometry data is expected to be received more frequently then depth sensor data. If network delay between sensors and mapping software in not too long, additional synchronization between two data sources is not needed. Depth information is received and odometry is expected to arrive in short time span after.

4.5 Point-cloud processing

Images taken in RGB-D provide depth information in a form of a point-cloud. This kind of point-cloud is defined in a way similar to a raster image taken by a camera. It contains image resolution and depth information in an array representing pixels. Additional information such as projected color and FOV may be included providing more information.

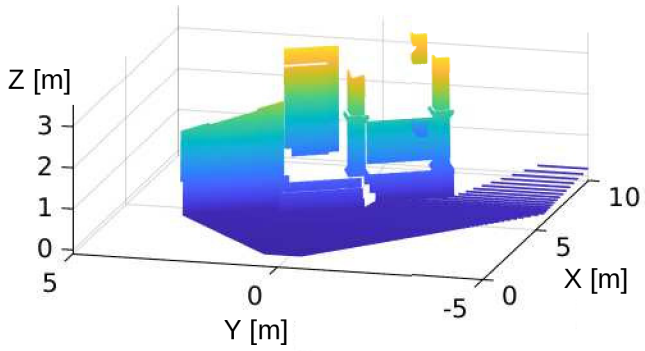


Figure 4.4: Point-cloud directly from the camera

The full height of image taken by a camera sensor will cover more FOV than is needed for obstacle detection. This can be seen in the Figure 4.4. Square sensor with FOV of 120 degrees may contain usable data only on up to middle of the sensor. Simple filtering as image cropping may be used for this approach. More sophisticated approach uses a selection of area in three dimensions after 2D image is transformed into a three-dimensional point-cloud structure. Projects such as Point Cloud Library (PCL) [41] or MATLAB point-cloud implementation in Computer Vision Toolbox [20] provide point-cloud segmentation algorithms for different three-dimensional volumes. Order of processing steps done on point-cloud is described in the Figure 4.5. Segmentation algorithm used in a point-cloud processing pipeline are introduces in following sections.

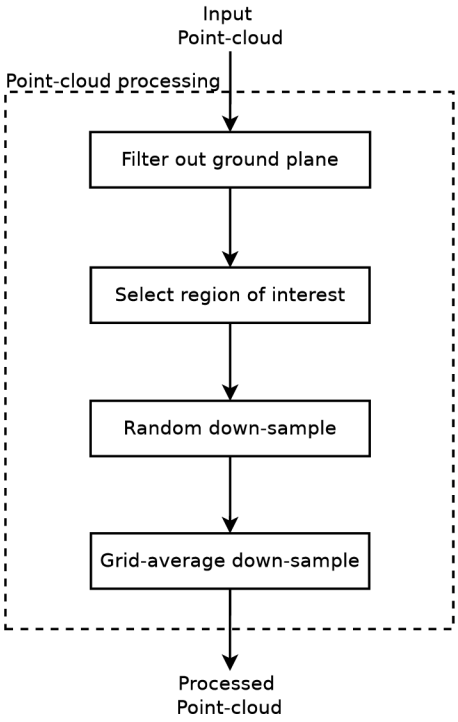


Figure 4.5: Point-cloud processing pipeline

4.5.1 Ground plane filtering

Generated point-cloud will contain points of objects of interest such as obstacles or rocks. These object are surrounded by points on the ground or obstacles, which are not of interest. The ground by itself should not be considered a obstacle which would not be the case if no further processing is used. The filtering algorithm should separate out any points which are considered to be non-threatening. The approach used for filtering of ground is described as plane fitting [24].

Simple plane fitting algorithm would detects small deviations as not being a part of ground plane. This problem is solved using maximum allowable distance from an inlier point (a point approximately fitted to the plane) and an maximal angular rate [24]. These two parameters provide option to set a distance from plane in case of first parameter and a maximum absolute angle from defined normal vector of plane in case of second parameter. This will remove point which may originate from noise. Maximal angular rate allows to filter out misalignment and relatively small terrain slopes.

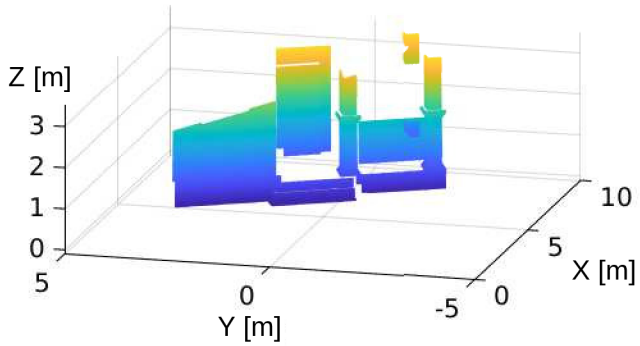


Figure 4.6: Point-cloud with ground plane filtered out

4.5.2 Region of interest

Points representing obstacles will be contained only in a small volume of overall point-cloud. Three-dimensional volume used for reduction of working area is Region Of Interest (ROI). This type of segmentation allows definition of box in three dimensions [23]. Total of six different values need to be specified for filtering. Set of two values define lower and upper boundary, three of these sets define boundaries for each axis. Segmentation result will contain segment of point-cloud in specified region. This will result in highly reduced number of points in the point-cloud. Later filtering operations will speed up the process.

The filtered area used in our application selects any point in region in front of the vehicle up to a height of 0.5 meters. The distance is limited to 20 meters. The overall boundary definition per each axis is specified in the Table 4.1.

Table 4.1: Region of interest boundaries

Axis	Lower boundary [m]	Upper boundary [m]
X	0	20
Y	−20	20
Z	0.05	0.5

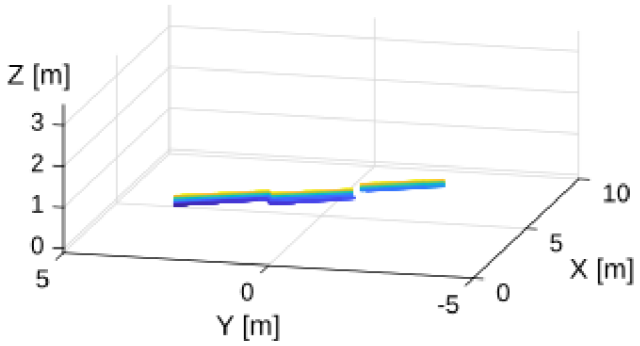


Figure 4.7: Point-cloud segment inside of ROI

4.5.3 Down-sampling

The ROI segmentation provides smaller but overall still rather large size of point-cloud. This comes as a result of resolution of image acquired from a sensor. If input image is of resolution 1000x1000 pixels and the previous processes reduce overall size of point image to $\frac{1}{10}$, the result will still contain 100-thousand points. This is just too many for SLAM algorithm anticipated to be used. The step of down-sampling is expected to reduce size of this point-cloud into more manageable size of ≈ 1000 points.

Large amount of down-sampling algorithms are available in point-cloud processing libraries [22, 39]. Some of these methods provide performance advantage or higher accuracy. The main goal behind it is to reduce the number of points in the point-cloud. In our case, two methods are considered for down-sampling. Grid averaging method returns points filtered using grid-box filter of a defined three-dimensional box size in a grid [22]. The random sampling takes random points with predefined probability, the resulting point-cloud will approximate only a fraction of points defined by the random sampling probability [22].

Combination of both algorithms is used. Both of these approaches have their advantages. The random sampling algorithm requires less processing power compared to the grid sampling, which provides evenly spaced out points. To take the most of both approaches, the sampling firstly takes random samples from point-cloud with a rate of 5%, then the grid-average sampling takes resulting samples and reduces them into one point in 5 centimeter cubes in a grid.

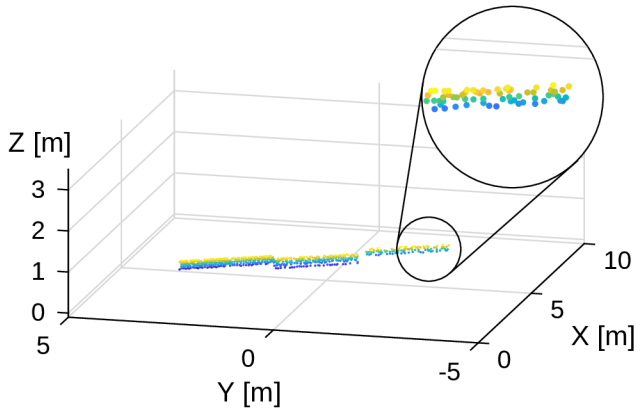


Figure 4.8: Down-sampled point-cloud

4.6 Two-dimensional scan

A two-dimensional LiDAR scan is a representation of two-dimensional point-cloud defined by set of angles and distances or a list of points in 2D space defined by a tuple (x, y). This form of data structure is used as a input to the selected SLAM algorithm. The result of processes described in previous parts of this section was three dimensional point-cloud. This step uses a projection of points onto a plane. Acquired points are expected to define most distinct edges and features in nearby environment of the rover. These points are expected to be on the surfaces of objects representing obstacles. The projection takes coordinates of all points and extracts only information about (x, y) positions of point. This effectively projects points onto x-y plane. Points are then used to create data structure equivalent to a LiDAR scan.

Chapter 5

MATLAB implementation of the navigation solution

MATLAB provides an all purpose system for different problem solving tasks. Proper toolboxes provide tools from linear algebra to advanced sensor simulation, image processing, autonomous driving, physics simulation and others. This makes it an ideal software for use in navigation. Navigation, and Image Processing Toolboxes will be used in our case. This chapter will discuss main parts of implementation handling point-cloud processing, and map creation. The integration of the algorithm into a simulation environment will be discussed in the next chapter. The base of the implementation of the navigation algorithm will be discussed in sequential order. First section will explain synchronization and transformations done on data received from sensors. Second part of this chapter will take a look into processing of three-dimensional point-cloud provided by RGB-D image. The later section will introduce configuration needed for SLAM algorithm to work properly and the generation of the output map and attitude.

5.1 Data acquisition, synchronization and pre-processing

Data provided by sensors described in the Section 4.1 will be sent over through a local network. Receiving process of all provided information is done through MATLAB's ROS Toolbox and will be discussed in the Section 6.6. This toolbox allows us to simultaneously receive and send data. Receiving is done through structures `odometrySub, lidarSub, pointCloudSub`. Use of local network expects little to none network delay. Another expectation is in data from sensors being received on a lower frequency than the data from the odometry source. If network communication is close to real-time, no additional synchronization may be needed. This expectation is reflected in implementation. Blocking calls are established to receive data point-cloud and LiDAR data first then odometry.

Asynchronous reception of messages would be needed, if data would be out-of-sync. This may be done by defining subscriber structure with a callback function. The odometry callback function would then save odometry data with a timestamp into circular buffer of pre-defined size. The timestamp of depth sensor's message would be used to find closes odometry timestamp. Closest timestamped pair would be used in later processing to avoid non-synchronized messages.

Coordinate frame transformation is done as pre-processing step on sensor data. Importance principles behind this step will be discussed in the Section 6.6.

Odometry data contains position in three dimensions, orientation defined by quaternion, linear and angular velocities and a co-variance matrix. Data used by SLAM algorithm expect odometry as relative pose to a previous scan. Relative pose should contain position in x-y plane and orientation.

A transformation of quaternion into Euler angles is done first. This step uses method `quat2eul`. Only last angle in vector of output angles is needed. A relative pose is then established by use of equations defined in the Section 4.3. The relative position in x-y plane and the relative heading are then combined into single pose `relPose`.

5.2 Input data processing

Point-cloud is a three-dimensional representation data structure. RGB-D format provides point-cloud in a form similar to a raster image. The depth data is stored in a grid representing pixels. This form of representation is converted and then processed using *MATLAB Computer Vision Toolbox* [20]. Conversion form RGB-D image is achieved using constructor of point-cloud structure. The method calls `pointCloud(readXYZ(ptCloud))` and provides conversion of RGB-D point-cloud into three-dimensional point-cloud used for processing.

Filtering the point cloud is done using different segmentation methods. Any form of segmentation is done in two steps. First a set of points to which a defined segmentation constrain applies is found. Secondly indexes of found points are used to select them from original point-cloud.

Two segmentation methods are used. First method is for segmenting of ground plane. This is achieved through function `pcfitplane`. This function returns array of indices, which are selected from original point-cloud using function `select`. Second segmentation method selects ROI with function `findPointsInROI`. Same `select` call is used to get new point-cloud.

Down-sampling is provided through function `pcdownsample`. This method combines both down-sampling approaches mentioned in the Section 4.5.3. Method used is selected with appropriate parameter of down-sampling method `'random'` or `'gridAverage'`. This step will return sampled point-cloud according to selected method and other configurable parameters.

Parameter of point-cloud `.Location` is used to get location of every point in three-dimensional space. Only x and y positions are extracted. Structure containing two-dimensional LiDAR scan is created with extracted x-y locations using method `lidarScan`. This structure is later used for SLAM.

5.3 MATLAB Navigation Toolbox

The main part of navigation in unknown terrain is creation of map and knowledge of location relative to the map's origin. As was mentioned before this, is achieved thorough SLAM. Localization and mapping is provided by *MATLAB Navigation Toolbox* [25]. This toolbox provides methods for working with SLAM algorithm described in the Section 3.3. Provided solution allows program to periodically add new scans and map updates. This section will describe configuration procedure and steps towards exporting a map. Integration of this map and pose into simulation environment will be described in the Section 6.6.

5.3.1 Integration of simultaneous localization and mapping

The structure containing parameters and overall configuration of SLAM approach used has to be created in order to use this algorithm. Function `lidarSLAM` creates this structure. A maximum scan range and resolution of underlying map are set by parameters of called function. Returned structure allows fine-tuning of other values. Some of the values include thresholds for loop closure detection and search radius for loop closure.

New scans are registered using function `addScan`, this function takes created scan, SLAM structure and relative pose estimate. It results in three return variables containing information about success of scan addition, information about detected loop closures and information about results of optimization.

5.3.2 Map and pose output

Process for generating a map starts once a new scan is successfully added and can be optionally turned off. Generated map will take a form of cost-map containing probable occupation of the cell. Poses and scans have to be acquired from SLAM object first. Method `scanAndPoses` takes SLAM object and returns list of scans and poses. The map is acquired form created scans and pose through function `buildMap`. This function takes list of scans, list of poses of scans, map resolution and maximum range of sensor as its input parameters. Generated map is suitable for visualization in MATLAB. Further processing is done in order to create suitable output. A example of produced map with traversed path can be seen in the Figure 5.1.

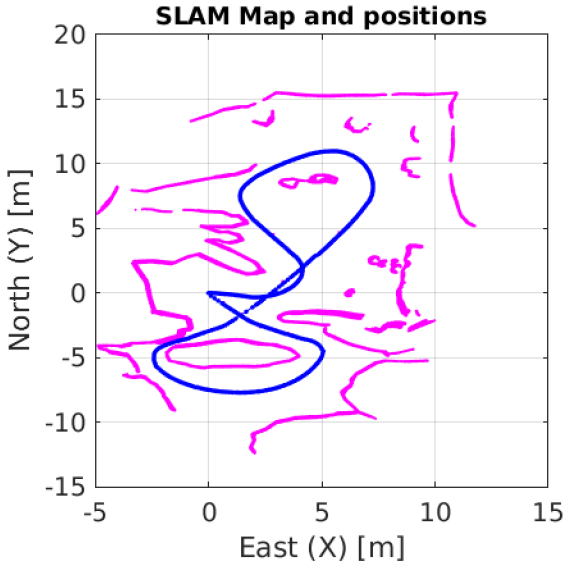


Figure 5.1: Example map and position output

Chapter 6

Integration of the navigation solution into a simulator

The algorithm by itself needs a testing environment providing input data. This can be achieved from real-world sources or a simulation. Sufficient amount of data from surface of another planet is not accessible so a simulation environment needs to be configured. A proper simulation environment should have the ability to be used for purposes of input data and rover simulation. This chapter will discuss robot model, model of Martian surface, models of sensors used as inputs, control software of the rover and integration of implemented algorithm.

6.1 Rover platform

The rover platform is inspired by the first Mars Rover sent by NASA, the rover Sojourner. This section describes designed approximation of the rover. Goal is to make Sojourner rover fully controllable. Rover model will be placed in a simulation environment Gazebo (see Section 6.3) which will provide it with other functionalities such as sensor models and integration into Robot Operating System (ROS).

The choice for rover Sojourner was made as tribute to a first successful rover mission. As one of total of three successful rovers sent so far, it established path toward unmanned rover exploration. The challenge of modelling such rover is the unavailability of existing 3D models. Unlike later missions as MSL or MER the provided information is limited to images and rough dimension measurements. Small overall size of this rover makes it a suitable platform for future work, mainly as physical model for real-world testing of the researched algorithm.

The model has been created using set of reference images of model in Smithsonian National Air and Space museum. Overall dimensions provided by NASA include height width and length of the rover. A reference volume was created for approximate dimension estimation. Free 3D modelling software Blender was used to create rough model for dimension measurements as can be seen in the Figure 6.1. These measurements were then used in parametric Computer Aided Design (CAD) software for precise 3D model creation. Up to ten different parts of the rover have been modeled. These include simplified drive-train, body and solar panel.

The overall mechanical model of rover is simplified for simulation purposes. The original vehicle has six-wheels, three on right and left side. Each wheel provides driving power in

form of electric motor. Two rear and two front wheels have ability to be steered. Steering is expected to be independent and limited in range to $\pm$ 60 degrees. Dimensions of the rover's drive-train, later referred to a simplified model for control, are $l_1 = 0.22[m]$, $l_2 = 0.2[m]$ and $l_3 = 0.36[m]$.

Suspension on original rover consist of seven different parts. Created model does contain all of those parts. Original suspension allows all six wheels to be on the ground in virtually all cases, which improves driving ability. Our simplified model has requirement of middle pair of wheels to be on the ground at all times. This is due to use of odometer. Original suspension was simplified in a way so this condition is satisfied. Front and middle wheels of the model are linked together with a independent swing arm. This allows front and middle wheel to contact the ground in most situations.

Rover is equipped with simulated form of sensors described in the Chapter 4. All sensors are located in the front of the rover. Simulated sensors include:

- LiDAR,
- stereo-graphic camera,
- depth camera.

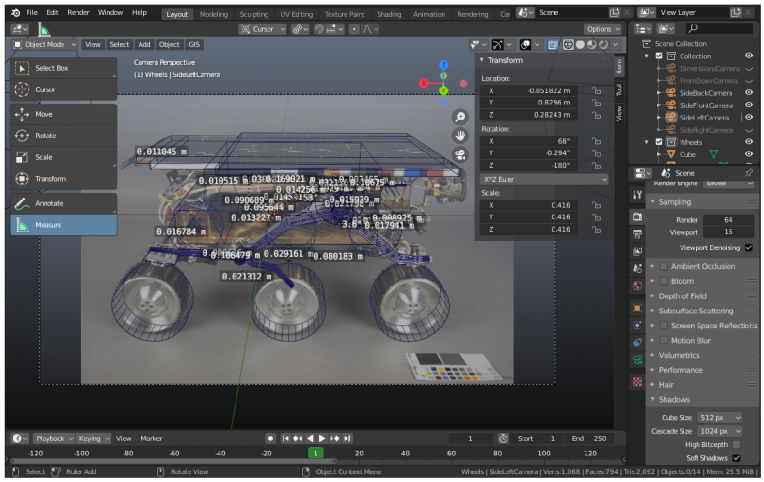


Figure 6.1: Rough measurements of reference images

6.1.1 3D model

The 3D model was created using SolveSpace modelling software. Overall drawing containing all reference dimensions was firstly created as shown in the Figure 6.2. This drawing contains positions of all necessary components. The model itself can be split into two different parts.

First part is body. Rover body models overall shape of the rover. The more car-like shape is used as a base of the model to which other parts are attached. The center of this part is considered to be geometric center of the bounding box. The second part of the body is a solar panel. This part's purpose is purely visual. Full body has two simplified collision boxes to reduce computational demand due to higher polygon count. A bounding box of body and solar panel are used as collision boxes. The simulation software allows us

to define inertial and mass properties of these two parts. Inertial and mass estimates are only approximate and are configured by a parameter. This allows us to later adjust it to reflect real world model. The mass of 4 kilograms is set as body mass, solar panel weights 1 kilogram. Inertia of both sub-parts is simulated by simplified model of bounding box and is set with macro provided in mybot Gazebo tutorial.

A drive-train may be considered as a second part of body. It can be split into right-side, left-side and single connecting part. Right and left sides are symmetrical. The model takes advantage of all six wheels. Wheel model is simplified into a cylinder. Steerable wheels are connected to the suspension using a separate steering joints with range of ± 60 degrees. Every wheel is connected to its parent part using a revolute joint. Steering joints are present on the front and rear wheels. Both steering and wheel joints have motors assigned. This allows every wheel to be controlled independently. The collision model for a wheel is equivalent to its shape, the collision model for part used in steering is simplified into 2 centimetre wide sphere.

The suspension model is consisting only of one moving part for each side. This part connects front and middle wheels and allows them both to swing independently from the rest or the rover. Real-world rover suspension contains 7 moving parts. Each part of the suspension is present in the model but model itself is simplified.

The parts are exported into `.stl` file format. This allows them to be used in robot description file. The robot description file is in `.xacro` file format. This file format is being used by multiple ROS nodes to describe configuration of the rover. It allows us to define exact position, rotation, collision boxes, inertial properties, friction properties, colour and meshes used for different parts of rover. It allows us to separately define different parts and to then connect them together using joints with different properties. Set properties are then used by simulation software to model rover. Four different `.xacro` files are used to describe rover model. They are split to overall rover description, macros, material definition and simulation specific configuration. Rendering of fully assembled rover without sensors done in Blender can be seen in the Figure 6.3 and in the Figure 6.4.

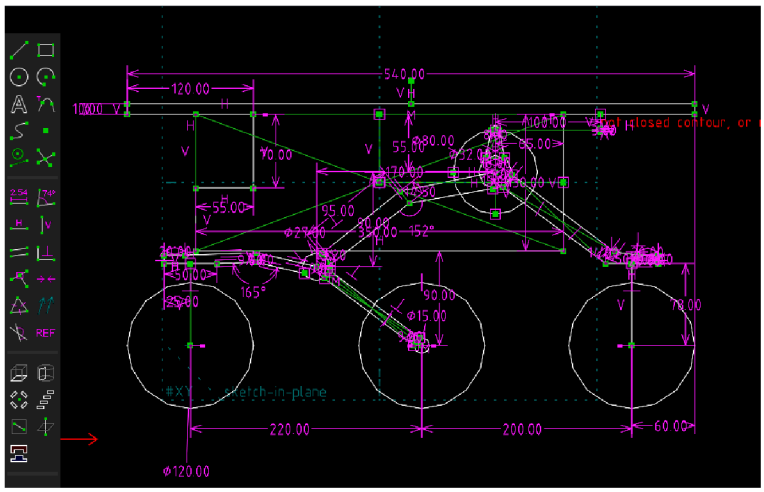


Figure 6.2: CAD model with all required dimensions

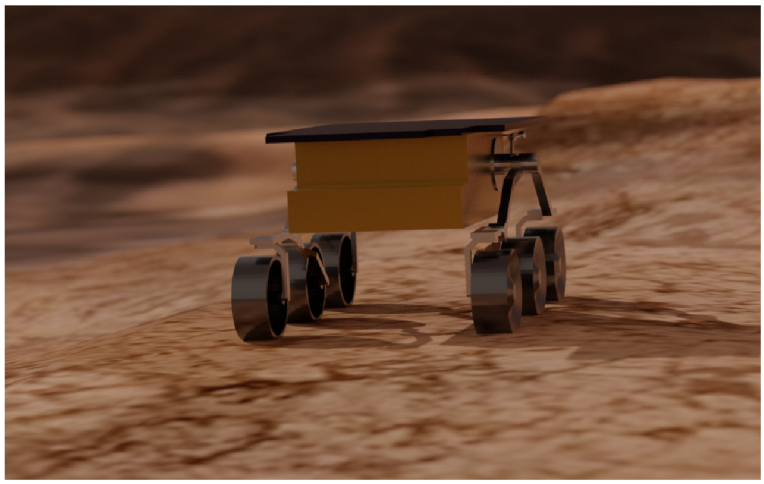


Figure 6.3: 3D model form the right side

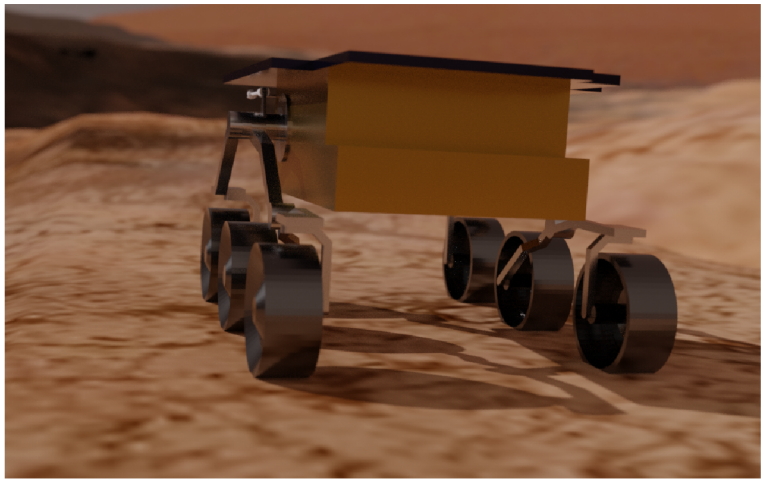


Figure 6.4: 3D model from the left side

6.2 Rover control

The rover steering is done using steering command. Command consists of two values linear velocity v_{cmd} and angular velocity ω_{cmd}. Units used in these variables are $m.s^{-1}$ for linear velocity and $rad.s^{-1}$ for angular velocity. The resulting path will be in a from of arc if the rover travels at a constant speed with a constant angular rate. The radius of arc for a given command r_{cmd} for cases where $\omega_{cmd} \neq 0$ equals to

$$r_{cmd} = \frac{|v_{cmd}|}{\omega_{cmd}} \tag{6.1}$$

The value of r_{cmd} is adjusted for reversing so steering angle will not change direction for the same angular velocity. Different behavior for reversing is not needed since the vehicle will be able to do in-place turns and sensors are expected to be on the front side. This equation by itself does not take into account physical limits of steerable wheels. The classification of different types of turn present in the Figure 6.5 is used to solve problems posed by mentioned limits.

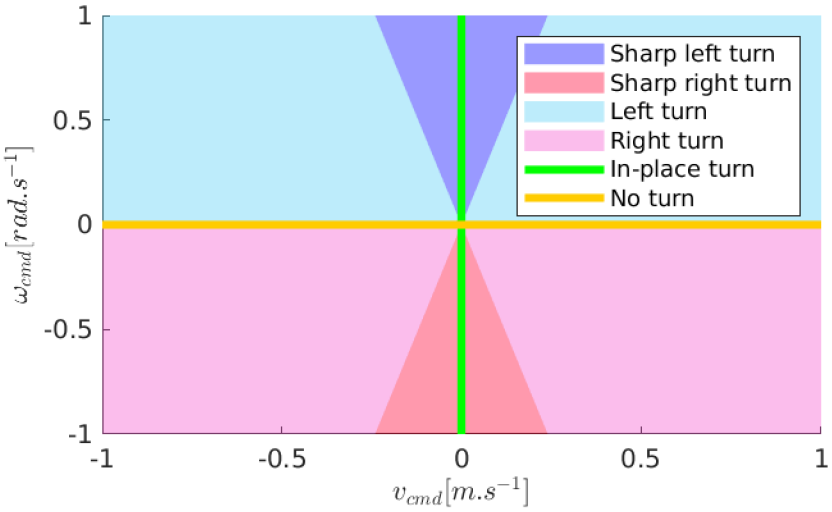


Figure 6.5: Turn classification

This classification is used to determine values of ω and v. New values of angular velocity and linear velocity are used for calculation of steering angles and angular velocities of wheels. The classification will use value of l_3 described later in the Figure 6.6. Equations for this classification are described in the Table 6.1.

Table 6.1: Turn calcification

Turn	Condition	ω	v
Left	$r_{cmd} \in \langle \frac{2l_3}{3}, \infty)$	ω_{cmd}	v_{cmd}
Sharp left	$r_{cmd} \in (0, \frac{2l_3}{3})$	$\frac{3\lvert v_{cmd}\rvert}{2l_3}$	v_{cmd}
Right	$r_{cmd} \in (-\infty, -\frac{2l_3}{3}\rangle$	ω_{cmd}	v_{cmd}
Sharp left	$r_{cmd} \in (-\frac{2l_3}{3}, 0)$	$-\frac{3\lvert v_{cmd}\rvert}{2l_3}$	v_{cmd}
In-place	$r_{cmd} = 0$	ω_{cmd}	v_{cmd}
None	$\omega_{cmd} = 0 \wedge v_{cmd} \in \mathbb{R}$	0	v_{cmd}

The center point of the turn circle C is in local coordinate frame of the rover where x-axis is in forward direction, y-axis is to the left and origin is in the center of the middle axle. The local coordinate frame of the rover can be seen in the Figure 6.6. Position is determined from radius r calculated using new values of angular velocity ω and velocity v. Once again behavior for reversing is not considered. These equation are defined only for turn types where $\omega \neq 0$.

$$r = \frac{|v|}{\omega} \tag{6.2}$$

$$C = \begin{bmatrix} 0, & r \end{bmatrix} \tag{6.3}$$

Position of turn center is then transformed into coordinate frame of each wheel. A simplified rover model is used to describe placements of different part of the rover. Parameters describing simplified model of rover shown in the Figure 6.6 are l_1 (distance from axis of rotation of front wheels to axis of rotation of middle wheels), l_2 (distance from axis of

rotation of middle wheels to axis of rotation of rear wheels), l_3 (distance of right and left wheel pairs, measured from center of wheels). All distances are expected to be measured in meters. Positions of wheels in the coordinate frame of the rover are presented in the Table 6.2.

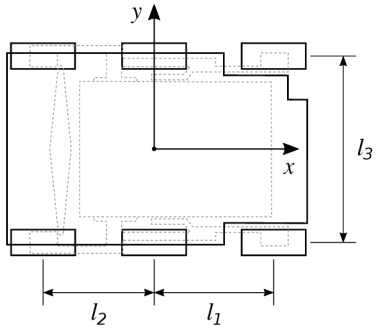


Figure 6.6: Simplified model of the rover

Table 6.2: Positions of wheels

Wheel	P_{wheel}
Right front	$\left[l_1, \quad -\frac{l_3}{2}\right]$
Right middle	$\left[0, \quad -\frac{l_3}{2}\right]$
Right rear	$\left[-l_2, \quad -\frac{l_3}{2}\right]$
Left front	$\left[l_1, \quad \frac{l_3}{2}\right]$
Left middle	$\left[0, \quad \frac{l_3}{2}\right]$
Left rear	$\left[-l_2, \quad \frac{l_3}{2}\right]$

The transformation of center point of the arc C into a coordinate frame of each wheel labeled as C_{wheel} is then done for all cases other than no turn.

$$C_{wheel} = C - P_{wheel} \tag{6.4}$$

The angular velocity on newly created arc will be the same for wheels and rover. The radius of arc is defined as distance of the transformed center point $||C_{wheel}||$ from origin of the coordinate frame. The absolute value of tangential velocity at which a wheel should rotate in cases other than no turn is equal to

$$|v_{wheel}| = ||C_{wheel}||.|\omega| \tag{6.5}$$

The direction of v_{wheel} is determined form a class of turn and direction of v or direction of ω. Different cases are described in following list.

- Left, sharp left, right and sharp right turn, both sides - $sgn(v_{wheel}) = sgn(v)$.
- In-place turn, right side - $sgn(v_{wheel}) = sgn(\omega)$.
- In-place turn, left side - $sgn(v_{wheel}) = -sgn(\omega)$.
- No turn, both sides - $v_{wheel} = v$.

Angular velocities of motor of the wheels ω_{wheel} can be calculated if radius of the wheel r_{wheel} is know.

$$\omega_{wheel} = \frac{v_{wheel}}{r_{wheel}} \tag{6.6}$$

Line from local origin to center of the turn will be perpendicular to desired direction of wheels. Angle β_{wheel} will equal to angle between direction of x-axis and line from origin to the center of the turn. The values of C_{wheelx} and C_{wheely} will represent position of center point in x and y-axis of local coordinate frame respectively. The value of β_{wheel} is calculated for all classes of turn except no turn.

$$\beta_{wheel} = atan2(C_{wheely}, C_{wheelx}) \tag{6.7}$$

The value of steering angle α_{wheel} will depend on direction of the turn, the type of the turn and the side the wheel is on.

- Left and sharp left turn, both sides - $\alpha_{wheel} = \beta_{wheel} - \frac{\pi}{2}$.
- Right and sharp right turn, both sides - $\alpha_{wheel} = \beta_{wheel} + \frac{\pi}{2}$.
- In-place turn, left side - $\alpha_{wheel} = \beta_{wheel} + \frac{\pi}{2}$.
- In-place turn, right side - $\alpha_{wheel} = \beta_{wheel} - \frac{\pi}{2}$.
- No turn, both sides - $\alpha_{wheel} = 0$

The different angles and center of the turn for front left wheel are shown in the Figure 6.7. Axis x_{wheel} represents local x-axis, axis y_{wheel} represents local y-axis in the coordinate frame of the wheel. All other variables have been described in previous parts of this section. The controlling software is implemented in Python 2.7.

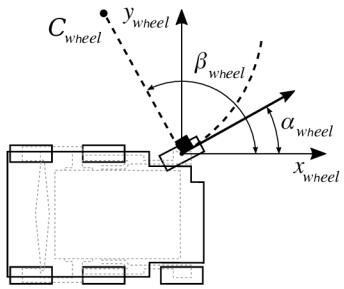


Figure 6.7: Wheel coordinate frame

6.3 Simulation environment

Researched algorithm expects to be supplied by live sensor data. This requires appropriate simulation environment. Provided feature-set guides the choice if this environment.

Gazebo offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments [33]. It provides extensive set of features including tools for robot modelling and environment modeling. It simulates different types of joints, physical properties of robot parts such as mass, collision shape and moments

of inertia. A selection of plugins provides simulation of sensors such as cameras, lasers and depth sensors with ability to introduce measurement error. The most notable feature is integration with ROS. This provides simulator with great amount of packages which supplying control, autonomous planning and SLAM algorithms.

6.3.1 Physics configuration

Multiple facts about real-world features of environment have to be taken into consideration in order to simulate environment properties on planetary surface. The decision was to simulate environment on the Martian surface thus physical properties of this environment have been set. The diameter of planet Mars is approximately $\frac{1}{2}$ to the diameter of the Earth, this results in gravitational acceleration on the surface to be $3.71 m.s^{-1}$ [30]. Gravitational acceleration may affect traction of wheels on the surface. Gazebo allows the definition of gravitational acceleration. A simulation uses a flat-surface model. This means, it does not simulate planet as a approximate spherical model but approximates small subsection using a model of a flat plane. Coordinate frame is defined by world coordinate frame with x-axis in the forward direction, y-axis in the left direction and z-axis upwards. Our use-case will consider local ENU coordinate system with x-axis to the East, y-axis to the North and z-axis Up. Gravitation acceleration towards plane is set as:

$$\begin{bmatrix} 0 & 0 & -3.71 \end{bmatrix} m.s^{-1} \tag{6.8}$$

A negative value is set on the z-axis since direction of z-axis is up and gravity is accelerating down. Mars surface has properties of surface friction yet unknown. We will consider an ideal case scenario where a soil has coefficient of friction $\mu = 1$.

6.3.2 Model of the Martian surface

A 3D model of surface was created for purposes of simulation. Base of the surface is modeled as a textured flat plane of size 200×200 meters. A repeating ground texture pattern is shown in the Figure 6.8. The flat surface is by itself not sufficient for a SLAM algorithm. SLAM algorithm and visual odometry expect features to be present in the surrounding environment. A selection of rocks provides a good feature set. Blender Extra meshes add-on provides a random rock generation capability. Generated rocks are used to create obstacles.

Figure 6.8: Repeating ground texture

The scene is designed using four different sets of rocks. A multiple rock sizes include small rocks (≈ 0.5 meters), medium sized rocks (≈ 2 meters) and large rocks (≈ 5 meters). The scene is surrounded by a range of larger rocks and overhangs. The overall scene size is $\approx 30 \times 25$ meters.

The placement of rocks is done in a pattern providing open spaces, narrow passages and dangerous obstacles which can be seen in the Figure 6.9. This provides a generic set of distinct features that can be used for navigation.

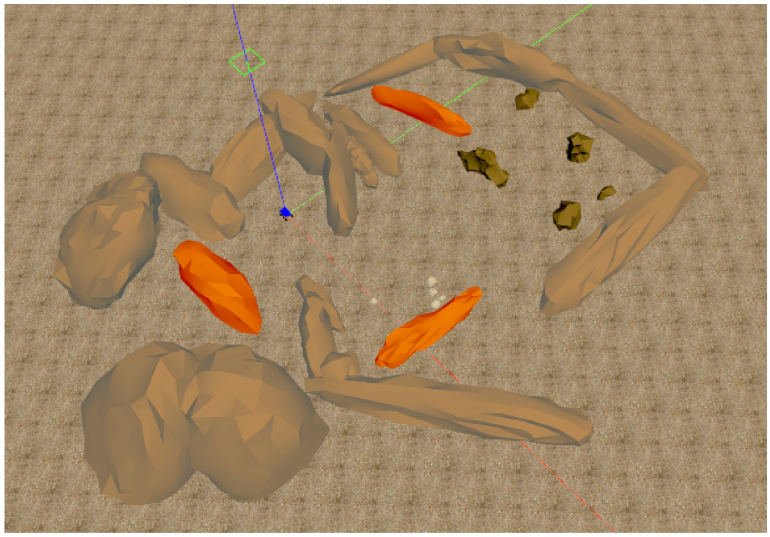


Figure 6.9: Martian surface model for simulation

6.3.3 Light Detection and Ranging sensor model

A LiDAR is simulated using laser simulation plugin. This plugin allows definition of parameter such as range and angular resolution. Noise uses Gaussian distribution with zero mean and standard deviation of 0.01. Modeled LiDAR modeled, the SLAMTEC RPLiDAR A1 has accuracy of 1% of measured distance. Other technical parameters include statically set angular range of 180 degrees limited to $\frac{1}{2}$ of the original range. The number of samples is set to 1000 which is $\frac{1}{2}$ of minimal setting of 2000. The shape of LiDAR is simplified into a basic cylinder.

6.3.4 Stereoscopic camera model

Gazebo simulation provides a module for simulation of multi-camera systems with a multi-camera plugin. This plugin allows definition of two cameras needed to simulate stereoscopic camera. Real-world sensor counterpart is considered to be the Intel® RealSense™ Tracking Camera T265. This sensor provides two wide angle lenses with FOV of 163 degrees. A image with resolution of 848 by 800 pixels is provided by both sensors. Distance between right and left lens is described to be 64 mm. The distortion parameters of image are not taken into account since they were not presented in the data-sheet. A 3D model of camera is represented by a box with dimensions of $24.5 \times 108 \times 12.5$ mm (height $\times$ width $\times$ depth) according to data-sheet [13].

6.3.5 Depth camera model

A current available solutions for depth cameras cameras include family of Intel® RealSense™ Depth Cameras. This family of sensors offers multiple options ranging form different types of Infra-Red (IR) projector, different sensor resolution, differing FOV and presence of Inertial Measurement Unit (IMU).

Intel® RealSense™ Depth Camera D435 is considered for simulation. This sensor provides 86 degrees of horizontal FOV and resolution of 1280x720 depth points. A maximum range of measured distance is around 10 meters a minimum range is 0.105 meters. Information about resolution of distribution of depth perception is not provided in a data-sheet, the same is true for image calibration parameters. 3D model of sensor is represented by box with size of $25 \times 90 \times 25$ mm (height $\times$ width $\times$ depth) according to information defined in data-sheet [14].

The simulation of the sensor is done using Gazebo Openni Kinect plugin. Plugin allows definition of before mentioned parameters defining resolution and depth properties. Plugin has ability to distort image based on parameters described in the Section 4.2.1. All distortion parameters are set to zero due to lack of this information in the data-sheet.

6.3.6 Model of the odometer

Gazebo by itself combines together simulation of functionality of actuators and encoders and provides different options for control. Simulation environment offers implementation in a form of Gazebo differential drive plugin. This plugin is used to control two middle wheels. It is configured to publish odometry data from simulated encoders. The future work can extend on creating a odometry based on true encoders on physical rover. This could be implemented into controlling software. If this approach would be chosen a full implementation of method described in the Section 3.1 or similar would need to be performed.

6.4 Data visualisation

Tool rviz is a 3D software providing visualization capabilities to ROS. It provides visualisation of any ROS topic with supported message types. Image, laser scan and point-cloud are some of supported message formats. Other data types such as octrees are supported by third party packages. A three-dimensional view-port is the main advantage of this form of visualization. View-port is able to display 3D model of rover defined in a URDF description file and 3D representation of sensor data.

A selected configuration of displayed topics can be saved into a configuration file for future use. Topics which are displayed in rviz for our use case include:

- rover 3D model,
- sensor data,
- map.

The Figure 6.10 shows rover model in rviz visualization software. This visualization includes the rover and a map generated by researched and implemented navigation solution.

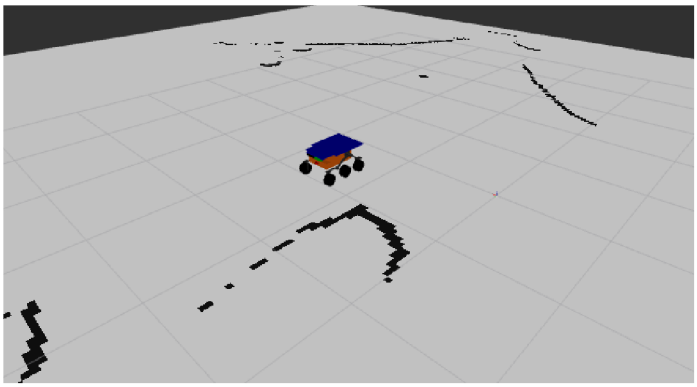


Figure 6.10: The rover model visualized in rviz

6.5 Robot Operating System

Robot Operating System (ROS) is an open-source collection of different tools, packages and conventions which aim to provide simple interface for implementation of Guidance Navigation Control (GNC) and other solutions. This system strongly encourages modular approach and re-usability. It can be used in variety of different applications including UAV, UGV and others. The ROS documentation [34] defines three levels of concepts:

- File-system level - this level contain number of different principles used to identify ROS resources which are present o the disk.
  - Package, a main unit for organizing software in ROS. Package contains processing node, libraries, specific configuration files, data-sets and others.
  - Metapackage, a specialized package which represent a group of related packages.
  - Package Manifests, a metadata about a package. It contains name, version, description, license information, dependencies, and other meta information.
  - Repositories, a collection of packages which share a common Version Control System (VCS) and can be released together.
  - Message type, a description of message, defines the data structures for messages.
  - Service type, a description of service, defines the request and response data structures for services.
- Computation Graph level - this level defines the peer-to-peer network of ROS processes that are processing data together. Computation Graph consist of:
  - Nodes, a processes performing computations.
  - Master, a main node which provides registration and lookup in a computational graph.
  - Parameter Server allows any configuration data to be stored in central location.
  - Message, a named unit of data used for communication of nodes.
  - Topic, a semantic used for communication of messages.
  - Service, a interface providing request/response semantic.
  - Bag, a format used to store and and later playback captured messages.

- Community level - concepts of this level enable separate communities to exchange software and knowledge with each other. Concepts of this level include:
  - Distribution, a collection of packages, and conventions defined by a single version.
  - Repositories, a federated network of code repositories.
  - The ROS Wiki, a documentation of ROS and available packages.
  - Bug Ticket System, Mailing Lists, ROS Answers and Blog.

Our solution will be dependent on ROS distribution Melodic Morenia. This distribution requires use of Linux operating system Ubuntu 18.04 LTS (Bionic Beaver) for which all supported ROS packages for are compiled.

6.5.1 Coordinate frames

Any sensors which provides information about position such as camera, LiDAR, odometry, motor position or range-finder is always measuring in its own coordinate frame. For example LiDAR sensor does only know what is directly in font of it and output is representation of this. Applications such as mapping require data to be in a common coordinate frame.

Transformation tree is a tree structure with nodes representing coordinate frames and edges defining transformations between them. Any parent may have multiple child coordinate frames. A way of node announcing transformation between it and parent frame is through stamped transformation messages [50]. These messages are sent through topics **/tf** [50] and **/tf_static** [35]. Sources of transformation messages can be rover model itself or components of rover such as motors, servos and encoders. Other kind of publishers are for example odometry sources or mapping nodes where a position to a point of origin e. g. starting position on the map or initial odometry position, is published.

Any node which needs sensor information provided by sensor with defined coordinate frame is responsible for transformation of this information into coordinate frame of it's interest. MATLAB supplies helper structures which automatically update locally stored transformation tree and allow any transformation between two nodes in this tree.

The ROS defines a convention of order and naming of the coordinate frames. A specification contains 4 coordinate frames used for different components of navigation system. The transformation between related coordinate frames should be in this order [27]:

- **earth** - A coordinate frame with origin of earth-centered, earth-fixed (ECEF). This reference frame has axis defined as x-axis in direction from center to 0° longitude and 0° latitude, y-axis goes from center to 90° longitude and 0° latitude and z-axis form center to the north pole.

- **map** - A world-fixed reference frame. The map has defined direction of z-axis which is upwards. The orientation of map is dependent on starting conditions. This coordinate frame can have aligned x-axis to the east, y-axis to the north and z-axis upwards if it is referenced directly to frame such as **earth**. The map frame should be positioned on surface of ellipsoid defined by WGS84 in this case.

- **odom** - This coordinate frames is computed using odometry source such as odometer or visual odometry. The usage of specific source may create slip which makes it inaccurate for world navigation. The transformation between **map** and **odom** should adjust for created error.
- **base_link** - A reference frame attached to the base of the robot. It can be attached to robot in any position or orientation. Our use-case will consider axes of **base_link** coordinate frame to be in the forward direction for x-axis, y-axis to the right and z-axis up. The origin is considered to be in the middle of middle axle and at height where rover should contact the ground.

The ROS adds upon these convention with a naming for the suffixes of coordinate frames. The suffixes such as **_enu** and **_optical** define different conventions for axis and use-cases of coordinate frames [9].

We will consider only coordinate frames **map**, **odom** and **base_link**. The **map** coordinate frame will represent coordinate frame with x-axis to the east and y-axis to the north pole of the planet Mars, this coordinate frame will be on the surface with z-axis in upwards direction. The **odom** coordinate frame will be used by selected odometry source to define transformation between the odometry coordinate frame and the **base_link** coordinate frame. The rover by itself will have other coordinate frames for its different parts such as cameras, LiDAR and encoders.

6.6 MATLAB Robot Operating System Toolbox

Researched navigation algorithm is a ROS node. Any MATLAB script which needs to communicate with other ROS node has to firstly initialize a node. This is done through method `rosinit` and equivalent for stopping this node `rosshutdown`. Initialized ROS node allows use of other functionalities for information exchange with other ROS nodes. The data is acquired form other ROS nodes on topics:

- **/laser/scan** - LiDAR scan.
- **/sojourner/odom** - Odometer.
- **/stereo/odom** - Visual odometry.
- **/sojourner/world_odom** - Ground-truth information (used for evaluation).
- **/camera/depth/points** - RGB-D point cloud.
- **/tf** - Transformations between coordinate frames .
- **/tf_static** - Static transformations between coordinate frames .

The node published data to the following list of topics:

- **/matlab_map** - Binary occupancy map.
- **/tf** - Transfromation from **map** to **odom** coordinate frame.

Data is constantly received in an infinite while cycle. Every new iteration a subscriber receives messages in a blocking call.

A configuration parameters which configure sources of information are set first. This is done through parameter server interface provided by toolbox through `rosparam`. Parameters of the navigation node include:

- **publish_map** - Allow publishing of map including transformations and map data.
- **use_rgbd** - Use RGB-D point-cloud filtering and processing as source for SLAM algorithm instead of LiDAR scan.
- **visual_odom** - Configure use of visual odometry instead of odometer.
- **get_ground_truth** - Option to receive and save information about the ground-truth.

MATLAB provides interface for transformation with ROS coordinate frames. This functionality is enabled by firstly initializing new local transformation tree structure with method `rostf`. Structure is used for transforming data and for creation or update of existing transformation tree.

Transformation of three-dimensional point-cloud data supplied by depth camera is done first. Data acquired form sensors contains positions in coordinate frame of sensor. These position are not the same as positions relative to rover body coordinate frame. Point-cloud data is transformed using transformation tree to coordinate frame named `base_link` with method `transform`. This coordinate frame represents center of the middle wheel axle of the rover on a ground plane.

A data processing form previous chapters accrues here. Results are then published. Data published includes two different sources, first a transformation to coordinate frame of map. This requires the use of single publisher for map data and already defined transformation tree structure. A map publisher publishes under topic `/matlab_map` with type of `nav_msgs/OccupancyGrid`.

Created map needs to be converted form probabilistic values into binary values. This is done by through a function `occupancyMatrix` and by comparing its results. Values grater than occupied threshold are stored. Parameter `.OccupiedThreshold` of occupancy map is used to determine occupied cells. A produced logical two-dimensional matrix is used to create binary occupancy map with function `binaryOccupancyMap`. Additional parameters of binary occupancy map need to be inherited from original occupancy map. These include map resolution and positions of local and global map origins. Binary occupancy map is then saved into a message of type `nav_msgs/OccupancyGrid`, the coordinate frame is set to `/map` and timestamp is set to current ROS time provided by `rostime('now')`.

Transformation to coordinate frame of map has to be published in order for map to be usable for navigation. The latest pose in this coordinate frame is a result of SLAM. This pose information is a single three-dimensional vector. Vector contains x and y-position in first two fields and heading in third. A transformation order in transformation tree expects position transform to be expected first, then followed by a rotation. This is a reverse from pose provided form SLAM. Additionally not a direct transformation to coordinate frame called `base_link` but transformation to `odom` (coordinate frame of odometry) coordinate frame is needed. This encouraged creation of two intermediate transformation frames from odometry frame to represent transformation to current position of coordinate frame based on current odometry information. A transformation from two intermediate frames into coordinate frame of map is established next.

Chapter 7

Evaluation of achieved results

This chapter will discuss testing-cases for evaluation of navigation algorithm. Firstly an overview of test-cases will be provided then a evaluation method of different approaches will be discussed. A search for more optimal configuration of researched navigation algorithm will be discussed in a dedicated section. Later sections will then evaluate achieved result and discuss further improvements and possibilities of future work.

7.1 Test cases

A simulation environment described in previous chapter includes an obstacle course. Path through this obstacle course is considered. The test track will consist of right turns, left turns and straight segments. Overall full path will take a shape similar to number eight. The path will start and end in roughly the same place as can be seen in the Figure 7.1.

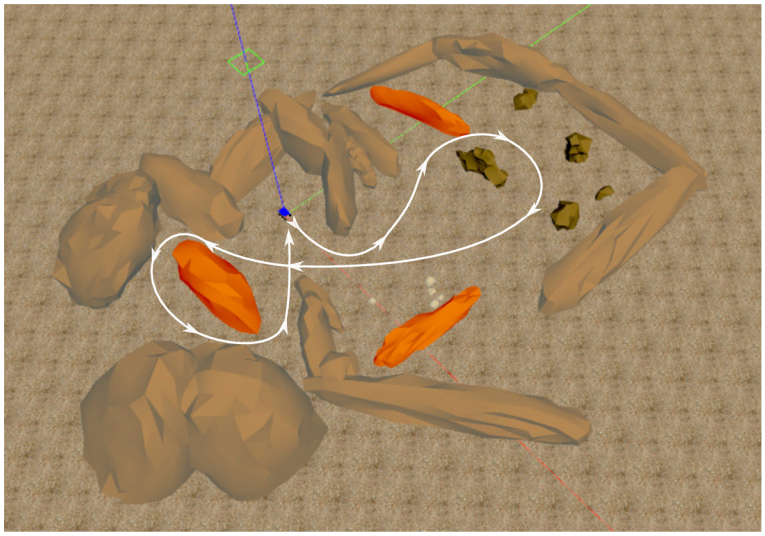


Figure 7.1: Traversed path

The path is traversed by rover and data provided by sensors is recorded through application rosbag. This will provide repeatably of results. Rosbag's playback functionality provides a real-time flow of data as would be present if full simulation was run. Recorded data includes transformation frames, odometry, ground-truth position, visual odometry,

wheel based odometry, LiDAR scan and point-cloud of RGB-D image. Rover will travel at a speed of $0.05m.s^{-1}$ with maximum turn rate of $0.05rad.s^{-1}$.

Evaluation of navigation algorithm will be done for different combinations of sensors. Combinations described in following list will be considered as test-cases.

- LiDAR and odometer.
- LiDAR and visual odometry.
- RGB-D point-cloud and odometer.
- RGB-D point-cloud and visual odometry.

7.2 Statistical evaluation

Navigation data by itself does not provide a measurable parameter for determination of accuracy or quality. In order to estimate the quality of resulting position a Ground-truth (GT) position is used. A position measured relative to GT is considered to be an error in position P_{err}. Calculation is done as difference of GT and position on the path P. Position error is further split into it's part in the x-axis P_{errx} and in the y-axis P_{erry}. Two different statistical methods are used. First method evaluating data is arithmetic mean. Arithmetic mean or average is determined through equation:

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n}(x_i) \tag{7.1}$$

Values of arithmetic mean are calculated for three different parameters. These are P_{errx}, P_{erry} and $||P_{err}|| = \sqrt{{P_{errx}}^2 + {P_{erry}}^2}$, as x. Results for one test-case will be in three different values of arithmetical mean.

Standard deviation is approach used to determine spread of values of error. This is used together with average as a determining factor for quality of given navigation approach. The equation used for calculation of standard deviation is described as

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2} \tag{7.2}$$

This calculation is done for values of P_{errx}, P_{erry} and $||P_{err}||$ as these parameters are of interest. The result thus will be in three different values of standard deviation.

7.3 Search for optimal parameters of navigation algorithm

The SLAM algorithm presented in previous sections allows fine-tuning of two significant parameters. These include loop-closure threshold and search radius of loop-closure detector. The different values of these parameters determine quality of produced pose and map. This requires a optimal set of these two parameters to be found. One approach is to choose a random set of parameters with desired result. Our approach uses method of selected samples to determine more optimal parameters. Initial limits for edges of searched intervals were selected using a random method. The method takes values standard deviation and mean as targets for optimization. These values are considered only for parameter $||P_{err}||$. A search for a local minimal value is done. Functions describing values of standard deviation and mean are rather complex an may be even impossible to describe by analytical form. Values depend on parameters such as data given by sensors, data-rate, current position of the rover and many others. If a consideration of computing power requirements for a single calculation is taken into account a choice of approaches is rather limited.

Table 7.1: Searched intervals and selected samples

		LiDAR	RGB-D
LoopClosureThreshold [1]	Interval	$(300, 450)$	$(500, 2000)$
	Samples	$\{300, 350, 400, 450\}$	$\{500, 1000, 1500, 2000\}$
LoopClosureSearchRadius [m]	Interval	$(1, 7)$	$(2.5, 10)$
	Samples	$\{1, 3, 5, 7\}$	$\{2.5, 5, 7.5, 10\}$

The Table 7.1 shows values used for search of optimal parameters. The approach will take combination of sampled values of LoopClosureThreshold and LoopClosureSearchRadius. Values of standard deviation and arithmetic mean will be evaluated for each combination. A linearly sampled values containing boundaries of intervals are considered. Four samples are taken from each interval which will result in 16 different combinations of values. The sampled values of standard deviation and arithmetic mean for LiDAR can be seen in in the Table 7.2. The sampled values of standard deviation and arithmetic mean for RGB-D can be seen in in the Table 7.3. Sampling was done with wheel based odometry for both cases.

Table 7.2: Arithmetic mean and standard deviation samples for LiDAR

$\overline{x}[m]$		LoopClosureSearchRadius [m]			
		1	3	5	7
LoopClosureThreshold [1]	300	0.8635	12.93	0.4461	0.4949
	350	2.297	0.4543	0.5218	0.5377
	400	3.591	0.3765	0.6149	0.5244
	450	1.591	0.5118	0.5191	0.475

$\sigma[m]$		LoopClosureSearchRadius [m]			
		1	3	5	7
LoopClosureThreshold [1]	300	0.4825	9.32	0.2659	0.3077
	350	1.38	0.3226	0.3534	0.3549
	400	1.525	0.3568	0.3265	0.3311
	450	2.63	0.2474	0.3444	0.3775

Table 7.3: Arithmetic mean and standard deviation samples for RGB-D

$\overline{x}[m]$		LoopClosureSearchRadius [m]			
		2.5	5	7.5	10
LoopClosureThreshold [1]	500	11.99	1.357	0.8367	1.381
	1000	7.074	1.288	0.7481	0.7218
	1500	12.05	0.937	3.076	1.265
	2000	11.89	0.7927	0.9109	1.042

$\sigma[m]$		LoopClosureSearchRadius [m]			
		2.5	5	7.5	10
LoopClosureThreshold [1]	500	8.389	0.858	0.7564	1.186
	1000	20.55	1.183	0.903	0.7924
	1500	8.173	0.9421	3.88	1.107
	2000	8.097	1.193	1.4	0.8509

Values of LoopClosureSearchRadius = $3m$ and LoopClosureThreshold = 400 were selected for LiDAR. Values of LoopClosureSearchRadius = $10m$ and LoopClosureThreshold = 1000 were selected for RGB-D. These values provide a approximate local minimum for a combination of standard deviation and mean error thus making a good option for accurate map creation and localization.

7.4 Results of testing

Each test case mentioned in the Section 7.1 was evaluated using methods mention in the Section 7.2. The parameters of SLAM algorithm were set according to estimated optimum found in the Section 7.3. Results form each test case are provided in a list of tables presented in this section and in appendices. Firstly a an error of odometry source is estimated. This error will be common among all test cases since the same data set is used.

Table 7.4: Odometer statistics

$[m]$	P_{erry}	P_{erry}	$\|\|P_{erry}\|\|$
$\overline{x}$	-3.711	1.036	4.042
σ	3.262	0.607	3.084

Table 7.5: Visual odometry statistics

$[m]$	P_{erry}	P_{erry}	$\|\|P_{erry}\|\|$
$\overline{x}$	1.510	-0.393	1.793
σ	1.112	0.942	1.158

A rather large error is expected to be present when only odometry source is used for navigation. This is especially true for odometer as can be seen in the Table 7.4. Visual odometry is there to improve precision of position tracking and as can be seen in the Table 7.5 a error is smaller then in case of odometer. Result for navigation algorithm will be considered acceptable if a value of mean or standard deviation for $||P_{err}||$ is smaller than these values.

The test case of LiDAR with a use of odometer is different from one used to determine optimal parameters. This allows us to determine expected difference for selected parameters. The overall values are better in our case and are within $0.1m$ of ones measured previously. The performance of algorithm in this test-case can be seen in the Appendix A. Overall mean value of error is improved from odometer by $3.7794m$. The value of standard deviation is improved by $2.9467m$ as can be seen in the Table 7.6.

Table 7.6: LiDAR and odometer statistics

$[m]$	P_{erry}	P_{erry}	$\|\|P_{erry}\|\|$
$\overline{x}$	0.208	-0.006	0.263
σ	0.147	0.152	0.137

The premise of visual odometry is to provide greater accuracy independent of physical properties of soil. The accuracy of it alone is better than accuracy of odometer. This accuracy is improved more in case of used SLAM solution. The produced map with combination of visual odometry and LiDAR can be seen in the Appendix B. The gain in accuracy is notable even for visual odometry where mean error is reduced by $1.5583m$ and standard deviation is improved by $1.0390m$. This can be seen in the Table 7.7. Overall performance is slightly improved from previous combination of LiDAR and odometer.

Another information is present in both the Table 7.6 and the Table 7.7 is the difference between mean value of error for x-axis and y-axis. The value is significantly larger in case of x-axis. The reason to this phenomena can be explained by considering previously known information. The method used for transformation of data into coordinate frame of rover cannot transform LiDAR scan. This will result in scan being misaligned with position on the map. The difference in position of LiDAR and coordinate frame of rover is $0.215m$.

Table 7.7: LiDAR and visual odometry statistics

$[m]$	P_{erry}	P_{erry}	$\|\|P_{erry}\|\|$
$\overline{x}$	0.147	0.015	0.235
σ	0.142	0.165	0.119

A RGB-D point-cloud is sourced from camera with horizontal FOV of 86 degrees. This is in comparison to LiDAR less than a half. A result of this is smaller size of of samples and less surrounding information. The SLAM algorithm depends on as much of this information as possible. The expected result is then a lower accuracy. The main goal is in this case to achieve more accurate position information than with odometry alone. This can be seen with odometer in the Appendix C. The overall accuracy is lower than with LiDAR. The accuracy compared to odometry is improved. This is true for both mean error where it is $3.3957m$ lower and standard deviation which is $2.8424m$ lower as can be seen in the Table 7.8.

The last test case takes into account visual odometry and RGB-D point-cloud. The overall results of standard deviation and mean error are higher then for odometer. This can be due to selection of parameters optimised for use with odometer. The improvements according to visual odometry can be still considered significant. Relative improvements to visual odometry are of $0.7277m$ in case of standard deviation and $0.9465m$ in case of mean. This can be seen in the Table 7.9. Overall map and position graph present in the Appendix

Table 7.8: RGB-D point-cloud and odometer statistics

$[m]$	P_{erry}	P_{erry}	$\|\|P_{erry}\|\|$
$\overline{x}$	0.018	-0.044	0.646
σ	0.504	0.471	0.241

D is less accurate than in case of LiDAR. The main significant inaccuracy is caused towards first sharp right turn. This might be caused by small amount of detected features when turn begins. A inaccuracy of odometry is also significant in this section which might increase overall inaccuracy for SLAM.

Table 7.9: RGB-D point-cloud and visual odometry statistics

$[m]$	P_{erry}	P_{erry}	$\|\|P_{erry}\|\|$
$\overline{x}$	-0.058	-0.578	0.847
σ	0.527	0.538	0.430

The Table 7.10 shows an overview of best to worst performing navigation approaches discussed in this section. This table provides new set of values describes as improvement over reference. These two values present a difference of error produced by a selected odometry source and an error produced by the SLAM algorithm. The best approach is a combination of LiDAR and visual odometry. The best improvement over odometry source is offered by combination of LiDAR and odometer. The more unexpected result is in case of RGB-D and visual odometry where better reference position did not result in improvement over combination with odometer. The overall worst performer is plain odometer. Combination with visual odometry or odometer have seen small differences for both LiDAR and RGB-D. The greatest difference between the use of visual odometry and odometer can be seen in case of the RGB-D. The RGB-D point-cloud has seen a worse result for the visual odometry and better for the odometer. Overall conclusion of this result is that the usage of SLAM increases the accuracy over both the visual odometry and the odometer.

Table 7.10: Best to worst performing results

#	Combination	$\|\|P_{err}\|\|$		Improvement over reference	
		$\overline{x}[m]$	$\sigma[m]$	$\overline{x}[m]$	$\sigma[m]$
1.	LiDAR and visual odometry	0.2348	0.1188	1.5583	1.0390
2.	LiDAR and odometer	0.2627	0.1368	3.7794	2.9467
3.	RGB-D and odometer	0.6464	0.2411	3.3957	2.8424
4.	RGB-D and visual odometry	0.8467	0.4301	0.9465	0.7277
5.	Visual odometry	1.7932	1.1578	-	-
6.	Odometer	4.0421	3.0835	-	-

Chapter 8

Conclusion

Autonomous navigation of unmanned vehicles is essential part of unmanned missions. This problem has seen a development in recent years. A selection of navigation principles used by planetary rovers have been researched in this thesis. The focus was on problematic of localization in an environment. The work later researched the problem of localization and mapping in a form of SLAM.

A navigation algorithm has been integrated to take advantage of sensors available to a planetary rover. This algorithm was built on top of an existing SLAM approach and added filtering of input data supplied from sensors. This solution was implemented in MATLAB with selection of toolboxes.

A rover model was constructed in the Gazebo simulation environment. Robot was based on fist planetary rover Sojourner. A focus was given towards control of designed rover platform. The bare-bones platform was then equipped with a selection of sensors measuring different parameters of surrounding environment. Sensor selection included depth camera, stereoscopic camera and LiDAR.

Algorithm was integrated as a ROS node in order to take advantage of data provided by Gazebo simulation environment. Output map and pose were then published for other ROS nodes to take advantage of. A list of parameters configurable through ROS was implemented to setup navigation solution.

The solution was tested in a designed scene. This scene contained multiple obstacles and provided simulation of Mars surface. Combinations of different sensors were tested on a test track. Results of these tests were then statistically evaluated in comparison to optimal solution in a form of ground-truth. A optimal configuration for selected SLAM approach was determined first. Different sensor configurations were then tested with determined optimal configuration and results determining performance of these configurations have been presented. The best performing result was determined to be a combination of LiDAR and visual odometry.

8.1 Future improvements

Results of this work provide multiple different areas which may be of interest for future work. A navigation algorithm itself can be improved to provide accurate positions for larger landscapes. A position in latitude longitude and altitude is a better navigation solution for long range missions. This future goal can be achieved with use of different sensor such as stereo camera and algorithms for feature detection in images. A new algorithm can be

designed for this case. The rover platform can be improved to contain IMU. This would make it possible to include attitude as a part of location information. Information about changes in altitude provides the ability to classify steep slopes, which are not accessible by rover, as obstacles.

Other area of improvements can be towards more optimal results when a RGB-D point-cloud is used. This can be achieved by use of multiple depth cameras or with combination of data from multiple sources. Main goal would be to increase FOV around the rover. Other approach can use three dimensional scan matching and projection of features detected as obstacles onto a two-dimensional map can be used.

A provided tensing environment can be extended to contain a three-dimensional surface model with different set of obstacles or more open areas for long-range navigation. Rover platform provides a base for testing of other navigation approaches such V-SLAM or other three-dimensional SLAM approaches. The option of this platform being transformed into real-world model can be exercised to provide real-world testing results.

A different area where improvement can be made is in visualization software. Package rviz provides extensive functionality for display of data. The main improvement in visualization software can be the ability to visualize three dimensional model of terrain or direct rover control with mission objectives or other goals.

Bibliography

[1] Amos, J. *China's Tianwen-1 Mars rover rockets away from Earth.* BBC, Jul 2020 [cit. 25. July 2020]. Available at: https://www.bbc.com/news/science-environment-53504797.

[2] National Aeronautics and Space Administration. *Autonomous Planetary Mobility - NASA Mars* [online]. [cit. 18. January 2020]. Available at: https://mars.nasa.gov/mer/mission/technology/autonomous-planetary-mobility/.

[3] Barbosa, w. b. R. C. and Barbosa, R. C. *China lands Chang'e-4 mission on the far side of the Moon.* Jan 2019. Available at: https://www.nasaspaceflight.com/2019/01/china-returning-moon-change-4-mission/.

[4] Bence, M. *Diff_drive_controller - ROS Wiki.* Open Source Robotics Foundation [cit. 25. February 2020]. Available at: http://wiki.ros.org/diff_drive_controller.

[5] National Aeronautics and Space Administration. *Brains | Rover – NASA's Mars Exploration Program* [online]. [cit. 22. January 2020]. Available at: https://mars.nasa.gov/msl/spacecraft/rover/brains/.

[6] National Aeronautics and Space Administration. *Cameras | Rover – NASA's Mars Exploration Program* [online]. [cit. 22. January 2020]. Available at: https://mars.nasa.gov/msl/spacecraft/rover/cameras/.

[7] Carsten, J., Rankin, A., Ferguson, D. and Stentz, A. *Global Planning on the Mars Exploration Rovers: Software Integration and Surface Testing* [online]. 2009 [cit. 25. January 2020]. Available at: https://www-robotics.jpl.nasa.gov/publications/Joseph_Carsten/dstar_jfr09_final.pdf.

[8] DataGenetics. *Ackerman Steering* [online]. 2016 [cit. 21. February 2020]. Available at: http://datagenetics.com/blog/december12016/index.html.

[9] Foote, T. and Purvis, M. *Standard Units of Measure and Coordinate Conventions.* Stanford Artificial Intelligence Laboratory et al., Oct 2010. [cit. 21. July 2020]. Available at: https://www.ros.org/reps/rep-0103.html.

[10] Harvey, B. *Soviet and Russian lunar exploration.* 1st ed. Springer and Praxis Publishing, 2007. ISBN 0387218963.

[11] Hess, W., Kohler, D., Rapp, H. and Andor, D. Real-time loop closure in 2D LIDAR SLAM. In: IEEE. *2016 IEEE International Conference on Robotics and Automation (ICRA).* 2016, p. 1271–1278.

[12] Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C. and Burgard, W. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous robots*. Springer. 2013, vol. 34, no. 3, p. 189–206.

[13] Intel®. *Intel® RealSense™ Tracking Camera Datasheet* [online]. September 2019 [cit. 30. June 2020]. Available at: https://www.intelrealsense.com/wp-content/uploads/2019/09/Intel_RealSense_Tracking_Camera_Datasheet_Rev004_release.pdf.

[14] Intel®. *Intel® RealSense™ Product Family D400 Series Datasheet* [online]. June 2020 [cit. 30. June 2020]. Available at: https://www.intelrealsense.com/wp-content/uploads/2020/06/Intel-RealSense-D400-Series-Datasheet-June-2020.pdf.

[15] Labbé, M. and Michaud, F. RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *Journal of Field Robotics*. Wiley Online Library. 2019, vol. 36, no. 2, p. 416–446.

[16] Litomisky, K. Consumer rgb-d cameras and their applications. *Rapport technique, University of California*. 2012, vol. 20.

[17] National Aeronautics and Space Administration. *MAHLI | Instruments – NASA's Mars Exploration Program* [online]. [cit. 22. January 2020]. Available at: https://mars.nasa.gov/msl/spacecraft/instruments/mahli/.

[18] National Aeronautics and Space Administration. *MARDI | Instruments – NASA's Mars Exploration Program* [online]. [cit. 22. January 2020]. Available at: https://mars.nasa.gov/msl/spacecraft/instruments/mardi/.

[19] National Aeronautics and Space Administration. *Mastcam | Instruments – NASA's Mars Exploration Program* [online]. [cit. 22. January 2020]. Available at: https://mars.nasa.gov/msl/spacecraft/instruments/mastcam/.

[20] MathWorks®. *Computer Vision Toolbox Documentation* [online]. 2020 [cit. 3. May 2020]. Available at: https://www.mathworks.com/help/vision/.

[21] MathWorks®. *Create occupancy map with probabilistic values - MATLAB* [online]. 2020 [cit. 2. July 2020]. Available at: https://www.mathworks.com/help/nav/ref/occupancymap.html.

[22] MathWorks®. *Downsample a 3-D point cloud - MATLAB* [online]. 2020 [cit. 1. July 2020]. Available at: https://www.mathworks.com/help/vision/ref/pcdownsample.html.

[23] MathWorks®. *Find points within a region of interest in the point cloud - MATLAB* [online]. 2020 [cit. 1. July 2020]. Available at: https://www.mathworks.com/help/vision/ref/pointcloud.findpointsinroi.html.

[24] MathWorks®. *Fit plane to 3-D point cloud - MATLAB* [online]. 2020 [cit. 1. July 2020]. Available at: https://www.mathworks.com/help/vision/ref/pcfitplane.html.

[25] MathWorks®. *Navigation Toolbox Documentation* [online]. 2020 [cit. 3. May 2020]. Available at: https://www.mathworks.com/help/nav/.

[26] MathWorks®. *Perform localization and mapping using lidar scans - MATLAB* [online]. 2020 [cit. 3. July 2020]. Available at: https://www.mathworks.com/help/nav/ref/lidarslam.html.

[27] Meeussen, W. *Coordinate Frames for Mobile Platforms*. Stanford Artificial Intelligence Laboratory et al., Oct 2010 [cit. 21. July 2020]. Available at: https://www.ros.org/reps/rep-0105.html.

[28] Morrison, J. and Nguyen, T. *On-board software for the mars pathfinder microrover* [online]. 1996 [cit. 25. November 2019]. Available at: https://www-robotics.jpl.nasa.gov/publications/Tam_Nguyen/SWROVER.pdf.

[29] Mur-Artal, R. and Tardós, J. D. ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras. *CoRR*. 2016, abs/1610.06475. Available at: http://arxiv.org/abs/1610.06475.

[30] National Aeronautics and Space Administration. *By the Numbers | Mars - NASA Solar System Exploration* [online]. [cit. 20. June 2020]. Available at: https://solarsystem.nasa.gov/planets/mars/by-the-numbers/.

[31] National Aeronautics and Space Administration. *Mars 2020 Perseverance Rover*. 2020 [cit. 30. July 2020]. Available at: https://mars.nasa.gov/mars2020/.

[32] National Oceanic and Atmospheric Administration (NOAA) Coastal Services Center. *Lidar 101: An Introduction to Lidar Technology, Data, and Applications.* [online]. 2012 [cit. 21. February 2020]. Available at: https://coast.noaa.gov/data/digitalcoast/pdf/lidar-101.pdf.

[33] Open Source Robotics Foundation. *Gazebo* [online]. [cit. 10. July 2020]. Available at: http://gazebosim.org/.

[34] Open Source Robotics Foundation. *ROS/Concepts - ROS Wiki* [online]. Open Source Robotics Foundation [cit. 6. June 2020]. Available at: http://wiki.ros.org/ROS/Concepts.

[35] Open Source Robotics Foundation. *Tf2/Migration - ROS Wiki* [online]. Open Source Robotics Foundation [cit. 6. June 2020]. Available at: http://wiki.ros.org/tf2/Migration#Addition_of_.2BAC8-tf_static_topic.

[36] OpenCV. *Camera Calibration and 3D Reconstruction* [online]. [cit. 8. March 2020]. Available at: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html.

[37] OpenCV. *Camera calibration With OpenCV* [online]. [cit. 8. March 2020]. Available at: https://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html.

[38] National Aeronautics and Space Administration. *Overview - NASA Mars: Mars Exploration Rovers Overview* [online]. [cit. 18. January 2020]. Available at: https://mars.nasa.gov/mer/mission/overview/.

[39] PointClouds.org. *Point Cloud Library (PCL): Module filters.* [cit. 2. July 2020]. Available at: https://pointclouds.org/documentation/group__filters.html.

[40] Rincon, P. *China lands Jade Rabbit robot rover on Moon.* BBC, Dec 2013. Available at: https://www.bbc.com/news/science-environment-25356603.

[41] Rusu, R. B. and Cousins, S. 3D is here: Point Cloud Library (PCL). In: *IEEE International Conference on Robotics and Automation (ICRA).* Shanghai, China: [b.n.], May 9-13 2011.

[42] Jet Propulsion Laboratory and California Institute of Technology and the National Aeronautics and Space Administration. *Description of the Rover Sojourner* [online]. 1996 [cit. 25. November 2019]. Available at: https://mars.jpl.nasa.gov/MPF/rover/descrip.html.

[43] Space Encyclopedia ASTROnote. *Lunokhod crews* [online]. 2020 [cit. 10. January 2020]. Available at: http://www.astronaut.ru/luna/crew.htm.

[44] Stone, H. W. *Mars Pathfinder Microrover: A Small, Low-Cost, Low-Power Spacecraft* [online]. 1996 [cit. 25. November 2019]. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.50.4648&rep=rep1&type=pdf.

[45] National Aeronautics and Space Administration. *Summary | Instruments – NASA's Mars Exploration Program* [online]. [cit. 22. January 2020]. Available at: https://mars.nasa.gov/msl/spacecraft/instruments/summary/.

[46] National Aeronautics and Space Administration. *Summary | Rover – NASA's Mars Exploration Program* [online]. [cit. 22. January 2020]. Available at: https://mars.nasa.gov/msl/spacecraft/rover/summary/.

[47] National Aeronautics and Space Administration. *The Panoramic Camera (Pancam) - NASA Mars* [online]. [cit. 18. January 2020]. Available at: https://mars.nasa.gov/mer/mission/instruments/pancam/.

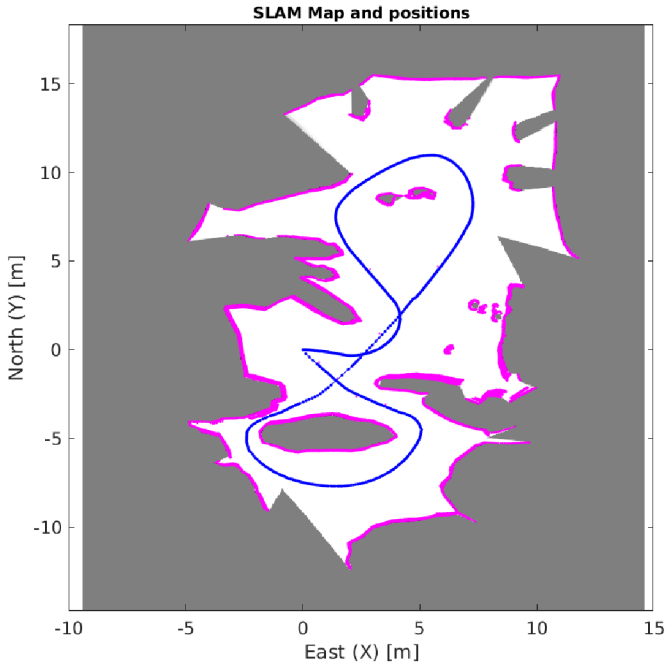
[48] National Aeronautics and Space Administration. *The Rover's "Eyes" and Other "Senses" - NASA Mars* [online]. [cit. 18. January 2020]. Available at: https://mars.nasa.gov/mer/mission/rover/eyes-and-senses/.

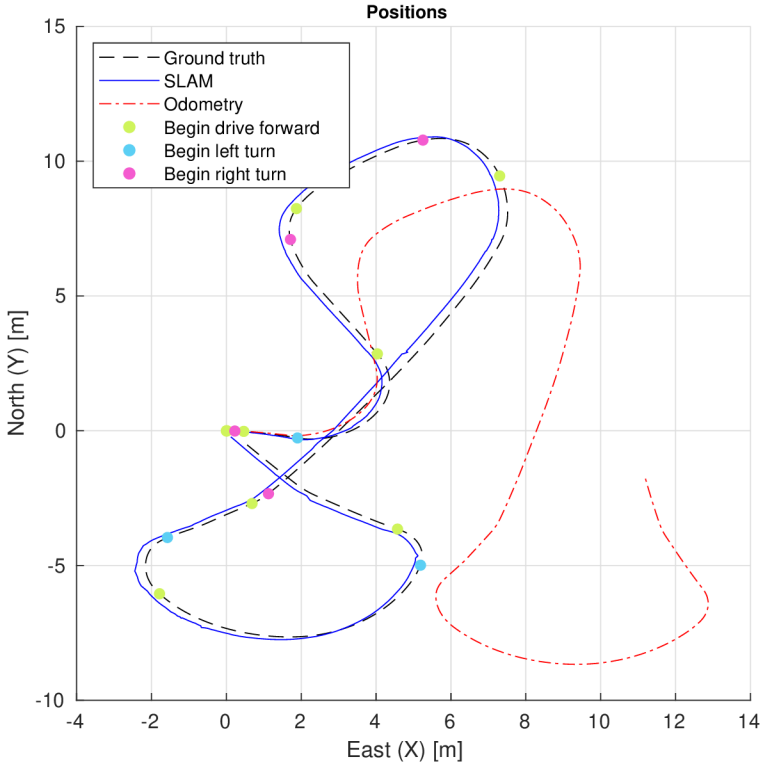
[49] Thrun, S., Burgard, W., Fox, D. and Arkin, R. *Probabilistic Robotics.* MIT Press, 2005. Intelligent Robotics and Autonomous Agents series. Available at: https://books.google.sk/books?id=2Zn6AQAAQBAJ. ISBN 9780262201629.

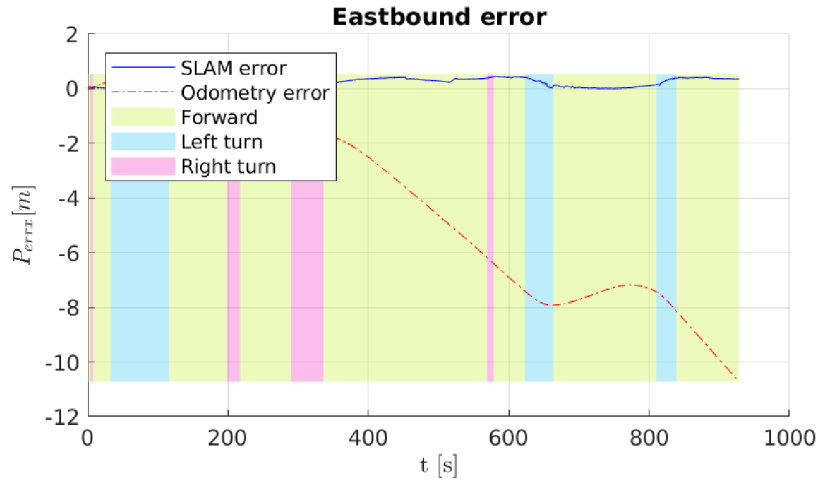
[50] Tully, F., Eitan, M.-E. and Meeussen, W. *Tf - ROS Wiki* [online]. Open Source Robotics Foundation [cit. 6. June 2020]. Available at: http://wiki.ros.org/tf.

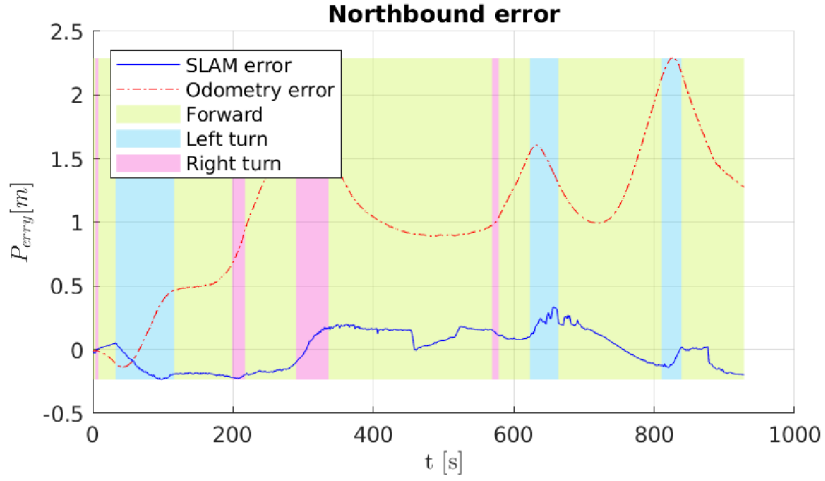
Appendix A

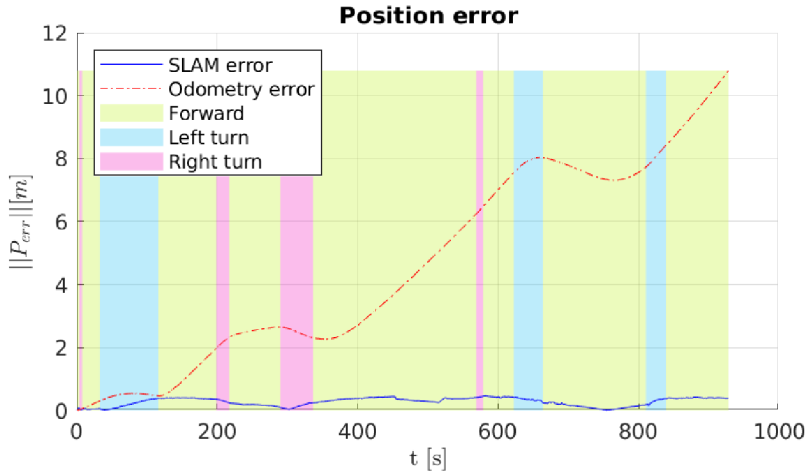
LiDAR and odometer


SLAM Map and positions
North (Y) [m]
East (X) [m]
15
10
5
0
-5
-10
-10
-5
0
5
10
15


Positions
Ground truth
SLAM
Odometry
Begin drive forward
Begin left turn
Begin right turn
North (Y) [m]
East (X) [m]
15
10
5
0
-5
-10
-4
-2
0
2
4
6
8
10
12
14

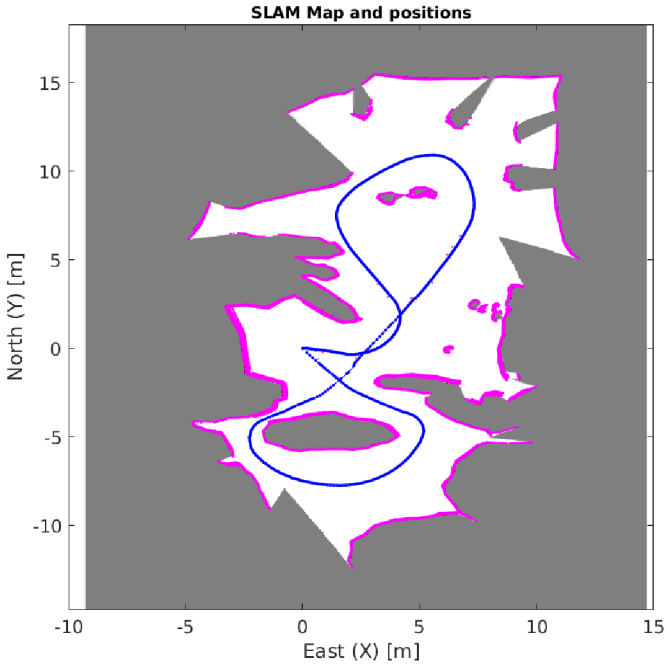

Eastbound error
SLAM error
Odometry error
Forward
Left turn
Right turn
$P_{errx}[m]$
t [s]

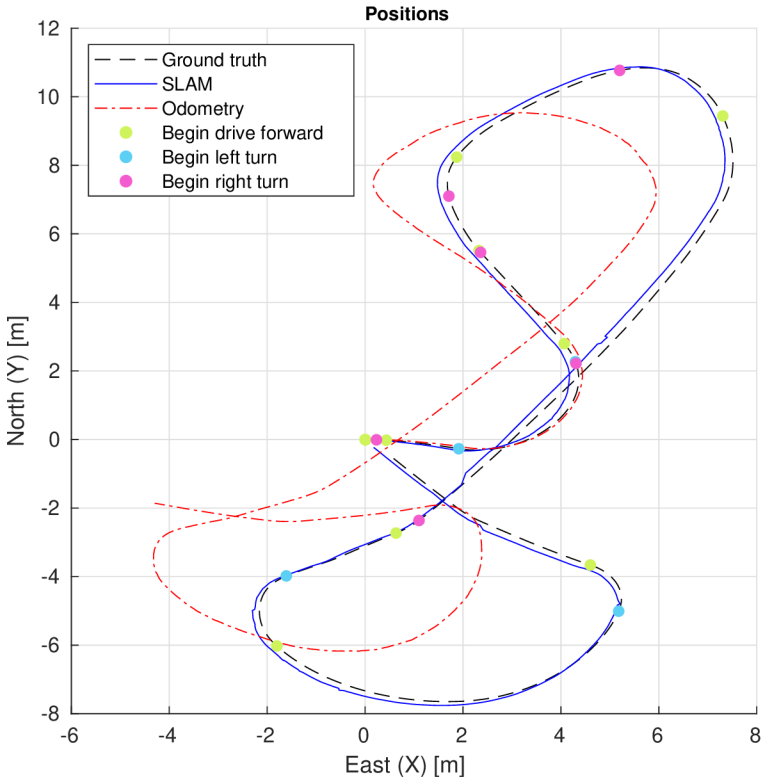

Northbound error
SLAM error
Odometry error
Forward
Left turn
Right turn
$P_{erry}[m]$
t [s]

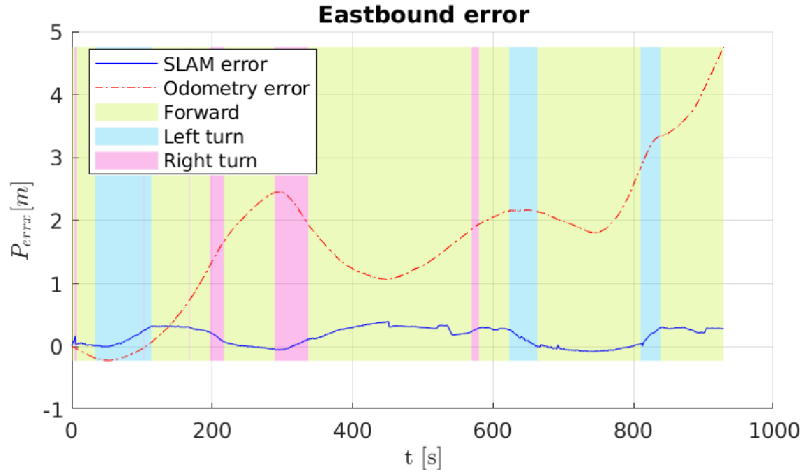

Position error
SLAM error
Odometry error
Forward
Left turn
Right turn
$||P_{err}||[m]$
t [s]

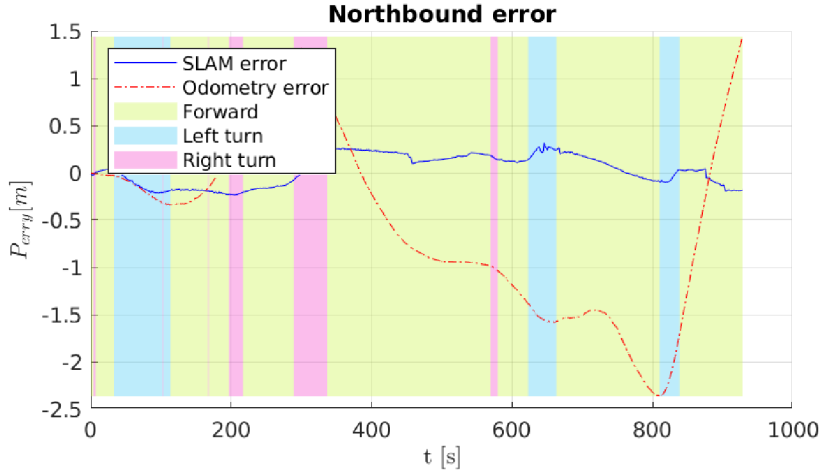
Appendix B

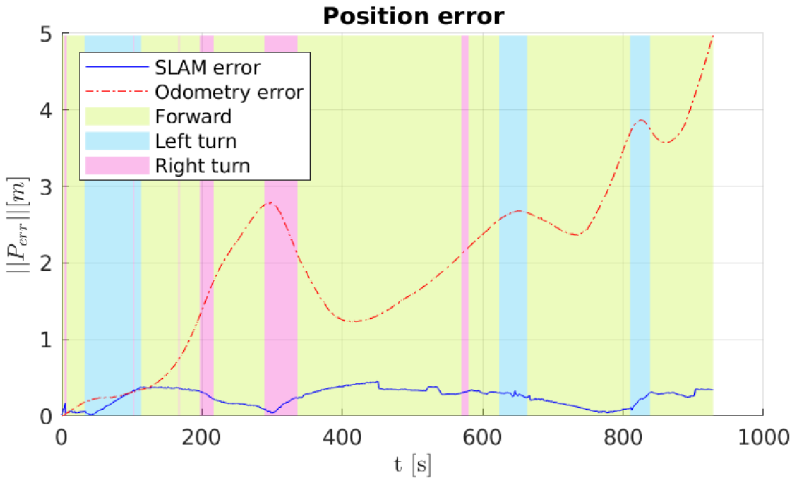
LiDAR point-cloud and visual odometry


SLAM Map and positions
North (Y) [m]
East (X) [m]
15
10
5
0
-5
-10
-10
-5
0
5
10
15


Positions
Ground truth
SLAM
Odometry
Begin drive forward
Begin left turn
Begin right turn
North (Y) [m]
East (X) [m]
12
10
8
6
4
2
0
-2
-4
-6
-8
-6
-4
-2
0
2
4
6
8

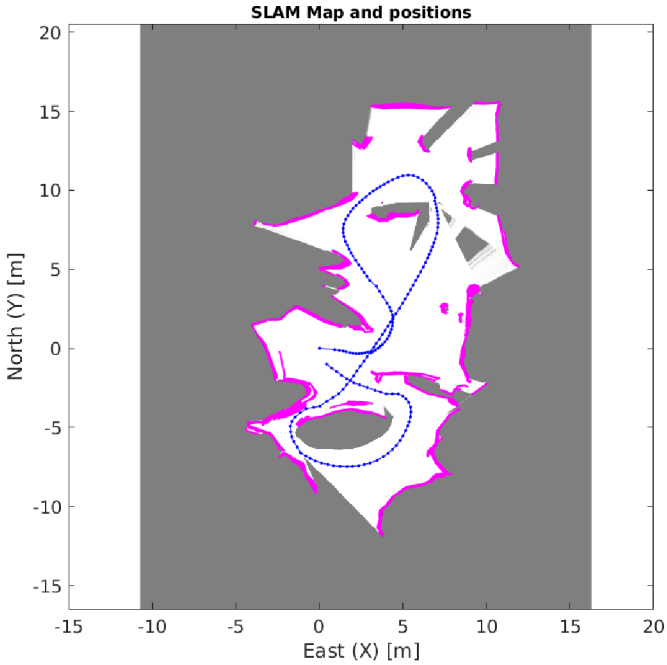

Eastbound error
SLAM error
Odometry error
Forward
Left turn
Right turn
$P_{errx}[m]$
t [s]

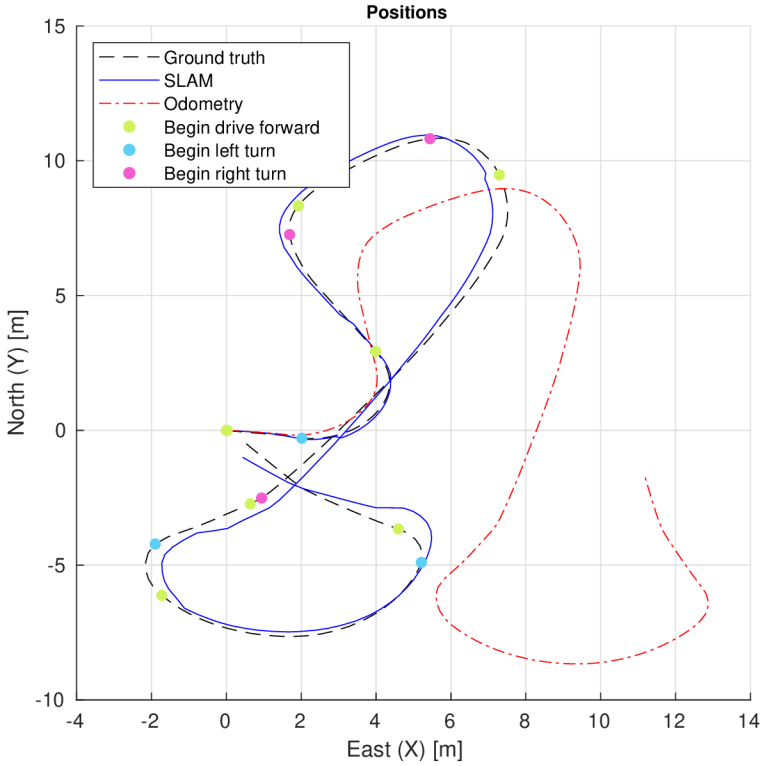

Northbound error
SLAM error
Odometry error
Forward
Left turn
Right turn
$P_{erry}[m]$
t [s]

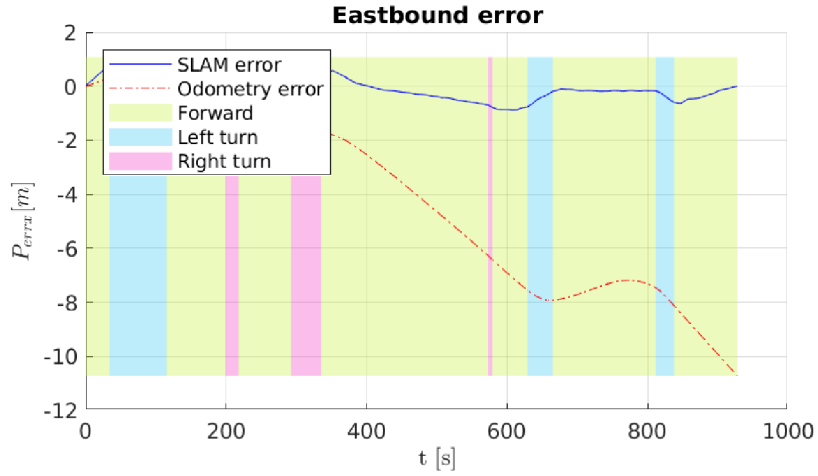

Position error
SLAM error
Odometry error
Forward
Left turn
Right turn
$||P_{err}||[m]$
t [s]

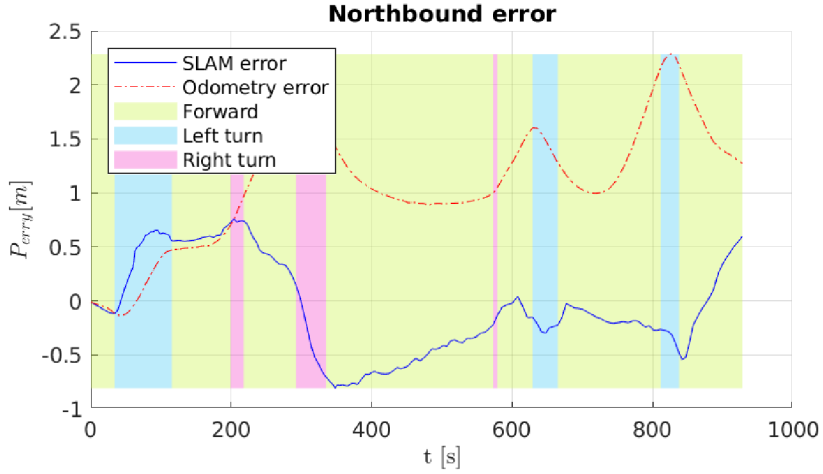
Appendix C

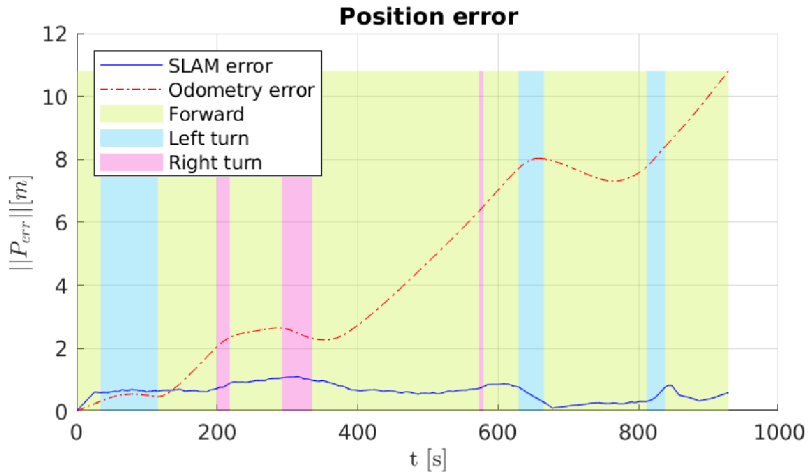
RGB-D and odometer


SLAM Map and positions
North (Y) [m]
East (X) [m]
20
15
10
5
0
-5
-10
-15
-15
-10
-5
0
5
10
15
20


Positions
Ground truth
SLAM
Odometry
Begin drive forward
Begin left turn
Begin right turn
North (Y) [m]
East (X) [m]
15
10
5
0
-5
-10
-4
-2
0
2
4
6
8
10
12
14

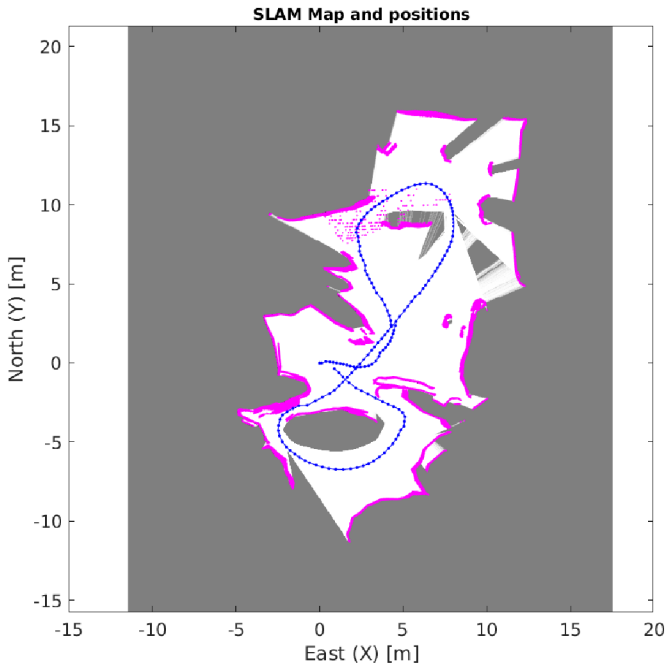

Eastbound error
SLAM error
Odometry error
Forward
Left turn
Right turn
$P_{errx}[m]$
t [s]

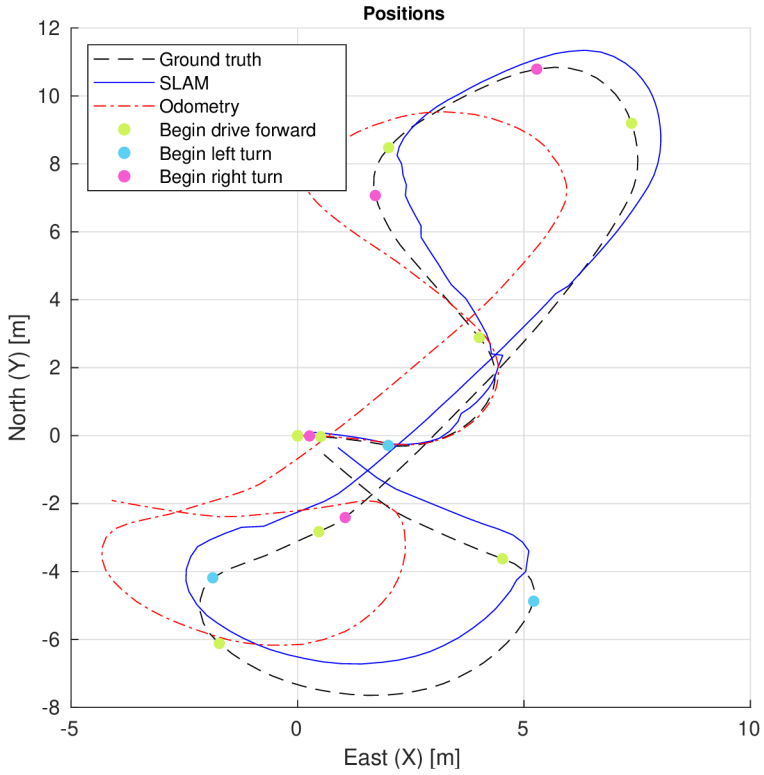

Northbound error
SLAM error
Odometry error
Forward
Left turn
Right turn
$P_{erry}[m]$
t [s]

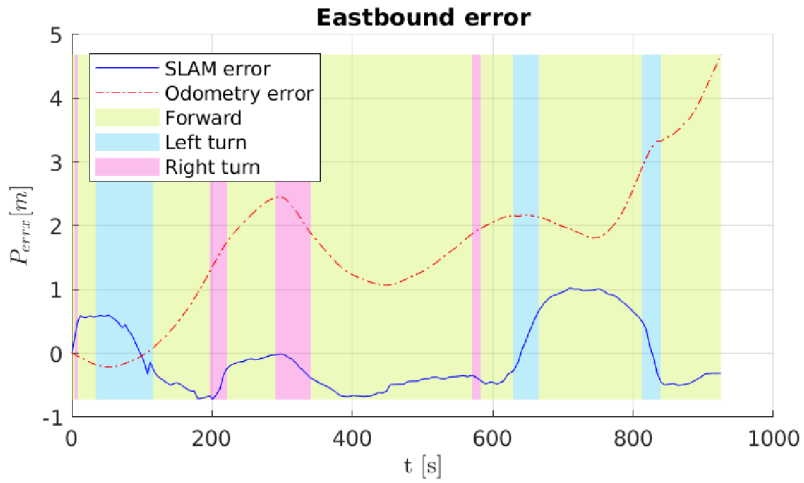

Position error
SLAM error
Odometry error
Forward
Left turn
Right turn
$||P_{err}||[m]$
t [s]

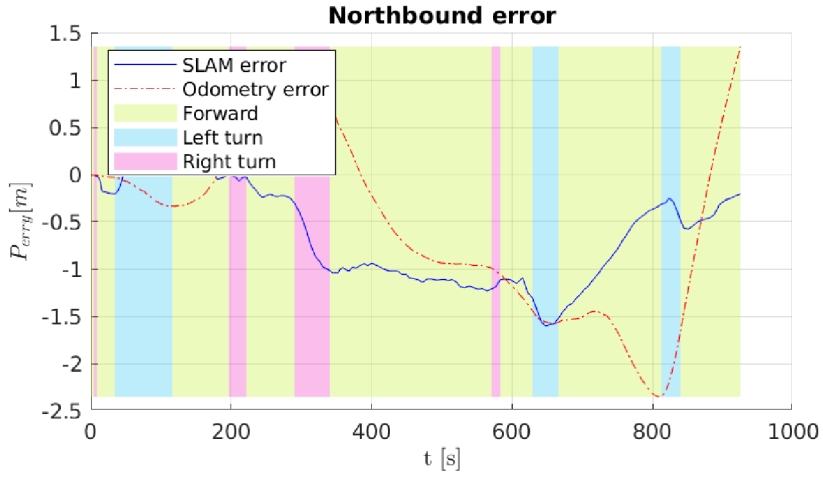
Appendix D

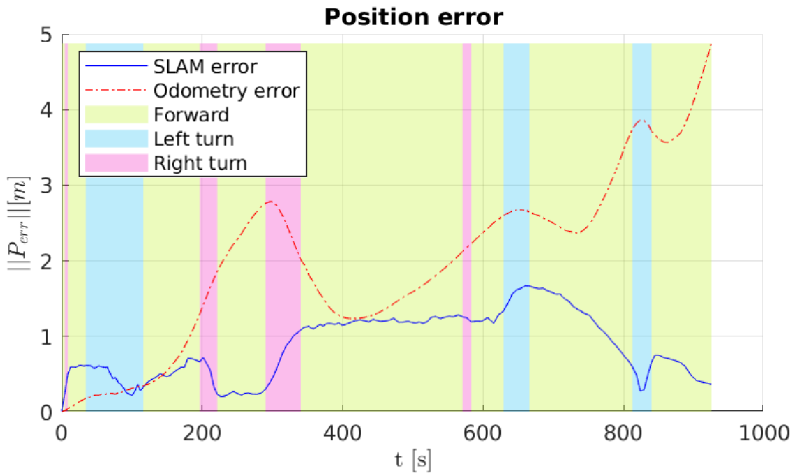
RGB-D point-cloud and visual odometry


SLAM Map and positions
North (Y) [m]
East (X) [m]
20
15
10
5
0
-5
-10
-15
-15
-10
-5
0
5
10
15
20


Positions
Ground truth
SLAM
Odometry
Begin drive forward
Begin left turn
Begin right turn
North (Y) [m]
East (X) [m]
12
10
8
6
4
2
0
-2
-4
-6
-8
-5
0
5
10


Eastbound error
SLAM error
Odometry error
Forward
Left turn
Right turn
$P_{errx}[m]$
t [s]


Northbound error
SLAM error
Odometry error
Forward
Left turn
Right turn
$P_{erry}[m]$
t [s]


Position error
SLAM error
Odometry error
Forward
Left turn
Right turn
$||P_{err}||[m]$
t [s]