KTH Royal Institute of Technology

MF2059

Mechatronics Advanced Course

Fall Semester Report

Authors:
Bryndís Björnsdóttir, Johan Ehrenfors, Martin Gylling, Stefan Ionescu, Nils Jörgensen, Kayla Kearns, Hugo Skeppström, Bohan Zhou

December 16, 2018

Abstract

The following report details the 2018 HK capstone project for the *Engineering Design, Mechatronics track* by Team Trosam. Completed for *Trosam Automation AB*, the report aims to provide insight into the development process of a fully autonomous path following robot. The report is an assessment for the fall semester component of the project, from course *MF2059*, which is a continuation of the project that was started during the spring semester in course *MF2058*.

The report will provide a brief introduction to the project, the (public) requirements agreed upon with the stakeholders, a summary of the current state-of-the-art, the prototype design process, and an evaluation of the prototype's capabilities.

The aim was to create a prototype that can function in a sparse and GPS-denied outdoor environment. The tool used by the robot must be positioned with a high precision and accuracy, ideally to the centimetre. The *Velodyne VLP-16* LiDAR, *Stereolabs ZED* stereoscopic camera, and *Hillcrest BNO-080* IMU were chosen as sensors for the application. The sensors were used with implementations of the ORB-SLAM2 and LeGO-LOAM algorithms, for which the performance has been tested on some campus parking lots.

ORB-SLAM2, using the ZED camera, does not work well in the sparse environments it was tested in, lacking sufficient features to continuously localise itself in all situations, but worked well in more cluttered environments. LeGO-LOAM, using the LiDAR, was able to consistently and smoothly detect the robot pose, but may have trouble in even sparser environments due to resolution issues. Supposedly, the IMU should be possible to add for improved performance, but the team was unable to integrate it. Instead the IMU was used to make sure the 2D map plane was orthogonal to the gravity vector.

The final prototype was not quite able to position the tool with centimetre level accuracy, in part due to movement instabilities caused by the chassis design. Despite these drawbacks, it shows promise for a proposed commercial product which will be developed based on these results.

Acknowledgements

We would like to thank our stakeholders, Joakim Trobeck and Ragnvald Løkholm-Alvestad from Trosam Automation AB for going over and above our expectations in providing support, and engineering knowledge. We thoroughly enjoyed their commitment to the project and their excitement compounded with ours provided a great working environment.

We would also like to thank the Mechatronics Department at KTH and our supervisor Naveen Mohan for their continuous support, keeping us on track with the project. Special thanks to Björn Möller and Vicki Derbyshire for organising the HK project and allowing it to run smoothly, and to Staffan Qvarnström, Hans Johansson, Thomas Östberg, Stork Drives AB, and Yoshua Nava for their technical expertise. A warm gratitude goes to Velodyne LiDAR as well for donating a VLP-16 LiDAR to KTH.

Thanks also to Raúl Mur-Artal (ORB-SLAM2), Tixiao Shan and Brendan Englot (LeGO-LOAM) for their open source code allowing us to implement our SLAM Algorithms. Further thanks to the ROS open-source community for providing a development platform.

Finally thanks to our friends and family for being there when it mattered the most.

Team Trosam
Stockholm, December 2018

Contents

List of Figures

List of Tables

Nomenclature

$BRIEF$ Binary Robust Independent Elementary Features

CAD Computer Aided Design

$CUDA$ Compute Unified Device Architecture

DOF Degrees of Freedom

EKF Extended Kalman Filter

$FAST$ Features from Accelerated Segment Test

$GPIO$ General-Purpose Input/Output

GPS Global Positioning System

GPU Graphics Processing Unit

$I2C$ Two wire interface

IMU Inertial Measurement Unit

$LeGO - LOAM$ Lightweight and Ground-Optimized LiDAR Odometry and Mapping

$LiDAR$ Light Detection and Ranging

MCU Microcontroller unit

NDA Non Disclosure Agreement

$ORB - SLAM2$ Oriented FAST and Rotated BRIEF- SLAM2

PCB Printed Circuit board

PID Proportional, integral, derivative controller

PWM Pulse Width Modulation

$QR - Code$ Quick Response - Code

ROS Robot Operating System

$SLAM$ Simultaneous Localisation and Mapping

$SotA$ State of the Art

SPI Serial Peripheral Interface

VLP Velodyne PUCK

1 Preamble

It should be noted that the specific application for the Trosam Project is covered by a non-disclosure agreement (NDA) until 2021. As such any research done which focuses or revolves around application specific topics will not be presented in this analysis. A 'proper' State of the Art (SotA) for this project, involving researching current competitors' solutions, has been conducted. Their pros and cons have been analysed and will be taken into consideration when designing the proposed solution, however cannot be presented here.

For the purpose of protecting the NDA, this project will be simplified to a '*fully autonomous virtual path following robot*'. Due to the project being split between two courses, the SotA section (originally part of the Spring semester report) has been scaled down in order to be included in this report for completeness.

2 Structure

Throughout this section, the structures of the report, project, and team are outlined. The aim is to provide an understanding of the planning and engineering process.

2.1 Report Structure

This report is divided into sections, covering different aspects of the product development and the technical details of the robot. Beginning with a brief problem definition, requirements and delimitations are presented. Following this, a summary of the State of the Art analysis is presented to provide a foundation for the design choices.

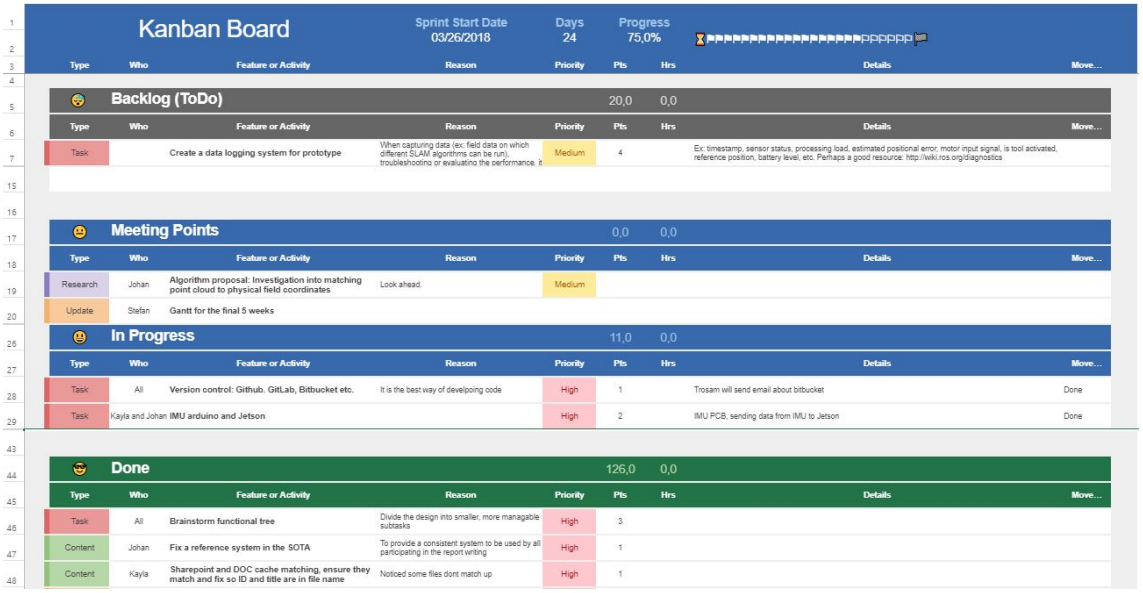
The overall system is presented adopting a top down approach beginning with an overview of the three general domains; Hardware, Control, and Software. Each domain is then expanded on with specifics outlined where detailed design information can be found.

The systems engineering process used is described in the verification and validation section with results presented afterword.

Finally, a discussion on the project accompanied by future developments is delivered.

2.2 Project Structure

Overall, the project was organised using a modified Kanban project structure with supporting Gantt Chart. The board has the following main headings: *Backlog, Meeting Points, In Progress/Testing,* and *Done*. An example of the board can be seen in Figure 1.



Kanban Board

Sprint Start Date 03/26/2018 | Days 24 | Progress 75,0%

Type	Who	Feature or Activity	Reason	Priority	Pts	Hrs	Details	Move...
Backlog (ToDo)					20,0	0,0		
Task		Create a data logging system for prototype	When capturing data (ex: field data on which different SLAM algorithms can be run), troubleshooting or evaluating the performance, it	Medium	4		Ex: timestamp, sensor status, processing load, estimated positional error, motor input signal, is tool activated, reference position, battery level, etc. Perhaps a good resource: http://wiki.ros.org/diagnostics	
Meeting Points					0,0	0,0		
Research	Johan	Algorithm proposal: Investigation into matching point cloud to physical field coordinates	Look ahead.	Medium				
Update	Stefan	Gantt for the final 5 weeks						
In Progress					11,0	0,0		
Task	All	Version control: Github. GitLab, Bitbucket etc.	It is the best way of developing code	High	1		Trosam will send email about bitbucket	Done
Task	Kayla and Johan	IMU arduino and Jetson		High	2		IMU PCB, sending data from IMU to Jetson	Done
Done					126,0	0,0		
Task	All	Brainstorm functional tree	Divide the design into smaller, more managable subtasks	High	3			
Content	Johan	Fix a reference system in the SOTA	To provide a consistent system to be used by all participating in the report writing	High	1			
Content	Kayla	Sharepoint and DOC cache matching, ensure they match and fix so ID and title are in file name	Noticed some files dont match up	High	1			

Figure 1: Modified Kanban Board Example

The project was spread over three quarters of the academic year which will be referred to as project sections. The first section was dedicated to research and the creation of the SotA. After the SotA, still in the first section, the focus shifted to high level solution design and presentation to the stakeholders. Also during this section, most of the required components were researched and ordered.

The goal of the second section of the project was to produce an initial hardware prototype, allowing for the last section of the project to be solely focused on further improvements and testing of the hardware. The main focus of the last section, however, ended up being further development of the overarching software/algorithm design, since this took more time than anticipated.

The project requirements were revisited in detail during the academic year quarter cross-over periods to ensure the project was still on track.

For the last two sections, a project Gantt Chart was developed as it became clear that the stakeholders and supervisors required a more visual confirmation of the project plan.

Every Monday morning, the team got together for a short meeting where pre-specified meeting points were brought up and tasks were updated based on the previous week's progress.

Every Thursday, two representatives from the team had a *stakeholder meeting* with Trosam where the current progress was presented. These meetings proved an ex-

cellent opportunity to gain valuable feedback from the stakeholders to ensure the project would ultimately align with their expectations and requirements.

Finally, weekly meetings with the team's project supervisor, Naveen Mohan, were undertaken, where the current progress of the project was presented and discussions on future work were had.

2.3 Team Structure

One of the requirements for the project was that all members contribute and work with all main hardware, software, and control aspects of the project. To accomplish this, two teams where formed. *Team Horrible* was mainly responsible for the **H**ardware and control structure of the robot. *Team Terrible* on the other hand was responsible for the **T**echnology, software, and algorithm structure of the robot.

Over the course of the project, each member has participated in each of the teams, working on large or small aspects of that teams requirements. The teams integrated so that all team members could focus on the project finalisation toward the end of the project, particularly once the hardware construction was nearing completion.

3 Background

This report was written for a capstone project for the *Engineering Design, Mechatronics track* courses *MF2058* and *MF2059*, which span over the Spring and Autumn semesters of 2018.

The university assignment is to produce a prototype of a mechatronic system for a stakeholder, which is either a research project or commercial company. In this specific case, the task was to create an autonomous robot that can accurately position itself and a tool within an outdoor environment. Due to the project being conducted for Trosam Automation AB, the details of the robot's specific purpose is covered by a non-disclosure agreement and is thus not explained. However, the solutions to the engineering problems which the team focused on can be disclosed.

Due to the large scope of the initial proposal, the team refined the scope of the project to creating a prototype that operates under ideal environmental conditions (for example flat terrain, well lit, non-adverse weather). However, the critical limitations of the initially proposed task were maintained as follows.

The main project limitation is the inability to utilise high-precision GPS for localisation. The stakeholders wish to maintain a competitive leverage by using an alternative self-contained technology. GPS can also lose accuracy under certain conditions in which the prototype may be used. The second limitation is the prototype also has to operate in an environment with sparse features available for localisation. The prototype must have a maximum weight of 50 kg and due to the nature of the application, the robot is only required to operate at speeds up to roughly 5 m/s.

3.1 Requirements

Over the course of the project the initial stakeholder requirements evolved as new information on the project came to light. These requirement changes were done in collaboration with Trosam and the following list shows their status. In Table 1 it can be seen which requirements were met.

The exhaustive stakeholder requirements have been expressed in Appendix B. Technical requirements have also been created however they will not be included due to issues with the NDA.

Table 1: Stakeholder requirement completion

Req.	**Done**	**Reasoning**
1	-	The full path was never tested.
2	✓	The prototype contains diagnostic LEDs and an Xbox 360 controller.
3	✓	The target mass of the prototype has not been exceeded.
4	✓	The prototype can operate in ideal conditions.
5	✓	Although a full lap was not tested, the speed the robot successfully operates at would allow it to satisfy this.
6	✓	Although not explicitly tested, the battery capacity allows for multiple runs of the task and it can all be remotely launched through the controller.
7	-	Obstacle avoidance has not been implemented.
8	x	This requirement failed due to the drivers and hardware construction limitations.
9	✓	The robot performs the same tasks repeatedly.
10	✓	The total cost for *Trosam Automation AB* was approximately 32 KSEK. For an in-depth look at the budget check Appendix A.
11	-	This was not investigated.
12	✓	The robot has successfully integrated the tool.
13	✓	Zero external modules where used.
14	✓	The robot successfully followed paths provided by the stakeholders.
15	✓	There where zero GPS modules used in the implementation.
16	✓	Unfortunately evidence of this cannot be provided however this requirement has been met.

3.2 Delimitations

As aforementioned, the team has refined the initial project proposal to its critical elements. Key aspects that the team does not focus on in design considerations are:

- Operation in non-ideal weather conditions (e.g low-visibility, rain).
- Operation on variable terrain.
- Independently controlling the tool motion/position to the robot's base frame.

In the following section sensors, algorithms, control theory, and design decisions are explored and outlined.

4 State of the Art

The following State of the Art analysis (SotA) is a condensed summary of the findings from the first third of the project. See the Spring semester report for the full SotA.

4.1 Sensor types

Different types of sensors were investigated in order to determine their suitability for localisation within the given context.

LiDAR

Light detection and ranging (LiDAR) is used for remote mapping of an area by reflecting directed laser beams on obstructions, and using the time-of-flight for distance calculation [3]. By sweeping and pulsing the laser beam in different directions, a point cloud of distances can be assembled for the surrounding area. There are many strengths of LiDAR usage such as many having a 360 degree view, 3D outputs, and long distance measurements with high precision and accuracy.

IMU

An Inertial Measurement Unit (IMU) is a component that is often used as an input for navigation systems, often based on multiple gyroscopes, accelerations, and magnetometers. The sensors are mounted in angles perpendicular to each other so that movement can be measured along the X-,Y-, and Z-axes [4]. Magnetometers can be used to get a global sense of direction and thus increasing the degrees of freedom that the IMU can measure.

Cameras

Digital cameras can be used for determining location changes, mainly through the use of visual odometry [5]. This is a method based on calibrating a camera for optical distortion, extracting image features or the optical flow between two subsequent images [6]. The simplest approach is through a monocular setup. However, a stereo vision setup yields improved results since pixel depth can be calculated assuming the captured images are taken far enough apart. The distance between the two sensors is called the 'baseline'. The resulting data can be used to build a point cloud of

an area similar to that generated through LiDAR mapping [5]. Another benefit of using stereo vision setups is that issues of scale uncertainty can be resolved through the known distance between the cameras [6].

4.2 Algorithms

Thus far, the discussion has been focused around the specific sensors to be used to obtain data for the localisation of the robot. First data is collected from a multitude of different sensors, each with their own strengths and weaknesses.

Localisation and Mapping

There are multiple strategies for localisation and mapping in GPS-denied environments of which the Simultaneous localisation and mapping (SLAM) algorithm family is the most promising [7]. SLAM is a method of localising a mobile robot in an unknown environment while simultaneously constructing a map. The advantage of the method is that the generation and updating of the map while concurrently localising the robot, which makes it the most appropriate alternative in the the field of autonomous mobile robots [8]. SLAM is not a single algorithm, but a technique which can be applied through different algorithms, e.g. EKF-SLAM, ORB-SLAM and FastSLAM [9].

The de-facto process for utilising SLAM is described in Figure 2. The data gathered from sensors is passed into the SLAM algorithm and the output of the algorithm (the pose of the robot or object) is passed into the motion planning section.

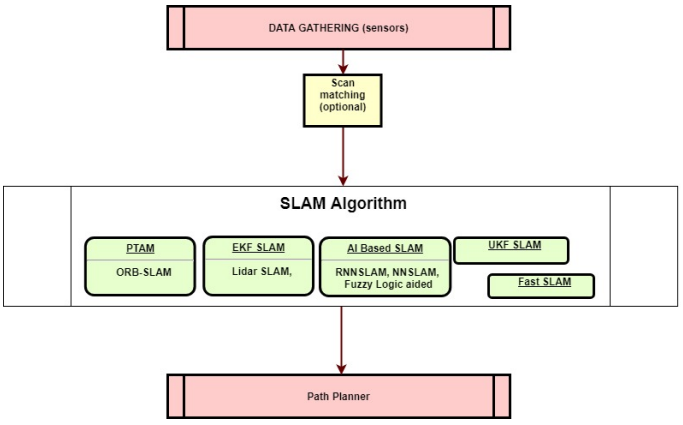


Figure 2: SLAM Process. Created by Team Trosam

SLAM algorithms

As previously mentioned, there are multiple different algorithms to solve the SLAM problem, each with their benefits and limitations. However, the following algorithms are the ones the report focuses on.

ORB-SLAM2 In the article [10], the open-source algorithm *ORB-SLAM2* is presented. It is the second and improved version built on [11]. ORB-SLAM2 features the use of monocular, stereo or RGB-D cameras. Differing to most other algorithms which use EKF-based mapping for Monocular SLAM, ORB-SLAM2 uses Bundle adjustment (BA) mapping. The computational cost is very high for computing camera pose in an unknown scene and BA is more efficient than EKF filtering in features per computational cost [12]. ORB-SLAM2 adopts the method of Parallel tracking and mapping (PTMAP) [13]. PTMAP splits the tracking and mapping in different tasks. This allows for computationally expensive batch optimisation techniques (such as BA) which normally are not associated with real-time operation. This results in a more detailed map and according to [14] and [12], this gives an advantage to the BA-based approach since it can handle more points and therefore more features can be extracted.

LeGO-LOAM LeGO-LOAM [15] is based on LOAM, but also features loop closure which will help the robot to avoid drift when returning to previously visited points. It has been developed with ground based robots in mind. The algorithm also supports IMU data which was verified by running a so called ROS data bag which plays back sensor data for later analysis. This algorithm is not described in the full SoTA because it was published after the SoTA research was completed.

Motion planning

Path planning and trajectory planning are crucial issues in the field of Robotics and, more generally, in the field of Automation [16]. Path planning algorithms generate geometric set-points in the robot's workspace, representing locations in the physical space, and defines in which order to visit them. To provide the system designer with more control over the dynamic behaviour, trajectory planning algorithms can be used. The goal is to produce, based on the geometric path, the necessary time information, such as the velocity and acceleration needed at every instance in time. This allows for optimising performance to create a smooth trajectory, and taking hardware limitations into consideration, avoiding excessive acceleration, jerk, or vibrations [16]. For the purpose of this project, the coordinates of each set-point

are already known, only the optimal order in which to visit them remains to be calculated. The trajectory planner explored in this project is written by the team.

4.3 Computation Boards

A variety of different computation boards have been investigated, however, three main computers/micro-controllers were implemented. The *Raspberry Pi 3B+* has 4 cores with 1.4GHz, a 64-bit ARM, 4 USB ports and 40 GPIO pins. The Jetson TX2 has 6 2GHz cores, a GPU, 1 USB, Ethernet, WiFi, Bluetooth, 32GB eMMC, and SATA/SD expandable storage. Finally, the Arduino ATMega2560 is used to interface with the IMU.

4.4 Software

Once the necessary hardware and algorithms were decided, a suitable software environment and external libraries had to be chosen.

Robot Operating System Middleware

In order to combine multiple programming and simulation environments, the Robot Operating System (ROS [17]) was chosen due to the simple integration with existing code packages, drivers, and more. Despite the name, ROS is mainly middleware used for inter-process communication, such as publishing messages and requesting a response from other modular processes. ROS can be deployed on multiple machines from which the nodes can communicate with one another. At the time of writing, ROS *Kinetic Kame* is the current long term support (LTS) version of the software which will be supported until April, 2021.

ROS programs depend on a system of *nodes*, *topics*, *services*, *messages*, *bags*, and data stored in the *parameter server*. This complex system is managed by the ROS Master. A node is a process that performs computation (written with the use of a ROS client library). They communicate with other nodes using topics (named buses over which nodes exchange messages), services and the parameter server. Messages are simple data structures with typed fields published on topics. They can take the form of standardised message definitions for common robotics applications, such as vectors, geometric transforms, odometry, and maps.

Raspberry Pi 3B+ OS

The Raspberry Pi is running the minimal Linux distribution Raspbian Stretch Lite. The main purpose for the Raspberry Pi is to handle the top-level position control and low-level control, e.g. motor actuation and LED control.

Nvidia Jetson TX2 OS

In order to support ROS, it is recommended to run the Linux distribution Ubuntu 16.04 on the Nvidia Jetson TX2. The Jetson is expected to handle the SLAM computation based on inputs from the LiDAR, stereoscopic camera, and IMU sensors.

4.5 Robot Design

Hardware placement

Early in the design process it was decided that the robot should have front-wheel differential drive for steering as well as moving and two swivel-wheels in the rear for support. The reason why front-wheel drive was chosen instead of rear-wheel drive was because rear-wheel drive would result in a pendulum motion in the front when turning, which is very hard to control and a behaviour that is undesired for the intended purpose of the robot. The conceptual design of the robot went through several iterations due to some requests and proposals from the stakeholders as well as design changes made from the project team. Figure 3 depicts the CAD during the early design stage and Figure 4 depicts the CAD design of the autonomous robot after several iterations.

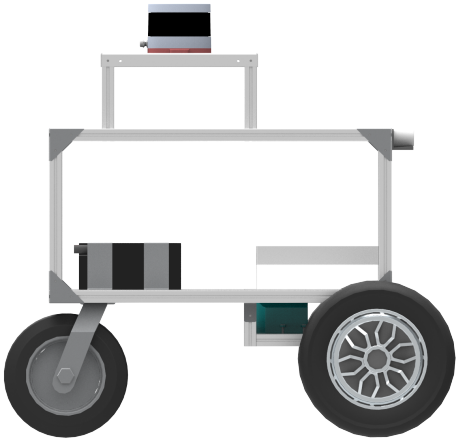


Figure 3: Early draft of the CAD design

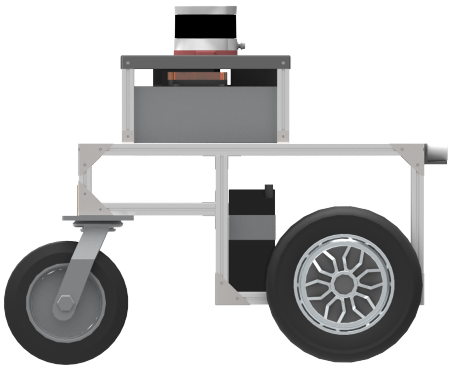


Figure 4: Late draft of a more compact CAD design

As can be seen from Figure 3 and Figure 4, changes to the frame were made in order to minimise the size of the robot and to raise the swivel wheels from the ground rather than extending the hub motors further down. The swivel wheels also had to be separated slightly to allow for reversing, where the swivel wheels may both turn inwards.

The placement for the rest of the hardware components were made so that the centre of mass would be as close to the centre line of the robot as possible in order to have an even weight distributions on the hub motors. It is possible to see the location of the centre of mass in Figure 5 and a section view of the robot in Figure 6.

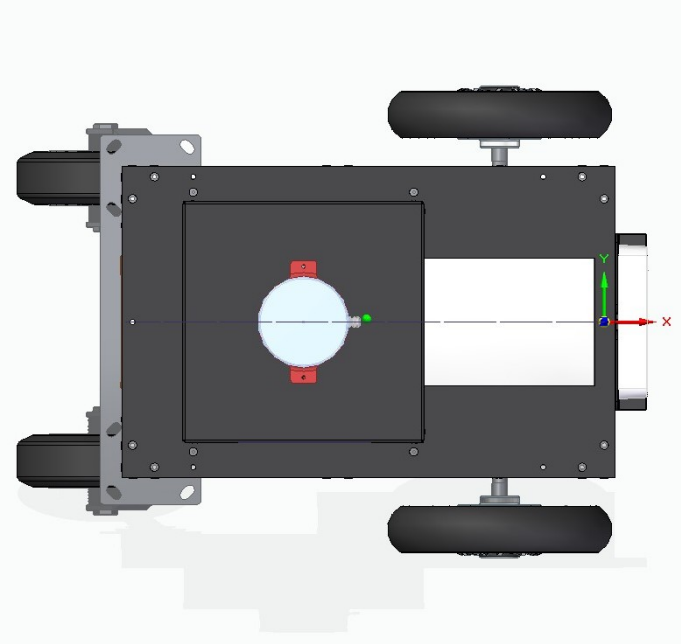


Figure 5: Top view displaying the centre line and centre of mass

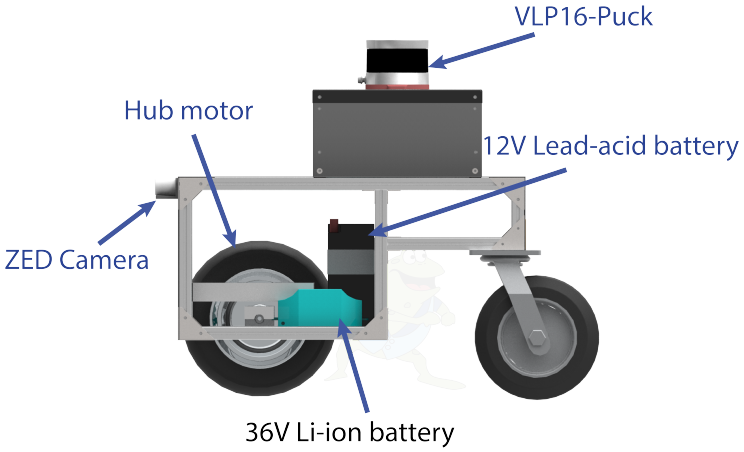


Figure 6: Section view from the left side of the robot with labelled components

5 System architecture

The project has lots of separate parts that are required to work together. In this section the final architecture of the system is described. The system architecture description will be split into 3 sections; hardware, control, and software.

5.1 Hardware Architecture

The components were mounted on the rig depending on their robustness against damage from the environment, such as mud splashes from the ground. The motors, mission critical tool components, and regulators are quite robust and form the base layer of the system. Figure 7 depicts the final design solution.

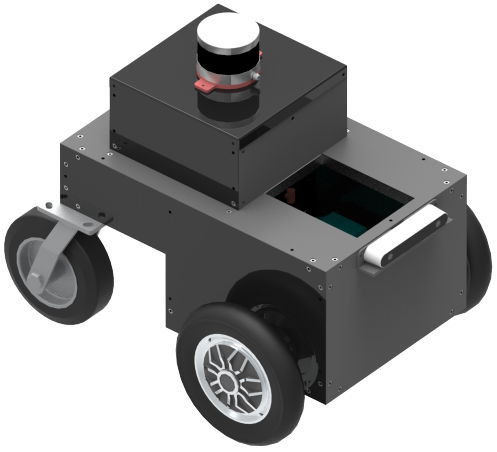


Figure 7: CAD representation of the robot

5.2 Software Architecture

As mentioned, the NVIDIA *Jetson TX2*, Raspberry Pi, and LiDAR are connected via Ethernet, employing the ROS architecture. ROS operates with a single master core which is situated on the Jetson. The Raspberry Pi connects to the Jetson's master, allowing for nodes to be able to communicate with each other freely. In essence, ROS nodes subscribe to topics containing information, modify it, and publish to a new topic. This allows sensor data to enter, be processed in order to extract location information, feed the controller the required movement, and consequently feed the

motors with the desired voltages. ROS is not pure real-time, so it is essential to use time stamps for some data structures. Note, the system has been set up to allow for a '*hot swapping*' of different SLAM algorithms allowing for new algorithms to be inserted without changes to the remainder of the software.

The overall software architecture can be seen in Figure 8.

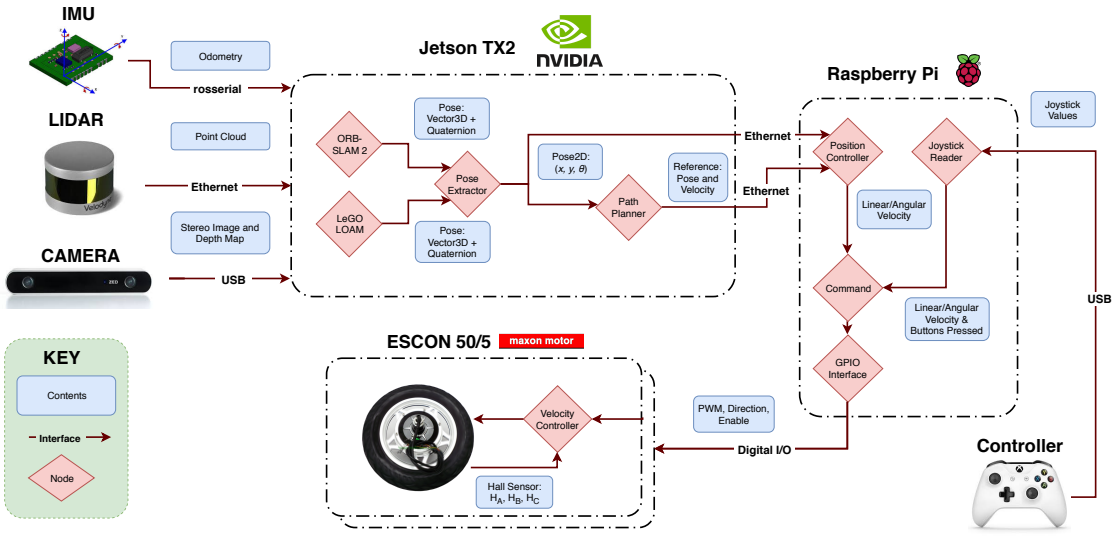


Figure 8: Software Architecture

As can be seen, the nodes are distributed on the hardware, in order to maximise the performance, utilise the hardware capabilities, and create a modular architecture. The project also made use of an external computer to allow for critical information to be monitored during tests.

Following is a brief description of each of the nodes:

Nvidia Jetson TX2 - Ubuntu 16.04 LTS

LiDAR Node Subscribes to and publishes LiDAR images

ZED Node Subscribes to and publishes required stereoscopic camera frames

Serial Server Recieves IMU data over serial port and publishes it as a ROS-message

SLAM Node Subscribes to sensor inputs and publishes position transform

Pose Extractor Publishes a 2D pose (x, y, θ) from the subscribed SLAM algorithm

Path Planner Generates and plans the reference path for the robot, publishing both the desired 2D pose and reference velocity

Raspberry Pi 3B+ running Raspbian Stretch

Position Controller Generates linear and angular velocity information given the estimated and reference position of the robot

Joystick Reader Subscribes to the Xbox 360 controller and publishes Linear/Angular Velocity as well as buttons pressed to control (start, stop, switch) other nodes

Command Node Publishes linear/angular velocity from either the position controller or Joystick Reader

GPIO Interface Provides control over the PWM and digital I/O pins for control over the hardware; such as motor drivers, too components etc

Arduino Mega 2560 running a ROS-Serial client

IMU Reader Reads IMU data, wraps it in a ROS-primitive data type and transmits it to the ROS-Serial server over the asynchronous serial protocol

External computer running Ubuntu 16.04 LTS

System interface System initialisation, mission control, and parameter setup

Rviz Visualisation of real-time sensor data, SLAM performance, and map location

Logs Display and saving of log data

5.2.1 Logs and testing environment

When working with ROS, all datastreams (topics) can be recorded in a *rosbag* for later use, such as troubleshooting or benchmarking, and makes it very easy to create an all encompassing log of what has happened. The bags can be replayed at a later time to reproduce the behaviour of the system. This allows viewing of the sequence of events multiple times. To work with the data offline the bags can be converted to e.g. MATLAB-readable format with 3$^{\text{rd}}$-party software libraries. The results are described further in Section 9.

6 Hardware Specifics

6.1 Aluminium frame

The frame of the robot was built using 2x2 cm aluminium extrusions. These were picked as building material because they are cheap, lightweight, and offer a lot of flexibility when it comes to assembly.

Length[mm]	**pcs**
555	2
310	9
295	2
240	4
220	2
140	4
76,5	2

Table 2: Table of dimensions for the aluminium extrusions

The extrusions were cut to size according to Table 2 by using a water jet.

Figure 9: Extrusions cut to size using a water jet with help from KTH Prototype centre

The baseplate, coverplates, and all the fasteners for the aluminium frame were also cut out using the water jet.

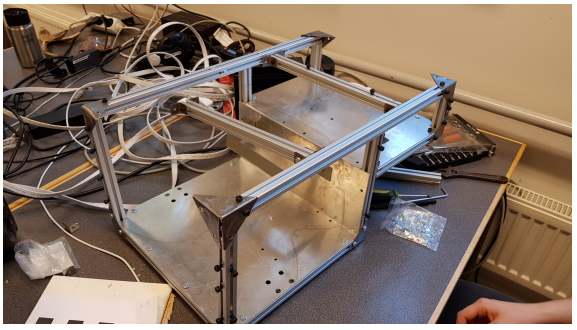


Figure 10: Aluminium frame mid construction on the 8th of October

6.2 Motors

Choosing Motors
An experiment was conducted to find out how large of a force that is needed to move a four-wheeled system on different terrains. The experiment consisted of one person sitting on a cart and being dragged with an attached Newton meter.

The experiment revealed that to move around a 100 kg payload on four wheels on grass required about 100 N on flat terrain and up to 300 N in a slight incline. These results provided an estimate of the required motor torque when calculated with an approximate wheel radius.

Getting Motors
Since the decision on what motors were needed came at such a late stage in the process there was no time to order new motors. Instead two 10" hub motors were taken from an electric hoverboard, as pictured in Figure 11. As this is a consumer product, no data sheets were available and no part numbers could be found on the hardware itself.

Figure 11: Denver DBO-10001WHITE [1]

6.3 Hall Sensor Decoding

During early testing, to find the velocity of the motors the phases from the hall sensor were read with external interrupts on an *Arduino Mega 2560*. The calculations were heavily dependent on Nich Fugal's calculations from "Efficiently Reading "[18].

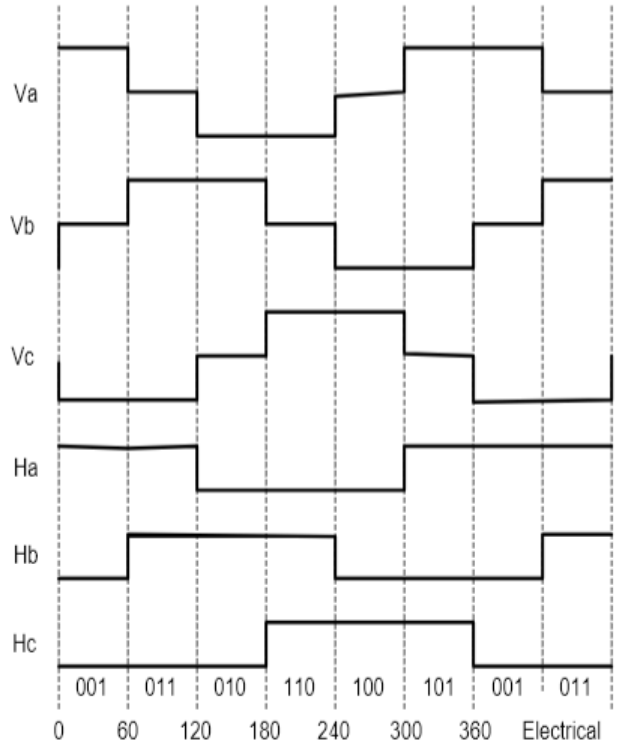


Figure 12: Six Steps Commutation Sequence [2]

First the six steps in the commutation sequence were found and used to put up a lookup-table, as seen in Table 3. When the wheels are going forward the direction is +1 and -1 when they are moving backwards.

With the lookup table it can be detected whether the wheel is spinning forward or backward. The motor has 15 electrical pole-pairs. The interrupts are triggered for each edge, rising or falling, and there are three phases. This means that 90 interrupts are triggered for each revolution of the wheel. Lastly, by summing the numbers produced by each interrupt periodically gives an estimate of the angular velocity. There is a trade-off here between sensor resolution and sample rate - increasing the sample frequency also increases the sensor quantisation, and can lead to oversampling.

6.4 Motor Drivers

Whilst it was initially planned on using the Electro-Mobil Styrdon BLDC-1KW Drivers [19], they became inadequate for the required task as they are not four

Direction	OLD C B A	NEW C B A	Binary	Decimal
+1	0 0 1	0 1 1	001011	11
+1	0 1 1	0 1 0	011010	26
+1	0 1 0	1 1 0	010110	22
+1	1 1 0	1 0 0	110100	52
+1	1 0 0	1 0 1	110100	37
+1	1 0 1	0 0 1	110100	41
-1	0 0 1	1 0 1	110100	13
-1	1 0 1	1 0 0	110100	44
-1	1 0 0	1 1 0	110100	38
-1	1 1 0	0 1 0	110100	50
-1	0 1 0	0 1 1	110100	19
-1	0 1 1	0 0 1	110100	25

Table 3: Lookup table for wheel direction

quadrant controllers. Four quadrant allows for counter/clock wise acceleration and deceleration, however, the Styrdon's were only two quadrant thus braking was not possible.

After consultation with Stork Drives AB, it was decided to purchase two Escon 50/5 [20]. Whilst these had inbuilt speed control (detailed in Section 7.5) they proved to have too low torque, causing problems with controllability and driving on certain surfaces.

6.5 Wheel fasteners

The wheels with integrated brushless DC hub motor needed to be securely fastened to the chassis. Since the motor fasteners could not be removed from the hover board and because no fasteners for the specific motors were found, the fasteners were manufactured manually by using the milling machines and lathes at the department of production engineering at KTH. The motor fasteners were manufactured based on a drawing made in the CAD software *Solid Edge ST10* where the design was based on the wheel axle. The mechanical drawing of the motor fasteners is available in Appendix D.

6.6 Main Computer and Micro-controllers

The chassis was designed with sensor placement in mind, in order to protect the sensitive components from the environment. The computational boards are placed close to the centre on a platform where they are protected from the elements. The IMU was also placed in this enclosure. The optical sensors, the LiDAR and stereoscopic camera, are placed up top and at the front respectively, where they have a good viewing angle of the surrounding environment.

The *Raspberry Pi 3B+* and NVIDIA *Jetson TX2 Development Kit* that were chosen for their computational capabilities and ease of use. In order to network them, a wireless network router was used to link the hardware together using Ethernet cables, for low latency and robustness. A laptop can be connected to the network from a distance for visualisation of the on-going process and to initiate or stop the command scripts.

The Stereolabs ZED stereoscopic camera was chosen for its built-in ROS support, wide camera spacing, and high performance over a large variety of distances. It can perform localisation based on small visual features, such as patterns on otherwise flat sections of ground, as well as larger objects. It was mounted to a 3D-printed bracket which keeps it parallel to the ground plane and is connected via USB to the NVIDIA Jetson TX2.

The Velodyne VLP-16 Puck is an entry level 3D-LiDAR from one of the leading manufacturers. The greatest difference to more expensive alternatives is the number of layers it can detect, while the accuracy and vertical field of view is still the same. Velodyne also provides a ROS wrapper for the sensor data, so that it can easily be read by other ROS nodes. Note how the LiDAR has been placed so that no part of the robot is seen by the sensor, removing the need for any masking. The Puck is connected directly to the router via Ethernet while the TX2 connects to it via the router and publishes the sensor data over ROS.

The BNO080-based HillCrest FSM300 IMU, that was chosen by the stakeholders for being a high-quality, versatile, and factory calibrated sensor, was attached to a custom circuit board. The circuitboard is in turn connected to an Arduino Mega 2560 microcontroller. The microcontroller, acting as a ROS-Serial client over USB, transmits the IMU data to the rest of the network through a ROS-serial server running on the Jetson. The IMU communicates over the I2C protocol, where specific "sensor reports" can be requested at a rate of the user's choice, as described in the Hillcrest SH-2 Manual [21]. Quaternion pose data, linear acceleration (including gravity), and rotational velocities were requested in order to populate a standardised sensor `sensor_msgs::Imu` ROS message once all three message types had been

received. No co-variance data was supplied, as the supplier had not specified this. Using ROS-Serial created a bottleneck in the rate at which sensor data could be delivered to the ROS network. By increasing the baud rate of the Serial connection, an average rate of approximately 50 Hz was achieved, compared to the IMU capability of 200 Hz.

Obtaining information from the IMU was a difficult task for the team and multiple methods and microcontrollers were used. The team initially tried an STM32 microcontroller with the Hillcrest supplied code however, integrating ROS-serial into that code was not achieved. Thus, an *Arduino Mega 2560* was used along with the command protocols to control the data output from the BNO080 from the SH2 manual provided on Hillcrests website for the FSM300 product. The *Arduino Mega 2560* is needed due to the large memory usage.

6.7 Power Supply

The various hardware components have varying electrical supply requirements. It is therefore of importance to design a power supply that can output the desired voltage to the respective electrical components. Figure 13 shows the power supply box that was designed. Notice that there are also LEDs indicating whether a switch is on. These LEDs will be explained further in Section 6.8.

Figure 13: Power supply switch box

To supply all of the components with the correct voltage, a voltage regulator PCB was designed using the software Autodesk Eagle. The design of the board is first made as a schematic where all of the electrical components are connected as desired.

Based on this schematic, the PCB board is designed and milled. The schematic of the power supply circuit can be seen in Appendix C.

The battery that powered the logic circuit of the autonomous rover provided a 12 V voltage output. Since the *Arduino Mega 2560* and the Raspberry Pi had a supply voltage of 5 V and because the other electrical components excluding the motors, e.g. NVIDIA *Jetson TX2* and LiDAR, had a supply voltage of 12 V, the required supply voltages are 12 V and 5 V. The 5 V supply voltage is obtained by a USB-hub.

Switches were added for each electrical component to facilitate the testing process of the robot. This enabled the possibility to turn off the electrical components that were not desired when the robot was under development. The circuit drawing of the voltage regulator board is found in Appendix C. Further, to provide protection for the power supply PCB against humidity and foreign objects, a 3D-printed case was designed and manufactured.

6.8 Hardware Diagnostics

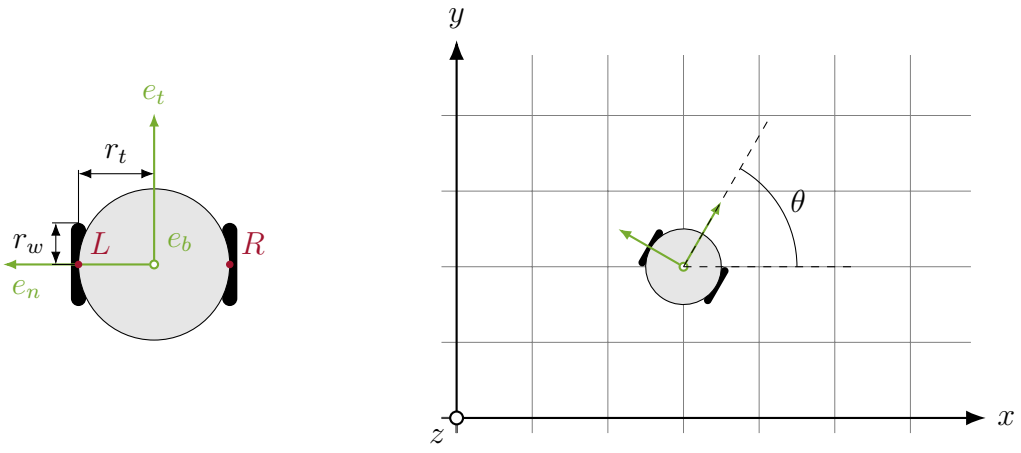
In order to ensure that the tool components is actually powered on when the power switch is in its ON state, a LED is turned on simultaneously. There is also one LED to indicate whether the LiDAR is on and another LED to indicate whether the robot's logic is on. Refer to Figure 13 for the final design of the switch box, including the three LEDs.

7 Control Specifics

This section is meant to elaborate on the underlying theory that governs the motion of the robot. This includes the top-level trajectory tracker, and the low-level GPIO interface. To avoid modelling the finished robot's inertia, dynamic friction etc, a kinematic approach with a non-holonomic constraint is used. A top-level trajectory tracker assumes direct control over the linear and angular velocity, and low-level servos track the individual reference wheel velocities to produce the desired motion.

7.1 Kinematic Model

The robot is modelled as a mobile two-wheeler with it's own 3D reference frame, given by unit vectors (e_t, e_n, e_b) as seen in Figure 14. The *tangential* unit vector e_t always points in the direction that the robot travels, and it cannot move along the *normal* direction e_n since it cannot translate perpendicular to the wheel direction. The motion is planar, so $z \equiv 0$ and the *binormal* unit vector e_b is fixed parallel to the z-axes.



(a) Lengths, characteristic points, and coordinate frame orientation

(b) Example of planar motion

Figure 14: Kinematic 2D model of the vehicle

Let $q = \begin{bmatrix} x & y & \theta \end{bmatrix}^T$ be the generalised state vector describing the position and orientation of the robot's reference frame, relative to a world frame. The kinematic

equations of the robot are

$$\begin{aligned} \dot{x} &= v\,\cos\theta \\ \dot{y} &= v\,\sin\theta\,, \\ \dot{\theta} &= \omega \end{aligned} \tag{1}$$

where v is the linear velocity in the direction given by θ, and ω is the rate of change of this angle. This can be written as the Linear Time-Variant system

$$\dot{q} = S(q)\,u, \tag{2}$$

with the velocity vector $u = \begin{bmatrix} v & \omega \end{bmatrix}^T$, and

$$S(q) = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix}. \tag{3}$$

7.2 Kinematic Control Law

Introducing the reference state $r = \begin{bmatrix} x_r & y_r & \theta_r \end{bmatrix}$, and it's derivative $\dot{r}$, the error state is simply $e = r - q = \begin{bmatrix} x_e & y_e & \theta_e \end{bmatrix}$.

Using the kinematic model (2), under the assumption that the velocity can be controlled directly, a stabilising control law for trajectory tracking is then simply

$$u = S(q)^{-1}\,(\dot{r} + K_p\,e)\,, \tag{4}$$

which can be shown using a Lyapunov energy candidate to be globally stable, and the error tend to zero as long as the reference does not violate the non-holonomic constraint. A block diagram of the process can be seen in Figure 15.

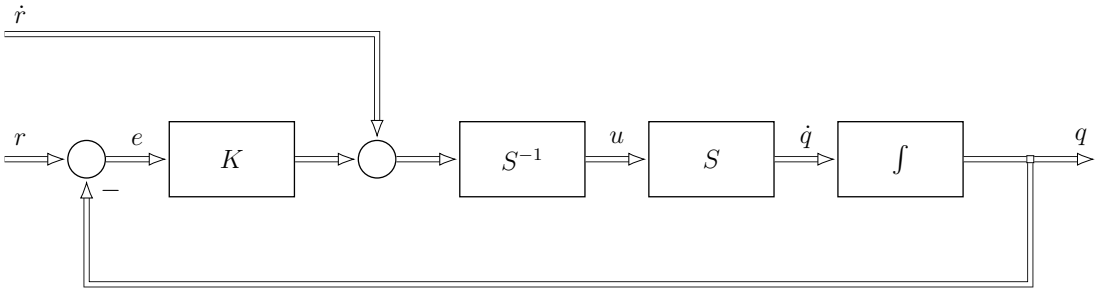


Figure 15: Block diagram of the velocity controller

7.3 Software-in-the-Loop

A standalone graphical interface written in Qt, based on the package `turtlesim` can be used to evaluate the trajectory tracker's performance. It is written to subscribe to the output of the trajectory tracker, and to publish the 2D-pose of the robot, utilising the same topic as the pose extractor would. This allows for easy Software-in-the-Loop testing, since same controller code is used for both simulation, and in the field. An example of how this looks can be seen in Figure 16.

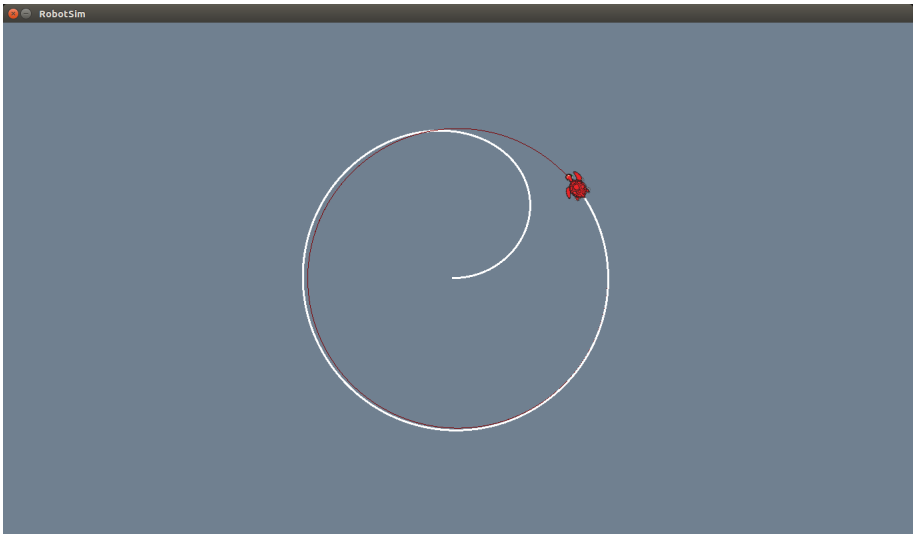


Figure 16: Simulation platform for Software-in-the-Loop testing of the controller

7.4 Kinematic Model, Extended

The top-level controller produces a control signal in the form of a desired angular and linear velocity. Next, the kinematic model is extended to capture the differential drive characteristics of the robot. Two additional points need to be considered, fixed in the robot reference frame, sitting on the e_n-axis, at a distance r_t from each side of the origin. These are the points directly above where the wheels make contact with the ground, and can be denoted R and L.

The kinematics tells us that the velocity vector of any arbitrary point A on the robot in the global frame is given by

$$v_A = v \cdot e_t + (\omega \cdot e_b) \times r_A. \tag{5}$$

It is assumed that there is no slip between the surface and the wheels, so that the velocity of the two points on the robot equals the peripheral velocity of the corresponding wheels. Finally, letting the wheel radius be r_w, the angular velocity of each wheel is given by

$$\begin{aligned} \omega_R &= \frac{1}{r_w}\left(v + \omega\, r_t\right) \\ \omega_L &= \frac{1}{r_w}\left(v - \omega\, r_t\right), \end{aligned} \tag{6}$$

which is the conversion necessary to transform the velocity vector into the individual velocities of the wheels.

7.5 Low-level Motor Controller

The top-level controller assumes full control over the wheel velocities. Even though this is not feasible in reality, if the wheel dynamics are sufficiently fast in comparison to the overall vehicle dynamics, this approximation can still be made.

Each motor is controlled by its own Escon 50/5 driver from Maxon motor. The driver has closed-loop velocity control functionality which uses the hall sensor feedback to estimate velocity and synchronise commutation.

The drivers come with a graphical user-interface that allows the user to configure the mode of operation, automatically tune controller parameters, and set limitations on e.g. current and velocity.

Both drivers in this project are configured to use closed-loop velocity control, with automatically tuned parameters. The reference velocity is read on digital input pin 1 as a 5 kHz PWM signal. The numerical value is determined by the duty cycle, and is calculated by linearly interpolating within the range of 10 % to 90 %, which represent a reference of 0 and 150 rpm respectively. The PWM duty cycle calculation for each wheel is based on (6), and is given by

$$\mathrm{PWM}_{R,L} = 10\,\% + |\omega_{R,L}| \cdot \frac{60\,\mathrm{rpm}}{2\pi\,\mathrm{rad/s}} \cdot \frac{80\,\%}{150\,\mathrm{rpm}}. \tag{7}$$

The direction of rotation is determined by the state of the digital pin 2, and pin 3 is used to enable/disable the driver.

The digital inputs of the drivers are connected to the GPIO pins on the Raspberry Pi, and a ROS-node provides the interface between the digital pins and the ROS

communications. The node is dependent on a library called `pigpio`, which provides a layer of abstraction that makes it possible to call functions such as `gpio_write(...)` and `hardware_PWM(...)` directly in the C++ code.

8 Software Specifics

This section provides an in-depth look at the algorithms and nodes used for localisation and navigation. It details the extraction of the estimated position by following the information flow from the sensors through the SLAM algorithms including transforms required. It also outlines the generation of the reference signal required in order to successfully control the robot.

8.1 Simultaneous Localisation and Mapping

Once the hardware platform was set up according to the previously mentioned system architecture, the actual localisation could be tested in practice. This included evaluating multiple SLAM algorithms and investigating how they could be adapted for this specific usage scenario. The task requires that the position is not only estimated in terms of starting position, but also with regards to certain objects within the operational environment, so that the tool is activated at the locations.

8.1.1 Evaluation of SLAM algorithms

While research papers can show promising results, implementing the techniques can often be hard if the software is not maintained or may require significant changes in order to be compatible with other hardware or software environments. Due to this reason, the plan was required to change and instead the localisation performance of two separate SLAM algorithms LeGO-LOAM and ORB-SLAM2 were compared.

8.1.2 LOAM

LOAM (LiDAR Odometry and Mapping in Real-time) is a LiDAR- and IMU-based method that places very well in the KITTI dataset benchmark, but does not offer loop closure. The code has been pulled from being open source, but has previously been part of an official ROS package. Forks of the old code exist, but integrating the IMU was difficult. The author of the forked code had not tested using an IMU either. Luckily, another even more promising alternative, LeGO-LOAM, was found.

The LeGO-LOAM paper was published during the summer of 2018 and was thus not available during the SotA research during the spring semester.

8.1.3 LeGO-LOAM

A visual representation of the point cloud generated by the algorithm and its estimated path for a circular reference is shown in Figure 17. The greatest difficulty with the implementation of LeGO-LOAM was the integration of the physical IMU. The problem was isolated to be aligning the IMU data to the correct coordinate system considering the LiDAR, the IMU and LeGO-LOAM all had different frame orientations. The authors of LeGO-LOAM note that the frame of the IMU must match the frame of the LiDAR so the team investigated the sensor data included in the data bags to correctly match the data to the correct axes. Unfortunately the attempts to correct the IMU frame alignment were unsuccessful and due to time constraints the team decided to postpone working on IMU integration. However, the IMUs linear acceleration output is used in determining the correct orientation for the global position frame so that it is perpendicular to the gravity vector. This global frame initialisation is also used in the ORB-SLAM2 algorithm and is discussed in Subsection 8.2.

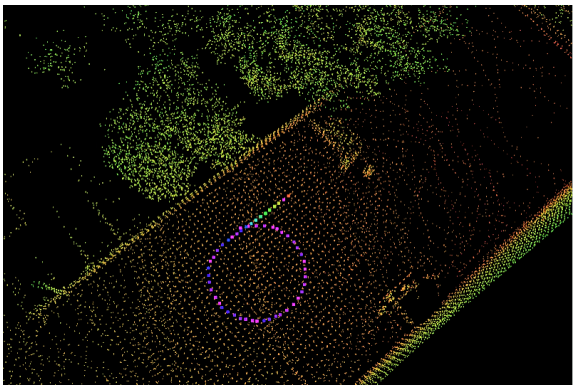


Figure 17: LeGO-LOAM visualisation

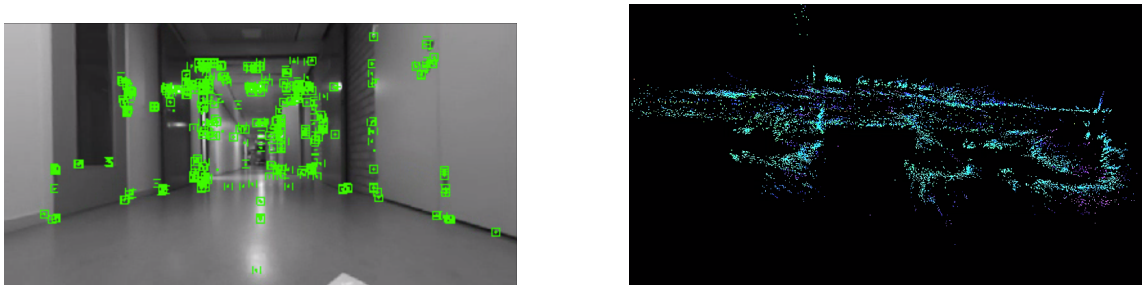
8.1.4 ORB-SLAM2

Oriented FAST and rotated BRIEF - SLAM2 [10] is a real-time SLAM system which can make use of Monocular, Stereo, or RGB-D cameras. The system features loop closure and re-localisation in real time. This way of solving the SLAM problem was chosen because of the performance presented in the article in urban environments

and on the KITTI dataset. Taking in data from only the ZED camera, the system is capable of mapping and localising within the area without any other sensors required. Calibration of the camera to perform at 30 frames per second and VGA resolution provided a smoother performance of the system which were satisfactory for the application.

ORB-SLAM2 can run in two different modes. SLAM mode and localisation mode. In SLAM mode, three threads are run in parallel: Tracking, localising, and mapping. While in localisation mode the local mapping and loop closing is deactivated and only the localisation is active. The localisation mode reduces computational cost since the map is no longer updated. Being able to save a map and only use the localisation mode is preferable during run time because of the reduced cost and thereby the energy consumption. The original version of ORB-SLAM2 was created without the feature to save the map. A fork of the original system developed by the community was modified and makes it possible to both save, and retrieve a map, which in turn enable the localisation mode to be activated from the start of run time after an initial thorough and time consuming exploration of the area to create a detailed map. Further, the code was altered to allow for publishing the live camera feed of ORB-SLAM2, which has ORB features superimposed on the ZED Camera live feed which can be seen in Figure 18 (a) and an example point cloud generated by the algorithm in an indoor environment (b).

Finally, the ability for detecting when the map is lost, was added so that in the event that it happens, the system can pause and disable drivers to ensure a more controlled system.



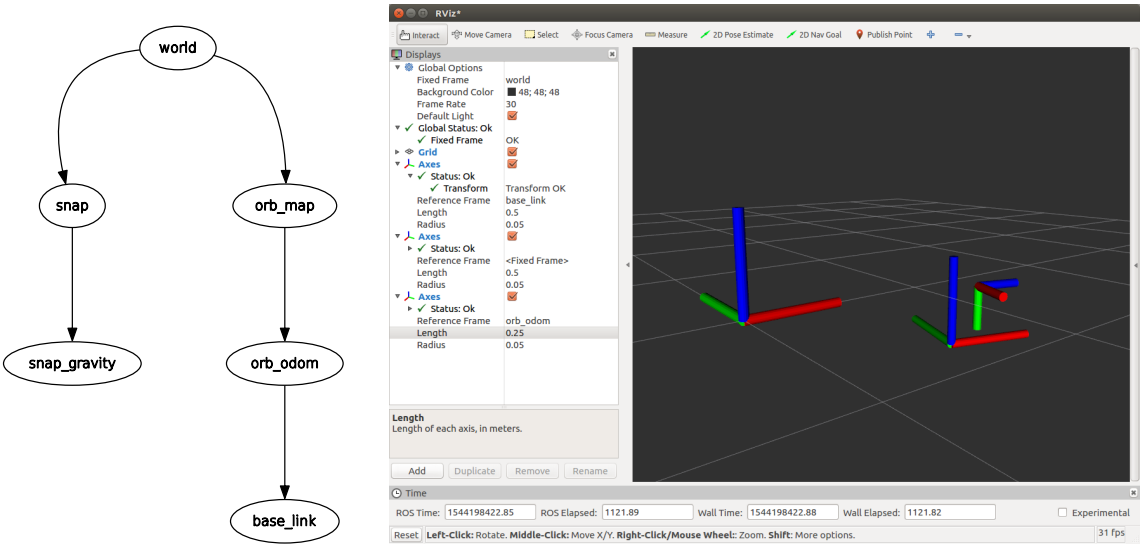
(a) Feature extraction (b) Generated point cloud

Figure 18: ORB-SLAM2 visualisation

8.2 Transforming to Planar Motion

The SLAM algorithms produce a 6-DOF pose, expressed relative to the pose the robot had when the algorithm was initialised. A series of mathematical transforms need to be performed to retrieve a 2D-pose (x, y, θ) that can be fed into the top-level position controller. To avoid doing the underlying mathematical calculations, a ROS-library called `tf2` is used. This requires the user to set up a tree of frames, and to determine the geometric relationship between them, otherwise known as a transform. A transform consists of one rotation and one translation along the newly rotated axes.

Frames of Reference For the sake of the next discussion, ORB-SLAM 2 will be used as an example. Figure 19 shows a simplified rendering of the transform tree.



(a) Transform tree (b) Visualisation in `rviz`

Figure 19: Graphical representation of reference frames

When the algorithm is initialised, the coordinate frames `orb_map` and `orb_odom` are initialised with it. These frames do not necessarily contain any information, but what is important is the *transform* between them. As the estimated pose of the camera changes, this is reflected by the transform between the frames being updated accordingly.

Since the `orb_odom`-frame expresses the position of the ZED camera, a *static transform* is needed to introduce the `base_link`-frame. This has its origin at the centre-

point of turning of the robot, and is identical to the (e_t, e_n, e_b)-frame used in Section 7.

By default, the `orb_map`-frame is set up in accordance with the pinhole camera model, which means that the z-axis is oriented along the optical axis [10]. As a result, if the robot moves in a straight line from how it was initialised, the origin of the `base_link`-frame will move along the z-axis of the `orb_map`-frame. To conform with the kinematic model used in Section 7, another frame called `world` is introduced, and another static transform is defined, consisting of pure rotation, that orients its x-axis along the optical axis, the y-axis to its left, and the z-axis pointing upwards.

Next, a separate branch in the tree is added, which is the `snap`-frame. It is a child of the world frame, and is determined by a static transform, but can be updated during runtime. It is the snapshot of the `base_link` origin, relative to the world frame. Expressing the `base_link` position relative to this frame allows for resetting the origin arbitrarily during runtime.

Lastly, to ensure that the projection onto 2D planar motion is done with as high accuracy as possible, the gravity vector of the IMU is used to determine the "true" direction of the z-axis. This solves the problem that if the snapshot frame is initiated at a slight tilt on an otherwise planar surface, the z-coordinate would not remain zero, but instead be be proportional to the the distance from the starting point. This introduces the frame `snap_gravity`, and this transform is calculated every time the snapshot is initialised.

8.3 Reference Generation

The trajectory tracking algorithm elaborated on in Section 7 requires not only an estimated position, but also a continuous reference signal. This is accomplished with a ROS node written by the team, called `reference_generator`. It produces a continuous stream of reference positions (x, y, θ) and velocities $\left(\dot{x}, \dot{y}, \dot{\theta}\right)$, at a given rate. It takes in discrete waypoints, maximum velocity, the type of motion (linear, angular, or circular) and some extra information such as the radius of circles to be drawn. The transition between each waypoint follows a trapezoidal velocity, with the acceleration and deceleration phase computed using basic kinematic equations for linear motion.

The reference generator can be started, paused, and reset, and waypoints can be added and removed during runtime, all by calling various ROS-services.

8.4 Startup Procedures

ROS allows for nodes to be launched with the use of 'launch files'. Based on XML syntax, launch files allow for the nodes to be called at any time, with a number of arguments and settings to be passed in. The robot contains a package known as the 'main launcher' which contains launch files for running key system nodes, ORB-SLAM2 specific nodes, and LeGO-LOAM specific nodes. These launch files are controlled by the startup node, the key interface node between the Xbox controller and the software system. The startup node listens to the input from the Xbox controller and launches or starts nodes as required. Figure 20 shows a brief look at the commands available to the controller.

```
Manual control:
Driving throttle  --------  RT + Right stick (U/D)
Steering          --------  RT + Left stick (L/R)
Deploy tool       --------  X (hold)
Brake             --------  B
Disable drivers   --------  LT + LB

Autonomous control:
Start LeGO-LOAM   --------  LT + B
Start ORB-SLAM2   --------  LT + A
Snapshot (origin) --------  LT + RB
Start ref. path   --------  LT + Start
Stop SLAMing      --------  LT + Back

Miscellaneous:
Turn off computers --------  LT + Guide
```

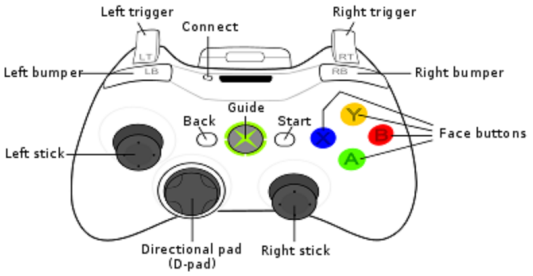


Figure 20: Xbox 360 controller robot commands

8.5 Software Diagnostics

For the most part, running diagnostics on the system is conducted through the ROS network. Since there is a router connected to the network, connecting an external laptop to the network over Wi-Fi was trivial. This allows for easy debugging of software by monitoring nodes, topics, and overall information flow for the whole network.

A set of 8 LEDs are installed on the robot with each LED indicating the status of important nodes such as, SLAM Status, Reference generator status and so on. A lit up LED indicates the node is active allowing for fast debugging when the robot is operating in the working environment.

Figure 21: LEDs for Software Diagnostics

9 Verification and Validation

This section outlines the verification and validation procedures undertaken in order to ensure that the requirements were being met, and developed in line with the stakeholders expectations.

9.1 Validation

Throughout the first few months when requirements were being developed, multiple iterations were undertaken with Trosam Automation AB. This led to an initial set of accepted requirements which were then used to develop initial concepts. During the implementation phase, the team had meetings with the stakeholders every Thursday. During these meetings the requirements, Appendix B, have constantly been discussed and adapted to ensure they meet stakeholder expectations. More precise technical requirements have been used, but have not been disclosed due to the fact that NDA-covered application specific terms are mentioned in most requirements.

9.2 Verification

The team periodically reviewed the technical requirements of this project to ensure that they verify the Stakeholder Requirements. As aforementioned, the technical requirements have not been disclosed in this report due to NDA restrictions, however, the tests that verify them can be briefly discussed in the following section. The team also completed tests throughout the project to ensure that software and hardware were functional before integrating them.

9.2.1 Completed Test Descriptions

SLAM Localisation Accuracy Tests In order to accurately position the robot in the environment, it is important that the generated SLAM map and pose estimation is scaled according to reality, as well as having adequate accuracy and precision. This is used to evaluate the performance of the ORB-SLAM2 and LeGO-LOAM algorithms with the chosen sensors, in a relatively sparse environment with some repetitive features.

An experiment based around driving the robot in a straight line to positions measured to be 2, 4, ..., 18, 20 metres in front of the robot, and extracting the pose

estimation data at each location. The known distance positions were marked with duct tape in the environment pictured in Figure 22. In order to account for any offsets in the starting angle, the Euclidean distance was used, incorporating both the x and y component of the estimated 2D-pose.

Figure 22: The testing environment with limited features visible to the sensors

The robot was manually driven between the known positions, which means that there are some errors due to simply positioning the robot accurately. The positioning of the robot was considered to be manual, but the 360 degree sensing of the LiDAR means that the robot could not be approached without changing the supposedly static environment. Unfortunately, cars and people passed by occasionally with such frequency that discrepancies were forced to be tolerated, despite potentially affecting the results. The experiment was also conducted in a short amount of time in order to avoid changes in illumination. To benefit of the experiment, the weather was overcast, removing any fleeting shadows and the risk of glare.

Control Accuracy Tests Once the positional accuracy is determined, the potential for the robot to position itself at a given location can be tested. Since any errors from the SLAM pose estimate will be included in this test, it is important to determine what the cause of the error is if the actual position does not match.

For the purposes of this test, the estimated position from the SLAM algorithms is treated as the ground truth, since all control required to stick to the path is using

this estimate only. When given a reference path to follow, the estimated pose will be graphed over time. The test cases used were a 10 meter straight line, a 5 meter straight line followed by a 3 meter radius circle. Some examples of ROS-bagged data samples, collected in a sparse environment similar to that pictured in Figure 22, will be shown.

General Tests A number of tests were carried out in order to test the accuracy and repeatability of the overall system using LeGO-LOAM, including the use of the (undisclosed) tool. Unfortunately, the testing process for ORB-SLAM2 was obstructed by limited light conditions and did not perform well in straight line tests outside so the team did not complete this test on ORB-SLAM2.

The test involves the robot travelling a specified trajectory. The trajectory consists of travelling in a straight line and continuing into a circle. The reference and the estimated trajectory of the robot are compared and the error is measured. This is calculated from analysing the ROS-bag. The enclosure of the circle is also measured which is the difference between the start position of the circle and the final point.

Extra tests or design considerations were made to verify the (public) Stakeholder Requirements from Appendix B and (undisclosed) technical requirements, conducted throughout the implementation process. A brief explanation of how the tests were verified and the results thereof are mentioned in the results.

9.2.2 Omitted Tests

Some tests have been omitted from this report due to the NDA and thus cannot be disclosed, especially the tests relating to the tool.

10 Results and Discussion

In this section the results of the tests will be presented. In order to evaluate the accuracy of the tools positioning, various tests were performed. The methodology and the results of the tests are stated in the subsections below.

10.1 SLAM Localisation Accuracy Results

The true distance travelled by the robot seems to have been accurately estimated by both SLAM algorithms: ORB-SLAM2 and LeGO-LOAM. The group is especially interested in whether the scaling is correct. If this is not the case, the robot would not have any way of detecting that it is actually over or under compensating the given path.

Figure 23 show the results of moving from 0 to 20 meters in a straight line from ORB-SLAM2 and LeGO-LOAM respectively. Similarly, Figure 24 shows the calculated Euclidean distance travelled according to the pose estimate of the robot for both algorithms. Each line refers to three separate trials of each test case. Assuming that the end of the blue line, which dips off at 20 meters in Figure 23, is not representative behaviour, the overall estimated distance seems to be very accurate.

Take note that both the LeGO-LOAM and ORB-SLAM2 results seem to have a bias towards a positive error. This is likely due to a systematic error from positioning the robot accurately when driving it manually. There should be an actual positioning error with a standard deviation of approximately 0.1 meters due to this factor, which may be skewed in either direction. There does not appear to be any error that is proportional to the distance driven, which supports the claim that the output from both algorithms is correctly scaled.

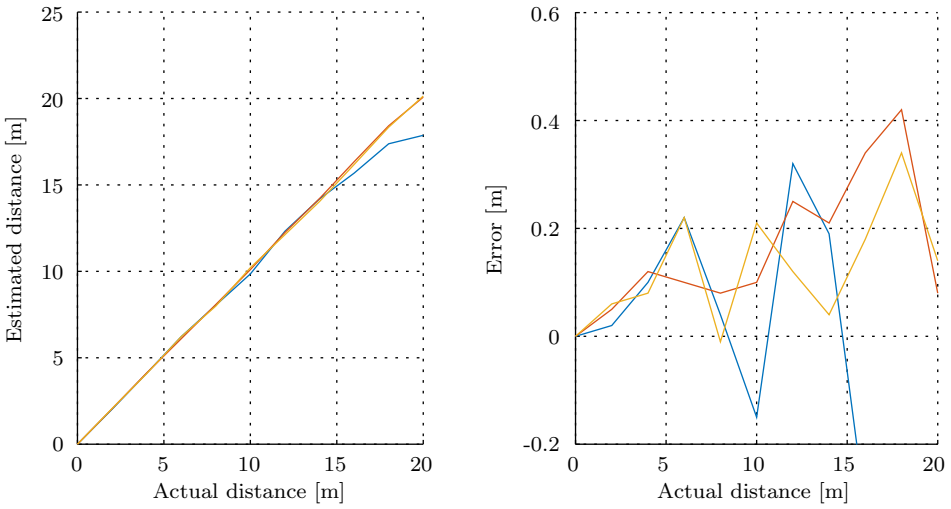


Figure 23: Straight line localisation - ORB-SLAM2

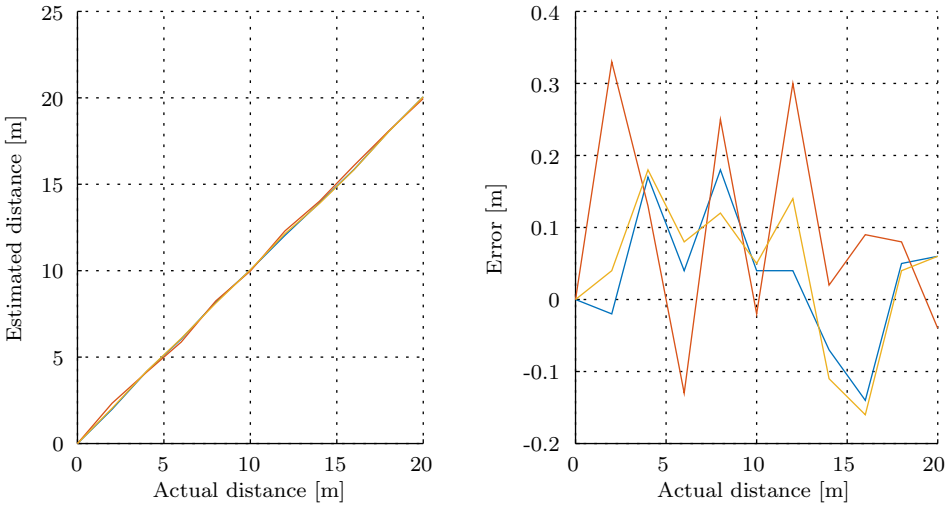


Figure 24: Straight line localisation - LeGO-LOAM

10.2 Control Accuracy Results

The control accuracy was tested through a variety of test cases, described previously. A few ROS-bags have been graphed here, mainly for LeGO-LOAM test runs.

In Figure 26 and Figure 25, a drift towards the positive y-direction can be seen, while the final distance has a very small error. This drift is probably due to turning issues, caused by the chassis design and back wheel placement. A small overshoot can be seen in the x-direction in Figure 25, which is fixed over time. This behaviour is not desired for a precise path following robot, but does not seem to consistently appear. In Figure 27 the y-direction drift instead occurs in the negative direction, but is not as prominent.

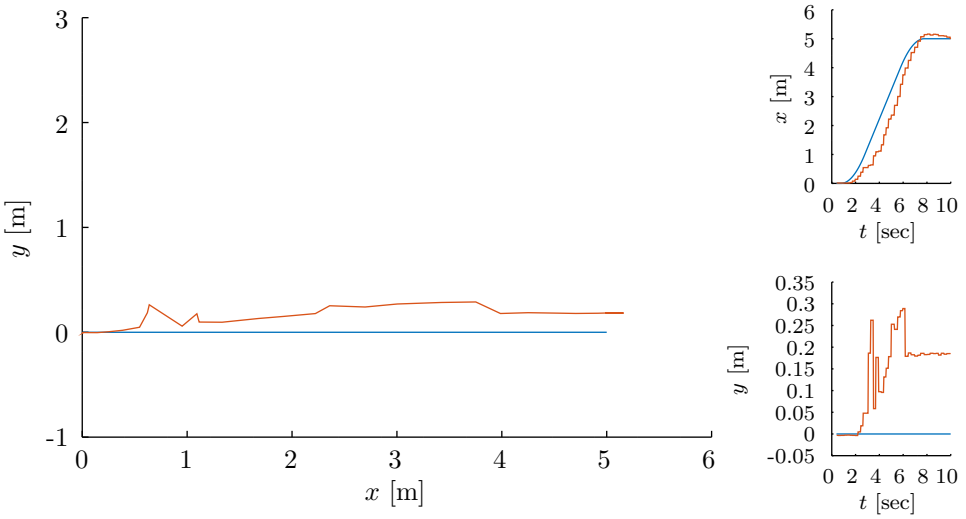


Figure 25: 5 metre straight line - ORB-SLAM2

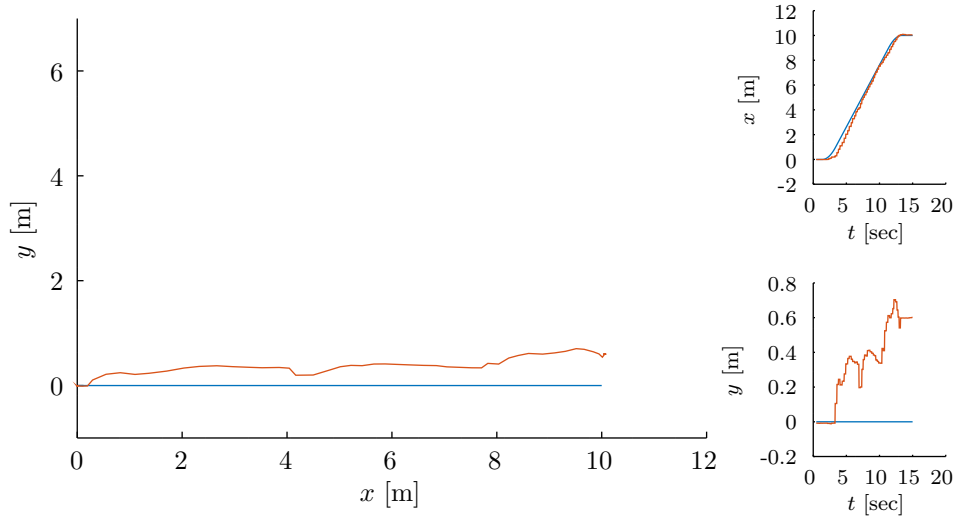


Figure 26: 10 metre straight line - ORB-SLAM2

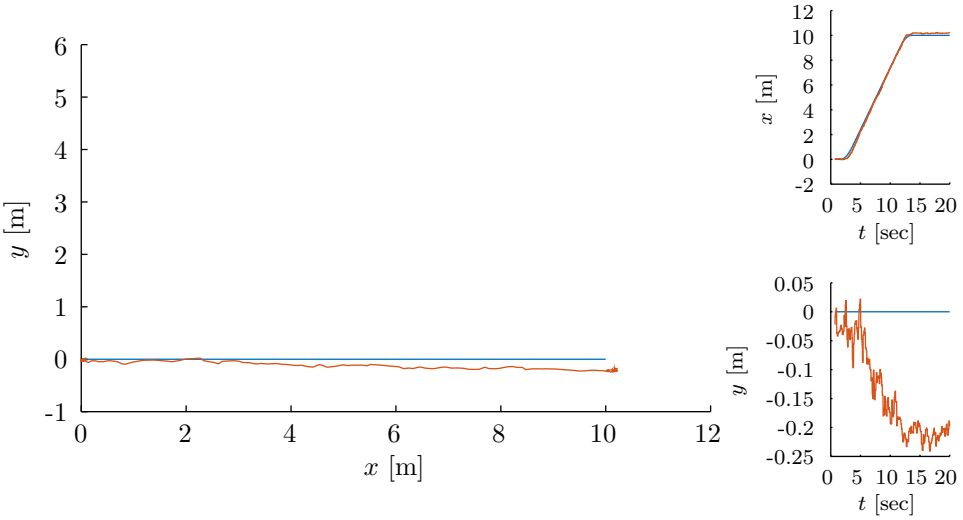


Figure 27: 10 metre straight line, run 1 - LeGO-LOAM

When it comes to the circle tests, shown in Figure 28 and Figure 29, it can be observed that the circle is closed quite well. It also can be seen that the robot is aware that it is drifting away from the center of the circle, which means that any problems can most likely be remedied.

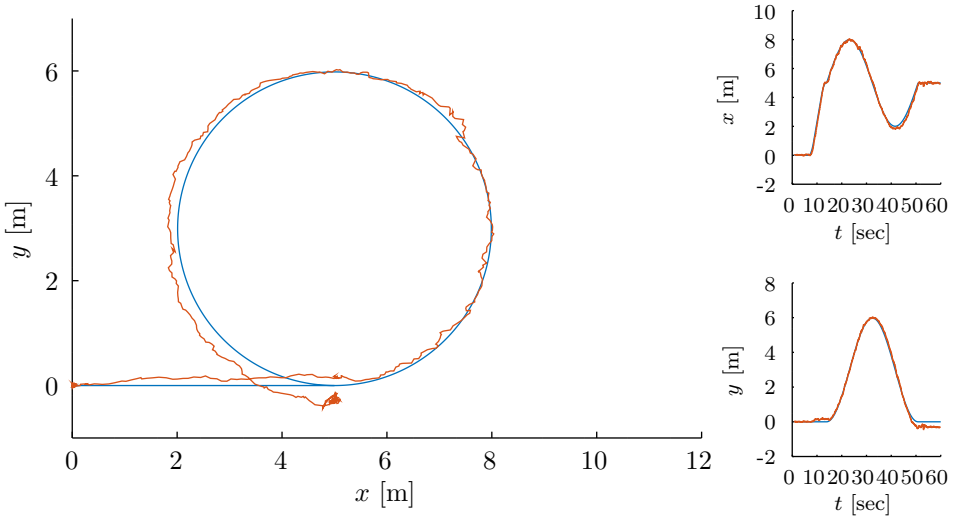


Figure 28: 3 metre radius circle, run 1 - LeGO-LOAM

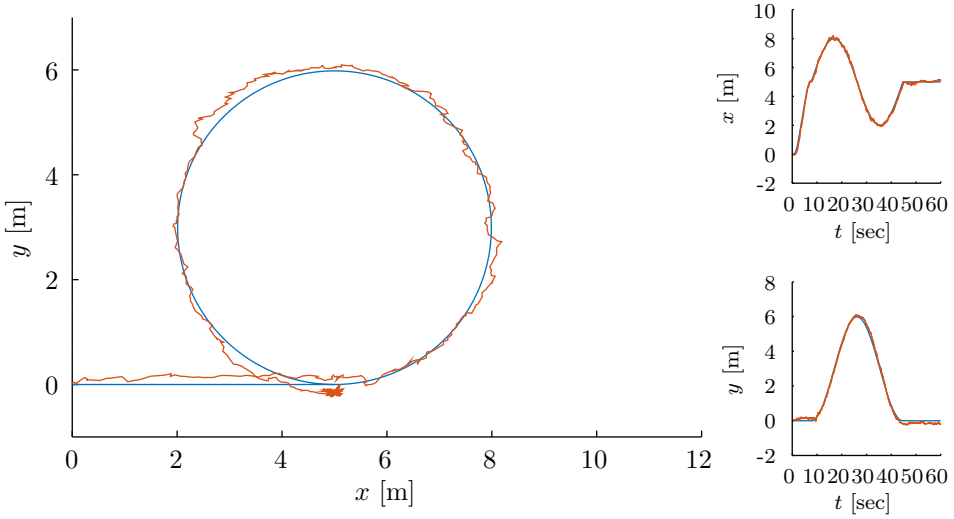


Figure 29: 3 metre radius circle, run 2 - LeGO-LOAM

Overall, the control accuracy seems to work quite well, although the y-directional drift should be looked into, since this error will add up over longer distances if the control algorithm is not able to counter act it. From physical observation, the group could determine that the nature of the (undisclosed) tool makes turning quickly an issue, and chassis construction leads to instabilities at high speeds. These is thought

to be the main issue for positioning the robot, rather than problems with the control algorithm itself.

10.3 Requirement based testing

The Stakeholder Requirements were verified as follows.

- **SR-1 Minimal (...) path crossing** - OMITTED. The virtual path was pre-programmed manually. No automatic complete path generation for the (undisclosed) task was implemented.
- **SR-2 User interface** - Diagnostic LEDs were added to quickly see what ROS-nodes are running, an Xbox 360 controller was used to perform certain commands, while ROS-network commands were used for finer tasks, such as changing parameters. This UI is implemented for testing purposes, per the requirement wording, and is not meant to be implemented into a final product.
- **SR-3 Main unit mass** - The prototype was weighed to be 33 kg including all batteries, instead of relying on the CAD part weights and supplier datasheets with their own respective errors. This is less than the required maximal prototype mass of 50 kg, according to Stakeholder Requirement SR-3 in Appendix B. There is the possibility to reduce the weight below the product target weight (excluding batteries) maximum of 25 kg.
- **SR-4 Operating conditions** - Through extended testing of the robot's path following functionality and tool use, the team was able to test the robot at a temperature of -5 to 5 degrees Celsius, primarily under overcast conditions. The actual proposed environment was not possible to use, due to a failure to meet SR-8, but there is no reason to believe this should not be the case once this is fixed, assuming enough features are present.
- **SR-5 Operation time per task** - Although the full path required for SR-1 was never tested, the robot was able to operate at speeds that would meet the requirement of maximum 17 minutes.
- **SR-6 Operational time without user maintenance** - The batteries last for more than one task to be completed, probably even four, without requiring any additional user maintenance, such as charging or providing other consumables for the robot.
- **SR-7 Obstacle handling** - OMITTED. Obstacle avoidance has not been implemented, but the modular system architecture and sensors allow for this to be implemented as future work.

- **SR-8 Terrain** - FAILED. The driver choice and hardware construction led to trouble moving in the (undisclosed) terrain required.
- **SR-9 Repeatability** - When the robot conducted positional tests, the results were repeatable between tests.
- **SR-10 Prototype cost** - Meets requirement for total cost to stakeholders, see final budget in Appendix A. Note that some hardware was available for use by KTH.
- **SR-11 Mass production cost** - OMITTED. The prototype cost exceeds the mass production cost, but it is likely that the mass production cost will meet the requirement, even if this was not investigated.
- **SR-12 Tool interfaces** - The robot has successfully integrated the tool, as was demonstrated in (undisclosed) tool tests.
- **SR-13 Limited external modules** - No external electronic modules were used by the robot, and only traffic cones and reflective tape were required externally, meeting the requirement.
- **SR-14 Virtual path following** - As demonstrated by the control accuracy tests, the robot is able to follow a virtual path.
- **SR-15 Not reliant on high precision GPS** - There were no GPS modules used in the implementation of the robot, meeting the requirement.
- **SR-16 Precision** - The driven path appears to be "perfectly straight" to the human eye, through (undisclosed) tool tests.

10.4 Ethics

The ethics of this project can be split into multiple domains, as described in the following section.

10.4.1 Development Practices

In the development of complex mechatronic systems, there are plenty of known and unknown hazards. Testing all possible states of the machine is simply impossible to do, so instead unit tests have to be designed for verifying local functionality. When run simultaneously, the programs may interact with one another, causing unintentional side effects. A simple example is that of a control loop which has to be performed at a given rate, which may not be guaranteed depending on the

utilisation level of the system. Due to the limited time frame, further testing and verification of the robot should be performed. The team did not always have the time or knowledge to figure out what was causing specific problems.

Since the robot uses sensors to perceive the world around it, and some of this data has been stored for analysis during development and testing (mostly in the form av ROS bags), the project members made sure to not save samples when anyone not within the group was around. This was not too much of an issue during final testing, since the specific application could not be disclosed to outsiders, so steps were taken to avoid contact with outsiders whenever possible.

There is a potential conflict of interest in developing this prototype for a commercial company without any promise of payment, apart from university credits. The stakeholders offered the members of the development team an opportunity to continue working on the robot after completing the school assignment. Could this lead to potentially obfuscating the functionality or otherwise sabotaging the product for the future, in order to be made non-replaceable for the stakeholders? It is unlikely, especially as the code had to be commented for review by other team members. The code is also backed up using version handling software, meaning that the code development over time is entirely visible. Some tacit knowledge of the system may inadvertently not be documented for the future.

The software and architecture developed for this specific project can be transferred to other projects, and is generally not a new approach. Large parts of the design has been made publicly available by this report, which means that any competitors could simply attempt to copy the prototype to save development costs. It is a balancing act of how much the authors can disclose of their work here, without disclosing particularly valuable details of the robot design.

10.4.2 Safety and Social Impact

Like many automated robots, the use of this specific platform is not likely to cause the removal of any full-time positions, but can instead remove the need for some repetitive tasks [22]. This can arguably be a good thing, as the human potential can instead be used for tasks that require a human touch and are resistant to automation under current economical and technological constraints.

This specific robot has been designed to operate in an environment with few, if any, humans present, but it is still heavy enough to cause damage if it goes out of control or manages to leave the designated area. This is particularly an issue since it does not use any type of active obstacle detection at the time of writing, although this is certainly possible to add using the existing sensors. During the testing phase of

the prototype, it was realised that a quick way to stop the robot was required. The Xbox 360 controller that was used for manual driving of the robot had a command for disabling the motor drivers, and a physical switch was included for all power circuits should the controller become unresponsive.

It is also a relatively expensive solution that could easily be stolen if left unsupervised in its operational environment, which may be hard to block off completely. This is especially important to consider since the LiDAR alone has a market value of roughly a monthly salary.

10.4.3 Environmental Impact

If used continuously this robot may be an economically viable alternative that can spread it's environmental impact over a long life cycle. The developed chassis is very modular and can be taken apart for recycling the aluminium profiles and metal sheets. There is also a significant amount of cables used internally to connect the different subsystems, from which the wiring can be recycled. The sensors, computation boards, and batteries have to be taken care of in a responsible manner. Unfortunately, they are not easily recyclable if a replacement is needed, and should be re-purposed at the end of the robot's life if they are still operational. If possible, the batteries should be charged using only renewable sources of electricity.

The consumables used by the robot are purposefully made to degrade over time, and is the same product as what is used when the task is performed manually. This means that any developments in the industry standard can most likely be applied directly to this specific robot. There is some potential for dynamical fine-tuning the rate of consumption during run-time, but this has not been considered for the prototype.

11 Conclusion

The project was completed on time with no major issues, even if there is plenty of room for improvements for future iterations of the prototype platform that came up during development. This means that the delivered prototype may not quite match up with the specified requirements set at the beginning of the project, since one can not know what issues may arise beforehand.

11.1 Project Outcome

The resulting prototype can be used as a proof-of-concept for the specific application of the robot, but it requires further development to fulfil all the requirements that were the end goal of the project. The main issue appears to be the sparse environment as the mapping and localisation worked quite well in feature dense environments. Improving the performance in sparse environments will require a better LiDAR and possibly additions of features to that environment in order to facilitate the localisation.

11.2 Further Development

The robot prototype produced in the project was designed under significant time pressure, while the authors often studied multiple courses simultaneously and occasionally struggled to complete particular parts of the prototype. This meant that some corners had to be cut, and many improvements were not logistically possible to implement within the given time frame.

In order to continue the project a number of suggestions are made, divided into their respective fields as described below.

11.2.1 Software

SLAM algorithms

- LeGO-LOAM contains parameters that can be adjusted and also additional functionalities that are not used on the prototype. These are listed as follows.
  - LiDAR data rate - reduce the number of data points from the LiDAR to reduce the computation time

  - `c_th` - lower threshold for the number of LiDAR points in a detected feature that is used for mapping
  - `groundScanInd` - the number of LiDAR layers to be used specifically for ground segmentation
  - Loop Closure Flag - Investigate the difference between loop closure on and loop closure off. It would be interesting to determine the difference in computation time and performance.
  - IMU Integration - The IMU integration can be finalised and the difference between the inclusion and non-inclusion of its data in the LeGO-LOAM algorithm would be interesting to investigate.
  - GPU utilisation - Implement LeGO-LOAM on the GPU.
- ORB-SLAM2
  - ZED camera calibration - Calibrating the ZED camera in various ways presented different results in the indoor environment. However, it would be interesting to see if calibration outdoors would make a difference.
  - GPU utilisation - Implement ORB-SLAM2 on the GPU.
- Alternative ideas
  - Different SLAM algorithms - Only LeGO-LOAM and ORB-SLAM2 are applied in the prototype however, other algorithms could perform better.
  - Algorithm Fusion - Fusing the position estimates of both algorithms and determining which output is more reliable would be an interesting future works.

Localisation Strategy Currently the system determines its world frame from its start position or a particular position when triggered. However, an alternate approach, one more in tune with the intended operational application, is to use known features to determine a "global" frame.

There are some objects in the operational environment (the exact nature of these can not be specified due to the the NDA), that need to be located. The robot then activates the tool at specific points in reference to these locations. The objects can be manipulated to be correctly identified, such as using QR-codes, visual object recognition techniques, or (naively) by separating LiDAR point cloud data based on reflectivity after applying reflective tape to them.

The latter approach is based on the assumption that the reflectivity measurements are constant regardless of distance from the sensor, describing an object property rather than a sensor value. The drawbacks to this method are if the desired points are not the only ones within a pre-scribed reflectivity range, for example if a human wearing a reflective jacket happens to be present in the operational environment.

The features are roughly located in a plane, so once the locations of the points is found, a plane is fit to the points. The placement of the points is known, which means that a global coordinate system can be set up with reference to one of the points in the plane. The estimated position of the robot can then be expressed in this coordinate system.

Processing Speed Real time algorithms benefit from sampling sensor data as quickly as possible, allowing for a balanced trade-off between the amount of data points per sample versus computation time. The result is increased SLAM accuracy otherwise, since new data can be parsed more often, meaning that the robot has not moved very far between pose estimates. The authors of the report drove their test platform around at an approximate speed of 1.3 m/s (comparable to walking speed), while the stakeholders require slightly higher speeds to beat competitors.

The difference between running the algorithm on the same *Velodyne HDL-64E* dataset on an NVIDIA *Jetson TX2* or an *Intel i7 4710MQ*-based laptop computer is described in the LeGO-LOAM report, where down-sampling to fewer laser samples is required for real time execution. Note that the SLAM algorithms implemented in this project were not modified in order to actually use the CUDA cores of the NVIDIA *Jetson TX2*, meaning that the computation time may be possible to improve significantly through software optimisation.

Network Traffic It was difficult and inconsistent to be able to connect via SSH to the NVIDIA *Jetson TX2* when the LiDAR was actively running. It may be a case of reaching the maximum networking throughput of the Jetson or perhaps even the router, but this seems unlikely. This should be investigated further, in case it is a potential failure mode.

Future Additions ORB-SLAM2 had trouble detecting enough features in sparse environments using the ZED camera. However, the SLAM pose estimation worked well when testing in an indoor environment (with plenty of furniture as visual features), but struggled when testing commenced in a nearby parking lot. It is clear that the surface of a parking lot is not patterned distinctly enough to be used for

features, which would most likely cause trouble in the actual environment as well. However, the inclusion of even a monocular camera opens up for the possibility of detecting certain occasional features that may be of use for an improvement of the robot, for a slightly different use case.

11.2.2 Hardware

Hardware Stability It was observed that the robot's swivel wheels, located at the back of the robot, would start to oscillate uncontrollably at higher speeds, leading to difficulties steering. An improvement would be to simply drive "backwards", but with the designed chassis, the tool placement would not be suitable for the given application. Since there was limited time to re-design the chassis, this is left as future work.

Motor Drivers Another large setback was the choice of motor drivers. The first implementation employs hub motors taken from a self-balancing scooter, and borrowing available no-name, entry-level drivers from KTH. Unfortunately it was realised that the drivers did not have all the required capabilities. Thus a pair of motor drivers from STMicroelectronics were ordered, however they proved to not be suitable for the job. When the drivers delayed the robot tests according to schedule, it was decided to purchase some local drivers. Whilst these drivers performed much better it was discovered that they where primarily high speed low torque drivers, due to their 5 A output current limit. Upon research there are 3 possible solutions to the issue. The problem could be solved by purchasing motor drivers that can output higher current and thus output higher torque or by purchasing hub motors with an acceptable data sheet. Another possibility to solve the issue is to modify the drivers to overwrite the output current limit. However, the last solution is not optimum since there is a high risk that the driver gets destroyed.

LiDAR Resolution The Velodyne VLP-16 LiDAR that was used did unfortunately not provide sufficient point cloud resolution to properly detect the features in the operational environment, but did work well when plenty of additional features (traffic cones) were placed in the environment, in order to make the environment less sparse. A high-end LiDAR from Velodyne can potentially be directly paired with a Nvidia Jetson AGX Xavier computer, which is more expensive (and had not been released at the time of the SotA analysis), but offers improved processing power by an order of magnitude. If the CUDA cores are not used, an Intel NUC-series computer may be a plausible alternative which can handle the increased LiDAR data stream. The team verified the capabilities of the LiDAR resolution by wrapping a team member in reflective tape, and seeing how far away said person was able to

move from the LiDAR before it was unable to reliably detect the location of the person from a point cloud shown in RViz. The break point occurred at approximately 15 metres, where the angular resolution was the limiting factor, rather than the distance itself. Many points corresponded to objects further away, such as trees and buildings, but they were too sparse to discern any details.

Battery Status Indicator Another aspect that was not implemented properly was a method of detecting when the battery voltage drops below a certain value, and eventually shutting down the system to avoid damage to the batteries. Instead, active monitoring of the voltage using multimeters was required. Additionally, brown outs where experienced when everything was connected to a single battery, and the easy solution was to add an additional battery for the tool components itself. It would be more convenient to only rely on a single battery, in terms of charging, compactness, etc.

References

[1] D. Electronics, "Denver dbo-10001white," 2018.

[2] J. S. Pawin Jawayon, "Speed estimation of 3-phase bldc motorusing genetic algorithm," *International Journal of Engineering Science and Innovative Technology (IJESIT)*, vol. 2, 2013.

[3] T. Lee, "Why experts believe cheaper, better lidar is right around the corner," 2018-01-01. Accessed: 2018-04-16.

[4] N. Inc, "Imu errors and their effects," *APN-064*, vol. Rev A, 2014-02-21. Accessed: 2018-04-26.

[5] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *CoRR*, vol. abs/1607.02565, 2016.

[6] M. Aqel, M. Marhaban, I. Saripan, and N. Ismail, "Review of visual odometry: types, approaches, challenges, and applications," *SpringerPlus 5.1*, 2016-10-28.

[7] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *Robotics, IEEE Transactions on*, vol. 32, pp. 1309–1332, December 2016.

[8] A. R. Khairuddin, M. S. Talib, and H. Haron, "Review on simultaneous localization and mapping (slam)," *2015 IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, pp. 85–90, 2015.

[9] A. Huletski, D. Kartashov, and K. Krinkin, "Evaluation of the modern visual slam methods," *2015 Artificial Intelligence and Natural Language and Information Extraction, Social Media and Web Search FRUCT Conference (AINL-ISMW FRUCT)*, pp. 19–25, 2015.

[10] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[11] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "Orb-slam: A versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, pp. 1147–1163, Oct 2015.

[12] H. Strasdat, J. Montiel, and A. J. Davison, "Visual slam: Why filter?," *Image and Vision Computing*, vol. 30, no. 2, pp. 65 – 77, 2012.

[13] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," in *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pp. 225–234, Nov 2007.

[14] T. Taketomi, H. Uchiyama, and S. Ikeda, "Visual slam algorithms:

a survey from 2010 to 2016," *IPSJ Transactions on Computer Vision and Applications*, vol. 9, December 2017.

[15] T. Shan and B. Englot, "Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4758–4765, IEEE, 2018.

[16] G. Carbone, *Motion and Operation Planning of Robotic Systems Background and Practical Approaches*. Mechanisms and Machine Science, 29, Springer, 2015.

[17] O. S. R. Foundation, "About ros," 2018.

[18] N. Fugal, "Efficiently reading quadrature with interrupts," 2013-10-02. Accessed: 2018-09-29.

[19] E.-M. Drivers, "Styrdon bldc-1kw," 2005.

[20] M. Motors, "Escon 50/5 drivers," 2005.

[21] I. HillCrest Laboratories, "Sh-2 reference manual (document number: 1000-3625)," 2017.

[22] L. Schlogl and A. Sumner, "The rise of the robot reserve army: Automation and the future of economic development, work, and wages in developing countries," 2018.

A Budget

Trosam Expenditure to Date					13/12-2018
Total SEK 32 470,28					
KTH Paid SEK 10 580,00					
Item	QTY	KTH Bought?	Cost	**Cost(SEK)**	**KTH Cost(SEK)**
Zed Camera	1,00	0	SEK 4 300,08	SEK 4 300,08	SEK 0,00
Velodyne Puck LIDAR	1,00	0	Donated	Donated	SEK 0,00
Nvidia Jetson TX2	1,00	0	SEK 4 227,14	SEK 4 227,14	SEK 0,00
IMU - FSM305	2,00	0	SEK 665,00	SEK 1 330,00	SEK 0,00
Raspberry Pi	1,00	0	SEK 240,00	SEK 240,00	SEK 0,00
Raspbery Pi Accessories	1,00	0	SEK 360,00	SEK 360,00	SEK 0,00
Reflective Tape	1,00	0	SEK 342,50	SEK 342,50	SEK 0,00
Motors and Drive Train	1,00	0	SEK 2 400,00	SEK 2 400,00	SEK 0,00
Drivers – STM	2,00	0	SEK 560,47	SEK 1 120,94	SEK 0,00
Drivers – Maxxon	2,00	0	SEK 2 141,81	SEK 4 283,62	SEK 0,00
Wjet	15,00	1	SEK 450,00	SEK 6 750,00	SEK 6 750,00
Nucleo-64	1,00		SEK 150,00	SEK 150,00	SEK 0,00
Wjet-Metal	1,00	1	SEK 200,00	SEK 200,00	SEK 200,00
3D printed material	1,50	1	SEK 300,00	SEK 450,00	SEK 450,00
Arduino Mega	1,00	1	SEK 500,00	SEK 500,00	SEK 500,00
PCB and Electronics	2,00	1	SEK 300,00	SEK 600,00	SEK 600,00
NödStopp	1,00	1	SEK 300,00	SEK 300,00	SEK 300,00
Wheels	2,00	2	SEK 200,00	SEK 400,00	SEK 400,00
Traffic Cones	6,00	0	SEK 206,00	SEK 1 236,00	SEK 0,00
Various Nuts and Bolts	1,00	1	SEK 300,00	SEK 300,00	SEK 300,00
Aluminium Profiles	1,00	0	SEK 900,00	SEK 900,00	SEK 0,00
Batteries and Tool related	1,00	0	SEK 600,00	SEK 600,00	SEK 0,00
Cabling From KTH	1,00	1	SEK 150,00	SEK 150,00	SEK 150,00
Cabling Bought	1,00	1	SEK 190,00	SEK 190,00	SEK 190,00
SSD	1,00	1	SEK 400,00	SEK 400,00	SEK 400,00
SSD Cable	1,00	1	SEK 90,00	SEK 90,00	SEK 90,00
Laser Print Material	1,00	1	SEK 100,00	SEK 100,00	SEK 100,00
USB Hub	1,00	0	SEK 400,00	SEK 400,00	SEK 0,00
Power AUX Connections	3,00	1	SEK 50,00	SEK 150,00	SEK 150,00

B Stakeholder Requirements

SR-1 - Minimal previously covered virtual path crossing
The virtual path should be optimised in order to create the pattern with minimal cross-overs of previously covered pattern areas.

SR-2 - User interface
While a user interface is required for a final product, the prototype only needs a basic text based user interface for debugging purposes and initialising the prototype to deliver one pre-set pattern.

SR-3 - Target mass
The production robot excluding payload should not exceed 23 kg according to European work regulations, since the prototype will be lifted in and out of a transportation vehicle. Separate components, such as the consumables and batteries are OK to remove before lifting in order to reduce the lifting weight. The prototype is allowed to be 50kg along as there is the possibility for the weight to be reduced to 23 kg in series production.

SR-4 - Operating conditions
The prototype shall operate in ideal daytime outside conditions. This includes optimal weather (no fog, no rain, soft sunlight). The prototype may utilise fixed features marking the corners of the operational environment. There may also be removable features placed by hand within the operational environment.

SR-5 - Operation time per task
The robot should perform the task in the largest allowable pattern in under 17 minutes, excluding calibration time or first time setup.

SR-6 - Operational time without user maintenance
Minimum one task, optimally four, should be possible to complete without any user maintenance.

SR-7 - Obstacle handling
Known obstacles should be approached with care to apply tool, unknown obstacles should be avoided.

SR-8 - Terrain
The robot should be designed to primarily operate on uneven surfaces, for example grass or gravel.

SR-9 - Repeatability
The robot should be able to perform the same tasks repeatedly and deliver consistent results.

SR-10 - Prototype cost
The prototype cost should not exceed 5000 EUR.

Some ball park recommendations:

- Mechanical chassis: 500 EUR
- Sensors: 2500 EUR
- Consumables per operation: 20 EUR

SR-11 - Mass production cost
The cost for components should not exceed 4000 EUR in mass production.

Some ball park recommendations:

- Mechanical chassis: 500 EUR
- Sensors: 2500 EUR for 100 units per year
- Consumables per operation: 20 EUR

SR-12 - Tool interfaces
The robot shall be compatible with the tools provided by the stakeholders.

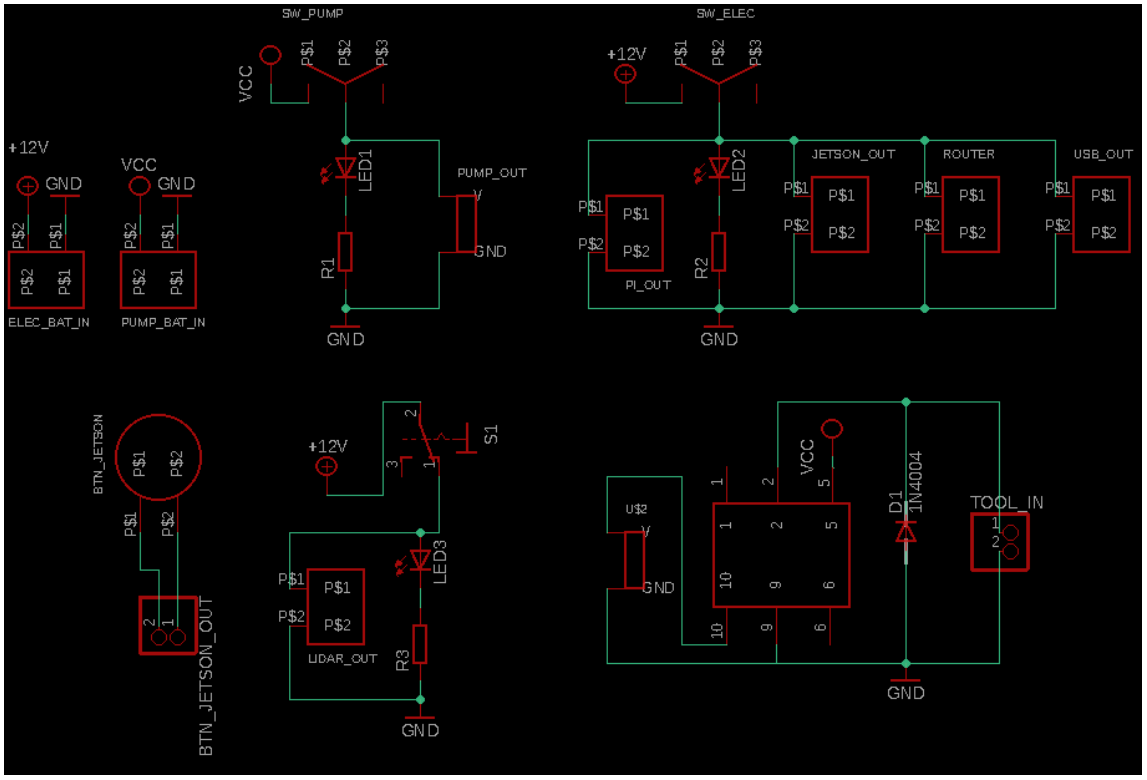
SR-13 - Limited external modules
The prototype system's electronics should be placed on the main unit, with as few external modules as possible. These external modules should cost less than 200 EUR in total to replace, in order to safely keep them at the operational environment, while the main unit shall be transported between operational environments.

SR-14 - Virtual path following
The robot should be able to follow straight and curved paths. When complete, the robot should be able to follow a basic pattern decided together with the stakeholder.

SR-15 - Not reliant on high precision GPS
Other alternatives for precise positioning should be explored because high precision GPS are generally expensive, competitors use them (competitive differentiator

desired), and has difficulties in certain operational environments for this product.
SR-16 - Precision
The pattern shall appear "perfectly straight" to the human eye.

C Switchbox Schematic



D Motor Fastener Drawing