Norges miljø- og
biovitenskapelige
universitet

Master's Thesis 2017 30 ECTS
Faculty of Science and Technology

Safety System for Agricultural Robot

Eirik Wormdahl
Mechanics and Process Technology

Preface

This thesis is the fulfillment of my Master's degree at the Norwegian University of Life Sciences.

First of all, I would like to thank my supervisors Prof. Pål J. From and Lars Grimstad for hours of discussions and guidance. A special thanks to Pål, for introducing me to the field of robotics, and for sharing his knowledge with me. My gratitude also goes to Lars for hours of guidance and for being available well beyond office hours.

Thanks also to my family and Solveig, for moral support and an endless stream of useful comments on the manuscript. I would also like to thank my fellow master students from The Loft: Erling, Eirik, Magne and Helene. This semester wouldn't have been as enjoyable without you!

Ås, May 12th, 2017

Eirik Wormdahl

Abstract

Development of autonomous robots requires strict demands to safety, as robots are able to do severe damage. In order for the NMBU-developed mobile agricultural robot, Thorvald, to work autonomously on the fields, a safety system must be in place. This system must, above all, make sure the robot does not kill, or injure humans.

In this thesis, a safety system for Thorvald is designed. Various sensor technologies, applicable controllers and safety systems applied in other robotic platforms are studied. A set of system requirements is formulated.

The suggested system is layer-based, specifically consisting of four layers, working independently. Layer 1 uses both 2D and 3D-LIDARs for navigation and obstacle avoidance, while layer 2 stops the robot, if a separate 2D-LIDAR senses an object within thirty centimeters of the front of the robot. In case of a collision while Thorvald is equipped with heavy or sharp tools, layer 3 stops the robot at impact, by the use of safety edges. Layer 4 gives humans the possibility to override the system with an emergency stop button.

Furthermore, a case study is conducted, as a start of developing layer 2 of the safety system. The selected 2D-LIDAR is connected to an Arduino. A C++ program is written and run on the Arduino. As a substitute for motor control, the Arduino gives commands to a LED light that lights up when an object is too close. Both the scanner and program worked as intended.

Sammendrag

Utvikling av autonome roboter stiller strenge krav til sikkerhet. For at NMBUs selvkjørende landbruksrobot, Thorvald, skal kunne arbeide selvstendig på jordet, må et sikkerhetssystem være implementert. Dette systemet må først og fremst sørge for at roboten ikke dreper eller skader mennesker.

Hovedmålet med denne oppgaven er å designe et sikkerhetssystem for Thorvald. Ulike sensorteknologier, egnede kontrollere, og sikkerhetssystemer fra andre robottyper blir studert. I tillegg formuleres en rekke krav som sikkerhetssystemet må oppfylle.

Sikkerhetssystemet som blir utarbeidet har en lagstruktur og består av fire uavhengige sikkerhetslag. Det første laget bruker både 2D- og 3D-laserskannere (LIDAR) for å navigere og unngå å kjøre på hindringer. Lag nummer to sørger for at roboten stanser hvis en hindring dukker opp mindre enn tretti centimeter fra robotens front: dette ved hjelp av en enkeltstående 2D-LIDAR. Dersom en kollisjon likevel skulle inntreffe mens Thorvald er utstyrt med tungt eller skarpt utstyr, vil sikkerhetslag 3 stoppe roboten ved hjelp av en støtfangersensor. Lag fire, i form av en nødstopp-knapp, åpner for menneskelig kontroll.

Videre utføres en case study, som en start på utviklingen av sikkerhetslag to. Her blir en 2D-LIDAR, koblet til en Arduino. Videre skrives et C++-program som blir kjørt på Arduinoen. Som en erstatning for motorkontroll, sender Arduinoen signaler til et LED-lys, som lyser når et objekt kommer for nært. Både skanner og program fungerer som det skal.

List of Figures

Contents

Chapter 1

Introduction

1.1 Motivation

In his short story *Runabout*, science fiction writer Isac Asimov formulates *Three Laws of Robotics* [26]. Their purpose is to protect humans from robots and to protect robots from themselves [27]. The three laws are as follows [26]:

1. A robot may not injure a human being, or, through inaction, allow a human being to come to harm.

2. A robot must obey the orders given it by human beings except where such orders would conflict with the First Law.

3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Laws.

These laws are too general to be implemented directly into robotic systems, but still provide guidelines for a safety system to be designed. The ever-growing amount of autonomous robots, results in a greater need for high safety requirements. Autonomous robots that operate in uncontrolled environments can through unsafe operation potentially cause significant losses: human, material and economical. Having safety systems implemented will be required for the presence of such robots to be accepted. A mobile robot's working environment is often dynamic and unstructured. The challenge is to make the robot execute its often rather complex tasks in collaboration with its human co-workers, without causing damage.

A system's safety refers to its ability to operate without causing an accident. An accident is here defined as an unplanned and undesired event that causes at least some defined level of loss [27]. In order to prevent accidents, we must be able to define the robot's state in the time prior to a potential collision. For example, before

a robot hits a tree, there must be a period of time when the tree is in the robot's planned driving path. These precursors of accidents must be under the control of the system designer.

For some industries, the development of autonomous robots has come a long way, and self-driving cars and buses are now reality. Vacuuming and lawn-mowing are being done by robots, airplanes have autopilot, and robots are bringing supplies around in hospitals and warehouses. All of these robots have a safety system implemented. Studying these systems will be very helpful when determining how a safety system for an autonomous agricultural robot should be designed.

1.2 Thorvald

The development of NMBU's agricultural robot, Thorvald, continues at full speed [32, 31]. The goal is to develop a commercial, autonomous, mobile robot platform that will be able to perform various farming tasks, such as seeding, weeding and harvesting. A key property of the robot is its low weight, which allows it to operate during wet periods without causing damage to the soil structure. Soil compaction can thus be significantly reduced compared to heavier, conventional farming vehicles. To make Thorvald more versatile, the new model, Thorvald II, is module-based, allowing for several different assembly configurations [31]. Some of the assemblies are shown in Figure 1.1.

Figure 1.1: Lineup of some possible assemblies of Thorvald's building modules [31].

Figure 1.2: The standard module assembly of Thorvald.

The standard version of Thorvald is shown in Figure 1.2. To enable turning around its own axis, this version has four-wheel drive and four-wheel steering. The track width is 1.5 meters and the frame has a mass below 200 kg. The robot's internal workspace is large, and it is capable of carrying a payload up to 200 kg [31].

Thorvald is currently being used in various research projects. The applications are many: For example, the robot can do various tasks in an open field, under both dry and wet conditions. It can also be used for farm logistics, like carrying full boxes of strawberries from a field to a storage area.

There are several large challenges associated with making an autonomous agricultural robot, e.g. regarding design, fitting of equipment and navigation. For navigation, Thorvald will calculate driving paths by sensing its environment, and compare the sensed objects to a map of the surroundings. Even though a path looks the same from day to day on a map, an outdoor field is always in development, and obstacles can occur in various places at different times. The robot must be able to work alongside humans, and avoid hitting obstacles in its path, living or not. This might be farm workers, children playing, animals or equipment that has been moved, to name a few. To be able to sell the robot, a safety system making sure that Thorvald does not crash, thereby hurting humans, animals or itself, must be in place.

1.3 Scope of the Thesis

The purpose of this thesis is to design a safety system for the agricultural robot Thorvald, and select the components to be used in this system, as well as their placement. The system must fulfill a set of requirements, the most important being not to kill humans. Further, it must be transferable between the various Thorvald configurations.

Firstly, sensor selection will be done based on a study of various applicable sensor types, with regards to criteria including accuracy, simplicity of analyzing sensor output, sensor range, usability under different weather conditions, need for computational power and price.

Thereafter, studying safety systems implemented in other mobile robotic platforms will give some insight into how other robot developers solve their safety challenges.

Finally, a case study will be conducted, with the aim to determine whether a chosen sensor will be applicable for use.

Chapter 2 discusses various applicable sensor technologies for use in safety systems. Thorvald's software framework and some of its properties are explained in Chapter 3, in addition to a brief discussion of two other controllers. In Chapter 4 we look into other autonomous robotic platforms and their safety systems. A number of safety requirements are defined in Chapter 5. Further, the designed safety system is presented in Chapter 6. Chapter 7 contains a case study, where the functionality of a laser scanner is tested. The designed safety system and the case study are discussed in Chapter 8.

Chapter 2

Sensor Technologies

Obtaining knowledge about its environment is an important task for any autonomous system, mobile robots included. The robot will solve this task by receiving measurements from different sensors and extracting useful information from those measurements. These sensors can be used to measure simple values like rotational speed of the motor. More sophisticated sensors are used when information about the robot's environment is wanted. As mobile robots move around unattended, they will regularly encounter unforeseen changes and variations in their environment. This makes the ability to sense the environment crucial when designing a safety system for a mobile robot [53]. This chapter introduces how we can classify sensor technologies, before describing specific sensor types that should be considered for Thorvald's safety system.

2.1 Sensor Classification

Sensors are classified based on two characteristics: whether they measure internal or external conditions, and whether they only receive, or also emit, energy [53].

- *Proprioceptive* sensors measure the robot's internal values, such as battery voltage or motor speed.

- *Exteroceptive* sensors obtain information from the robot's surroundings, such as distances or light intensity. Thereby, these measurements must be interpreted by the robot to extract environmental features.

- *Passive* sensors measure energy entering the sensor from the environment. Such sensors include microphones, cameras, gyroscopes and compasses.

– *Active* sensors emit energy, before measuring the reaction from the environment. These sensors are often superior in terms of performance, since they have more controlled interactions with the environment than the passive sensors. The emitted energy can, however, affect the very characteristic that is being measured. An active sensor can also experience interference between its own signal and another beyond its control, for instance from a similar sensor placed on the same robot. Ultrasonic sensors and laser scanners are examples of active sensors.

2.2 Safety System Sensors

The list of sensors that could be used in a safety system, is very long. In this chapter the most relevant for use on Thorvald are discussed.

2.2.1 Laser Scanners

Laser scanners are active, exteroceptive sensors that can be used to measure the distance between the scanner and its surrounding objects. These sensors are often called LIDARs, which stands for Light Detection and Ranging. A light beam is rapidly being fired from the instrument, hitting its surroundings, at up to 150,000 pulses per second [7]. A part of the light beam is reflected back to the scanner. The most common way of using a laser sensor for distance measurement is the time-of-flight method [36]. The instrument measures the time it takes before the reflected beam returns. The distance between the sensor and the object is then given by the equation

$$D = \frac{c * t_f}{2} \tag{2.1}$$

where D is the distance to the obstacle, c is the speed of light and t_f is the time of flight.

By firing the light beams at a rotating mirror making an angle of 45 ° degrees with the horizontal plane, a point cloud of what the scanner "sees" of its surroundings, in the horizontal plane, can be made. It is also possible to get 3D laser scanners, that can make a three-dimensional point cloud of its surroundings, which computer software can then convert to a readable map.

The maximum range of a laser scanner is dependent on the reflectivity of the surrounding surfaces [36]. Lighter surfaces will typically reflect better than darker, and smooth surfaces better than rough.

Some laser scanners are not eye safe, and eye damage can occur if hit by a laser beam. Therefore, they are categorized by classes, ranging from 1 to 4, where class-

1 lasers are considered safe, while class-4 lasers will damage both skin and eyes by exposure. Most laser scanners belong to the sub-class 3R, which can be hazardous to the eyes if viewed directly [23]. However, since the scanner's beam often is rotating, the energy is not focused in the same place over time, making eye safety less of a problem.

Laser scanners are not dependent on daylight, which makes them operable day and night. The target must however be visible, hereby making operating in thick fog impossible [49].

As a price example of a low-priced 2D laser scanner, the RPLIDAR A2 360° Laser Scanner costs $449.00 [25]. The Kokvyo UTM-30LX, a more high-end 2D-scanner costs $5,290.00 [6], while an example of a 3D laser scanner is the Velodyne VLP-16, which can be bought for $7,999.00 [18].

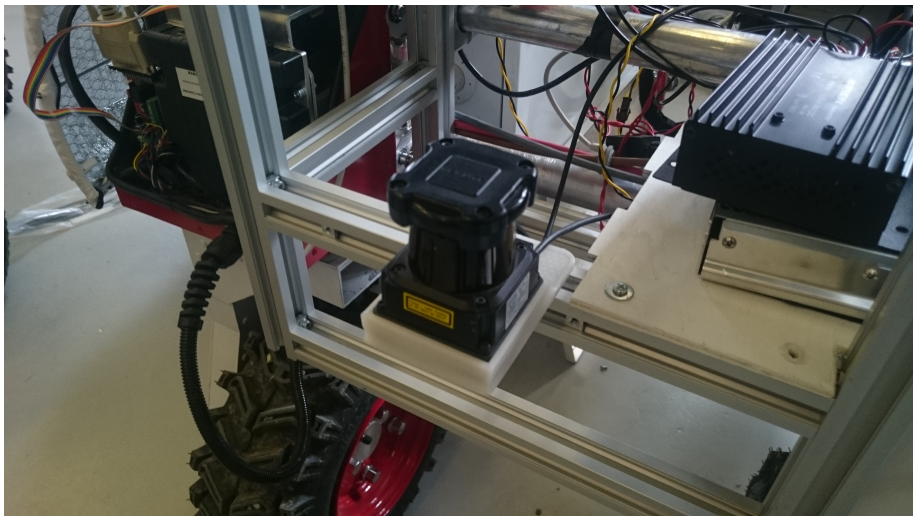


Figure 2.1: Hokuyo utm 30LX-ew mounted on Thorvald's sensor rack.

2.2.2 IR Cameras

Infrared radiation is electromagnetic radiation with wavelengths between about 740 nanometers and 30 centimeters [42]. This puts it between the visible light and microwave portion of the electromagnetic spectrum [17], as shown in figure 2.2. These waves are invisible to human eyes, but transfers heat between objects.

Infrared cameras can, in contrast to the human eye, detect infrared radiation, and convert this into an electric signal [67]. These signals are then processed to produce an image on a video monitor. The images show the heat distribution of the motives [17]. The ability to differentiate objects by temperature makes this technology useful in regards to recognizing living, warm-blooded obstacles for an agricultural robot working in a field.

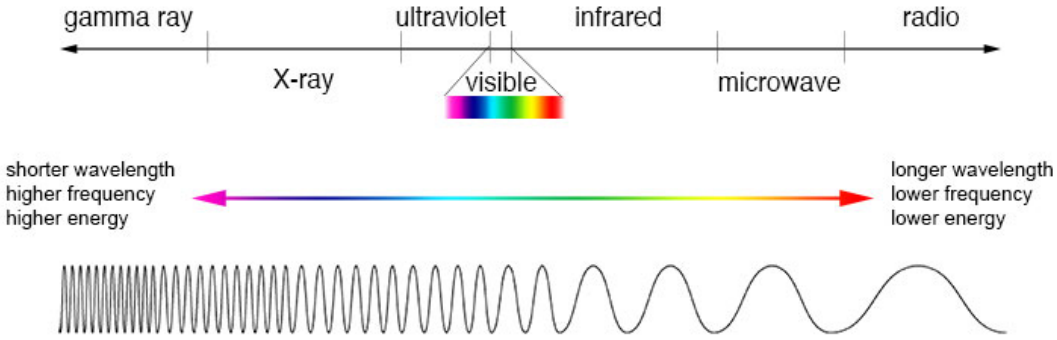


Figure 2.2: Showing the connection between wavelength, frequency and energy for the electromagnetic spectrum [22].

Being able to record objects by heat, independent of visual light, enables thermographic cameras to work just as well in the dark as in broad daylight. Furthermore, IR cameras are also able to see through light fog and rain. Through denser fog, however, there is a negligible difference between what an IR camera and the naked eye can see [66].

When it comes to price, IR cameras come in many categories, where resolution is an important aspect. As an example of a more high-end camera, the FLIR E60 handheld camera costs $7999.99 [24]. An example of a cheaper model can be the FLIR C2, which can be bought for $499.99 [5].

2.2.3 Safety Edges

Safety edges, also called pressure-sensitive edges, are used in various industries to protect both personnel and equipment from crushing accidents. They appear as long, slim rubber bumpers placed at surfaces that are somewhat likely to collide with their environment. Safety edges are found on automatic garage doors and as collision sensors on mobile robots, among other places [11].

The edge is an electrical switching device consisting of a compliant housing, such as a rubber profile, and an internal ribbon switch. This is shown in Figure 2.3. When a force is applied to the housing, it causes contact closure in the ribbon switch. At such contact, the robot will be programmed to stop, preventing further damage.

The characteristics of safety edges are defined by several properties [11]:

- **Profile:** The cross section of the edge can vary in for instance height, width and mechanical elements, which will affect the actuation characteristics.

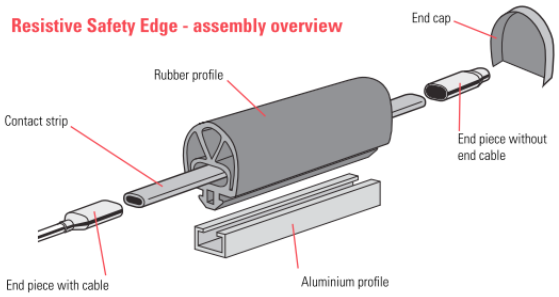


Figure 2.3: Showing the different parts that make a safety edge [14].

- **Actuation Force:** Necessary force to cause a contact closure, using an object of specific size and shape.
- **Compliance:** Amount of deformation before the switch actuates. When immediate actuation is required, low compliance is desirable, while higher compliance reduces probability of false actuations.

When fastened to an autonomous mobile robot, a pressure-sensitive edge is not a collision avoidance tool, but rather a collision limitation tool, to be used if all other collision avoidance systems fail. Furthermore, an obstacle will have to be in the same height as the safety edge to actuate the switch. This is not a problem facing walls or stems, but is potentially problematic facing more irregularly shaped obstacles, e.g. animals lying down, branches or rocks on the ground. Safety edges operate independently of rain, fog and light conditions. However, temperature can impact the material in the compliant housing, changing the characteristics of the edge. There are many different kinds of safety edges. The prices vary somewhat, with most of the models ranging from $200 to $350 per edge [15].

2.2.4 Sonar

Sonar is short for *Sound Navigation and Ranging* [20]. As the name suggests, it is a device that uses sound waves as input when determining what its surroundings look like. Sonars are widely used in underwater applications, because sound waves can travel farther in water than both radar and light waves [20], but are also applicable on land. Sonars can be divided into two groups, active and passive.

Active Sonar

The active sonar technology has certain similarities to the laser scanner. Instead of light beams, sounds waves are being emitted from the sonar. When the sound

waves hit an object, some of the waves will echo back towards the sonar. As with the laser scanner, time of flight, i.e. the time until the sound's echo returns, is used to measure distances from sensor to objects. The sound waves will spread like a fan from the sound's point of origin, as shown in Figure 2.4.

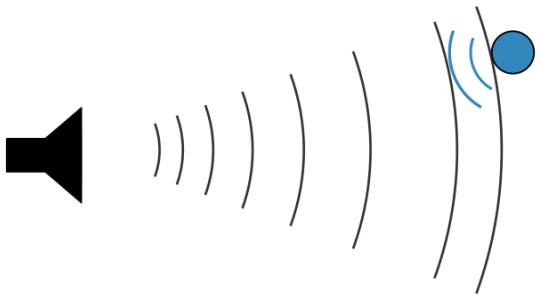


Figure 2.4: Sound waves spreading with distance

This means that a sonar scan will cover a much larger area than a laser scan. However, this also makes it harder to determine exactly where an echo is coming from, making maps based on sonar scans much less accurate than those created from a laser scan [41]. Acoustic wavelengths are typically quite long, around 4 mm. Therefore, most surfaces will have the function of an acoustic mirror. This entails that surfaces that are not orthogonal to the moving direction of the sound waves will reflect the energy away from the sound source, thereby not being detectable for the scanner, and not appearing on a map [41]. This way, maps created by sonars alone must be expected to differ substantially from the actual surroundings. However, sonars might be useful for navigation if used in combination with an actual map of the surroundings. The map can help predicting the data a sonar sensor will produce from a given position.

Passive Sonar

Unlike the active sonar, the passive sonar does not emit its own sound waves, but rather detects the sound waves coming towards it. This is useful in applications where not being discovered is key. Military vessels use passive sonars to locate submarines or ships without being discovered. Measuring distance with a single passive sonar is not possible, as time-of-flight calculations are impossible without knowing when the sound was made. With multiple passive sonar devices however, it is possible to calculate the position of a sound source by triangulation [20].

2.2.5 Radar

Radio Detecting and Ranging, or radar, is based on the use of radio waves, as indicated by the name [45]. Radio waves are the electromagnetic waves with the longest wavelength, see Figure 2.2. The development of the radar technology was driven by the threat of air attack during World War II. Today, radars help ships and airplanes navigate in fog and heavy precipitation weather, detects speeding cars and detects several sorts of atmospheric phenomena [45].

When it comes to the way they work, radars are similar to lasers and sonars: Radio wave pulses are sent out from a transmitter, which makes the radar an active sensor. The waves may be reflected by objects in a pulse's path, partially reflecting the pulse back to the radar. As with lasers and sonars, the distance to an object can then easily be calculated by time of flight, using Equation 2.1, as radio waves travel with the speed of light. Furthermore, the Doppler effect can be used to determine if an object is moving towards or away from the sensor. This principle is used in traffic law enforcements.

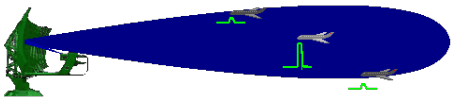


Figure 2.5: Echo signal strength varies as radar wave beam moves across target. [64]

Most radar systems have an antenna that will focus the energy into a one-

directional beam, as shown in Figure 2.5. The amplitude of the echo depends on at what point in the beam's cross-section the object is located. As radar antennas usually rotate constantly, an object's precise position can be determined by observing at what point it creates the strongest reflected signal.

Radars are not dependent on light to function, and works just as well at night. They are also able to see through fog, snow and heavy rain [54].

2.2.6 Cameras and Optical Flow

An input stream from a regular camera can be used for obstacle avoidance purposes. One way of resolving the problem of determining the distance to an object with the use of a video stream, is a method called optical flow [58].

Each pixel in an image corresponds to the projection of an object in the 3-D space onto a 2-D image plane. If the object or the camera should move, this would also change the object's projection onto the 2-D plane. The core of the problem is to determine where the pixels in an image at time t are found in the image at the next time step, $t+1$. By finding the movement for multiple pixels in an image, a web of flow vectors can be created, showing the relative motion between the camera and the objects in the environment.

Using the divergence of an object from the focus of expansion from one frame to another, it is possible to estimate the time to contact [58]. This requires that the object, and or camera, moves with a constant speed.

Using cameras as sensors has certain pros and cons: Cameras have superior resolution compared to most other sensors, and the ability to differentiate between colors. This makes it, at least in theory, possible to understand the environment more precisely than with for example LIDARS or sonars. Possible applications include seeing traffic lights and reading signs. They are dependent on ambient light to work, which means that separate lighting is needed if working in the dark. Camera vision will be limited by weather phenomena, which potentially makes cameras useless in weather conditions like heavy rain, fog, snow and direct sunlight. Furthermore, using camera as a sensor in a robotic system will take up a considerable amount of CPU [51].

2.2.7 Stereo Cameras

Most animals use stereo-vision to calculate distances to surroundings objects. In a stereo camera this function is being emulated by getting input from two parallel cameras. An object infinitely far away from the cameras will appear in the same corresponding pixel in the two cameras, while a closer object will have different pixel placements. The nearer the object is, the bigger displacement it will have between

the two pictures. An invert relationship between this displacement and the distance to an object seems to be a good approximation [40].

$$D = \frac{C}{\delta} \tag{2.2}$$

where D is the distance to the object, C is a constant proportional to the distance between the camera centers, and δ is the displacement between the two images.

Stereo cameras are passive, which means that they only use ambient lighting. Therefore, they cannot operate outdoors at night, unless provided with an external light source, such as a flashlight.

As a price example, the *ZED Stereo Camera* costs $449.00 [21]. However, cheaper models exist, as the *Tara Stereo Vision Camera*, which has a price tag of $249.00 [16].

2.3 Sensor Performance and Reliability

An important point to keep in mind, is that a sensor is not simply a sensor. There will be big differences in performance and reliability from model to model of the same sensor type. As producers rarely will admit to having produced an inferior product, it is difficult to say which products are less reliable without testing several. As a rule of thumb, however, costlier models will be more reliable than cheaper models. In a safety system, being able to trust the output produced by the attached sensors is essential. Therefore, some testing of different models will be required before any final decisions are made.

Chapter 3

Software and Computer Systems

In this chapter the software framework of Thorvald is discussed. We also look briefly at two other computer systems that are of relevance for safety robotic safety systems: the microcontroller and programmable logic controller.

3.1 Robot Operating System (ROS)

Thorvald uses Linux Ubuntu as its operating system. The framework used is ROS (Robot Operating System). ROS is a collection of tools, libraries and conventions that combined make a flexible framework for writing robotic software [30]. Although experimental versions exist for other platforms, most developers use Linux Ubuntu, which fully supports ROS [29].

ROS' aim is to make the task of creating complex and robust robot behavior simpler. Since its start, the ROS core has been licensed under the 3-clause BSD license, which is a very permissive open license. The contributors to ROS are however many, and other licenses are commonly used in the community packages [1].

By supporting code reuse, and collaborative robotics software development, ROS provides a general framework that can be used in a wide range of robotic software. ROS is applicable for various robotic systems, including manipulators, mobile robots, autonomous cars to name a few [12].

Figure 3.1: The ROS logo [1].

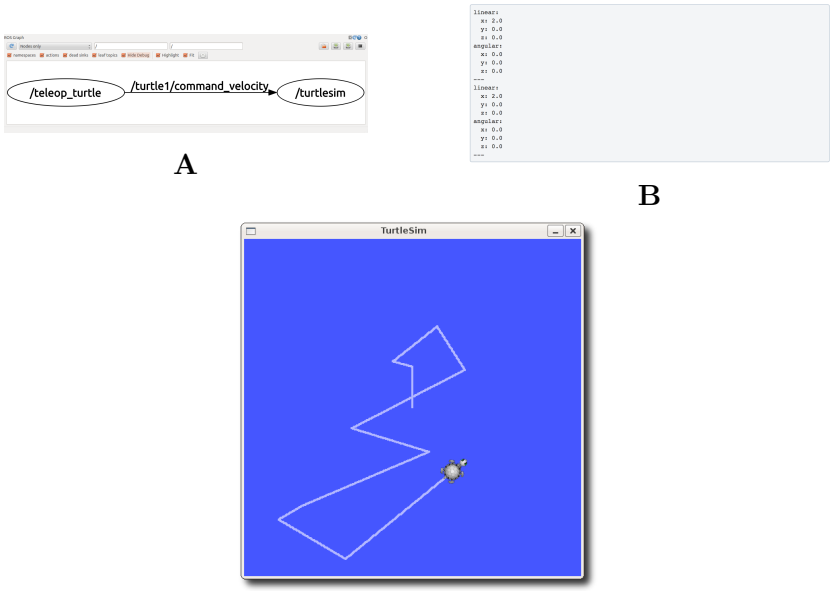


Figure 3.2: **A**: The node turtlesim subscribes to the message command_velocity, which the node teleop_turtle publishes. **B**: Print of a ROS message. **C**: Visualization of the turtlesim node [12].

3.1.1 Fundamentals of ROS

The computation processes working in a ROS-based system are called *nodes*. These nodes have different tasks, and together make up a control system. For instance, one node can be in control over a motor, another manages path planning and navigation, while a third node can be in control over a laser scanner. These nodes can be coded in a wide variety of programming languages, as long as a ROS client library exists, but C++ and Python are considered to be main languages [29]. There are many benefits associated with a node-based system: Crashes will be isolated to individual nodes, and the code will be less complex compared to a monolithic system [12].

All nodes will have a name that uniquely identifies them with the rest of the system. To communicate with each other, nodes subscribe or publish *messages* via different *topics*. Each topic carries messages of one specific data type, which makes it possible to separate the production of data from the consumption of data. In other words: a node is not aware of other nodes' existence. A node can publish and/or subscribe to several topics, and it is possible for each topic to have several publishing/subscribing nodes. A visualization of the communication process is shown in Figure 3.2.

This publisher/subscriber model of the ROS topics is not suitable when re-

quest/reply interactions are wanted. For this kind of communication a *service* is used. A service is defined by one request message and one reply message, and allows nodes to send a request and receive a reply.

Powerful visualization and simulation tools are included in the ROS framework. This can potentially save costs and increase safety, as simulations can be run before testing the programs on an actual robot. Creating a visual representation of what different sensors see is also possible, and makes it easier for humans to analyze sensor data.

3.1.2 Reasons for Choosing ROS

There are a few different reasons for why ROS was chosen as the framework for Thorvald. The main ones are the big community of contributing users, and ROS' versatility. Because of the large user group, some robots and sensors are delivered with ROS support.

There are already developed a large number of packages that can control anything from a robot's motor to vision systems. Most of these packages are available for re-use under permissive open-source licenses. Using already existing packages instead of creating new ones, can be a substantial time-saver. Coding in packages makes ROS module-based, and allows users to choose only the packages needed for their use. In addition, ROS' online beginner tutorial is comprehensive, whereas packages will often come with its own tutorial. This significantly simplifies operations and shortens the time spent on making things work [29].

3.1.3 Navigating with ROS

Maps

2D grids are used to represent navigation maps in ROS. Each cell in this grid will have a value, which corresponds to the likelihood of that cell to be occupied by some object. These maps can be learned directly from sensor data. Map files are stored as a grayscale image, in some commonly supported format, as PNG or JPG. Each map has an associated YAML file, which contains some additional information, such as resolution, where the origin is placed, and thresholds for deciding the occupancy of a cell. There are three classifications of occupancy status: unoccupied, occupied, and unknown. Occupied cells will appear as black on a map, unoccupied as white, and unknown as gray. Image files are easily editable, which makes it easy to clean up a map created from sensor data. This enables removing lines that shouldn't be there, and add fake obstacles to prevent path planning through certain areas of the map [48].

ROS Navigation Stack

A robot running on ROS will be able to move around on its own, interacting with the ROS navigation stack, often called the *nav stack*. This system is complex, and this section will merely provide an overview of how the nav stack works.

In short, the system will allow a robot that runs on ROS to move efficiently to a specified goal position without hitting anything along the way. This is done by integrating information from the map, localization system, sensors and odometry[1]. These inputs are used to calculate a good path from the current position to the goal, which the robot then follows as accurately as possible. This should enable the robot to reroute and recover if unmapped obstacles are encountered.

The nav stack works like this [48]:

1. The nav stack receives a *navigation goal*. This goal specifies the desired position and orientation in some coordinate frame.

2. The *global planner* plans the shortest path to the goal, using a map and a path-planning algorithm.

3. The *local planner* receives this path, and tries to drive the robot along it. In addition, the local planner will use information from sensors to avoid encountered obstacles that are not found in the map.

4. The action terminates when the robot gets close to the goal pose.

As a first step, the nav stack will create a *global costmap*. The global costmap is a data structure that evaluates to which extent a place in the map is a good or bad place for the robot to be located. Colliding with a wall would be bad, and therefore has a high cost. Being in open space is good, resulting in a low cost, while being close to a wall is somewhere in between. When calculating a global path, the nav stack will keep the path in places with low cost.

That process gives the robot a path it will aim to follow. However, the local planner will have to adjust this global path to avoid local obstacles that is detected by one or more of the robot's sensors as it drives around. *Costmap_2d* is a ROS package that builds an occupancy grid based on sensor input and the static costmap [4]. The costmap will automatically subscribe to sensor topics and update itself. A sensor can be used to add obstacle information to the costmap (called marking), remove it (called clearing), or both.

A process called *inflation* is used for calculating the cost values around occupied cells. These values will decrease with distance to the obstacle, and are categorized into 5 groups:

[1]Odometry: Use of data acquired from motion sensors to estimate change in position.

– *Lethal* cost: There is an obstacle in a cell. The robot will obviously be in collision if it's center is in such a cell.

– *Inscribed* cost: A cell lies closer than the robot's inscribed radius from an obstacle. This means that if the robot's center is in a cell that is at or above the inscribed cost, it will definitely be in collision. The inscribed and circumscribed areas of a robot are shown at the bottom of Figure 3.3.

– *Possibly circumscribed* cost: Uses robot's circumscribed radius as cutoff distance. Whether this results in a crash or not is dependent on the orientation of the robot. The wording *possibly* is used because this cost also can be a result of a user preference, not only an actual obstacle. Users can insert costs independently of obstacles, to make the robot avoid certain areas.

– *Freespace* cost: There is no reason for a robot to avoid a free space, and the costs of free spaces are assumed to be zero.

– *Unknown* cost: When the map has no information about a cell, the users of a costmap can interpret that as they like.

– All other costs are assigned values ranging between *Freespace* and *Possibly circumscribed*. These costs depend on the distance to the nearest *lethal* cell, and a decay function of the user's liking.

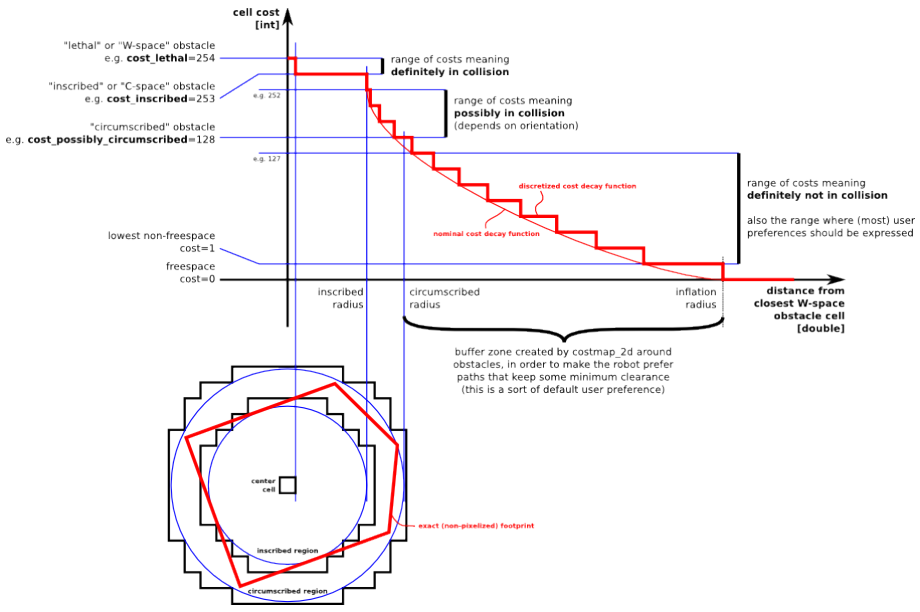


Figure 3.3: A cell's cost shown as a function of distance to closest cell containing an obstacle [4].

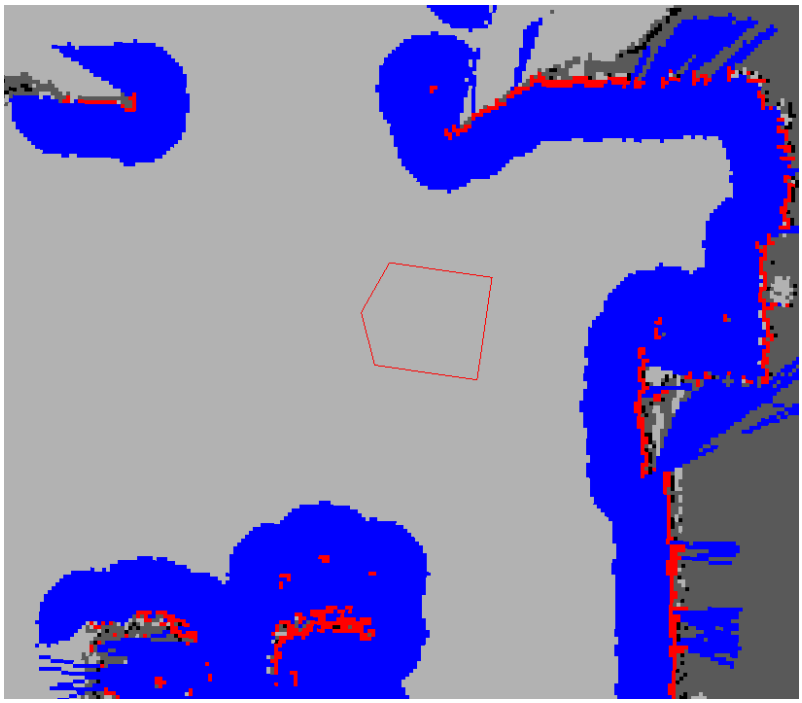


Figure 3.4: Costmap: Red cells represent obstacles, while blue cells represent the obstacles inflated by the robot's inscribed radius. Red line figure represents the robot's footprint. To avoid collision, the robot's footprint should never intersect a red cell, and the robot's center point should never be in a blue cell. [4]

3.1.4 Open Source and Reliability

Using open source software is convenient, and a cost-effective way to get a robot up and running. However, this also means that the software provided does not meet all industrial standards. As Thorvald aims to be an affordable agricultural robot, having industrial standard pieces at all levels would result in the robot becoming too expensive. One possible effect of using this open source software is ROS systems crashing or failing more rapidly than more expensive, industrial frameworks. This, in turn, implies that a safety system based solely on ROS might not be reliable enough to be the only means of safety implemented. Therefore, implementing an additional layer of safety that meets industrial standards is necessary.

3.2 Microcontroller

A microcontroller is essentially a small computer. As opposed to desktop computers, that can run thousands of different programs, microcontrollers do one thing well [28]. Microcontrollers will match a number of the following characteristics:

- As with all computers, they have a CPU (Central Processing Unit) for executing programs.
- They have some random-access memory (RAM) where variables are stored.
- Microcontrollers are often embedded inside another device, with the purpose of controlling that product's actions or features.
- Dedication to one task and running one specific program.
- Low power consumption.
- It has a dedicated input device, and often a LED or LCD for displaying output.
- Small size and low cost.

The applications of a microcontroller are many, completing tasks with various degrees of complexity. Some of the simpler ones can be found operating in everyday convenience items, such as refrigerators, toasters and clock radios, facilitating the operation of their electromechanical system [10]. For these applications, the CPU demand will be fairly modest. More sophisticated microcontrollers can be responsible for crucial functions in for instance aircrafts, different kinds of robots, and ocean-going vessels.

Important features of a microcontroller are the input and output ports. The input ports take sensor data, while the output ports are used to give commands to external hardware, such as the motors of a mobile robot. The most common is to have analog input ports, and digital output ports [9].

There is no standard when it comes to programming microcontrollers. Which languages and compilers that are used varies largely between brands.

3.3 Programmable Logic Controller (PLC)

PLCs are basically microcontrollers designed for industrial applications, like controlling machinery or processes [37]. They are computer control systems that control one or more parameters of an output device based on data acquired from a continuously monitored input device [19].

Programming a PLC is done on a computer with dedicated software from the manufacturer. Ladder logic is the most commonly used form of programming. Here, symbols are used instead of words for programming, with lines connecting these symbols to indicate the current flow through a coil or contact [46]. Troubleshooting will often be a simpler task than with a microcontroller, as having a display unit is common [37]. When it comes to reliability, PLCs are designed to handle, and tested in, harsh condition, such as dust, humidity or heat. Which means they are reliable to a whole other level than the average microcontroller.

Chapter 4

Other Robotic Safety Systems

Actuators and industrial robots have been working side by side with human workers in factories for decades. The first industrial robot was invented by George Charles Devol in 1954. In the years following, robot development took off, and by the mid 1970's, several robotic arms doing various tasks were on the market [62].

Making safety systems for industrial robots can be done by having areas in the robot's immediate proximity where people are not allowed to be when the robot operates. Laying a safety mat on the floor around the robot, stopping the robot if stepped on, is a rather easy step that would keep the workers safe from robot-inflicted damage.

Designing safety systems for mobile robots is a different story. In contrast to the industrial robots, where the humans were stepping into the robots' space, the mobile robots are now stepping into the humans' space. This means that the main assumption that people shouldn't be in a robot's working area is no longer valid, and the robot must be programmed accordingly. Humans aside, a mobile robot's surroundings will continuously change. Therefore, a mobile robot must be able to sense its surroundings at all times in order to make good decisions regarding movement and actions.

This is a challenge that various robotic communities have been working on in recent years. Several mobile robotic platforms are already out in the real world doing a job, while others are still under development. There are many different kinds of mobile robots, and several producers of each kind. This chapter describes some of the robotic platforms that exists, and gives examples of how their safety systems are built.

4.1 Autonomous cars

The development of self-driving cars has taken place for some years already. Car companies, such as Tesla, GM and Ford, and technology companies, like Google, are all developing their own system for autonomous cars [43, 60]. As cars operate in environments where interaction with humans and other vehicles is a nearly constant state, the demands for a working safety system are strict. They must be able to navigate through cities and countryside, reading signs, following the road, without hitting any the other road users or causing dangerous situations.

Figure 4.1: "Junior", a self-driving car developed by the Volkswagen Automotive Innovation Lab (VAIL) driving around in a parking lot, 2009 [35].

4.1.1 Tesla's Autopilot

One of the companies that have come the farthest in developing autonomous cars, is Tesla. All Teslas being produced now are equipped with necessary hardware for full self-driving capability which Tesla claims have "a safety level substantially greater than that of a human driver" [60]. This means that the vehicles have the sensors and computer power required to make a system capable of driving people around without any human interaction at all. The current autopilot does, however, require a driver to be ready to take control over the vehicle if needed.

The set of sensors Teslas are being equipped with, includes cameras, ultrasonic sensors and a forward-facing radar.

Cameras

The cars each have a set of eight cameras, even though only one of them is being used in Tesla's first version of the autopilot [39]. These eight cameras provide a 360° vision of the car's surroundings at up to 250m distance to the car [59].

First of all, three cameras are placed behind the windshield:

- **Wide**: A 120° of vision fisheye lens that sees traffic lights, obstacles at close range and objects cutting into the car's travel path.
- **Main**: Covers a broad range of cases. The only one currently in use.
- **Narrow**: Sees distant features, which is particularly useful at high speeds.

Secondly, two side cameras will be looking straight outwards for cars that unexpectedly enters your lane on the highway, in addition to providing additional safety in intersections where the visibility is limited. Two cameras will be monitoring the rear blind spots of the car, to make file changes and merging safe. The last camera is pointing straight backwards from the rear end of the car. This rear end camera is also particularly useful for complex parking maneuvers.

Furthermore, a rather sophisticated image processing system is integrated. This system recognizes the shape of humans, reads road signs and traffic lights, and sees the road surface markings, to name a few properties [60].

Radar

The autopilot system uses a forward-pointing radar that passes through fog, dust, rain, snow and the car ahead. This will help detecting objects ahead of the car. The radar was initially only meant to be a supplement to the primary camera and image processing [57], but has later been given a more central role in terms of obstacle sensing.

Ultrasonic sensors

Lastly, Tesla cars are being equipped with ultrasonic sensors (sonars) for proximity sensing. These are placed so that they cover all 360 degrees around the car, with several overlapping areas between the sensors. These overlaps make it possible to more accurately estimate the position of nearby objects, based on whether several sensors receive reflections from the same object. The sonars are used to detect nearby cars and provide parking guidance.

Fleet learning

A system Tesla calls fleet learning plays a major role in their autopilot system. Tesla collects data from their entire fleet, regarding how drivers behave in different locations, for later to use that information for decision making. As an example, a system based on radar feedback can be "tricked" into thinking that a bridge with a dip under it is an obstacle, thereby creating a *false positive*. To solve this problem,

Tesla cars will take note of how human drivers behave at such places, e.g if they brake or not. This information is then uploaded to Tesla's database, and locations where several drivers drive past such an obstacle without braking are added to a "geocoded whitelist" of objects [57].

4.1.2 Other Autonomous Cars

It is worth noting that Tesla's choice of using a radar as a primary obstacle detection sensor is not common among companies developing self-driving cars. Most major players in the autonomous car market use LIDARs as their primary sensor for obstacle detection. These include Google, Uber and Ford, to name a few [44, 33, 63]. This one-sided focus from the rest of the industry might imply that radar might not be the best choice of sensor for obstacle detection.

4.2 Tractors

Self-driving tractors are being developed by several different companies and research groups around the world. Harper Adams University has an autonomous tractor, called Pomona [61]. The development of the tractor was part of a project entitled *USability of Environmentally sound and Reliable techniques in Precision Agriculture*, or USER-PA for short. This project brought together scientists from the United Kingdom, Germany, Turkey, Switzerland, Italy, Israel and Denmark.

4.2.1 The Seven Layers of Safety

The project decided that the tractor needed what is described as *seven layers of safety* [38]. The layers are the following, ordered in distance from the vehicle:

1. **Route Planning**: For obstacle avoidance
2. **Laser Rangefinder**: For obstacle avoidance.
3. **Proximity Detection Ring**
4. **Deadman's Handle**
5. **Deformable Impact Bumper**
6. **Emergency Stop Buttons**
7. **Small/Slow/Lightweight Design**

Emergency stop buttons, a front bumper and a dead man's handle is what sums up to the physical contact part of the safety system. The proximity detection ring consists of a series of diffuse ultrasonic sensors mounted, one sensor every 8 degrees around the whole circumference of the tractor. The accuracy of the diffuse sensors is 1 mm, which allows for having different zones around the vehicle. It will slow down if an obstacle is detected in the "orange" zone, and stop completely if the obstacles moves into the "red" zone. The laser rangefinder is used to observe persons or obstacles within about 80m from the vehicle. These observations are compared to an already generated map of the surrounding, making it possible to determine if the obstacle was there before, or not. Such obstacles till be avoided by rerouting if possible. If not, the tractor stops.

Furthermore, a manual restart of the tractor will be required should the safety system stop the vehicle [38].

4.3 Hospital Robots

Automated guided vehicles (AGVs) are found in numerous hospitals around the world, doing transportation tasks. These include moving laundry, food, and delivering supplies to the different departments. For navigation, the hospital floors often have a web of magnetic tape or spots on the floor for the AGVs to follow [55].

An example of such a vehicle is the *Swisslog TransCar AGV*. As these AGVs operate in areas where encountering people is common, there must be a safety system in place, to avoid collisions. For safety, the Swisslog is equipped with [56]:

- **Laser scanners**: For slowing down the vehicle if it encounters an obstacle.
- **Ultrasonic sensors**: Listens for obstacles above the laser scanner, for additional safety.
- **Safety edge**: Vehicle stops immediately when in contact with an obstacle.
- **Emergency buttons**: Brigs the vehicle to an immediate stop.

4.4 Lawn Mowers

Robotic lawn mowers are becoming a popular appliance. An example of such a mower is the Bosch Indego [3]. When it comes to safety and navigation, robotic lawn mowers are rather simple robots. For navigation, a wire needs to be placed around the perimeter of the lawn that is to be mowed. The robot will then only operate within the area defined by this wire.

For safety when driving, the Bosch lawn mower relies on the following system [3]:

- **An infrared sensor**: Sensing if there is an obstacle in the way, replanning the driving route if so.
- **A pressure sensitive bumper**: Installed in the front of the mower. This allows it change direction and drive any obstacle it might encounter.
- **A manual stop button**
- **A pressure-sensitive sensor**: Senses if the mower is being lifted from the ground, or flipped upside-down, stopping the blades.

A robot mower is light and slow, which is why bumping of obstacles as a safety measure is considered a satisfactory one.

4.5 Warehouse Robots

Warehouses around the world are beginning to use AGVs and mobile robots for the moving of inventory. The greatest example of this is Amazon, which has over 45 000 of its mobile robot, Kiva, driving around in its fulfillment centers [52].

The inventory is placed on mobile racks around the center. When getting an order, a robot drives to the rack that contains the right product. When the Kiva reaches the bottom of the rack, it lifts the rack of the ground and drives the whole thing to a human picker.

The safety system of Amazon's robots is largely based on IR sensors pointing in all directions, sensing if other objects are close. A robot approaching an obstacle stops until the obstacle is gone, at which point it drives on [34].

Chapter 5

System Requirements

To be able to design an effective safety system for Thorvald, we need to define exactly what such a system is supposed to achieve. Here, we define a set of system requirements:

1. **Thorvald shall not kill**: First and foremost, it is essential that the robot does not kill humans. This is absolutely crucial, and a requirement that must be met.

2. **Thorvald shall not injure**: Furthermore, the robot must not injure people, animals, itself or other material things. This means that it must be able to stop prior to crashing into any obstacle it may encounter. Therefore, the robot must be able to sense its environment efficiently, and act accordingly to what it senses.

3. **Three-dimensional sensing**: Obstacles can occur at different heights, which means that one cannot be certain to discover all obstacles in the robot's path by only scanning a horizontal plane at a certain height. Animals can be lying on the ground, or objects can hang down from a ceiling. Therefore, the system requires a sensor that sees the environment in 3D.

4. **Backup computer system**: Computer systems can fail or crash. For that reason, at least two separate layers of safety needs be implemented, so that a backup exists, should the main layer fail. The robot must therefore be equipped with a safety sensor that does not transfer data to ROS, but rather to a separate controller.

5. **Navigation**: Safety, and more specifically obstacle avoidance, is intertwined with navigation. Knowing its own position related to static and dynamic obstacles in the environment enables the robot to find an efficient path to its destination, without crashing.

6. **Possibility for human overriding**: Should the robot for some reason keep on driving after having hit an obstacle, and thereby proving the entire safety system faulty, humans must have an on-site way to override the system and immediately bring the robot to a stop.

7. **Transferability**: Thorvald is a module-based robot, and it is important that the safety system works on all of the platforms.

Furthermore, it is possible for motors to fail. Thorvald will mainly operate on relatively flat surfaces, and the tires and gear makes for quite a bit of moment of inertia. Therefore, it is for such instances assumed that the robot will not roll fast or for a long distance, and does not require any additional safety measures.

Chapter 6

System Description

With the system requirements from Chapter 5 in mind, we propose a system with four separate layers of safety. These are listed below, ordered by distance to obstacle. The reasoning for choosing this particular system, and a more detailed description of each layer, follow on the coming pages.

1. **Navigation**
   - **Path Planning:** Global path planning with ROS Navigation Stack.
   - **Local Path Planning:** Rerouting if obstacles are sensed in the robot's path.
2. **Low Level Control System**: Sensor giving input to a dedicated, real-time computer system that does not run ROS. Sensing if an obstacle gets closer than a set limit to the robot's front. In which case the robot is slowed down and stopped.
3. **Safety Edge:** Placing safety edges on the front and sides of the robot, stopping it if it hits an undetected obstacle.
4. **Emergency Stop Buttons:** Placing four emergency stop buttons on the robot will give humans an override possibility, and add a layer of safety.

Layer 1 is related to the navigational system of the robot, sensing the environment up to almost 100 meters from the robot. Layer 2 is meant to stop the robot before collision, if 1 should fail. In the case of both 1 and 2 failing, and a crash occurs, point 3 will stop the robot immediately. Should all sensor-based means fail, point 4 gives humans the possibility to override the system and shut the robot down.

The robot can turn around its own axis. Therefore, it is assumed that it will never drive backwards, even though it is able to do so. This means that proximity

sensors will not need to cover all 360 degrees around the robot, but can focus on what is in front of and to the sides of the robot.

6.1 Layer 1: Path Planning and Navigation

As stated in system requirement 5, navigation and safety are connected. The navigational layer of the safety system will consist of two parts: a global and a local path planner.

6.1.1 Global Path Planning

The bottom layer of the safety system is the global path planning. Having a good plan for the driving path, that avoids known, and static obstacles, makes for an important contribution to safe autonomous driving. The ROS navigation stack will be used to make a global costmap of the robot's environment. This is done by equipping the robot with a 3D laser scanner, such as the Velodyne VLP-16. This is a rather costly scanner, at $ 7,999/piece, but can be exchanged with a cheaper model in case of commercial production. The Velodyne is shown in Figure 6.1. Using a better, and thereby more expensive, scanner in the development and testing stages, will nearly eliminate scanner malfunction as a potential cause of whatever problem might arise. This makes for more smooth system development. The ROS-compatible scanner continuously feeds the nav stack with information about the robot's environment. ROS uses this information to determine the robot's position on a map of the surroundings, while simultaneously using input from the scanner to add scanned objects to the same map. Making a map will only need to done once per area, as an already generated map can be loaded for later use.

As described in Section 3.1.3, adding a global costmap to this map is done by *inflation*. By giving ROS values for parameters such as circumscribed and inscribed radii, and the wanted margin of clearance, a global costmap will be created. This will show where the robot can and cannot drive. From this, the robot will be able to calculate the best global path from one point to another. An example of a costmap is shown in Figure 6.2.

Moreover, in some places it is essential that the robot keeps to a specific path. An example of such a place is a strawberry field, where the robot must keep between the same two rows of plants. For such instances, "walls" with *lethal cost* can be inserted to the costmap by a user, thus preventing the robot from crossing a row.

Using a 3D-sensor, such as a 3D-LIDAR is important to be able to map obstacles at different heights. When all the static obstacles in an area are properly mapped, the nav stack can calculate global paths for the robot to follow when driving in that area.

6.1.2 Local Path Planning

The robot's environment changes with time. People, animals, or even other robots can move into Thorvald's planned global path. In such cases, rerouting around the obstacles will be much more efficient than coming to a full stop. This way, continuous operation can be achieved even when some unforeseen obstacle is encountered. Therefore, it is essential to get information about what is in front of the robot.

As described in Section 3.1, the ROS nav stack has packages for local, online path planning. The main input for the nav stack will be provided by a 3D laser scanner. A LIDAR is chosen for a variety of reasons: The output a LIDAR provides is very accurate: distance and angle to surrounding objects, regardless of material, as long as it is not transparent. Many LIDAR models come with ROS support, which means that it will be fairly easy to set them up and get a logical output out of the information that it provides. Using images from different kinds of cameras to spot and determine distance to various obstacles would require sophisticated image analysis software, in addition to quite a lot of computing power. Both normal and stereo vision cameras are also susceptible to direct sunlight. LIDARs can work at night without a light source, in contrast to cameras. Sonars would give very noisy output, which would require a lot if interpretation. The amount of noise and uncertainty increases with distance to the scanned object, which makes sonars especially unfit for path planning, as a bigger scanned area is wanted. IR cameras have a big advantage of seeing objects with different temperatures, but comes up short at seeing obstacles with the same temperature as their surroundings. Determining distances to obstacles are also easier with a laser scanner than an IR camera.

Since a 2D-laser scanner will only scan the horizontal plane, and thereby be missing any obstacle that are placed at another height, as well as parts of obstacles leaning towards or away from the robot, a 3D scanner will be used for the local path planning. The sensor we have chosen, at least for development, is the Velodyne VLP-16, which has 16 separate rotating channels two degrees apart, and gives a field of view of 30 degrees; 15 degrees up and down off the horizontal plane. This spreading of beams enables the sensor to detect obstacles located at different heights. The Velodyne has a 100-meter range, and gives the ROS navigation system time and distance to calculate an alternate path. The Velodyne is shown in Figure 6.1. Since it is assumed that the robot will not be moving backwards, the sensor will not need to have clear rear-end vision. To ensure a good scanning range, the sensor must be placed a bit off the ground. It will therefore be placed 1 meter up, at the front of the sensor rack that is mounted on the front part of the robot. That means having some angles to the sides and backwards blocked by the sensor rack and/or working equipment.

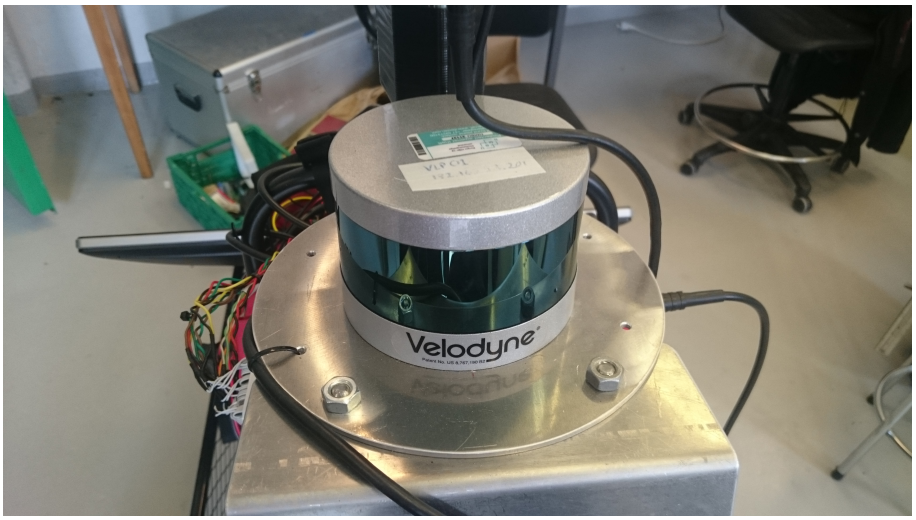


Figure 6.1: Velodyne VLP-16 3D Laser scanner.

The steepest beam from the Velodyne travels 15 degrees down from the horizontal plane. Trigonometry gives that the beam then will hit the ground after a flight of 3.9 meters, given the 1 meter height of the sensor. To be able to also sense what goes on in the immediate surroundings, and the other blind spots of the Velodyne, two 2D-LIDARs, such as the RPLidar A2, are placed on the robot, at the bottom of the corner modules in the front left and back right corners of the robot. The height is here around 40 cm. By scanning a field of close to 270 degrees each, we acquire an almost complete 360 ° scan of the immediate surroundings at 40 cm height.

The sensors give input to the ROS nav stack, which adds sensed obstacles to the local costmap. If an obstacle is detected in a spot that interferes with the global path plan, either by being in or close to the desired path, the plan is changed if possible. The robot will come to a stop if rerouting is found impossible by the path planner. Since there is a chance that the obstacle is mobile, like an animal or a human, the robot will continue to scan the environment. If the obstacle should go away, the robot can continue on its course.

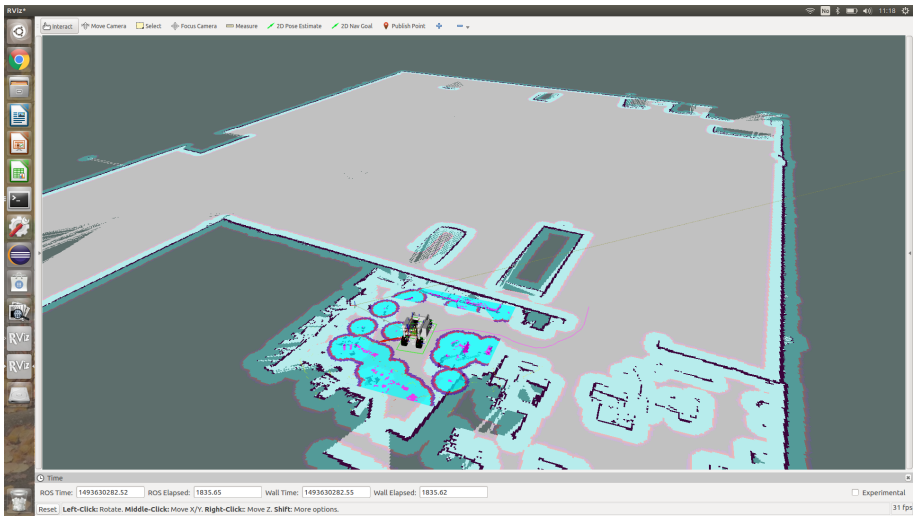


Figure 6.2: ROS nav stack costmap of the robotics lab parking lot at NMBU. The grey areas and black lines represent open space and obstacles, respectively. More shaded areas surrounding the black lines are the global costmap, while the sharper colors represent the current local costmap sensed by a LIDAR. The red, crooked line leading from the robot out to more open space is the path driven by Thorvald.

6.2 Layer 2: Low Level Computer System

As it is possible for computer systems to fail, and maybe even more so for open source-based systems, an independent rangefinder-based safety layer should be added, to comply with requirement 4. This will work as follows:

We mount a fourth sensor. This sensor scans the environment immediately in front of the robot. To ensure that this system works even if ROS crashes, this sensor is connected to a separate control device, a PLC, which is the industry standard for safety systems.

This layer is not concerned with navigation, but is simply there to make sure that the robot doesn't crash into anything if the navigation layer should fail. If the obstacle comes close enough, the robot should be forced to stop. However, to force-stop the robot is a rather dramatic step, and resetting it after an emergency stop is an extensive procedure, involving recalibration of the wheels, etc. Therefore, the robot should not be force-stopped unless it is absolutely necessary.

The stopping distance of a vehicle can be found by the equation

$$d = \frac{v^2}{2\mu g} \tag{6.1}$$

where v is the velocity of the vehicle, μ is the friction coefficient, and g is the

gravitational acceleration.

The braking distance of a Thorvald going at its maximum speed of 5 km/h will by Equation 6.1 be 16.4 cm, assuming a friction coefficient of 0.60, which is the value for tire on gravel [65]. However, Thorvald does not have brakes, and will thereby lose velocity due to moment of inertia and regeneration of energy. This will result in a somewhat longer stopping distance.

As stopping distance is a function of the velocity squared, halving Thorvald's speed to 2.5 km/h would make its braking distance drop to 4.1 cm. We conservatively add a safety factor of around 7 to account for the missing brakes, and the fact that the ground rarely is horizontal. This safety factor demands that a Thorvald, going at half speed, must be stopped if the system detects an obstacle around 30 cm ahead. As Thorvald will be working in narrow spaces, the system should watch only straight forward, with a scanning area limited by the robot's width.

We suggest the following system:

- **Zone 1:** Stopping the motors if an obstacle comes within 0.3 meters.

- **Zone 2:** Halving the maximum velocity when an obstacle is detected between 0.3 and 0.5 meters in front of the robot.

For this layer, a 2D laser scanner, The RPLidar A2, is chosen. The reasons for picking a LIDAR are mainly the same as they were in Section 6.1.2. As the program's task is rather simple - stop if chance of impact is imminent - a camera's high resolution and possibility to distinct between different kinds of objects are unnecessary. The simple point cloud input from a LIDAR should be enough to determine if there is an object right in front of the robot.

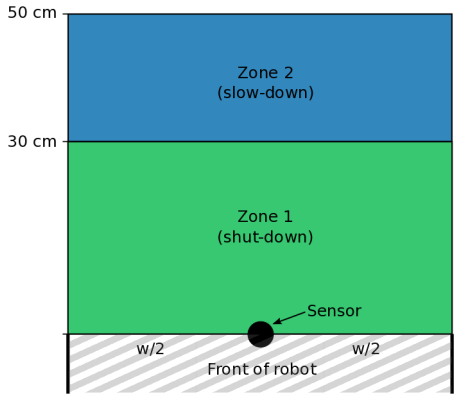


Figure 6.3: Showing the scanning zones of the second safety layer.

A 2D-scanner is chosen over a 3D-scanner because it is cheaper, and requires much less computational power. Furthermore, colliding with birds are assumed not to be an issue. Therefore, all living obstacles are assumed to be on the ground. This means that scanning a horizontal plane close to the ground will have very good chances of discovering most living creatures, as well as most non-living obstacles. With that in mind, the 2D-scanner is to be placed at the bottom of the sensor rack, at about 0.4 m height, at the middle of the robot's width.

6.3 Layer 3: Safety Edges

The by far most important requirement for the safety system is to make sure that the robot does not accidentally kill humans. The heaviest present Thorvald assembly has a mass of under 200 kg, and a modest top speed of 5 km/h. For those reasons, any crash that might occur is assumed to be non-lethal, and no extra "if-impact-"layer of safety is seen as necessary. However, for some applications, Thorvald will be equipped with heavy equipment or tools with moving sharp parts. For such applications it is crucial that the robot stops if it hits an obstacle.

To make sure that nobody is killed in such instances, if layers 1 and 2 should fail, pressure sensitive edges should be added to Thorvald's front. This will make sure that the robot stops immediately if it were to crash into something. These will need to have an actuation force so that the system kicks in if driving into a child, but does not kick in while driving through grain in the fields.

The obstacles most likely to be missed by the sensors, are those that lies on the ground. We suggest placing the edge on a bumper going across the front of the robot, this edge should be low enough to hit humans or animals lying on the ground, without hitting the ground where it is uneven. A height of 20 cm is suggested to achieve these requirements.

6.4 Layer 4: Emergency Stop Buttons

As it is possible for all the previous layers to fail, there is a possibility, however slim, for all of them to fail at the same time. Should that happen, and the robot continues to drive after hitting an obstacle, humans must be able to intervene. Therefore, emergency stop buttons are added to the robot. Installing emergency stop buttons is a simple and cheap operation. On the standard configuration, four buttons should be installed, one on each corner module of the robot. That way, there is at least one easily accessible button for anyone standing close to the robot. Other configurations will have different placements of the buttons, with easy accessibility as the main criterion. After an emergency stop button has been pushed, the robot will have to be manually reset.

Figure 6.4: Emergency stop button.

Chapter 7

Case Study: Laser Scanner

An experiment was conducted as a first step in implementing layer 2 of the safety system, which is described in Section 6.2. We used the sensor RPLidar A2, shown in Figure 7.1. Some technical specifications are shown in Table 7.1. The objective was to write a program that gives a response if the LIDAR senses an object in the zones that would call for a decrease in velocity or complete shutdown of the motors. The zones from the safety system were simplified to the following, due to computational simplicity: Zone 1 is a circle sector by a 140-degree arc and radii of 30 cm. Zone 2 is the continuation of the sector, with radii between 30 cm and 50 cm. See Figure 7.2 for illustration of the zones. As a substitute for motor control, LED lights should light up whenever an object would appear in one of the zones. Objects appearing closer than 50 cm from the scanner, but outside of the zones, should not result in the lighting of a LED light.

Table 7.1: Some key specifications of the RPLidar A2 [13].

Item	**Unit**	**Typical Value**
Distance Range	Meter (m)	0.15 - 6
Angular Range	Degree	0 - 360
Distance Resolution		<1% of the distance
Angular Resolution	Degree	0.9
Sample Frequenzy	Hz	up to 4000
Scan Rate	Hz	10

Figure 7.1: The RPLIDAR A2.

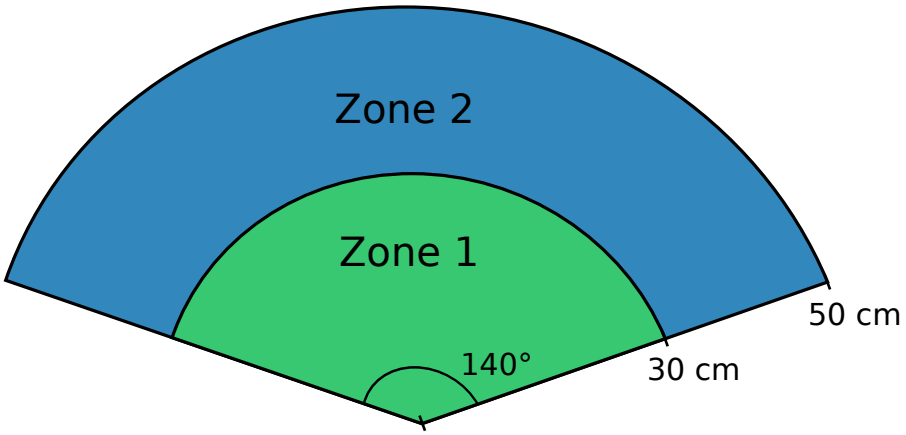


Figure 7.2: The scanning zones, with the corresponding LED light coloring.

7.1 Program Requirements

The program should give the following responses to the specified conditions:

1. If the closest object appears in zone 2, the blue LED should light up.

2. If the closest object lies within zone 1, the green LEDs should light up.

3. If all objects within 50 cm are placed outside of the circle sector, no LED should light up.

4. If the closest object is outside of the zone, but another object is placed within one of the zones, the closest object should be disregarded, and the program should respond to the in-zone object in accordance with point 1 or 2.

Point 3 and 4 are important to make sure that the program does not see parts of the robot as obstacles, hence shutting down the motors unnecessarily.

7.2 Methods and Implementation

7.2.1 Set-up

An Arduino Uno microcontroller was used for control, see Figure 7.3. Even though a more sophisticated and powerful controller, more specifically a PLC, is going to be used in the final product, an Arduino has some advantages when doing experiments: It is cheap and accessible, has input and output ports that can easily be reprogrammed to perform various tasks, which makes the board applicable for many different projects. Furthermore, uploading new code to the board, hence changing its tasks, is done by pushing a button. This is advantageous when doing experiments where trial and error, and rapid minor modifications are common. The Arduino board was connected to a MacBook via USB.

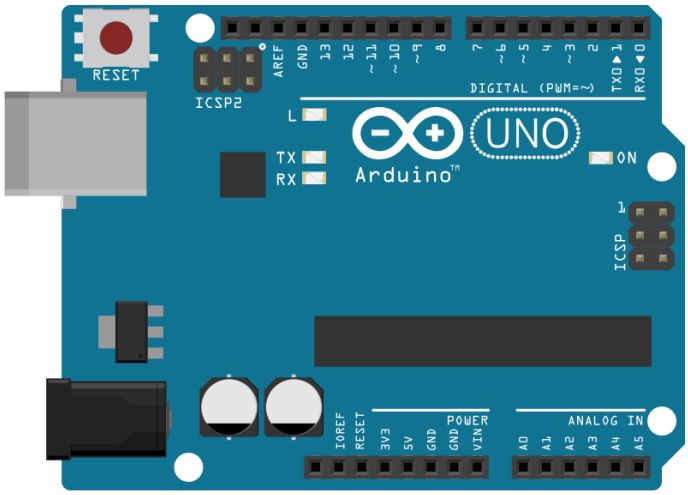


Figure 7.3: Shows the ports of an Arduino UNO [2].

Further, the RPLidar was connected to the Arduino, see Figure 7.4. The sensor uses a Universal Asynchronous Receiver/Transmitter (UART) to perform serial communication with its host. Figure 7.5 shows the wires used to connect the scanner with the Arduino. The communication and power transmission occurs through five separate wires with different tasks, as described in Table 7.2. These wires were connected to fitting input or output ports on the Arduino.

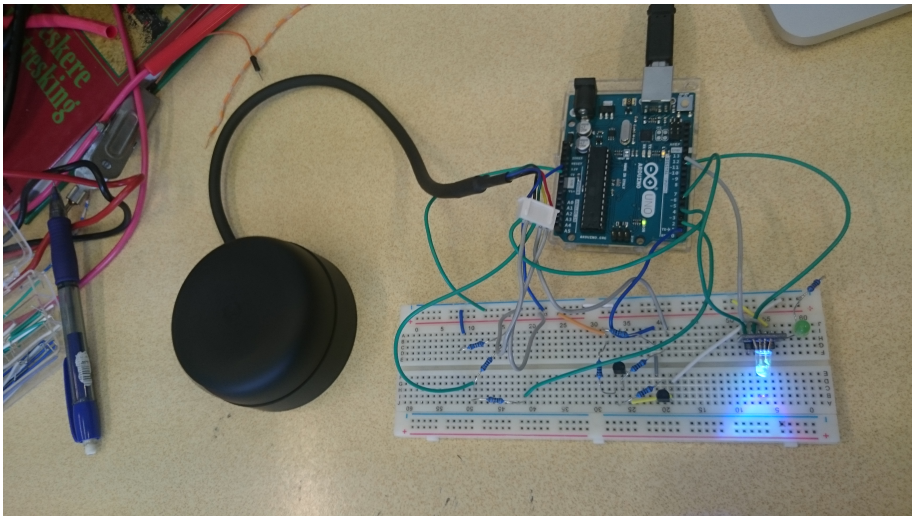


Figure 7.4: The RPLidar paired with an Arduino. The Arduino uses data from the scans to control a multicolored LED light on the circuit board.

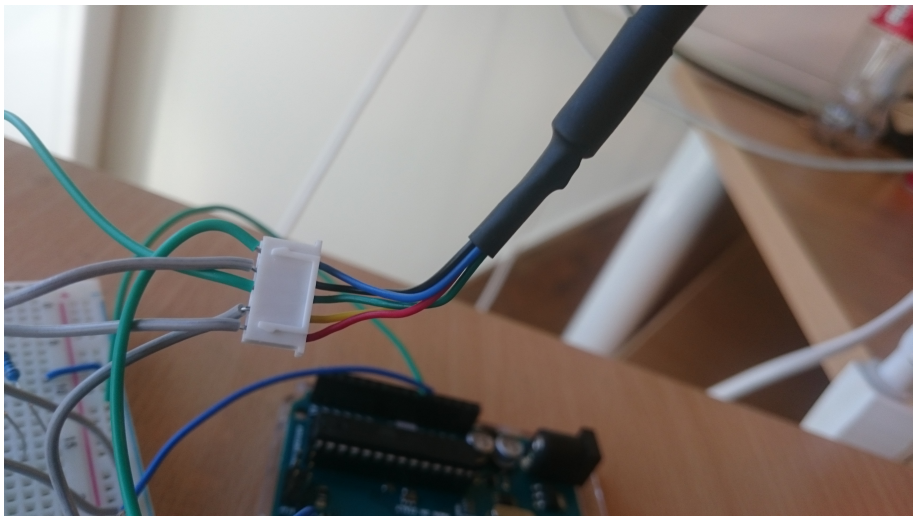


Figure 7.5: The communication and power wires of the RPLidar.

Table 7.2: The serial communication wires of the RPLidar, with color code, signal name, type, description of function, and typical values for each wire [13].

Color	Signal Name	Type	Description	Typical Value
Red	VCC	Power	Total Power	5V
Yellow	TX	Output	Serial port output of scanner core	3.3V
Green	RX	Input	Serial port input of scanner core	3.3V
Black	GND	Power	Ground	0V
Blue	MOTOCTL	Input	Control of motor	3.3V

A problem that arose was that the TX and RX ports of the RPLidar are designed to tolerate and deliver 3.3V signals, while the designated ports on the Arduino operate at 5V. A unidirectional level converter would need to be put together on a circuit board, to prevent frying of the RPLidar UART ports. The recipe for the level converter is shown in Figure 7.6 [8].

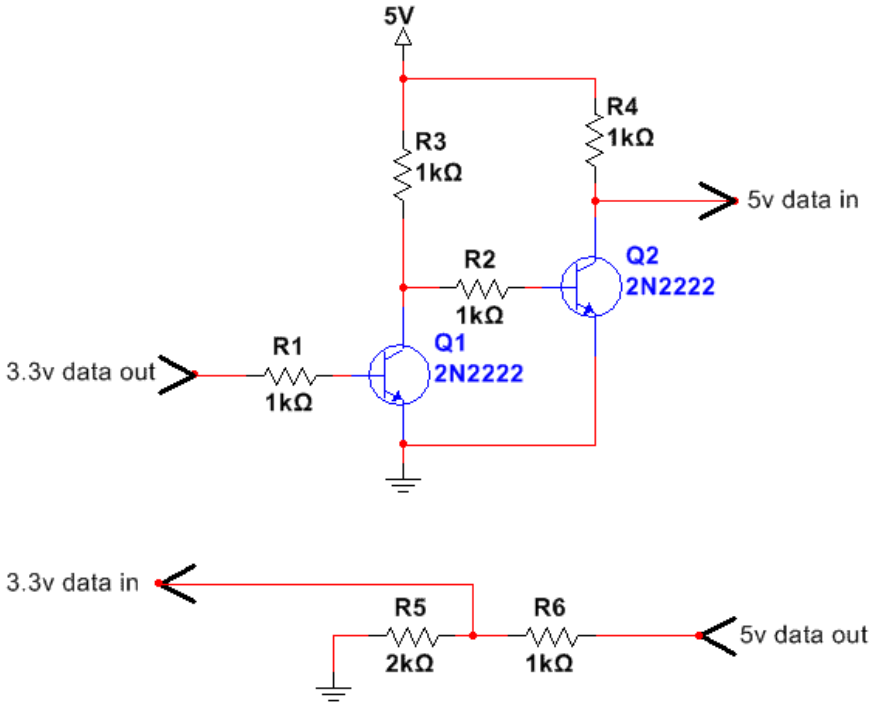


Figure 7.6: The wiring of the circuit board, to obtain the 3.3V RX/TX-voltages required by the scanner from the 5V delivered by the Arduino [8].

7.2.2 Methods

We started by connecting the Arduino to a MacBook, and ran a test program to check functionality and communication. An RPLidar driver library for Arduino was downloaded from GitHub [50]. The level converters were then wired up on the circuit board, as shown in Figure 7.6, and the voltage drops were checked by use of a multimeter. We connected the RPLidar to the Arduino, with the following wire connections (see Figure 7.3 for map over the Arduino's ports):

- TX from the scanner via level converter to RX (digital port 0) on the board.
- RX from the scanner via level converter to TX (digital port 1) on the board.
- VCC from the scanner to 5V power port on the board.
- Ground wire from the RPLidar was connected to ground on the board.
- MOTOCTL was connected to digital port 3 on the board.

A multicolored LED light was then placed on the circuit board. The light and the Arduino were coupled by three connections: green light to port number 10, blue light to port number 11 and ground to ground.

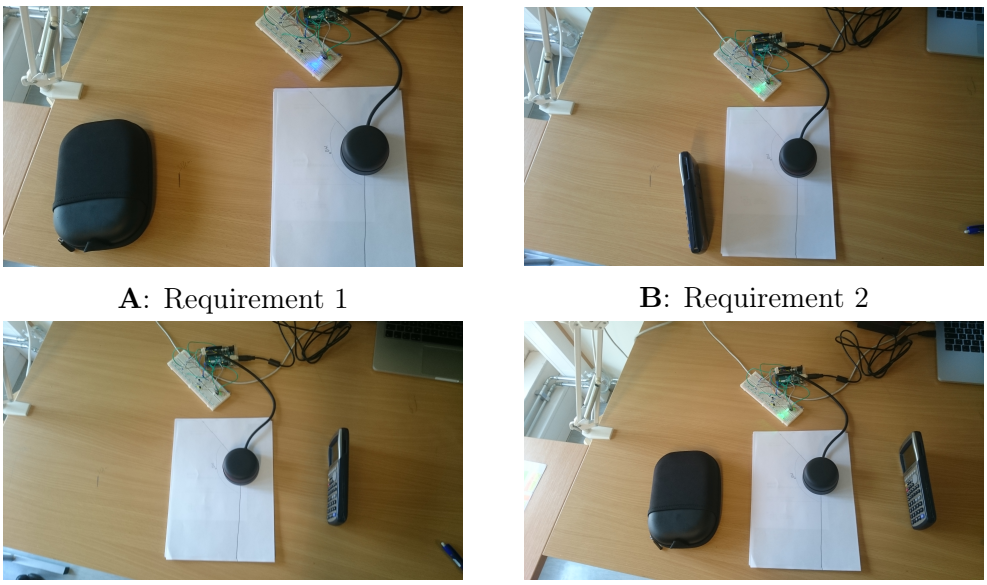
With all the necessary connections made, coding could begin. An example program from Robopeak's GitHub page [50] was used as a basis for the program. It was written in C++ in a designated Arduino Integrated Development Environment (IDE). The IDE compiled the code and uploaded it to the board. An excerpt of the computer program is shown below, while the full program can be found in Appendix A.

Listing 7.1: Part of the program written for the case study.

```
#include <RPLidar.h>

// Creating a driver instance
RPLidar lidar;

// Setting PIN mapping.
#define LED_G        10
// The PWM pin for drive the Green LED
#define LED_B        11
// Blue LED
#define RPLIDAR_MOTOR 3
// The PWM pin for control the speed of RPLIDAR's motor.

void displayColor(float distance, float angle)
{
// Setting the LED to full light
// if object is detected within zones

  if (distance <= 500){
        analogWrite(LED_B, 255);
        analogWrite(LED_G, 0);

        if (distance <= 300){
          analogWrite(LED_G, 255);
          analogWrite(LED_B, 0);
          }
  }
  else {
        analogWrite(LED_G, 0);
        analogWrite(LED_B, 0);
  }
}
```

7.3 Results

Running the program and placing objects in different positions around the LIDAR suggested that the program worked as intended, some of the results are shown in Figure 7.7. All conditions listed in Section 7.1 were tested several times. Since the sensor has a minimum scanning range of 15 cm (see Table 7.1), due to the transmitter and the receiver being placed some distance apart, no lights lit up when an object was placed closer than 15 cm from the scanner's core.



A: Requirement 1 **B**: Requirement 2

C: Requirement 3 **D**: Requirement 4

Figure 7.7: Showing the lighting of the LED light in accordance with the program requirements.

Chapter 8

Discussion and Conclusion

8.1 Findings

A safety system for the agricultural robot Thorvald has been designed. A four-layered system was proposed to fulfill a set of defined requirements. The layers work independently of each other. Layer 1 is concerned with navigation and obstacle avoidance, while layer 2 stops the robot should it come too close to an obstacle. Should a collision happen while Thorvald is equipped with heavy or sharp tools, layer 3 stops the robot at impact, and layer 4 gives humans possibilities for overriding the rest of the system. Furthermore, sensors and their placement on the robot body were chosen. These sensor choices were based on a background study of various sensor technologies and examples of safety systems implemented in other autonomous robotic platforms. Selection criteria included accuracy, simplicity of analyzing sensor output, sensor range, usability under different weather conditions, need for computational power and price.

A case study was conducted, where an RPLidar A2, a 2D laser scanner, was connected to an Arduino microcontroller board. The purpose of the study was to determine whether this laser scanner was fit to be the designated sensor for layer 2 of the system, as well as the study being the beginning of the implementation of that layer. A C++ program was written to make LED lights light up when the LIDAR detected objects that were too close, as a substitute for shutting down the robot's motors. Both the program and the LIDAR seemed to work as intended.

8.2 System Design

The major advantage of the system design is its layered structure. In a layer-based system different elements work independently of each other. The point of a safety system is to be able to handle unforeseen events. One possible unforeseen event can be a computer system failing. In such instances, having the safety system spread over several different systems will substantially lower the probability of something going wrong.

A layer-based system is easier to comprehend than a single complex system. This makes for easier testing and configuration, as different layers can be tested individually.

Also, having a layer-based design with easily modifiable layers, allows for smooth system changes between different Thorvald module configurations.

When it comes to use of collision avoidance sensors, the system is equipped exclusively with LIDARs. Given that the different models used are of roughly the same quality, this will mean that layer 1 and 2 will have trouble under the same conditions. An example of this is operation in dense fog. As the robot should not be allowed to drive when either of these layers are not functioning, it should not pose an extra problem that they are out simultaneously.

However, having the same sensor types in the two layers also raises the possibility of an obstacle that for some reason is missed by one layer, also will be missed by the other.

ROS is open source. Having a navigation system that runs on open source software magnifies the need for a separate layer using a controller that meets industrial standards. This because open source software is more likely to crash, and if it does, there is no legally responsible manufacturer behind the software. Avoiding open source software is at this point not an option, for economical reasons.

It will always be possible to design an even safer system. It's merely a question of time and money. This needs to be weighted against the potential for damage. For example, Google's initial self-driving car was equipped with extra equipment worth $150,000, including a $70,000 LIDAR [47]. Such amounts are of course out of the question in this project. Neither is it necessary, as driverless cars are much heavier, faster and operates in more unpredictable environments than Thorvald.

The specifics of each layer of the safety system are described in the following sections.

8.2.1 Layer 1

The first layer of the safety system consists of path planning and navigation. The navigation layer is the outermost layer. As long as this layer works properly, it is the only layer actually controlling the robot. Even though they are always active in the background, the other layers will not need to take control over the robot as long as the navigation layer works as intended. Therefore, the navigation layer was given a 3D-scanner, that is, the most expensive and advanced sensor. This to minimize the probability of other layers having to kick in, which will in turn result in the operation running as smoothly as possible.

When it comes to navigation, Thorvald already has the ROS nav stack implemented. A Velodyne 3D LIDAR is now being tried as a sensor for localization and obstacle avoidance. The testing suggests that the scanner gives more output than the robot's computer is able to process. A separate computer dedicated to handling sensor output only has been ordered, and is assumed to solve this problem.

Other sensor types can also be a good fit for this layer. Cameras or stereo cameras with good image-processing algorithms would be able to detect and determine the positions of obstacles. As digital cameras are cheap off-the-shelf products, this could result in lower costs. However, as most cameras are fixed, several would be needed to be able to see what is to the sides and back of the robot, making the cost higher and data processing requirements greater. Also, to what extent direct sunlight influences the quality of the output would need to be tested.

Tests where the Velodyne is fastened at different heights should be carried out, to find a good compromise between range and the size of the scanner's blind area around the robot. As the robot's vicinity is covered by additional 2D-LIDARs, the range of the 3D-LIDAR should be the most important factor.

8.2.2 Layer 2

Layer 2 of the safety system is a 2D-LIDAR connected to a PLC, so that it is independent of ROS.

As this layer scans the environment in 2D, it is possible for obstacles in the air or at the ground to go by unnoticed, thereby risking being hit. As not hitting humans is the main objective, the LIDAR should be placed as low as possible without cutting the motors when driving through crops. This height might differ depending on crop size, and thereby by seasons.

A rather large uncertainty of this layer is the stopping distance, as we don't know how quickly the regeneration and moment of inertia will stop the robot. This distance might differ between Thorvald configurations. Determining this distance by testing will in turn help to more accurately decide the proper sizes for the obstacle search zones. The smaller they are allowed to be, the less likely it is that the robot

will come to unnecessary stops. The stop zone could potentially be determined to be so big that objects enter the zone even when navigation works properly. If that is the case, it might be necessary to install a mechanical braking system that stops the wheels when the motors are shut down, to minimize the breaking distance.

As the distances associated with this layer are rather small, having small margins of error is essential. One of LIDARs' strengths is their accuracy, i.e to be able to inform accurately where an obstacle is located. The chosen sensor, an RPLidar A2 has a maximum distance error of 1% of the scanned distance, and takes a sample every 0.9 degrees, which should be good enough to accurately determine the position of objects in front of the robot.

The distance to the detected obstacle is here of greater importance than the exact location: if the robot is about to crash into an obstacle 30 cm in front of it, it is not important if that obstacle is located slightly to the right or left of the robot's center. Therefore, a set of sonars might be a viable sensor choice for this layer. The directional inaccuracy sonars bring would not necessarily be as problematic as with longer distances. However, for making the edges of the scanned zones sharp, LIDARs will be advantageous.

Furthermore, the width of the robot and the length of the zones are the only variables relevant for this LIDAR-setup. This will make for an easy transferability between different Thorvald configurations.

The RPLidar is at this date not waterproof, but a waterproof model should be available this summer [1].

8.2.3 Layer 3

In layer 3, a pressure-sensitive safety edge should be mounted whenever Thorvald is equipped with heavy or sharp tools.

It was suggested to place the edge on a bumper installed at the front of the robot, 20 cm off the ground. To determine if 20 cm is an applicable clearing distance, it must be tested in various terrains. If the safety edge frequently bumps into the ground, it will need to be placed higher above the ground.

Despite efforts to design a system that defends the animal life from injury, there are animals too small to be picked up by this bumper. Making sure that the robot stops when hitting a human, child or adult, is the primary focus of this layer.

The choice not to apply the safety edge unless Thorvald is carrying heavy or sharp equipment, might result in the robot causing damage to small animals. It is, however, considered non-lethal for humans to get run over by any of the current Thorvald configurations if no such equipment is attached.

[1] Personal communication with Lars Grimstad

8.2.4 Layer 4

Having emergency stop buttons installed is an obvious priority. The probability of having taken all possible scenarios into consideration and implemented the proper automated safety measure is low, and it is important to have a human stop-switch if the sensor-based system should fail.

8.3 Case Study

The case study showed that the RPLidar A2 would be able to serve as a sensor for layer 2 of the safety system. Based on the LEDs' behavior it seemed to give the desired output, which was easily interpreted by the microcontroller.

Using an Arduino for the final system is of course out of the question, as they are not made for industrial use. However, it is a convenient and easily accessible controller to run a test program on, to check that the scanner operates as intended.

The scanned zones in the case study was computationally simpler than the zones to be implemented in the final version of the system, as the final zones will require the program to do a sine or cosine calculation for each scan. A program for scanning the actual system zones was also written, but when running this program, the LED lights were flickering, albeit usually in the right color according to the closest object. It is assumed that the code was requiring more computational power than what was available in the Arduino, thereby creating an unstable result.

The thinnest scanned object was a pencil. The program did react to having the pencil in the scanned area. As Thorvald will be operating on fields and thereby running through various kinds of crops, thin objects should be allowed to pass through the scanned zones without the motors being stopped frequently. This will require changes to the code.

The minimum scanning range of the RPLidar is 15 cm. More specifically, objects closer than 15 cm away from the center of the scanner will not be detected. As long as the shut-down distance is 30 cm, this will not pose a problem. It can, however, become problematic if a shorter shut-down distance is needed.

8.4 Outlook

All decisions made in this thesis, with regards to choice of sensors and their placement, are made based on information acquired from written sources. Except for the layer 2 case study in chapter 7, no tests have been conducted. Therefore, to determine the liability of the safety system described in this thesis, extensive testing must be done.

Reagarding layer 1, implementation of the nav stack is already been done by the NMBU robotics laboratory staff. They will perform several tests with different sensor technologies for creating costmaps, determining if the 3D-LIDAR Velodyne or some other sensor gives the most reliable results.

The case study showed that the RPLidar provided easily interpretable output for use in layer 2. Therefore, the next step will be connecting the scanner to a PLC and mount it onto the robot. The C++ program must be altered so that it cuts the robot's motors instead of lighting up LED lights. When this is done, testing of the stopping distance should be performed in order to determine a logical shut-down distance.

Further, safety edges must be ordered, and a bumper for the safety edge must be designed and made. Testing of various edges with different actuation forces must be conducted, to make sure the robot doesn't stop unnecessarily.

After extensive sensor testing, and the installation of the emergency stop buttons, Thorvald will be equipped with a safety system that allows for autonomous driving.

Bibliography

[1] About ros. www.ros.org/about-ros/. Accessed: 2017-03-23.

[2] Arduino home page. https//www.arduino.cc. Accessed: 2017-05-02.

[3] Bosch indego ii leaflet. https://www.bosch-do-it.de/media/media/microsites/indego_2/pdfs/leaflets_1/BOLG_13_1648_21A_Indego_Connect_LF_gb-17_low.pdf. Accessed: 2017-04-23.

[4] Costmap 2d. http://wiki.ros.org/costmap_2d?distro=kinetic. Accessed: 2017-03-23.

[5] Flir c2. http://www.flir.com/instruments/c2/. Accessed: 2017-03-13.

[6] Hokuyo utm-30lx-ew scanning laser rangefinder. http://www.robotshop.com/en/hokuyo-utm-30lx-ew-laser-rangefinder.html. Accessed: 2017-04-20.

[7] How does lidar work? http://www.lidar-uk.com/how-lidar-works/. Accessed. 2017-03-13.

[8] How to convert uart voltage from 5v to 3.3v? https://electronics.stackexchange.com/questions/186168/how-to-convert-uart-voltage-from-5v-to-3-3v. Accessed: 2017-05-02.

[9] Introduction to microcontrollers. www.societyofrobots.com/microcontroller_tutorial.shtml. Accessed: 2017-03-30.

[10] Microcontroller. http://internetofthingsagenda.techtarget.com/definition/microcontroller. Accessed: 2017-03-29.

[11] Pressure-sensitive sensing edges. http://www.londonmat.com/Sensing_Edges/print/ed_what_is.pdf. Accessed: 2017-03-14.

[12] Ros documentation. www.wiki.ros.org. Accessed: 2017-03-23.

[13] Rplidar a2 introduction and data sheet. http://www.robotshop.com/media/files/pdf2/rpk-02-datasheet.pdf. Accessed: 2017-05-01.

[14] Safety edge overview. http://www.gatemotors.co.uk/page-758-46-1. Accessed: 2017-03-15.

[15] Safety edges. http://www.elitegates.net/products.asp?cat=Safety+Edges. Accessed: 2017-03-15.

[16] Tara - usb 3.0 stereo vision camera. https://www.e-consystems.com/3D-USB-stereo-camera.asp. Accessed: 2017-03-21.

[17] Teachers guide to the infrared. http://coolcosmos.ipac.caltech.edu/image_galleries/ir_zoo/lessons/background.html. Accessed: 2017-03-08.

[18] Vlp-16. http://velodynelidar.com/vlp-16.html. Accessed: 2017-04-20.

[19] What is a plc? https://www.amci.com/industrial-automation-resources/plc-automation-tutorials/what-plc/. Accessed: 2017-04-28.

[20] What is sonar? http://oceanservice.noaa.gov/facts/sonar.html. Accessed: 2017-03-17.

[21] Zed 2k stereo camera. www.stereolabs.com. Accessed: 2017-03-21.

[22] The electromagnet spectrum. https://imagine.gsfc.nasa.gov/science/toolbox/emspectrum1.html, 2013. Accessed: 2017-03-15.

[23] Laser safety factsheet. https://www.ehs.iastate.edu/sites/default/files/uploads/publications/factsheets/laserfactsheet.pdf, 2016. Accessed: 2017-03-13.

[24] The best infrared cameras of 2017. http://www.toptenreviews.com/electronics/photo-video/best-infrared-cameras/, 2017. Accessed: 2017-03-13.

[25] Rplidar a2 360 ° laser scanner. http://www.robotshop.com/en/rplidar-a2-360-laser-scanner.html, 2017. Accessed: 2017-03-13.

[26] Isac Asimov. *I, Robot*. Gnome Press, USA, 1950.

[27] J.A. Barchanski. Safety aspects of autonomous robot software development. 2004.

[28] Marshall Brain. How microcontrollers work. http://electronics.howstuffworks.com/microcontroller.htm. Accessed: 2017-03-29.

[29] Lars Grimstad. Powertrain, steering and control components for the nmbu agricultural mobile robotic platform. Master's thesis, Norwegian University of Life Sciences, Norway, 2014.

[30] Lars Grimstad. Introduction to ros. University lecture, 2017.

[31] Lars Grimstad and Pål J From. Thorvald ii - a modular and re-configurable agricultural robot. In *Conference: IFAC World Congress*, pages 1–6. IFAC, 2017.

[32] Lars Grimstad, Cong D Pham, Huynh T Phan, and Pål J From. On the design of a low-cost, light-weight, and highly versatile agricultural robot. In *Advanced Robotics and its Social Impacts (ARSO), 2015 IEEE International Workshop on*, pages 1–6. IEEE, 2015.

[33] Andrew Hawkins. You can hail a self-driving uber in san francisco starting today. https://www.theverge.com/2016/12/14/13921514/uber-self-driving-car-san-francisco-launch-volvo-xc90. Accessed: 2017-05-07.

[34] Hong-jian Liu Jun-tao Li. Design optimization of amazon robotics. *Automation, Control and Intelligent Systems*, 4:48–52, 2016.

[35] Steve Jurvetson. Image of stanford university's self-driving car. https://www.flickr.com/photos/44124348109@N01/4044601053. Accessed: 2017-04-10.

[36] David Kohanbash. Lidar (laser scanner) fundamentals. http://robotsforroboticists.com/lidar-fundamentals/, 2014. Accessed: 2017-03-13.

[37] Vinod Kumbhar. What is the difference between plc and microcontroller? http://plc-scada-dcs.blogspot.no/2013/12/what-is-difference-between-plc-and.html#axzz4gJsP3Qpg. Accessed: 2017-05-06.

[38] D.Coleman L. Biggs and S. Blackmore. User-pa hau annual progress report. http://www.ict-agri.eu/system/files/project_reports/N2_projectManagement_ARO.pdf. Accessed: 2017-04-05.

[39] Fred Lambert. Tesla autopilot is currently using only 1 out of 8 cameras of the new hardware suite. https://electrek.co/2017/02/28/tesla-autopilot-cameras-autosteer-beta/. Accessed: 2017-04-07.

[40] Sylvian Lecorné and Alfredo Weitzenfeld. Robot navigation using stereo-vision. http://weitzenfeld.robolat.org/wp-content/uploads/2015/01/OpticFlow.pdf. Accessed: 2017-03-20.

[41] John J Leonard and Hugh F Durrant-Whyte. *Directed sonar sensing for mobile robot navigation*, volume 175. Springer Science & Business Media, 2012.

[42] Jim Lucas. What is infrared? http://www.livescience.com/50260-infrared-radiation.html. Accessed: 2017-03-08.

[43] Jim Lynch. Ford says self-driving cars on track for 2021. http://www.detroitnews.com/story/business/autos/ford/2017/04/04/ford-fully-autonomous-vehicles/100017088/. Accessed: 2017-04-06.

[44] Doug Newcomb. Ford introduces new autonomous development vehicle. https://www.forbes.com/sites/dougnewcomb/2016/12/28/ford-introduces-new-autonomous-development-vehicle-expands-test-fleet-to-9#643e14654e96.

[45] Australian Bureu of Meteorology. How radar works. http://www.bom.gov.au/australia/radar/about/what_is_radar.shtml. Accessed: 2017-04-22.

[46] plcdev.com. How plc's work. http://www.plcdev.com/how_plcs_work. Accessed = 2017-05-06.

[47] Alisa Priddle and Chris Woodyard. Google discloses costs of its driverless car tests. http://content.usatoday.com/communities/driveon/post/2012/06/google-discloses-costs-of-its-driverless-car-tests/1#.WRS7oWSGOYU. Accessed: 2017-05-11.

[48] Morgan Quigley, Brian Gerkey, and William D Smart. *Programming Robots with ROS: a practical introduction to the Robot Operating System*. " O'Reilly Media, Inc.", 2015.

[49] Mike Renslow. Lecture notes in digital image analysis. http://web.pdx.edu/~jduh/courses/Archive/geog481w07/Lec10b_LiDAR.pdf, 2007. Accessed: 2017-03-13.

[50] RoboPeak. Robopeak rplidar driver for arduino and arduino-compatible devices. https://github.com/robopeak/rplidar_arduino. Accessed: 2017-04-23.

[51] Davide Santo. Autonomous cars' pick: Camera, radar, lidar? http://www.eetimes.com/author.asp?section_id=36&doc_id=1330069. Accessed: 2017-04-30.

[52] Sam Shead. Amazon now has 45000 robots in its warehouses. http://www.businessinsider.com/amazons-robot-army-has-grown-by-50-2017-1?r=UK&IR=T&IR=T.

[53] Roland Siegwart, Illah R Nourbakhsh, and Davide Scaramuzza. Autonomous mobile robots. *Massachusetts Institute of Technology*, 2004.

[54] Michael J Smith, Richard John Koubek, Gavriel Salvendy, and Don Harris. *Usability Evaluation and Interface Design: Cognitive Engineering, Intelligent Agents, and Virtual Reality*, volume 1. CRC press, 2001.

[55] Hilde Erlingsen; Ketil Strebel. Sykehusets tekniske hjerte. https://www.nrk.no/ostfold/sykehusets-tekniske-hjerte-1.12343198. Accessed: 2017-04-11.

[56] Swisslog. Transcar agv system overview. Brochure. Accessed: 2017-04-11.

[57] The Tesla Team. Upgrading autopilot, seeing the world in radar. https://www.tesla.com/blog/upgrading-autopilot-seeing-world-radar?redirect=no. Accessed: 2017-04-10.

[58] Selim Temizer. Optical flow based robot navigation. http://people.csail.mit.edu/lpk/mars/temizer_2001/Optical_Flow/. Accessed: 2017-04-24.

[59] Tesla.com. All tesla cars being produced now have full self-driving hardware. https://www.tesla.com/blog/all-tesla-cars-being-produced-now-have-full-self-driving-hardware. Accessed: 2017-04-07.

[60] Tesla.com. Autopilot. www.tesla.com/autopilot. Accessed: 2017-04-06.

[61] Harper Adams University. Robot tractor demonstration. http://www.harper-adams.ac.uk/news/202742/robotic-tractor-demonstration-userpa-project#.WOTTHWR941I. Accessed: 2017-04-05.

[62] Johanna Wallén. *The history of the industrial robot*. Linköping University Electronic Press, 2008.

[63] Ryan Whitwam. How google's self-driving cars detect and avoid obstacles. https://www.extremetech.com/extreme/189486-how-googles-self-driving-cars-detect-and-avoid-obstacles. Accessed: 2017-05-07.

[64] Christian Wolff. Radar basics: Direction-determination. http://www.radartutorial.eu/01.basics/Direction-determination.en.html. Accessed: 2016-04-23.

[65] Jo Yung Wong. *Theory of ground vehicles*. John Wiley & Sons, 2008.

[66] www.flir.com. Seeing through fog and rain with a thermal imaging camera. http://www.flir.com/uploadedFiles/FOG_techNote_LR.pdf. Accessed: 2017-03-10.

[67] www.flir.com. How does an ir camera work? http://www.flir.com/corporate/display/?id=41523, 2015. Accessed: 2017-03-07.

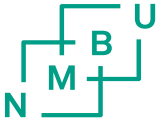
Appendix A

Arduino Code

```
// This code is based on the RPLIDAR driver library
//provided by RoboPeak.
#include <RPLidar.h>

// Creating a driver instance
RPLidar lidar;

// Setting PIN mapping

#define LED_G       10
// The PWM pin for drive the Green LED
#define LED_B       11
// Blue LED
#define RPLIDAR_MOTOR 3
// The PWM pin for control the speed of RPLIDAR's motor.

void displayColor(float distance)
{
  // Setting the LED to full light
  // if object is detected within zones

  if (distance < 500){
        analogWrite(LED_B, 255);
        analogWrite(LED_G, 0);

        if (distance <= 300){
          analogWrite(LED_G, 255);
```

```
          analogWrite(LED_B, 0);
        }
  }
  else {
        analogWrite(LED_G, 0);
        analogWrite(LED_B, 0);
  }
}

void setup() {
  // bind the RPLIDAR driver to the
  //arduino hardware serial
  lidar.begin(Serial);

  // setting pin modes
  pinMode(RPLIDAR_MOTOR, OUTPUT);
  pinMode(LED_G, OUTPUT);
  pinMode(LED_B, OUTPUT);
}

float minDistance = 100000;
float angleAtMinDist = 0;

void loop() {
  if (IS_OK(lidar.waitPoint())) {

        float distance = lidar.getCurrentPoint().distance;
        float angle = lidar.getCurrentPoint().angle;

        if (lidar.getCurrentPoint().startBit) {
          // a new scan,
          // display the previous data...
          displayColor(minDistance);
          minDistance = 100000;
          angleAtMinDist = 0;
        } else {
          if ( distance > 0 &&
          distance < minDistance &&
          angle < 140) {
                minDistance = distance;
                angleAtMinDist = angle;
          }
        }
```

```
  } else {
        analogWrite(RPLIDAR_MOTOR, 0);
        //stop the rplidar motor

        // try to detect RPLIDAR...
        rplidar_response_device_info_t info;
        if (IS_OK(lidar.getDeviceInfo(info, 100000))) {
          //detected...
          lidar.startScan();
          analogWrite(RPLIDAR_MOTOR, 255);
          delay(1000);
        }
  }
}
```



Norwegian University
of Life Sciences

Postboks 5003
NO-1432 Ås, Norway
+47 67 23 00 00
www.nmbu.no