Pathfinder
Autonomous Guided Vehicle using Infrared Light

OSKAR NORDSTRÖM
ALEXANDER AXELSSON
Pathfinder

Autonomous Guided Vehicle using Infrared Light

ALEXANDER AXELSSON, OSKAR NORDSTRÖM

Bachelors’s Thesis at ITM
Supervisor: Nihad Subasic
Examiner: Nihad Subasic

TRITA-ITM-EX 2018:60
Abstract

In the world, research on autonomous navigation vehicles (AGV) is growing by the day. The goal with this project was to create an AGV and explore the possibility of using infrared reflections as a navigational method and how to achieve distinct reflection measurements from a surface. The thesis also discusses the possibility of using several units on a larger scale. In the progress of the project, a prototype vehicle was built to conduct the experiments to identify the suitability of infrared navigation. The testing of the prototype shows that navigation by IR can be very reliable under controlled circumstances. The project also explored how hierarchical software architecture stands in comparison to purely local or centralized software architecture.

Keywords
Mechatronics, Software architecture, Autonomous guided vehicle, Breadth-first Search
Referat


Nyckelord
Mekatronik, Mjukvaruarkitektur, Autonomt styrt fordon, Bredden-först-sökning
Acknowledgements

First of all, we would like to thank our course supervisor and examiner Nihad Subasic for his warm welcome at the start of the course as well as a well executed introduction. We would also like to thank Staffan Qvarnström and Tomas Östberg for their assistance regarding the physical construction of the prototype as well as general advice on electronics and such. A big thank you to Stefan that helped us identify and fix any problems that arose when we wanted to use any of the laser cutters or 3D-printers. Without the help of Johan, Ruben and Stanislav, many additional hours would have been spent on developing the source code for the prototype. Not to mention the help we got from our fellow students during conversations.
## Contents

1 Background 3  
1.1 Introduction .................................................. 3  
1.2 Purpose .......................................................... 3  
1.3 Scope ............................................................ 4  
1.4 Method ........................................................... 4  

2 Theory 5  
2.1 Microcontroller .................................................. 5  
2.2 Pathfinding ....................................................... 5  
2.3 Interpreting Environment ....................................... 6  
2.4 Pulse Width Manipulation (PWM) .............................. 7  
2.5 Motors ............................................................. 7  
   2.5.1 DC Motors .................................................... 7  
   2.5.2 Stepper motor ............................................... 8  
2.6 H-Bridge .......................................................... 8  

3 Prototype 11  
3.1 Electronics ....................................................... 11  
3.2 Software .......................................................... 12  
3.3 Physical design .................................................. 16  

4 Result 19  

5 Discussion 21  
5.1 Electronics ....................................................... 21  
5.2 Software .......................................................... 21  
5.3 Physical design .................................................. 22  
5.4 Result disturbance .............................................. 23  

6 Conclusion 25  

7 Bibliography 27  

Appendices 28
A  Source code  
   A.1  Linked queue (Python)  29
   A.2  Breadth first search algorithm (Python)  30
   A.3  Main program (Arduino IDE)  34
   A.4  Discarded Node.h (Arduino IDE)  39
   A.5  Discarded Node.cpp (Arduino IDE)  39

B  Electrical schematics and datasheets  41
   B.1  Full schematic  41
   B.2  Arduino Uno Datasheet  43
   B.3  Maxon 226754 DC Motor  46

C  Testing results  49
   C.1  Table of measurements  49
   C.2  Frequency Diagrams  51
List of Figures

2.1 Illustration of an ancestor node with four child nodes which have a parent pointer to the ancestor, made in Paint .......... 6
2.2 Illustration of a DC motor’s structure [5] ................. 8
2.3 H-Bridge schematic[11] .................................. 9
3.1 Reflex detector schematic made in CircuitLab [6] ............... 11
3.2 Illustration of how the breadth first search algorithm works, made in draw.io [9] ........................................ 14
3.3 Flow chart of the main program, made in draw.io [9] .......... 15
3.4 Photo of the prototype’s top side (Left) and under side(Right) ... 16
3.5 Photo of the prototype with cover on ........................ 17
4.1 Sensor Values Running (Left) and Sensor Values Test(Right) .... 19
4.2 Illustration of how testing was conducted, made in Paint .......... 20
C.1 Example frequency table used for frequency diagrams ........... 50
C.2 Frequency diagram of values middle reflex detector ............. 51
C.3 Frequency diagram of values right reflex detector ............... 52
C.4 Frequency diagram of values right reflex detector ............... 53
C.5 Frequency diagram of values right reflex detector ............... 54
C.6 Frequency diagram of values front reflex detector .............. 55
C.7 Frequency diagram of values front reflex detector .............. 56
List of Tables

4.1 Sensory Values: Running state ........................................ 20
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGV</td>
<td>Autonomous Guided Vehicle</td>
</tr>
<tr>
<td>ANSI</td>
<td>American National Standards Institute</td>
</tr>
<tr>
<td>BFS</td>
<td>Breath-First Search</td>
</tr>
<tr>
<td>DC</td>
<td>Direct Current</td>
</tr>
<tr>
<td>IR</td>
<td>Infrared</td>
</tr>
<tr>
<td>L298</td>
<td>L298n dual motor controller</td>
</tr>
<tr>
<td>LED</td>
<td>Light Emitting Diode</td>
</tr>
<tr>
<td>LDR</td>
<td>Light Dependent Resistor</td>
</tr>
<tr>
<td>PT</td>
<td>Photo Transistor</td>
</tr>
<tr>
<td>PWM</td>
<td>Pulse-Width Modulation</td>
</tr>
<tr>
<td>SRAM</td>
<td>Static Random Access Memory</td>
</tr>
</tbody>
</table>
Chapter 1

Background

1.1 Introduction

The E-commerce market is growing by the day. In this type of business, products are ordered online and sent from warehouses to the customers. Having transport units in these warehouses allows companies to operate 24 hours a day. When employees are off for the day, an automated system can function in their place, or even at the same time. The system could keep track of the inventory and deliver it to the transport section of the business. This is just one example of the usefulness of an autonomous guided vehicle (AGV).

1.2 Purpose

The results of this Bachelors project in Mechatronics at the Swedish Royal Institute of Technology (KTH) is addressed in this thesis. The purpose of this project was to build a prototype that could navigate an open space based on markings on the ground, an AGV. By scanning the ground with a number of Infrared (IR) detectors, the prototype will navigate itself through space by using a breadth-first search (BFS) to compute a path.

The construction and refinement of AGVs is an active and intriguing field of study. A common solution is a vision based AGV using a camera but we believe a camera to be unnecessary and that it can be done in a simpler way using cheaper components while also making the underlying program less resource heavy. This could be achieved using a "vision" based system with a fraction of the resolution of most cameras.[17]
CHAPTER 1. BACKGROUND

Research questions:

1. How can autonomous navigation in an open space be achieved?

2. How should the markings differ from the ground in order to give a distinct difference in sensory output?

3. Is this project possible to be scaled up in terms of having more than one unit operating at the same time?

1.3 Scope

This project was subject to the following constraints due to budget and allotted time.

The prototype unit would operate in a small scale model of a warehouse. During the tests the conditions were artificially ideal, meaning that most disturbances could be controlled.

1.4 Method

To answer the research questions, experiments were conducted with a created prototype. The sensor circuit was created first and used to measure how reflective different surfaces were, in order to know which materials to use for the area model during tests later on. The software was structured in a hierarchical way and a path-finding algorithm was implemented for use in a normal computer while the ability to follow this path was programmed into an Arduino Uno on the prototype. When the physical prototype was finished, a series of empirical test were conducted to gather data. These were put into frequency diagrams for analysis, that would improve the fine-tuning of variables to increase robustness and to provide an easily readable visual representation of the results.
Chapter 2

Theory

2.1 Microcontroller

A microcontroller is a small computer integrated on a single circuit board. The microcontroller usually has several connections which can be set as inputs or outputs when programming the controller. These can either be analogue or digital or both depending on the sophistication of the controller itself. Through these connections the microcontroller can receive feedback from sensor-circuits or send control signals to sub-systems. The microcontroller and the sub-systems which it controls can have different power supplies. This enables a modular approach to the design process of systems using a microcontroller which is very useful. Microcontrollers are both cheap and small which makes them an ideal control unit in a smaller autonomous system like this project.[1]

2.2 Pathfinding

In order to find a path between one point and another, the breadth-first algorithm can be used. The breadth-first algorithm is a search algorithm. The point of it is to find the first occurrence of the desired data in a binary search tree or other types of graphs. A binary search tree is made up by a set of nodes which are then linked to each other. A node is a data type which holds at least one memory pointer and piece of data. This memory pointer is used to link said node to another in order to form a graph. The very first node in the tree is called the root or ancestor node and would in our case be the starting location of our prototype. Every node has the potential to have more nodes linked to it at a lower level and these are considered to be children of said node. One of these children would be the prototype’s final destination. The graph used by the prototype would require each node to have the possibility of having up to 4 children, one for each direction the prototype can possibly drive on the grid [Figure 2.1].
What the breadth-first algorithm would do is to search each "generation" (level) of nodes until the desired child node (final destination) is discovered. By recursive back tracking the node path can be extracted. This path would be the first possible and fastest path if the following assumption is true. In this project the cost of traversing between each node is set to be identical (a turn is as fast as a straight forward drive), while in reality a turn takes longer time. In order to not end up in a never ending loop of calculations, possible children that have already been a child to any node should be put on a "black list", as would off-limit areas. Black listed nodes will not be generated as children again.[2]

2.3 Interpreting Environment

In order for the prototype to be able to tell where it should go it has to be able to receive data from the outside world. This can be achieved by for example marking a grid pattern on the ground. This pattern could then be read by sensors. A sensor is a component that detects the changes in the surrounding environment, could be light or heat for example. Light Dependent Resistors (LDR) are one type of sensor that could sense a pattern on the ground.

Each intersection in the grid could be considered a node for the path finding algorithm to consider while doing the path calculation at the very start [Section 2.2]. By using at least three sensors along a line which is orthogonal to the drive direction the prototype can detect each intersection, using that information for a
When choosing LDR:s a few things have to be considered. The range, field of view, and sensitivity must be acceptable for the application. You have to consider a source of light for the LDR. A seemingly suitable version of an LDR for this project is the reflex detector. A reflex detector consists of a Light Emitting Diode (LED) and a Photo Transistor (PT) which can sense the light reflected from said LED. A PT is simply a Transistor where the base senses light. The reflex detector used in this project uses IR light, which is not part of the light spectrum humans can see, to reduce environmental disturbances from common unnatural light sources. The microcontroller has to interpret the voltage being fed to it from the LDR-circuit and use that information to determine what it is looking at.

By performing empirical tests, the sensor values corresponding to each surface can be measured. When the detectors gives a signal of this measured value the prototype should know that it has crossed a line in the grid.

In order to avoid colliding with unexpected obstacles some sort of distance sensor can be used. There are several kinds of sensors that detect distances. These include but may not be limited to lasers and ultrasonic sensors. When the prototype detects that there is an obstacle in its path it has to either wait until the obstacle is removed or calculate a new path.

### 2.4 Pulse Width Manipulation (PWM)

PWM is a way of emulating an analogue signal with a digital signal. The digital signal from the Arduino is either 5V (high) or 0V (low) and thus a square wave signal can be generated. Depending on the percentage of time the digital output is set as "high" versus "low" the shape of this square wave, the pulse width, can be chosen and it can then be interpreted as an analogue output by the receiver. A pulse width of 60% would mean that the signal is set as high 60% of the time and the signal would be interpreted as 60% of the max source voltage (for this to work, the square wave has to be sent on a high enough frequency, for Arduino this is around 500 Hz). An example of an application for PWM is a switch that determines how much voltage is sent to a motor.

### 2.5 Motors

#### 2.5.1 DC Motors

A DC Motor is a way to exert mechanical energy from an electric current. By sending a direct current (DC) through a coil, an electromagnetic field is generated. This field has the same charge (positive or negative) and direction as the
field from the nearby field of a permanent magnet. As seen in Figure 2.2 the electrical field will repel from the nearby magnet and attract to the far away magnet, which generates a force vector. This force vector is perpendicular to the axle, which generates torque to turn the axle.

2.5.2 Stepper motor

A stepper motor is another kind of electrical powered motor which is used for applications in measurement and control. There are a few things that make them ideal for such applications. They are not as prone to failure and do not create, for some environments, dangerous electrical arcs like conventional motors. Their rotation speed is independent of the applied load, as long as the load exerts a torque lower than the motors rated torque. Because of their design of moving in incremental steps they do not need an external feedback control to know which position they are in, as long as the load is not higher than the rated torque.

It may sound like a stepper motor is superior to a DC motor in every way but it does however require double the amount of control circuits compared to a DC motor.[19]

2.6 H-Bridge

An H-bridge is an electrical construction which can be used to control the direction of current. The component employs Transistors as switches which block or let pass the input current. Paired together with a PWM signal, they are used...
2.6. H-BRIDGE

to control the speed of a motor. Figure 2.3 illustrates how an H-bridge is constructed.

![H-Bridge schematic](image)

**Figure 2.3.** H-Bridge schematic[11]
Chapter 3

Prototype

The following chapter will explain the prototypes physical components and the software behind the prototype.

3.1 Electronics

Reflex detectors

The prototype uses four reflex detectors in order to keep track of the markings on the floor in order to operate. Each reflex detector is made up of an IR LED and a PT sensitive to IR light. The amount of light from the IR LED that is reflected back to the PT is dependent on the surface it illuminates as well as the distance of said surface. The schematic [Figure 3.1] shows the three main reflex detectors in one circuit but there are actually four on the prototype [Appendix B.1]; D, R and PT corresponds to diode, resistor and Photo Transistor respectively and the index indicates which reflex detector they belong to. Each reflex detector sends out a signal based on the amount of IR light the PT receives. Please note that the symbol of the PT is not the standard schematic symbol of a PT, however the software used to sketch this schematic did not have that symbol. The used reflex detector is of the model OPB - 745.[21]

![Figure 3.1. Reflex detector schematic made in CircuitLab [6]](image-url)
CHAPTER 3. PROTOTYPE

H-bridge

The prototype uses a L298N dual motor controller, which is a dual H-bridge. Its specifics can be found in its datasheet.[22] This component allows control of two DC motors or one stepper motor, the prototype uses the L298N to control two DC motors. The L298N has 6 inputs, 4 digital and 2 analogue inputs, all sent from the Arduino. The digital inputs control the direction of the motor. Each motor takes two digital inputs, as well as one analogue input, which is the PWM signal from the Arduino.

Power supply

The prototype is powered by three 9V batteries. Two batteries are connected in a series connection to supply 18V to the motor driver (L298N H-bridge), which directs appropriate voltage to the motors according to the PWM-signal from the Arduino. The third 9V battery is used to supply the Arduino Uno with power.

Motors

The motors used are a Maxon motor: Re-Max 29: Part Number: 226754.[Appendix B.3] The complete set of DC motor and gear is known as: Maxon Re-Max 301929 Swiss Mode x 03

3.2 Software

The prototype uses an Arduino Uno microcontroller to execute its task. Code has been written in both Python and the Arduino Integrated Development Environment (IDE) which is built on the programming language C.[12] There are three stages: the path finding computation stage, set-up stage, and the loop stage.

In the path finding computation stage a computer computes a path from its location to its destination and generates an array of data which is then dumped into a .txt-file. The information in this .txt-file translates to a to-do list to go through during the loop (where the prototype actually traverses the space). This information consists of an array of numbers from one to five. The numbers from one to four represents directions relative to a grid (not relative to the prototype) while the number five tells the prototype that there are no more tasks to do.

In the set-up stage the information in the previously mentioned .txt-file is entered into an array in the Arduino code. The prototype’s starting direction relative to the grid also has to be entered as a number next to this array. The rest of this stage defines what each pin on the microcontroller should do when the program enters the loop stage later on.
3.2. SOFTWARE

In the loop stage the prototype follows an algorithm to translate its position from
the start to the goal.[Figure 3.2]

Node

A custom node class was made in Python by refurbishing code made in a previous
course at KTH. The original code was written by Alexander Axelsson and Carl
Foghammar Nöntak. The node class contains a parent pointer, coordinates and
functions to access all of these from outside the class.[2]

Linked queue

A simple linked queue class is required in order to enqueue the nodes that are to
be evaluated. This code was re-used and taken from the same course at KTH as
the previously mentioned refurbished code. Authors were also Alexander Axelsson
and Carl Foghammar Nöntak.[2]

Breadth-first search algorithm (BFS)

The BFS search algorithm is implemented in the prototype. Grid size, start lo-
cation, and destination has to be declared by the user before a path can be gen-
erated. This is how it works: First of all a linked queue, q, and two arrays are
declared, VisistedCoordinates and taskList. The linked queue is used to store
generated nodes that are to be evaluated. The array 'VisitedCoordinates' keeps
track of which coordinates have already been visited and which shouldn’t be gen-
erated again; if the user knows that certain coordinates in the grid are blocked
those coordinates should be added to this list before generating the path. The
array "taskList" is the list of instructions which is sent to the prototype. This
list will consist of numbers between one and five; the numbers one to four rep-
resents directions while the number five means that the prototype has reached its
goal. This code comes from the same file as the node class and was heavily refur-
bished.[2]

A flowchart explaining the flow of the code has been made according to ANSI
standard.[13, See Figure 3.2]
Hierarchical Architecture

To perform the tasks of computing a path and interpreting signals from the sensors a hierarchical structure of computation was implemented. What this means is that at what level in the computation structure any given process will be handled can be chosen at any level in the structure, to tune for maximum performance. The local nodes (units) perform basic processing, and only transmit the result of the interpreted raw data to higher hierarchical computation units (a central control computer). This lessens the need for a higher bandwidth compared to if a central unit performed all computing. This also allows for the use of the Arduino Uno in this project regardless, of its limited static random access memory (SRAM) of 2 kB. Depending on the task taken, the computing is done on different hierarchical levels.[Appendix B.2]

The prototype does not contain a wireless module of any sort so the BFS can not be done in real time without a cable plugged into the prototype during a run, however the principle is the same. This kind of structure was deemed essential due to the research question concerning the ability to scale up with more than one unit operating in the same space at the same time.[14]

Which functions are used to get from start to finish once the path is found?

The prototype has a number of functions to call upon based on the list of tasks generated in the BFS algorithm. These functions will be called upon between each
3.2. SOFTWARE

node based on which direction, relative to the grid, the next node is and the current orientation of the prototype.

If the next node is to the right relative to the prototype it will first call the function to turn 90 degrees to the right and then drive forward for example. This required the prototype to keep track of its orientation in the grid and that was easy to implement with a simple integer variable with values from 1-4, each corresponding to a direction relative to the grid itself. This integer is adjusted in the functions that turn the prototype. Figure 3.3 illustrates the general logic flow of the prototype while running.

Note that some minor tweaks have been added during the transition phases between certain actions to improve robustness and reliability. These tweaks include light signal processing which requires satisfactory readings a certain times in a row in order for them to count. The tweaks also include rotating slightly when back on track after steering in a direction for example.

Figure 3.3. Flow chart of the main program, made in draw.io [9]
3.3 Physical design

Designing with standard parts in mind was important to consider so as to meet the scalability requirement set out in the third research question. All fasteners in the model have been downloaded from Solid Components and used in an assembly in the CAD software to ensure that standard fasteners do work.[10]

It was deemed important that the prototype could easily rotate on the spot because it means that it was easier to stay on track when turning in a grid. This is why all the wheels turn around the same point, the middle of the prototype. Everything was mounted on a flat and relatively thin base plate because many manufacturing materials come in that form factor and it is easier to just cut out holes in such a surface compared to a more complicated one. The design also accounts for the possibility of switching out any of the parts. If a need for different wheels is in question, the distancers can easily be changed to account for it. If a need for another motor arises, the motor mount can easily be scaled differently to accommodate for this. The mounting holes for each type of component is grouped together, so simply editing the definition of one measurement will adjust it for all of that type. Below are a few photos of the prototype.

Figure 3.4. Photo of the prototype’s top side (Left) and under side(Right)
3.3. PHYSICAL DESIGN

Note that there are no 9V batteries in these images but that is because during testing, AA batteries were readily available, while 9V batteries were not. The 3D models created and used for 3D printing and laser cutting are not included in the thesis, however they are available on request to authors.
Chapter 4

Result

In Figure 4.1 are analogue readings from the left reflex detector. The data was collected from two different types of tests. Microsoft Excel was used to compile this data.

Figure 4.1 (Left) shows undistinguished data taken during a run from one point to another in the grid. The Figure indicates that there are two distinct spans of sensor readings which indicates that there is a clear distinction between the readings from the markers and the ground. Figure 4.1 (Right) confirms this with its distinguished values from controlled tests. The readings included in Figure 4.1 (Right) were collected by having all sensors placed over selected spots in the grid where all were over a marking and also having the prototype rotating in place in all the squares on the ground surface. These images show the same pattern and were thus used to calibrate the values which determines if a reflex detector is currently above the markings or surface. These tests were conducted for each reflex detector to ensure maximum robustness while running. [Appendix C.2] In Figure 4.2 below a visual representation of the tests is given. Every green dot represents a node at which the tests for distinguished values were taken from and the blue path is the path where the undistinguished readings were taken from.
As seen in [Table 3.1, Figure 4.1] the sensory outputs between ground and markings have a distinct measurable difference. As per the second question set out in the thesis, it can now be stated how sensory readings should differ to get distinct values for navigation. All tests were conducted with a reflex detector height of 3 millimetres above ground and markings.
Chapter 5

Discussion

The prototype worked to a satisfactory degree, during the exposition ’KEXPO 2018’ where we demonstrated its functionality. It only failed to complete its full run three times, two of which were because of half dead batteries resulting in poor readings and the last one was probably due to contaminations on the surface after driving on it the whole day. We tried adding a payload with a mass of 8 kg without any difference in performance.

5.1 Electronics

We had expected the three reflex detectors to be sufficient to do a precise 90 degree turn. While being true in theory, it didn’t work out that well in reality due to our limited feedback control system. The prototype usually ended up slightly translated from the ideal position and thus we added an additional reflex detector in the front to get precise 90 degree turns.

We did notice that the reflex detectors were extremely sensitive to minor changes in distance. This means that a smoother surface leads to higher robustness while running. This is however a flaw specific to our selected method of identifying the markings on the surface. Using magnetic sensors and embedding magnetic lists in the surface would probably negate this problem from example since the surface isn’t magnetic at all.

5.2 Software

An improvement to make it even more stable on the software side would be to use threshold intervals instead of a value for each reflex detector to tell it if it is on or off a marking. If we were to narrow it down to an interval instead, we could negate the effect of any disturbance that does not cause the same reflectivity as either the marking or the desired ground surface.
Software architecture proved its benefits

Even though we already knew about the benefits of a hierarchical software architecture we wanted the prototype to calculate the path itself because it would seem to work more seamlessly during the exposition at the end of the project.

So a custom node class was first made in the Arduino IDE, based on a tutorial found on Youtube.[7] The node class contained a parent pointer, coordinates and functions to access its attributes from outside the class. The BFS search algorithm was implemented in the prototype’s Arduino Uno. Grid size, start location and destination had to be declared before a path could be generated. How did it work? First of all two queues were created, myNodeList and DontVisitList. A third party library for a linked list for use as the queue was downloaded.[8] Then the program determined if the start location and destination were within the grid. Then an ancestor node with the same coordinates as the start location was generated and was put into the myNodeList as well as the DontVisitList queue. After this initial set-up in the set-up part of the program it would continue with the actual search for a path.

We did however run into an issue where we ran out of static random access memory (SRAM) on our Arduino Uno if the goal destination wasn’t close enough. This was due to the limited 2 kB of SRAM and our limited knowledge in C based programming languages, which probably lead to a less than ideal memory allocation. This resulted in variables being reset at random and the Arduino Uno getting stuck in endless loops. Thus we chose to discard all the BFS code we made in the IDE and did the path finding algorithm in Python instead which we were much more familiar and competent with. The source code for the discarded node class made in the Arduino IDE.[Appendix A.4 and A.5]

5.3 Physical design

Since our project didn’t include any requirement on speed, we tried several motors readily available at KTH. The first set of motors were very fast but this lead to a huge loss in controllability with our feedback system. It was difficult to decelerate the prototype because of the low gear ratio, weight and the fact that our feedback system doesn’t include a way of knowing if the prototype is near a node. Following this discovery several different sets were tried. The current ones are very strong but very slow due to high gear ratios.

An improvement to the prototype would be a motor with a somewhat lesser gear ratio coupled with a better feedback system so a higher speed can be achieved while remaining controllable. A way of improving the feedback system would be to change the current sensor chip used could be changed into a grid of reflex de-
5.4. RESULT DISTURBANCE

tectors of say, five times five or greater resolution. This enables the possibility of implementing a proportional integrating and derivative feedback controller (PID feedback controller). The point of a PID feedback controller is to minimize the deviation from a desired result. See the source in the bibliography for more information.[15]

Another benefit to the physical design with greater resolution would be the possibility of including global landmarks in the ground marking system. This would be a confirmation for the hierarchical computing system where each possible unit is currently located. If the units were to be equipped with horizontal distance sensors they could also detect which nodes that have become blocked without the higher systems knowing. This new information could then be relayed and a new path be generated. This would give the higher systems a chance to have real time spatial awareness of the environment. This kind of spatial awareness is called robotic mapping and more specifically a world-centric map. The act of updating a world-centric map is called simultaneous localization and mapping (SLAM) and the most common way of doing this is using cameras or lasers on the unit in question.[18, 20]

5.4 Result disturbance

During tests of the prototype, the frontal reflex detector picked up some varying results on ground measurements[Appendix C.7]. Some of these values were in the range of the marking measurements for this specific testing, where we measured on the ground and markings on specific spots [Figure 4.2]. However during further tests and during KEXPO 2018, these disturbances were not detected, so we consider them as circumstantial disturbances. We believe these readings were due to temporarily disturbance of surface.
Chapter 6

Conclusion

The purpose of this bachelors project was to answer the questions: How a prototype can achieve autonomous navigation, how ground markings need to differ from surrounding area in order to get clear distinctive measurements, and if it is possible to maintain performance when scaling up.

The navigation was achieved by using a pathfinder algorithm (BFS) to compute a path, from the path a set of instructions is sent to the prototype. The prototype utilizes its feedback system (reflex detectors) to navigate according to the instructions.

For the markings to differentiate from the ground, several different materials and markings were tested by measuring their reflectivity at a set distance from a reflex detector. When sufficiently differentiating ground (Whiteboard) to marking (electrical tape) measurements were found, several tests were conducted on these.[Table 4.1, Appendix C.1] From these tests it can be concluded that for markings and ground sensor values to differentiate enough from each other, it is recommended that the span of each value should have a clear separation.[Figure 4.2]

The last research question to be answered was about the scalability of the project. In [5.2] we discussed why it was impossible to scale up if all computing was performed on the prototype’s microcontroller, because of its limited hardware. However, we brought up the benefits of a hierarchical software structure which would allow for essentially infinite amount of upscaling with enough funding. The conclusion is that, yes, it is possible to scale this project up.
Chapter 7

Bibliography


[8] Linked list library https://www.arduinolibraries.info/libraries/linked-list (Online: Accessed 05/04-2018)


Appendix A

Source code

A.1 Linked queue (Python)

```python
# Linked Queue class
# Original authors: Alexander Axelsson & Carl Foghammar Nomtak
# Edited by: Alexander Axelsson
# Bachelor project in mechatronics, KTH, 2018
# This class is a linked queue used for storage of data.
# Included functions:
# enqueue: enqueues an object
# dequeue: dequeues the next object in the queue
# isEmpty: checks if the queue is empty or not
# This code is a previously created code re-used as it fit the project. The original code was written by Alexander Axelsson and Carl Foghammar Nomtak in a programming course at KTH.

# Node object: used as object storing object in the linked queue
class Node:
    def __init__(self, value, next):
        self.value = value
        self.next = next

# Linked queue class: Used to enqueue and dequeue node objects
class LinkedQ:
    def __init__(self):
        self._last = None
        self._first = None

# enqueues a value stored in a node object in the queue
def enqueue(self, value):
    if self.isEmpty() == True:
```

29
APPENDIX A. SOURCE CODE

```python
self._last = Node(value, None)
self._first = self._last
else:
    self._last.next = Node(value, None)
    self._last = self._last.next

# dequeues the next node object in the queue
def dequeue(self):
    value = self._first.value
    self._first = self._first.next
    return value

# checks if the queue is empty or not
def isEmpty(self):
    return self._first == None
```

A.2 Breadth first search algorithm (Python)

```python
# Breadth first search program that searches for a path in a limited space with hindrances on select coordinates
#
# Original authors: Alexander Axelsson & Carl Foghammar Nomtak
#
# Edited by: Alexander Axelsson
# Bachelor project in mechatronics, KTH, 2018
#
# Included functions:
# makechildren(node, q): generates adjacent (not diagonal) nodes to the node which is used as an argument if within bounds and not already generated. These are then put in the queue, q, for nodes which are to be evaluated
# notVisited(x, y): Checks if these coordinates have not been visited already and returns True if not and false if it has
# writechain(child): recursive function which prints the path from the start to the goal
# generateTasks(child): generates the list of tasks when the goal is found, child=goal. The parent node’s coordinates is compared with the child’s coordinates recursively to generate tasks
#
# This code is refurbished from a previously created code to fit the project. The original code was written by Alexander Axelsson and
```
Carl Foghammar Nømtak in a programming course at KTH.

```python
from linkedQFile import LinkedQ # imports a custom made linked queue class which is critical for the BFS search
import json # used to dump instructions to a .txt file when BFS is complete

areaBounds = [6, 4]; # this array specifies the "width" and "height" of the area in which the BFS in conducted (boundaries)
VisitedCoordinates = [] # this array is used to store coordinates which have already been put in queue and are not supposed to be visited during the BFS. Append blocked coordinates at the start before generating the path
taskList = [] # The content of this list will be saved to a .txt file for use as instructions to the robot

# Breadth-first search
# Searches each coordinate in a grid until the sought coordinate is found and saves a list of tasks to a text file

# Node class. Contains attributes for coordinates and a parent pointer
class Node:
    def __init__(self, x = 0, y = 0, parent = None):
        self.x = x
        self.y = y
        self.parent = parent

# The makechildren function generates adjacent (not diagonal) nodes to the node which is used as an argument if within bounds and not already generated. These are then put in the queue for nodes which are to be evaluated

def makechildren(node, q):
    # print("makechildren")
    VisitedCoordinates.append([node.x, node.y]) # add current coordinates to list of visited coordinates

    if node.x + 1 <= areaBounds[0] and notVisited(node.x+1, node.y): # if not visited and if the coordinate to the right is within bounds. Create that node and put
        ...
child_node = Node(node.x+1, node.y, node)
q.enqueue(child_node)
VisitedCoordinates.append([node.x+1, node.y])

if node.x -1 >= 0 and notVisited(node.x-1, node.y):
    child_node = Node(node.x-1, node.y, node)
    q.enqueue(child_node)
    VisitedCoordinates.append([node.x-1, node.y])

if node.y + 1 <= areaBounds[1] and notVisited(node.x, node.y + 1):
    child_node = Node(node.x, node.y+1, node)
    q.enqueue(child_node)
    VisitedCoordinates.append([node.x, node.y+1])

if node.y - 1 >= 0 and notVisited(node.x, node.y - 1):
    child_node = Node(node.x, node.y-1, node)
    q.enqueue(child_node)
    VisitedCoordinates.append([node.x, node.y-1])

# Create children adjacent to current node if not already visited
# and within bounds

def notVisited(x, y):
    # Checks if these coordinates have not been
    # visited already and returns True
    # if not and false if it has
    for coordinates in VisitedCoordinates:
        if coordinates[0] == x and coordinates[1] == y:
            return False;
    #print(coordinates[0], " ", coordinates[1])
    return True

def writechain(child):
    # Recursive function which prints the path from
    # the start to the goal
    if child.parent:
        writechain(child.parent)
    print(child.x, " ", child.y)

def generateTasks(child):
    # 1 for up, 2 for left, 3 for down and 4 for
    # right. Directions are relative to

32
A.2. BREADTH FIRST SEARCH ALGORITHM (PYTHON)

```python
def main():
    x_start = int(input("Please enter x_start: "))
    y_start = int(input("Please enter y_start: "))
    x_goal = int(input("Please enter x_goal: "))
    y_goal = int(input("Please enter y_goal: "))
    startNode = Node(x_start, y_start)
    q = LinkedQ()
    q.enqueue(startNode)
    counter = 0;
    while not q.isEmpty() and counter < 500:
        node = q.dequeue()
        # print("Current Node : ", node.x, " ", node.y)
        if node.x == x_goal and node.y == y_goal:
            # if current node is goal node
            print("Found a path: ")
            writechain(node)
            generateTasks(node)
            break
        makechildren(node, q)
        counter += 1
    else:
        print("Did not find a path...")
        #print(VisitedCoordinates[0])
        taskList.append(5) # critical addition to the list of tasks. This
        # instruction means that the goal is reached
        print("List of tasks: ")
        print(taskList)
        # dumps the list of task into a .txt file called "tasks.txt"
        with open("tasks.txt", "w") as outfile:
            json.dump(taskList, outfile)
    if __name__ == "__main__":
        main()
```

```python
if child.parent.parent: # if the child’s parent has a parent
    generateTasks(child.parent)
if child.y > child.parent.y:
    task = 1
if child.x < child.parent.x:
    task = 2
if child.y < child.parent.y:
    task = 3
if child.x > child.parent.x:
    task = 4
    taskList.append(task)

def main():
    x_start = int(input("Please enter x_start: "))
    y_start = int(input("Please enter y_start: "))
    x_goal = int(input("Please enter x_goal: "))
    y_goal = int(input("Please enter y_goal: "))
    startNode = Node(x_start, y_start)
    q = LinkedQ()
    q.enqueue(startNode)
    counter = 0;
    while not q.isEmpty() and counter < 500:
        node = q.dequeue()
        # print ("Current Node : ", node.x, " ", node.y)
        if node.x == x_goal and node.y == y_goal: # if current node is goal node
            print ("Found a path: ")
            writechain(node)
            generateTasks(node)
            break
        makechildren(node, q)
        counter += 1
    else:
        print ("Did not find a path...")
        # print (VisitedCoordinates[0])
        taskList.append(5) # critical addition to the list of tasks. This
        # instruction means that the goal is reached
        print ("List of tasks: ")
        print (taskList)
        # dumps the list of task into a .txt file called "tasks.txt"
        with open ("tasks.txt", "w") as outfile:
            json.dump(taskList, outfile)
    if __name__ == "__main__":
        main()
```
A.3 Main program (Arduino IDE)

/*Control program for a path following robot using IR-light as input

* Alexander Axelsson, Oskar Nordstrom
* Bachelor project in mechatronics, KTH, 2018
* This program does not compute the path from point A to B, that is done in another program written
* in python. This program requires a single list of instructions consisting of numbers from 1 to 4
* which corresponds to directions relative to a grid. The robot uses a orientation variable to keep
* track of which direction it is facing. The robot should start at the start coordinate used when
* calculating the path in the other program.
*/

#include "stdlib.h"
define enA 6 //left motor
define enB 5 //right motor

//Name for pin and corresponding pin on Arduino Uno board that determines drive direction of motor A
#define in1A 11
#define in2A 12

//Name for pin and corresponding pin on Arduino Uno board that determines drive direction of motor A
#define in1B 13
#define in2B 10

int orientation = 4; //Starting direction relative grid
int tasks[] = {4, 4, 1, 1, 1, 4, 4, 3, 3, 5};

int taskNumber = 0; //Which element in the task list that is computed next
int compensate = 0; //Variable used for compensation steering based on previous action the robot has attempted
int taskType; //This is a number which equals to the task on index taskNumber in the list of tasks so the robot knows what to do next
int reflex_pivot_L = 440;
int reflex_pivot_M = 440;
int reflex_pivot_R = 450;
int reflex_pivot_F = 560;

//The Turn functions drives each motor in opposite directions which causes the robot to rotate in place.
//TurnLeft rotates counter clockwise
//TurnRight rotates clockwise

void TurnLeft()
{
  //Direction of motor A and B
  digitalWrite(in1A, HIGH);
digitalWrite(in2A, LOW);
digitalWrite(in1B, LOW);
digitalWrite(in2B, HIGH);
//PWM settings for A and B
analogWrite(enA, 255);
}
A.3. MAIN PROGRAM (ARDUINO IDE)

```c
analogWrite(enB, 255);
}

void TurnRight()
{
    // direction of motor A and B
    digitalWrite(in1A, LOW);
    digitalWrite(in2A, HIGH);
    digitalWrite(in1B, HIGH);
    digitalWrite(in2B, LOW);
    // PWM settings for A and B
    analogWrite(enA, 255);
    analogWrite(enB, 255);
}

// The Turn90 functions uses the turn functions to turn the robot 90 degrees in one direction
// Turn90Left rotates counter clockwise
void Turn90Right()
{
    TurnRight();
    delay(5000);
    int condition = 1;
    while(condition == 1){
        int reflexValueFront = analogRead(A3); // frontal reflex detector
        Serial.println(reflexValueFront);
        if(reflexValueFront <= reflex_pivot_F){
            int possiblyTurned = 0;
            // filter to limit the impact of a few bad readings
            for(int filterCounter = 1; filterCounter <= 5; ++filterCounter){
                reflexValueFront = analogRead(A3); // frontal reflex detector
                if(reflexValueFront <= reflex_pivot_F){
                    ++possiblyTurned;
                }
                delay(10);
            } // for
            if(possiblyTurned == 5){
                Stop();
                condition = 0;
            }
        }
    } // if
}

void Turn90Left()
{
    TurnLeft();
    delay(5000);
    int condition = 1;
    while(condition == 1){
        int reflexValueFront = analogRead(A3); // frontal reflex detector
        Serial.println(reflexValueFront);
        if(reflexValueFront <= reflex_pivot_F){
            int possiblyTurned = 0;
            // filter to limit the impact of a few bad readings
            for(int filterCounter = 1; filterCounter <= 5; ++filterCounter){
                reflexValueFront = analogRead(A3); // frontal reflex detector
                if(reflexValueFront <= reflex_pivot_F){
                    ++possiblyTurned;
                }
                delay(10);
            } // for
            if(possiblyTurned == 5){
                Stop();
                condition = 0;
            }
        }
    } // if
}

// The DriveForward function drives both motors in the same direction
// causing the robot to drive forward
void DriveForward()
{
    // direction of motor A and B
}
```

35
APPENDIX A. SOURCE CODE

digitalWrite(in1A, HIGH);
digitalWrite(in1B, HIGH);
digitalWrite(in2A, LOW);
digitalWrite(in2B, LOW);

//PWM settings for A and B
analogWrite(enA, 255);
analogWrite(enB, 255);
}

//The Reverse function drives both motors in the same direction causing the robot to reverse
void Reverse()
{
  //Direction of motor A and B
  digitalWrite(in1A, LOW);
digitalWrite(in1B, HIGH);
digitalWrite(in2A, LOW);
digitalWrite(in2B, HIGH);
  //PWM settings for A and B
analogWrite(enA, 255);
analogWrite(enB, 255);
}

//The Steer functions steer the robot in a direction as opposed to rotating
void SteerRight()
{
  //Direction of motor A and B
  digitalWrite(in1A, HIGH);
digitalWrite(in1B, HIGH);
digitalWrite(in2A, LOW);
digitalWrite(in2B, LOW);
  //PWM settings for A and B
analogWrite(enA, 255);
analogWrite(enB, 10);
}

void SteerLeft()
{
  //Direction of motor A and B
  digitalWrite(in1A, HIGH);
digitalWrite(in1B, HIGH);
digitalWrite(in2A, LOW);
digitalWrite(in2B, LOW);
  //PWM settings for A and B
analogWrite(enA, 10);
analogWrite(enB, 255);
}

//The Stop function stops all motors from any driving
void Stop()
{
  //Direction of motor A and B
  digitalWrite(in1A, HIGH);
digitalWrite(in1B, HIGH);
digitalWrite(in2A, LOW);
digitalWrite(in2B, LOW);
  //PWM settings for A and B
analogWrite(enA, 0);
analogWrite(enB, 0);
}

//The Align function is used to orient the robot according to the taskType. 1 represents "up"
//relative to the grid, 2 represents left, 3 represents down and 4 represents right.
//Based on the robot's current orientation it will rotate either
counter clockwise, clockwise or do nothing.
void Align()
{
  while(tasks[taskNumber] != orientation)
  {
    Serial.print(orientation);
    Serial.print(taskNumber);
    if (orientation == 1 && tasks[taskNumber] == 4)
      Turn90Right();
    else
      orientation--;
  }

36
A.3. MAIN PROGRAM (ARDUINO IDE)

else if (orientation == 4 && tasks[taskNumber] == 1){
    Turn90Left();
    if (orientation == 4){
        orientation = 1;
    }
    else{
        orientation ++;
    }
}
else if (orientation != 1 && orientation > tasks[taskNumber]){  
    Serial.println("orientation != 1 && orientation > tasks[taskNumber] ");
    Turn90Right();
    if (orientation == 1){
        orientation = 4;
    }
    else{
        orientation --;
    }
}
else if (orientation != 4 && orientation < tasks[taskNumber]){  
    Turn90Left();
    if (orientation == 4){
        orientation = 1;
    }
    else{
        orientation ++;
    }
}
//gives the robot a "nudge" out of the current node and onto the path to the next node
DriveForward();
delay(1000);
Stop();
//Align()

void setup() {
    Serial.begin(9600);
    Serial.println("setup");
    pinMode(2, OUTPUT); //The pin that powers the reflex detector’s LEDs
    pinMode(8, OUTPUT); //The pin that powers the reflex detector’s LEDs
    pinMode(A3, INPUT); //Detects signal from front reflex sensor
    pinMode(A2, INPUT); //Detects signal from reflex sensor 1
    pinMode(A1, INPUT); //Detects signal from reflex sensor 2
    pinMode(A0, INPUT); //Detects signal from reflex sensor 3
    digitalWrite(8, HIGH); //Powers the LEDs on the reflex sensors
    digitalWrite(2, HIGH); //Powers the LED on the frontal reflex sensors
    pinMode(enA, OUTPUT); //PWM for engine A
    pinMode(enB, OUTPUT); //PWM for engine B
    pinMode(in1A, OUTPUT); //Output to decide engine A direction
    pinMode(in2A, OUTPUT); //Output to decide engine B direction
    pinMode(in1B, OUTPUT); //Output to decide engine B direction
    pinMode(in2B, OUTPUT); //Output to decide engine B direction
    delay(1000);
    Align(); //makes sure the prototype always aligns itself before executing the first instruction
    Serial.println("end of setup");
}

void loop() {
    int reflexValueLeft = analogRead(A0); //left reflex detector
    int reflexValueMiddle = analogRead(A1); //middle reflex detector
    int reflexValueRight = analogRead(A2); //right reflex detector
    int reflexValueFront = analogRead(A3); //frontal reflex detector
    taskType = tasks[taskNumber]; //Stores the current Task in variabel taskType
}
APPENDIX A. SOURCE CODE

/* These printouts is for troubleshooting/data collection via the serial monitor
Serial.print(reflexValueLeft);
Serial.print(\t);
Serial.print(reflexValueMiddle);
Serial.print(\t);
Serial.print(reflexValueRight);
Serial.print(\t);
Serial.print(reflexValueFront);
Serial.print(\t);
Serial.print("taskType:orientation:");
Serial.print(taskType);
Serial.print(\t);
Serial.println(orientation);
*/

//If all sensors are off the line, function will reverse until line found on any sensor
if(reflexValueLeft >= reflex_pivot_L &&
   reflexValueMiddle >= reflex_pivot_M &&
   reflexValueRight >= reflex_pivot_R){
   Reverse();
delay(200);
}

if(reflexValueLeft <= reflex_pivot_L &&
   reflexValueMiddle <= reflex_pivot_M &&
   reflexValueRight <= reflex_pivot_R){
   //if all reflex detectors over marking
   int possiblyOnNode = 0;
   //filter to limit the impact of a few bad readings
   for(int filterCounter = 1; filterCounter <= 5; ++filterCounter){
      reflexValueLeft = analogRead(A0); //left reflex detector
      reflexValueMiddle = analogRead(A1); //middle reflex detector
      reflexValueRight = analogRead(A2); //right reflex detector
      if(reflexValueLeft <= reflex_pivot_L &&
         reflexValueMiddle <= reflex_pivot_M &&
         reflexValueRight <= reflex_pivot_R){
         ++possiblyOnNode;
      }
      delay(10);
   }
   //for all reflex detectors over marking for x amount of times in a row
   //this is to increase robustness by reducing the impact of temporary environmental noise
   if(possiblyOnNode == 5){
      Serial.println("Detected\a\node");
      Stop();
delay(700);
      Serial.print(taskType);
taskNumber ++;
      if(tasks[taskNumber] == 5){ //if there are no more tasks
         Stop();
         while(tasks[taskNumber] == 5){
            Serial.print("Goal\a\Reached\a\Succesfully");
         }
      }
      else{
         Align();
      }
   }
   else if(reflexValueRight <= reflex_pivot_R){ //else if the right reflex detector over marking
      SteerRight();
      compensate=1;
   }
   else if(reflexValueLeft <= reflex_pivot_L){ //else if the right left detector over marking
      SteerLeft();
      compensate=2;
   }
   else if(reflexValueMiddle <= reflex_pivot_M){ //else if the middle reflex detector over marking
      //notOnNode:
   }
   else{ //else if reflexValueLeft < reflex_pivot_L &&
      //this is to increase robustness by reducing the impact of temporary environmental noise
   }
}
A.4. DISCARDED NODE.H (ARDUINO IDE)

```c
if (compensate == 1){
    TurnLeft();
    delay(700);
}
if (compensate == 2){
    TurnRight();
    delay(700);
}
DriveForward();
compensate=0;
}
```

// put your main code here, to run repeatedly:

A.4 Discarded Node.h (Arduino IDE)

```c
/* (Discarded) This is a header file for a node class in the arduino IDE

* Alexander Axelsson
* Bachelor project in mechatronics, KTH, 2018
* Node class
* ---------------------
* Node.h
* ********************
* Node class which has the intended use for a breadth first search algorithm
* - it will store coordinates [x,y]
* - it will store a parent pointer
*/

#include <LinkedList.h>
//Followed tutorial: https://www.youtube.com/watch?v=1sFLyfHz9FU
//linked list library from: https://www.arduinolibraries.info/libraries/linked-list
#if !defined NODE_H
#define NODE_H

class Node{
public:
    Node(); //default constructor
    Node(int _x, int _y); //overload for our constructor1
    Node(int _x, int _y, Node* _parent); //overload for our constructor2

    //Node overloaded constructor
    int GetX(); //access the member coordinates outside of this class
    int GetY(); //access the member coordinates outside of this class
    Node* GetParent(); //access the parent of this node outside of this class

    //void Display(); //displays the content in the node
private:
    int coordinates[2]; //store coordinates
    int x;
    int y;
    Node* parent; //this is a pointer to our parent node
}; //Node class
#endif
```

A.5 Discarded Node.cpp (Arduino IDE)

```c
/* (Discarded) .cpp file for Node class in the Arduino IDE

* Alexander Axelsson
* Bachelor project in mechatronics, KTH, 2018
```
APPENDIX A. SOURCE CODE

* This code defines the attributes of a node.
* It contains the attributes of x and y coordinates as well as a parent pointer.
* There are a few overloaded constructors to make it more intuitive to create new
* objects of this type
*
*/

#include 'Node.h'
//followed this tutorial: https://www.youtube.com/watch?v=1sFLyfHz9FU

Node::Node(){ //default constructor
    this->x = 0;
    this->y = 0;
    this->parent = nullptr;
}

Node::Node(int _x, int _y){ //overloaded constructor1
    this->x = _x;
    this->y = _y;
    this->parent = nullptr;
}

Node::Node(int _x, int _y, Node* _parent){ //overloaded constructor2
    this->x = _x;
    this->y = _y;
    this->parent = _parent;
}

int Node::GetX(){ //method of retrieving the x coordinate
    return this->x;
}

int Node::GetY(){ //method of retrieving the y coordinate
    return this->y;
}

Node* Node::GetParent(){ //method of retrieving the parent pointer
    return this->parent;
}
Appendix B

Electrical schematics and datasheets

B.1 Full schematic
B.2. ARDUINO UNO DATASHEET

B.2 Arduino Uno Datasheet
The Arduino Uno SMD R3 is a microcontroller board based on the ATmega328. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip.

Additional features coming with the R3 version are:

- ATmega16U2 instead 8U2 as USB-to-Serial converter.
- 1.0 pinout: added SDA and SCL pins for TWI communication placed near to the AREF pin and two other new pins placed near to the RESET pin, the IOREF that allow the shields to adapt to the voltage provided from the board and the second one is a not connected pin, that is reserved for future purposes.
- stronger RESET circuit.

"Uno" means "One" in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version 1.0 will be the reference versions of Arduino, moving forward. The Uno is the latest in a series of USB Arduino boards, and the reference model for the Arduino platform.
<table>
<thead>
<tr>
<th>Feature</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcontroller</td>
<td>ATmega328P</td>
</tr>
<tr>
<td>Operating Voltage</td>
<td>5V</td>
</tr>
<tr>
<td>Input Voltage (recommended)</td>
<td>7-12V</td>
</tr>
<tr>
<td>Input Voltage (limit)</td>
<td>6-20V</td>
</tr>
<tr>
<td>Digital I/O Pins</td>
<td>14 (of which 6 provide PWM output)</td>
</tr>
<tr>
<td>PWM Digital I/O Pins</td>
<td>6</td>
</tr>
<tr>
<td>Analog Input Pins</td>
<td>6</td>
</tr>
<tr>
<td>DC Current per I/O Pin</td>
<td>20 mA</td>
</tr>
<tr>
<td>DC Current for 3.3V Pin</td>
<td>50 mA</td>
</tr>
<tr>
<td>Flash Memory</td>
<td>32 KB (ATmega328P) of which 0.5 KB used by bootloader</td>
</tr>
<tr>
<td>SRAM</td>
<td>2 KB (ATmega328P)</td>
</tr>
<tr>
<td>EEPROM</td>
<td>1 KB (ATmega328P)</td>
</tr>
<tr>
<td>Clock Speed</td>
<td>16 MHz</td>
</tr>
<tr>
<td>LED_BUILTIN</td>
<td>13</td>
</tr>
<tr>
<td>Length</td>
<td>68.6 mm</td>
</tr>
<tr>
<td>Width</td>
<td>53.4 mm</td>
</tr>
<tr>
<td>Weight</td>
<td>25 g</td>
</tr>
</tbody>
</table>
B.3 Maxon 226754 DC Motor
maxon RE-max

RE-max 29  ø29 mm, Precious Metal Brushes CLL, 15 Watt

M 1:2

Stock program
Standard program
Special program (on request)

Motor Data

<table>
<thead>
<tr>
<th>Values at nominal voltage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  Nominal voltage V</td>
</tr>
<tr>
<td>6320 7020 6010 5490 5810</td>
</tr>
<tr>
<td>2  No load speed rpm</td>
</tr>
<tr>
<td>43 41.7 23.8 18.9 13.4</td>
</tr>
<tr>
<td>3  No load current mA</td>
</tr>
<tr>
<td>9.45 9.01 7.87 6.1 4.65</td>
</tr>
<tr>
<td>4  Nominal speed rpm</td>
</tr>
<tr>
<td>6080 6710 5430 6020 4700</td>
</tr>
<tr>
<td>5  Nominal torque (max. continuous torque) mNm</td>
</tr>
<tr>
<td>8.67 9.78 15.6 21.2 25.9</td>
</tr>
<tr>
<td>6  Nominal current (max. continuous current) A</td>
</tr>
<tr>
<td>0.84 0.84 0.84 0.84 0.84</td>
</tr>
<tr>
<td>7  Stall torque mNm</td>
</tr>
<tr>
<td>200 205 156 219 177 190</td>
</tr>
<tr>
<td>8  Stall current A</td>
</tr>
<tr>
<td>18.4 16.8 8.22 8.49 5.68</td>
</tr>
<tr>
<td>9  Max. efficiency %</td>
</tr>
<tr>
<td>91 91 90 91 91 91 91</td>
</tr>
</tbody>
</table>

Characteristics

| Terminal resistance Q |
| 0.39 0.536 1.48 |
| 11  Terminal inductance mH | 0.0353 0.0447 0.108 0.199 0.292 |
| 12  Torque constant mNm/A | 10.9 12.2 19 |
| 13  Speed constant rpm/V | 879 781 502 370 306 242 |
| 14  Speed / torque gradient rpm/mNm | 31.6 34.3 38.6 |
| 15  Mechanical time constant ms | 4.77 4.63 4.42 |
| 16  Rotor inertia gcm² | 14.4 12.9 10.9 13.6 13.3 13.1 |

Operating Range

<table>
<thead>
<tr>
<th>n (rpm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>22756</td>
</tr>
<tr>
<td>11000</td>
</tr>
<tr>
<td>0.1 - 0.2 mm</td>
</tr>
<tr>
<td>0.012 mm</td>
</tr>
<tr>
<td>5 N</td>
</tr>
<tr>
<td>0.1 - 0.2 mm</td>
</tr>
<tr>
<td>0.025 mm</td>
</tr>
<tr>
<td>75 N</td>
</tr>
<tr>
<td>20.5 N</td>
</tr>
</tbody>
</table>

Thermal data

| Thermal resistance housing-ambient |
| 15.8 K/W |
| 18 | Thermal resistance winding-ambient |
| 4.0 K/W |
| 19 | Thermal time constant winding |
| 15.4 s |
| 20 | Ambient temperature |
| -30...+65°C |
| 21 | Max. winding temperature |
| +85°C |

Mechanical data (sleeve bearings)

| Max. speed 11000 rpm |
| 24 | Axial play 0.1 - 0.2 mm |
| 25 | Radial play 0.012 mm |
| 26 | Max. axial load (dynamic) 5 N |
| 27 | Max. force for press fits (static) 80 N |
| 28 | Max. radial load, 5 mm from flange 5.5 N |

Mechanical data (ball bearings)

| Max. speed 11000 rpm |
| 23 | Axial play 0.1 - 0.2 mm |
| 24 | Radial play 0.025 mm |
| 25 | Max. axial load (dynamic) 5 N |
| 26 | Max. force for press fits (static) 75 N |
| 27 | Max. radial load, 5 mm from flange 20.5 N |

Other specifications

| Number of pole pairs | 1 |
| Number of commutator segments | 13 |
| Weight of motor | 159 g |

RPM values listed in the table are nominal. Explanation of the figures on page 151.

Planetary Gearhead Ø32 mm

| Planetary Gearhead Ø32 mm |
| 1.0 - 6.0 Nm |
| Page 343 |

Spindle Drive Ø32 mm

| Spindle Drive Ø32 mm |
| Page 370-372 |

Maxon Modular System

Overview on page 20–27

Continuous operation

In observation of above listed thermal resistance (lines 17 and 18) the maximum permissible winding temperature will be reached during continuous operation at 25°C ambient.

= Thermal limit.

Short term operation

The motor may be briefly overloaded (recurring).

Assigned power rating

Ball bearings in place of sleeve bearings

Pigtails in place of terminals

Without CLL

maxon DC motor 227
Appendix C

Testing results

C.1 Table of measurements

This is only an example of measurements presented in frequency tables. Due to the amount of measurements (in the range of 40 000 measurements) it is not feasible to include all of them in the appendix. All measurements are therefore available upon request to authors.
### Figure C.1. Example frequency table used for frequency diagrams

<table>
<thead>
<tr>
<th>Left</th>
<th>Frequency</th>
<th>Middle</th>
<th>Frequency</th>
<th>Right</th>
<th>Frequency</th>
<th>Front</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>20</td>
<td>0</td>
<td>20</td>
<td>0</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>40</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td>60</td>
<td>0</td>
<td>60</td>
<td>0</td>
<td>60</td>
<td>0</td>
<td>60</td>
<td>0</td>
</tr>
<tr>
<td>80</td>
<td>0</td>
<td>80</td>
<td>0</td>
<td>80</td>
<td>0</td>
<td>80</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>120</td>
<td>0</td>
<td>120</td>
<td>1</td>
<td>120</td>
<td>0</td>
<td>120</td>
<td>0</td>
</tr>
<tr>
<td>140</td>
<td>0</td>
<td>140</td>
<td>0</td>
<td>140</td>
<td>0</td>
<td>140</td>
<td>0</td>
</tr>
<tr>
<td>160</td>
<td>1</td>
<td>160</td>
<td>1</td>
<td>160</td>
<td>1</td>
<td>160</td>
<td>1</td>
</tr>
<tr>
<td>180</td>
<td>1</td>
<td>180</td>
<td>3</td>
<td>180</td>
<td>1</td>
<td>180</td>
<td>3</td>
</tr>
<tr>
<td>200</td>
<td>3</td>
<td>200</td>
<td>11</td>
<td>200</td>
<td>5</td>
<td>200</td>
<td>1</td>
</tr>
<tr>
<td>220</td>
<td>8</td>
<td>220</td>
<td>79</td>
<td>220</td>
<td>2</td>
<td>220</td>
<td>2</td>
</tr>
<tr>
<td>240</td>
<td>32</td>
<td>240</td>
<td>266</td>
<td>240</td>
<td>3</td>
<td>240</td>
<td>3</td>
</tr>
<tr>
<td>260</td>
<td>89</td>
<td>260</td>
<td>556</td>
<td>260</td>
<td>42</td>
<td>260</td>
<td>2</td>
</tr>
<tr>
<td>280</td>
<td>142</td>
<td>280</td>
<td>88</td>
<td>280</td>
<td>156</td>
<td>280</td>
<td>10</td>
</tr>
<tr>
<td>300</td>
<td>278</td>
<td>300</td>
<td>19</td>
<td>300</td>
<td>125</td>
<td>300</td>
<td>11</td>
</tr>
<tr>
<td>320</td>
<td>161</td>
<td>320</td>
<td>18</td>
<td>320</td>
<td>116</td>
<td>320</td>
<td>28</td>
</tr>
<tr>
<td>340</td>
<td>39</td>
<td>340</td>
<td>31</td>
<td>340</td>
<td>122</td>
<td>340</td>
<td>126</td>
</tr>
<tr>
<td>360</td>
<td>17</td>
<td>360</td>
<td>52</td>
<td>360</td>
<td>72</td>
<td>360</td>
<td>389</td>
</tr>
<tr>
<td>380</td>
<td>7</td>
<td>380</td>
<td>5</td>
<td>380</td>
<td>9</td>
<td>380</td>
<td>202</td>
</tr>
<tr>
<td>400</td>
<td>6</td>
<td>400</td>
<td>9</td>
<td>400</td>
<td>3</td>
<td>400</td>
<td>71</td>
</tr>
<tr>
<td>420</td>
<td>11</td>
<td>420</td>
<td>6</td>
<td>420</td>
<td>3</td>
<td>420</td>
<td>30</td>
</tr>
<tr>
<td>440</td>
<td>8</td>
<td>440</td>
<td>3</td>
<td>440</td>
<td>2</td>
<td>440</td>
<td>19</td>
</tr>
<tr>
<td>460</td>
<td>5</td>
<td>460</td>
<td>5</td>
<td>460</td>
<td>1</td>
<td>460</td>
<td>9</td>
</tr>
<tr>
<td>480</td>
<td>4</td>
<td>480</td>
<td>7</td>
<td>480</td>
<td>1</td>
<td>480</td>
<td>6</td>
</tr>
<tr>
<td>500</td>
<td>5</td>
<td>500</td>
<td>7</td>
<td>500</td>
<td>4</td>
<td>500</td>
<td>2</td>
</tr>
<tr>
<td>520</td>
<td>13</td>
<td>520</td>
<td>15</td>
<td>520</td>
<td>2</td>
<td>520</td>
<td>4</td>
</tr>
<tr>
<td>540</td>
<td>1419</td>
<td>540</td>
<td>1067</td>
<td>540</td>
<td>1523</td>
<td>540</td>
<td>3</td>
</tr>
<tr>
<td>560</td>
<td>0</td>
<td>560</td>
<td>0</td>
<td>560</td>
<td>56</td>
<td>560</td>
<td>2</td>
</tr>
<tr>
<td>580</td>
<td>0</td>
<td>580</td>
<td>0</td>
<td>580</td>
<td>0</td>
<td>580</td>
<td>3</td>
</tr>
<tr>
<td>600</td>
<td>0</td>
<td>600</td>
<td>0</td>
<td>600</td>
<td>0</td>
<td>600</td>
<td>2</td>
</tr>
<tr>
<td>620</td>
<td>0</td>
<td>620</td>
<td>0</td>
<td>620</td>
<td>0</td>
<td>620</td>
<td>2</td>
</tr>
<tr>
<td>640</td>
<td>0</td>
<td>640</td>
<td>0</td>
<td>640</td>
<td>0</td>
<td>640</td>
<td>1</td>
</tr>
<tr>
<td>660</td>
<td>0</td>
<td>660</td>
<td>0</td>
<td>660</td>
<td>0</td>
<td>660</td>
<td>4</td>
</tr>
<tr>
<td>680</td>
<td>0</td>
<td>680</td>
<td>0</td>
<td>680</td>
<td>0</td>
<td>680</td>
<td>4</td>
</tr>
<tr>
<td>700</td>
<td>0</td>
<td>700</td>
<td>0</td>
<td>700</td>
<td>0</td>
<td>700</td>
<td>6</td>
</tr>
<tr>
<td>720</td>
<td>0</td>
<td>720</td>
<td>0</td>
<td>720</td>
<td>0</td>
<td>720</td>
<td>1307</td>
</tr>
</tbody>
</table>
C.2 Frequency Diagrams

![Frequency diagram of readings from middle reflex detector during a test run](image)

**Figure C.2.** Frequency diagram of values middle reflex detector
APPENDIX C. TESTING RESULTS

Figure C.3. Frequency diagram of values right reflex detector
Figure C.4. Frequency diagram of values right reflex detector
Figure C.5. Frequency diagram of values right reflex detector
C.2. FREQUENCY DIAGRAMS

Figure C.6. Frequency diagram of values front reflex detector
Figure C.7. Frequency diagram of values front reflex detector