|  |
| --- |
|  |
| **Autonomous Maze Mapping and Running Rover** |
| Operation Systems and Embedded Linux, Parallel Programming |
|  |
| **Team 3 / IOSLX4, IPARP4** |
| **5/1/13** |

**Supervisors:**

Ole Rydahl, Henning Haugaard

**Student name Student number Signature**

Johannes Jørgensen 120428

Benjamin Prakash 100100

Ivo Drlje 101108

Daniel Machon 092039

Table of contents

1. Introduction 4

1.1 Project start 4

1.2 Problem formulation 4

1.3 Milestone plan 4

2. Problem analysis 4

2.1 Requirements 5

2.2 Analysis 5

3. Project delimitation and methods 5

3.1 Proposed solution strategy 6

3.2 Theory and methods (Technologies) 7

3.2.1 Motor 7

3.2.1.1 Stepper Motor 7

3.2.1.2 DC Motor 7

3.2.2 Auto correction input 9

3.2.2.1 Odometry 9

 Mouse 9

 Encoder 9

3.2.2.2 Sensors 9

3.2.3 Auto correction 11

3.2.3.1 PID 11

3.2.3.2 Iterative 12

3.2.4 Path finding 12

3.2.4.1 Dijkstra 13

3.2.4.2 Flood fill 14

3.2.5 Communication 15

3.2.5.1 Zeroconf 15

3.2.5.2 Tcp 15

3.2.6 Mapping 15

3.2.6.1 Coordinate system 15

3.2.6.2 Depth first search 15

3.2.6.3 Stack 15

3.2.6.4 Graph 15

3.3 Resources 16

4. Problem solution 16

5. Conclusion 16

5.1 Process assessment 16

6. Bibliography 16

7. GlossaryAppendices 16

Appendix 1. Milestone plan 20

Appendix 2. Action item list 20

Appendix 3. Usecase specifications 20

Appendix 4. GUI layout 20

Appendix 5. Overall block diagram 20

Appendix 6. Unit tests 20

Appendix 7. Usecase diagram 20

Appendix 8. User Manual 20

Appendix 9. Video demo 20

Preface

# Introduction

## Project start

This paper documents the development process of the 4th semester project “**Autonomous maze mapping and running rover**”, which is an interdisciplinary project, involving the courses ‘Parallel programming’ and ‘Operating systems and embedded Linux’.

The project comprises the construction of an autonomous rover, using available hardware parts provided by the teachers, along with the development of a software solution to control the rover and map the maze. The software solution also includes the implementation of a client computer, that will request the mapped maze, and show it graphically.

Given a multitude of different technologies, software strategies and general development approaches, we as a group will discuss and compare these, to delimitate our project solution and present what we believe is the engineerically correct/best solution to our project requirements- and formulation.

## Problem formulation

## Milestone plan

The milestone plan is enclosed in the appendices: see Appendix 1

# Problem analysis

## Requirements

|  |  |
| --- | --- |
| **Functional Requirements** | |
| **R1** | The Rover must be able to map any maze. |
| **R2** | On the basis of the mapping done. The rover moves from the specified start position to the target position as fast as possible. |
| **R3** | The rover must communicate with a central computer (laptop) before and after the maze discovery, as well as before and after the fast traversal of the known maze. |
| **R4** | After your rover has done the mapping the information is transferred to a PC. The maze is then displayed graphically. |
| **R5** | The rover must be equipment with start/stop button |
| **R6** | For a known maze a sequence of orders should be stored in the rover. The rover then executes those orders moving it as fast as possible from a specified start position to a specified target position. |
| **R7** | During mapping and traversal the rover should record a log of sensor readings and motor orders. |
| **R8** |  |
| **R9** |  |

|  |  |
| --- | --- |
| **Non Functional Requirements** | |
| **R1** | The robot must use 2 stepper motor. |
| **R2** | The robot must use a Raspberry Pi model B for controlling the unit. |
| **R3** |  |

## Analysis

# Project delimitation and methods

## Proposed solution strategy

We plan to structure our development process, around maintaining a weekly ‘week description’ which summarizes and documents all project activities of the week. This week description will keep track of project- and code development as well as time spent on different problems, in order to backtrack and pinpoint badly selected choices.

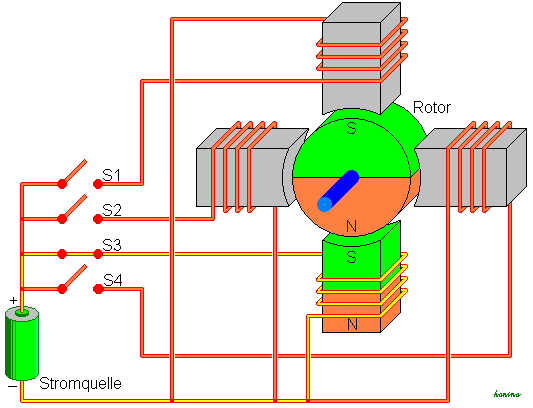
**Maybe something about the use of Git here**

## Theory and methods (Technologies)

### Motor

To make the robot go forwards, make turns or drive backwards, we have to make some decisions on what kind of electrical motors we want to use. In the following section the theory behind the motors will be explained and compared.

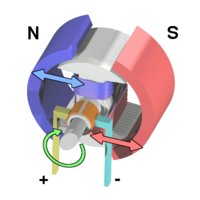
#### Stepper Motor

A stepper motor consist of a minimum of 4 iron cores. These iron cores are entangled in copper wire and by supplying electricity to these core a magnetic field is created. This is also called a Coil-magnet setup. In the middle of all the cores a shaft is placed. This shaft is magnetic and therefore has South and a North pole. When adding electricity to one of the coils in the shaft will align its North pole with the coils South pole. By alternating between witch coil gets power the shaft will start tuning. By only tuning one coil on at the time the shaft will preformed a Full-Step rotation. This will make the shaft turn 90 degrees each time the next coil is activated. Turning on 2 coil, 1 coil, 2 coil, 1 coil etc, the motor will generate a Haft-Step rotation and the shaft will turn 45 degrees on each impulse. This makes the increments much finer when the shaft is rotation and therefore more accurate. All these diffent combination of turning on coils is called StepModes.

To control this switching of electricity impulses to each coil, a motor controller is needed. The motor controller controls witch coil needs power when, to performe the requested rotation.

<http://da.wikipedia.org/wiki/Step-motor>

#### DC Motor

A DC motor uses almost the same setup internally as a stepper motor, but the magnet-shaft and coils are switch and there are in most cases only 2 Coils. This means that the magnet are on the "outside" and the coil is on the "inside". (See the illustration). A DC motor receives direct current, and uses brushes located on the shaft to switch the polarity's. The switching of poles happens internally inside the DC motor

and therefore in most cases a motor controller is not needed to make it work. '

Due to the lack of a motor controller most DC motor, is more responsive then a stepper motor and also accelerate faster.

<http://en.wikipedia.org/wiki/Brushed_DC_electric_motor>

**The difference**

Both DC motor and a Stepper motor has their strong and weak sides. To regulate a Stepper motor speed, acceleration, torque and "travel-distance" an Motor controller is needed. This motor controller provides also feedback on how far the motor have traveled. A DC motor's speed is controlled on the voltage it receives, and unless a encoder haven't been build in the motor, there is no way to know accurate how far the motor have turned.

<http://www.ehow.com/facts_6777501_dc-motor-vs_-stepper-motor.html>

### Auto correction input

#### Odometry

Odometry is the collection of data from actuators, encoders or sensors, in order to estimate a position relative to the starting position.

#### Mouse

The idea of using mice odometry to estimate position is an idea we have had since the beginning of the project. Two mice are used to input relative coordinate-changes to our RPi whenever a displacement of the robot has happened. Several mathematical calculations are done based on the given x,y coordinate pair, and ultimately an length and angle of the movement is calculated. These two values are then used to attempt to estimate the change in position over time, and fed to a PID controller- calculation-algorithm to minimize heading errors, respectively**[[1]](#footnote-1)**.

#### Wheel Encoders

Another alternative to using mouse was the idea of using wheel encoders to determinate distance and direction. Setup on the robot is done by mounting gray-code Ref til et appendix med et gray code wheel to each wheel, with an sensor to read the bit-pattern. When the wheels are turning the sensors will recognize changes in the bit pattern and a position of the wheels can be determinate.

#### Sensors

The robot must be able to determine distances to the surrounding walls, in order to maintain a heading and know when to make a turn. Two primary options are given in this context; Infra-red sensors or Ultrasonic-sensors (or a combination of both). Either choice will work on a robot, but have different impacts on how the robot will function when running a maze. We have to look at the two sensor specifications and the robot behavior requirements, to make a satisfying engineerical choice

* **IR Sensor**

The infra-red sensor (which in our case is a GPD120x Sharp sensor) uses triangulation and CCD array to calculate a distance to objects. It works by emitting pulses of infra-red light, which in case of a present object, will be reflected and caught by the detector. This will create a triangle between the emitter, detector and point of reflection. The distance of the reflecting object will determine the angle in this triangle, which is then used by the CCD array to calculate a distance to the object.

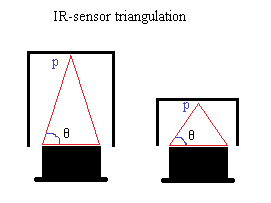


Figure 1: Shows the IR-sensor triangulation, where ‘p’ is the point of reflection and ‘theta’ is the angle created

The sensor output is non-linear with respect to the distance, so it is not insignificant where the sensors are mounted on the robot. Based on the graph of the analog voltage as a function of the distance, one will have to make decision of an optimal reading interval**[[2]](#footnote-2)**.

* **Ultrasonic sensor**

The ultrasonic sensor works differently from the IR-sensor, by emitting inaudible sound to detect surrounding objects, instead of light. We know that sound travels at a certain speed in air, so by keeping track of time elapsed since the sound was emitted and until the echo is detected, a very accurate distance can be calculated.

The sound emitted by the ultrasonic sensors spread out radially, which can cause problems, if several ultrasonic sensors are used, and not placed properly on the robot. If the sensors are pinging too close to each other, one sensor may read another sensors echo as its own. Furthermore materials and object angles might cause other problems such as lost echoes, or ghost echoes.

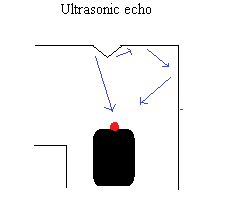


Figure 2: Shows ultrasonic sensor echoes. One echo reflects fine on the wall, the other turns into a ghost echo as a result of the angle on the wall

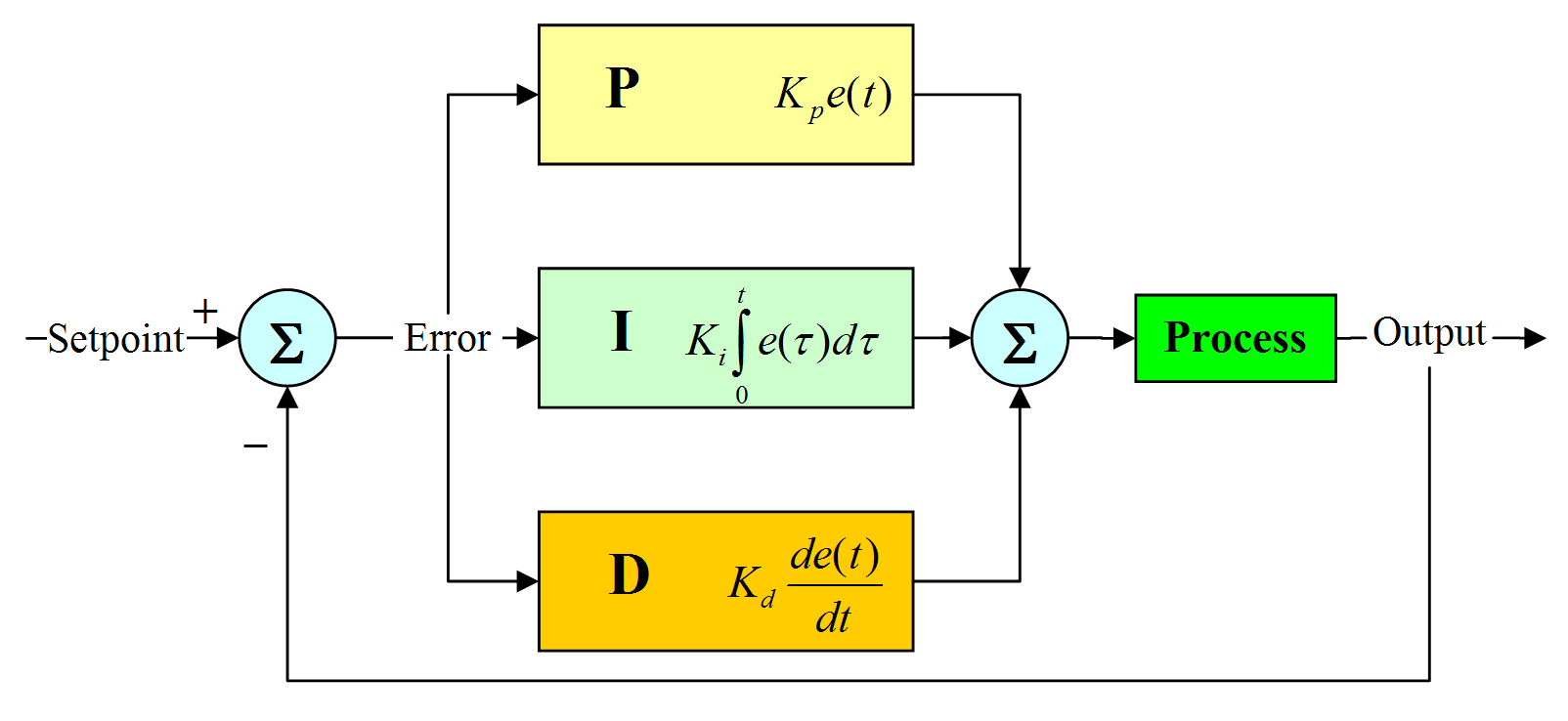
### Auto correction

#### PID

A Proportional – Integral – Derivative controller is a feedback controller, this means that it uses feedback response to determinate its output. A PID Controller calculates an error value in form of a difference from an input to a reference value.

Readouts from the Process block are added to the Setpoint value. From this sum an error will occur, it can be positive or negative. This error is past into the 3 blocks, P, I, and D block. All these values from the outputs are summed together after they have been multiplied with their individually gain factors Kp,Ki and Kd. The sum is then past into the Process. This loop is repeated until the Possess is terminated. (Look below for illustration)

|  |  |  |
| --- | --- | --- |
| P = Current error | I = The sum of previously errors | D = Estimat around future errors |



#### Iterative

The iterative solution consists of moving cell by cell and doing trigonometry calculations. The calculations is made when the robot have moved to its next target no calculations is made on ongoing movement. The robot moves from its current position to the middle of the next cell. First when it has reached its target the robot will first turn so it will be facing straight down its path, the degrees it needs to turn is calculated with help of the width of cell and the length of the distance read from the sensors + the distance from the sensors to the middle of the robot. Secondly the distance it moves each time is calculated with help of trigonometry by knowing the distance of a whole cell and the distance from half of the width – new distance from sensor + the distance from sensor to the middle of the robot. Finally it will find the degrees it needs to turn to be facing its next target; this is done by taking acos((half of the width – (new distance from sensor + the distance from sensor to the middle of the robot))/( the distance it needs to move)). There are two different scenarios depending on if the robots movement last going to the left or right.

### Path finding

Path finding is the process of finding a way between a source and a target. In computer science path finding is normally associated with finding the optimized path from A to B in terms of cost.

Path finding algorithms operate in a system of nodes connected by edges.

For easy understanding a node can be a station, an edge can be a transport route and the cost can be the time it takes to get from station A to station B. The system is described as a graph.

In such a system terminal cost from station A to station B can vary after the chosen path. You can choose to take route that goes near the harbour for fresh air but will end up with extra minutes used on the bus if the fastest route dictates not getting near the harbour.

In path finding the relationship between nodes is often described as parent <-> child.

In such a way the cost between a parent and a child describes the cost to move on in the process. Where the cost between a child and its parent can describe the memory of how much it cost to get to your present position.[[3]](#footnote-3)

A lot of different algorithms for path finding exist.

Important characteristics of algorithms can be divided in two.

1. Greedy:  
   Greedy algorithms that doesn't remember cost. They look ahead and make choices based on distance to target as well as cost to nodes.
2. Informed:  
   Informed algorithms which takes the cost until present position into account when making choices about future travel. On top of that they can have the characteristics of greedy algorithms with heuristics and cost to next coming nodes.

#### Dijkstra

Dijkstra finds the minimum cost to all nodes from its origin, which can be done for all nodes as its origin.

Dijkstra algorithm is often used for routing as it searches all paths.

If negative edges are used it's not possible to guaranteed the shortest path.

Dijkstra works by defining a source node and a target node. The source node sets a cost to 0 and all else nodes get a cost of infinity.

Dijkstra uses a priority queue to hold info on cost to nodes from source node.

The priority queue is prioritized with minimum cost as highest priority.

Dijkstra iterates through nodes starting with the source node and expanding with its neighbouring nodes.

While iterating it decides between investigating a neighbouring node or going back and investigating another branch defined in previous iterations.

Reconstruction of a path is based on each node having a pointer to its previous node.

At the initialization all nodes have their previous node set to null.

In this way a path can be reconstructed in the end by starting with the target node and iterating through previous nodes until previous node is set to null.

Dijkstra is mainly used for situations where the knowledge of cost to all nodes is needed. This is due to Dijkstra lacking direction in its search.

#### Flood fill

Flood fill can be used to scan an area and check for connections between nodes or for path finding. By flooding the area from a point (start node) to all other nodes you will be able to map the whole area. With path finding you can find each neighbour to the current node and mark it as example where you start will be node number 1 every node that can be connected to this node will be node number 2 then you will investigate node number 2 and mark all possible connections to those as number 3 and continue till you find them all then by numbering you will be able to find the cheapest path to your chosen target and use that path to follow

### Communication

#### Zeroconf

#### Tcp

### Mapping

#### Coordinate system

#### Depth first search

Depth first search (DFS) is used for traversing or searching graph. it marks one node as its start/root and then it explores one of the branches all the way till it cant go any further then it will start backtracking till it finds a unexplored branch and goes all the way to the end and will repeat this till it have explored evry possibility the order which it choses it priority to screach is choosen by the user of the DFS. you can go left right and straight in which ever you chose to to priorities you set.

#### Stack

#### Graph

A graph refers to a set of nodes (cells) with edges (connection between nodes) connecting.

An undirected graph is a graph where there is no difference between the direction for the edges meaning that from node A to node B is the same, as from node B to node A.

A directed graph is a graph where a edges are not the same meaning that it got a direction from node A to node B, is not the same edge as from node B to node A and its possible to go one way only.

An edge can be weighted meaning it will have a cost (distance/time/price).

The most used ways to store a graph are adjacency matrix and adjacency list.

Adjacency matrix is a matrix that is representing which nodes of a graph are connected together with other nodes. The matrix is n x n where n = nodes. A matrix structure provides faster access but is consuming a lot of memory.

Adjacency list is a list of unordered collection of a graph one list for each node in the graph, each list shows a set of the nodes neighbours. List structures takes more time to access then a matrix but most cases uses less memory.

If the concern is memory then with a dense graph a matrix would be better than a list, but if it’s a sparse graph the list is better option as less memory is wasted.

## Resources

**Hardware:**

**Software:**

# Problem solution

# Conclusion

## Process assessment

# Bibliography

# Glossary

# Appendix 10: RPi-TMC222 baud rate Documentation

In order to maintain a continuous movement with the robot, we need to make sure the actual position of the robot, never reaches the target position. This is accomplished by reading certain registers in the TMC222 chip. Our getFullStatus2() method provides us with this information, in the form of:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Address | ActPos. | TagPos. | SecPos. | SecPos. | NA |

During our testing of getting this data, we have discovered huge variations of the information we receive, as well as complete useless return data. We suspected that this could be the result of wrong/inefficient baud rate of the I2C bus.

We have conducted three tests that show 40 status readings of the TMC222 chip, while the robot moves towards a target position. In each reading, the I2C baud rate is set differently. Also in each reading, the target position is set to 32000 micro-steps, followed by 25 intervals of 2500 micro-steps. Readings are done in a 1 second interval.

**First test: Baud rate = 100000**

**Expectation: We would expect ActPos to increase from 0 to 32000, and then increasing by 2500 until it overflows the TagPos limit of  and starts over. Also Address, SecPost, and NA should always stay the same (0xE0, 0x00, 0xFC and 0xFF).**

**Results:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Address | ActPos. | TagPos. | SecPos. | SecPos. | NA |

0xe0 32896 64896 0x80 0xfe 0xff

0xe0 34996 48895 0xff 0xff 0xff

0xe0 37859 64896 0x80 0xfc 0xff

0xe0 41094 64896 0x80 0xfc 0xff

0xe0 44453 64896 0x80 0xff 0xff

0xe0 48318 64896 0x80 0xff 0xff

0xe0 50565 64896 0x80 0xfc 0xff

0xe0 53171 64896 0x80 0xfe 0xff

0xe0 56534 64896 0xff 0xff 0xff

0xe0 59893 64896 0x80 0xfc 0xff

0xe0 63630 64896 0xff 0xff 0xff

0xf0 65535 65535 0xff 0xff 0xff

0xe0 64896 64896 0x80 0xfc 0xff

0xe0 64896 64896 0x80 0xfc 0xff

0xe0 64896 64896 0x80 0xfe 0xff

0xe0 64896 64896 0x80 0xff 0xff

0xf0 65535 65535 0xff 0xff 0xff

0xe0 37000 37000 0x80 0xfe 0xff

0xe0 39628 39628 0x80 0xff 0xff

0xf0 65535 65535 0xff 0xff 0xff

0xe0 44500 44500 0x80 0xfc 0xff

0xe0 47000 56319 0xff 0xff 0xff

0xe0 49628 49628 0x80 0xff 0xff

0xf0 65535 65535 0xff 0xff 0xff

0xe0 54500 54500 0x80 0xfc 0xff

0xe0 57000 57000 0x80 0xff 0xff

0xe0 59628 59628 0x80 0xfe 0xff

0xe0 62128 62128 0x80 0xfc 0xff

0xe0 64500 64500 0x80 0xfe 0xff

0xe0 34232 34268 0xff 0xff 0xff

0xe0 36860 36860 0x80 0xfc 0xff

0xe0 39360 39360 0x80 0xfc 0xff

0xe0 41860 41860 0x80 0xfe 0xff

0xe0 44232 38655 0xff 0xff 0xff

0xf0 65535 65535 0xff 0xff 0xff

0xe0 49360 49360 0x80 0xfc 0xff

0xe0 51860 51860 0x80 0xfe 0xff

0xe0 54232 43519 0xff 0xff 0xff

0xf0 65535 65535 0xff 0xff 0xff

0xe0 59360 59360 0x80 0xfe 0xff

**The test shows that all received data is complete rubbish and in no way useable to achieve a continuous movement.**

**Second test: Baud rate = 375000**

**Expectation: We would expect ActPos to increase from 0 to 32000, and then increasing by 2500 until it overflows the TagPos limit of  and starts over. Also Address, SecPost, and NA should always stay the same (0xE0, 0x00, 0xFC and 0xFF).**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Address | ActPos. | TagPos. | SecPos. | SecPos. | NA |

0xe0 0 32000 0x0 0xfc 0xff

0xe0 2222 32000 0x0 0xfc 0xff

0xe0 4951 32000 0x0 0xfc 0xff

0xe0 8179 32000 0x0 0xfc 0xff

0xe0 11656 32000 0x0 0xfc 0xff

0xe0 15384 32000 0x0 0xfc 0xff

0xe0 17622 32000 0x0 0xfc 0xff

0xe0 20350 32000 0x0 0xfc 0xff

0xe0 23578 32000 0x0 0xfc 0xff

0xe0 27056 32000 0x0 0xfc 0xff

0xe0 30783 32000 0x0 0xfc 0xff

0xe0 32000 32000 0x0 0xfc 0xff

0xe0 32000 32000 0x0 0xfc 0xff

0xe0 32000 32000 0x0 0xfc 0xff

0xe0 32000 32000 0x0 0xfc 0xff

0xe0 32000 32000 0x0 0xfc 0xff

0xe0 34500 34500 0x0 0xfc 0xff

0xe0 37000 37000 0x0 0xfc 0xff

0xe0 39500 39500 0x0 0xfc 0xff

0xe0 42000 42000 0x0 0xfc 0xff

0xe0 44500 44500 0x0 0xfc 0xff

0xe0 47000 47000 0x0 0xfc 0xff

0xe0 49500 49500 0x0 0xfc 0xff

0xe0 52000 52000 0x0 0xfc 0xff

0xe0 54500 54500 0x0 0xfc 0xff

0xe0 57000 57000 0x0 0xfc 0xff

0xe0 59500 59500 0x0 0xfc 0xff

0xe0 62000 62000 0x0 0xfc 0xff

0xe0 64500 64500 0x0 0xfc 0xff **Overflow!**

0xe0 1464 1464 0x0 0xfc 0xff

0xe0 3964 3964 0x0 0xfc 0xff

0xe0 6464 6464 0x0 0xfc 0xff

0xe0 8964 8964 0x0 0xfc 0xff

0xe0 11464 11464 0x0 0xfc 0xff

0xe0 13964 13964 0x0 0xfc 0xff

0xe0 16464 16464 0x0 0xfc 0xff

0xe0 18964 18964 0x0 0xfc 0xff

0xe0 21464 21464 0x0 0xfc 0xff

0xe0 23964 23964 0x0 0xfc 0xff

0xe0 26464 26464 0x0 0xfc 0xff

**The test shows that with a baudrate of 375000, the robot behaves as expected, and returns correct information.**

In conclusion, we can derive that the serial communication between the robot and our raspberry pi, won’t work with a low baud rate. My idea is that with a low baud rate, we cannot read the data from the TMC222 registers fast enough, and as a result the register overflow and return corrupt information.

# Appendices

1. Milestone plan
2. Action item list
3. Usecase specifications
4. GUI layout
5. Overall block diagram
6. Unit tests
7. Usecase diagram
8. User Manual
9. Video demo

**Appendix 10. RPi-TMC222 Baud rate Documentatiion**

1. The complete mice odometry documentation is enclosed in the appendices [↑](#footnote-ref-1)
2. Full voltage output documentation is enclosed in the appendices [↑](#footnote-ref-2)
3. <http://cs.stackexchange.com/questions/553/how-do-common-pathfinding-algorithms-compare-to-human-process> [↑](#footnote-ref-3)