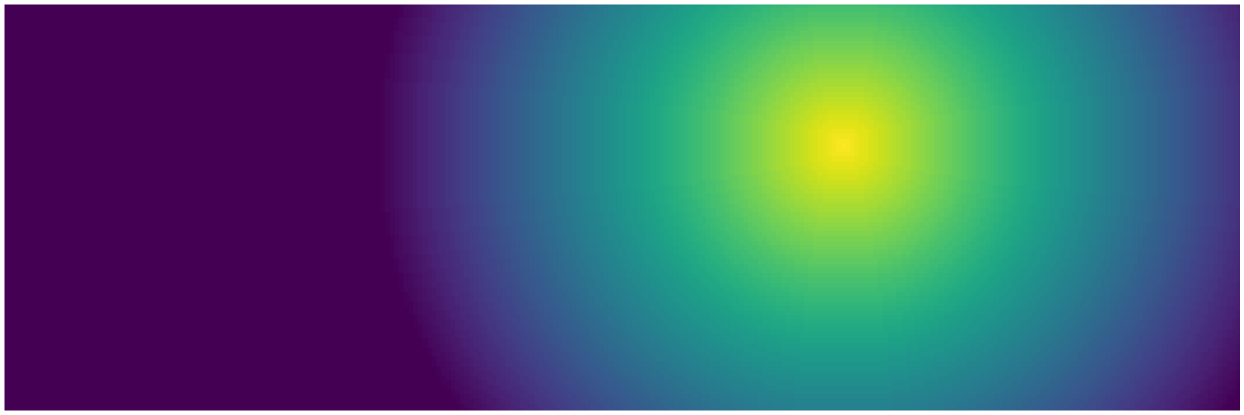


Online Algorithms Applied to Automated Alignment of Microwave Antennas

Bachelor of Science Thesis in Electrical Engineering

Gustav Lilliebrunner

Thanh Binh Nguyen

Online Algorithms Applied to Automated Alignment of Microwave Antennas

Gustav Lilliebrunner, Thanh Binh Nguyen

Supervisor and Examiner: Giuseppe Durisi

Department of Electrical Engineering

Chalmers University of Technology

SE-412 96 Gothenburg

Sweden

Telephone +46 (0)31 – 772 1000

Cover page: Simulated graphic print of RSSI map

Department of Electrical Engineering

Gothenburg, 2017

ACKNOWLEDGEMENTS

This bachelor thesis was carried out at Ericsson AB. The project could not have been done without our knowledgeable supervisor at Chalmers University of Technology and Ericsson AB. Therefore, we would like to thank our supervisors at Ericsson AB, Per Ligander, Thomas Emanuelsson and Torbjörn Wästerlid. We would also like to thank Giuseppe Durisi at Chalmers University of Technology.

Gothenburg, May 2017

ABSTRACT

This report covers a project of product development at the company Ericsson AB. Due to Ericsson corporate language this report is written in English. The work is executed by Gustav Lilliebrunner and Thanh Binh Nguyen as Bachelor of Science thesis within electronic engineering at Chalmers University of Technology.

Microwave links is a popular choice for connecting remote radio base stations to the core network. The microwave antennas are usually aligned by hand, a costly and time-consuming process that requires two operators present on each site. This project aims to reduce the manual alignment process. Today's choice of high frequency used in microwave links yield smaller antennas, which can be moved by embedded servomotors. This report covers the development of an alignment algorithm for microwave antennas. It also covers development of different hardware solutions to move the antennas

The method used for developing the algorithm is a combination of simulation of the problem as well as verification on hardware. Earlier work has been done on similar problems but with use of external server motors rather than embedded in the product, a simpler case because it admits more powerful motors and studier mechanics. The finished algorithm was able to perform a good alignment in both simulated environment and on hardware. However, due to mechanical problems and lack of time to solve them the verification was done with only one movable node and one stationary. A simpler test case than working with two movable nodes. A few challenges for further development is discussed which should be considered for a well working prototype in the future.

TABLE OF CONTENT

TABLE OF FIGURES

TERMINOLOGY

Node	Communication unit for receiving and transmitting information with radio signals.
RSSI	Received Signal Strength Indication
FPGA	Field Programmable Gate Array
NEMA	National Electrical Manufacturers Association

1 INTRODUCTION

1.1 Background

Microwave technology is the basis for the development of today's mobile networks. For linking remote base station to the core network a fiber optic cable or microwave link is commonly used. When using a microwave link, a radio transceiver node is mounted on each base station where one of them has a link to the core network. Microwave links are in many cases a less expensive solution for interconnection of base stations which makes them very popular.

Traditionally the nodes operate in frequencies between 6-40 GHz. Today, however, there are several products that works in the frequency range of 60-80 GHz. Higher frequency provides significantly smaller antennas, which now practically can be moved by small embedded servo motors. The motors open the possibility to automatically align the antennas toward each other for best reception. This work is currently done manually by two operators present on each base station. Thus, this is very costly and time-consuming work.

1.2 Purpose

The project aims to develop a system where the nodes automatically align toward each other for best reception. If the system works as desired only a rough alignment of the nodes must be done manually. That would preferably be done visually by the operator when fitting the new nodes. When the manual visual alignment of the node is done, the operator can leave the site and install the second node on the other site. After this is done an automatic scanning algorithm starts and searches for the best signal reception and stays in the location where it was discovered. The nodes should then scan for best operating frequency, modulation, and amplification. By making the node self-aligning and self-configuring it will reduce the set-up costs and make it an easy-to-use and affordable microwave link solution.

1.3 Boundaries

The work covers the development of only one type of algorithm for alignment of the antennas. Test and verification of the algorithm is executed on existing and developed hardware. To a limited extent research is done for better hardware solutions.

1.4 Clarification of the issue

Our main objective is to develop an alignment algorithm. It also includes mounting together new hardware and some modification of it as well as research of better solutions. To make the work easier a few points were set up.

- How should the antennas move to find each other? i.e. search pattern.
- How should the algorithm be designed?
- How should the hardware be designed with respect to cost efficiency and minimum slack.

2 THEORY

2.1 Alice and Baxter

The main problem is for two antennas to find each other in a predefined space. This quite complex issue could however be simplified with a simple mind experiment. Two dogs, Alice and Baxter, are lost in a room at the veterinary. Both have got themselves a collar to prevent scratching. The collar limits their field of view which prevent them from seeing more than a narrow sector of the hole room at a time. Both Alice and Baxter are completely white except for the black nose. In the room, it is impossible to see them against the white walls if they are not facing their black nose towards the viewer. When they start facing their head away from the viewer the nose is soon covered by their collar and they "vanish" in the white background and can only be seen again when they turn their head back. For Alice and Baxter, the goal is to find each other by simultaneously facing each other at straight angle like in figure 2.1.

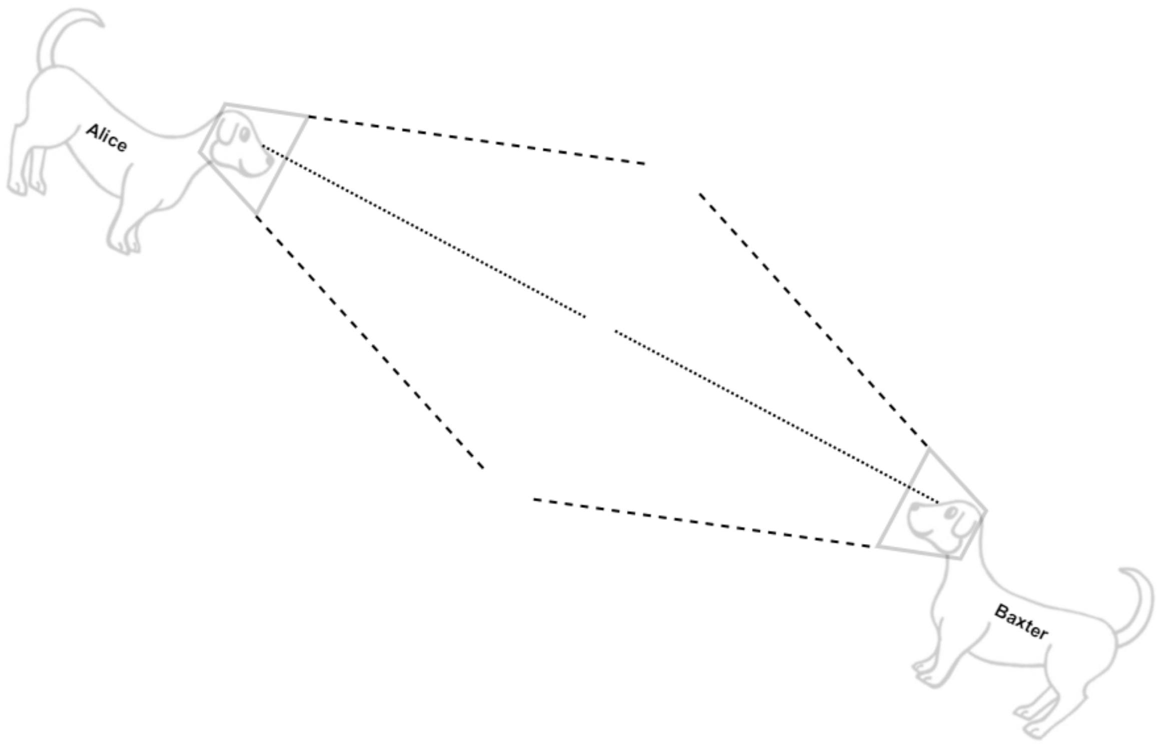


Figure 2.1: Alice and Baxter are simultaneously facing each other.

Two types of different situations where they do not see each other can occur. The first one is being seen in figure 2.2 were both dogs are facing their head away from each other.

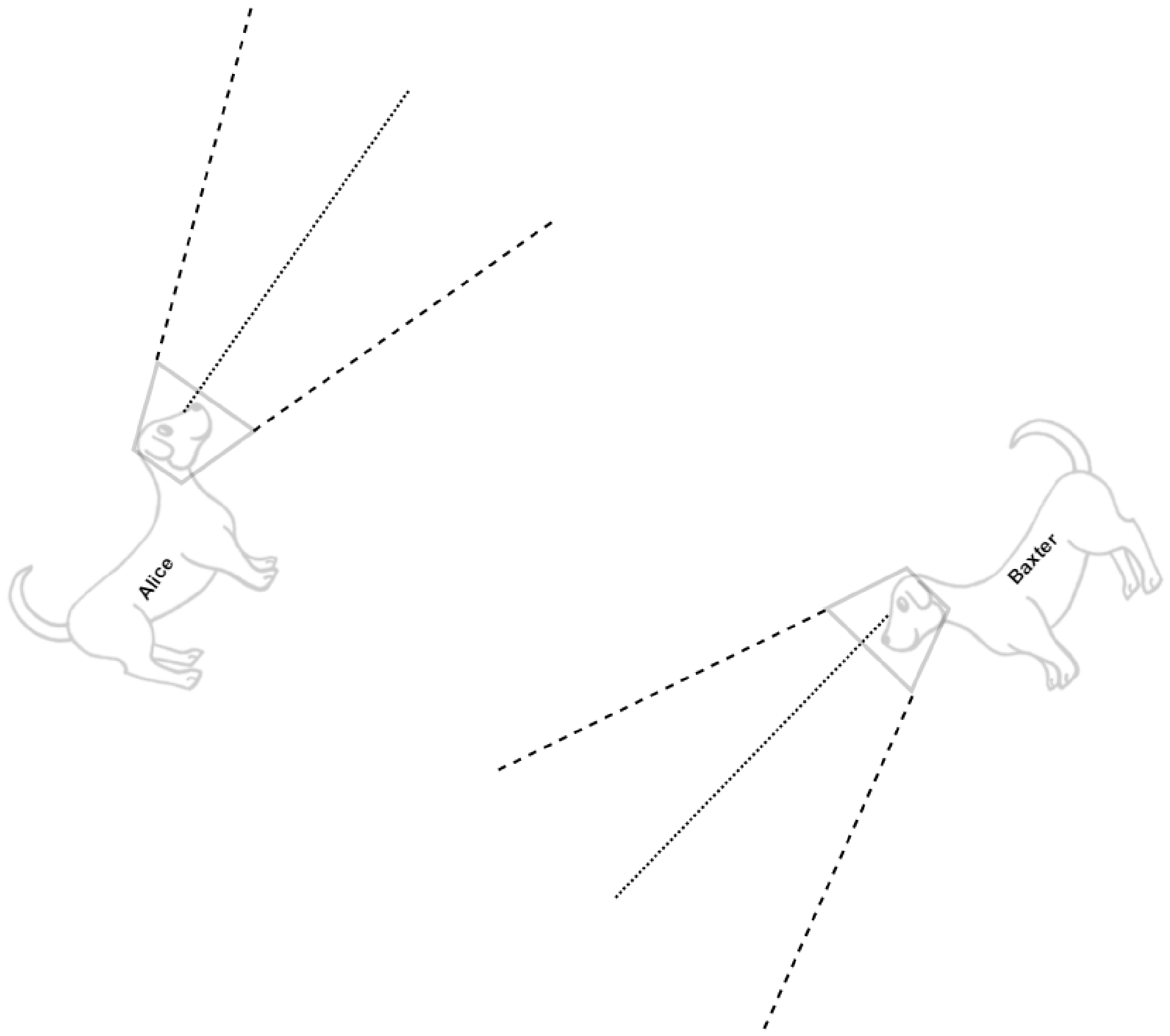


Figure 2.2: Both Alice and Baxter are misaligned.

This would of course result in that neither of the dogs see each other. There are also a second situation when one of the dogs is faced in the right direction like in figure 2.3 where Baxter is looking at Alice.

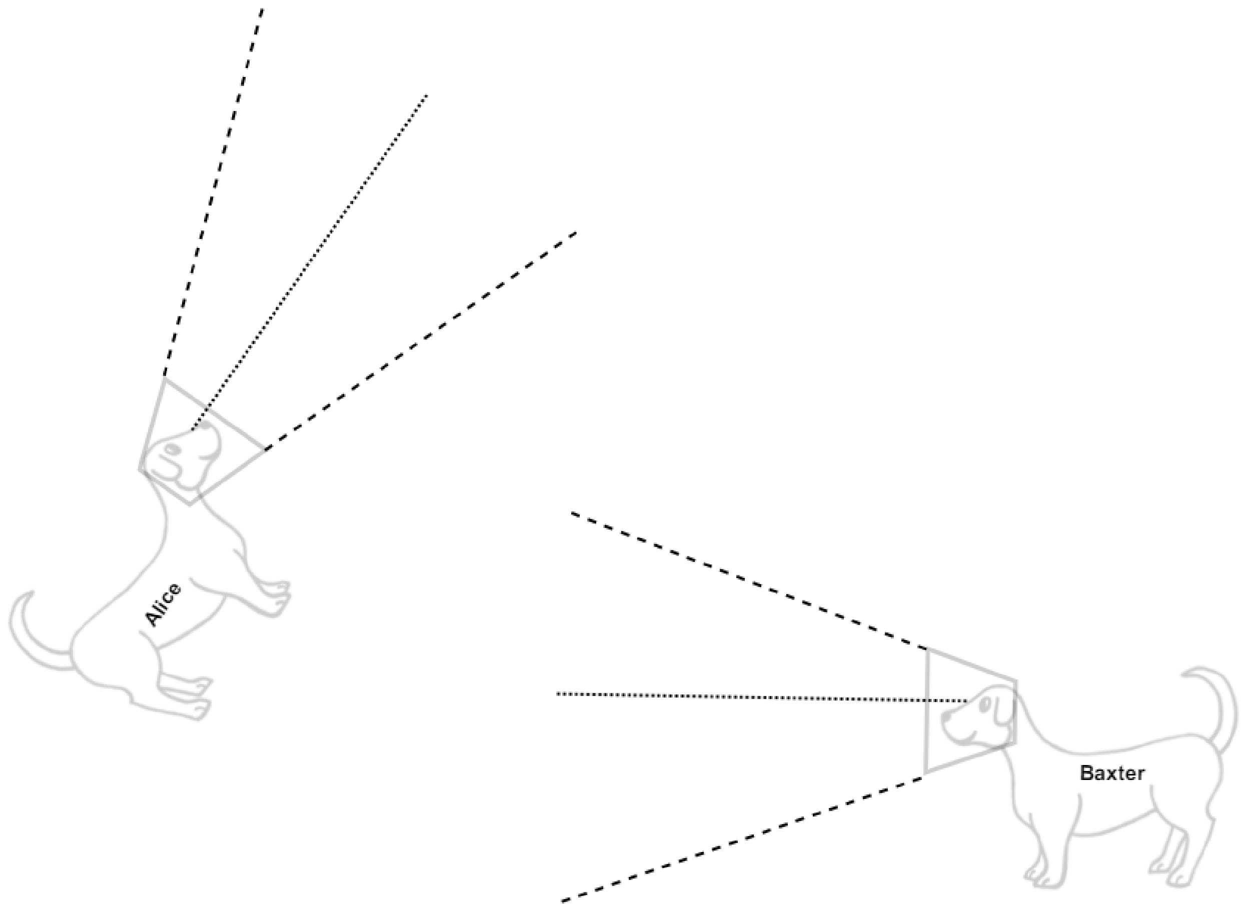


Figure 2.3: Baxter is looking in the right direction without noticing Alice.

This will unfortunately also result in that nether of the dogs see each other because Alice is facing her head away which prevent Baxter from discovering her black nose. Therefore, Baxter will be completely unaware that he was looking in the right direction in the first place. For Alice and Baxter it is important to find a search pattern that would make sure they at least once facing each other in fairly the right direction. If the nose was discovered only in the periphery this could be solved with a more precise search in a much smaller area later.

2.2 Alignment of antennas

In a real scenario Alice and Baxter represents two nodes, A and B. The goal is to achieve a perfect automatic alignment of the two nodes (see figure 2.4).

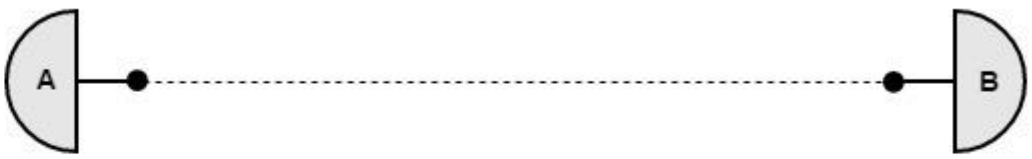


Figure 2.4: Both nodes are perfectly aligned towards each other.

To do so, two servo motors were mounted in a way that makes it possible to move the antenna horizontally and vertically, later referred to as X and Y. For each angle, X and Y it is possible to read the RSSI value from an internal register on the node. Each node is completely unaware of its own location and the location of the other. Initially each node does not have the ability to communicate another. As in the case with Alice and Baxter the antennas must align towards each other at the same time for a good RSSI value to occur. Due to reciprocally antennas both nodes discover the same signal attenuation over the radio link in any given situation. Therefore, one misaligned node as in figure 2.5 is enough for making low RSSI value for both nodes.

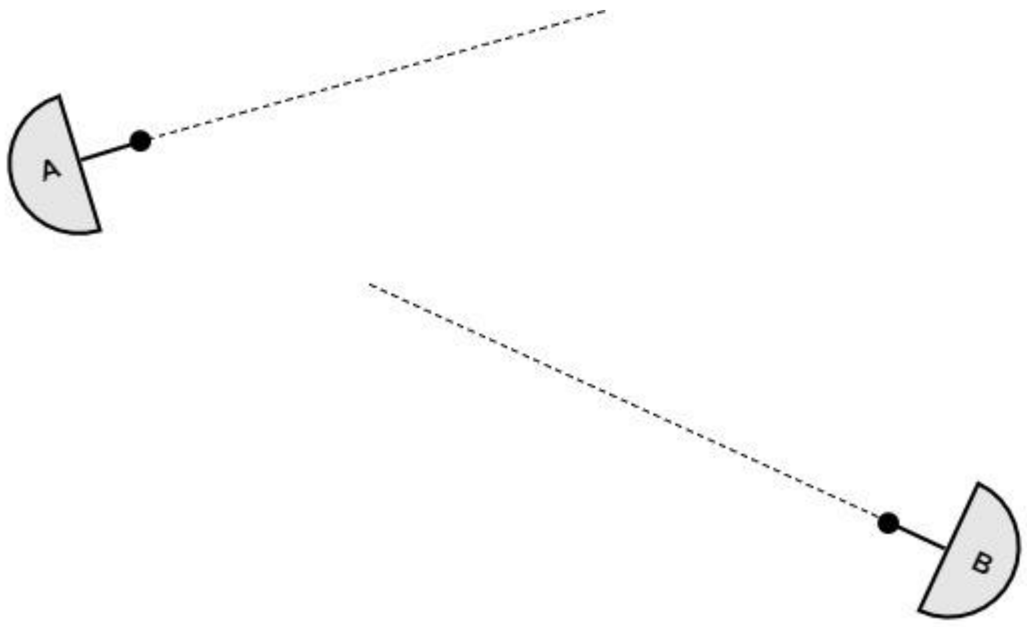


Figure 2.5: Only one node aligned.

Each node has mechanical boundaries which it cannot move outside. This were mechanically limited to ± 30° in X and ± 10° in Y. Therefore, for a perfect alignment to occur the opposite node must be mounted somewhere within that space and vice versa.

2.3 Earlier work

There was a project executed at Ericsson late 2015 [1], which has many similarities to this work. The big difference in the executed work was to self-align two antennas with help of external hardware, instead of internal hardware. The hardware used was based on a robust motorized pan and tilt unit for cameras. The unit used big servo motors for angular movement. This gives an advantage when it comes to higher velocity and therefore allowed the antenna to self-align in shorter amount of time. Due to the advanced hardware and technology it makes the earlier project an expensive solution. The current project is focusing on develop a smart and cheap solution for the self-aligning antenna. The earlier work provided a good overview when it comes to the theoretical part and hardware. With the information specified, it facilitated a smarter solution, where we could use less components and a different smart search of the self-alignment. In the previous solution, the self-alignment took approximately less than 15 minutes. With smaller motors this will take more time but would be a much more affordable choice.

2.4 Hardware

2.4.1 Ericsson HW

Each node consists of several components stacked together. One power board which take power over Ethernet and convert it into different voltage levels suitable for the modem board. On this board, a number cables are soldered on suitable outputs on an FPGA. These are used later for connecting the servo controller boards to Ericsson HW. Stacked on this board is the modem board that handles all the logic. On this board, a microwave board together with a cavity filter is mounted. On the filter, a waveguide is fitted together with the antenna. All this is encapsulated in an aluminum cabinet that provide cooling for the components as well as mechanical protection. Normally the modem and the microwave board together with the filter is mechanically fixed together. For this prototype, however, the components are separated and connected with a special designed flexible ribbon cable which gives them ability to move in relative to each other. For the prototype, a new mechanical encapsulation has been made containing a gyro suspension for the antenna. Here the antenna and the microwave board together with the filter can move in X and Y angle in relation to other hardware. Two embedded servo motors provide the angular movement.

2.4.2 Stepper motors

Initially for this prototype, small stepper motors with a reduction gearbox were chosen as servo motors. Unlike a regular DC motor a stepper motor works in discrete angle step. A commonly used step angle is 1.8° per step i.e. 200 steps per revolution. It does not require location feedback, can provide torque when it is standing still, high torque while moving and is easy to control from a digital system [2]. All this make is suitable for motion control of different mechanics. Normally, a system with stepper motors require three things, a stepper motor, a driver, and a power source.

2.4.3 Motor drivers

For the stepper motors to work, a driver of some sort is needed. Normally the driver takes in discrete pulses for stepping and direction of the motor. These signals could be provided from a simple microcontroller. The controller could also provide more complex functions as current control for the motor windings as well as micro stepping of the motors. Micro stepping is a technique for making the motors stay in between each step and hence provide a higher step resolution. For this project, the Easy Drive v4 were used. It is a simple and easy-to-use driver with lots of functions, including micro stepping down to ⅛ of a step as well as current control for the motor windings. A common external AC adaptor were used as power source for the drivers. In the future, it is desired to take this directly from the power board.

2.5 Python

To develop a self-aligning algorithm a software is needed. This is designed in an object oriented, high level programming language, called Python. The reason for using this language is due to that Python supports modules and packages and is available to all major platforms without charge. The advanced programming can utilize and qualify all the modules functions as stepping functions, the velocity of the stepper motors and all digital signals as desired, by stacking local and global variables through codes in data structures. The programed codes in Python can easily be debugged by initiating Pythons own debugger, which traces errors by inspecting the codes with breakpoints and stepping through one line at a time. This makes it highly effective when debugging [3].

3 METHOD

The project contains several parts that must be considered to reach the desired result. These parts sometimes overlap each other in different places and are ongoing at the same time. Gather basic knowledge about the product and its purpose is essential for making an improvement of it. Different ideas for improvement are considered though the complete project and evaluated in the end. These working points were set up to reach the goal.

- Studying the existing hardware.
- Assemble and try out hardware version 1.0.
- Research for the alignment algorithm.
- Programming of the algorithm.
- Assemble and try out hardware version 2.0.

4 SPECIFICATIONS

- Scanning limits: ± 30° in X and ± 10° in Y.
- Antenna -10 dB beam width: ± 3.7°.
- The nodes are installed in clear sky with no reflection from buildings, ground etc.

5 ASSEMBLIY AND VERIFICATION HW 1.0

Initially, a plastic model was used for demonstration purpose of the concept. To make a sturdier construction, an aluminum model was ordered for further testing. Two camera tripods with mounting bracket were used to fit the node during testing. First problem discovered was that the cabinet mounting holes for the main M8 bolts were too big. The problem was solved by drilling up the mounting holes, thread them to M10 and insert a steel bushing. Later, it was discovered that a steel bushing should have been inserted from factory but were missing.

Before disassembly the plastic prototype a wiring diagram was drawn based on the previous connections (see appendix 1). The power board and modem board was moved into the new cabinet. Two new drivers and stepper motors were soldered together according to the wiring diagram. While testing the motors, it was soon discovered communication issue. The motor did not always respond to command sent from the computer. After some troubleshooting using an oscilloscope and checking all logic signals from the power board it was discovered that the logic levels was around 2.1 V. The drivers however work by default with 5 V logic levels and therefore did not respond correctly. The drivers could also support 3 V logic with a simple HW configuration which solved the problem.

While testing the hardware, it was discovered that the motors and gearboxes had a huge slack. The motors also had difficulties providing enough torque for the antenna to move. A simple measurement set up in figure 5.1 showed that the slack was around 4°.

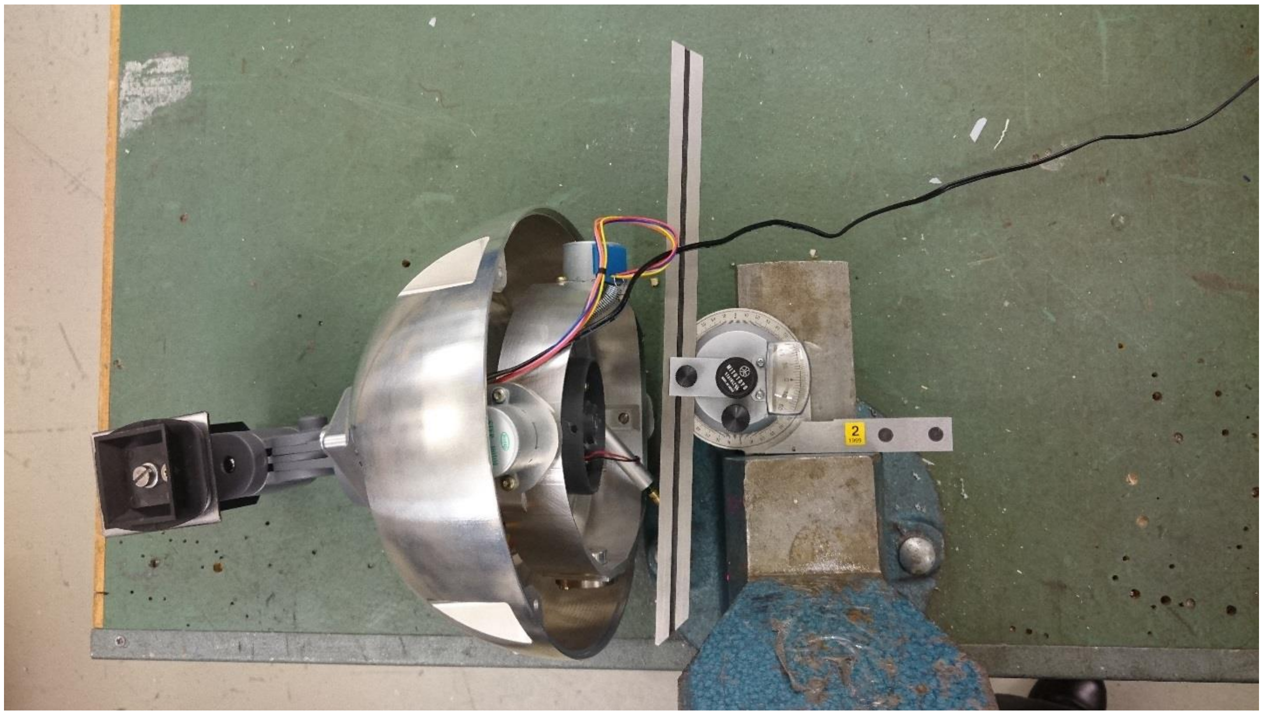


Figure 5.1: The slack in the cheap motors was measured by moving the outer aluminum ring back and forth while measuring the difference in angle.

An example of two nodes being 100m apart from each other show that the center of the lobe can move almost 7m at the opposite side, which is more than acceptable.

Some basic requirement for new motors were set up:

- They must be small enough to fit in the enclosure without cutting hole in it (d*w*h = 20 * 40 *40).
- They should provide more torque than the original ones.
- They should have significantly smaller slack.
- They should be possible to drive with the same driver boards.

Two different stepper motors types without gearbox were ordered. Due to the lack of gearbox the slack is assumed to be 0°.

	Old, With gearbox	New 1, Without gearbox	New 2, Without gearbox
Size (mm)	Ø 28 x 19	35 x 35 x 20	35 x 35 x 26
Torque (Ncm)	2.9	5	7
Slack	≈ 4°	≈ 0°	≈ 0°

Fitting the new motors required lots of modification of the existing hardware. Both of the new motors use the standardized NEMA 14 hole pattern, and is therefore easy to switch between. The only difference is in the depth of the motors where just the smaller one fit the enclosure without cutting hole in it. First the hole pattern was drilled out for the new motors using a standard milling machine (see figure 5.2).

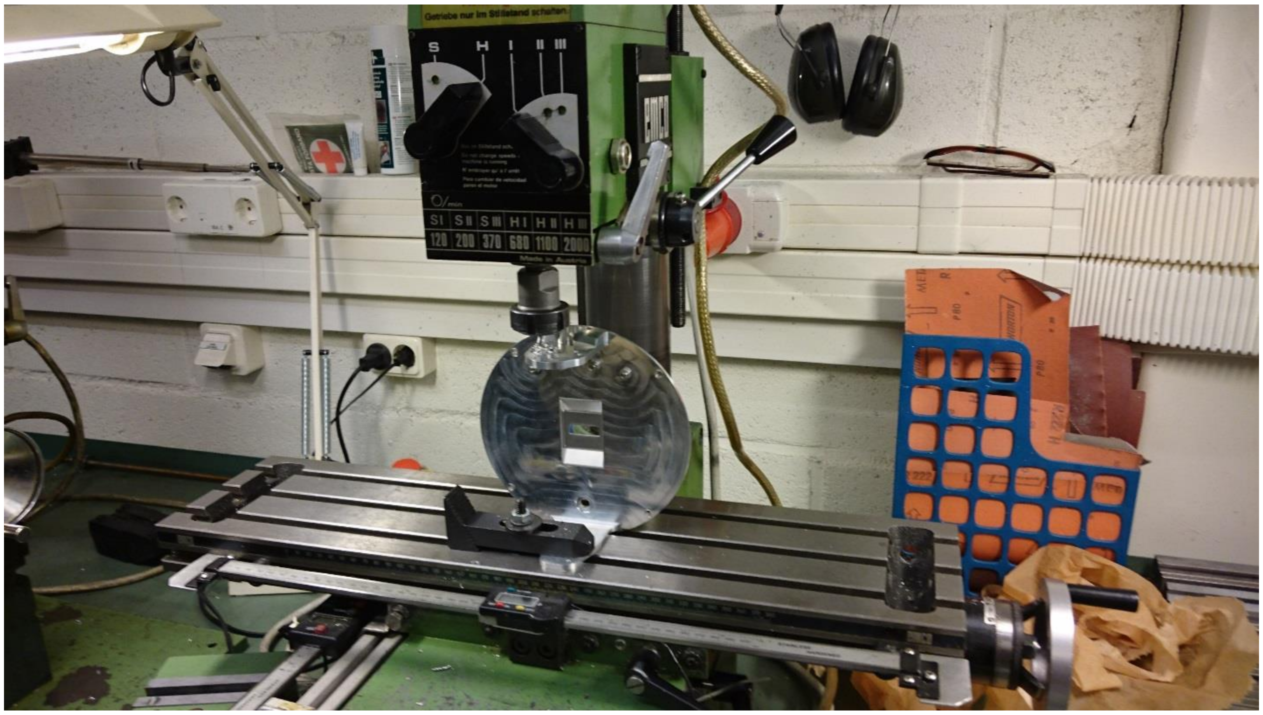


Figure 5.2: Drilling a hole pattern for the new motors.

The new NEMA 14 motors has bigger flange going into the aluminum bracket seen of figure 5.3. Therefore, it was needed to drill the flange hole bigger.

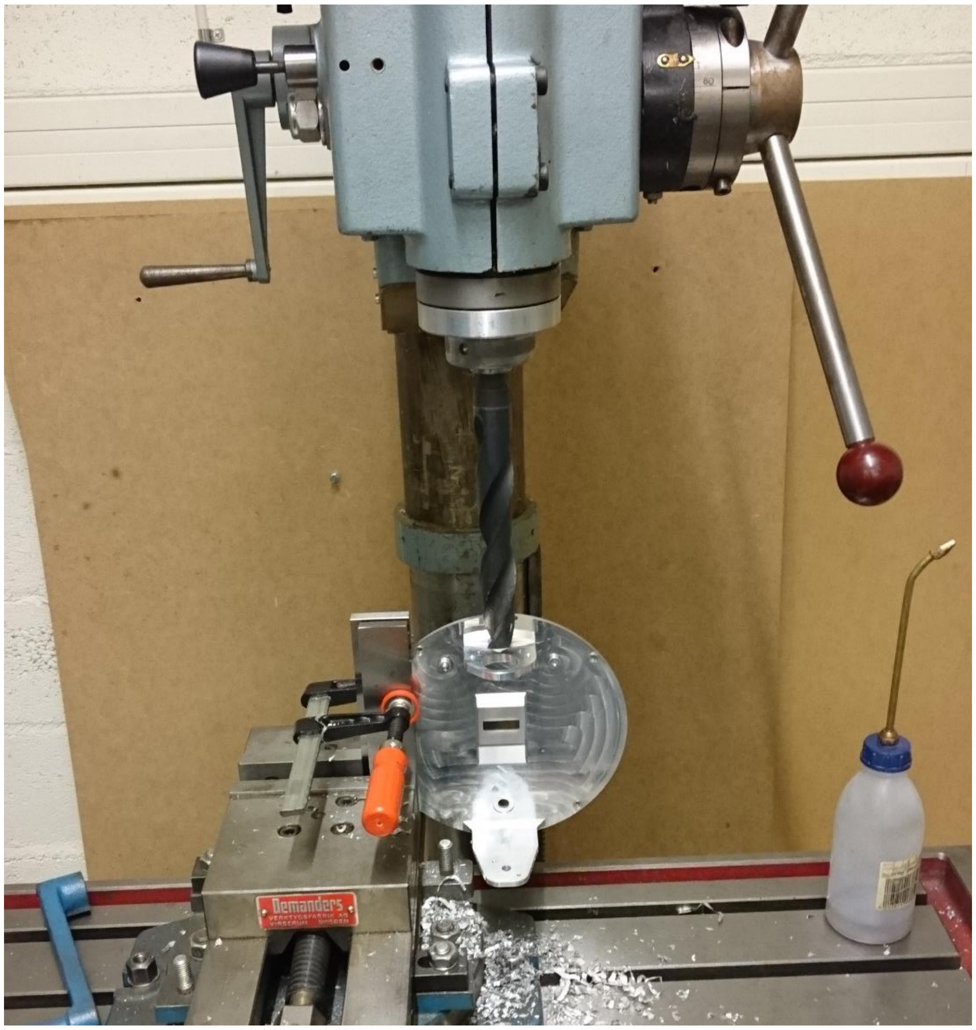


Figure 5.3: Drilling a bigger hole to make room for larger flange.

The shaft of the new motors was longer than the original once and was shorted using lathe. Finally, the motor shaft needed a secure locking to the mechanics. It was solved by drilling and threading a hole for a set screw working against the motor shaft (see figure 5.4).

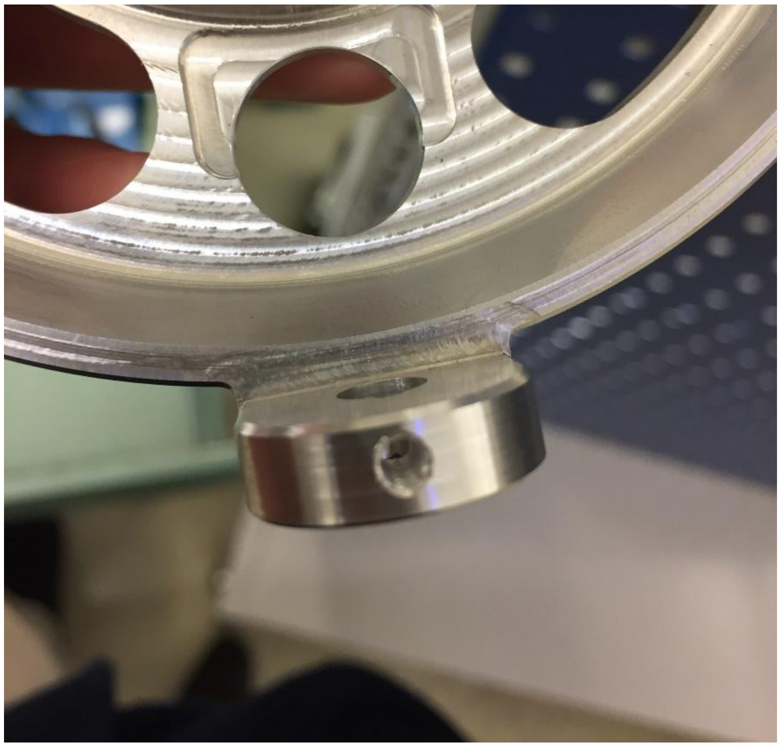


Figure 5.4: Threaded hole for a set screw securing the motor shaft.

All parts except for the ribbon cable, wave guide and antenna were then assembled and tried out (see figure 5.5).

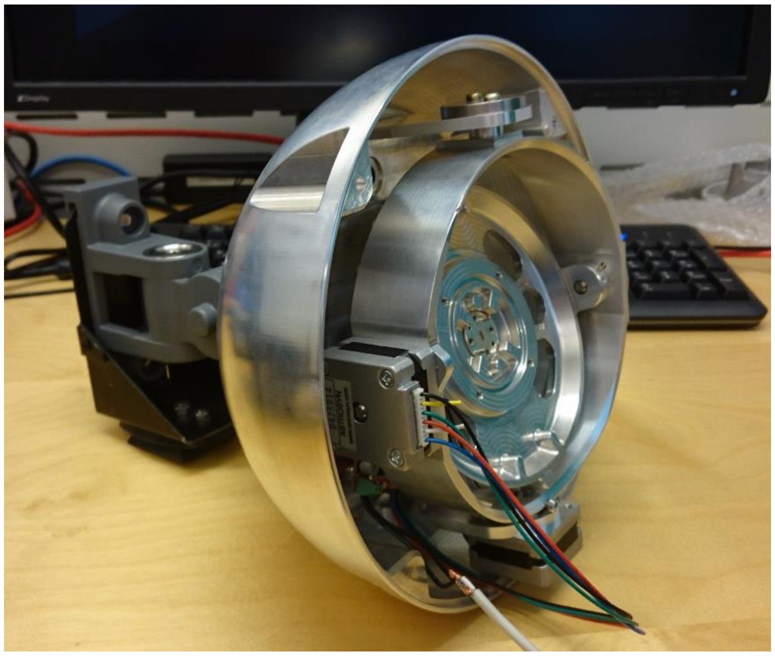


Figure 5.5: Assembled node without ribbon cable, waveguide and antenna.

6 ALIGNMENT ALGORITHM

6.1 S – Pattern

As seen later the algorithm form an s-pattern when searching, therefor called the s-pattern search. The main idea of the s-pattern search is to always provide a good alignment independent of where the two nodes are in the predefined space. The idea is based on the fact that the main lobe has a certain beam with in degrees where the attenuation is still acceptably low. Outside the limits the attenuation is lower and is therefore not desirable. As a result, the nodes do not have to search through every point on the map and still can detect the other node. With the assumption that the opposite node is already perfectly aligned, the attenuation over the air is neglected and the search width and height are known, it is possible to calculate the number of sweeps needed for making the rough alignment. This calculation will provide an alignment that is equally good or better than the acceptably low attenuation used in the calculation. Using the -10dB beam width of ± 3.7° and the mechanical limits of ± 30° in X and ± 10° in Y the number of sweeps n_X and n_Y can be calculated according to equation 6.1 and 6.2.

$$n_X = \frac{X}{beam\ with} \approx 8.1\ Sweeps \tag{6.1}$$

$$n_Y = \frac{Y}{Beam\ with} \approx 2.7\ Sweeps \tag{6.2}$$

With knowledge of the number of sweeps needed a search pattern can be designed. One way is to let the node search in an S-pattern with n_Y number of horizontal or n_X number of vertically sweeps. In this case were the search area is wider in X than Y it is more convenient to do horizontal sweep because it requires less stop at the boundary points seen in figure 6.1.

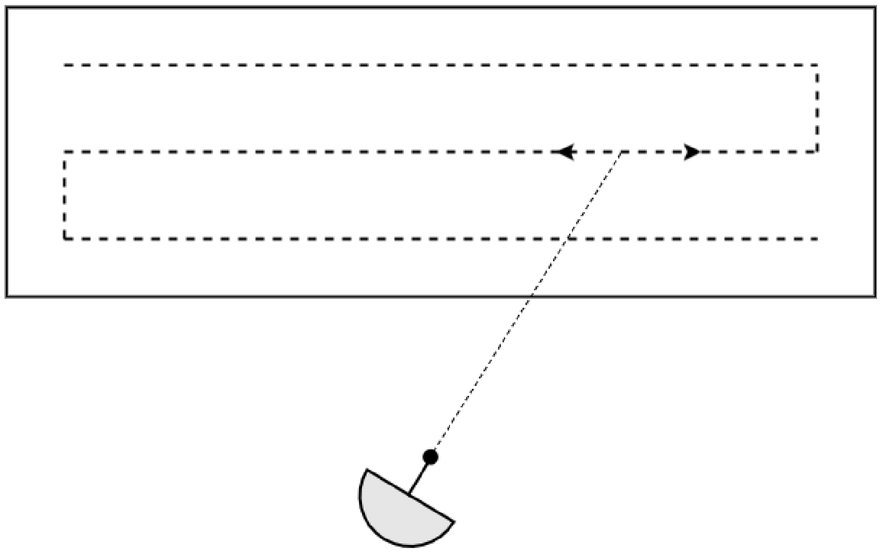


Figure 6.1: Node sweeping horizontal lines in an S-pattern.

To further optimize the search algorithm there is no needed to search the boundary points all the way to the edges because the lobe has a certain acceptable beam with. In our case the boundary points can be moved in 3.7°.

In real case scenario, the assumption that one node is perfect aligned is not valid. Also, one node is not aware of the other nodes location or where it is pointing. Therefore, if both nodes used the same S-pattern search right away there would be no guarantee that they would ever find each other. One could imagine a case were both node is pointing away from each other and searching with the same speed. This would result in a case where they never cross each other.

An approach for overcoming this issue is to move one node in the same S-pattern but in discreet steps and different speed instead of a continues movement. Each step should overlap the previous so that a maximum attenuation of -10dB occur. The time in between each step should be exactly the time it takes for the other node to do one complete sweep. The idea with this method is to try all combinations necessary with the two antennas that would at least generate an alignment equal or better than -10dB + -10dB i.e. -20dB.

6.2 Fine Search

The strength with the S-pattern search is also its weakness, it will always find an alignment with a maximum attenuation of -20dB but there is no guarantee that it will find anything better. If a better alignment is desired, more sweep must be done to cover the search area with better precision. This is unfortunately very time consuming. Another approach is to let one node at a time do a fine search. This could be done in a much smaller area where good reception from the other node was discovered. The idea is to start by doing a square around the point where best RSSI value was discovered. That would likely provide the node with a better RSSI value somewhere on that square. Now a second square with half the radius could be done around the point where the better RSSI value was discovered. By doing this repeatedly the search algorithm converts to a perfect alignment towards the other node, here represented by a red dot in figure 6.2.

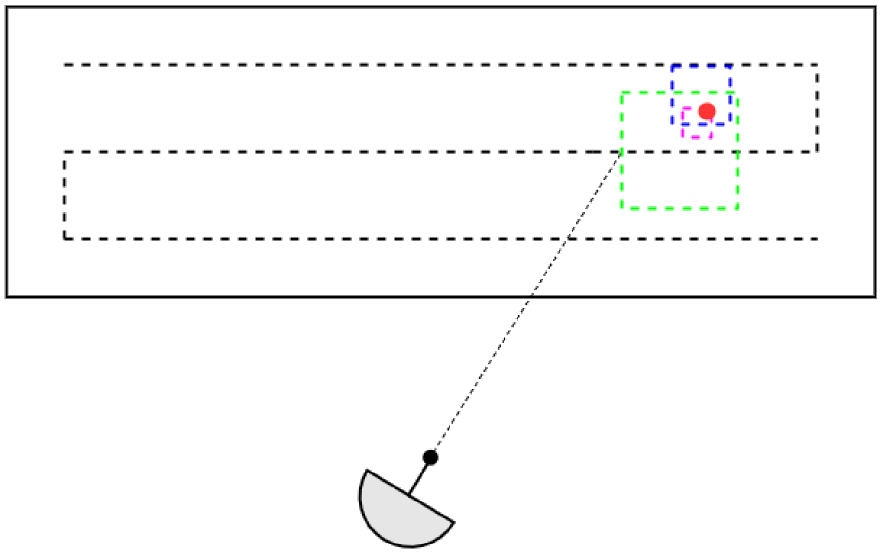


Figure 6.2: The search algorithm converts towards the other node in three iterations.

7 Programming and Simulation

7.1 Code Structure

The code is built around several functions with specific tasks. This make the code easy to reuse in other functions. In the main loop, important functions are called to form a program. When working with hardware some extra functionality is needed to communicate with the server running on the node. The code for working with simulated nodes are found in appendix 2. The code for working with hardware are found in appendix 3.

7.2 Simulator Function

The code has a simple built in RSSI signal simulation function that was designed to debug the code during development. If the function is used with the graphical heat map print function it is easy to debug and evaluate the code. The linear function decreases from 100 dBm to 0 dBm depending on how far away from optimum alignment the algorithm is. The function calculates the distance from where the algorithm discovers the best RSSI value to a point where the opposite simulated node is (see equation 7.1).

$$RSSI = 100 - \sqrt{|X_{opp} - X|^2 + |Y_{opp} - Y|^2} \qquad (7.1)$$

$X_{opp} = horisontal\ location\ of\ the\ opposite\ node$

$Y_{opp} = vertical\ location\ of\ the\ opposite\ node$

$X = current\ horisontal\ posision\ of\ the\ node$

$Y = current\ vertical\ posision\ of\ the\ node$

It is possible to change the location of the simulated node by changing the global X_{opp} and Y_{opp} variables. This gives the freedom to try different location scenarios for evaluating the code. To the output RSSI value it is also possible to multiply a random variable between 1 and 0. It represents that the simulated node is moving and does not always face in the right direction during the search procedure.

7.3 Algorithm Verification Using Simulator Function

The algorithm starts by doing a number of S-pattern sweeps. Doing only one sweep provide very uncertain information about where the other node is due to movement at the opposite node. This can be seen as discontinuous colors in the RSSI value heat map in figure 7.1 where bright yellow represents the best RSSI value and purple represent the least good.

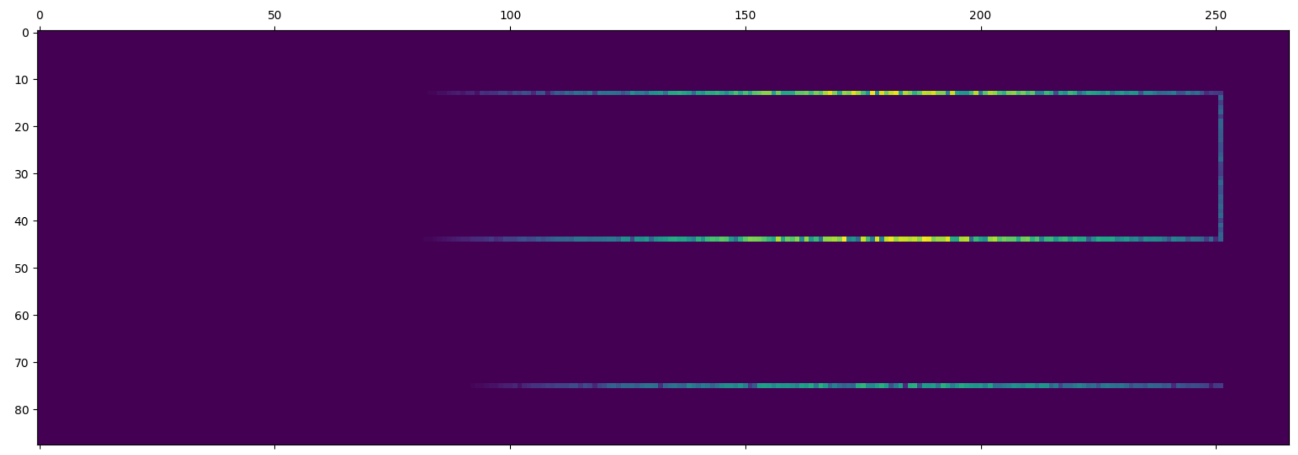


Figure 7.1: Only one S-pattern sweep provides very uncertain information about the other nodes location.

Repeating this procedure multiple times and updating the RSSI value map if better RSSI value is discovered in a point, provide the node with more precise information. In figure 7.2 this procedure is repeated 50 times which provide a more precise idea of the opposite node's location.

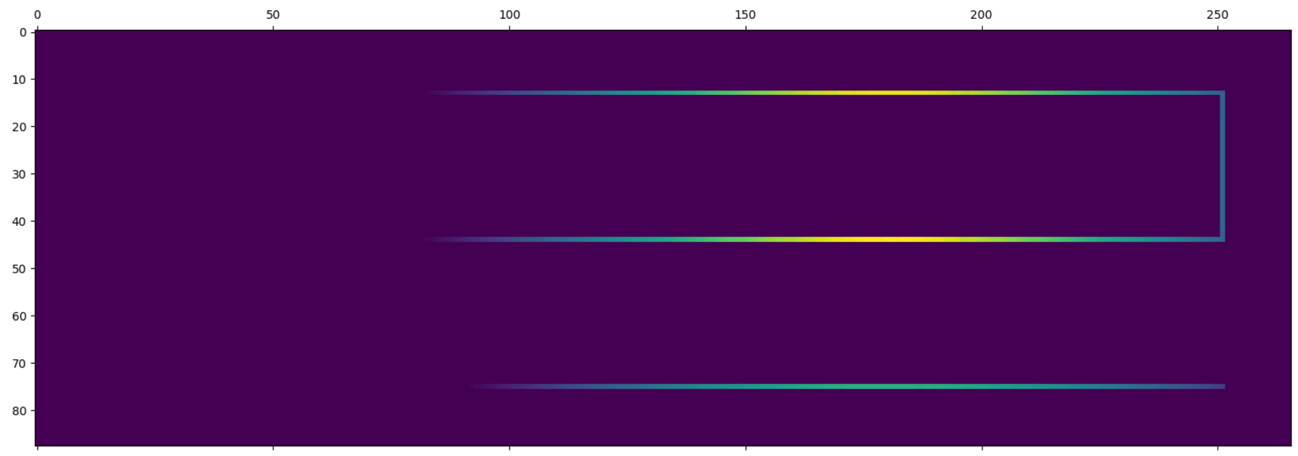


Figure 7.2: The S-pattern sweep is repeated 50 times to provide better precision.

From here on it is possible to do a fine search around the point where the best RSSI value was discovered. This can be seen in figure 7.3 as a square around bright yellow point in the middle sweep.

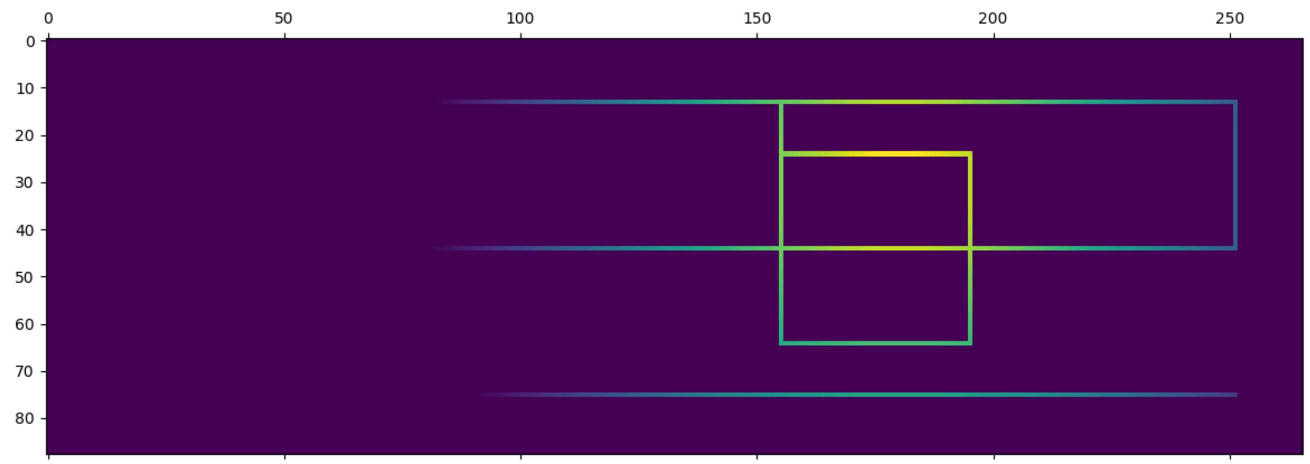


Figure 7.3: Fine search around the point with the best RSSI value.

In next iteration, the search is done around the better RSSI value. The radius of the next square is halved which makes the algorithm converges toward a better RSSI value. This procedure is repeated until there is no possibility to make a new smaller square seen in figure 7.4.

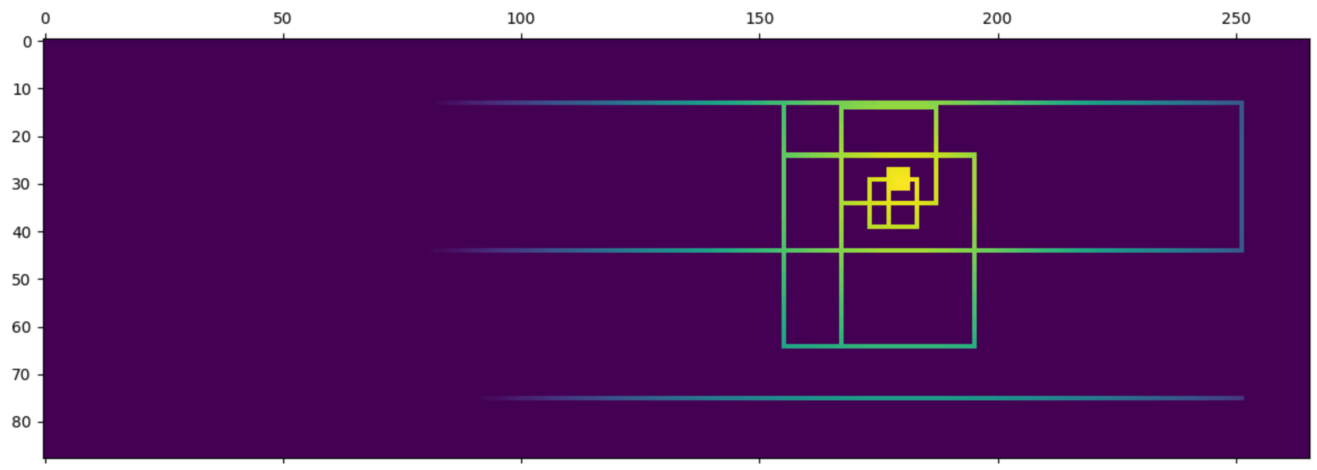


Figure 7.4: Alignment is finished in five iterations.

Using a dummy function, it is possible to fill the complete signal map in figure 7.5 with RSSI values.

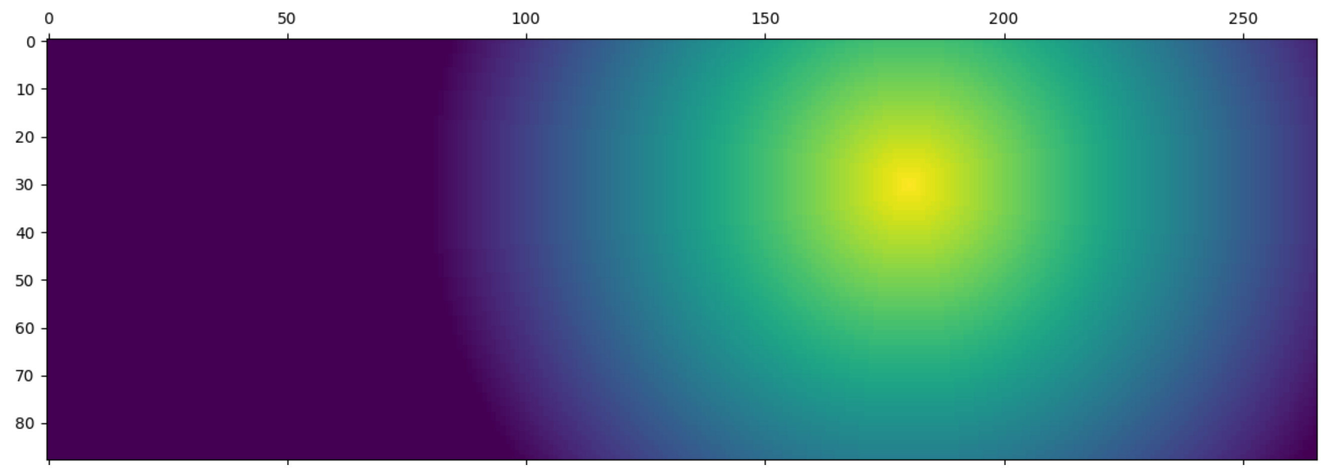


Figure 7.5: Completely filled signal map using dummy search function.

This represent the node searching through every point possible. Due to the time required this is not possible to do in reality but provide good visualization of the problem.

8 RESULTS

8.1 Simulation Results

When using the simple integrated simulator, the algorithm worked well. It was possible to find the other node and do a perfect alignment towards it. Most of the software function was finalized and the self-aligning algorithm worked well in simulation. The fine search function worked perfect and converged quickly towards the simulated node. However, the simulator provided a simplified version of reality and the results could therefore not tell for certain if the algorithm worked properly in reality.

8.2 Hardware Results

Unfortunately, the result of our project didn't meet our expectations due to errors with the hardware and the lack of time to solve the problems. The biggest issue was with the stiff ribbon cable and week servomotors which led to that our hardware cannot move as desired with our ribbon cable connected due to its stiffness. This caused limitations of the movement of the antenna due to the lack of flexing and blocking the antennas vertical and horizontal movements, thus gave a minimized searching area. The ribbon cable and the motors was not the only problem that set a limitation of the antennas movement. The design was also a big factor to this due to the overweight from the microwave card and filter which puts too much force on the steeper motors. The combination of these two errors almost paralyzes the antennas and made it hard to move. With all components installed the antenna could self-align within an angle of around ± 5° in vertical and ± 7.5° in horizontal.

8.3 Outdoor Test

The hardware test setup consisted of two nodes, one movable running the algorithm and one stationary, both see in figure 8.1. The code used for this test is found in appendix 3.

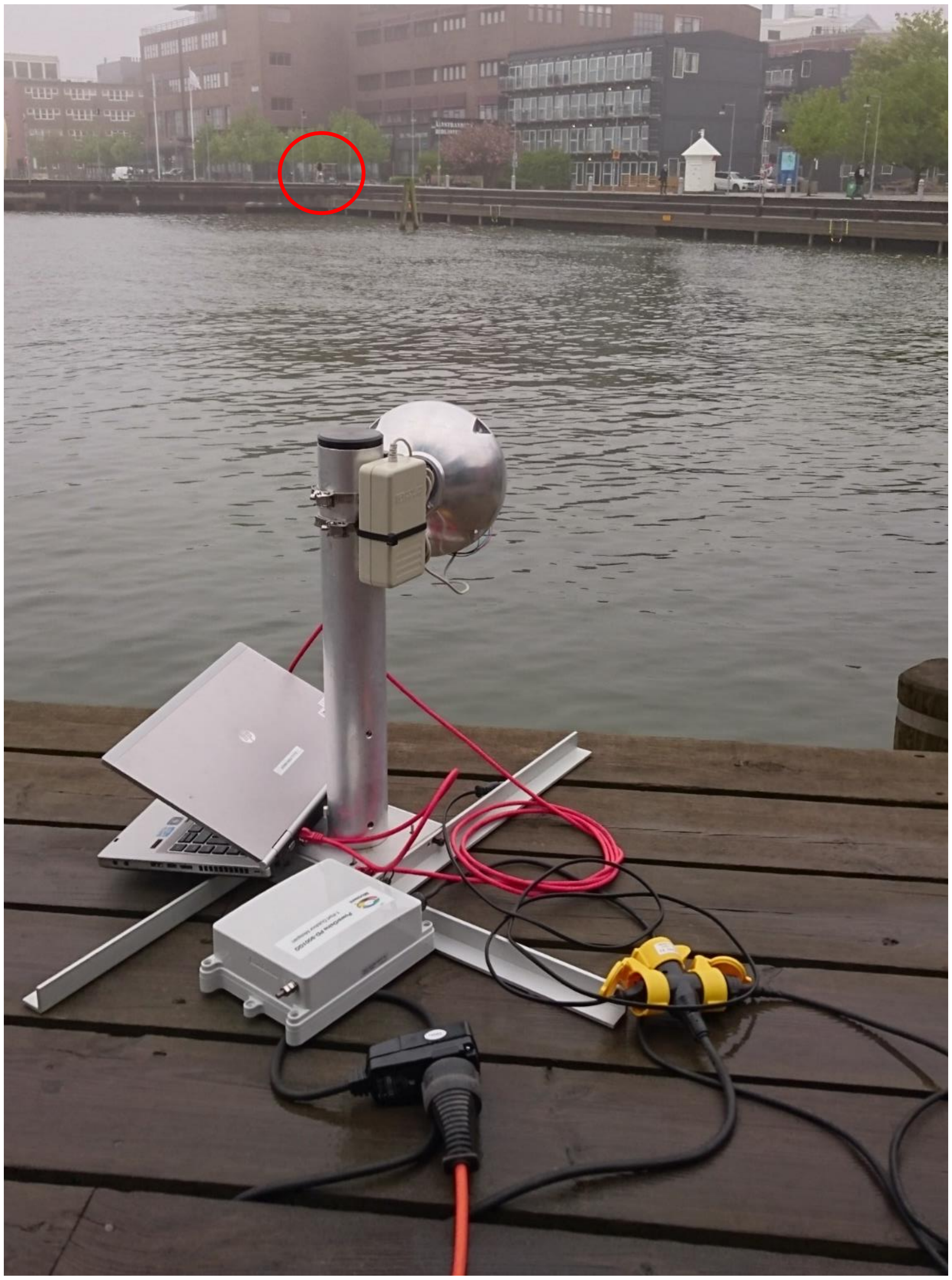


Figure 8.1: Outdoor test setup, observe the stationary marked with red circle.

The stationary node was aligned by hand toward the movable node. The search angle on the movable node was limited to ± 5° in vertical and ± 7.5° in horizontal due to hardware limitations. The distance in between the two nodes where approximately 100m. A simple graphic evaluation like the simulation were done to determine how the algorithm worked. Seen in figure 8.2 is the out print from the rough search.

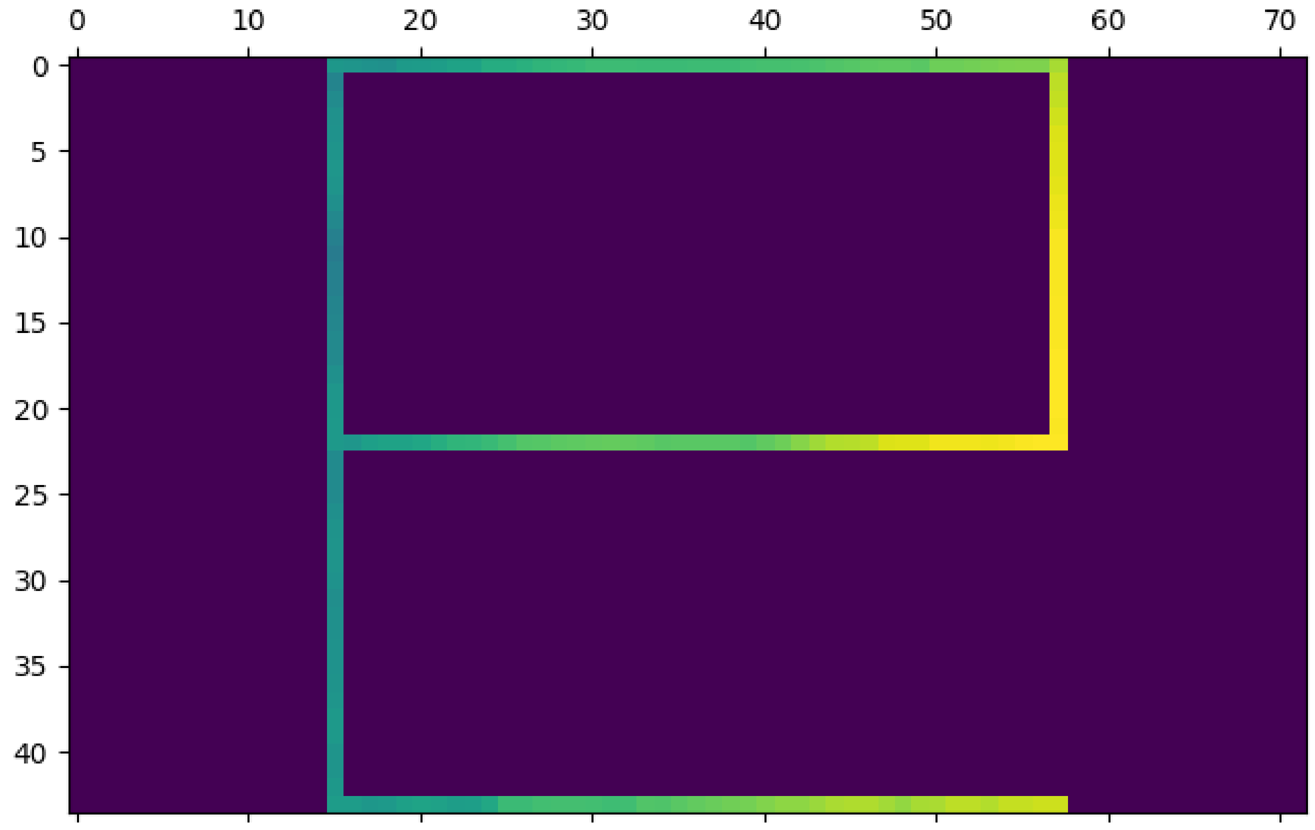


Figure 8.2: First rough search.

From here the fine search takes over seen in figure 8.3.

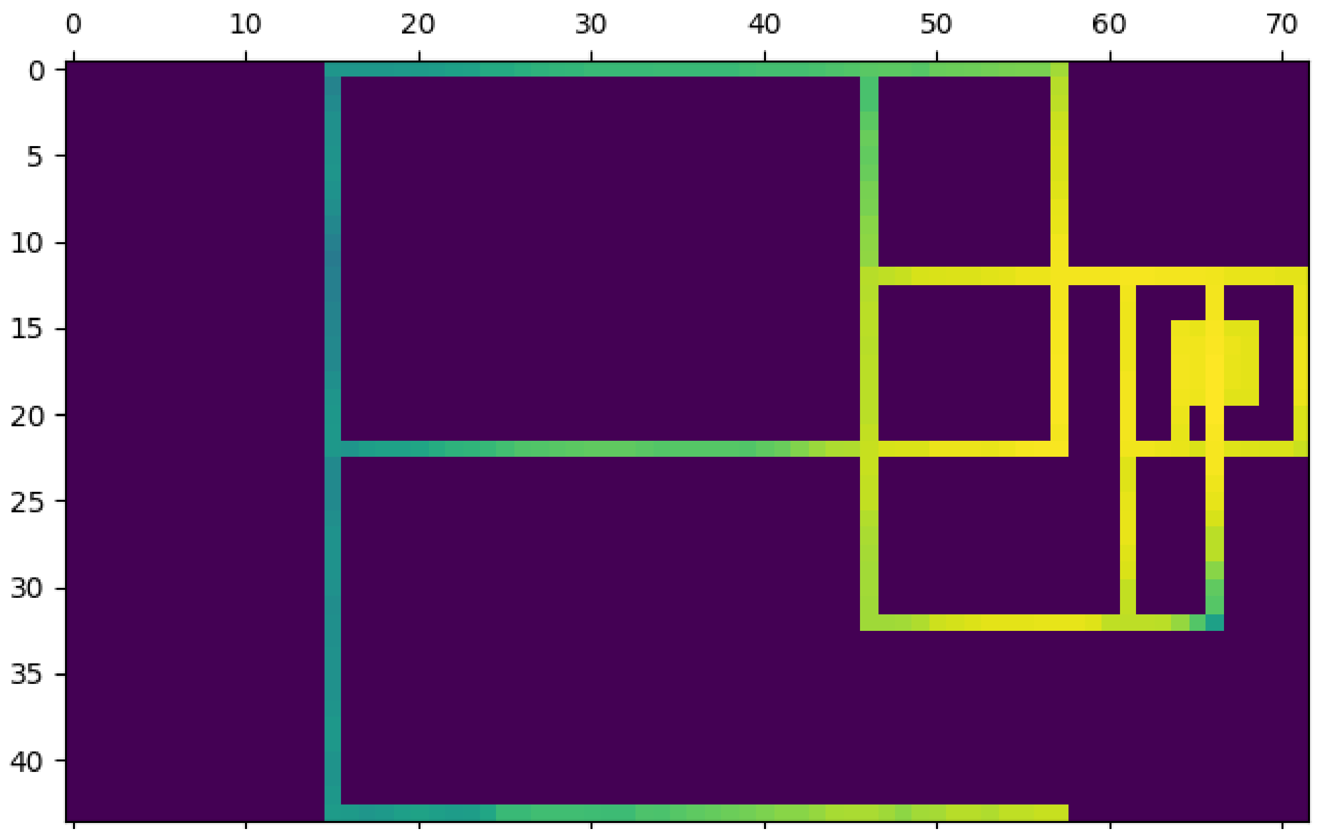


Figure 8.3: Fine search.

With the setup, we could perform a good alignment. Some fluctuations in the input signal can be seen in in the figure as none continues color change. This was probably due to various reflections from the water surface.

9 DISCUSSION

The project did not go exactly as planned. There were several reasons for that but mainly it depended on that the hardware did not work as expected. The expectations did not reflect reality and therefore some time-consuming modifications had to be done. The result was less time to develop the algorithm that did not came as far as we wanted. The established requirements for the new motors ordered perhaps were too strict which made it hard to find a suitable motor. If the motor could be slightly bigger in size it would have provided significantly better options. Never the less, the product seems to hold a great potential for the future. A few big challenges needs to be solved for a well working product and are listed below.

The stepper motors did not provide enough torque. They were also unable to hold the antenna while powering off. This is a big issue because the alignment had to stay the same even if the power is lost. For power saving purpose, it is desirable to turn of the motor after the alignment is done. One simple and reliable solution for this would be to use a worm gear between the motor and the antenna. A worm gear can only transmit power in one direction which would cause the antenna to stay the same even if the motor is turned off. The motor, however, could move the antenna but not the other way around. Normally, a worm gear is a reduction gearbox which could also solve the problem with lack of force from the motors.

A more flexible ribbon cable is needed. The cable was the main cause preventing the motor from moving the antenna. It is also very likely that the stiff cable would eventually break due to material fatigue. Unfortunately, the cable needs to contain lots of conductors which gives a stiff cable. It might be possible to use a flexible wave guide and have the filter and microwave board stationery. A completely different approach would be to move the motors outside the node and move the complete node instead. Of course, this require bigger motors and heavy-duty mechanics but it would solve the ribbon cable problem.

To make a faster alignment, a better algorithm must be developed. The S-pattern search did not converge toward the other node at all and was quite simple in its nature. Yet it always provides some result in a predictable time which is its greatest advantage. A more advanced algorithm that quickly converge toward the opposite node could probably solve the alignment a lot faster. To make a good algorithm, reflections from the surrounding area must be considered. Reflection from a close building could provide a good input signal and is easy to mistake for the correct alignment. A smart algorithm should be able to discard these false maxima for the right one.

Doing a traditionally installation of microwave nodes requires two operators travel to each site. If the nodes could self-aligned only one operator is needed to install the link. This would be a huge benefit both economically and environmentally due to less traveling. Another important aspect of self-aligning nodes is the quality of the alignment. The amount of traffic that can be transmitted over a radio link is related to how good alignment that can be achieved. If the self-aligning algorithm can perform a better alignment than a human, this would admit bigger traffic volumes over a single link. This, in turn, give a more effective use of the resources and less hardware and less energy is needed. Using less hardware admit less use of the limited frequency spectrum.

Over all the project provided some useful results and highlighted a few difficulties in different technical areas. Hopefully this will be a good base for further development of this product.

REFERENCES

[1] F. H. Lars Manholm, "Automatic antenna alignment – Part-3 (Outdoor demo evaluation) [PowerPoint]," 07 03 2016. [Online]. [Accessed 17 05 2017].

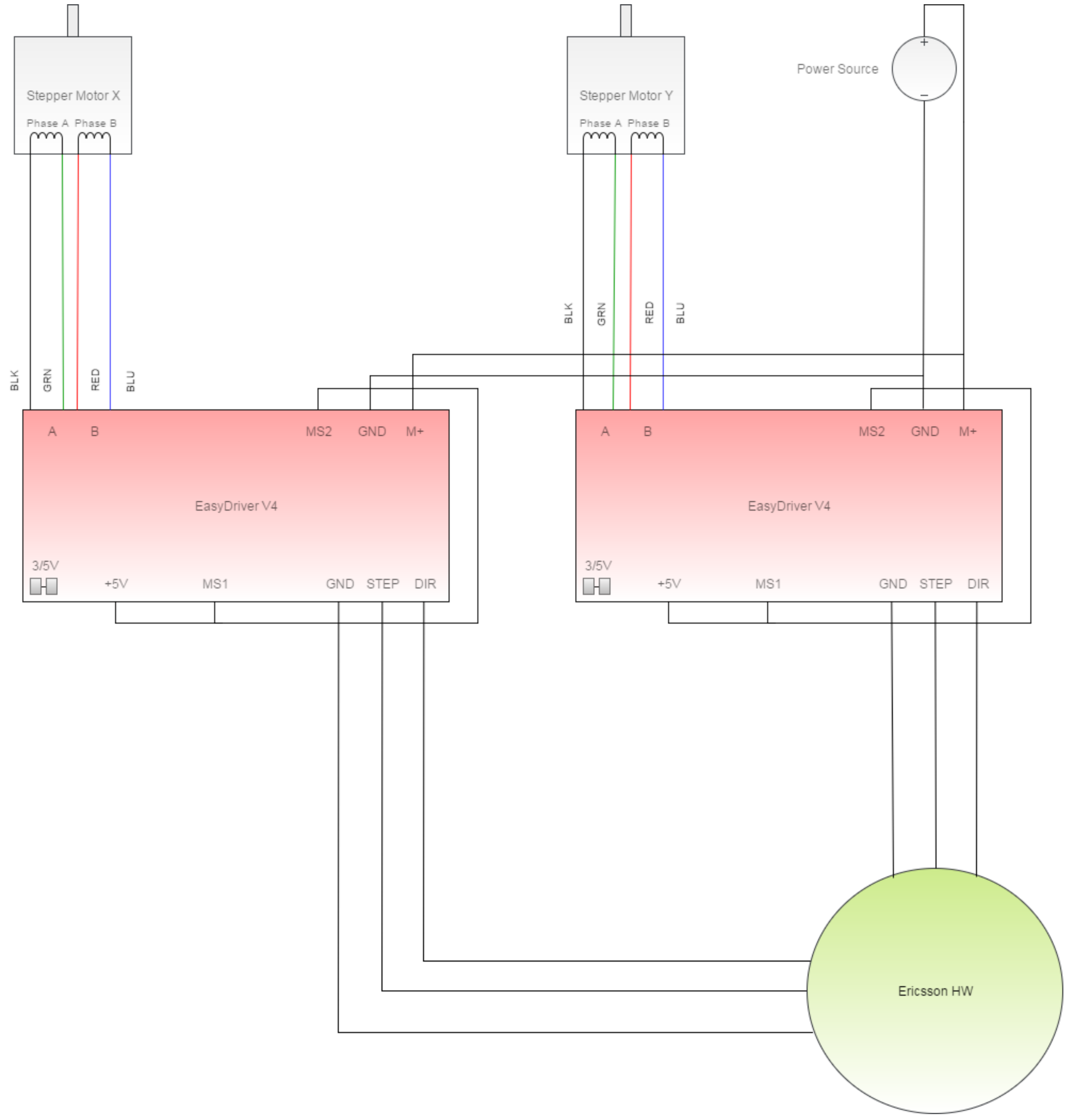
[2] "PRINCIP - STEGMOTOR," drivteknik, [Online]. Available: http://www.drivteknik.nu/skolan/motor/stegmotor. [Accessed 27 04 2017].

[3] "What is Python? Executive Summary," python, [Online]. Available: www.python.org/doc/essays/blurb/. [Accessed 04 05 2017].

APPENDICES

Appendix 1

Wiring diagram for stepper motors (chapter 5).



Appendix 2

Code used for simulation (chapter 7).

```
"""==========================================================================================


                                Antenna Algorithm

                    for automatic alignment of microwave antenna


                                    Made by

                              Gustav Lilliebrunner

                                 Binh Nugen




                                 Node set up:


                         X1 < -----      ------ > X2


=========================================================================================="""


import matplotlib.pyplot as plt                         #needed for graphic search print

import matplotlib.animation as animation                #needed for animation

import numpy as np

import random                                           #needed for simulate random movement at
node X2

from time import sleep                                  #needed for delay in step pulse


global fakeXMax                                         #set node X2 simulated maxpoints

global fakeYMax

fakeXMax = 1

fakeYMax = 1


global searchRadius                                     #set fine search radius

searchRadius = 20


global mapLimitX                                        #set to physical machine search limits or
less

global mapLimitY

mapLimitX = 88                                          #set to 266 for +- 30 deg azimuth
```

```
mapLimitY = 44                                        #set to 88 for +- 10 deg elevation


global xStep

global yStep

xStep = 22

yStep = 22


signalMap = []                                        #define RSSI signal matrix



def makeSignalMap():                                  #function creating RSSI signal map matrix

    global mapLimitX

    global mapLimitY

    for y in xrange(mapLimitY):

        signalMap.append([0] * mapLimitX)


def printSignalMapNummerical():                       #print signal map

    for x in range(len(signalMap)):

        print ' '.join(map(str, signalMap[x]))        #make output print without , and []

    print ""                                          #separate output print from each other


def printSignalMapGUI():                              #print RSSI signal map with graphic user
interface

    plt.matshow(signalMap)

    plt.show()



def getInputPower():                                  #dummy function to simulate RSSI signal

    global fakeXMax

    global fakeYMax

    global xStep

    global yStep

    x = abs((fakeXMax) - xStep)

    y = abs((fakeYMax) - yStep)

    hyp = (((x**2)+(y**2))**(0.5))                    #calculate vector length from simulated
max point to current point

    signal = 100 - int(hyp)                           #scale the signal value from 100 - 0
    if signal < 0:                                    #ignore negative signal value

        return 0

    else:

        #signal = int(signal * random.random())       #used to simulate the other node moving
at the same time

        if signal > (signalMap[yStep][xStep]):

            signalMap[yStep][xStep] = signal

        return signal




def dummySerch():                                     #dummy search that search through every
possibole descreate signal value

    global mapLimitX

    global mapLimitY

    global xStep

    global yStep

    for y in range(mapLimitY):                        #iterate through every possible discrete
point

        yStep = y

        for x in range(mapLimitX):

            xStep = x

            getInputPower()                           #get input power for every desecrate
point




def roughSearch():



"""=================================================================================================

    The roughSerch algorithm seach one time in an "S" pattern with help of moveToPosition
function. The moveToPosition function execute the command and map a signal value to each descrete
map point it passes by. The beam has a - 10 dB with of ~ +- 3.5deg, therefore ther is no need to
search the boundary points

    Search pattern:


    |---------------------------|

    |                           |

    |   1-------------------2   |

    |                       |   |

    |   4--------Cen--------3   |
    |   |                       |

    |   5--------------------6   |

    |                           |

    |---------------------------|


=================================================================================================="""
    global mapLimitX

    global mapLimitY

    cenX = mapLimitX / 2                                # find center x coordinate in map

    cenY = mapLimitY / 2                                # find center y coordinate in map

    boun = 15                                           # Boundary points

    maxSweepDis = 31                                    # Maximum distance between search sweep


    moveToPosition(0 + boun, cenY - maxSweepDis)        # move to point 1

    moveToPosition(mapLimitX - boun, cenY - maxSweepDis)# move to point 2

    moveToPosition(mapLimitX - boun, cenY)              # move to point 3

    moveToPosition(0 + boun, cenY)                      # move to point 4

    moveToPosition(0 + boun, cenY + maxSweepDis)        # move to point 5

    moveToPosition(mapLimitX - boun, cenY + maxSweepDis)# move to point 6


    moveToPosition(0 + boun, cenY + maxSweepDis)        # move to point 5

    moveToPosition(0 + boun, cenY)                      # move to point 4

    moveToPosition(mapLimitX - boun, cenY)              # move to point 3

    moveToPosition(mapLimitX - boun, cenY - maxSweepDis)# move to point 2

    moveToPosition(0 + boun, cenY - maxSweepDis)        # move to point 1



def fineSearch():

    """==================================================================

    The fineSearch algorithm take the coordinates for maximum signal point and search in a square
pattern around them.

    After each loop the a new max is discovered and the square radius is halved.

    |---------------------------|

    | 1-------2                 |

    | |       |                 |

    | |  max  |                 |
    | |       |                 |

    | 4-------3                 |

    |                           |

    |                           |

    |---------------------------|



    ==================================================================="""

    global searchRadius                                  #predefined radius of the fine search
area

    r = searchRadius

    while r > 0:

        maxValue, x, y = getMaxSignalValue()             # get coordinate for maximum input signal

        moveToPosition(x - r, y + r)                     # move to point 1

        moveToPosition(x + 1, y + r)                     # move to point 2

        moveToPosition(x + r, y - r)                     # move to point 3

        moveToPosition(x - r, y - r)                     # move to point 4

        moveToPosition(x - r, y + r)                     # move to point 1

        r = int(r/2)

        printSignalMapGUI()



def getMaxSignalValue():                                 #iterate through the RSSI signal map and
return the signal value and coordinates for that value

    global mapLimitX

    global mapLimitY

    maxSignalValue = 0

    xMax = 0

    yMax = 0

    for y in range(mapLimitY):

        for x in range(mapLimitX):

            if signalMap[y][x] > maxSignalValue:

                maxSignalValue = signalMap[y][x]

                xMax = x

                yMax = y

    return maxSignalValue, xMax, yMax
```

```
def moveToPosition(newXStep, newYStep):

    global xStep

    global yStep

    stepDelay = float(0.0001)

    global mapLimitX

    global mapLimitY


    while newXStep >= mapLimitX or newXStep < 0 or newYStep >= mapLimitY or newYStep < 0:   #make
sure the antenna do not move outside its physically limits

        while newXStep >= mapLimitX:                        #if new position is outside physical
limits -> move in new position

            newXStep -= 1

        while newXStep < 0:

            newXStep += 1

        while newYStep >= mapLimitY:

            newYStep -= 1

        while newYStep < 0:

            newYStep += 1


    while (xStep < newXStep) and (xStep != newXStep):       #move antenna until it is in new
position

        getInputPower()

        sleep(stepDelay)                                    #make small delay before next step

        getInputPower()

        sleep(stepDelay)

        xStep += 1                                          #update current position


    while (xStep > newXStep) and (xStep != newXStep):       #move antenna until it is in new
position

        getInputPower()

        sleep(stepDelay)                                    #make small delay before next step

        getInputPower()

        sleep(stepDelay)

        xStep -= 1                                          #update current position


    while (yStep < newYStep) and (yStep != newYStep):       #move antenna until it is in new
position
```

```
        getInputPower()

        sleep(stepDelay)                                    #make small delay before next step

        getInputPower()

        sleep(stepDelay)

        yStep += 1                                          #update current position


    while (yStep > newYStep) and (yStep != newYStep):       #move antenna until it is in new
position

        getInputPower()

        sleep(stepDelay)                                    #make small delay before next step

        getInputPower()

        sleep(stepDelay)

        yStep -= 1                                          #update current position




#main

makeSignalMap()

roughSearch()

printSignalMapGUI()

for i in range(50):

    roughSearch()

printSignalMapGUI()


fineSearch()

printSignalMapGUI()

dummySerch()

printSignalMapGUI()


print getMaxSignalValue()
```

Appendix 3

Code used with hardware (chapter 8).

```
"""==========================================================================================


                                Antenna Algorithm

                    for automatic alignment of microwave antenna


                                    Made by

                              Gustav Lilliebrunner

                                  Binh Nguyen




                                     Node set up:


                              X1 < -----      ------ > X2


=========================================================================================="""


import matplotlib.pyplot as plt                         #needed for graphic search print

import matplotlib.animation as animation                #needed for animation

import numpy as np

import random                                           #needed for simulate random movement at
node X2

import socket

from time import sleep                                  #needed for delay in step pulse


global fakeXMax                                         #set node X2 simulated maxpoints

global fakeYMax

fakeXMax = 38

fakeYMax = 22


global searchRadius                                     #set fine search radius in steps

searchRadius = 10


global mapLimitX                                        #set to physical machine search limits or
less

global mapLimitY
```

```
mapLimitX = 72                                        #set to 266 for +- 30 deg azimuth

mapLimitY = 44                                        #set to 88 for +- 10 deg elevation


global xStep

global yStep

xStep = 38                                            #define were the start position is

yStep = 22


signalMap = []                                        #define RSSI signal matrix


global stepDelay                                      #define variable used as delay

global s

global BUFFER_SIZE


def setAntennaSpeed(speed_set):                       #set speed in deg/s

    global stepDelay

    stepDelay = 1 / float(16 * speed_set)             #use half delay, because used 2 times
every step

    return stepDelay


def initiateServer():

    global s

    global BUFFER_SIZE

    TCP_IP = '192.168.1.1'                                #node server IP-Address

    TCP_PORT = 45554                                      #node server port

    BUFFER_SIZE = 1024                                    #input buffer size

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    s.connect((TCP_IP, TCP_PORT))

    return "done"


def callServer(message):

    global s

    global BUFFER_SIZE

    s.send(message)

    return s.recv(BUFFER_SIZE)


def closeServer():
    global s

    s.close()

    return "done"


def makeSignalMap():                                        #function to create RSSI signal map
matrix

    global mapLimitX

    global mapLimitY

    for y in xrange(mapLimitY):

        signalMap.append([-100] * mapLimitX)


def printSignalMapNummerical():                             #print function for signal map

    for x in range(len(signalMap)):

        print ' '.join(map(str, signalMap[x]))              #make output print without , and []

    print ""                                                #separate out print from each other


def printSignalMapGUI():                                    #print RSSI signal map with graphic user
interface

    plt.matshow(signalMap)

    plt.show()


def getInputPower():                                        #get signal values from node

    global s

    global BUFFER_SIZE

    global xStep

    global yStep

    s.send("rssi")

    signal = float(s.recv(BUFFER_SIZE))/100

    if signal > (signalMap[yStep][xStep]):

        signalMap[yStep][xStep] = signal

    return signal




def dummySerch():                                           #dummy search that search through every
possibole descreate signal value

    global mapLimitX

    global mapLimitY
    global xStep

    global yStep

    for y in range(mapLimitY):                           #iterate through every possible discrete
point

        yStep = y

        for x in range(mapLimitX):

            xStep = x

            getInputPower()                              #get input power for every desecrate
point




def roughSearch():


"""=================================================================================================

The roughSerch algorithm search one time in an "S" pattern with help of moveToPosition function.
The moveToPosition function execute the command and map a signal value to each descrete map point
it passes by. The beam has a - 10 dB with of ~ +- 3.5deg, therefore there is no need to search
the boundary points

    Search pattern:


    |---------------------------|

    |                           |

    |   1-------------------2   |

    |                       |   |

    |   4--------Cen--------3   |

    |   |                       |

    |   5-------------------6   |

    |                           |

    |---------------------------|



================================================================================================"""

    global mapLimitX

    global mapLimitY

    cenX = mapLimitX / 2                                 # find center x coordinate in map

    cenY = mapLimitY / 2                                 # find center y coordinate in map

    boun = 15                                            # Boundary points equal 3,5 deg

    maxSweepDis = 31                                     # Maximum distance between search sweep
    moveToPosition(0 + boun, cenY - maxSweepDis)        # move to point 1

    moveToPosition(mapLimitX - boun, cenY - maxSweepDis)# move to point 2

    moveToPosition(mapLimitX - boun, cenY)              # move to point 3

    moveToPosition(0 + boun, cenY)                      # move to point 4

    moveToPosition(0 + boun, cenY + maxSweepDis)        # move to point 5

    moveToPosition(mapLimitX - boun, cenY + maxSweepDis)# move to point 6


    moveToPosition(0 + boun, cenY + maxSweepDis)        # move to point 5

    moveToPosition(0 + boun, cenY)                      # move to point 4

    moveToPosition(mapLimitX - boun, cenY)              # move to point 3

    moveToPosition(mapLimitX - boun, cenY - maxSweepDis)# move to point 2

    moveToPosition(0 + boun, cenY - maxSweepDis)        # move to point 1



def fineSearch():

    """==================================================================

    The fineSearch algorithm take the coordinates for maximum signal point and search in a square
pattern around them.

    After each loop the a new max is discovered and the square radius is halved.

    |---------------------------|

    | 1-------2                 |

    | |       |                 |

    | |  max  |                 |

    | |       |                 |

    | 4-------3                 |

    |                           |

    |                           |

    |---------------------------|



    ==================================================================="""

    global searchRadius                                 #pre-defined radius of the fine search
area

    r = searchRadius

    while r > 0:

        maxValue, x, y = getMaxSignalValue()            # get coordinate for maximum input signal

        moveToPosition(x - r, y + r)                    # move to point 1
        moveToPosition(x + 1, y + r)                    # move to point 2

        moveToPosition(x + r, y - r)                    # move to point 3

        moveToPosition(x - r, y - r)                    # move to point 4

        moveToPosition(x - r, y + r)                    # move to point 1

        r = int(r/2)

        #printSignalMapGUI()



def getMaxSignalValue():                                #iterate through RSSI signal map and
return the signal value and coordinates for that value

    global mapLimitX

    global mapLimitY

    maxSignalValue = -1000

    xMax = 0

    yMax = 0

    for y in range(mapLimitY):

        for x in range(mapLimitX):

            if signalMap[y][x] > maxSignalValue:

                maxSignalValue = signalMap[y][x]

                xMax = x

                yMax = y

    return maxSignalValue, xMax, yMax





def moveToPosition(newXStep, newYStep):

    global xStep

    global yStep

    global stepDelay

    global mapLimitX

    global mapLimitY


    while newXStep >= mapLimitX or newXStep < 0 or newYStep >= mapLimitY or newYStep < 0:   #make
sure the antenna do not move outside its physically limits

        while newXStep >= mapLimitX:                    #if new position is outside physical
limits -> move in new position

            newXStep -= 1

        while newXStep < 0:
            newXStep += 1

        while newYStep >= mapLimitY:

            newYStep -= 1

        while newYStep < 0:

            newYStep += 1


    if xStep < newXStep:                                #decide which direction the antenna 
should move

        callServer("horpos")

    else:

        callServer("horneg")

    if yStep < newYStep:

        callServer("verpos")

    else:

        callServer("verneg")


    while (xStep < newXStep) and (xStep != newXStep):   #move antenna until it is in new position

        getInputPower()

        callServer("horon")

        sleep(stepDelay)                                #make small delay before next step

        callServer("horoff")

        getInputPower()

        sleep(stepDelay)

        xStep += 1                                      #update current position


    while (xStep > newXStep) and (xStep != newXStep):   #move antenna until it is in new position

        getInputPower()

        callServer("horon")

        sleep(stepDelay)                                #make small delay before next step

        callServer("horoff")

        getInputPower()

        sleep(stepDelay)

        xStep -= 1                                      #update current position


    while (yStep < newYStep) and (yStep != newYStep):   #move antenna until it is in new position

        getInputPower()

        callServer("veron")
        sleep(stepDelay)                                #make small delay before next step
        callServer("veroff")
        getInputPower()
        sleep(stepDelay)
        yStep += 1                                      #update current position

    while (yStep > newYStep) and (yStep != newYStep):   #move antenna until it is in new position
        getInputPower()
        callServer("veron")
        sleep(stepDelay)                                #make small delay before next step
        callServer("veroff")
        getInputPower()
        sleep(stepDelay)
        yStep -= 1                                      #update current position



if __name__ == "__main__":
    initiateServer()
    makeSignalMap()
    setAntennaSpeed(int(input("Input speed in deg/s: ")))
    printSignalMapGUI()
    for i in range(1):
        roughSearch()
    printSignalMapGUI()
    fineSearch()
    printSignalMapGUI()
    print getMaxSignalValue()
    closeServer()
```