DEGREE PROJECT IN TECHNOLOGY,
FIRST CYCLE, 15 CREDITS
STOCKHOLM, *SWEDEN 2021*

Kinetic Art Table

Polar sand plotter

SERHAT TÜRK

KRISTOFFER MÜLLER

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF INDUSTRIAL ENGINEERING AND MANAGEMENT

Kinetic Art Table

Polar sand plotter

SERHAT TÜRK, SERHATT@KTH.SE
KRISTOFFER MÜLLER, KMUL@KTH.SE

Bachelor's Thesis at KTH
Supervisor: Nihad Subasic
Examiner: Nihad Subasic

TRITA-ITM-EX 2021:19

Abstract

CNC machines are used with plenty of different implementations, one of which is in this project where a polar CNC machine was used to draw mesmerizing patterns on a table with fine sand. This construction read G-code and converted it to polar coordinates. The capabilities of what the plotter could draw were tested, everything from ODE plots to custom-made patterns and drawings with the help of Sandify. Although the patterns were drawn properly with small errors the ODE was too difficult to draw because it required a smaller magnetic ball and an even more precise system than what was used. This machine also generated noise at roughly 33 dB when it was in use.

Keywords: Mechatronics, Stepper-motor, Arduino, Polar plotter, forward Euler method.

Referat

CNC-maskiner används med massor av olika implementationer, en av dem är i det här projektet där en polar CNC-maskin användes för att rita fascinerande mönster på ett bord fylld med fin sand. Denna konstruktion läste in G-kod och konverterade det till polära koordinater. Förmågan av vad maskinen kunde rita testades, allt från ODE grafer till specialtillverkade mönster och ritningar med hjälp av Sandify. Även om de olika mönstren ritades ordentligt men med mindre små fel var ODE för svårt att rita på grund av att det krävde en mindre magnetisk kula och ännu mer noggrannhet jämfört med detta system. Denna maskin alstrade också ljud på cirka 33 dB under användning.

Nyckelord: Mekatronik, Steg-motor, Arduino, Polär plotter, Eulers stegmetod.

Acknowledgements

We would like to thank our examiner Nihad Subasic for increasing our understanding of the field of mechatronics and giving us a proper base of knowledge to proceed with this construction. Thank you goes out to the assistants Amir Avdic and Malin Lundvall for giving us advice in great times of need and guide us to the right path. We would also like to thank our fellow students Kristian Jandric, Algot Lindestam, and Viktor Kårefjärd that helped us with some of the programming parts of the project. A big thanks to Staffan Qvarnström for helping us with the purchases of parts and different electronics. This project would not have been possible without these humbling people.

Contents

1 Introduction **1**
- 1.1 Background . . . 1
- 1.2 Purpose . . . 1
- 1.3 Scope . . . 2
- 1.4 Method . . . 2

2 Theory **3**
- 2.1 Motor . . . 3
- 2.2 Different moving mechanisms . . . 4
- 2.3 Arduino . . . 4
- 2.4 Shields . . . 5
- 2.5 Ordinary Differential equations . . . 5
  - 2.5.1 Forward Euler Method . . . 6

3 Demonstration **7**
- 3.1 Problem formulation . . . 7
- 3.2 Electronics and components . . . 7
  - 3.2.1 Microcontroller . . . 7
  - 3.2.2 Stepper motors . . . 8
- 3.3 Software . . . 9
  - 3.3.1 System control . . . 9
  - 3.3.2 G-code . . . 9
  - 3.3.3 Generate coordinates . . . 10
  - 3.3.4 Storage . . . 11
  - 3.3.5 Solving the ODE . . . 11
- 3.4 Hardware . . . 12
  - 3.4.1 Computer-aided design . . . 12
  - 3.4.2 Rotating part . . . 13
  - 3.4.3 Linear part . . . 14
  - 3.4.4 Linear and rotation . . . 15
  - 3.4.5 Table . . . 16

4 Application testing and results **19**

4.1 Hardware 19
4.2 Ratio between linear and rotating part 19
4.3 Noise comparison of the sand 20
4.4 Patterns 21

5 Discussion and conclusion **23**

Bibliography **25**

6 Appendix A **29**

7 Appendix B **33**

8 Appendix C **54**
8.1 Stepper motor datasheet 54
8.2 l293d motor shield datasheet 55

List of Figures

2.1 The principle of a stepper motor.[17] 4
2.2 Arduino Uno, Imagen take by Kristoffer Müller 5

3.1 Connection diagram and overview of the electronics made in the program Fritzing. [16] 8
3.2 This is a pattern generated with the Sandify program. [12] 10
3.3 Flowchart made with app.diagrams.net.[19] 11
3.4 Finished CAD assembly in Solid Edge. [1] 13
3.5 Rotating part in Solid Edge. [1] 14
3.6 Linear part in Solid Edge. [1] 15
3.7 ρ and θ. Image taken by Serhat Türk. 16
3.8 Final table. Image taken by Serhat Türk. 17

4.1 Spiral in Sandify.[12] 21
4.2 Spiral drawing on table. Image taken by Serhat Türk. 21
4.3 Spiral into a star in Sandify.[12] 21
4.4 Expanding star drawn after a spiral. Image taken by Serhat Türk. . . . 21
4.5 Square drawn in Sandify. [12] 22
4.6 Square drawing test in sand. Image taken by Serhat Türk. 22
4.7 The ODE plot in the sand. Image taken by Serhat Türk. 22
4.8 The ODE displayed in Matlab. [13] 22

6.1 Mechanical parts side view. Image taken by Serhat Türk. 29
6.2 Mechanical parts over view. Image taken by Serhat Türk. 30
6.3 Table side view. Image taken by Serhat Türk. 30
6.4 Table over view. Image taken by Serhat Türk. 31
6.5 Finished construction. Image taken by Serhat Türk. 31

List of Abbreviations

- CNC - Computer Numerical Control
- DC - Direct current
- SEK - Swedish Kronor
- IDE - Integrated Development Environment
- PMW - Pulse-Width Modulation
- USB - Universal Serial Bus
- 3D - Three-dimensional space
- 2D - Two-dimensional space
- ODE - Ordinary Differential Equation
- CAD - Computer-aided design
- SKF - Svenska Kullagerfabriken
- mm - Millimeters
- min- Minimum
- max - Maximum
- dB - Decibel

List of Tables

4.1 Noise measured with Sound Meter. [23] 20

Chapter 1

Introduction

1.1 Background

Being able to create mesmerizing patterns has always been a huge part of human creativity throughout centuries. Having a machine that expresses those patterns while people are taking a coffee break would be soothing. Not only would it be entertaining but also a fascinating way to recreate differential equations on a layer of sand.

This thesis will improve the understanding of how to use coding and computing to replicate and calculate digital movement in a real-life machine. This type of technology is called CNC, also known as Computer Numerical Control. CNC essentially uses two or three-dimensional movements to complete certain tasks.

Usually, CNC technology is utilized to complete different tasks such as control of workshop machines, 3D printers, or in the medical field where a tool like this can be used with precise movements for surgery. However, this technology could also be used to create something which is the opposite of that. In this case, it will be used to move a magnetic ball in very fine sand that prevents jagged drawings and disturbing noise.

1.2 Purpose

The purpose of this project was to build and program a table that draws patterns in fine sand using a magnetic ball, motors, and 3D-printed parts for the arms. The following thesis and questions will be answered:

- How to design and build an arm allowing movement in two dimensions
- How to create a program which allows user input and to print patterns in sand?
- Is it possible with the program to plot approximations of Ordinary differential equations into the sand?

- How to reduce the sound for different parts of the machine?

1.3 Scope

This thesis will be limited so that it becomes possible to complete the project with the given time and resources. The budget was limited to 1000 SEK and because of the corona virus pandemic, some places in the university were limited. The main goal was to construct a functioning art table, therefore the sound analysis was not prioritized. Another limiting factor was the storage of the Arduino. With regards to that, the focus of plotting differential equations will be on Ordinary Differential Equations (ODE).

1.4 Method

To create this table system multiple tools were utilized. At first, to create a prototype and then later on the finished product, a program called Solid Edge was used. Solid Edge is a Computer-Aided Designing program that makes 3D models digitally. [1] With the use of that program, it was possible to later on 3D print some parts of the project and the other parts were ordered online from Electrokits website [2] such as stepper motors and a bearing. Those parts together with an Arduino UNO, L293D motor shield as well as software and code from a computer to control the motors led to this project construction.

Chapter 2

Theory

2.1 Motor

For this build, the system will revolve around motors to achieve movement in two dimensions. There are different types of motors that can be used in projects. Direct current (DC) motors are one of them. The DC motor has a wide variety of applications. The way they work is by simply applying voltage for the motor to start spinning. If the direction of the current changes, so will the rotation of the motor. The basic principle of DC motors is simply using electricity and magnetism to make the motor rotate, usually with electromagnets and normal magnets. With that comes different types of motors. One of them is the servo motor. [6]

Servomotors are often seen in robotic arms and it is most likely because of their compact form factor and using their feedback to control their movement. The closed feedback system makes it easy to control exactly how much the motor is supposed to rotate and keeping it constant is not difficult. The feedback system often consists of a sensor that keeps track of the rotor. [9]

Another motor is a stepper motor. A stepper motor is essentially a brushless DC motor that can move with precision which was necessary for this build. It achieves accurate movement because the motors are constructed with multiple toothed electromagnets that surround a gear in the center as seen in figure 2.1. The electromagnets are then utilized by multiple micro-controllers or driver circuits to drive the iron gear. It is done by powering on and off these electromagnets surrounding the gear which is dependant on the alignment of the cogs on the gear. When the cogs are aligned with one of the electromagnets, the next one will be slightly offset. Once that is the case, the aligned one will turn off and automatically power on the offset electromagnet, which in turn makes the gear spin.

The position and exact movement of the motor is accurate which is essential to always know when and where the magnet will be on the table. [7]

The stepper motor also has its benefits with high torque at low speed but this requires higher current. Another asset of the motor is that by changing the direction of the current, the motor can then be used to move forward and backward.

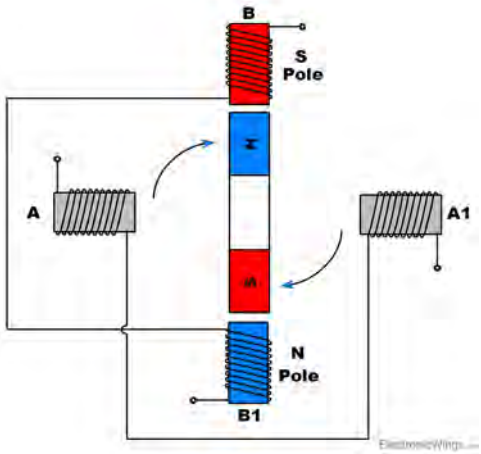


Figure 2.1: The principle of a stepper motor.[17]

2.2 Different moving mechanisms

There are different ways to systematically move a ball in a limited orientation. The balls are going to move in a two-dimensional plane.

The first possible build was to use normal Cartesian coordinates the same way a 3D printer works except it is in 2D. It uses a square-based system to move the ball using x and y coordinates. Although this should be easy to build and program it takes up space.

The second system is more challenging which is polar coordinates. This makes use of an angle and a distance from the origin to calculate the position of the ball and where it is going to move, which makes this a better choice since a lot of patterns that are drawn will have circular characteristics. The space of the table might also be smaller compared to the Cartesian system.

2.3 Arduino

The open-source electronics platform Arduino offers simple and easy-to-use hardware and software such as boards and Integrated Development Environment (IDE) software. It offers all kinds of micro-controllers for different usages and comes in different sizes making it a useful controller for various projects: from small school projects to more advanced projects.[3] Arduino boards are able to read input signals and transmit output signals to various objects such as motors or sensors. The IDE software used to program the Arduino is based on C and open source. The Arduino boards are based on the ATMega microcontroller. All of this makes the Arduino a good controller for this Bachelor's thesis. [4]

One of the most popular Arduino boards is the Arduino UNO as seen in figure 2.2, which was used in this project. It was powered by an ATmega328 processor

which operates at 16MHz. It includes 14 digital Input/Output pins, 6 analog pins, and supports 5V and 3.3V of power.[8] The 6 analog pins 3, 5, 6, 9, 10, 11 can be used as Pulse Width Modulation, or PWM, outputs.

Figure 2.2: Arduino Uno, Imagen take by Kristoffer Müller

The Arduino board consists of two parts. The first part is the hardware. The Arduino Uno consist of many different components such as the Digital Pins, USB connector, Reset Switch, Power Port, and a micro-controller which all together make it function. The second part is the software which consists of the Integrated development environment which translates the code the user writes to the language the Arduino reads.

2.4 Shields

An Arduino Shield is a modular circuit, often simplifying a specific task or giving the Arduino extra functions, for example, an Ethernet Shield, that is piggybacked onto the Arduino. An Arduino L293D Motor Driver Shield allows the Arduino to operate DC motors, stepper motors and different kinds of relays. It supports up to 4 DC motors or 2 Stepper and 2 servo motors.[24] It also features an extra power port enabling to connect motors having voltages between 4.5 to 25V.[6] [27]

2.5 Ordinary Differential equations

This build should theoretically be able to plot graphs or other equations in the sand other than just patterns. Another feature is that it could potentially use different methods to approximate the solution to an ODE and then draw it in the sand. Normally when a function is drawn in a calculator or other tools it is easy to draw

a function that is simple or even more complex. For instance to draw $y = x^2$ is not very difficult. However, to go even further and seeing if drawing ODE in the sand is possible would test the upper limits of the construction. There are different ways to plot differential equations, one is to actually solve it mathematically to get a proper plot or graph. The other option is to approximate the values of the solution using the unsolved equation instead as a baseline together with iterative calculations.

2.5.1 Forward Euler Method

The method used to approximate any ODE is called the forward Euler method. The requirement to use this method is to have a starting position of the ODE as well as the ODE itself. With the Euler formula

$$y_{n+1} = y_n + hf(t_n, y_n) \tag{2.1}$$

It is possible to iterate multiple points for each ODE. The formula utilizes the derivative of the function in the ODE to take steps. Meaning that it takes one point, tries to predict where the next point is supposed to be using a tangent on that point. From the tangent, it moves with a small step h in the tangent direction. Where the h is how big of a step the iteration is taking between each calculated point. h decides how sharp and accurate each calculated point will be compared to the exact solution. Once the new point is found in the tangent, a new tangent is calculated and the procedure continues until the boundary limits are met. The $f(t_n, y_n)$ is essentially set to what the equation is equal to, usually the derivative in the equation, one example is y' for $y' = y$. The t_n defines the interval of the equation which is usually on the x-axis. Finally, the y_n are the answers to each solution that is used as input, the y_{n+1} is then set back to y_n so the iteration can continue until the end of t_n.[18]

Chapter 3

Demonstration

3.1 Problem formulation

During the project there were some problems and obstacles that had to be cleared:

- Convert a pattern or an ODE into polar coordinates and draw these coordinates in sand using motors and gears.
- Design proper mechanical parts for the construction.
- Write code to calibrate the system.

3.2 Electronics and components

Some components were purchased and the rest was 3D printed. For this design of the project it was necessary to have:

- two stepper motors
- an Arduino UNO
- a belt drive kit
- different 3D printed parts
- two linear bearings
- axial bearings
- Arduino L293D Motor Driver Shield

3.2.1 Microcontroller

To control the kinetic table an Arduino UNO with an Arduino L293D Motor Driver Shield was used. The Driver Shield was piggybacked onto the Arduino UNO. Both stepper motors were connected to the Driver Shield.

3.2.2 Stepper motors

The polar CNC table was driven by two of the same stepper motors. One that moved the arm and the other one that was driving the angle of the arm itself. Both of them were bipolar stepper motors with a step-angle of 0.9° and sustainability of 0.41 Nm from the website electrokit. [11] How everything was set up can be seen in figure 3.1.

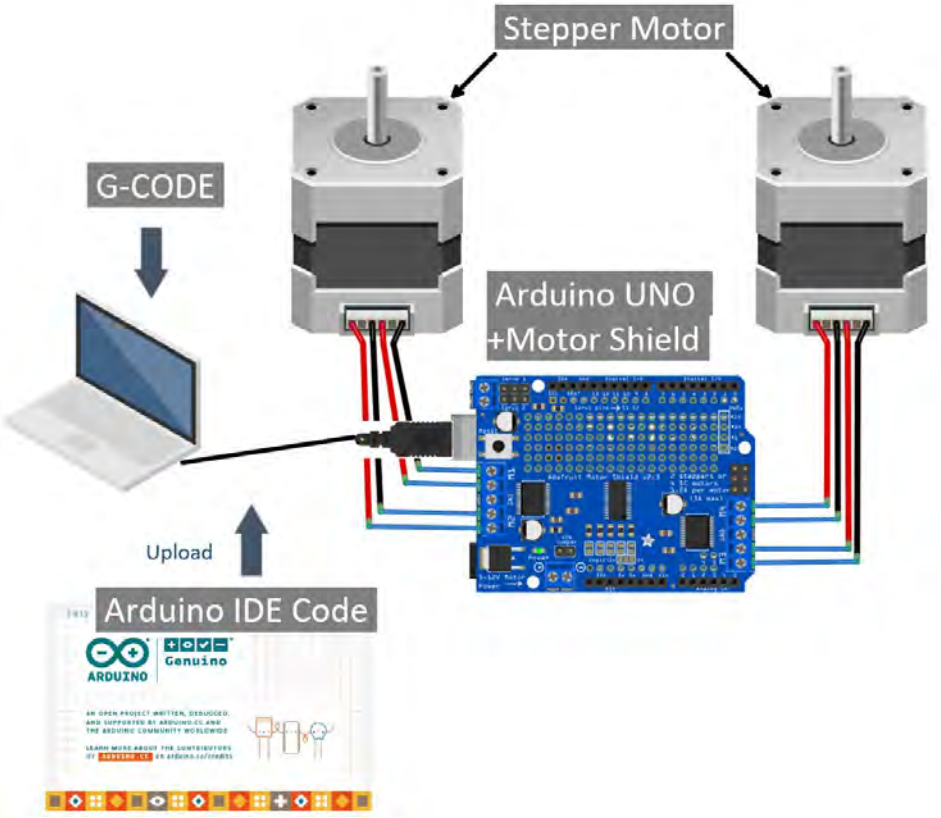


Figure 3.1: Connection diagram and overview of the electronics made in the program Fritzing. [16]

3.3 Software

3.3.1 System control

The code that was written in the Arduino can be seen under Appendix B. This code manages and controls how the motors were supposed to operate and move together to draw properly in the sand. The coordinates were downloaded and used as input for the Arduino. The prerequisite settings for the stepper motors were to calculate how long one step from the motors were, as well as the area on which the machine could draw on. The motors always made the metal ball move in straight lines from point to point mapped out by G-code which will be explained under section 3.3.2. Each point was usually close to one another and makes the straight-line hard to see. From there the code calculates an angle and the distance from the origin to understand where the magnet ball was supposed to position itself. The lines and coordinates, however, had to be drawn continuously without any jumps because the magnet arm could not move in the Z- directions.[27] So the coordinates need to stay consistently close to each other. The base of the codes for the project was found online but had to be modified and changed to work with this construction. [15]

3.3.2 G-code

G-code is basically the programming language generally used in 3D printers or other machines similar to CNC machines, where each row in the code represents the actions together with a position and speed. An example of one block written in G-code looks something like this: G01 X240 Y250. The G01 tells the machine to move in a straight line, and the X240 and Y250 translate to coordinates of a point where that straight line is supposed to move towards.[26] The numbers are usually in mm.[5] There are plenty of different functions in G-code but these are the only ones necessary for the machine to work. Since the code is used by 3D printers it also takes care of the z-axis. This was not considered in the project because the magnetic ball for this build could not move in that direction. The speed did not need to be modified either which is why it stayed constant when it performed.[14]

Although the system was built with polar characteristics, the inputs for the Arduino were in G-Code which is read in Cartesian X and Y coordinates. The Arduino had to recalculate the coordinates from the G-code and describe them with a ρ distance from the origin and a θ angle from the X-axis.[25]

3.3.3 Generate coordinates

To generate coordinates, the program Sandify [12] was used. The interface can be seen in figure 3.2. It allowed the creation of different patterns and output different points in terms of G-code. There were different basic patterns like a star shape, circle, and polygon which could be customized and added into more complex patterns. Other components could also be customized such as:

- Size of the pattern can also be scaled with a mathematical function
- Offset in X and Y-axis
- Rotation of the pattern
- The spin of the shape can also be scaled with a mathematical function
- The green dot indicates the starting point and the red dot is the end point

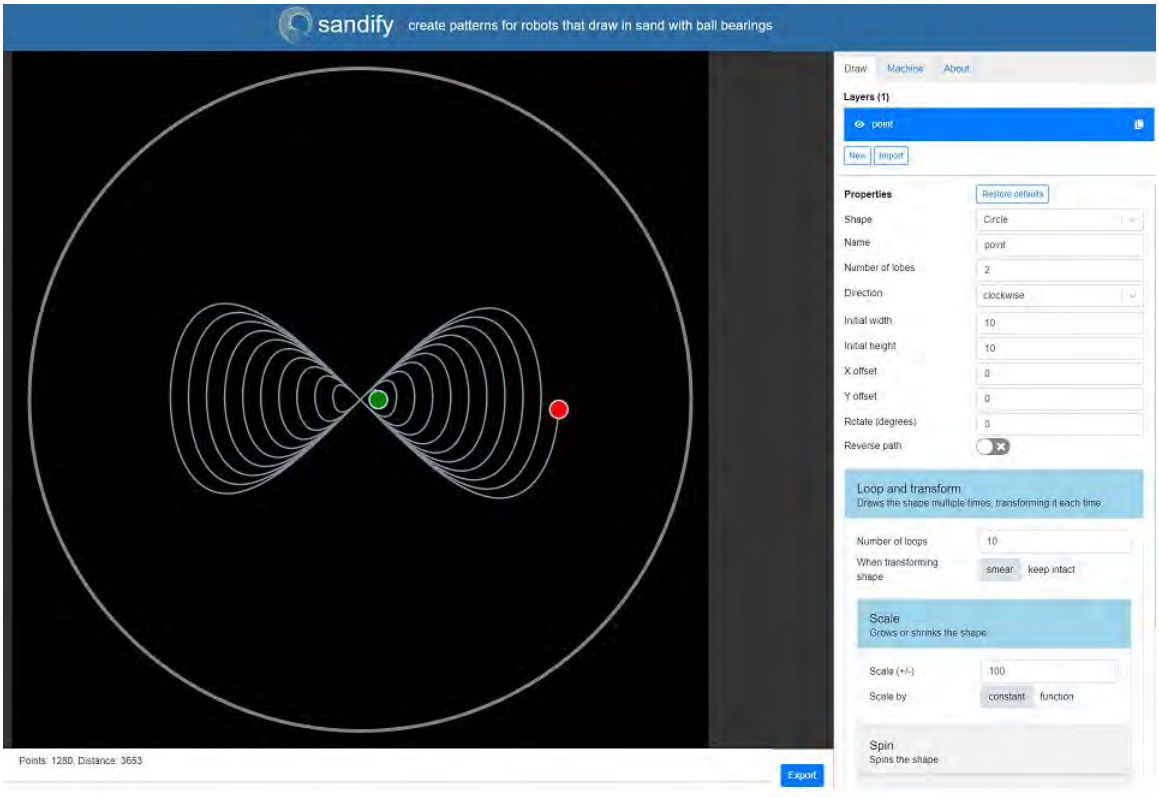


Figure 3.2: This is a pattern generated with the Sandify program. [12]

The way the entire process works from the Arduino reading G-code to the metal ball being moved was easily explained with a flowchart in figure 3.3.

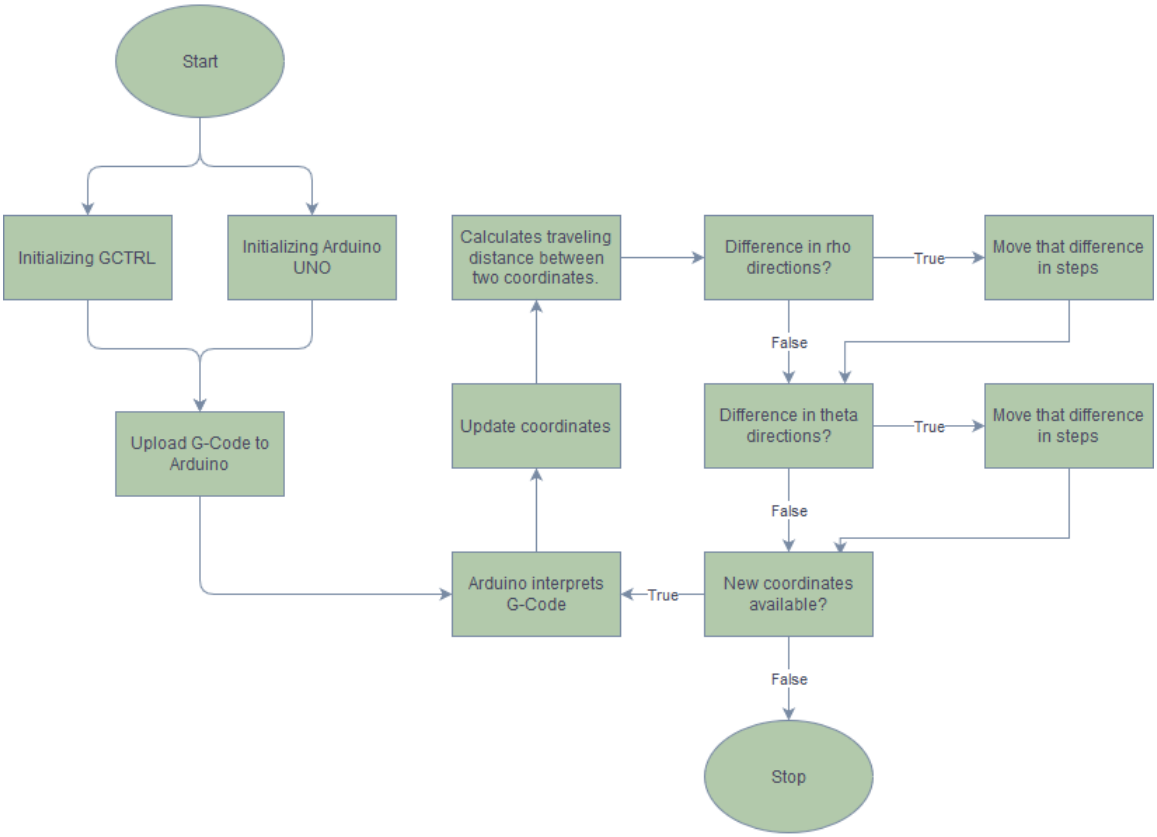


Figure 3.3: Flowchart made with app.diagrams.net.[19]

3.3.4 Storage

To store these patterns and send them over to the Arduino required a memory card. Since all the Arduino input pins where occupied its was not possible to install a memory card. The solution was to save all the files from the website Sandify to a computer and to send it to the Arduino through a code called GCTRL written in Processing 3.[20] Processing 3 is a software-based of Java code with added functionalities to make it easier to create mechanical art. What GCTRL essentially did was to read text files and sends the information over to the Arduino where the rest of the interpreting was done.[15]

3.3.5 Solving the ODE

For the ODE solution at first, the Arduino was used to solve the differential equations. Due to the limited storage on the board, another solution was required. Matlab is capable of creating and editing text files with the command `fprintf`. This made it possible to solve the ODEs in Matlab and then export the points given

by the Euler method in G-code format. From there GCTRL was used to upload the G-code similarly to the patterns download from Sandify. Allowing the usage of Matlab which can solve more complex differential equations than first anticipated when starting with the project.

3.4 Hardware

3.4.1 Computer-aided design

To be able to create the mechanism for the table, a 3D model was created with the CAD program Solid Edge.[1] All of the created parts were based on the stepper motors dimension. The larger pulley and the axial bearings had both premade CAD files which were imported into the project. The remaining parts were all created to fit with the pre-existing parts. The finished project consisted of two main parts, the rotating part which was responsible for the rotation in theta direction, and the linear part which was responsible for the radius in the ρ direction. This can be seen in figure 3.4.

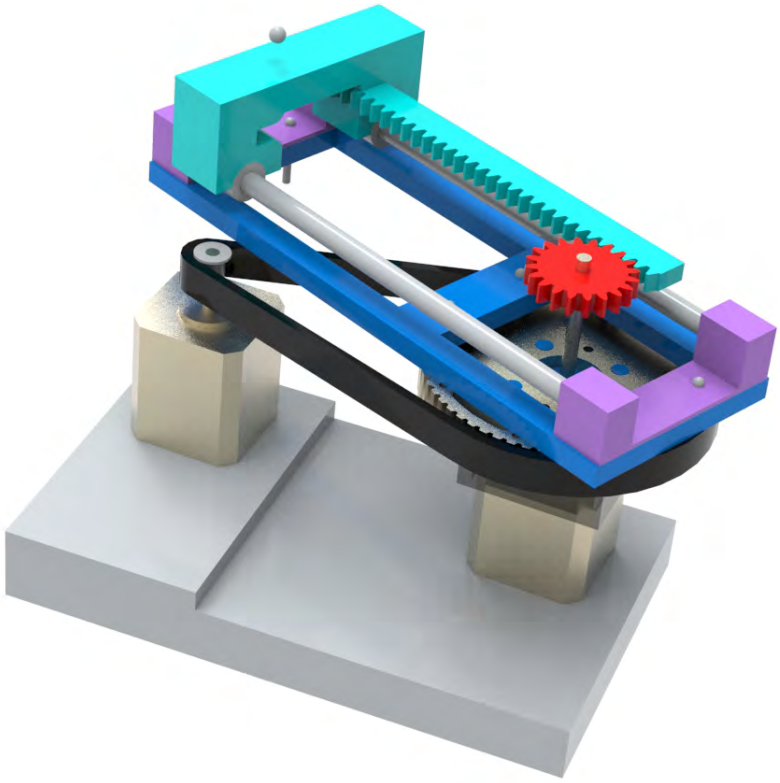


Figure 3.4: Finished CAD assembly in Solid Edge. [1]

3.4.2 Rotating part

The rotation was made using a stepper motor connected with a belt drive which can be seen in figure 3.5. The whole belt drive system was bought. [10] Given the smaller pulley having 10 numbers of teeth and the larger pulley, 60, the gear ratio can be calculated using equation 3.1.

$$Gear\ ratio = \frac{\#\ teeth\ large\ pulley}{\#\ teeth\ small\ pulley} = \frac{60}{10} = 6 \tag{3.1}$$

The larger pulley was mounted on top of the second stepper motor. To be able to rotate the pulley independently from the stepper motor it was mounted on an axial bearing which was mounted between the pulley and the stepper motor.
The axial bearing was a 51104 from SKF and both stepper motors were a 42BYGHM809 from JiangSu WanTai Motor Co., Ltd. Stepper motor datasheet seen under appendix C 8.1.

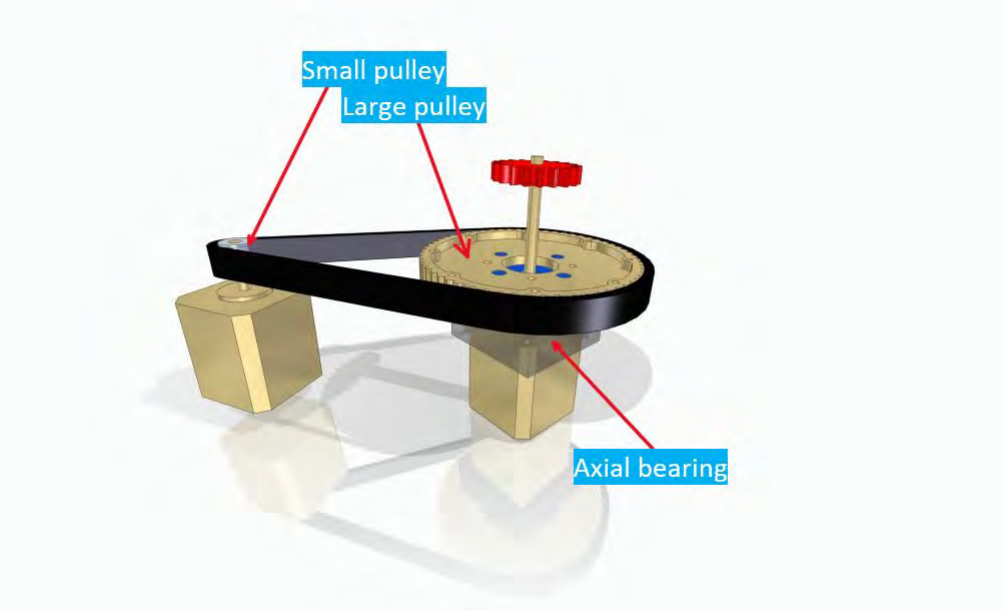


Figure 3.5: Rotating part in Solid Edge. [1]

3.4.3 Linear part

The linear movement was created using a pinion mounted to the stepper motor axis. The rack was fixated onto the magnet carrier seen in figure 3.6. The magnet carrier was mounted on two linear bearings which were on two axes only allowing linear movement. The rack and pinion, magnet carrier, and the mounts for the axis were all 3D printed. The linear bearing and axis were bought. [10] The construction for this project was designed in a way that when the rotating part was moved the linear part would also move along. To determine the ratio between the linear and rotating part a few tests were conducted, see section 4.2.

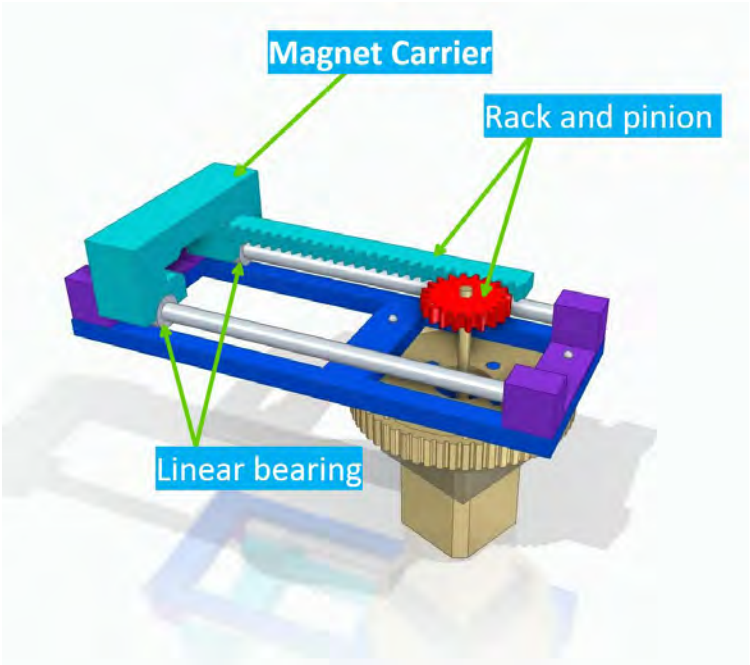


Figure 3.6: Linear part in Solid Edge. [1]

3.4.4 Linear and rotation

For both the linear and rotational systems to work simultaneously, they had to be built so that the cables did not tangle and move at all. This is why the system is built around the motors. For the ρ and θ movement, an example can be seen in figure 3.7.

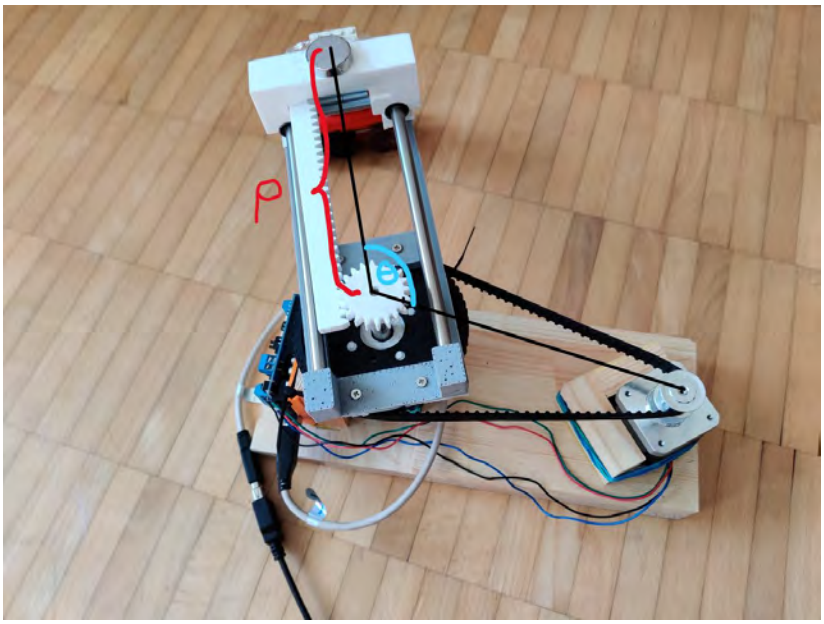


Figure 3.7: ρ and θ. Image taken by Serhat Türk.

3.4.5 Table

To be able to draw patterns in the sand a table was designed to hold the sand and have the right diameter to support the arm. The table was made out of wood. The main parts for the table are the body which holds the sand and the legs, see figure 3.8.

The sand which was used was at first normal aquarium sand[21], due to it being too rough it was creating too much noise. The size of the grains was roughly between 0,4 mm - 1,4 mm. The second sand which was used was chinchilla sand[22] which was more silent compared to the aquarium sand. A comparison was conducted, see section 4.3.

Figure 3.8: Final table. Image taken by Serhat Türk.

Chapter 4

Application testing and results

4.1 Hardware

The first problem that came up when the build was completed was that the linear and rotating part was too heavy to be balanced on their own. The easiest solution was to attach a wheel and a LEGO structure to the construction to support the weight. This helped the construction to maintain the balance even when the weight of the magnet carrier was shifted throughout the process of creating patterns.

4.2 Ratio between linear and rotating part

The ratio between the linear moving part and the rotating part was estimated with a simple test. The rotating part was set to move 100 steps around the own axis, by then measuring how many rotations the linear part would make a ratio of approximately 6:1 was estimated. That ratio was implemented into the code so that when the rotating part would take 6 steps the linear part compensates with one extra step so that the arm does not move forward or backwards while it rotates.

4.3 Noise comparison of the sand

Due to the difference in the size of the sand grains the noise produced by rolling a metal ball over it is different. The noise was measured using an app for the phone [23] and the values for the three different tests can be seen in table 4.1. These values were measured one meter above the table.

Sand type		Aquarium sand	Chinchilla sand
Size of the grain		0,4 mm - 1,4 mm	0,1 mm - 0,3 mm
Test 1	Min:	32 dB	22 dB
	Max:	61 dB	50 dB
	Average:	49 dB	33 dB
Test 2	Min:	30 dB	22 dB
	Max:	60 dB	47 dB
	Average:	47 dB	34 dB
Test 3	Min:	32 dB	23 dB
	Max:	55 dB	48 dB
	Average:	46 dB	32 dB

Table 4.1: Noise measured with Sound Meter. [23]

From table 4.1 the chinchilla sand was more silent than the aquarium sand. Not only did the finer chinchilla sand provide a more silent drawing, but it also gave a smoother path.

4.4 Patterns

The table is supposed to make different patterns in the sand. To test the accuracy and how well the mechanism works some simple and little more complex ones were tested.

The first one was a spiral that started in the middle and was drawn outwards eight spins. As seen in figure 4.2 and the corresponding drawing in Sandify in figure 4.1.

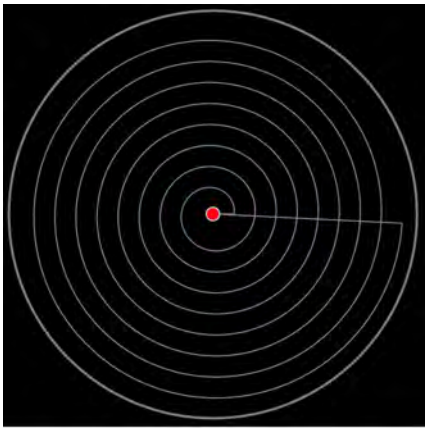


Figure 4.1: Spiral in Sandify.[12]

Figure 4.2: Spiral drawing on table. Image taken by Serhat Türk.

This was well drawn with some minor vibration patterns in the path of the ball as well as slightly worse resolution at the end spin.

The second test was to draw the same spiral but with an expanding star on top afterward to test if drawing a second pattern right after the first one is possible as seen in figure 4.4 and figure 4.3.

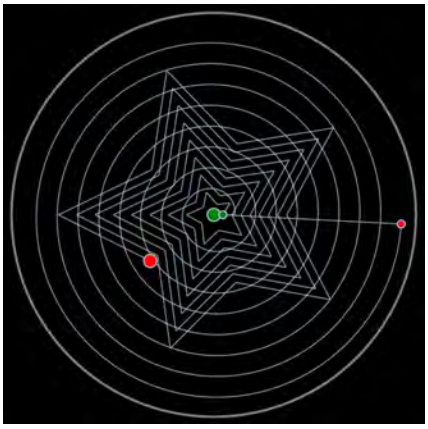


Figure 4.3: Spiral into a star in Sandify.[12]

Figure 4.4: Expanding star drawn after a spiral. Image taken by Serhat Türk.

Although the patterns were drawn properly, the drawing kept getting stuck on the Arduino cable and therefore the drawing was not exactly as anticipated.

Since the coordinates were converted from cartesian to polar. A test to draw a default square was done as seen in figure 4.6 and from Sandify in figure 4.5.

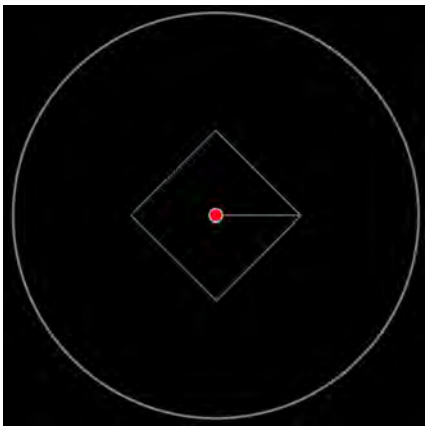


Figure 4.5: Square drawn in Sandify. [12]

Figure 4.6: Square drawing test in sand. Image taken by Serhat Türk.

The ordinary differential equation that was tested was

$$y' = \frac{x^3}{10} + \frac{x^2}{2} + 2x - 8. \tag{4.1}$$

To use the Euler method in Matlab the initial value was $y(1) = 0$ with the increments of $h = \frac{1}{5}$.

The resulting drawing in the sand is seen in figure 4.7. The ball rotated around the center of the table before the straight line was drawn. For comparison, figure 4.8 was the plot made in Matlab using the Euler method approximation.

Figure 4.7: The ODE plot in the sand. Image taken by Serhat Türk.

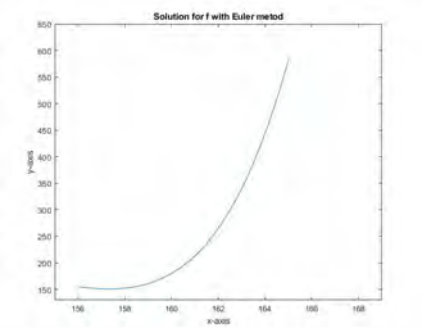


Figure 4.8: The ODE displayed in Matlab. [13]

Chapter 5

Discussion and conclusion

The goal for this project was to see if it was possible to draw patterns and differential equations. This was achieved using two stepper motors and several 3D printed and bought parts.

For the construction, there were multiple flaws. The first problem was with the linear motor, axial bearing, and larger pulley arrangement. The 3D printed carrier for the axial bearing was slightly skew which gave a slight shift in the weight distribution.

Another flaw in the construction was that the weight of the metal axes was underestimated. To solve this problem a quick and easy solution was provided in form of adding a LEGO structure with some wheels to counteract the weight of the axis. Another problem that occurred was that the wheels rolled over the USB cable connection to the Arduino. That resulted in a wobbling construction and sometimes the arm got stuck on the cable.

Other flaws were mainly related to the 3D printers not having sufficient precision and reoccurring technical difficulties. Which resulted in some parts sightly faulty constructed.

For the code part of the project drawings that lines crossing the y-axis, x-axis, or other straight lines could not be drawn from Sandify. Since the Arduino had to convert points from G-code to polar coordinates that automatically came with some conversion problems. For instance, to draw a square in the middle of the table. When the square is drawn in Sandify, the drawing only requires coordinates in each corner to be drawn. However, when later converted to polar coordinates on the Arduino, the conversion did not take care of the ρ difference between each corner. This was because the ρ was the same distance from the origin to each corner of the square. Meaning that drawing a square would result in a circle. What the drawing needed was one extra coordinate in between each corner so that the ρ difference was accounted for. This could have been solved by either inputting extra coordinates manually in between each point or use some form of interpolation.

Some complex patterns could not be drawn properly simply because of the extensive coordinate changes. Having 180 $^\circ$ difference between each coordinate

gave some difficulty converting that to proper linear movement since the entire build had to rotate 180 ° after the arm was back to the origin and from there move the ball in a straight line again. Instead, the arm moved to the new distance and afterward was rotated to the right position. Although the coordinates were correct, the execution had some room for improvement. This was a computational flaw in the code.

The ODE plots that were tested in this project were difficult to plot properly in the sand. The reason why was because of the G-code that was created in Matlab, the values had a very small difference in the beginning as seen in figure 4.8. Later on, the values increased tremendously with each new step that was taken. The small changes were not possible to draw because the machine itself could not draw that small of a change in the values. The magnetic ball was also too big to show tiny movement, so for this to work a smaller ball is needed. When the ODE increased significantly, the drawing attempted to draw a line because the value did not change that much.

In conclusion, this was a successful project with regard to the drawing capabilities. The art table can draw different patterns as well as anything made with G-code. Although the table can draw almost anything, there are some errors such as the construction of the arm that could not work properly with regards to the rotation. The ODE drawings essentially needed better resolution and more precise movement than what was constructed during the build.

As for the sound levels, it depends on the environment. It was possible to ignore the sound levels if focused on other tasks but for daily use as a table, this might have been inconvenient.

A number of improvements for the future could be to modify the code so that the machine can draw squares, straight lines, and better precision. With regards to the construction, these could also improve by modifying the weight distribution, correct the errors and improve the faulty 3D printed parts.

Bibliography

[1] Siemens PLM Software, 2013, *Solid Edge*. accessed 2021-02-10. https://solidedge.siemens.com/en/

[2] *Electrokit.com*. accessed 2021-02-10. https://www.electrokit.com

[3] Zoe Romano, 2013, *Using Arduino on industrial digital printing machines*, accessed 2021-02-10. https://blog.arduino.cc/2013/07/04/using-arduino-on-industrial-digital-printing-machines/

[4] ARDUINO, 2021, *Arduino*, accessed 2021-02-06. https://www.arduino.cc/

[5] A. C. Brown and D. de Beer, "Development of a stereolithography (STL) slicing and G-code generation algorithm for an entry level 3-D printer," 2013 Africon, 2014. *How G-code works and it is generated.*, accessed 2021-02-06. https://ieeexplore.ieee.org/abstract/document/6757836?casa_token=13WVGxZJEVUAAAAA:U7juTojtluahDY2AcNvhslJQeNCEaOTBSLDsXwSqbKsw3k5UusRO17vZLQyKHQ8oEWevxfn3tw

[6] *Arduino L293D Motor Driver Shield Tutorial*, accessed 2021-02-06. https://create.arduino.cc/projecthub/electropeak/arduino-l293d-motor-driver-shield-tutorial-c1ac9b

[7] *What is a Stepper Motor : Types Its Working*, accessed 2021-02-15. https://www.elprocus.com/stepper-motor-types-advantages-applications/

[8] Robin Mitchell, 2018, *Use this comparison of the UNO, Nano, Mega, and Due Arduino boards to help you choose the best board for your projects.*, accessed 2021-02-10. https://maker.pro/arduino/tutorial/a-comparison-of-popular-arduino-boards

[9] Dejan, *How Servo Motor Works How To Control Servos using Arduino*, accessed 2021-02-15. https://howtomechatronics.com/how-it-works/how-servo-motors-work-how-to-control-servos-using-arduino/

[10] *Rotating parts were bought from Electrokit.com.* accessed 2021-02-10.
https://www.electrokit.com/produkt/kuggremskiva-xl-60t-1-3-80/

[11] *Motors were bought from Electrokit.com.* accessed 2021-02-10.
https://www.electrokit.com/produkt/stegmotor-400-steg-varv-bipolar/

[12] *The patterns were generated with Sandify program*, accessed 2021-04-01.
https://sandify.org/

[13] MathWorks Matlab, 2021, *Matlab.* accessed 2021-02-10.
https://se.mathworks.com/products/matlab.html

[14] *An Introduction to G-Code and CNC Programming*, accessed 2021-03-25.
https://www.thomasnet.com/articles/custom-manufacturing-fabricating/introduction-gcode/

[15] Sandeep, 2019, *How to make Arduino mini CNC plotter machine*, accessed 2021-03-28. https://electricdiylab.com/how-to-make-arduino-mini-cnc-plotter-machine/

[16] *Connection Diagram was generated with Fritzing program*, accessed 2021-04-04.
https://fritzing.org/

[17] *Stepper Motor*, accessed 2021-04-06.
https://www.electronicwings.com/sensors-modules/stepper-motor

[18] University of Cambridge, 2003, *A First Course in the Numerical Analysis of Diferential Equations*, page 4-6, accessed 2021-04-06.
https://bit.ly/31S7R97

[19] *Flowchart designer*, accessed 2021-04-09.
https://app.diagrams.net/

[20] *Processing 3 program*, accessed 2021-03-15.
https://processing.org/

[21] *Aquarium sand from hornbach.se*, accessed 2021-04-15.
https://www.hornbach.se/shop/Akvariesand-5kg-vit/4066916/artikel-detaljer.html

[22] *Chinchilla sand from zoo.se*, accessed 2021-04-15.
https://www.zoo.se/vitapol-chinchilla-sand-badsand.html

[23] Sound Meter app, downloaded 2021-05-05. *App used to measure the sound levels*
https://play.google.com/store/apps/details?id=com.gamebasic.decibelhl=svgl=US

[24] Kalhapure Vrushali Arun, 2015, *Implementation of Carving Machine Controller Based on L293D*, accessed 2021-02-10.
https://www.ijaent.org/wp-content/uploads/papers/v2i3/C0263022315.pdf

[25] D. L. Zhang, X. S. Chen, R. Du, 2013, *A CNC program module based on polar coordinate system*, accessed 2021-02-10.
https://link.springer.com/article/10.1007/s00170-013-4974-1

[26] Kaushik Kumar, Chikesh Ranjan, J. Paulo Davim, 2020, *Polar Coordinates*, accessed 2021-02-11.
https://www.ijaent.org/wp-content/uploads/papers/v2i3/C0263022315.pdf

[27] Wisnu Wijaya et al. 2020, *Two Axis Simple CNC Machines Based on Microcontroller and Motor Driver Shield IC L293D*, accessed 2021-02-11.
https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=arnumber=9310882

Chapter 6

Appendix A

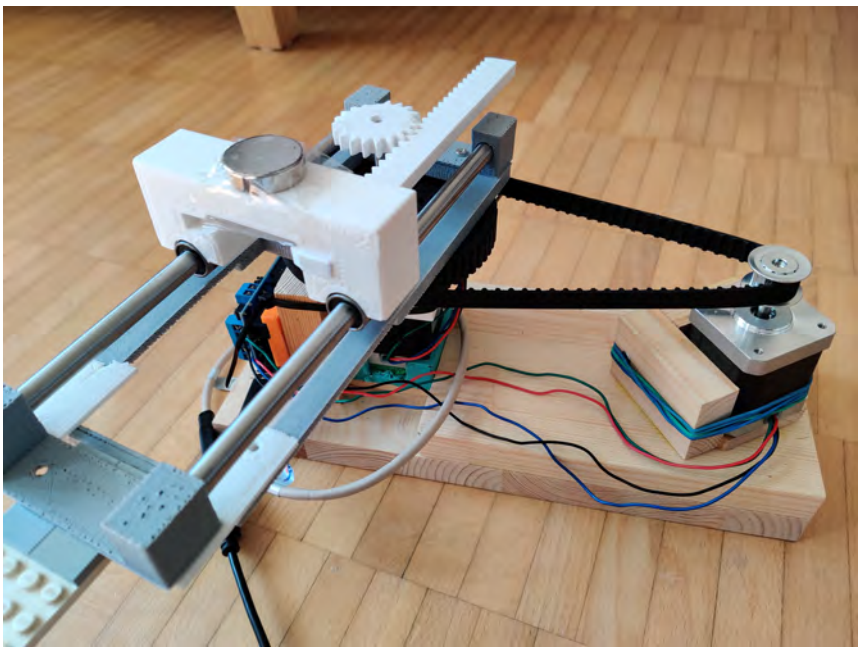


Figure 6.1: Mechanical parts side view. Image taken by Serhat Türk.

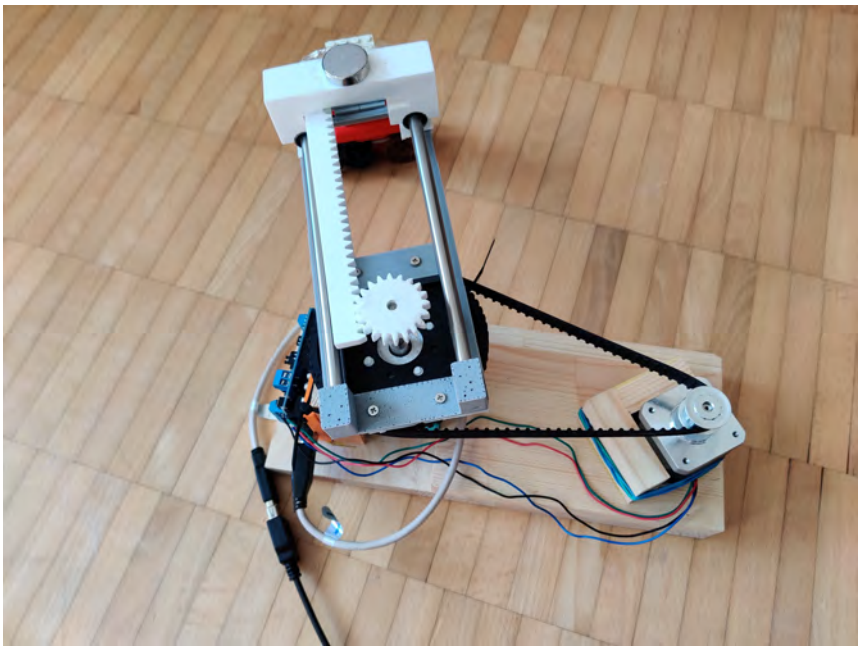


Figure 6.2: Mechanical parts over view. Image taken by Serhat Türk.

Figure 6.3: Table side view. Image taken by Serhat Türk.

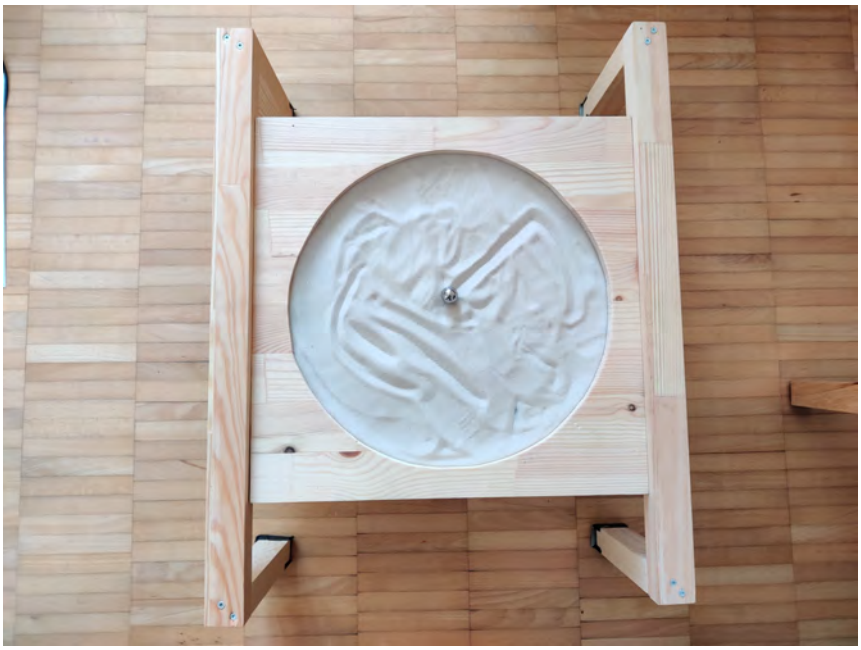


Figure 6.4: Table over view. Image taken by Serhat Türk.

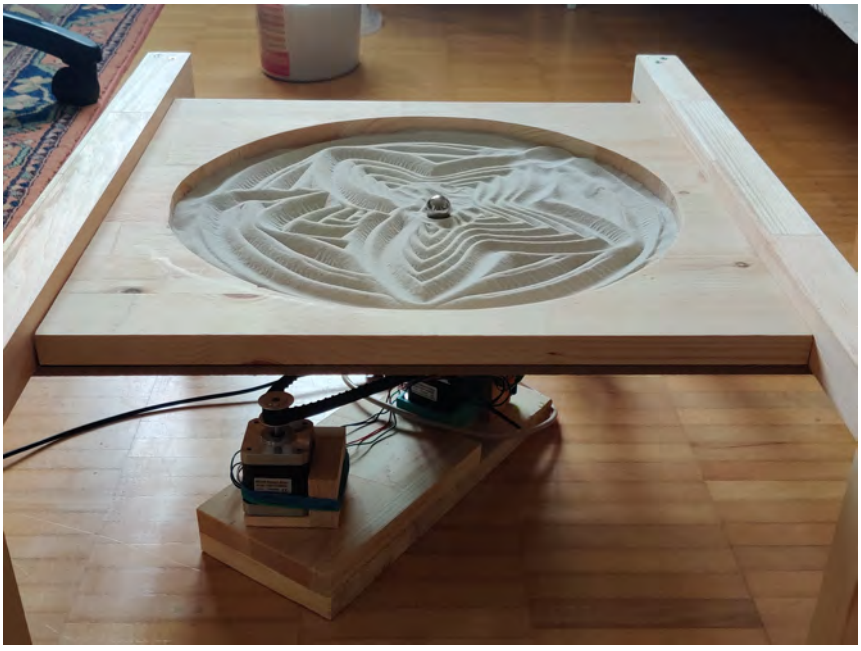


Figure 6.5: Finished construction. Image taken by Serhat Türk.

Chapter 7

Appendix B

Matlab Code

```
% Made by Serhat Turk, Kristoffer Muller.
% 30/04 - 2021
% This code takes a differential equation and solves it with Euler forward
% method. After solving it a G-code file is created with all the points
% stored.
% The file can then easily be uploaded to our GTCRL code
%


clc
clear all

h = 0.2;  % step size
x = (1:h:10);  % the range of x
y = zeros(size(x));  % allocate the result y
y(1) = 0;  % the initial y value
n = numel(y);  % the number of y values

%The loop to solve
for i=1:n-1

    f = 0.1*x(i).^3 +0.5*x(i).^2 +2*x(i)-8 ;  % y' in your DE
    y(i+1) = y(i) + h * f;
end

```

```
26
27 %% Van der Pol oscillator, its possible to get a gcode for
       this aswell,
28 % doesnt work to display on the kinetic art table
29
30 % tspan = [0 20];
31 % y0 = [2; 0];
32 % Mu = 1;
33 % ode = @(t,y) vanderpoldemo(t,y,Mu);
34 % [x,y] = ode45(ode, tspan, y0);
35 % A1 = 155+7.5*[x];
36 % A2 = 155+7.5*[y(:,1)];
37
38
39
40 A1 = 155+[x]; % 155 is our zero position for our maschine
41 A2 = 155+[y]; % 155 is our zero position for our maschine
42
43 %Plot with Matlab
44 plot(A1,A2)
45 title('Solution for f with Euler metod')
46 xlabel('x-axis')
47 ylabel('y-axis')
48 xlim([155 169])
49 ylim([130 650])
50
51 % Convert matlab vector into Gcode format
52
53 fileID = fopen('exp.gcode','w'); %Creates a new file called
       exp in gcode format
54 fprintf(fileID,'; Created with MATLAB\n; Kristoffer och
      Serhat\n; Version: 0.1.2\n;\n; Machine type: Polar\n;\n
      '); % Some information
55 formatSpec = 'G01 X%4.3f Y%4.3f \n'; % defining the format
      of the output data
56 fprintf(fileID,formatSpec,A1,A2); %applies the formatSpec to
       all elements of arrays A1 and A2 (coordinates of the
      Euler metod).
57 fclose(fileID); %Closes the file
```

GCTRL Code

```
/*
*
* This code is by sandeep and it was found on https://electricdiylab.com/how-to-make-arduino-mini-cnc-plotter-machine/
* It reads a text file and sends over any G-code to the Arduino.
*
*/

import java.awt.event.KeyEvent;
import javax.swing.JOptionPane;
import processing.serial.*;

Serial port = null;

// select and modify the appropriate line for your operating system
// leave as null to use interactive port (press 'p' in the program)
//String portname = null;
//String portname = Serial.list()[0]; // Mac OS X
//String portname = "/dev/ttyUSB0"; // Linux

String portname = "COM3"; // Windows port for the arduino

//initla values
boolean streaming = false;
float speed = 0.001;
String[] gcode;
int i = 0;

void openSerialPort(){
  // opens right serial port
  if (portname == null) return;
  if (port != null) port.stop();

  // port that communicates with the arduino
  port = new Serial(this, portname, 9600);

  port.bufferUntil('\n');
```

```
37 }
38
39 void selectSerialPort()
40 {
41   String result = (String) JOptionPane.showInputDialog(frame
       ,
42     "Select the serial port that corresponds to your Arduino
         board.",
43     "Select serial port",
44     JOptionPane.PLAIN_MESSAGE,
45     null,
46     Serial.list(),
47     0);
48     // if there is no port
49   if (result != null) {
50     portname = result;
51     openSerialPort();
52   }
53 }
54
55 void setup()
56 {
57   // sets up program display
58   size(500, 250);
59   openSerialPort();
60 }
61
62 void draw()
63 {
64   // different opptions of actions to choose from when it is
       running.
65   background(0);
66   fill(255);
67   int y = 24, dy = 12;
68   text("INSTRUCTIONS", 12, y); y += dy;
69   text("p: select serial port", 12, y); y += dy;
70   text("arrow keys: jog in x-y plane", 12, y); y += dy;
71   text("5 & 2: jog in z axis", 12, y); y += dy;
72   text("$: display grbl settings", 12, y); y+= dy;
73   text("h: go home", 12, y); y += dy;
74   text("0: zero machine (set home to the current location)",
       12, y); y += dy;
75   text("g: stream a g-code file", 12, y); y += dy;
76   text("x: stop streaming g-code (this is NOT immediate)",
```

```
      12, y); y += dy;
77    y = height - dy;
78    text("current jog speed: " + speed + " inches per step",
      12, y); y -= dy;
79    text("current serial port: " + portname, 12, y); y -= dy;
80  }
81
82  void keyPressed()
83  {
84    // speed change
85    if (key == '1') speed = 0.001;
86    if (key == '2') speed = 0.01;
87    if (key == '3') speed = 0.1;
88    // keypresses with to move the system (we do not use these
        ones)
89    if (!streaming) {
90      if (keyCode == LEFT) port.write("G21/G90/G1 X-10  F3500\
          n");
91      if (keyCode == RIGHT) port.write("G21/G90/G1 X10 F3500\n
          ");
92      if (keyCode == UP) port.write("G21/G90/G1 Y10 F3500\n");
93      if (keyCode == DOWN) port.write("G21/G90/G1 Y-10 F3500\n
          ");
94      if (key == '5') port.write("M300 S50\n");
95      if (key == '2') port.write("M300 S30\n");
96      if (key == 'h') port.write("G90\nG20\nG00 X0.000 Y0.000
          Z0.000\n");
97      if (key == 'v') port.write("$0=75\n$1=74\n$2=75\n");
98      //if (key == 'v') port.write("$0=100\n$1=74\n$2=75\n");
99      if (key == 's') port.write("$3=10\n");
100     if (key == 'e') port.write("$16=1\n");
101     if (key == 'd') port.write("$16=0\n");
102     if (key == '0') openSerialPort();
103     if (key == 'p') selectSerialPort();
104     if (key == '$') port.write("$$\n");
105   }
106   // when pressed g, sends the gcode text file in to the
        arduino serial monitor
107   if (!streaming && key == 'g') {
108     gcode = null; i = 0;
109     File file = null;
110     println("Loading file...");
111     selectInput("Select a file to process:", "fileSelected",
          file);
```

```
112   }
113   // when x is pressed the streaming is canceled. does not
          work right away.
114   if (key == 'x') streaming = false;
115 }
116
117 void fileSelected(File selection) {
118   if (selection == null) {
119     println("Window was closed or the user hit cancel.");
120   } else {
121     println("User selected " + selection.getAbsolutePath());
122     gcode = loadStrings(selection.getAbsolutePath());
123     if (gcode == null) return;
124     streaming = true;
125     stream();
126   }
127 }
128
129 // if gcode is sent, print it in the console.
130 void stream()
131 {
132   if (!streaming) return;
133
134   while (true) {
135     if (i == gcode.length) {
136       streaming = false;
137       return;
138     }
139
140     if (gcode[i].trim().length() == 0) i++;
141     else break;
142   }
143
144   println(gcode[i]);
145   port.write(gcode[i] + '\n');
146   i++;
147 }
148
149 // checks if it is propertly sent to the arduino.
150 void serialEvent(Serial p)
151 {
152   String s = p.readStringUntil('\n');
153   println(s.trim());
154
```

```
155     if (s.trim().startsWith("ok")) stream();
156     if (s.trim().startsWith("error")) stream(); // XXX: really
            ?
157 }
```

Arduino Code

```
/*
 * Serhat Turk, Kristoffer Muller
 * 28/4 - 2021
 *
 * The basics of the code is by sandeep and it was found on:
 * https://electricdiylab.com/how-to-make-arduino-mini-cnc-plotter-machine/
 * It was heavily modified to work with our specific construction.
 * This program reads in G-code and discards all the uncesessary letters and symbols
 * that gets sent from anther program called GCTRL via processing program.
 * These G-code coordinates is read as x and y coordinates but later on converted to
 * rho and theta coordinates which makes the motors move accordingly.
 */

#include <AFMotor.h>
#include <Coordinates.h>
//define coordinates class to calculate polar values.
Coordinates point = Coordinates();

// array size used later.
#define LINE_BUFFER_LENGTH 512

// microstepping for motors.
char STEP = MICROSTEP;

const int stepsPerRevolution = 400;

// Initialize steppers for rho and theta using L293D shield
AF_Stepper motorrho(stepsPerRevolution,2);
AF_Stepper motortheta(stepsPerRevolution,1);

// Structures global variables, these are for coordinates.
struct point {
  float x;
  float y;
};
```

```
36
37 // Current position of magnetic ball
38 struct point actuatorPos;
39
40 //  Drawing settings
41 int StepInc = 1;
42 int StepDelay = 1;
43 int LineDelay = 0;
44 float scale = 155.0;
45 float addtheta = 0.0;
46 float addrho = 0.0;
47 int extrastep = 0;
48
49 // calculated with MICROSTEPS. DIVIDE BY 2 IF YOU FORLOOP
      INSTEAD OF MAKING ALL THE MOVES INSTANTLY
50 float StepsPerMillimeterRho = 275.0/scale; // (max step of
      the arm/max mm that the arm can move freely)
51 float StepsPerRadianTheta = 1206.0/(2.0*PI); // (steps for 1
       full rotation/(2*pi))
52
53 // Drawing robot limits, in mm
54 float rhomin = 0.0;
55 float rhomax = scale;
56 float thetamin = 0.0;
57
58 //start positions (0,0)
59 float rhopos = rhomin;
60 float thetapos = thetamin;
61
62 //  Needs to interpret
63 //  G1 for moving
64 //  Discard any other command!
65 void setup(){
66   //  Setup
67   Serial.begin(9600);
68   delay(100);
69
70   int motorspeed = 10;
71   motorrho.setSpeed(motorspeed);
72   motortheta.setSpeed(motorspeed);
73
74   //  Notification
75   Serial.println("everything is running propertly");
76 }
```

```
77
78 void loop(){
79   delay(100);
80   char line[ LINE_BUFFER_LENGTH ]; // creates an array that
       can store 512 chars
81   char c; // creates check variable.
82   int lineIndex; // creates line index
83   bool lineIsComment, lineSemiColon; //creates bools for
       comments and semicolons for line.
84
85   lineIndex = 0;
86   lineSemiColon = false;
87   lineIsComment = false;
88
89   while (1) {
90     // Serial reception - Mostly from Grbl, added semicolon
         support
91     // This reads and stores Serial input from GCTRL. These
         inputs comes in rows.
92     while ( Serial.available()>0 ){
93       c = Serial.read();
94       if (( c == '\n') || (c == '\r') ){              // End
           of line reached
95         if ( lineIndex > 0 ){                         // Line
             is complete. Then execute!
96           line[ lineIndex ] = '\0';                    //
               Terminate string
97           processIncomingLine( line, lineIndex );
98           lineIndex = 0;
99         }
100        else{
101          // Empty or comment line. Skip block.
102        }
103        lineIsComment = false;
104        lineSemiColon = false;
105        Serial.println("ok");
106      }
107      else{
108        if ( (lineIsComment) || (lineSemiColon) ){   //
             ignore all comment characters
109          if ( c == ')' )  lineIsComment = false;     // if
               end of comment is reach resume line.
110        }
111        else{
```

```
112             if ( c <= ' ' ){ // delete empty space.
113             }
114             else if ( c == '/' ){ // Block delete not
                    supported. Ignore character.
115             }
116             else if ( c == '(' ){  // Enable comments flag and
                    ignore all characters until ')' or EOL.
117               lineIsComment = true;
118             }
119             else if ( c == ';' ){
120               lineSemiColon = true;
121             }
122             else if ( lineIndex >= LINE_BUFFER_LENGTH-1 ){
123               Serial.println( "ERROR - lineBuffer overflow" );
124               lineIsComment = false;
125               lineSemiColon = false;
126             }
127             // storing values if the letters in line
128             else if ( c >= 'a' && c <= 'z' ){ // Upcase
                    lowercase
129               line[ lineIndex++ ] = c-'a'+'A';
130             }
131             else{
132               line[ lineIndex++ ] = c;
133             }
134           }
135         }
136       }
137     }
138 }
139
140 void processIncomingLine( char* line, int charNB ){
141   int currentIndex = 0;
142   char buffer[ 64 ];  // 64 for 1 parameter. the buffer to
        store the bytes in
143   struct point newPos;
144
145   newPos.x = 0.0;
146   newPos.y = 0.0;
147
148   //  Needs to interpret
149   //  G1 for moving
150   //  G1 X60 Y30
151   //  G1 X30 Y50
```

```
152   //  Discard any other command!
153   while( currentIndex < charNB ){
154     switch ( line[ currentIndex++ ] ){                //
          Select command, if any
155     case 'U':
156       break;
157     case 'D':
158       break;
159     case 'G':
160       buffer[0] = line[ currentIndex++ ];           // /!\
            Dirty - Only works with 2 digit commands
161       //      buffer[1] = line[ currentIndex++ ];
162       //      buffer[2] = '\0';
163       buffer[1] = '\0';
164
165       switch ( atoi( buffer ) ){                  // Select G
            command // atoi takes a str and converts it to int
166       case 0:                                     // G00 & G01
            - Movement or fast movement. Same here
167       case 1:
168         // /!\ Dirty - Suppose that X is before Y
169         // Get X/Y position in the string (if any)
170         char* indexX = strchr( line+currentIndex, 'X' );
171         char* indexY = strchr( line+currentIndex, 'Y' );
172         // compares positons indexes for the ball and sets
              new x1, y1 coordinates
173         if ( indexY <= 0 ){
174           newPos.x = atof( indexX + 1);
175           newPos.y = actuatorPos.y;
176         }
177         else if ( indexX <= 0 ){
178           newPos.y = atof( indexY + 1);
179           newPos.x = actuatorPos.x;
180         }
181         else{
182           newPos.y = atof( indexY + 1);
183           *indexY = '\0';
184           newPos.x = atof( indexX + 1);
185         }
186         // starts drawing from new coordinates.
187         drawLine(newPos.x, newPos.y );
188         // Serial.println("ok");
189         actuatorPos.x = newPos.x;
190         actuatorPos.y = newPos.y;
```

```
191           break;
192         }
193         break;
194         // stores new buffer values from line array.
195       case 'M':
196         buffer[0] = line[ currentIndex++ ];        // /!\
              Dirty - Only works with 3 digit commands
197         buffer[1] = line[ currentIndex++ ];
198         buffer[2] = line[ currentIndex++ ];
199         buffer[3] = '\0';
200         switch ( atoi( buffer ) ){
201         case 300:
202           {
203             char* indexS = strchr( line+currentIndex, 'S' );
204             float Spos = atof( indexS + 1);
205             //          Serial.println("ok");
206             break;
207           }
208         case 114: // M114 - Repports position
209           Serial.print( "Absolute position : X = " );
210           Serial.print( actuatorPos.x );
211           Serial.print( "  -  Y = " );
212           Serial.println( actuatorPos.y );
213           break;
214         default:
215           Serial.print( "Command not recognized : M");
216           Serial.println( buffer );
217         }
218       }
219     }
220 }
221
222 // initiall values
223 float dtheta_norm_sum = 0;
224 float drho_norm_sum = 0;
225
226 /*********************************
227  * Draw a line from (x0;y0) to (x1;y1).
228  * int (x1;y1) : Starting coordinates
229  * int (x2;y2) : Ending coordinates
230  *********************************/
231 void drawLine(float x1, float y1) {
232
233   // start values
```

```
234   float rho0 = rhopos;
235   float theta0 = thetapos;
236
237   // calculatin the polar coordinates to y1 and x1.
238   // !!! The coordinates are scaled with Sandify so it
         starts at (155,155) as the origin.
239   // !!! For anything else change the scale to a number
         instead or remove it for (0,0).
240   point.fromCartesian(x1-scale,y1-scale);
241   float rho1 = point.getR();
242   float theta1 = point.getAngle();
243
244   // sets max drawing distance rho
245   if (rho1 <= rhomin){
246     rho1 = rhomin;
247   }
248   if (rho1 >= rhomax){
249     rho1 = rhomax;
250   }
251
252   // rho and theta difference.
253   float drho = abs(rho1-rho0);
254   float dtheta = abs(theta1-theta0);
255
256   // deciding which directions the motors are supposed to
         spin
257   int srho = rho1>rho0 ? StepInc : -StepInc;
258   int stheta = theta1>theta0 ? StepInc : -StepInc;
259
260   // takes care of angle movement from quadrant 1 to 4 and
         vice versa.
261   // makes it move in the right direction instead of spinnig
          opposite directions.
262
263   if (theta1 >= 3.0*PI/2.0 && theta0 <= PI/2.0){ // remove
         2*Pi if it goes from quadrant 1 -> 4
264     dtheta = -2.0*PI + theta1 - theta0;
265     stheta = -1;
266     Serial.println("+ 2*pi");
267   }
268   if (theta0 >= 3.0*PI/2.0 && theta1 <= PI/2.0){ // adds 2*
         Pi if it goes from quadrant 4 -> 1
269     Serial.println("- 2*pi");
270     dtheta = -2.0*PI - theta1 + theta0;
```

```
271       stheta = 1;
272     }
273     dtheta = abs(dtheta);
274
275     // calulates amount of motorsteps to move each motor.
276     float drho_norm = round(drho*StepsPerMillimeterRho);
277     float dtheta_norm = round(dtheta*StepsPerRadianTheta);
278
279     // Take care of rounding errors for the motor steps.
280     addtheta += (dtheta*StepsPerRadianTheta - dtheta_norm);
281     addrho += (drho*StepsPerMillimeterRho - drho_norm);
282     if (addtheta >=1){
283       ++dtheta_norm;
284       --addtheta;
285     }
286     if(addtheta <= -1){
287       --dtheta_norm;
288       ++addtheta;
289     }
290     if (addrho >=1){
291       ++drho_norm;
292       --addrho;
293     }
294     if(addrho <= -1){
295       --drho_norm;
296       ++addrho;
297     }
298     // bunch of prints for the values, this used in error
          testing.
299     /*
300     Serial.print("rho0: ");
301     Serial.println(rho0);
302     Serial.print("rho1: ");
303     Serial.println(rho1);
304     Serial.print("drho: ");
305     Serial.println(drho);
306     Serial.print("drho_norm: ");
307     Serial.println(drho_norm);
308
309     Serial.print("theta0: ");
310     Serial.println(theta0);
311     Serial.print("theta1: ");
312     Serial.println(theta1);
313     Serial.print("dtheta: ");
```

```
314     Serial.println(dtheta);
315     Serial.print("dtheta_norm: ");
316     Serial.println(dtheta_norm);
317     Serial.print("stheta = ");
318     Serial.println(stheta);
319     /*
320     dtheta_norm_sum += dtheta_norm;
321     Serial.print("dtheta_norm_sum: ");
322     Serial.println(dtheta_norm_sum);
323     /*
324     drho_norm_sum += drho_norm;
325     Serial.print("drho_norm_sum: ");
326     Serial.println(drho_norm_sum);
327     */
328
329     float over = 0.0;
330
331     // Moving motors so that they move together and ends at
          the same time.
332     if (drho_norm > dtheta_norm){
333       float divider = dtheta_norm/drho_norm;
334       for (int i=0; i<drho_norm; ++i) { // loop to make right
            amount of steps.
335         moverho(srho);  //moves rho motors
336         //Serial.println("rho");
337         over += divider;
338         // makes theta motor move so the drawing becomes
              correct.
339         if (over >= 1) {
340           movetheta(stheta);
341           --over;
342           //Serial.println("theta");
343         }
344         //delay(StepDelay);
345       }
346     }
347     else if(dtheta_norm > drho_norm){ // if motor theta moves
          more steps than motorrho
348       for (int i=0; i<dtheta_norm; ++i) {
349         float divider = drho_norm/dtheta_norm;
350         movetheta(stheta);
351         over += divider;
352         if (over >= 1) {
353           moverho(srho);
```

```
354           --over;
355           //Serial.println("rho");
356         }
357         //delay(StepDelay);
358       }
359     }
360     // saves new values to the old ones.
361     rhopos = rho1;
362     thetapos = theta1;
363     // turns of the motors.
364     motorrho.release();
365     motortheta.release();
366   }
367
368   // to move motor rho
369   void moverho (int s){
370     if (s == -1){
371       //BACKWARD
372       motorrho.step(1, BACKWARD, STEP);
373     }
374     else{
375       //FORWARD
376       motorrho.step(1, FORWARD, STEP);
377     }
378     return;
379   }
380
381   // move motor theta and rho so that the arm doesnt move when
        it rotates.
382   void movetheta(int s){
383     int stepamount = 1;
384     int extra = 6;
385     if (s == -1){
386       //BACKWARD
387       motortheta.step(stepamount, BACKWARD, STEP);
388       --extrastep;
389       if (extrastep == -extra){
390         moverho(s);
391         extrastep = 0;
392       }
393     }
394     else{
395       //FORWARD
396       motortheta.step(stepamount, FORWARD, STEP);
```

```
397       ++extrastep;
398       if (extrastep == extra){
399         moverho(s);
400         extrastep = 0;
401       }
402     }
403     return;
404   }
```

Acumen Code for simulation

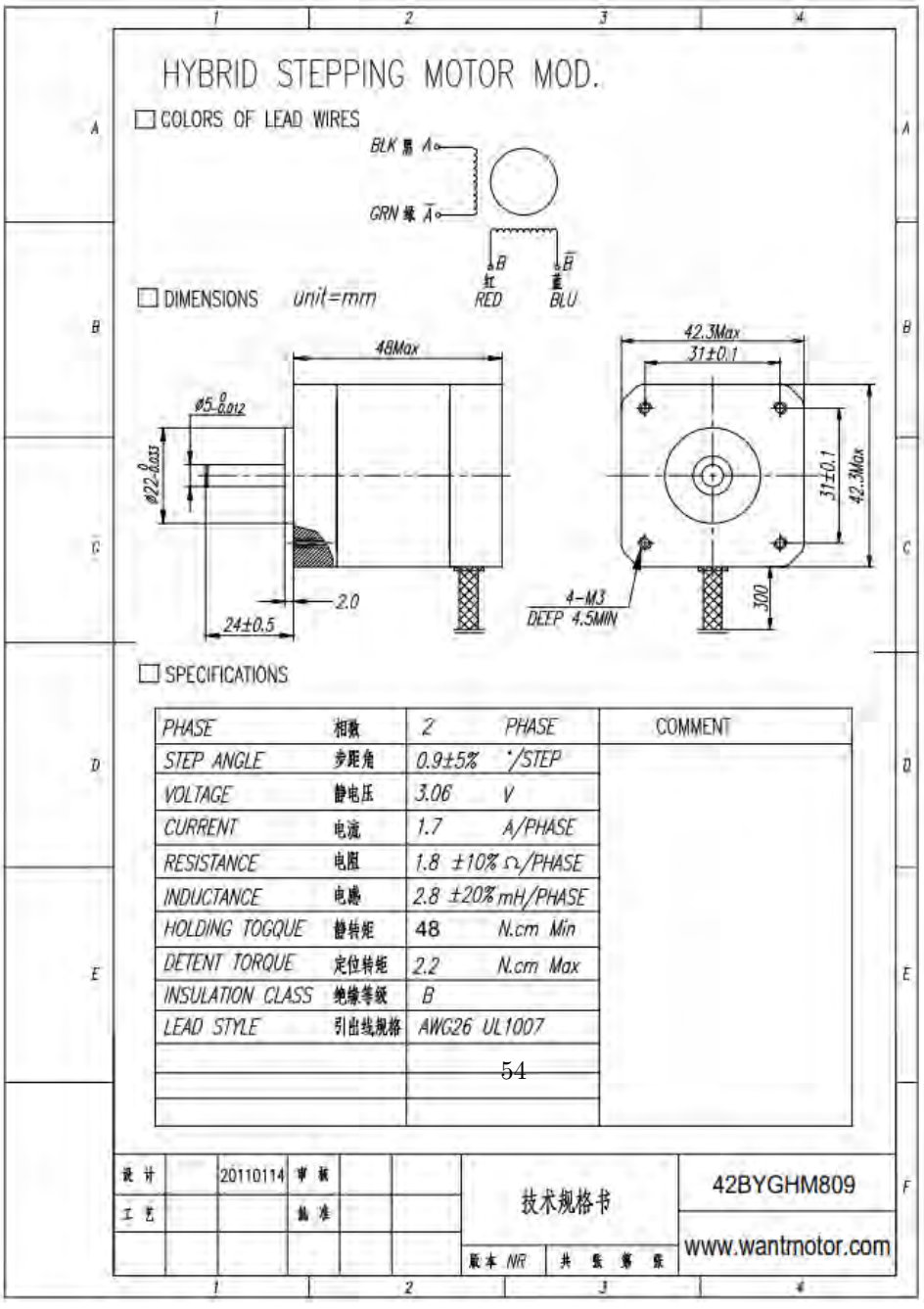
```
// Made by Serhat Turk, Kristoffer Muller.
// 23/3 - 2021
//
// This is a simulation on how our construction will move as it is operating.
// It is essentially an arm moving in and out and a rotating disk.
//
//


model Main(simulator) =
initially
// creates a c1 is red cirkular base, c2 is the green arm
c1 = create Arm((0,0,0),(0,0,0)), // input for postion och rotation
c2 = create Platta((0,0,0),(0,0,0)), // input for postion och rotation

//start data, everything is set to 0 in the beginning
x1=0, x1'=0, x1''=0, // distance, velocity, acceleration for x1 used for angles later
x2=0, x2'=0, x2''=0, // distance, velocity, acceleration for x2 used for angles later
// v = 0, v' = 0, v'' = 0,
rho = 0.35, // friction coefficiency for normal sand
m = 10, // mass och the system
//g = 9.82 // acceleration konstant (not used in this case)

// always loop that starts and ends the simulation.
always
if x1<15
then x1'' = -(x1'-0.3) // sets an angular acceleration for x1
else if x1'>0
then x1'' = -0.3 // slows down the system if this goes through
else x1'' = 0, // continues the animation.
x2'' = -100*(x2-x1)-10*(x2'-x1'), // making the green arm move propertly with the red circle
c2.rot = (0,x2,0), // maknig the red circle rotate with and
```

```
    acceleration and retardation
// changes the position and rotation of the green arm to
    match the rotation of the red circle
c1.pos = (-x2/2*cos(x1),-0.5,x2/2*sin(x1)),
c1.rot = (0,x1,0)

// start model which is used to develope the green arm
model Arm(pos,rot) =
initially
_3D = (),_Plot=()
always
_3D = (Box
center = pos + (0,0,0)
color = green
size = (4,0.5,1)
rotation = rot
)

// start model which is used to develope the red circle base
model Platta(pos,rot) =
initially
_3D = (),_Plot=()
always
_3D =(Cylinder
center=(0,0,0) + pos
size = (0.5,2)
color = red
rotation = rot + (0,0,0)
)
```

Chapter 8

Appendix C

8.1 Stepper motor datasheet



8.2 l293d motor shield datasheet

Rajguru Electronics www.rajguruelectronics.com

L293D Based Arduino Motor Shield

Features:

- 2 connections for 5V 'hobby' servos connected to the Arduino's high-resolution dedicated timer - no jitter!
- Up to 4 bi-directional DC motors with individual 8-bit speed selection (so, about 0.5% resolution)
- Up to 2 stepper motors (unipolar or bipolar) with single coil, double coil, interleaved or micro-stepping.
- 4 H-Bridges: L293D chipset provides 0.6A per bridge (1.2A peak) with thermal shutdown protection, 4.5V to12V • Pull down resistors keep motors disabled during power-up
- Big terminal block connectors to easily hook up wires (10-22AWG) and power
- Arduino reset button brought up top
- 2-pin terminal block to connect external power, for separate logic/motor supplies
- Tested compatible with Mega, UNO & Duemilanove
- Dimensions: 69mm x 53mm x 14.3mm (2.7in x 2.1in x 0.6in)

The L293D is a dedicated module to fit in Arduino UNO R3 Board, and Arduino MEGA. It is actually a motor driver shield that has full featured Arduino Shield can be used to drive 2 to 6 DC motor and 4 wire Stepper motor and it has 2 set of pins to drive a SERVO.

8.2. L293D MOTOR SHIELD DATASHEET

L203D is a monolithic integrated that has a feature to adopt high voltage, high current at four channel motor driver designed to accept load such as relays solenoids, DC Motors and Stepper Motors and switching power transistor. To simplify to used as two bridges on each pair of channels and equipped with an enable input. A separate supply input is provided for the logic, allowing operation at a lower voltage and internal clamp diodes are included.

The device is suitable for use in switching applications at frequencies up to 5kHz. The L293D is assembled in a 16 lead plastic package which has 4 centre pins connected together and used for heat sinking. The L293D is assembled in a 20 lead surface mount which has 8 centre pins connected together and used for heat shrinking.

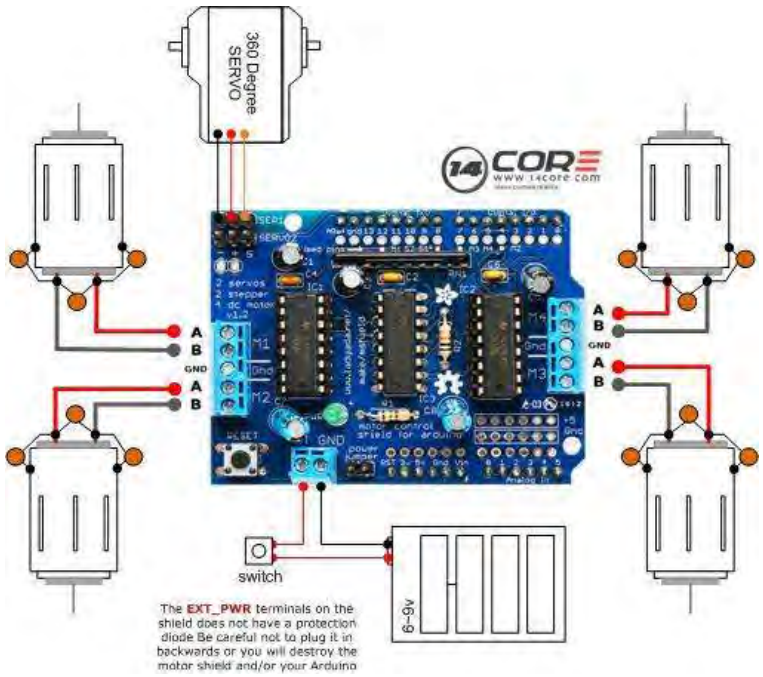
Items	Min	Typical	Max	Unit
Control Voltage	4.5	5	5.5	V
Driver Voltage	6	9	15	V
Output Current			1.2	A
Dimensions				cm
Weight				gm

Control up to 4 DC motors.

- Control 2 Servos.
- Logic Control Voltage VSS: 4.5 ~ 5.5 V
- Motor Supply Voltage VSS: 15v
- Drive operating current IO: 1.2A
- 8 Stage Serial Shift Registers

Wiring a DC Motor

Rajguru Electronics www.rajguruelectronics.com



Motor requires more energy specially cheap motors since chip motors less efficient. The important thing you need is find out what voltage require your going to use. some small motors are only intended to run at 1.5 volts but it is just a common to have 6 ~ 12v motors. The motor controller on L294D shield is design to run at 4.5v to 25v. most 1.5 ~ 3 volts motor will not works on this shield. another thing you need is to figure it out how much current the motor will support? The L293D chip support up to 600 mA per motor, with 1.2A peak current. Note ones you head towards 1A you'll probably want to put a heat sink on the chip, otherwise it will get thermal failure or burning out the chip.

Note:

Do not connect the motor to 5v line on the board. There are two places you can get your motor high voltage supply.

Wiring and Installation the DC Motor to the l293D Shield

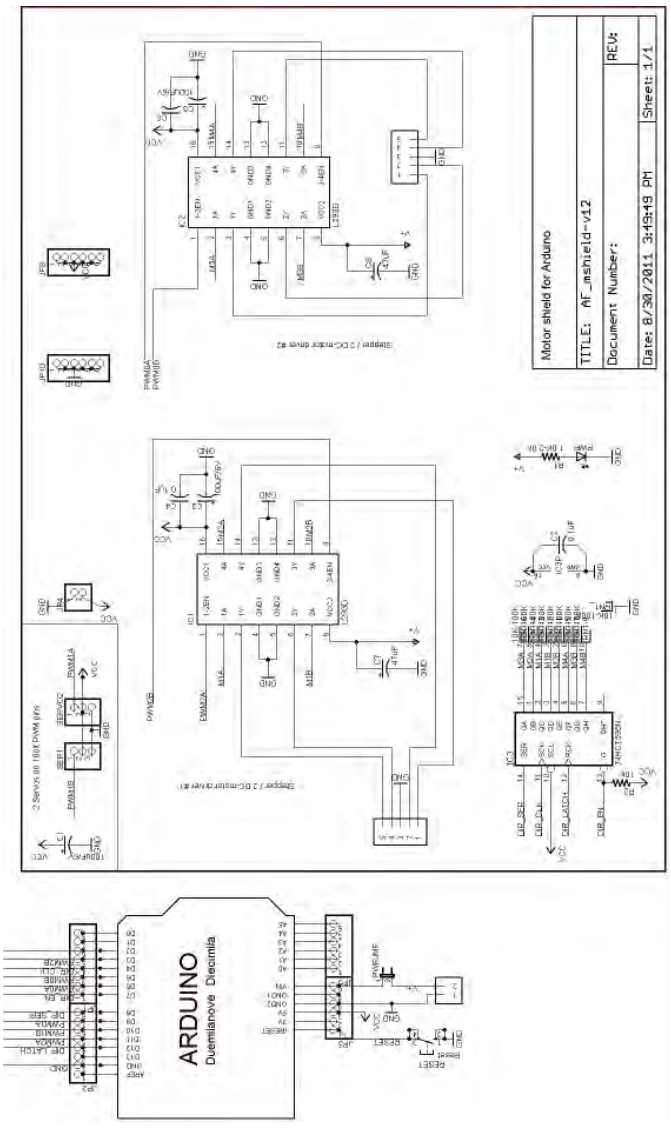
The DC motor are used for all sort of robotics projects. The motor shield can drive up to 4 or 6 DC motors bi directional, it means that they can be driven forward and backward. The speed can also be varied at 0.5% increments using PWM(Pulse with Modulation) this means that speed can be controlled.

Note:

The H-Bridge Chip is not supported for driving load over 0.6A over 1.2A so this it means that this chip is for small motors. Check the datasheet below to learn more. To connect simply place the 2 wires to the terminal with screw and then connect them to either M1, M2, M3, or M4 follow the example diagram above.

Rajguru Electronics www.rajguruelectronics.com

Schematic :



TRITA-ITM-EX 2021:19