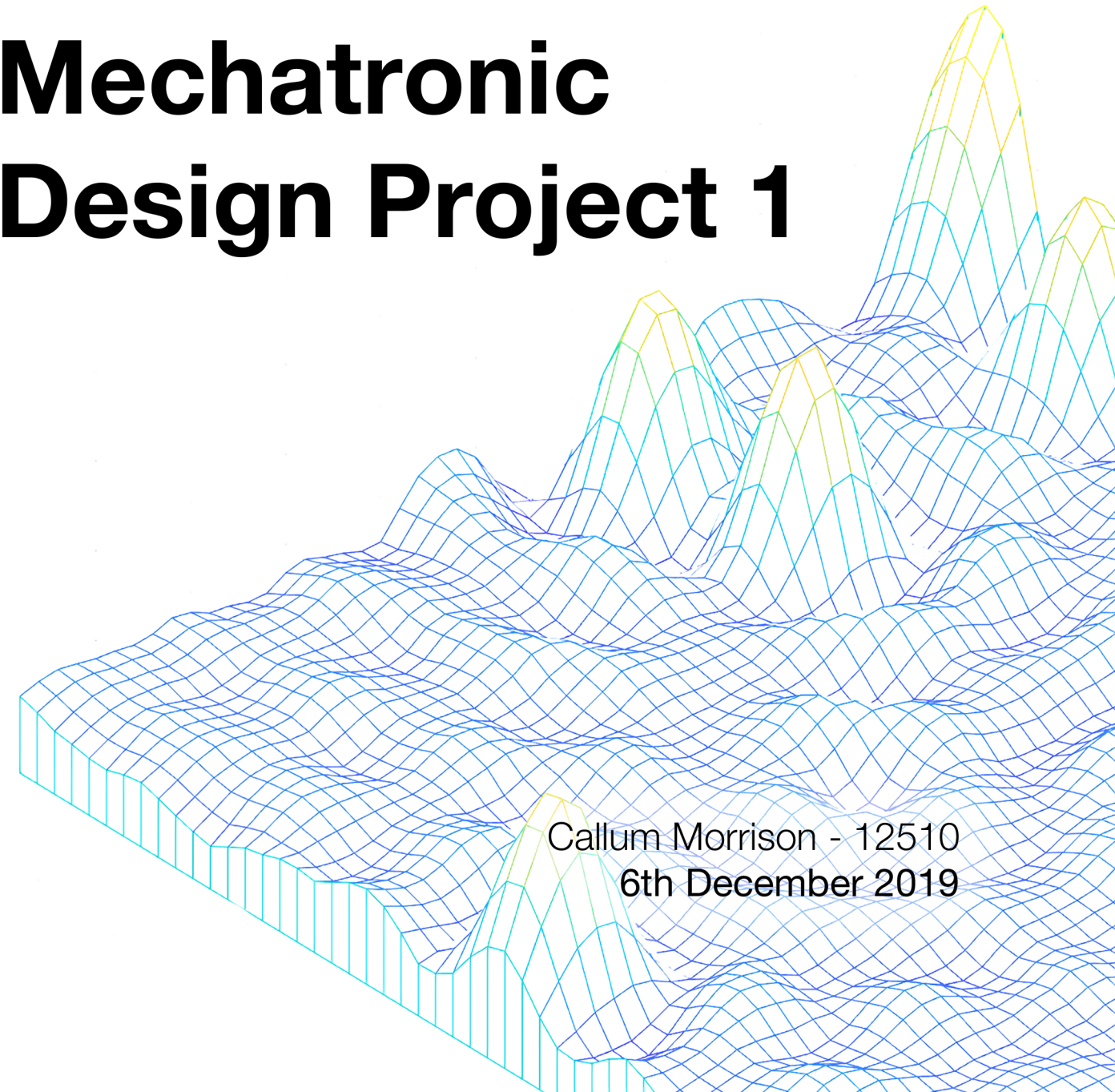
Faculty of
Engineering &
Design

UNIVERSITY OF
BATH

Mechatronic Design Project 1



Callum Morrison - 12510

6th December 2019

Summary

Magnetic treasure is found through the use of a two axis gantry, controlled via MATLAB code interfacing through an Arduino compatible development board.

Gantry movement is achieved through the use of a function intended to drive piezo buzzers, which functions as the pulse train required by the gantry stepper motors. Ramping is used to allow the gantry to run at higher speeds than the stepper motors self-start frequency. A closed loop positional feedback system allows for corrections to position to be made, ensuring gantry movement is accurate and reliable. A supporting stripboard allows the stepper motor to be switched between a single Arduino ramp pin, and a tied low (no pulse) input. The board is adapted from a previous design iteration incorporating a 555 astable oscillator circuit.

Scanning of the board for treasure is done through three hall effect sensors, which after a scanning sweep, produce a matrix of sensor values corresponding to the magnetic field strength at that board position. This data is processed in various ways, tested through experimentation with randomly produced sample scan data. An S curve filter allows removal of most noise, and boosting of weaker magnet signals. Wiener filtering removes any remaining noise. A contour curve traces signal strength boundaries, resulting in polygon shapes surrounding the strongest signal points (magnet centres). The centre points of these polygon shapes are fed into a k-means algorithm, which is repeated with target number of clusters from one to twelve; the maximum number of magnets. The outcomes are evaluated using Davies–Bouldin indexing, to determine the number of magnets present on the board, and their positions.

Table of Contents

1.0 Introduction

The *Pirates Sans Frontiers ARGGG!* have assigned a treasure-finding task. Magnetic treasure randomly placed within a defined area must be located using a two axis gantry, each axis controlled by a stepper motor. The task can be split into three main subsections; movement of the gantry; scanning for treasure and data processing; and marking of the treasure. This report focuses on the first two areas.

A number of technical requirements must be adhered to *(Georgilas, 2019)*, including the use of an Arduino Mega2560 compatible development board (referred to as *Arduino*), to interface with the gantry and other components via the *MATLAB Support Package for Arduino Hardware*. The program developed must be fully autonomous once started.

2.0 Gantry Movement

2.1 Pulse Generator

As time taken to complete the given task was marked, with shortest times awarding higher marks, the speed of the gantry should be maximised.

The gantry stepper motors complete one step when an input pulse is provided. Writing an Arduino pin low, then high, then low will count as one pulse. Creating a loop will result in an output pulse train, shown in *Figure 2.1.1*. However, the speed at which instructions are sent to the Arduino is limited by the USB / serial connection. *Equation 2.1.1* calculates the maximum allowable pulse frequency given a delay of 30ms.

```
while true
      writeDigitalPin(a, pulsePin, 0)
      writeDigitalPin(a, pulsePin, 1)
end
```

Figure 2.1.1a: *MATLAB code to produce pulse pulse train on* `pulsePin`.

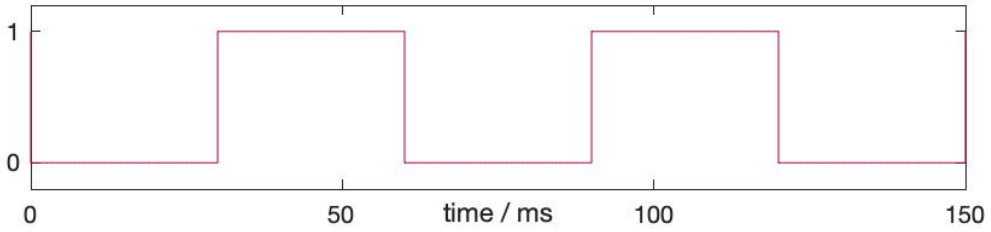


Figure 2.1.1b: *Output reading from* `pulsePin`.

$$30ms\ per\ state\ change\ \therefore\ 60ms\ per\ pulse$$
$$\frac{1\ pulse}{60ms} = 16.\dot{6}Hz$$

Equation 2.1.1: *Maximum output frequency of pulse train by setting pin high and low recursively, as a result of command delay between MATLAB and Arduino hardware.*

This resulted in incredibly slow movement of the gantry, and so was disregarded as a potential solution, and it was determined that pin control must be done by the Arduino alone, to remove the USB connection delay. Pulse Width Modulation (PWM) can be used to control DC motor speed, and perceived brightness of LEDs. It does this by switching the output signal between high and low at a frequency high enough that the motor's inertia limits jerkiness, and our eyes cannot perceive the flickering due to persistence of vision *(Wikipedia, 2019)*. The output is a square wave, which works as a pulse train input to the gantry stepper motors. The Arduino has dedicated functions to output PWM, the results shown in *Figure 2.1.2*.

```
writePWMDutyCycle(a, pulsePin, 0.5);
```

Figure 2.1.2a: *MATLAB code to enable PWM on* `pulsePin`. The `0.5` determines a duty cycle of 50%; or equal time for the output on and off.

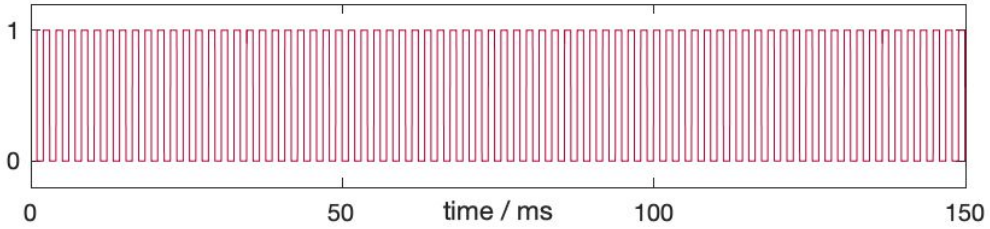


Figure 2.1.2b: *Output reading from PWM at 50% duty cycle.*

With this function, the frequency is fixed at 980Hz *(Shirriff, 2009)* and cannot be altered (technically it can, however this is unfeasible for this project; *eTechnophiles, 2017*). This is acceptable, however the gantry was capable of much higher speeds.

Another function which can output a square wave is a function called `playTone`. Designed to play a musical note on a piezo buzzer, it also produces a square wave output, and importantly the output frequency can be varied through input variables. The results are shown in *Figure 2.1.3*.

```
playTone(a, pulsePin, 200, 0.05)
playTone(a, pulsePin, 500, 0.05)
playTone(a, pulsePin, 2000, 0.05)
```

Figure 2.1.3a: *MATLAB code to output a square wave of 200Hz, 500Hz, and 2000Hz for 50ms each*.

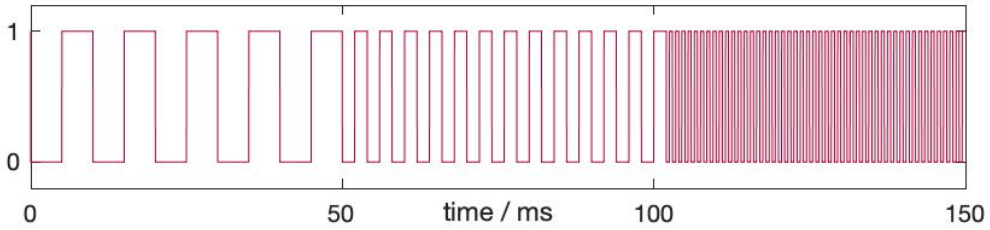


Figure 2.1.3b: *Output reading from* `playTone` *at different frequencies.*

By increasing the frequency in small increments, the speed of the stepper motor could be 'ramped up', to reach higher speeds while avoiding stall *(Craig, 2017)*. The major limitation with this method is that the function cannot be used on two pins simultaneously.

2.2 Initial Logic Idea

The initial motion concept is outlined simply in *Figure 2.2.1*. It employs the one dedicated `playTone` pin only for ramp up and ramp down. When an axis is at full speed, the input is switched to pulses generated by an astable 555 oscillator *(Eater, 2016)*.

```
find longest distance; assign to L
find shortest distance; assign to S

ramp up L
once at full speed, switch L input to 555 circuit
ramp up S

if either axis within ramp distance of destination
      ramp down axis
```

Figure 2.2.1: *Pseudocode overview of initial logic idea.*

It was quickly realised that this concept was much more complex than initially thought. A number of additional conditions which would require consideration are shown in *Table 2.2.1*.

Table 2.2.1: *Some of the required and optional considerations when implementing the initial logic solution.*

Required / Optional	Condition
Required	Both axis shorter than ramp distance
Required	Both axis within ramp distance of each other
Required	Frequency / time is linear, frequency / steps is not
Optional	Both axes ramp up together
Optional	Box axes ramp down together
Optional	Correction made while both axis at full speed
Required	Corrections made after both ramp down
Required	Discrete time steps, or function interpolation
Optional	Running of shorter axis at less than full speed

Some of these problems were incredibly complex, and would require a large number of conditions and extensive testing to limit failure cases. Furthermore, after testing various gantry

speeds, the solution was determined to have minimal impact on time, even if implemented perfectly.

2.3 Revised Logic Idea

The 555 timer circuit is no longer used. Movement when distance x ≠ distance y can be broken into two sections; the first travels to the shortest distance first, and the second completes the longest distance. The concept is outlined in *Figure 2.3.1*.

```
find longest distance; assign to L
find shortest distance; assign to S

ramp up both axes simultaneously

if S axis within ramp distance of destination
      ramp down both axes

disable S

ramp up L

if L axis within ramp distance of destination
      ramp down axis

make corrections if required
```

Figure 2.3.1: *Pseudocode overview of revised logic idea.*

This concept is substantially simpler to implement than the original idea, with very little time impact. The concept is implemented in `goToPos`, shown in *Appendix B*.

A possible consideration was that cups already placed must not be knocked in the process of placing new cups. It would likely be possible to implement an iterative algorithm to ensure that placed cups would be avoided (check if the calculated path will pass within the radius of a cup, and tweak the traversal accordingly). However, it was determined that there would be some cases it would be impossible to avoid placed cups (trapping the gantry in one corner), and that a simpler solution would be through mechanical means.

2.4 Circuit Design

The electronic circuit was designed with the original concept in mind, and later adapted to suit the redesigned logic. A schematic was designed to allow an output pin from the Arduino to toggle between two inputs; initially the 555 oscillator circuit, and the `playTone` pin. A schematic for one axis is shown in *Figure 2.4.1*, which is duplicated for the other axis.

The design requires a total of two NOT gates, four AND gates, and two OR gates. An initial breadboard design used only NAND gates; a universal gate *(Bouhraoua, 2005)*. A full design

would require 16 NAND gates. Using 7400 series ICs, four NAND chips would be required, versus three total for the other design, which was resultantly chosen.

It was originally intended to create a 'shield' for the Arduino, minimising the requirement for cables and negating the chance of connecting the board incorrectly after the initial setup. However, during initial assembly it was realised that due to the intricacy of this design, the probability of electrical issues was high, and debugging could easily take more time than was available. Therefore as a backup, an equivalent circuit was designed with a larger footprint; shown in *Appendix A*. 'Holes' were used to mark where tracks should be broken, to help avoid errors during manufacture. After testing, a decoupling capacitor was added to the 555 timer circuit.

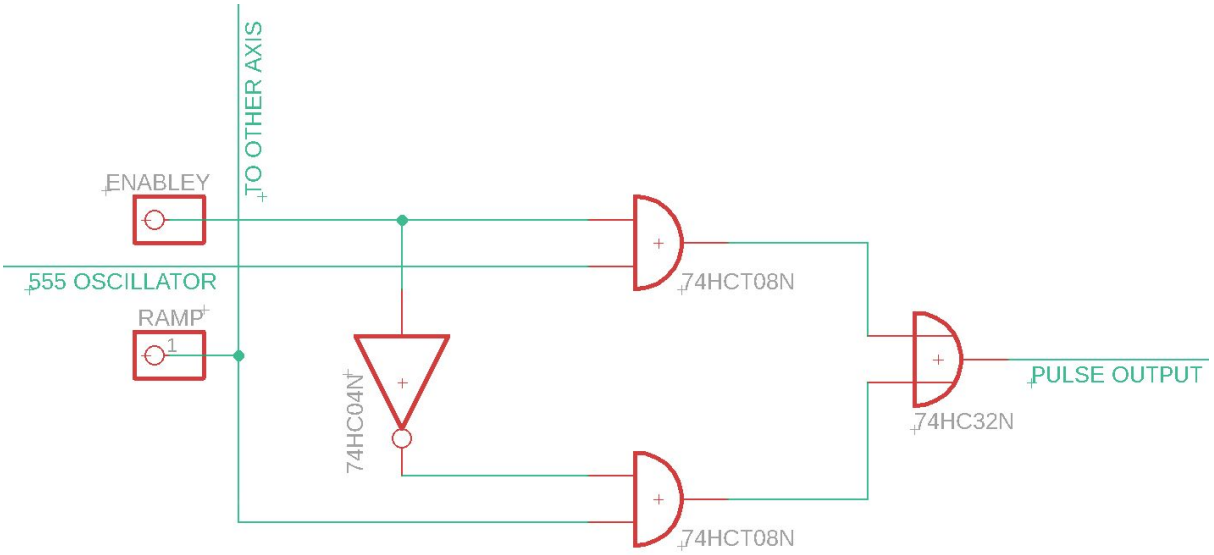


Figure 2.4.1: *Schematic for pulse train switch circuit for one gantry axis. The ramp pin and 555 oscillator input is shared between both axes, but they can be switched independently.*

To implement the redesigned logic, the 555 timer chip was removed from the board, and the switch circuit toggled to a tied low signal, instead of the original oscillator circuit. With this use case, the switching circuit could be simplified to a single AND gate, reducing resource requirements, board size, and complexity.

2.5 Positional Feedback

The number of pulses is proportional to the distance traveled by the gantry. Using a steady, finite `playTone`, the number of pulses can be calculated accurately as shown in *Figure 2.5.1*.

```
pulseFrq    = 100; % in Hz
pulseTime   = 2;   % in seconds
playTone(a, pulsePin, pulseFrq, pulseTime)
nPulses = pulseFrq * pulseTime; % number of pulses sent
```

Figure 2.5.1: *Function to calculate the number of pulses sent from the Arduino using the `playTone` function.*

Limitations arise when the previous `playTone` is not finished before a new one 'overwrites' it. Furthermore, the original design employing the 555 timer circuit would be inaccurate to estimate. A solution involves counting the pulses sent to the gantry; assuming no steps are skipped mechanically, this allows for the highest positional accuracy.

Counting steps by repeat readings of an Arduino digital pin is impossible, as the polling rate with MATLAB delay is substantially slower than the pulse state change frequency. A library allows code to run on the Arduino untethered from a MATLAB instance. Although creation of a custom library was investigated, it was determined too complex within the given time constraints. A quadrature rotary encoder library is available, and so this was adapted to use.

Rotary encoders are used for positional feedback of shafts. Quadrature encoders also determine direction, through use of two phase offset pulse trains *(Georgilas, 2019)*. Testing determined that the rotary encoder library would not count steps from a single pulse train. A second pulse train with a phase offset needed to be generated to 'trick' the library into counting the steps from one pulse train. This was achieved through an edge triggered pulse generator *(Jensen, 2016)*. A comparison between an expected input and the resultant input to the Arduino is shown in *Figure 2.5.2*.

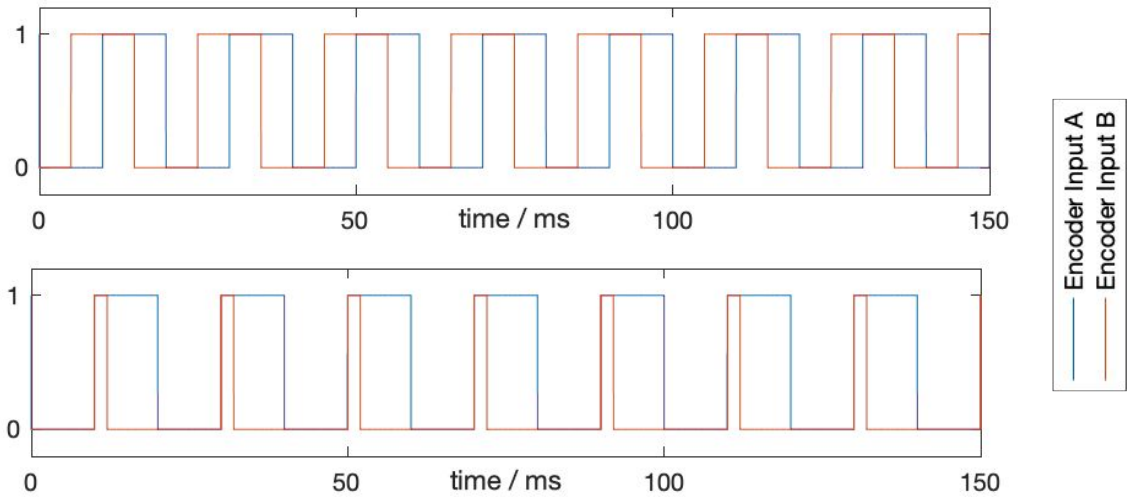


Figure 2.5.2: *Comparison between a standard quadrature encoder output (top) and the output from the positional feedback circuit used for gantry control (bottom). In this design, the rising edges occur simultaneously. Therefore, it is likely that the encoder library measures pulses on the falling edge.*

This is duplicated for each axis (using both available rotary encoder libraries and four interrupt pins). Currently, the rotary encoder count will increase with every step, regardless of gantry direction, and so positional feedback is not yet achieved.

To determine position, the change in count must be positive when direction is positive, and negative when direction is negative. The function `getPos`, summarised in *Figure 2.5.3* and shown in *Appendix B*, is called be called whenever there is a change in direction, or the gantry position must be known.

```
% Loop between d = 'x' and d = 'y'
rc = readCount(a.encoder.(d));

if v.direction.(d) % Travelling in positive direction
      % update position with change between old count and current count
      v.pos.(d) = v.pos.(d) + (v.count.(d) - rc);

      % update current count
      v.count.(d) = rc;
else % Travelling in negative direction
      % update position with change between old count and current count
      v.pos.(d) = v.pos.(d) + (rc - v.count.(d));

      % update current count
      v.count.(d) = rc;
end
```

Figure 2.5.3: *Function `getPos`, used to update `pos` variable by adding or subtracting change in encoder reading, depending on the direction of travel.*

While travelling to a specific location, this function is called repeatedly to get accurate positional information. To improve efficiency, both axes are only checked if both are enabled, otherwise the script will only check the axis expected to change.

In the hours before the demonstration, the step counting system began to freeze at high frequencies. Maximum frequency was previously limited to 4kHz, as it was found this gave a good balance between speed, while staying well within the gantry limits. For the demonstration, maximum frequency was limited to 1.8kHz, as this was below where positional feedback appeared to stay reliable. Although full diagnosis could not be performed, some possible explanations for this change in behaviour could include induced noise from circuitry added to control the gripper, or loose connections. One other group with a similar setup observed similar problems appearing at a similar time.

3.0 Magnet Detection and Marking

3.1 Sensing Sweep

Various sensing paths were investigated *(Figure 3.1.1)*. The first three methods would allow for complete coverage of the board. The final, probability based method, intended to skip areas of the board to reduce scanning time through estimations on where the magnets would be placed (namely, less likely to be near the edges of the board due to cup placement), or based off readings while scanning (very low signal suggesting no magnets are nearby). After analysis through controlled convergence, this concept was removed due to its complexity to implement and difficulty to test effectiveness with simulated magnet positions.

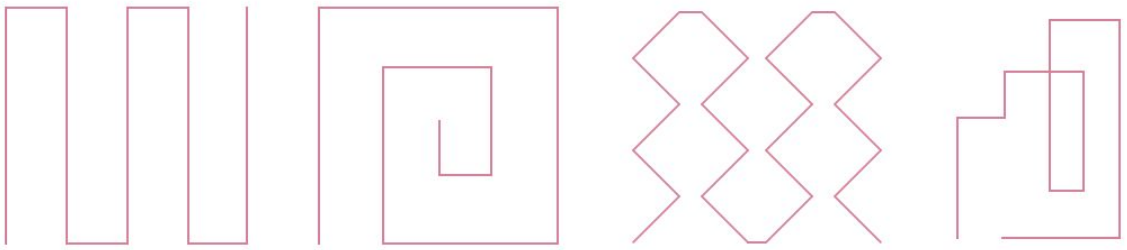


Figure 3.1.1: *The four main sensing paths considered for the scanning phase.*

An initial sensor head concept consisted of three hall effect sensors in a triangle, allowing for triangulation of strongest signals to help determine magnet centres. This was determined to be difficult to implement without necessarily improving detection accuracy, and so the final design consisted of three sensors placed linearly. This means optimum scanning is performed bidirectionally, as shown in *Figure 3.1.2*, and therefore the square sensing path was selected.

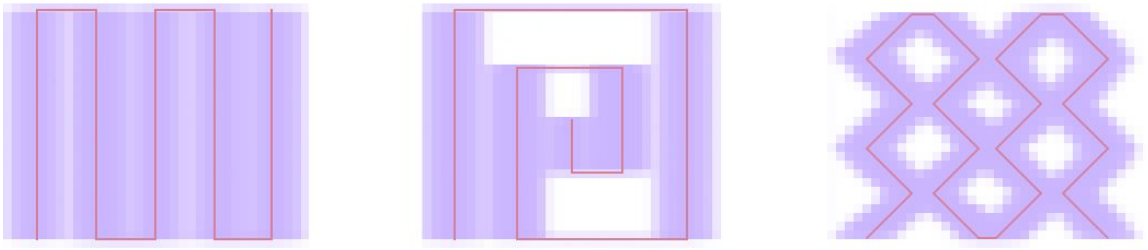


Figure 3.1.2: *Comparison of sensing paths, with an overlay showing input signal from three hall effect sensors placed horizontally. Blue indicates signal is received, whereas white indicates a lack of sensing within that area.*

Another benefit of the selected sensing path was the ease of collecting data. An overview of the data collection function is shown in *Figure 3.1.3*, the full function available in *Appendix B*. The data is collected based on the position of the gantry. As updating gantry position is comparatively slow, data readings are offset slightly in the direction of travel *(Figure 3.1.4)*. Due to the constant scanning speed, an alternative method to determine measurement points is by taking readings at set time increments. However, this requires high timing accuracy, which may be offset by time taken to read sensors. Any timing offset would distort the entire columns values, and substantial errors could occur if the MATLAB instance hung. The method implemented was comparatively 'error correcting', should a delay occur. The output is an m by n matrix of sensor readings, corresponding to their physical position on the gantry board.

```
calculate desired scanning grid resolution
determine number of vertical sweeps required

for number of vertical sweeps
      start moving in vertical direction at constant speed
```

```
    while vertical end has not been reached
          take sensor readings, add to matrix in correct column for scan
          update gantry position
          wait until gantry is in next grid row

    move gantry horizontal, in line with next vertical sweep
```

Figure 3.1.3: *Pseudocode for scanning phase logic. Special cases are implemented to allow the sensor to scan vertically up or down, and on the final vertical scan, the gantry will not move horizontally, to maximise scan area.*

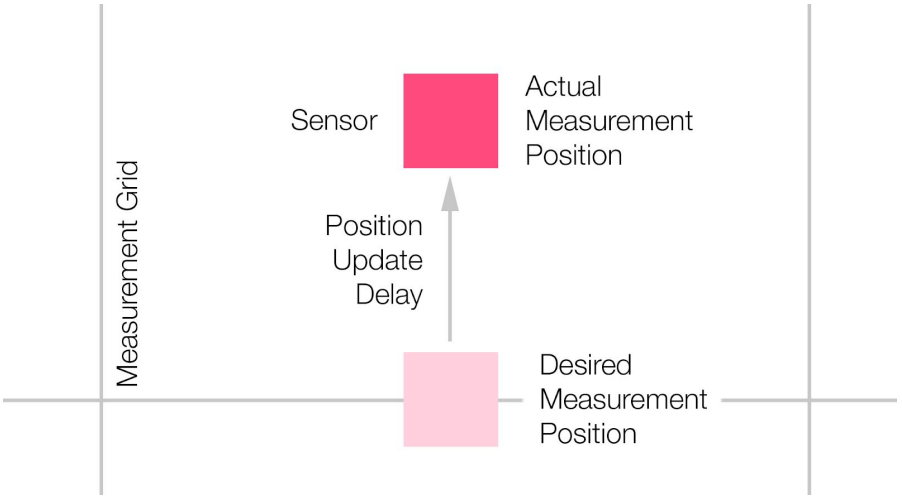


Figure 3.1.4: *Representation of delay in updating gantry position resulting in movement of the gantry by the time the hall effect sensors are read. The position offset will vary based on the previous position update, however during testing it was determined to be insignificant. If the gantry completely misses a measurement due to read delays, two (or more) measurements will be taken in quick succession to 'catch up', and an error is displayed for reference.*

3.2 Data Cleaning and Processing

As electronic and mechanical system completion occurred late in the design timeline, physical scanning data was not available when implementation of data processing was required. Simulated data from Bonvoisin *(2019)*, was used to test algorithms. As the test data introduced random noise, noise reduction methods were researched. Due to existing experience with photography, videography, and 3D rendering, methods for noise reduction in these areas were considered for use on the sensor data. MathWorks' *(2016a)* Image Processing Toolbox includes several common image processing functions. Several of these, including a pre-trained convolutional neural network denoiser, are compared in *Figure 3.2.1*.

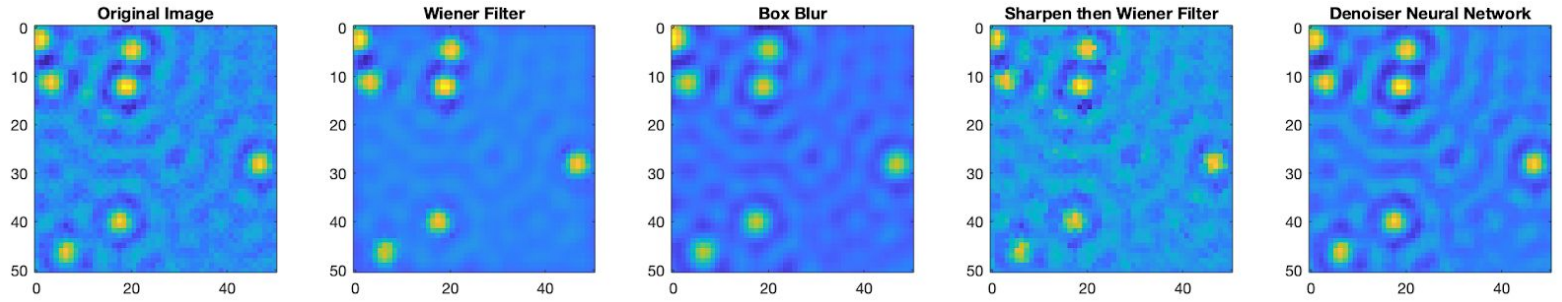


Figure 3.2.1: *Comparison of various image noise reduction techniques built into MATLAB Image Processing and Deep Learning toolboxes. The denoiser neural network was unimpressive, this is likely as the noise training images are dissimilar to the noise present in this sensor sample data.*

Wiener filtering was determined the most effective. The optimal input neighborhood size was determined through experimentation, as shown in *Figure 3.2.2*.

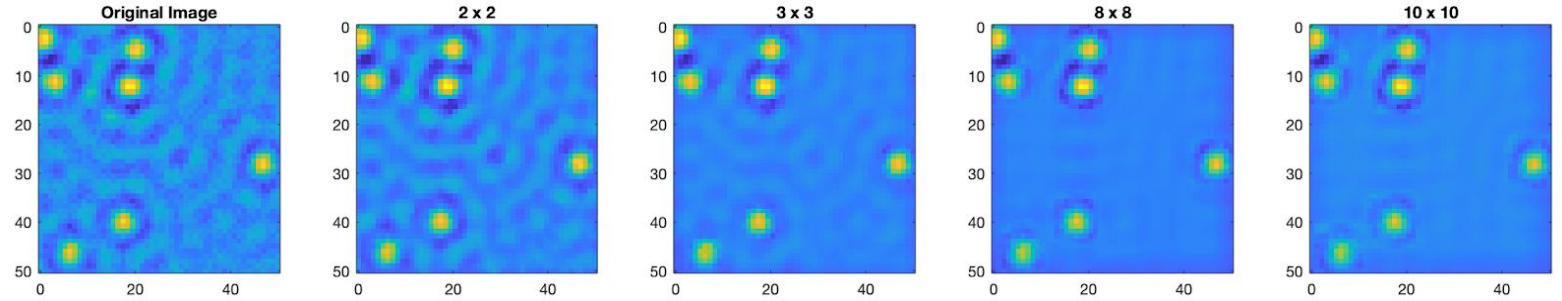


Figure 3.2.2: *Comparison of neighbourhood sizes (area at which surrounding pixels will have an effect on the filtering result). Square filters were used as the image was not biased in one direction (i.e. the magnets were circular not oval).*

Although higher neighbourhood sizes produced less noisy images, distortions near magnets were observed (high value signals) which would be more likely to cause issues than standard noise (low value signals). Use of a floor (high pass) filter to remove remaining noise was experimented with in *Figure 3.2.3*.

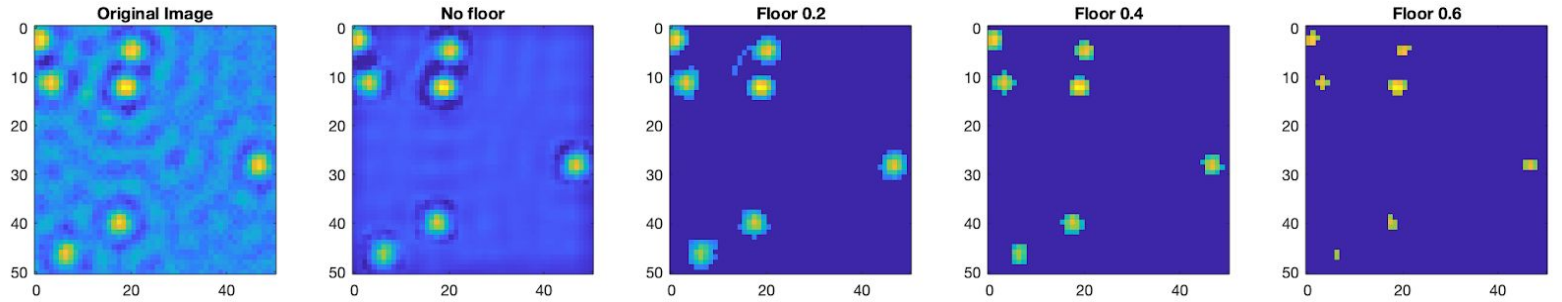


Figure 3.2.3: *Comparison of high pass filters. The input was between 0 and 1, with strongest readings closest to 1. This worked as noise was primarily weak; in fact with the sample data, no noise was stronger than 0.2, and therefore all noise was removed from non-magnet areas above this filter level.*

Although effective at removing noise, a floor filter will completely destroy signals below its threshold. This may not be preferred if magnet signals are lost during filtering. An alternative, similar filter is an 'S curve' filter; similar to the 'curves' tool in most image adjustment software. It allows signals to be boosted or suppressed based on their input strength. A comparison between S curve filter parameters is shown in *Figure 3.2.4*.

All of the parameters tested were tuned for the sample data, however as soon as a real scan could be performed, the input parameters were altered to maximise filtering efficiency. The real data was substantially less noisy than the sample data, and so only a modest Wiener filter was resultantly used alongside an S curve.

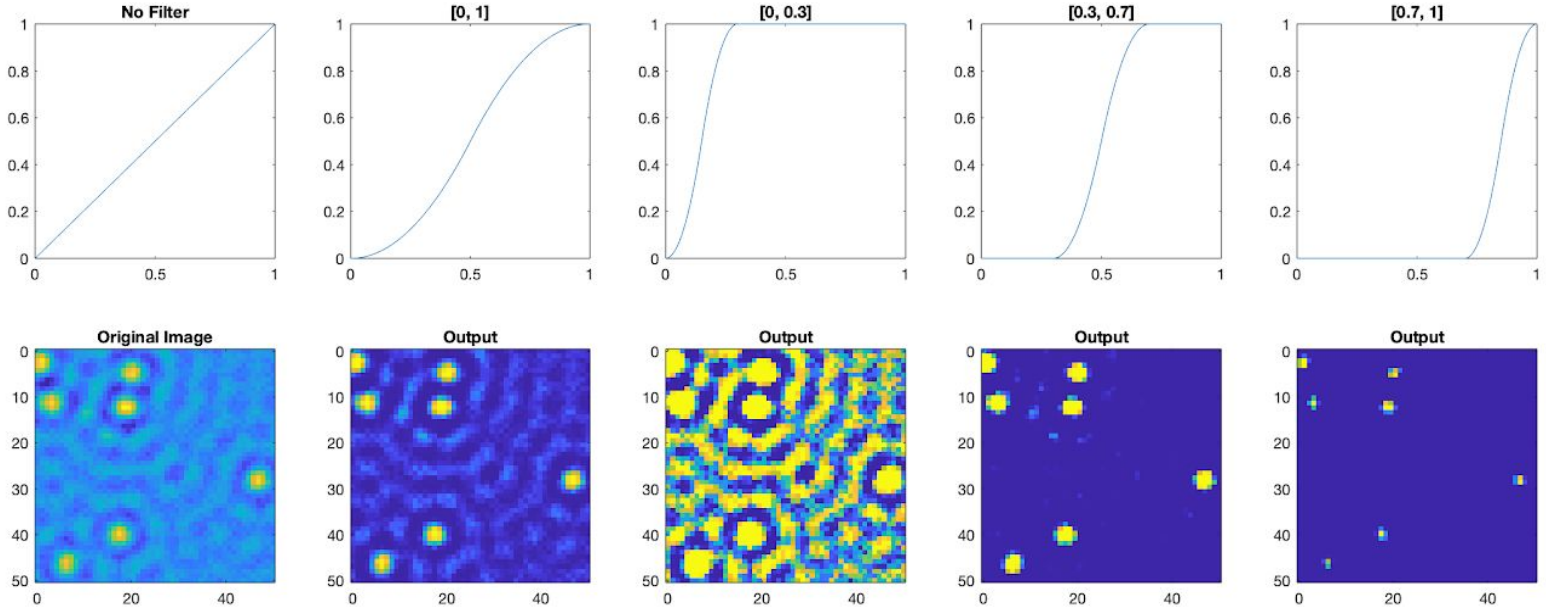


Figure 3.2.4: *Comparison of S curve filters, and their output after processing.*

3.3 Centre Point Detection

Within the Image Processing Toolbox, the function `goToPos` uses circular Hough transform *(Pedersen, 2007)* to find circles present within an image. As the magnets appear as rough circles, this was the first attempt at magnet positioning. However, after testing this method appeared unreliable, likely due to the non-sharp edges of magnet sensing.

Upon examining provided lecture scripts, it was noticed that use of a contour plot results in a number of circles surrounding the strongest signals; the magnet centres. Research into the `contour` function revealed that the vertices of the plot could be returned as a vector, in the form shown in *Figure 3.3.1*.

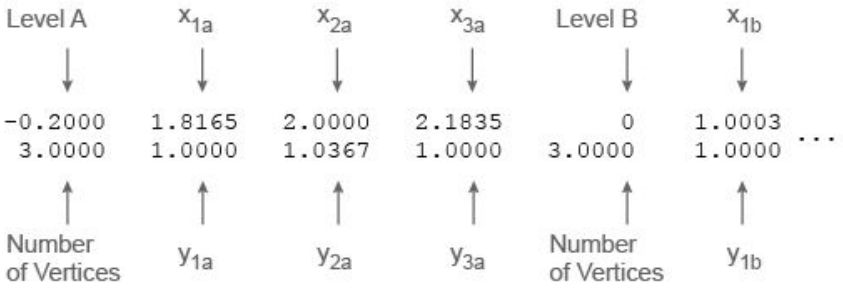


Figure 3.3.1: *Output matrix from a contour plot. The pattern will repeat for all shapes within the plot (MathWorks, 2016c).*

A script was produced which loops through all vertices within one shape, and averages their x and y values, therefore finding the centre point of the shape *(Appendix B)*. This script loops over all shapes in the contour plot, until a number of centre points corresponding to various sensor level boundaries are collected. The 'strength' of the contour shape is also indicated in the matrix. In order to weight the data in favour of stronger readings, each centre point was duplicated proportionally to its strength reading. For example, a large shape following low level signal boundaries, more likely to be affected by noise, may be duplicated once or twice. A smaller shape following a high level signal may be duplicated up to 100 times (for a strength of 100%).

To determine the magnet centres from these points, the k-means algorithm is implemented. An explanation of this can be found in *Appendix C,* and from Lavrenko *(2014)*. To evaluate clusters, the number of clusters must be provided. As this is unknown, k-means is repeated from one to twelve clusters (the maximum possible number of magnets). The Davies–Bouldin index *(Wikipedia, 2018)* is used to evaluate the best 'match', and therefore the number of magnets determined to be present. Practical testing determined the k-means default squared Euclidean distance metric was inaccurate, however the `cityblock` metric produced accurate results ~97% of the time, using the randomly generated sample data. The algorithm was inaccurate when only one magnet was present, however it was assumed this would not occur in the final demonstration.

Another algorithm improvement area is when a magnet is near the boundary of the scanning area. This causes the contour shapes to be 'cut off' at the boundary *(Figure 3.3.2)*, therefore shifting the mean away from the actual centre point. Weighted averaging helps this considerably, and it was determined to be insignificant.

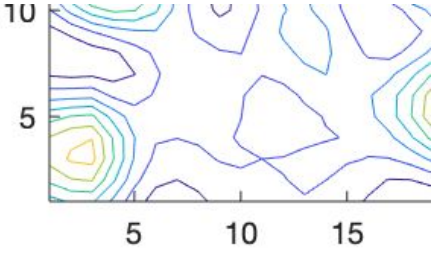


Figure 3.3.2: *Contour plot where a magnet is near the boundary of the scanning area, resulting in flat sides of the contour shapes, which offset their mean vector values from the true intensity centre.*

To determine the fastest path between collecting a cup, and marking a magnet, an implementation of the nearest neighbour algorithm was implemented *(Wikipedia, 2017)*. The problem is a variation on travelling salesman *(GeeksforGeeks, 2013)*. Instead of travelling through one dataset in the shortest path, the gantry must alternate between cup and magnet, therefore ruling out some more complicated algorithms. Exact algorithms were too computationally heavy, given the task was timed and a laptop was performing the calculations. The code can be found in *Appendix B*.

4.0 Reflection on Learning Outcomes

At the start of the course, I described my 'desired learning outcomes' as:

> I want to improve general software (MATLAB and other applicable languages) understanding so I can focus more on the outcome of future software programs, instead of how to implement them. Better understanding of algorithms and implementation of them. Better understanding of electronics whether through helping design, or just learning from other group members. Better understanding of using software to interface with physical electronics, including actuators and sensors. Better understanding of underlying technologies used within these physical electronics (i.e. not only using an ultrasonic sensor in code, but how the microcontroller 'speaks' to it), in general terms.

My understanding of MATLAB code has substantially improved, as would be expected through practice. My knowledge has broadened into areas of image processing and clustering algorithms, which have applicability in additional areas such as machine learning *(Pound, 2016)*. A major area of new understanding is object oriented programming *(Rouse, 2019)*. A basic approach to OOP was used throughout programming, where three objects were used to store all data and variables shared between multiple functions. Some examples are outlined in *Figure 4.0.1*. The objects are assigned in the `setup` function, as shown in *Appendix B*.

```
a   arduino related objects
 ├──── a.a
 │    object - arduino
 ├──── a.s
 │    object - stepper motor
 └──── a.encoder
       ├──── a.encoder.x
       │    object - encoder for x dimension
       └──── a.encoder.y
            object - encoder for y dimension
```

```
v   variables
 ├──── v.cup
 │    boolean - is cup held in gripper
 └──── v.pos
       ├──── v.pos.x
       │    integer - gantry step x position
       └──── v.pos.y
            integer - gantry step y position

p   arduino pins
 ├──── p.ramp
 │    string - output pin for playTone
 └──── p.mswitch
       ├──── p.mswitch.x
       │    string - gantry x end microswitch
       └──── p.mswitch.y
            string - gantry y end microswitch
```

Figure 4.0.1: *Example objects used to store required data while performing the given task, and their components.*

My electronics understanding has dramatically increased. Before this project, I had a basic understanding of logic circuits, and the ability to use basic circuitry and microcontrollers. Combined with knowledge from ME30295 *(University of Bath, 2019)*, through this project I used Karnaugh maps to generate logic circuits, and greatly improved my understanding of 7400 series logic ICs. My core understanding of circuitry improved greatly, for example the requirement of pull down resistors and decoupling capacitors. My understanding of 555 timer chips has increased from incredibly limited and high level, to a basic understanding of the low level functionality of the integrated circuit *(Eater, 2016)*. My confidence in developing logic circuits has increased dramatically, and understanding of general circuit design has improved.

My understanding of the interface between software and electronics has improved. I better understand the limitations of digital electronics (non instantaneous state change, signal overshooting, noise). I learned about how encoders function, and how they can interface with software to provide positional feedback. This project was the first time I have used stripboards, and through the development and assembly of various required circuits I have developed by prototyping ability within electronics. My understanding of underlying technology behind software/hardware communication has improved slightly, however due to limited understanding requirements I have not developed this area as much as other focus areas.

5.0 Conclusion

The implementation of software and electronics to control the gantry, and detect magnetic treasure was largely successful. Gantry movement was controlled accurately and reliably. Desired gantry position (in x and y steps from zero) could be entered, and `goToPos` would make corrections to ensure the gantry stopped within a predefined area surrounding that point

(20 steps was the ‘close enough’ limit). A test script involved repeating requests to an array of different positions, before returning to zero. The movement appeared to repeat with no systematic errors. Logic of gantry movement appeared to be without issue upon testing.

Adaptation of the original circuit to simplify and improve functionality was successful through repurposing of the 555 timer circuit. The switch circuit functioned without errors throughout testing. The positional feedback circuit developed issues late in the project, and so alterations to software were made to accomodate.

The sensing and magnet detection process was incredibly successful. Although the square sensing path chosen was fairly basic, it produced reliable data form the three hall effect sensors. Various standard data processing methods were compared, and the selected methods were incredibly accurate in testing. Using simulated data, the combination of signal cleaning (S curve and Wiener filtering), and cluster analysis (k-means and Davies–Bouldin indexing) was highly effective at accurately detecting the magnet positions (successful ~97% of the time). The limitations of the processing were understood and determined to be insignificant (magnets near board edges, and condition where only one magnet is present). Limited testing with real sensing data appeared to be even more reliable than simulated data due to lower noise levels.

Future improvements could be made in the scanning phase; the gantry must traverse the entire board to build a scan map, and data points were frequent but still limited. Higher resolution could be achieved through alternative positional feedback mechanisms, and smarter traversal algorithms such as probability based or live-scanning could be implemented given more time.

References

Bonvoisin, J. (Oct 2019) Mechatronics Design Project 1. Available at: https://github.com/jbon/mechatronics-design-project (Accessed: 5 Dec 2019)

Bouhraoua, A. (25 Sep 2005) Fundamentals of Computer Engineering

Craig, K. (27 Mar 2017) How Does an Engineer Predict Step Motor Pull-Out Torque?. Available at: https://www.designnews.com/content/how-does-engineer-predict-step-motor-pull-out-torque/75338778433048 (Accessed: 4 Dec 2019)

Eater, B. (17 Mar 2016) Astable 555 timer - 8-bit computer clock - part 1. Available at: https://www.youtube.com/watch?v=kRlSFm519Bo&t=1403s (Accessed: 4 Dec 2019)

eTechnophiles. (26 Dec 2017) How To Change Frequency On PWM Pins Of Arduino UNO. Available at: https://etechnophiles.com/change-frequency-pwm-pins-arduino-uno/ (Accessed: 4 Dec 2019)

Georgilas, I. (2019) Mechatronics Design Project 1

Jensen, R. (13 Jan 2016) How to generate edge-triggered pulse. Available at: https://electronics.stackexchange.com/questions/211024/how-to-generate-edge-triggered-pulse (Accessed: 5 Dec 2019)

Lavrenko, V. (19 Jan 2014) K-means clustering: how it works. Available at: https://www.youtube.com/watch?v=_aWzGGNrcic (Accessed: 5 Dec 2019)

MathWorks. (2016a) Image Processing Toolbox

MathWorks. (2016b) Noise Removal. Available at: https://uk.mathworks.com/help/images/noise-removal.html (Accessed: 5 Dec 2019)

MathWorks. (2016c) Contour. Available at: https://uk.mathworks.com/help/matlab/ref/contour.html#mw_6bc1813f-47cf-4c44-a454-f937ea210ab6 (Accessed: 5 Dec 2019)

Pedersen, S. (Nov 2007) Circular Hough Transform

Pound, M. (14 Sep 2016) K-means & Image Segmentation - Computerphile. Available at: https://www.youtube.com/watch?v=yR7k19YBqiw (Accessed: 5 Dec 2019)

Rouse, M. (Apr 2019) Object-Oriented Programming (OOP) . Available at: https://searchapparchitecture.techtarget.com/definition/object-oriented-programming-OOP (Accessed: 5 Dec 2019)

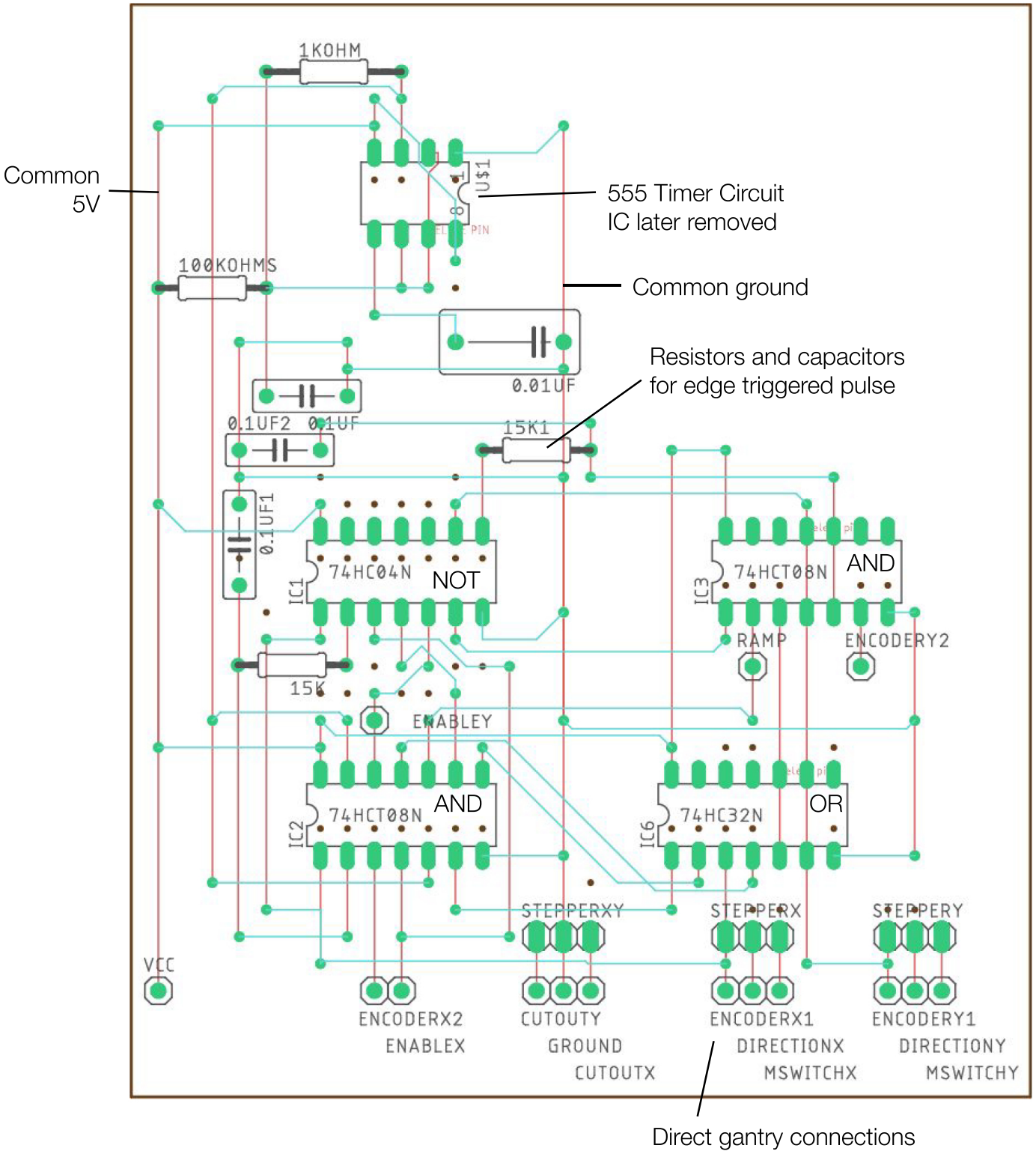
Shirriff, K. (Sep 2009) Secrets of Arduino PWM. Available at: http://www.righto.com/2009/07/secrets-of-arduino-pwm.html (Accessed: 4 Dec 2019)

University of Bath. (2019) Electronics, Signals and Drives

Wikipedia. (16 Apr 2018) Davies–Bouldin index. Available at: https://en.wikipedia.org/wiki/Davies%E2%80%93Bouldin_index (Accessed: 5 Dec 2019)

Wikipedia. (20 Sep 2019) Flicker fusion threshold. Available at: https://en.wikipedia.org/wiki/Flicker_fusion_threshold (Accessed: 4 Dec 2019)

Appendix A: Redesigned PCB



Appendix B: MATLAB Functions

All functions are uploaded on GitHub, and can be viewed and downloaded through a web browser.

Main Repository (all scripts)

goToPos (input desired position to travel to)

getPos (update current position variables)

scanBoard (to build scan map of gantry table)

centreDetection (data processing and clustering algorithm)

traverse (nearest neighbour)

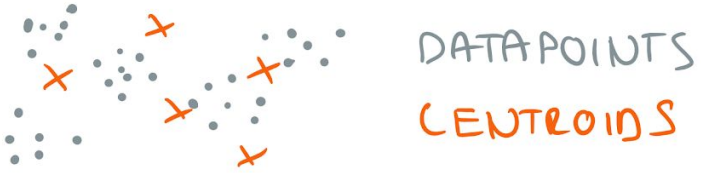
setup (assigns and initiates variables)

Appendix C: k-means Algorithm Explanation

```
function[centroids] = kmeans(npoints, datapoints)
```

CENTRES OF MAGNETS — HOW MANY MAGNETS — CONTOUR CENTRE POINTS

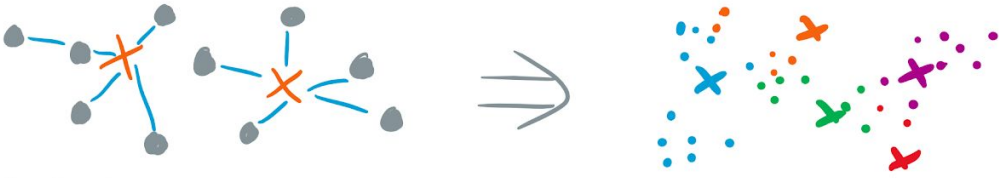
```
place initial centroids C(1:npoints) at random locations
```




```
while convergence is not reached
```

LOOP UNTIL CLUSTERS ARE FOUND

```
for each point in the dataset
    find the nearest centroid C(n)
    assign the point to cluster n
```



```
for each cluster n
    average the positions of all data points in cluster n
    move centroid C(n) to this average position
```

```
%% Convergence is reached when no points change clusters between
   iterations; the system is steady state.
```