Safety Critical 2D Pen Plotter

Arran Jones

954187

Project Dissertation submitted to Swansea University in Partial Fulfilment

for the Degree of Master of Science



Department of Computer Science

Swansea University

15th December 2020

# 

# Declaration

This work has not been previously accepted in substance for any degree and is not being con- currently submitted in candidature for any degree.

Signed Arran Jones**** (candidate)

Date 15/12/20****

Statement 1

This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed Arran Jones**** (candidate)

Date 15/12/20****

Statement 2

I hereby give my consent for my thesis, if accepted, to be made available for photocopying and inter-library loan, and for the title and summary to be made available to outside organisations.

Signed Arran Jones**** (candidate)

Date 15/12/20****

# 

# 

# 

# Abstract

This document is a report on the progress and findings of a dissertation to implement a 2D pen plotter using safety critical design and implementation techniques. This project is split into 4 major sections. The conversion of DXF files into GCODE, the conversion of rasterized images (such as a PNG file) into GCODE, a simulator for a 2d pen plotter robot and a robust GUI for controlling the entire system. The original plan was to create a robot out of lego mindstorms which would move a pen around a sheet of paper to draw the product, but unfortunately COVID-19 caused the country to go into lockdown and obtaining the lego mindstorms kit became problematic. It was then that a simulator would be necessary to complete the project.

This project will require rigorous testing, multiple fail-safes and emergency stop buttons for each section in order to produce a safe and reliable product. In order to ensure this is all complete on time, a strict time schedule will be required. This will be maintained by using a modified version of scrum. This will need to be modified as scrum is usually used for a team of developers, whereas this will be a solo project.

# Table of Contents

[Declaration 3](#_Toc58800042)

[Abstract 5](#_Toc58800043)

[Table of Contents 7](#_Toc58800044)

[Chapter 1 Sprint 1 - Background Research 9](#_Toc58800045)

[1.1 Objective 9](#_Toc58800046)

[1.2 GCODE 10](#_Toc58800047)

[1.3 DXF 11](#_Toc58800048)

[1.4 Rasterized Images 11](#_Toc58800049)

[1.4.1 Contours – Marching Squares 11](#_Toc58800050)

[1.4.2 Entire Image 12](#_Toc58800051)

[Chapter 2 Sprint 2 - DXF to GCODE 13](#_Toc58800052)

[2.1 Design 13](#_Toc58800053)

[Chapter 3 Sprint 3 – Rasterized Image to GCODE 14](#_Toc58800054)

[3.1 14](#_Toc58800055)

[Chapter 4 Sprint 3 – Simulation Engine 15](#_Toc58800056)

[4.1 15](#_Toc58800057)

[Chapter 5 Sprint 4 – Integrate Systems 16](#_Toc58800058)

[5.1 16](#_Toc58800059)

[Chapter 6 Sprint 5 – GUI 17](#_Toc58800060)

[6.1 17](#_Toc58800061)

[Chapter 7 Sprint 6 – Working Prototype 18](#_Toc58800062)

[7.1 18](#_Toc58800063)

[Chapter 8 Sprint 7 – System Testing 19](#_Toc58800064)

[8.1 19](#_Toc58800065)

[Chapter 9 Evaluation 20](#_Toc58800066)

[9.1 20](#_Toc58800067)

[Bibliography 21](#_Toc58800068)

# Chapter 1 Sprint 1 - Background Research

Before starting the project, a lot of background research is required, as there is no prior knowledge of GCODE, DXF files or safety critical systems. This chapter outlines researched topics required to gain enough understanding to begin the project and what is learnt during the research

## 1.1 Objective

The first section which requires research is safety critical systems. This is the main topic of the dissertation and there is no prior knowledge of the topic going into the project. This means that a lot of prior research is required in order to ensure the system follows a safety critical level design, implementation and testing.

Safety is a not usually talked about in conventional computer science. It is not related to reliability or security but is its own quality. Safety is the act of preventing of accidents or loss. These can come in the forms of injuries, deaths or damage to equipment.

Alongside the accidents, the main concepts to consider with safety are the risks, hazards, failures, errors and faults. The risk is a combination of probability of an accident occurring and the severity of the accident should it occur. The equation risk = p(a) ∗ s(a) is usually used as a guideline for the risk of an accident, although quantifying severity can be difficult in some unforeseen cases. Hazards are the criteria that must be fulfilled for an accident to take place. When an accident occurs, the hazard is usually very minor, but minor hazards have the ability to cause accidents of major severity. A failure is when a component of the system fails randomly due to a fault. An error is when a component fails due to a predictable fault in the system. A fault is either an error or a failure.

Safety is far more expensive to add to a system as an afterthought than it is to incorporate into the design and implementation phase during the entire development of the system. There is an eight-step guide to safety design and implementation:

1. Identify Hazards
2. Determine Risks
3. Define Safety Measures
4. Create Safe Requirements
5. Create Safe Designs
6. Implement Following Safe Designs
7. Review Previous Safety Steps
8. Test Extensively

Stages one through 3 are known as the safety analysis. These are the steps which require most forethought, as all following steps will be ensuring the chance of an accident occurring from all hazards stated in these steps are minimised. As the project is now using a simulation, instead of a robot, the hazards have already been reduced, however some will be considered during the simulation stage as if a robot were still being created.

## 1.2 GCODE

The second topic that requires research is GCODE. Prior to beginning the project, GCODE was an unfamiliar language as we had minimal knowledge of CAD. When looking through the many instructions the language consisted of, some instructions can be chosen as relevant to this project:

* G0 - Rapid Movement to specified co-ordinates. This will be useful for moving the pen when it is not meant to be drawing. This command is used in the format: G0 X1 Y1
* G1 - Linear Movement to specified co-ordinates. This will be useful for moving the pen when it is meant to be drawing. This command is used in the format: G1 X1 Y1
* G2 - Linear movement to specified co-ordinates in a clockwise arc. The arc is centred around 2 other co-ordinates. This command is used in the format: G2 X1 Y1 I1 J1
* G3 - Linear movement to specified co-ordinates in a counter-clockwise arc. The arc is centred around 2 other co-ordinates. This command is used in the format: G3 X1 Y1 I1 J1
* M0 - Program Stop
* M1 - Optional Stop
* M2 - End of Program
* F - Feed Rate, or in the case of the pen plotter, Movement Speed.

There are many more GCODE instructions, however they seem unnecessary to the project at this point in time.

In order to ensure the GCODE created is always created in an industry standard format we will be using a python library called pygcode.

## 1.3 DXF

At a basic level, DXF files are separated into features called entities. These entities define many things, such as the shape of an object, the thickness, the start and end co-ordinates, the angle at which it begins and ends, and much more.

As we are only creating a 2D pen plotter, we will be minimising the number of entities we use in order to maintain simplicity and minimise errors. The entities we will use for this project will be "LINE" which is a straight line from one co-ordinate to another, "ARC" which is a curved line, "POLYLINE" which is a combination of the 2 previously stated entities and "CIRCLE" which is a 360-degree arc defined by its centre point and radius.

In order to deal with these entities, we will be using a library called ezdxf. This will help simplify the code we produce and means that any changes in the DXF structure can be updated in the program by updating the library version.

## 1.4 Rasterized Images

In order to covert rasterized images into GCODE, the image will first need to be converted into grayscale in order to allow for a single value to govern what is considered a desired part of the image and what is not. This will be the black value of the image, a number between 0 and 1. The user will then be given the option of drawing the contours of the image, or the entire image of black value greater than the given threshold. In order to find the contours of the image, the marching squares algorithm will be used.

### 1.4.1 Contours – Marching Squares

Marching squares is an image analysis algorithm, used to determine the contours. This is done by splitting the image into small squares and finding the black value of the corner of each square, these values are then compared to the threshold provided by the user to check if the corner is within an object or not. These squares can then be categorized into 16 unique cases, defining how the contours travel through each square. Once the case of each square is determined, a form of interpolation is required to approximate the co-ordinate at which the contour crosses the boundaries of the squares. This is usually done, and will also in the case of this project, using linear interpolation. The main issue with this algorithm is the large amount of approximation and that 2 of the cases contain ambiguity as to the orientation of the contour. This would not cause any real-world problem should the program require the objects remain close but may cause issues when drawing the images.

### 1.4.2 Entire Image

In order to draw the entire image, an algorithm will be required to read the image row by row and convert the lines of pixels with a black value within the threshold of the user to line GCODE. This will probably be done with a self-defined algorithm.

# Chapter 2 Sprint 2 - DXF to GCODE

After completing the background research, the first sprint was dedicated to the conversion of a DXF file into basic GCODE instructions.

## 2.1 Design

In order to simplify this section and ensure that it can be completed during the sprint, 2 libraries will be used. These are ezdxf and pygcode. The current versions being used are pygcode 0.2.1 and ezdxf 0.13. Since completing this section, a new version of ezdxf has been released, which is causing issues with the code. Unfortunately, due to time constraints these cannot be addressed.

The first thing required for this section is to read the DXF file into python. In order to do this, we can use a function defined in ezdxf called “readfile”. This will allow us to store all entities in a variable and manipulate them as required by ezdxf.

Ezdxf also has a backend addon for matplotlib which can create a preview image of DXF files. This will be very useful later down the line when we want to give users a preview of the file chosen. As this feature is available, we can incorporate it into the file reading section.

After reading the file, the first thing that will be required is the understanding the output of the ezdxf entities, this will be done using basic print functions and then using the knowledge gained from this to extract the necessary information to create the relevant GCODE statement.

## 2.1 Implementation

While reading the file was a very simple task, creating the preview image was far from simple. When first implementing this section of the project, the matplotlib backend had a bug which just making a reference to this section of code caused it to call a method that was not yet implemented. This was very frustrating as no other library was as complete as ezdxf and after searching for a long period of time, no other library was found that could produce an image from a DXF file. It was at this point it seemed futile to implement a DXF preview. Fortunately, the library is still getting regular updates and the bug was fixed before this section of the project was completed. With some minor adjustments to the code, this feature worked well. However, the image preview produced for some DXF files is a blank image. This may be fixed in future updates, however due to the time constraints there is not much that can be done about this.

The conversion of the LINE entity into GCODE was very simple. The feature extraction of ezdxf allowed for the start and end co-ordinates of the line to be stored as lists. This meant that the x co-ordinate was simply the first element in the list and the y co-ordinate was the second co-ordinate. The pygcode functions GCodeRapidMove and GCodeLinearMove, which take x and y co-ordinates and convert them into the G0 and G1 gcode statements, could then use the start and end co-ordinates to simply convert the dxf entity to gcode.

Although the LINE entity was extremely simple, the ARC is where things started to change. As the project had now changed into a simulation rather than a robot, some changes had to be made. For simplicity, python turtles was chosen to create the simulator for the robot. Although turtles is a simple library, there are some features that meant it would not work well with conventional GCODE. The main thing that differed is that in GCODE, and arc is defined by the start point, and the centre point of the curve. Turtles however works using a start point, a radius and the angle travelled through while drawing the arc (also known as the extent of the angle) in a counter-clockwise direction. This can also be defined by: extent = end\_angle – start\_angle. In order to keep some form of industry standard feeling for this project we modified G3 to the format G03 R1 X1 Y1 S1 E1, where R is the radius of the arc, S is the start angle and E is the end angle. The start and end angles were added instead of the extent as this was how it was stored in the DXF file and this made it feel nearer to some form of industry standard. The extent can then be calculated by the simulation engine.

In order to draw CIRCLEs in turtles it is possible to set the co-ordinate to the most southern point of the circle and call the function t.circle(r). However, this did not feel like building a simulator and meant it was necessary to find a way to draw a circle in a more robotic style. This first problem to overcome was that there is no start co-ordinate stored in a DXF file, this was easily calculated by using the x co-ordinate of the centre and subtracting the radius from the centre y co-ordinate. The next issue was one that was not foreseen. Turtles measures all angles plus a multiple of 360 as the same angle, e.g. -90 and 270 are seen as the same angle. This meant that in order to draw a circle in turtles you cannot simply tell it to draw an extent of 360 as it would not draw anything. In order to overcome this, we defined a circle as two arcs of 180 extent.

After implementing the LINE and ARC entities, POLYLINEs were extremely simple as all that was required was an if statement to check each entity in the polyline to determine whether the entity is a line or an arc. It can then call the method implemented earlier to convert the entity.

The final function that is required in this sprint is to output the gcode to an output file. This will allow users who understand how GCODE works to edit it if required. However, this will not be a feature implemented in the program as it could allow users with insufficient knowledge to change the GCODE and this could be problematic.

## 2.2 Testing

# Chapter 3 Sprint 3 – Rasterized Image to GCODE

## 3.1

# Chapter 4 Sprint 3 – Simulation Engine

## 4.1

# Chapter 5 Sprint 4 – Integrate Systems

## 5.1

# Chapter 6 Sprint 5 – GUI

## 6.1

# Chapter 7 Sprint 6 – Working Prototype

## 7.1

# Chapter 8 Sprint 7 – System Testing

## 8.1

# Chapter 9 Evaluation

## 9.1

# 

# Bibliography