VECTOR GRAPHIC CONTENT LASER ENGRAVING PROGRAM ON DOBOT MAGICIAN

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Bachelor’s thesis

Mechanical Engineering and Production Technology

Spring 2022

Thien Hoang Le

Acknowledgement

I would like to thank my supervisor, Prof. Dr Sc. Christophe, who supports me in the thesis implementation process of the project with his helpful counsel and insightful feedbacks. The Robotic Department of HAMK in Riihimäki campus gave me opportunities to approach the technical devices I used in the project.

In addition, I would like to thank my parents and my girlfriend who have always supported and encouraged me. Finally, I could not finish this thesis without the help and advice from my good friends.

In memory of Grandpa,

Vantaa, May 10th, 2022

Le Thien Hoang

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Mechanical Engineering and Production Technology Abstract

Author	Thien Hoang Le	Year 2022
Subject	Vector Graphic Content Laser Engraving Program on Dobot Magician	
Supervisor	Francois Christophe	

ABSTRACT

The aim of the thesis is to develop a program of controlling an educational robot, Dobot Magician, in laser cutting and engraving process. The input source of the program is vector graphic SVG files. The Python program developed in this thesis is able to interpret the input SVG file as G-code commands and control the robot according to these commands.

The program meets the requirement of controlling the robot to execute the laser process with a vector graphic file and laser process settings as the input content. However, the program still has limitations in handling the diversity of input formats and customization ability.

Further research for alleviating the limitation of the program improves the teaching capability of the educational robot about the laser processing and G-code commands in Mechanical Engineering field.

Keywords Dobot Magician, laser, G-code, vector graphic, SVG, Python
Pages 28 pages and appendices 11 pages

Contents

Appendices

1 INTRODUCTION

In the last decade, modern technology has developed rapidly, and programming skill has become one of the most important skills to learn for young people. As a consequence, educational Robots are invested as the teaching facilities in high schools as well as universities to help students study programming. Compared to programming on computers with ordinary screen output, the educational robots are more efficient and fascinating to the students due to the visual demonstration of the programming process by robot's movements. The educational robots are not only used in teaching programming, their application in educating students in Automation and Mechanical lectures is significant. Many educational robots have the ability to perform basic functions of some industrial processes such as laser engraving, cutting, milling, and 3D printing.

Consequently, the need for research in programming and controlling the educational robot to execute some light industrial processes is becoming more critical. Additionally, laser engraving is a particular example in robot automation and CNC procedures. Therefore, this thesis research project is expected to provide the educational robots the improved capabilities in assisting students to interact with the laser engraving process.

The purpose of the thesis is to research the solution to control educational robots, specifically the Dobot Magician, in the laser engraving process. The program system has the ability to input an SVG vector graphic file and control the Dobot Magician to engrave the graphic onto the operating surface. Furthermore, the program is expected to manipulate the power of the laser head and the settings of the engraving operation.

The program provided by the publisher of Dobot Magician contains the laser engraving function. The limitation of this add-in function is its inability to process the vector graphic files. The program only supports raster graphic files, specifically .png, .jpg and .bmp. Additionally, it is impossible for users to customize the process setting. The program developed in this thesis is expected to alleviate this limitation and give the user opportunities to interact with the setting. With this, the educational application in laser engraving process can be enhanced.

The software developed in the thesis project has the capability to process only one format of vector graphic content, the SVG format. The programming interpreter of the software is Python 3.8, there is no guarantee that the software can run without any error. On the other hand, the thesis can be used as a foundation for further research in the educational robot programming field.

2 THEORETICAL AND TECHNICAL BACKGROUND

The aim of the thesis project is to find a feasible system to control the Dobot Magician in laser engraving process. An SVG vector graphic file should be used as the input of the system. The approaching method to solve the problem is parsing the vector graphic geometry elements into G-code commands. The system is programmed to process the G-code file and control the robot according to the G-code commands. For education and research purposes in the Mechanical field, G-code, a widely used CNC language in laser processes, was chosen to be operated with. The parsing procedure from SVG geometry factors to G-code is common in commercial NC laser machines. Therefore, the open-source appropriate solutions for this procedure are accessible on the Internet. This theoretical background section gives an overview of SVG, G-code, Python language, and the Dobot Magician robot. Additionally, the parsing methods from SVG geometry elements to G-code and from G-code to the robot behaviors are mentioned. Finally, the modules, libraries and its algorithms are also discussed.

2.1 Necessary terminologies and abbreviations

This section gives the definition of the terminologies used in the report.

1. CNC: abbreviated as Computer Numerical Control is a machining process in which the movements of the manufacturing tools are controlled by pre-programmed computer software. As the operation is dictated by the computer software, the CNC process commonly provides higher precision compared to the manual control machining. Therefore, CNC machining is applied widely in modern industrial manufacturing, especially when the tasks require accuracy. (Hess, 2017)
2. XML: eXtensible Markup Language, a text format from SGML (ISO 8879). This format is simple and flexible that was designed as the solution for *"large-scale electronic publishing".* (Quin, 2016)
3. Interpreted program language: The program language which is interpreted into another program language to communicate with the computer instead of directly communicating.

4. Object-oriented programming (OOP): *"Is a computer programming model that organizes software design around data, or objects, rather than functions and logic."* (Alexander, 2021)
5. High-level language: A programming language considered as high-level program language when the code written in the programming is similar to human language. Oppositely, the low-level program language is difficult for regular people to understand but it runs faster because it communicates with computer better. The advantage of the high-level program language is its comprehensibility for the amateur programmer in the moderate tasks.
6. Python packages/modules: Libraries in Python language which are compilations of code was written by other programmer(s) to perform any task. Programmers can use libraries written by others to optimize performing the required tasks.
7. Parsing and Compiling: Parsing is the process that reads the text into the internal representation such as geometry shapes, graphs, trees, etc. Compiling is translating the internal representation into another format.
8. Software extension: An add-in function which is created to perform the related additional tasks in the software.
9. Function and sub-function: Function is a block of code that runs when it is called. Sub-function is an internal function of a function.
10. Open-source: The designs or source codes of the open-source objects are publicly approachable.

2.2 Vector graphic and SVG

The input of the system is the vector graphic file. This chapter refers the characteristic of the Vector graphic and specifically the SVG format.

2.2.1 Vector graphic

Vector graphics are computer graphics images that are defined in terms of 2-dimensional (2D) points, which are connected by lines and curves to form polygons and other shapes. Each of these points has a definite position on the x- and y-axis of the work plane and determines the direction of the path. Furthermore, each path may have various properties including values for stroke colour, shape, curve, thickness, and fill. In other words, the vector graphic is a combination of countless vector objects.

In the raster graphic formats, the image is defined by the colour of many pixels. Therefore, when the image is magnified, the individual pixels are visible to human eye. On the other hand, as the content of the vector graphics are defined vectors, the basic property of vector graphics is the preservation of all the details when enlarging. Vector graphics are widely applied in logo and poster design for their capability of enlarging without loss of details. Vector graphics can be found in the SVG, EPS, PDF or AI graphic file formats. (Lutkevich, 2021)

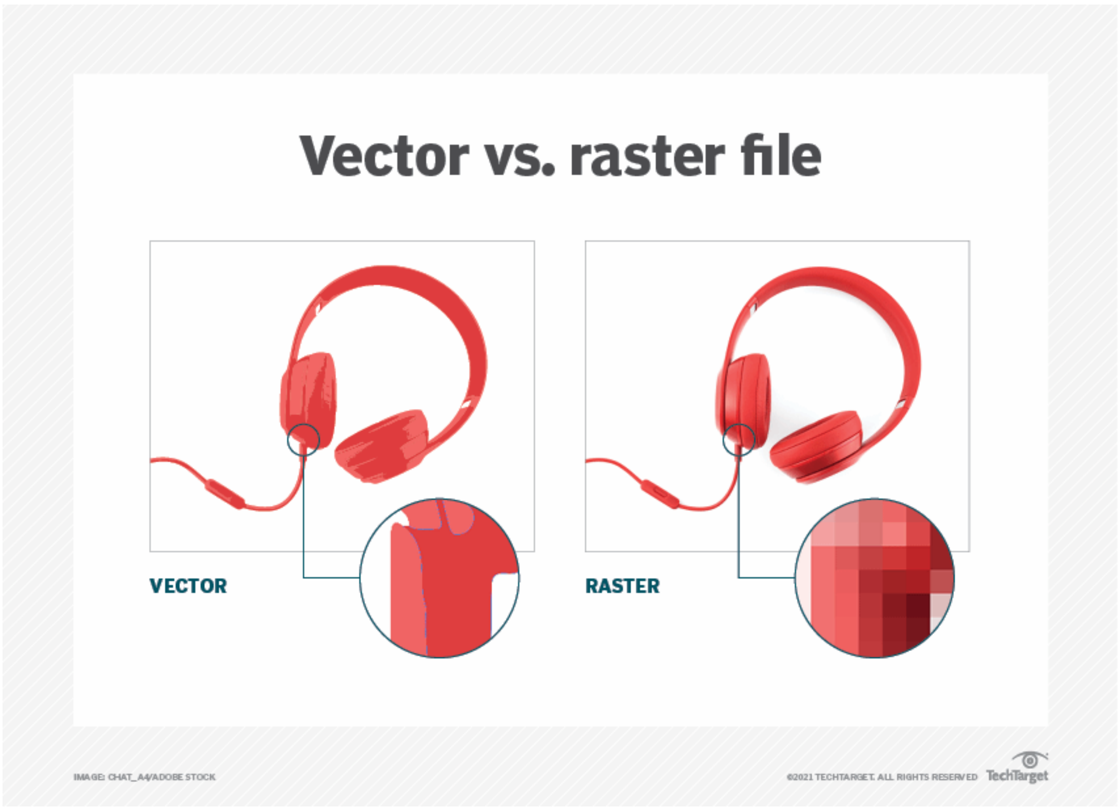
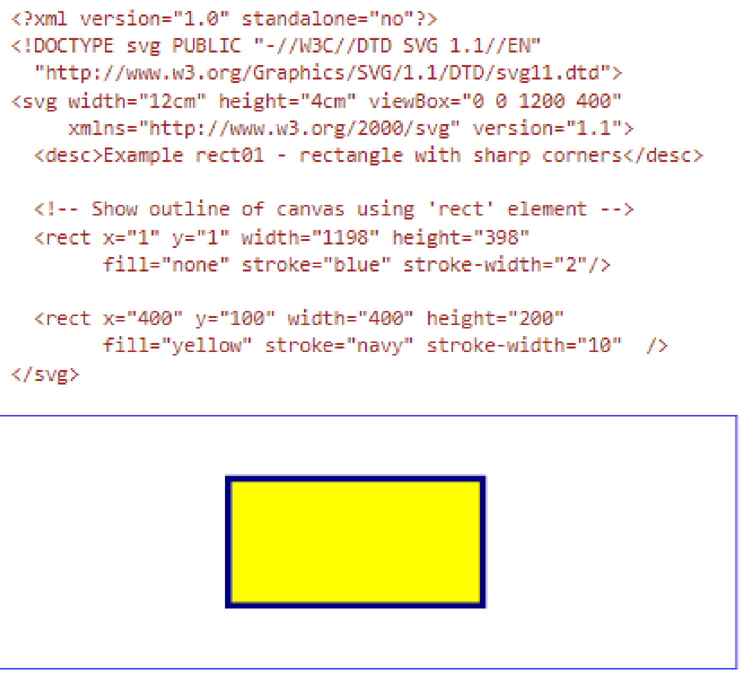

Vector vs. raster file
VECTOR
RASTER
IMAGE: CHAT_A4/ADOBE STOCK
©2021 TECHTARGET. ALL RIGHTS RESERVED
TechTarget

Figure 1. The difference between vector graphics and rastor graphics (Lutkevich, 2021)

2.2.2 SVG format

As mentioned, there are several formats of vector graphics. In this project, the most common format chosen to be the research subject is the SVG vector graphic format.

Scalable Vector Graphics, or SVG, was developed by World Wide Web Consortium (W3C). SVG is based on Extensible Markup Language (XML) which uses syntax and keywords to determine the graphic objects on 2D plane. These graphic objects are vector graphic shapes, images and text. SVG is an open standard that allows for effortless customization and editing of graphical objects. The SVG format is commonly used in Web applications where the graphic requires scalability for different display resolutions. (Dahlström, et al., 2011)



```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <desc>Example rect01 - rectangle with sharp corners</desc>

  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="1198" height="398"
        fill="none" stroke="blue" stroke-width="2"/>

  <rect x="400" y="100" width="400" height="200"
        fill="yellow" stroke="navy" stroke-width="10"  />
</svg>
```

Figure 2. An example for SVG code to represent a triangle. (Dahlström, et al., 2011)

In the document about the SVG, W3C mentions the rules to write the syntax and the keywords as well as their arrangement to determine the graphical objects in SVG. The parsing module used these rules as reference to develop the algorithms for parsing and compiling the graphical objects defined in the SVG file into the G-code commands. (Dahlström, et al., 2011)

2.3 SVG-to-gcode module

SVG-to-gcode is a library programmed in Python 3 language. The library was developed by Alex Padula. The library is the foundation for developing an Inkscape extension called J-Tech Photonics Laser Tool. The function of the library and the extension is to parse the SVG graphic file to G-code commands in the laser-cut process. The module is divided into 3 sub-modules: geometry, parser and compiler. The geometry sub-module represents the geometry curves. The parser sub-module parses and converts the SVG files to geometric curves. The compiler transforms geometric curves into G-code commands. (Padula, 2021)

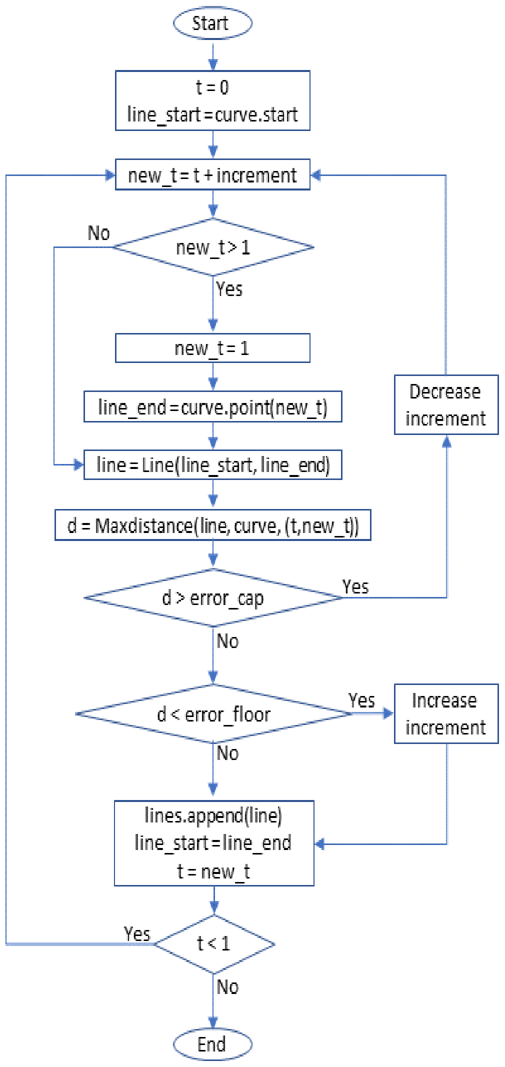


Figure 3. The flow chart of the curve approximate method

The most important function of the Padula's library is the *line_segment_approximation* function. The purpose of this function is to approximate the curves utilizing straight line segments. This function converts any curve path into small straight lines which has the approximating tolerance within the defined restriction. The function creates a list of lines which is comprised of the straight lines that approximate the curve. If the curve is a straight line, the list of lines contains only that straight line. When the curve is not a straight line, curve approximating method is applied to determine the approximate straight lines to be appended into the created list of lines. There is a coefficient "t" in the function which indicates the portion of the curve under evaluation. For example, curve.point(t=0) is the starting point of the curve; curve.point(t=1) is the ending point of the curve; similarly, curve.point(t=0.5) is the point at the middle of the curve. First, a straight line is created by connecting the starting point and the ending point of the curve (“t”=0 to “new_t”=1). The program evaluates the maximum distance between the generated line and the whole curve. If the maximum distance exceeds the upper limit which is defined in the tolerance section, the program considers the line as inappropriate and decreases the “new_t” value to find a more accurate line. When the function defines a suitable “new_t” which generates a qualified straight line, the line is then appended to the list of lines created in the beginning of the function. After a straight line is accepted into the list of lines, the loop is repeated to determine a new line by assigning the t value of the new cycle to the new_t of the previous cycle. By doing so, the starting point of the next line is identical to the ending point of the previous line in the list. The loop is ended when the t value exceeds 1, which expresses the ending of the curve. There is also a lower limit for the tolerances value. If the error between the segment line and the curve is lower than this value, the increment of the t value is increased to reduce the unnecessary amount of line segments.

The module has a file for the user where all the sub-modules and functions are imported to. The user is able to adjust the name of the input/output directory. Most importantly, the user has the capability to set the parameters of the functions to decide the settings of the laser process. G-code file is generated based on the settings which were set by the user. The program then outputs the G-code file and saves it in the directory folder for the next stage of the process.

2.4 G-code

G-code is an ISO standard (ISO6983-1) used in program and dictating the movement of the CNC machines or the robot arms (Nilsson, 2016). The G-code commands control the direction of the movement, the movement speed, the depth of the cutting, etc. The generated G-code is loaded onto the machine to operate the process. The G-code can be manually written by the operator or auto-generated by the CAM software.

The G-code used in laser processing is a simplified version of the G-code in different CNC machining processes such as milling, turning, turning or drilling. The G-code for laser process which was researched in this thesis only relies on 5 G-command. G0 or G00 is created for rapid movement when the laser head approaches the starting point and leaves the ending point of an operating path. G1 or G01 defines the linear movement in laser operating heat procedure. Lastly, G2 or G02 and G3 or G03 respectively perform the arc movement in clockwise and counterclockwise orientation. The G-commands with the F-parameter set the speed of the corresponding G movement. Additionally, 2 M-commands are involved in the laser operation. While M3 or M03 controls the power of the laser head when it performs the cutting or engraving, M5 or M05 has only function to turn the laser off. The M5 commands commonly come with the G0 commands afterwards.

2.5 Dobot Magician

The educational robot researched in the thesis is Dobot Magician. Dobot Magician is a 3-axis robot arm commercialized in 2017 by Shenzhen Yuejiang Technology Co., LTD. The robot is usually used for educational purposes in schools and universities. Dobot Magician supports secondary development by over 20 programming languages. The robot arm has the ability to connect with multiple external devices via its 13 I/O connection ports. The supplier provides various demo packages in over 20 programming languages to control Dobot Magician based on the communication protocol of the robot. (DOBOT Magician - Lightweight Intelligent Training Robotic Arm - An all-in-one STEAM Education Platform, n.d.)

2.6 Dobot Magician DLL and API Description

The Dobot Magician producer provides the Dynamic Link Library (DLL) file which instructs other programs to call the controlling functions of the robot. In the DLL source code written in Python, the developers can find information on how to call the functions and the meaning of the parameters in those functions. This information is further explained in Dobot Magician API Description. The Dobot Magician API Description used in the project is the version V1.2.2 released on 06 November 2018 by Shenzhen Yuejiang Technology Co., Ltd.

2.7 Python 3

Python 3 is an interpreted, object-oriented, high-level programming language. Python syntax is simple and easy to learn, therefore suitable for beginners. Python 3 supports modules and packages which help programmers reuse code and arrange code neatly. As Python is an open-source language, it has a large community and free-of-charge standard library so that programmers can easily use external modules and functions.

Dobot Magician supports Python as a secondary development language. Dobot Studio software programming function generates Python code to interact with the robot. On the official website of the supplier, users can download a Dobot Magician demo package for Python. This package includes controlling programs for Dobot Magician written in Python 3 based on communication protocol of the robot.

2.8 Analytic Geometry

The theory of analytic geometry was applied in calculating and defining the circular point in the arc movements of the robot.

The line equation in x-y coordinate system is written as

$$y = ax + b \tag{1}$$

The circle equation in x-y coordinate system which has the coordinates of the center point $I(h, k)$ and radius r is defined as:

$$(x - h)^2 + (y - k)^2 = r^2 \tag{2}$$

In the process of interpreting the arc command, the linear equation and the circle equation were applied to define the coordinates of the suitable circular point on the arc movement path.

Slope m of the line that ggoesthrough 2 points $A(x_{1,}y_1)$ and $B(x_{2,}y_2)$ is:

$$m = \frac{y_2 - y_1}{x_2 - x_1} \tag{3}$$

If 2 lines l_1 and l_2, which respectively have the slope m_1 and m_2, are perpendicular, then

$$m_1 . m_2 = -1 \tag{4}$$

(Boljanovic, 2016).

2.9 Orientation of 3 ordered points

Analyzing the slope of the line segments generated by the connected lines of 3 points helps in defining the orientation of this *"triplet of point".* The input is the coordinates of 3 points set in order, which are $p_1(x_1, y_1)$, $p_2(x_2, y_2)$ and $p_3(x_3, y_3)$.

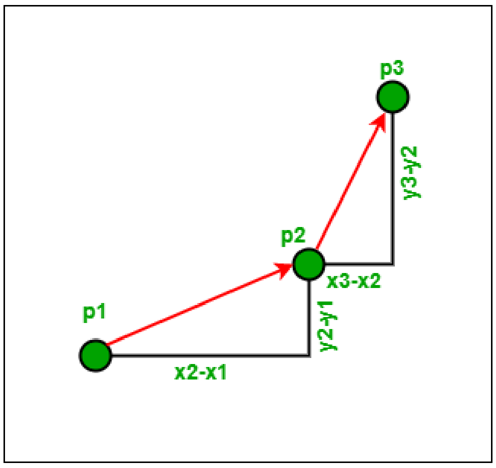


Figure 4. The figure explains the slope calculation of the connecting lines (Agrawal, 2021)

To define the orientation of the *"triplet of point",* the "*slope of line segment*" created by p_1 and p_2 is compared to that of the p_2 and p_3 points. The equation (3) in section 2.7 was applied to calculate the slope of each pair of points in the triplet.

After 2 slope values are calculated, they are analyzed to define the orientation. There are 3 possibilities of result to determine the orientation of the set of 3 points p_1, p_2, p_3:

- $slope(p_1, p_2) > slope(p_2, p_3)$: the orientation is clockwise.
- $slope(p_1, p_2) < slope(p_2, p_3)$: the orientation is counterclockwise.
- $slope(p_1, p_2) = slope(p_2, p_3)$: the orientation is collinear. (Agrawal, 2021)

3 IMPLEMENTATION

3.1 Overview

Below is the flow chart from the beginning to the end of the processing system.

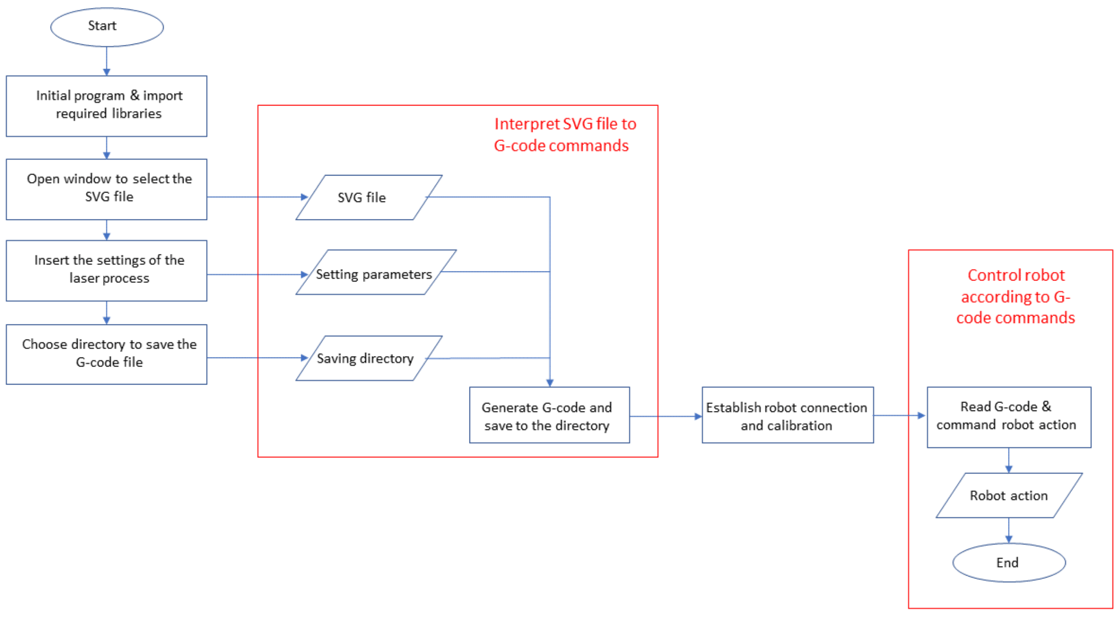

Start
Initial program & import required libraries
Open window to select the SVG file
Insert the settings of the laser process
Choose directory to save the G-code file
Interpret SVG file to G-code commands
SVG file
Setting parameters
Saving directory
Generate G-code and save to the directory
Establish robot connection and calibration
Control robot according to G-code commands
Read G-code & command robot action
Robot action
End

Figure 5. The overall flowchart of the developed program

According to the flow chart, the SVG file is input to the SVG-to-gcode module to be parsed to the G-code file. There are some parameters that can be modified such as operating speed, number of passes and pass depth. Subsequently, the generated G-code file is handled by the Read-gcode function. The robot movements are controlled by the program based on the data from the g-code commands. The user needs to input the required distance from the laser head to the operating surface. In this implementation section, each step in the process is explained thoroughly.

3.2 Vector graphic to G-code

The main procedure of this process is to interpret the SVG file into G-code commands, which is the responsibility of the SVG-to-gcode module. In the source code of the module, there is a basic_usage.py Python file which allows the user to input the directory of the SVG file to interpret it into G-code. The source code of basic_usage.py and the required modules are imported to the main program. The input settings of the laser operation (including movement speed, cutting speed, pass depth and number of passes) are programmed to be the parameters which users can input whenever the main code starts.

3.3 Process the G-code file to control the robot

3.3.1 Overview

This function of the system is specialized in reading each line of g-code orderly and controlling the robot according to the g-code command in that line. To initiate this procedure, each line of G-code is analyzed by the sub-function called line-to-param. Line-to-param sub-function is programmed to read the parameters in the G-code line and write them into a pre-made dictionary. The program uses the data in the created dictionary to decide which movement function to call out. As seen in figure 4, the data process works based on an if-else structure. Specifically, the “G” value in the dictionary is checked if it is G00, G01, G02 or G03 to continue to the following level of the structure. In this level, if the “F” value of the dictionary exists, the system proceeds to call speed setting functions according to the “F” value. Otherwise, the coordinate values are detected, and the robot movement functions are called afterwards. If the ‘G” value does not exist, “M” value is investigated. For this situation, there are 2 possibilities: M03 means power on the laser head and the power is set on “F” value, M05 sets the power to 0, which means power off the laser head. Consequently, the next line of g-code is processed until the end of the g-code command file.

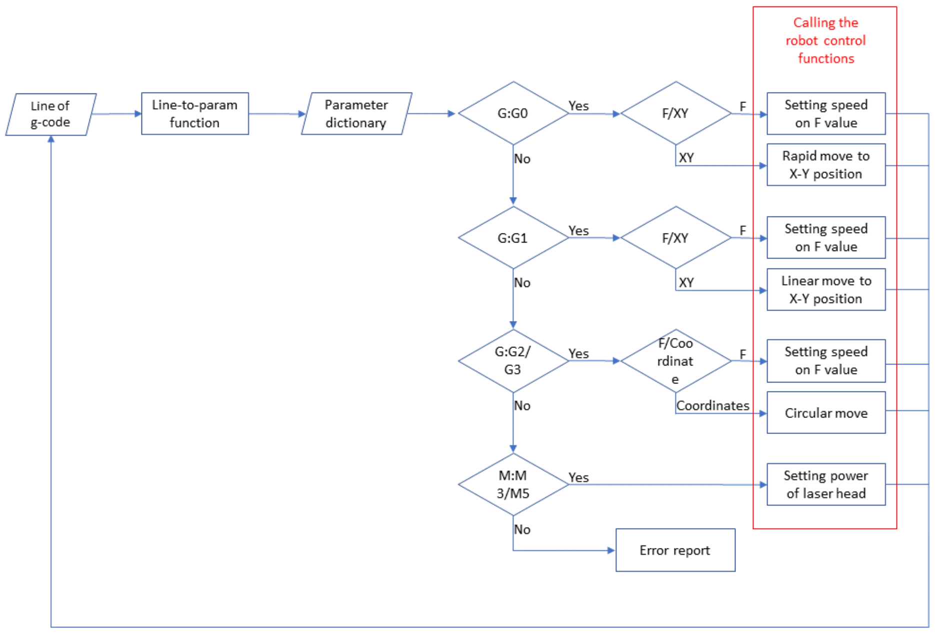
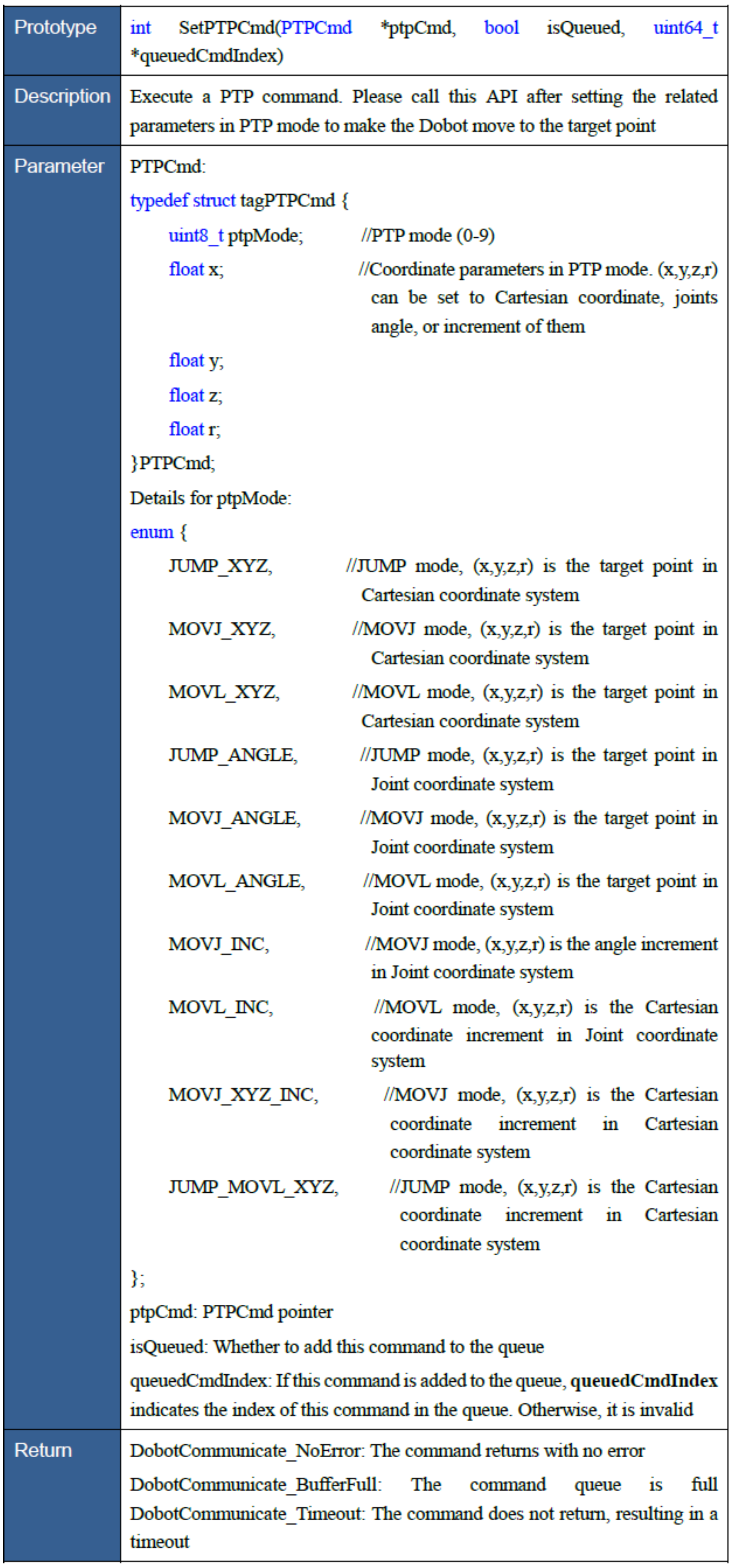


Figure 6. The detailed process to control the robot action according to the G-code commands.

The information of the parameters in the robot controlling function is determined in the DLL Python source code named DobotDLLType.py which is provided by Dobot Magician manufacturer. To call out the desired controlling function, the API description of the robot is used as a reference. The table 1 is an example for the information of the PointToPoint (PTP) command provided in the robot API description.

Table 1. Execute PTP command and the information of the parameter in the command. (Shenzhen Yuejiang Technology Co., 2018)



Prototype	int SetPTPCmd(PTPCmd *ptpCmd, bool isQueued, uint64_t *queuedCmdIndex)
Description	Execute a PTP command. Please call this API after setting the related parameters in PTP mode to make the Dobot move to the target point

Parameter	PTPCmd:
	typedef struct tagPTPCmd {
	uint8_t ptpMode; //PTP mode (0-9)
	float x; //Coordinate parameters in PTP mode. (x,y,z,r) can be set to Cartesian coordinate, joints angle, or increment of them
	float y;
	float z;
	float r;
	}PTPCmd;
	Details for ptpMode:
	enum {
	JUMP_XYZ, //JUMP mode, (x,y,z,r) is the target point in Cartesian coordinate system
	MOVJ_XYZ, //MOVJ mode, (x,y,z,r) is the target point in Cartesian coordinate system
	MOVL_XYZ, //MOVL mode, (x,y,z,r) is the target point in Cartesian coordinate system
	JUMP_ANGLE, //JUMP mode, (x,y,z,r) is the target point in Joint coordinate system
	MOVJ_ANGLE, //MOVJ mode, (x,y,z,r) is the target point in Joint coordinate system
	MOVL_ANGLE, //MOVL mode, (x,y,z,r) is the target point in Joint coordinate system
	MOVJ_INC, //MOVJ mode, (x,y,z,r) is the angle increment in Joint coordinate system
	MOVL_INC, //MOVL mode, (x,y,z,r) is the Cartesian coordinate increment in Joint coordinate system
	MOVJ_XYZ_INC, //MOVJ mode, (x,y,z,r) is the Cartesian coordinate increment in Cartesian coordinate system
	JUMP_MOVL_XYZ, //JUMP mode, (x,y,z,r) is the Cartesian coordinate increment in Cartesian coordinate system
	};
	ptpCmd: PTPCmd pointer
	isQueued: Whether to add this command to the queue
	queuedCmdIndex: If this command is added to the queue, **queuedCmdIndex** indicates the index of this command in the queue. Otherwise, it is invalid

Return	DobotCommunicate_NoError: The command returns with no error DobotCommunicate_BufferFull: The command queue is full DobotCommunicate_Timeout: The command does not return, resulting in a timeout

3.3.2 Coordinate transformation

The coordinate system used in the SVG file is the same as the one in the drawing canvas. This coordinate system is different from the coordinate system of the Dobot Magician. Coordinate transformation between 2 coordinate systems is required to retain the direction of the drawing.

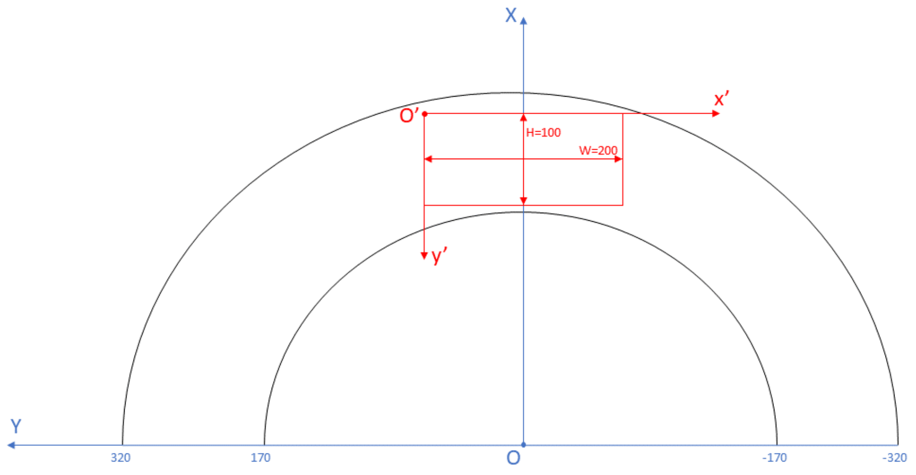


Figure 7. The graph of the SVG canvas on the Dobot Magician working space

Figure 7 illustrates the SVG canvas on the working space of the robot. The red rectangle with $x'y'$-coordinate system represents the canvas where the SVG images are drawn. The working space of the robot in the XY-coordinate system is limited by 2 semi-circles which have a radius of 170mm, respectively 320mm. The coordinates of the origin O' of the canvas coordinate system is (x_o, y_o) in the robot coordinate system. The coordinate matrix in the XY-coordinate system is the summary of the product of coordinate matrix in the $x'y'$-coordinate system with inverse of the matrix T and the coordinate matrix of the point O' in XY-coordinate system (Croft & Davison, 2019).

$$\begin{bmatrix} X \\ Y \end{bmatrix} = T^{-1} \begin{bmatrix} x' \\ -y' \end{bmatrix} + \begin{bmatrix} x_o \\ y_o \end{bmatrix} \tag{5}$$

The matrix T is the transformation matrix. In this transformation, the coordinate system with inverse y-axis to one of the $x'y'$-coordinate system is rotated an angle $\theta = -\frac{\pi}{2}$ compares to the XY-coordinate system, so the matrix T = $\begin{bmatrix} cos\frac{-\pi}{2} & sin\frac{-\pi}{2} \\ -sin\frac{-\pi}{2} & cos\frac{-\pi}{2} \end{bmatrix}$ and the matrix $\begin{bmatrix} x' \\ -y' \end{bmatrix}$ is applied to the equation (5)

Appendix 1 is the process to formed the equations to calculated the coordinates in XY coordinate system from the coordinates in $x'y'$ coordinate system using the equation (5), the formed equations are:

$$X = -y' + x_O \tag{6}$$

$$Y = -x' + \ y_O \tag{7}$$

From the formulas (6)(7), the coordinate values on the SVG file are transformed into values that correspond to the coordinate system of the robot. The origin O' of the canvas is assigned to point $(300{,}100)$ in the robot coordinate system. The value of x_o and y_o is then determined to support the coordinate transformation. The point $(300{,}100)$ is chosen to be the origin of the canvas coordinate system. This is because the space for the SVG image generated by that origin has the appropriate dimension and the reach of the robot covers it completely.

3.3.3 Linear movement

In the linear movement commands, the parameters in the G-code lines are identical to the parameters required in the robot control functions. The parameters are the coordinate values including the X- and Y- values of the destination position of the movement. In most of the other cases, the parameters in the g-code commands are different from the parameters in the robot control functions. The program has the ability to convert the parameters into the suitable form. In the linear speed setting function, the developer can control not only the speed but also the acceleration of the robot arm. The F-value in the G-code generated in the project is F-value in mm/min unit. The unit of the linear speed setting function of the robot has the same unit. Therefore, the velocity parameters is set as the same as the F-vaue in G-code command. The acceleration is decided to be equivalent vale of the velocity parameters.

3.3.4 Arc movement

Arc command parameters in G2 and G3 commands are coordinates of the destination point and the center point of the circular path of the movement. The required parameters of the robot control function are the destination point and the circular point, which is a random point on the circular path. The “G2G3_to_circularpoint” function was programmed to convert the center point coordinates into the circular point coordinates. First, the center point of the arc path is defined based on the end point and the I and J parameters given in the G-code arc command. As the I and the J value is the offset between the ending point $B(x_B, y_B)$ and the center point in X and Y ordinates respectively, the coordinates of the center point is

$$I(h, k) = (x_B - I, y_B - J) \tag{8}$$

Then, the radius r of the arc path is the distance between center point $I(h, k)$ and the ending point $B(x_B, y_B)$

$$r = \sqrt{(x_B - h)^2 + (y_B - k)^2} \tag{9}$$

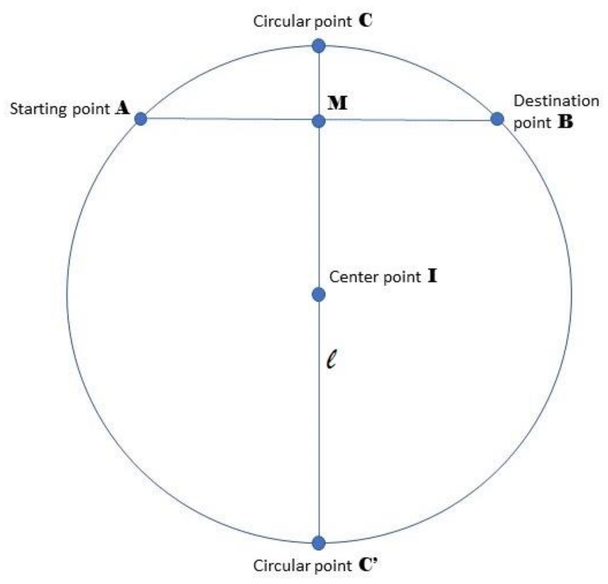


Figure 8. The illustration of related points in the arc movement command

The figure 8 illustrates the relevant points in the circle of the arc command. The line goes through the points $A(x_A, y_A)$ and $B(x_B, y_B)$ in figure 6 has the slope calculation created based on the equation (3).

The M point is defined as the midpoint of the line AB. The extension of the line IM meets the circle at C and C'. These 2 points are possible circular points determined as 2 symmetric points on the circle and the CC' line connects the center point I and midpoint M. The linear equation of the line l through the points I, M, C, C' is written based on the equation (1).

With the line IM being the altitude of the side AB in the isosceles triangle ΔABI, the line l is perpendicular with the line AB at point M. Then the slope of AB is the negative reciprocal of the slope of l. Applying the equation (3), we have

$$a = \frac{-1}{m} = \frac{x_B - x_A}{y_A - y_B} \tag{10}$$

As the line l go through the point $I(h,k)$, $x = h$, $y = k$ and $a = \frac{x_B - x_A}{y_A - y_B}$ are substituted in the equation $y = ax + b$:

$$k = \frac{x_B - x_A}{y_A - y_B}.h + b \tag{11}$$

⇔

$$b = k - \frac{x_B - x_A}{y_A - y_B}.h \tag{12}$$

The points C and C' lie on the arc which has center $I(h,k)$ and radius r, the circle equation of the arc is formed based on equation (2)

$$(x-h)^2 + (y-k)^2 = r^2 \tag{13}$$

The circular points are 2 points that have their coordinate values satisfying the linear equation of the line l with a in equation (10), b in equation (12) and the circle equations(13). After solving the simultaneous equation, the coordinates of 2 circular points are determined.

$$x = \frac{-ab + ak + h \pm \sqrt{a^2(r^2 - h^2) + b(2k - 2ah) + 2ahk - b^2 - k^2 + r^2}}{a^2 + 1} \tag{14}$$

$$y = ax + b \tag{15}$$

Where a and b are defined in respectively in equation (10) and (12).

For each value of x determined in (14), there is a corresponding y-value calculated by (15). Therefore, there are 2 pairs of x- y value for 2 possible circular points that are required to be found.

Finally, the program selects the suitable circular point for the G2/G3 command by the "Clockwise_check" function. The method for analyzing the slope of the line segment created by 3 points (starting point, circular point and ending point) mentioned in section 2.8 was applied. One of the 2 circular points is put into 3-point-group to check if the orientation is clockwise or counterclockwise. If it is equivalent with the required movement orientation, the circular point is loaded to the circular point parameter of the arc movement control function. Otherwise, the coordinates of the other circular point are loaded instead.

3.3.5 Laser head control

The laser head is connected to a PWM (Pulse Width Modulation) port and a digital port on the robot. The program commands the robot to export the required output to control the laser head. The output of the digital port controls the status ON/OFF of the laser head, and the PWM port controls the power of it. When M5 commands are detected in the G-code file, the program dictates the robot to output 0 at the digital port to turn the laser head off. With the M3 command, the digital output is set to 1. Additionally, the power value identified in the M3 G-code line is processed to control the power the laser head. The power value in the M3 command is S-value, which is in the range of 0 to 255. The duty cycle of the PWM used to control the power of the laser is set in the range of 0-100. As a result, the power value of the PWM output is calculated as below:

$$S' = S \times \frac{100}{255} \tag{16}$$

The value S', after having been determined, is applied as the duty cycle parameter of PWM to control the power of the laser head. The M3 and M5 commands are not the "queue" action, which means the robot can execute the other tasks when the control of the laser power is maintained. It helps the robot move the arm with the powered on laser head in the laser operation.

3.4 Template of the main program

The template of the main program was developed based on the demo program in “Dobot demo for python” package, which is provided by Shenzhen Yuejiang Technology Co., LTD. The demo program instructs developer to import the required library, load the DLL, establish computer-robot connection, clear the command queue, calibrate the robot and set up a new command queue. All these procedures must be initiated in the mentioned order in the main program. The commands interpreted in the reading g-code process are put in the generated command queue in the initializing process. Additionally, the basic parameters of the robot, consisting of home position and joint speed, can be set in the demo program as well as the initial section of the main program.

3.5 Technical setup

The SVG file requires the most effort in the set up phase. The size of the file must not exceed the working volume of the robot arm. To ensure this requirement, the SVG image was copied and edited in the 200mm x 100mm canvas in Inkscape software. The display unit was set to mm for synchronization. All the objects the created SVG file must be paths otherwise the program cannot export the equivalent G-code file. The image should be saved in SVG format after editing.

The Dobot Magician was set up and wired as instructed in the manual. The power supply cable was connected from the electricity socket to the DC coaxial connector on the robot. The USB-B port of the robot is connected to the USB port on the computer. When the program runs, the signal connection between the robot and computer is established.

For the movement test, the pen provided by the manufacturer was used as the end effector of the robot. The pen was attached and fixed to the robot arm by a tightening screw. The Z-ordinate of the robot when the pen tip touches the drawing platform was recorded. This value (in mm) is the Z value that should be inserted in the initial setting when the program starts to run.

For the final test, the laser head was used to test the movement controlling of the program as well as the powering of the laser controlling function. The laser head in use is provided by the manufacturer, so that the attaching is similar to the pen. The laser head has 2 connectors that need to be connected to a PWM port and a Digital port on the robot. The laser head is controlled by the robot through these ports. The vertical distance between the laser head and the operating platform should equal to the focal length of the laser beam. To define this distance, the Z-axis of the robot is adjusted until the laser beam converges at the operating platform. Similar to the pen processing, the Z-axis value needs to be recorded and input as an initial setting at the beginning of the process.

3.6 Result

This section evaluates the ability of generating G-code from the SVG vector graphic file and the performance of the robot in executing the laser operation. The G-code generated by the SVG-to-gcode module is imported to a G-code simulator called Webgcode developed by Nicolas Raynaud in 2020 to evaluate. The paths in the SVG file are approximated into several line segments. With the default tolerance designated in the module, the difference between the output G-code command and the input SVG image is invisible. The direction of the image in the G-code simulator is upside down due to the inversion of the canvas coordinate system compared to the simulator coordinate system. The amount of the line segments can be adjusted corresponding to the “error cap” value in the tolerance section of the SVG-to-gcode module. The G-code is repeated by the amount of times that equals the number of passes that user entered to the program.

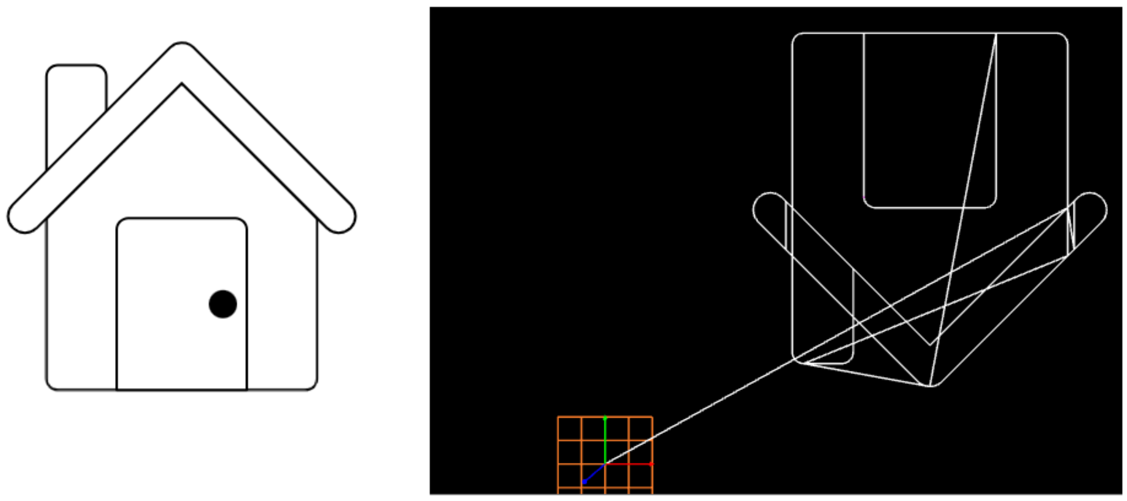


Figure 9. SVG image (left) and the output G-code in the simulator (right)

The purpose of the set up mentioned in section 3.6 is giving an overview about the laser operating ability of the robot. In the set up with the pen, the drawing result of the pen controlled by the robot is acceptable.

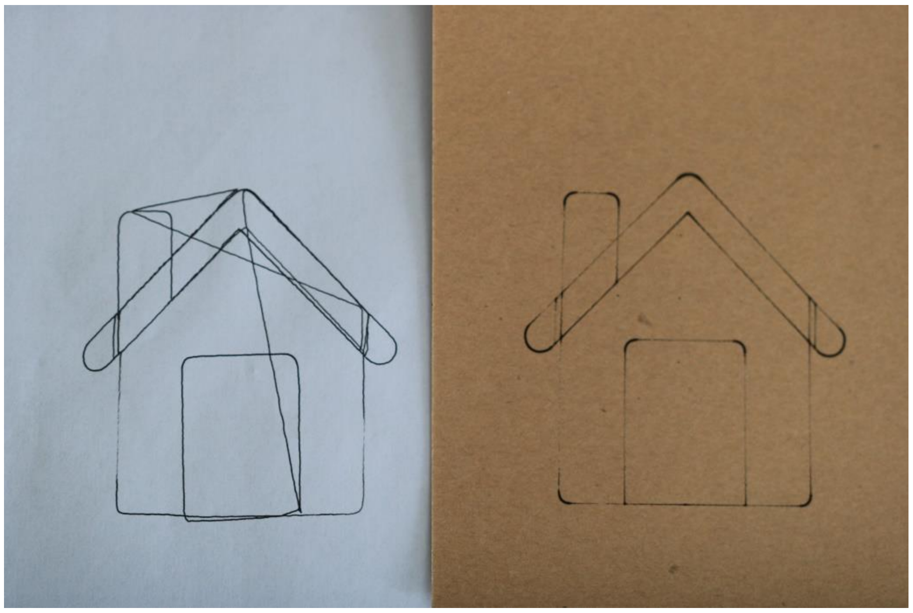


Figure 10. The result of the pen operation (left) and laser engraving operation (right)

The figure 10 shows the result of the laser process operated by the robot. While in figure 10 (left) the end effector is the pen, in figure 10 (right), the laser head was used as the end effector on the robot. As shown in the figure 10 (left), there is vibration in the drawing line due to the unstability of the robot arm when the pen tip contact with the surface, epscially in the curve section. However, it is insignificant when observe from distance. Then, the laser head was replaced by the laser head to evaluate the result of laser operation. As shown in figure 10 (right), the drawing line is accurate to the SVG image, the vibration in the drawing line was disappeared since the robot arm moved efforlessly in the air. The thickness of the drawing is uneven. The curve sections which approximated by more line segments have more thickness in the drawing line.

4 CONCLUSION

The project is successful in developing a system that control Dobot Magician execute laser process the SVG file as the input. The result was evaluated as acceptable performance by an educational robot. Additionally, the system have the ability to read the laser processing G-code file to control the Dobot Magician according to the G-code commands. However, there are some aspects that considered as limitations of the developed system. There are some disadvantages of the library which is functioned as the tool to convert the SVG file into the G-code commands. The library has the ability to parsing and compiling the image only in the SVG format, the capability to inteprete different formats of vector graphic content is one of its deficiency . The library applies the method that uses the line segments to approximate all path in the SVG file. This method gives the function ability to convert the complex curves into the straight lines, which supported by G-code commands, without loosing visible details of the curves. On the other hand, this method restricts the usage of the arc commands in the G-code system and makes the function cannot process any object in the SVG file but paths. The function to control the robot execute the G-code commands only support the G-code for the laser processing, which are G0, G1, G2, G3, M3 and M5 commands.

The recommendations for further research in the future is to alleviate the limitations mentioned above. The ability to operate other formats of vector graphic content is necessary in improving the system. The function to inteprete more types of object in the vector graphic file is demanding, and the usage of the arc movement needs to be increased to reduce the operating time of the robot. The function that transforms G-code commands to robot action need further development to process the universal standard G-code. Finally, the GUI aspect of the software is a possibility to progress. Viewing and editing tools for the vector graphic content built-in is a significant improvement. A pre-view window of the laser operating process increases the ability of the system in educational purpose due to the visualization of the process.

The application of the project in increasing the educational ability of Dobot Magician is significant. The education about G-code and laser processing can be take responsibility by Dobot Magician when the work of this thesis applied. Additionally, the educational robot programming process was researched and applied in the thesis, and these works can open the possibility of the further research in educational robot.

References

Agrawal, R. (2021, Sep 1). *Orientation of 3 ordered points*. Retrieved from Geeks for geeks: https://www.geeksforgeeks.org/orientation-3-ordered-points/

Alexander, S. G. (2021, July 21). *What is Object Oriented Programing?* Retrieved from TechTarget Web site: https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP

Boljanovic, V. (2016). *Applied Mathematical and Physical Formulas* (Second ed.). Connecticut: Industrial Press, Inc. Retrieved April 1st, 2022

Croft, A., & Davison, R. (2019). *Mathematics For Engineers* (Fifth ed.). Harlow: Pearson Education Limited. Retrieved April 15th, 2022

Dahlström, E., Dengler, P., Grasso, A., Lilley, C., McCormack, C., Schepers, D., & Watt, J. (2011, August 16). *Scaleable Vector Graphics (SVG) 1.1 (Second Edition).* Retrieved from W3.org: https://www.w3.org/TR/SVG11/

DOBOT Magician - Lightweight Intelligent Training Robotic Arm - An all-in-one STEAM Education Platform. (n.d.). Retrieved from Dobot Official Web site: https://www.dobot.cc/dobot-magician/product-overview.html

Hess, B. (2017, May 22). *What Is CNC Machining? A Comprehensive Guide.* Retrieved from Astro Machine Works: https://astromachineworks.com/what-is-cnc-machining/

Lutkevich, B. (2021, June). *What are Vector Graphic? Vector Art Explained*. Retrieved from techtarget.com: https://www.techtarget.com/whatis/definition/vector-graphics

Nilsson, D. (2016). *G-Code to RAPID translator for Robot-Studio.* Trollhättan, SWEDEN: University West, Department of Engineering Science.

Padula, A. (2021, January 27). *PadLex/SvgToGcode.* Retrieved from Github: github.com/PadLex/SvgToGcode

Quin, L. (2016, October 11). *Extensible Markup Language.* Retrieved from W3C: w3.org/XML/

Shenzhen Yuejiang Technology Co., L. (2018, November 06). Dobot Magician API Description. Shenzhen, China.

Appendix 1: TRANSFORMATION FORMULAS RECONSTRUCTION

$$\begin{bmatrix} X \\ Y \end{bmatrix} = T^{-1} \begin{bmatrix} x' \\ -y' \end{bmatrix} + \begin{bmatrix} x_o \\ y_o \end{bmatrix} \quad (5)$$

$$\Leftrightarrow \quad \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} cos\frac{-\pi}{2} & sin\frac{-\pi}{2} \\ -sin\frac{-\pi}{2} & cos\frac{-\pi}{2} \end{bmatrix}^{-1} \begin{bmatrix} x' \\ -y' \end{bmatrix} + \begin{bmatrix} x_o \\ y_o \end{bmatrix}$$

$$\Leftrightarrow \quad \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} x' \\ -y' \end{bmatrix} + \begin{bmatrix} x_o \\ y_o \end{bmatrix}$$

$$\Leftrightarrow \quad \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} x' \\ -y' \end{bmatrix} + \begin{bmatrix} x_o \\ y_o \end{bmatrix}$$

$$\Leftrightarrow \quad X = -y' + x_O \quad (6)$$

$$Y = -x' + y \quad (7)$$

Appendix 2: MAIN PROGRAM

```
from svg_to_gcode.svg_parser import parse_file
from svg_to_gcode.compiler import Compiler, interfaces

from read_gcode import*

def main():
########################SET UP GCODE FILE##################################
    #Create window to select svg file
    root = tk.Tk()
    root.withdraw()
    file_path = filedialog.askopenfilename()
    print(file_path)

    """Input the setting parameters of the operation
        - movement speed: speed of the laser head when not operating
        - operate speed: speed of the laser head when operating
        - Number of passes
        - Depth of pass controls how far the tool moves in Z-axis after every
pass"""
    movement_speed = int(input('Movement speed (mm/min): '))
    cutting_speed = int(input('Operate speed (mm/min): '))
    # depth = int(input('Depth of pass: '))
    passes = int(input('Number of passes: '))

    #input Z value for the operation
    Z = float(input('Input initial operating height in mm: '))

    gcode_compiler = Compiler(interfaces.Gcode, movement_speed=movement_speed,
cutting_speed=cutting_speed, pass_depth=0)

    curves = parse_file(file_path,canvas_height=100,transform_origin=False) #
Parse an svg file into geometric curves
    gcode_compiler.append_curves(curves) #Compile the curve to Gcode file
    gcode_compiler.compile_to_file("C:/Users/Welcome/Desktop/thesis/test
unit/new.gcode", passes=passes)

    #create tuple contains g-code lines from the Gcode file
    gcode = open(os.path.join("C:/Users/Welcome/Desktop/thesis/test
unit/new.gcode"),'r')
    a = [x[:-1] for x in gcode.read().splitlines()]
    tuple_of_lines = tuple(a)

########################EXECUTE MOVEMENT COMMAND##############################
    CON_STR = {
        dType.DobotConnect.DobotConnect_NoError:  "DobotConnect_NoError",
        dType.DobotConnect.DobotConnect_NotFound: "DobotConnect_NotFound",
        dType.DobotConnect.DobotConnect_Occupied: "DobotConnect_Occupied"}

    #Load Dll
    api = dType.load()

    #Connect Dobot
    state = dType.ConnectDobot(api, "", 115200)[0]
    print("Connect status:",CON_STR[state])

    if (state == dType.DobotConnect.DobotConnect_NoError):

        #Clean Command Queued
        dType.SetQueuedCmdClear(api)

        #Async Motion Params Setting
        dType.SetHOMEParams(api,250,0,50,0, isQueued = 1)
        dType.SetPTPJointParams(api,200,200,200,200,200,200,200,200, isQueued=1)
        dType.SetPTPCommonParams(api,100,100, isQueued = 1)

        #Async Home
        dType.SetHOMECmd(api, temp = 0, isQueued = 1)

        #Process and execute Gcode
        Gcode_execute(tuple_of_lines,canvas_origin=(300,100), Z=Z)

        #Move back to Home
        dType.SetPTPCmdEx(api,1,250,0,50,0, isQueued=1)

    #Disconnect Dobot
    dType.DisconnectDobot(api)

if __name__=='__main__':
    main()
```

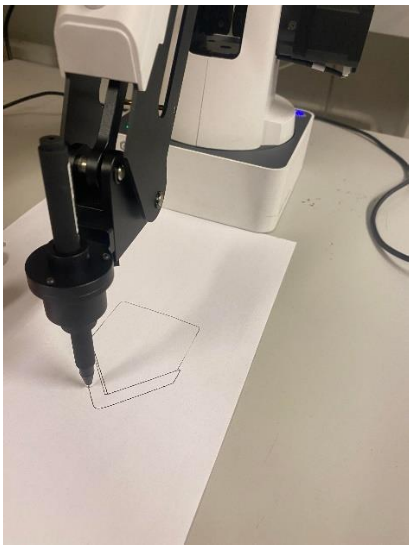
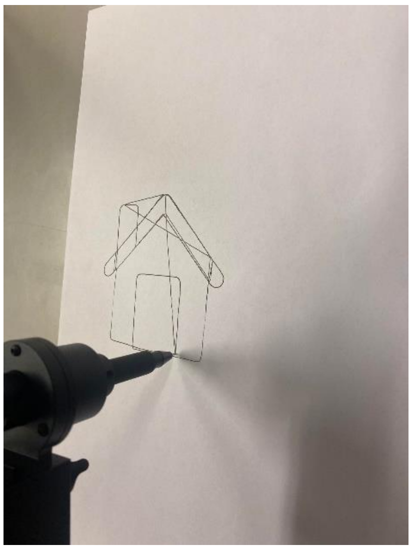
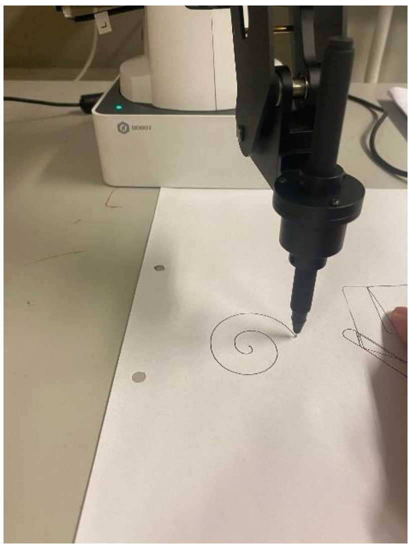
Appendix 3: LINE SEGMENT CHAIN FUNCTION

```
def line_segment_approximation(shape, increment_growth=11 / 10, error_cap=None,
error_floor=None)\
            -> "LineSegmentChain":
        """
        This method approximates any shape using straight line segments.

        :param shape: The shape to be approximated.
        :param increment_growth: the scale by which line_segments grow and
shrink. Must be > 1.
        :param error_cap: the maximum acceptable deviation from the curve.
        :param error_floor: the maximum minimum deviation from the curve before
segment length starts growing again.
        :return: A LineSegmentChain which approximates the given shape.
        """

        error_cap = TOLERANCES['approximation'] if error_cap is None else
error_cap
        error_floor = (increment_growth - 1) * error_cap if error_floor is None
else error_floor

        if error_cap <= 0:
            raise ValueError(f"This algorithm is approximate. error_cap must be
a non-zero positive float. Not {error_cap}")

        if increment_growth <= 1:
            raise ValueError(f"increment_growth must be > 1. Not
{increment_growth}")

        lines = LineSegmentChain()

        if isinstance(shape, Line):
            lines.append(shape)
            return lines

        t = 0
        line_start = shape.start
        increment = 5

        while t < 1:
            new_t = t + increment

            if new_t > 1:
                new_t = 1

            line_end = shape.point(new_t)
            line = Line(line_start, line_end)

            distance = Curve.max_distance(shape, line, t_range1=(t, new_t))

            # If the error is too high, reduce increment and restart cycle
            if distance > error_cap:
                increment /= increment_growth
                continue

            # If the error is very low, increase increment but DO NOT restart
cycle.
            if distance < error_floor:
                increment *= increment_growth
            lines.append(line)

            line_start = line_end
            t = new_t

        return lines
```

Appendix 4: READ G-CODE FUNCTION

```python
import re
import sys
import os
import threading
from G2G3_to_CircularPoint import*

import tkinter as tk
from tkinter import filedialog

import DobotDllType as dType

api = dType.load()

def line2param(line_gcode):
    """This function captures the parameter values in a line of Gcode command
    The input of function is the text form of a Gcode line
    The function outputs the parameters and their values in the dictionary
form"""

    param_list = list(line_gcode.split(" "))

    for item in param_list: #capture G-command
        if all(item[0] != 'G' for item in param_list):
            G = None
        else:
            for i in range(len(param_list)):
               if param_list[i][0] == 'G':
                    G = param_list[i]

    for item in param_list: #capture M-command
        if all(item[0] != 'M' for item in param_list):
            M = None
        else:
            for i in range(len(param_list)):
               if param_list[i][0] == 'M':
                    M = param_list[i]

    for item in param_list: #capture F-command
        if all(item[0] != 'F' for item in param_list):
            F = None
        else:
            for i in range(len(param_list)):
               if param_list[i][0] == 'F':
                    F = param_list[i][1:]

    for item in param_list: #capture S-command
        if all(item[0] != 'S' for item in param_list):
            S = None
        else:
            for i in range(len(param_list)):
               if param_list[i][0] == 'S':
                    S = param_list[i][1:]

    for item in param_list: #capture X
        if all(item[0] != 'X' for item in param_list):
            X = None
        else:
            for i in range(len(param_list)):
                if param_list[i][0] == 'X':
                     X = param_list[i][1:]

    for item in param_list: #capture Y
        if all(item[0] != 'Y' for item in param_list):
            Y = None
        else:
            for i in range(len(param_list)):
               if param_list[i][0] == 'Y':
                    Y = param_list[i][1:]

    for item in param_list: #capture Z
        if all(item[0] != 'Z' for item in param_list):
            Z = None
        else:
            for i in range(len(param_list)):
               if param_list[i][0] == 'Z':
                    Z = param_list[i][1:]

    for item in param_list: #capture I
        if all(item[0] != 'I' for item in param_list):
            I = None
        else:
            for i in range(len(param_list)):
               if param_list[i][0] == 'I':
                    I = param_list[i][1:]

    for item in param_list: #capture J
        if all(item[0] != 'J' for item in param_list):
            J = None
        else:
            for i in range(len(param_list)):
               if param_list[i][0] == 'J':
                    J = param_list[i][1:]

    return {
        'G': G,
        'M': M,
        'X': X,
        'Y': Y,
        'Z': Z,
        'I': I,
        'J': J,
        'F': F,
        'S': S
        }

def XYcoordinate_transform(X_canvas,Y_canvas,canvas_origin):
    """The function transforms the canvas coordinate of the SVG file to the
robot coordinate
       Input: XY_canvas is the coordinate values needed to be transform,
              canvas point (Xo,Yo) is the coordinate of the canvas origin on the
robot coordinate system
       The function return the transformed coordinate"""

    X_robot = -Y_canvas + canvas_origin[0]
    Y_robot = -X_canvas + canvas_origin[1]
    return (X_robot,Y_robot)

def G1_operation(X,Y,Z):#define actions for G1 command
    current_pose = dType.GetPose(api)
    dType.SetPTPCmdEx(api, 2, X,  Y,  Z, current_pose[3], 1)
    print('run G1 X',X,'Y',Y)

def G0_operation(X,Y,Z):#define actions for G0 command
    current_pose = dType.GetPose(api)
    dType.SetPTPCmdEx(api, 1, X,  Y,  Z, current_pose[3], 1)
    print('run G0 X', X, 'Y', Y)

def G2_operation(X,Y,Z,I,J):#define actions for G2 command
    current_pose = dType.GetPose(api)
    r = current_pose[3]
    #find the middle point in the Arc
    circular_point =
find_circular_point((current_pose[0],current_pose[1]),(X,Y),(I,J),G23=True)
    #Execute Arc command
    dType.SetARCCmd(api,(circular_point[0],circular_point[1],Z,r),(X,Y,Z,r),1)

def G3_operation(X,Y,Z,I,J):#define actions for G3 command
    current_pose = dType.GetPose(api)
    r = current_pose[3]
    # find the middle point in the Arc
    circular_point = find_circular_point((current_pose[0], current_pose[1]), (X,
Y), (I, J), G23=False)
    #Execute Arc command
    dType.SetARCCmd(api, (circular_point[0], circular_point[1], Z, r), (X, Y, Z,
r), 1)

def M3_operation(S):#turn laser on and set power for M3 command
    dType.SetEndEffectorLaserEx(api,enableCtrl=1,power=500,isQueued=0)
    print('Turn laser ON')

def M5_operation():#turn laser off for M5 command
    dType.SetEndEffectorLaserEx(api,enableCtrl=0,power=0,isQueued=0)
    print('Turn laser OFF')

def linear_speed_setting(F):#setting speed of linear travel
    dType.SetPTPCoordinateParamsEx(api, F, F, F, F, 1)
    print("Setting speed",F)

def circular_speed_setting(F):#setting speed of circular travel
    print("Setting speed",F)

def wait():
    pass

"""Read Gcode testing"""
# root = tk.Tk()
# root.withdraw()
# file_path = filedialog.askopenfilename()
# gcode = open(os.path.join(file_path),'r')
# a = [x[:-1] for x in gcode.read().splitlines()]
# tuple_of_lines = tuple(a)
# Z = float(input('Input height in mm: '))

def Gcode_execute(tuple_of_lines, canvas_origin, Z):
    """This function evaluate the parameters of the gcode file line by line
    Input of the function is the tuple of Gcode whereas each item in the tuple
is a Gcode command line"""
    for i in range(len(tuple_of_lines)):
        # print(line2param(tuple_of_lines[i]))
        #G1 command line
        if line2param(tuple_of_lines[i])['G']=='G1' or
line2param(tuple_of_lines[i])['G']=='G01' :

            if line2param(tuple_of_lines[i])['F']:
                F = float(line2param(tuple_of_lines[i])['F'])
                linear_speed_setting(F)

            if line2param(tuple_of_lines[i])['X'] and
line2param(tuple_of_lines[i])['Y']:
                X = float(line2param(tuple_of_lines[i])['X'])
                Y = float(line2param(tuple_of_lines[i])['Y'])
                (X,Y) = XYcoordinate_transform(X,Y,canvas_origin)
                G1_operation(X, Y, Z)


        #G0 command line
        elif line2param(tuple_of_lines[i])['G']=='G0' or
line2param(tuple_of_lines[i])['G']=='G00' :

            if line2param(tuple_of_lines[i])['F']:
                F = float(line2param(tuple_of_lines[i])['F'])
                linear_speed_setting(F)

            if line2param(tuple_of_lines[i])['X'] and
line2param(tuple_of_lines[i])['Y']:
                X = float(line2param(tuple_of_lines[i])['X'])
                Y = float(line2param(tuple_of_lines[i])['Y'])
                (X, Y) = XYcoordinate_transform(X, Y, canvas_origin)
                G0_operation(X, Y, Z)

        #G2 command line
        elif line2param(tuple_of_lines[i])['G']=='G2' or
line2param(tuple_of_lines[i])['G']=='G02' :

            if line2param(tuple_of_lines[i])['F']:
                F = float(line2param(tuple_of_lines[i])['F'])
                circular_speed_setting(F)

            if line2param(tuple_of_lines[i])['X']:
                X = float(line2param(tuple_of_lines[i])['X'])
                Y = float(line2param(tuple_of_lines[i])['Y'])
                (X, Y) = XYcoordinate_transform(X, Y, canvas_origin)
                I = float(line2param(tuple_of_lines[i])['I'])
                J = float(line2param(tuple_of_lines[i])['J'])
                (I, J) = XYcoordinate_transform(I, J, canvas_origin)
                G2_operation(X,Y,Z,I,J)

        #G3 command line
        elif line2param(tuple_of_lines[i])['G']=='G3' or
line2param(tuple_of_lines[i])['G']=='G03' :

            if line2param(tuple_of_lines[i])['F']:
                F = float(line2param(tuple_of_lines[i])['F'])
                circular_speed_setting(F)

            if line2param(tuple_of_lines[i])['X']:
                X = float(line2param(tuple_of_lines[i])['X'])
                Y = float(line2param(tuple_of_lines[i])['Y'])
                (X, Y) = XYcoordinate_transform(X, Y, canvas_origin)
                I = float(line2param(tuple_of_lines[i])['I'])
                J = float(line2param(tuple_of_lines[i])['J'])
                (I, J) = XYcoordinate_transform(I, J, canvas_origin)
                G2_operation(X, Y, Z, I, J)

        #M3 command line
        elif line2param(tuple_of_lines[i])['M']=='M3' or
line2param(tuple_of_lines[i])['M']=='M03' or
line2param(tuple_of_lines[i])['M']=='M106':
            S = float(line2param(tuple_of_lines[i])['S']) * (101/256)
            M3_operation(S)

        #M5 command line
        elif line2param(tuple_of_lines[i])['M']=='M5' or
line2param(tuple_of_lines[i])['M']=='M05' or
line2param(tuple_of_lines[i])['M']=='M107' :
            M5_operation()
```

Appendix 5: PEN OPERATION PICTURES

Appendix 6: LASER OPERATION PICTURES