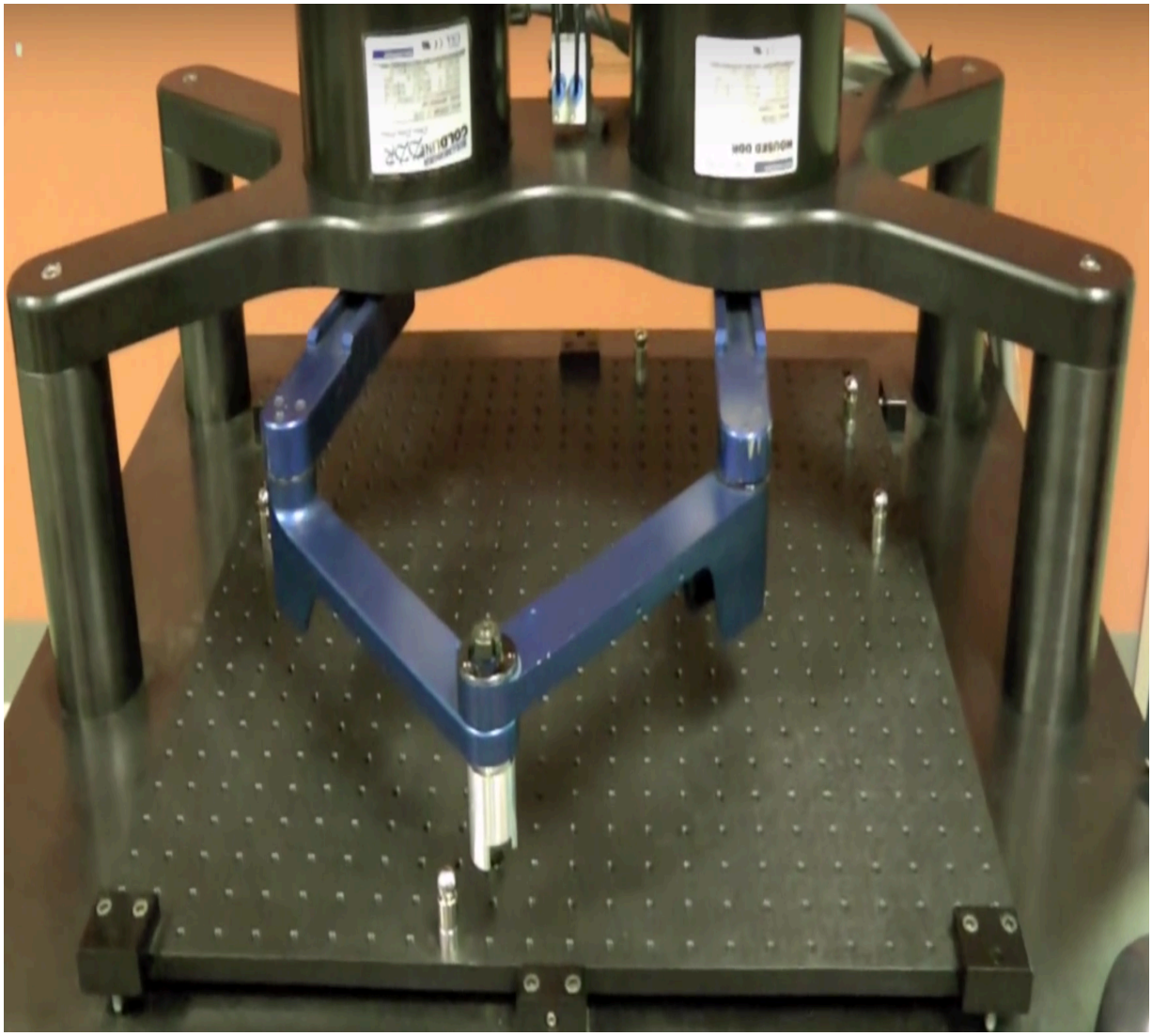


SCARA Arm Report

Student Name: Minping Yang (Rock)
Student ID:300364234
Team Number: Tuesday 11:10-13:00

Table of Contents

Abstract:

On 13th September 2016, our team came to a consensus on the top 3 goals that our robot needs to achieve. In the first meeting, we discussed about what is our eventual target, how to achieve the milestones ahead, and how can we continue to work effectively as a team. In addition, manageable workload was allocated for every team member. The team completed the hardware tasks and got 95% marks. This aim of this report is to illustrate what we have achieved and how the team can further improve. Furthermore, this project gives us a chance to understand the challenges we might face and gain valuable skills where we can contribute to the team in the future.

1. Introduction:

1.1. Motivation:

There are several motivating factors for the team to work together towards a common goal in this project:

1) Develop communication skills by working effectively as a team.
2) Develop coding skills through practical experience with programming a simulation control system before deployment on the hardware arm.
3) Manage time and the process of project to achieve the aims.
4) Development of good quality software and hardware.

1.2 Scope:

The scope of this project was to cover theories that were learnt from other papers besides ENGR110. For example, direct and inverse kinematic equations, implementation of singularity avoidance, and graphic skills were applied for developing the robot..
However, there also are several limitations to this project:

- Limited time as the project has to be completed in 3 main phases with the corresponding week: Derive the appropriate direct and inverse kinematic equations for the robot (16th September), Implement the equations using the framework java simulation for the arm that has been provided (23th September) and develop the code from simulation onto hardware (30th September).
- Other than help from the lecturers and tutors, outsourcing was not allowed.

1.3 Aims:

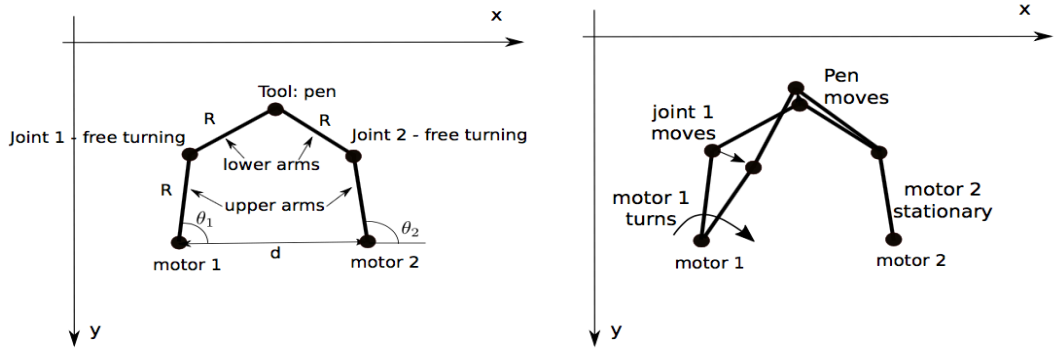
There are three main aims for the project:

1. Creating a functional, accurate and vivid java simulation
2. Ensuring the pen only moved within the expected range of motion of the arms.
3. Ensuring simulated drawing and actual product are as closely matched as possible.

2. Background:

This project uses SCARA arm (selective compliance assembly robot arm), implementation of singularity avoidance, direct and inverse kinematic equation and implementation of kinematic equations. This section will give a background to these scientific methods and technologies and show how they have been used to build, program and test the robot.

2.1 SCARA arm



(a) Five-bar mechanism (b) Motion driven by one arm

Figure 2

(Eldridge, ENGR110 Control Systems: A Robotic Arm, 2016)

SCARA arm (selective compliance assembly robot arm) is the most important component of the robot. It is the hand of the robot that can draw accurately based on the PWM (Pulse-Width Modulation) input. However, the robot arms do not function exactly like that of a human being as it has a limited range of motion.

SCARA arm uses a five-bar mechanism where two motors drive two separate upper arms that are connected to lower arms by free-rotating joints. Only one upper arm is moved by the motor at one time while the other stays still. Lower arms are able to freely move. Hence, if one upper arm is turned by the motor, the result will be a movement of the pen (Appendix: Figure 2).

In order for SCARA arm to operate, a model has to be drawn in the java simulation of SCARA arm. Then, the PWM data of the drawing from the java simulation are saved. Finally, PWM data are input to a RPi-Roboshield that controls the motors attached to the upper arms, resulting to the movement of the pen.

2.2 Singularity of SCARA arm

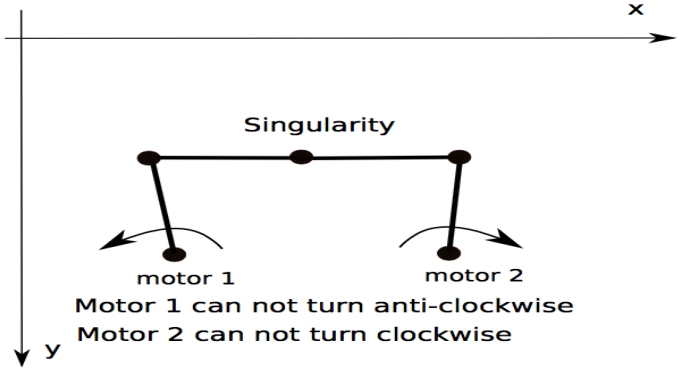


Figure 3: Singularity

(Eldridge, ENGR110 Control Systems: A Robotic Arm, 2016)

This project uses the singularity avoidance technology to combat the singularity issue of the SCARA. Singularity limits the operation of the arms within 180° where the maximum range of motion is reached. This may cause SCARA from overloading its motors and break, as shown in Figure 3, where two opposite movements from both motors compete with each other. Expected points to be drawn are not accessible due to limited range of motion when the pen

exceeds the singularity. Hence, singularity avoidance was employed to ensure SCARRA arm is working within a more logical and accurate range. Meanwhile, drawing in the singularity area was prevented in the java simulation as well.

2.3 Kinematic equations of SCARA arm

- **Direct kinematic equation** is used to calculate the position of the pen given positions of the motors and angles of rotation.
- **Inverse kinematic equation** uses the angle of one motor and the positon of the pen to compute the rotation angle of the other motor.

As SCARA arm drags pen along a series of lines or curves to generate the desired works of art, trail error will not be pleasant. Hence, mathematics in the form of the two main equations as shown above, is applied for finding the relation between the angles and positions.

2.4 Building and using java simulation

Actual outcomes produced by SCARA arm can be predicted by building a java simulation. Moreover, the software model is also an effective way to modify and develop the works we desire. In addition, by using kinematic equations, the java simulation generates the PWM that is read by the SCARA arm since SCARA arm is not able to compute the PWM data by itself.

3. Method:

This section details the methods used to build and program the robot according to the requirements of the project. Section 3.1 highlights explanation and implementation of the direct and inverse kinematic equations. Section 3.2 emphasises the purpose of building software model. Section 3.3 focuses on the implementation of singularity avoidance.

3.1 Direct and inverse kinematic equations

This section clarifies how to compute the values of angles or positions using kinematic equations. Moreover, the implementation of these equations will also be explained.

3.1.1. Explanation of direct kinematic equation

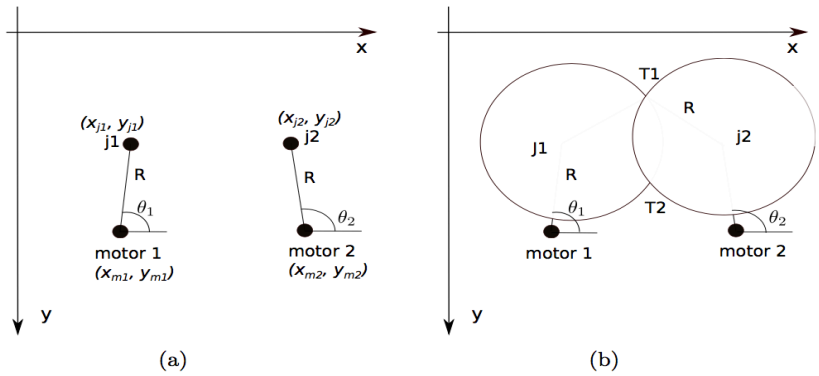


Figure 4: Direct kinematics

(Eldridge, ENGR110 Control Systems: A Robotic Arm, 2016)

The objective of using direct kinematic equation is to compute location of T1 and T2 (Appendix: Figure 4 (b)). Points T1 and T2 represent the only possible positions by recognising another constraint of the system – namely that the lower arms are connected by their tips.

Our general strategy here is as follows:

i. Find expression for the positons of both joints – (x_{j1}, y_{j2}) and (x_{j2}, y_{j2}).

The position of the motors (x_{m1}, y_{m2}) and (x_{m2}, y_{m2}) are known. So are the angles of the upper arms θ_1 , θ_2 and the length of the arms R. From that, the positions of joints j1 and j2 are fixed and can be calculated as:

$$xj1 = xm1 + R \cdot cos(\theta 1)$$
$$yj1 = ym1 + R \cdot sin(\theta 1)$$
$$\rightarrow xj2 = xm2 + R \cdot cos(\theta 2)$$
$$\rightarrow xj2 = xm2 + R \cdot sin(\theta 2)$$

ii. Find the location of the halfway point along the line that pass through both joints.

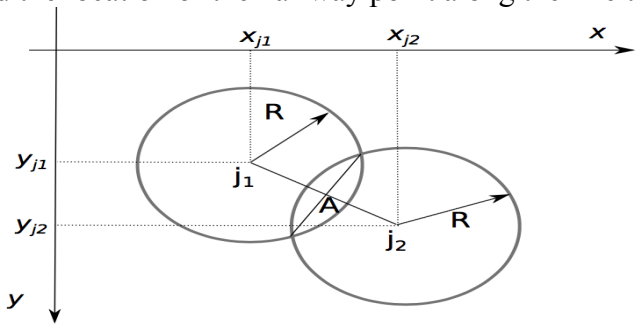


Figure 5: Intersection of two circles

(Eldridge, ENGR110 Control Systems: A Robotic Arm, 2016)

As shown in Appendix: Figure 5, point A is located at the midway point of a line between joints 1 and 2. The x coordinate of A can be found by:

$$\rightarrow x_A = x_{j1} + 0.5 \cdot (x_{j2} - x_{j1})$$
$$\rightarrow y_A = y_{j1} + 0.5 \cdot (x_{j2} - x_{j1})$$

iii. Use trigonometry to find the x and y positons of points T1 and T2

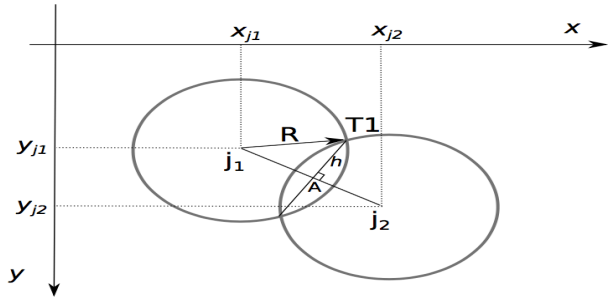


Figure 6: Intersection of two circles

(Eldridge, ENGR110 Control Systems: A Robotic Arm, 2016)

Finally, we can see that point A and point T1 forms part of a right angled triangle (Appendix: Figure 6). Hence, Pythagoras' Theorem can be applied to derive an expression for h in terms of R and the distance between j_1 and j_2

$$\rightarrow h = \sqrt{(R^2 - \left(\frac{yj2-yj1}{2}\right)^2 - \left(\frac{xj2-xj1}{2}\right)^2)}$$

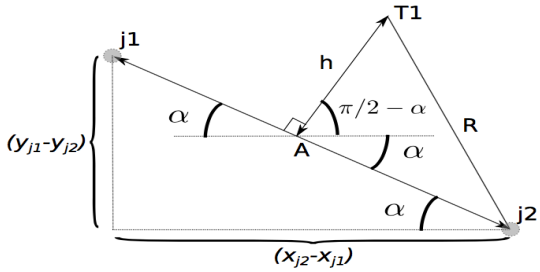


Figure 7

(Eldridge, ENGR110 Control Systems: A Robotic Arm, 2016)

However, only h is known which is not enough to calculate T1 in terms of x and y co-ordinates. The angle α (Appendix: Figure 7) is also needed

$$\alpha = \text{atan}\left(\frac{yj1-yj2}{xj2-xj1}\right)$$

then the final expression for the location of T1 will be derived simply:

$$x_{T1} = x_A + h \cdot \cos(\pi/2 - \alpha)$$
$$y_{T1} = y_A + h \cdot \sin(\pi/2 - \alpha)$$

By repeating the steps above, an expression for both the x and y coordinates of point T2 in terms of x_A, y_A, h and α can be written.

$$x_{T2} = x_A - h \cdot \cos(\pi/2 - \alpha)$$
$$y_{T2} = y_A - h \cdot \sin(\pi/2 - \alpha)$$

3.1.2 Explanation of direct kinematic equation

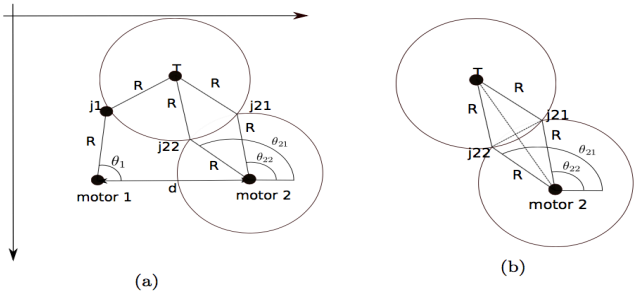


Figure 8

(Eldridge, ENGR110 Control Systems: A Robotic Arm, 2016)

The objective of using inverse kinematic equation is to compute the necessary angles - θ_{21} and θ_{22} of motor 2 (Appendix: Figure 8) from these factors; the position of the motors (x_{m1}, y_{m1}) and (x_{m2}, y_{m2}), the angle of the first motor θ_1 , and the length of the arms R and position T. Furthermore, by using direct kinematics, there are two possible arm configurations for point T, described by joints j_{21} and j_{22}.

By using the points xT and yT, an expression can be derived for the x and y positions of points j_{21} and j_{22}.

$$\rightarrow xj21 = \frac{xT+xM1}{2} + h*cos(\pi/2 - \alpha)$$

$$\rightarrow yj21 = \frac{yT+xM1}{2} + h*sin(\pi/2 - \alpha)$$

$$\rightarrow xj22 = \frac{xT+xM2}{2} - h*cos(\pi/2 - \alpha)$$

$$\rightarrow yj22 = \frac{yT+xM2}{2} - h*sin(\pi/2 - \alpha)$$

A few calculations show the expression of angles - θ_{21} and θ_{22}, from the positions of j_{21} and j_{22} and the motors:

$$\theta 21 = \text{atan2}(yj21 - ym2, xj21 - xm2)$$

$$\theta 22 = \text{atan2}(yj22 - ym2, xj22 - xm2)$$

In this case, either one of the motors can be static to derive the expression for the other.

3.2 Clarifying the implementation of kinematic equation

3.2.1 Implementation of direct kinematic equation

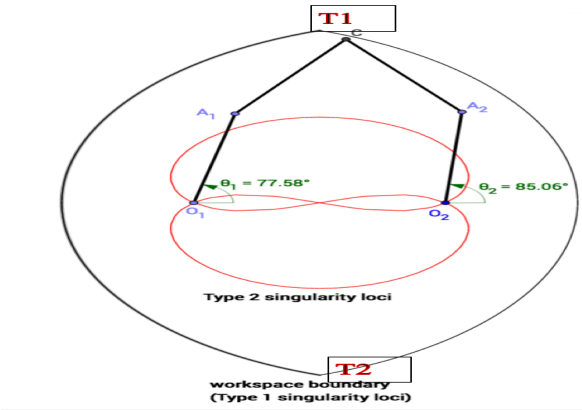


Figure 9: simulation SCARA arm workspace boundary

By using direct kinematic equation, the pen can reach the two critical points T1 and T2 (Appendix: Figure 9) which set the vertexes of the workspace boundary. In the java model, the direct kinematic java method is created to determine the valid positions of the SCARA arm that allows it to draw in an acceptable scope.

3.2.2 Implementation of inverse kinematic equation

Inverse kinematic equation plays an important role in drawing an accurate simulation SCARA arm in the java model. It calculates the two rotation angles (θ_1 and θ_2) and the coordinate positions of elbows of simulation arms (x_{j1}, y_{j1} and x_{j2}, y_{j2}) by using the given positions of motors (x_{m1}, y_{m1} and x_{m2}, y_{m2}) and the length of arm R.

The midpoints (x_{a1}, y_{a1}) and (x_{a2}, y_{a2}) between pen and motor1 and motor2 respectively, should be computed first in order to find the location of elbows.

$$Xa1=Xm1+0.5*(Xt-Xm1)$$
$$Ya1=Ym1+0.5*(Xt-Xm1)$$

$$Xa2=Xm2+0.5*(Xt-Xm2)$$
$$Ya2=Ym2+0.5*(Xt-Xm2)$$

Then, their elbow position can be derived,

$$x_{T1} = x_{a1} + h \cdot \cos(\pi/2 - \alpha)$$
$$y_{T1} = y_{a1} + h \cdot \sin(\pi/2 - \alpha)$$
$$x_{T2} = x_{a2} + h \cdot \cos(\pi/2 - \alpha)$$
$$y_{T2} = y_{a2} + h \cdot \sin(\pi/2 - \alpha)$$

The accuracy of the position of joints determines whether the lengths of the upper arm and lower arm are equal in the java model.

Once the elbow positions were known, the rotation angles of θ1 and θ2 would be computed.

$$\theta 1 = atan2\left(\frac{yj1 - ym1}{xj1 - xm1}\right)$$

$$\theta 2 = \text{atan2}\left(\frac{\text{yj2} - \text{ym2}}{\text{xj2} - \text{xm2}}\right)$$

Since rotation angles can be converted to PWM data which are read by the SCARA arm, inverse kinematic equation is used to calculate the rotation angles.

In addition, once the position of the joints were known, the angle between upper and lower arm - γ_1 and γ_2 for motor1 and motor2 respectively can be derived to avoid singularity. Section 3.1.4 shows how γ_1 and γ_2 function to avoid singularity by the arms.

3.3 The purpose of building software model.

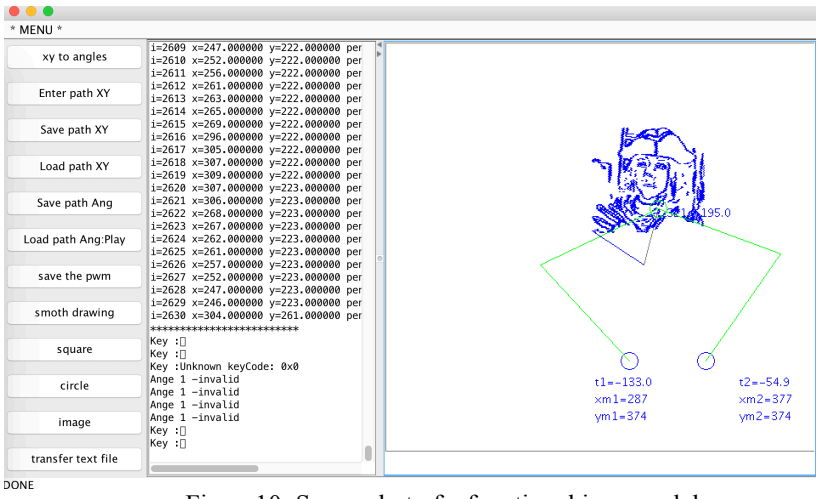
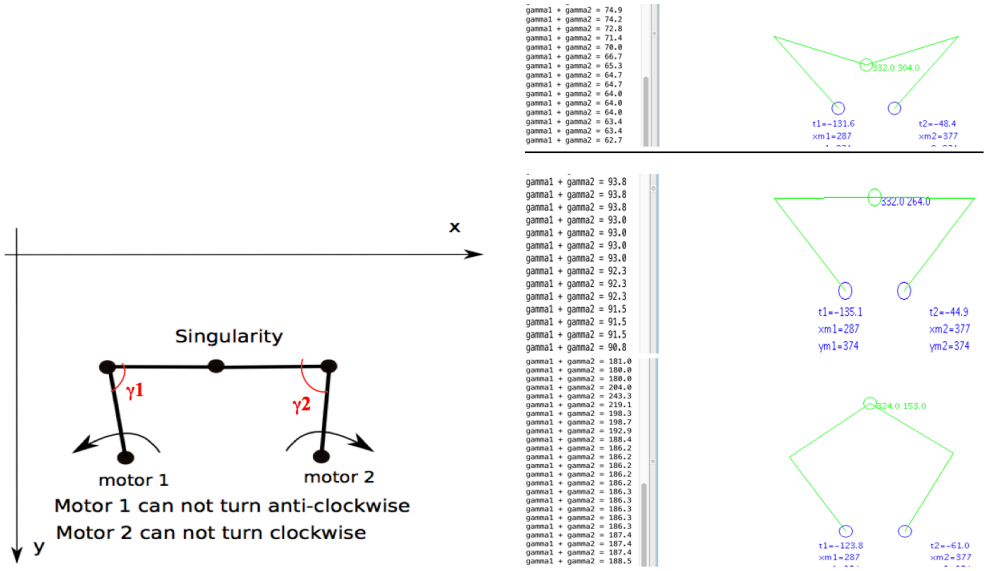


Figure10: Screenshot of a functional java model

A java simulation of SCARA arm (Appendix: Figure 10) is built to improve the efficiency of project development. Firstly, a software model is used to check theoretical outcomes before employment of real SCARA arm, thereby reducing number of trials. Secondly, the java model is used to convert rotation angles into PWM and send the resulting text file to SCARA arm. Furthermore, the drawing can be created in the java drawing board. Finally, programming in software model can perfect the drawing.

3.4 Implementation of singularity avoidance



(a) (b)

Figure 11 Singularity

By doing researches and tests (Figure 11 (b)), it was discovered that the SCARA arm stopped working if the sum of the angles γ1 and γ2 (Figure 11 (a)), between lower and upper arms, at motor1 and motor2 respectively is less than or equal to 90°.

Therefore, trigonometry and inverse kinematic equation are used to calculate γ1 and γ2. Firstly, the coordinates of joints can be found by calculating the distance (MP) between the position of motor and pen using inverse kinematic equations.

$$\rightarrow \mathrm{M1P} = \sqrt{((\mathrm{xj1} - \mathrm{xm1})^2 + (\mathrm{yj1} - \mathrm{ym1})^2)}$$

$$\rightarrow \mathrm{M2P} = \sqrt{((\mathrm{xj2} - \mathrm{xm2})^2 + (\mathrm{yj2} - \mathrm{ym2})^2)}$$

Since the length of upper arm or lower arm is R, then γ1 and γ2 equals:

$$\rightarrow \gamma 1 = \operatorname{acos}\left(\frac{R^2 - M1P^2 - R^2}{-2R^2}\right)$$

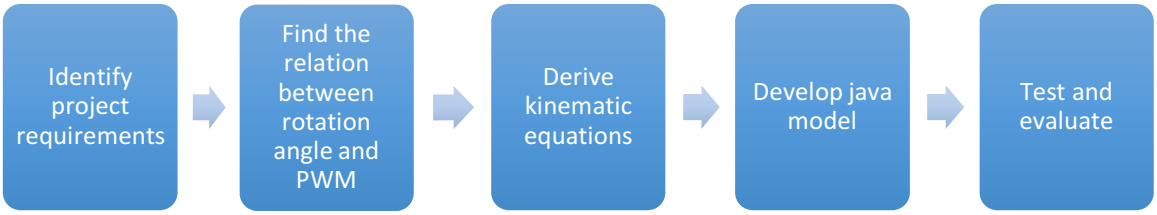
$$\rightarrow \gamma 2 = \operatorname{acos}\left(\frac{R^2 - M2P^2 - R^2}{-2R^2}\right)$$

Finally, the sum of γ1 and γ2 can be calculated. In addition, in the java model, γ1+ γ2>90 can be regarded as a condition to decide if the Boolean variable of *valid_state* is true or not. The code is this:

```
if (γ1+γ2 <= 90) valid_state = false;
```

By testing, singularities are avoided successfully in drawing.

3.5 Process



4. Results:

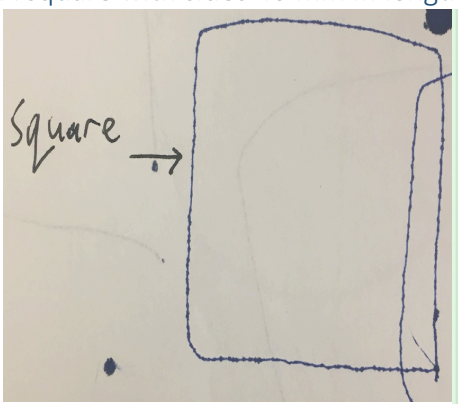
The Results will be structured in the following format, and all researches and code had been already uploaded in SCARA Arm' GitHub (https://github.com/Katiemouse/SCARAArm). The coding for hardware task was done will be displayed and explained in details. Finally, all the parts and features which were specific to the task will be shown and to finish there will be a brief result given on the team.

4.1 Presenting results of hardware tasks

4.1.1 A straight horizontal line is drawn across the page

After modifying the equation to improve its accuracy, a perfectly straight line was drawn in the java simulation using the mouse and the resultant line shown above was drawn by the SCARA arm.

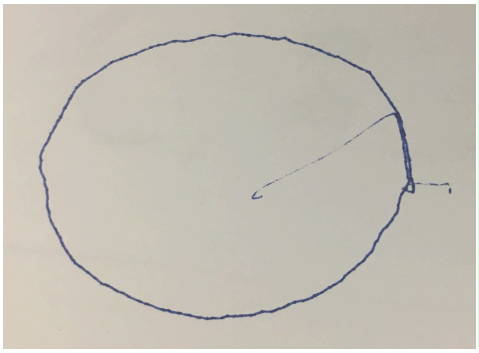
4.1.2 A square with sides 40 mm in length




```
public void doSquare() {
    doMouse("clicked", 267, 147);
    doMouse("clicked", 327, 147);
    doMouse("clicked", 327, 227);
    doMouse("clicked", 267, 207);
    doMouse("clicked", 267, 147);
}
```

The square shown above was drawn by the SCARA arm using the code that was designed to draw a square automatically. By measuring the distance between the actual motors and the distance between the simulation motors, the ratio between pixels and millimetre was obtained to be 1.5. Hence, by drawing a square with 60 pixels on each side, the actual square drawn by the arm would have sides of 40 mm each.

4.1.3 A circle with diameter of 50mm



```
public void doCircle(){
    double centreX = 340;
    double centreY = 140;
    double a = 35.0;
    double b = 35.0;
    double r = 37.5;
    for (double x = centreX-a; x < centreX+a; x+=5) {
        double y = Math.sqrt(b*b*(1-(Math.pow(x-centreX,2)/a/a)))+centreY;
        doMouse("clicked", x, y);
    }
    for (double x = centreX+a; x >= centreX-a; x-=5) {
        double y = -Math.sqrt(b*b*(1-(Math.pow(x-centreX,2)/a/a)))+centreY;
        doMouse("clicked", x, y);
    }
    for (double x = centreX-a; x <= centreX+a; x+=5) {
        double y = Math.sqrt(b*b*(1-(Math.pow(x-centreX,2)/a/a)))+centreY;
        doMouse("clicked", x, y);
    }

}
```

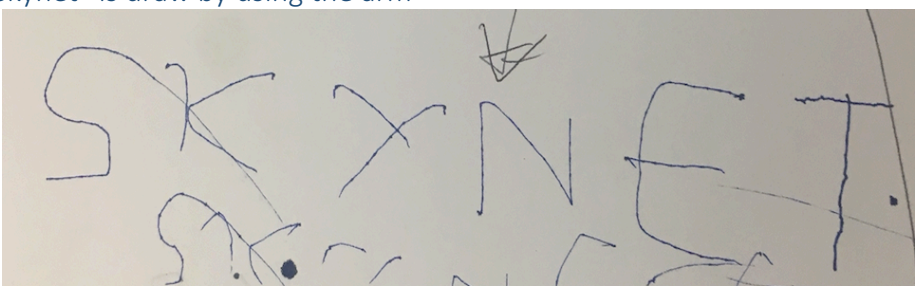
Figure 4.1.3 drawing a circle

As shown above: the circle drawn by SCARA arm and code designed for drawing a circle automatically in java simulation by using the circle equation

$$(x-h)^2 + (y-k)^2 = r^2$$

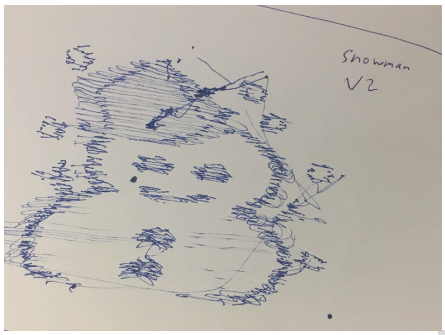
The first and second "for-loop" was designed to draw the top and bottom half of the circle respectively and the first "for-loop" was repeated a final time to ensure that the SCARA arm could finish the circle.

4.1.4 "Skynet" is draw by using the arm



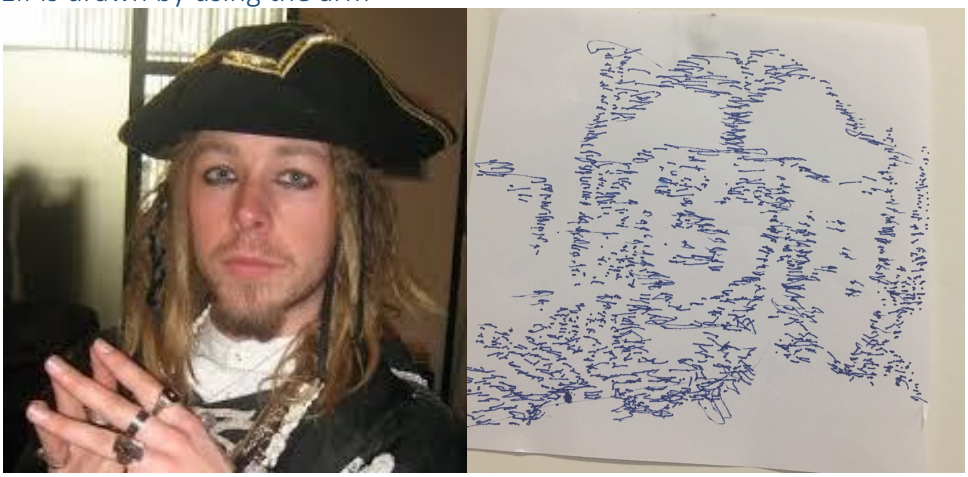
"Skynet" is drawn by using the mouse instead of drawing it automatically by using a code. The simulation pen is lifted up and down to prevent a continuous stroke from being drawn since there are gaps between each letter of "Skynet".

4.1.5 Snowman



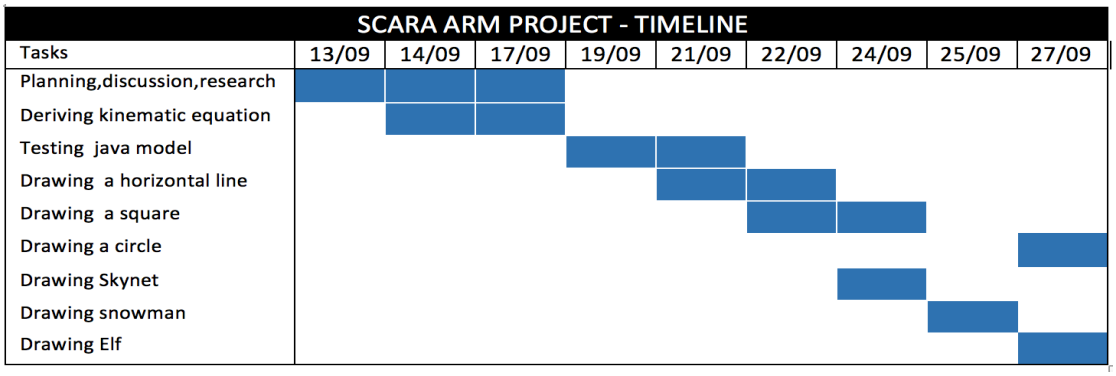
The above snowman was drawn by our teams SCARA arm. By using *BufferedImage* java class and 2D array, the image of the snowman was read and simulation SCARA was executed to draw only the black points. The load image java method is shown on the appendix and GitHub (https://github.com/Katiemouse/SCARAArm/blob/master/java%20code/Main.java)

4.1.6 Elf is drawn by using the arm

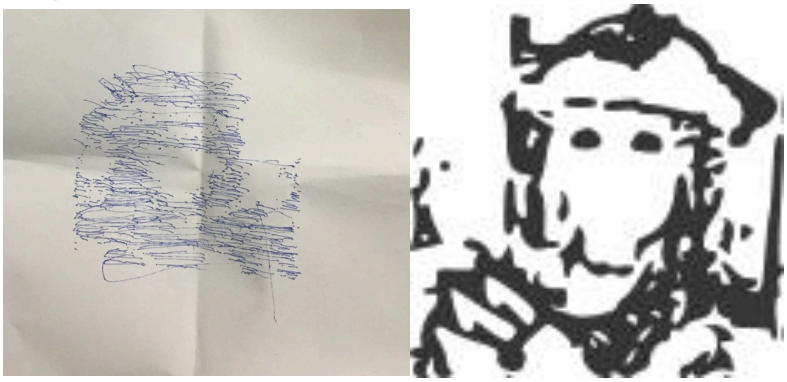


The above image of Elf was drawn by our teams SCARA arm. The same method to draw the snowman was used to do this. However, processing a readable white-black image of Elf from the colourful picture poses a challenge and the solution is explained in the discussion section in details.

4.2 Our team' timeline of events



4.3 SCARA arm testing week results



A B

Figure 4.3 First try of Elf

All the hardware tasks were attempted and achieved by the team before laboratories time and 95% of the marks were attained. In fact, on the second lab session, snowman and circle were not drawn perfectly and only 75% of the marks were achieved. The first try of Elf drawing (Figure 4.3A) was a failure. However, after further research, the original image and the SCARA arm were adjusted and improved. On the final lab session, clearer versions of the Elf, snowman and circle were produced by reading better images and using another SCARA arm.

5. Discussion

Even though 100% of the marks was not achieved, overall, the team has shown exemplar teamwork. The team experiences meaningful and important lessons such as collaboration, learning from mistakes from both software and hardware tasks, time management and problem solving for a project. For example, in the first laboratory session, the team only focused on the derivation of kinematic equation and too much time is wasted spent on discussion instead of conducting tests on the java simulation.

Instead of testing the codes while programming them, the team finished programming the codes and tested them at the end. As a result, the implementation of the kinematic equation and the operation of the simulation arm was not up to expectation. The failure is mainly due to a mistake made when using the inverse kinematic java method to calculate the positions of the two elbows of the simulation arm. Hence, more time is used to modify the codes to improve the operation of the simulation arm

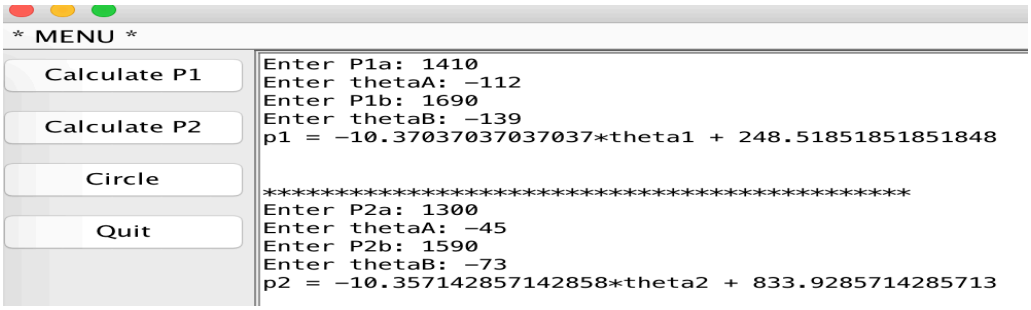


Figure 5.1: screenshot for the java program of calculating the PWM equation

In order to draw a straight line or more complicated things by SCARA arm, accuracy of linear equation involving PWM value and rotation θ needs to be precise. Hence, a small java program which only requires two input values ((PWM_A, θ_A) and (PWM_B, θ_B)), was designed to calculate the equation (shown on Figure 5.1) efficiently for different SCARA arms. Furthermore, there are two main ways to improve the accuracy of the PWM equation. Firstly, the two points with the largest variation between each other should be used. For example, Point A (1410, -112), point B can be either (1510, -120) or (1710, -130). But the better choice is (1710, -130), since 1710-1410 =300 > 100 =1510-1410. Secondly, the initial few points (PWM,θ) can be ignored as they are invalid for calculation the linear equation, since the difference between neighbouring points are not constant. Hence, the best choice for point A and B is the first point after the invalid points and the last point respectively. Once the accurate PWM equation was found, the SCARA arm can draw perfectly a straight horizontal line instead of a curve.

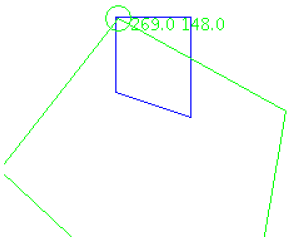


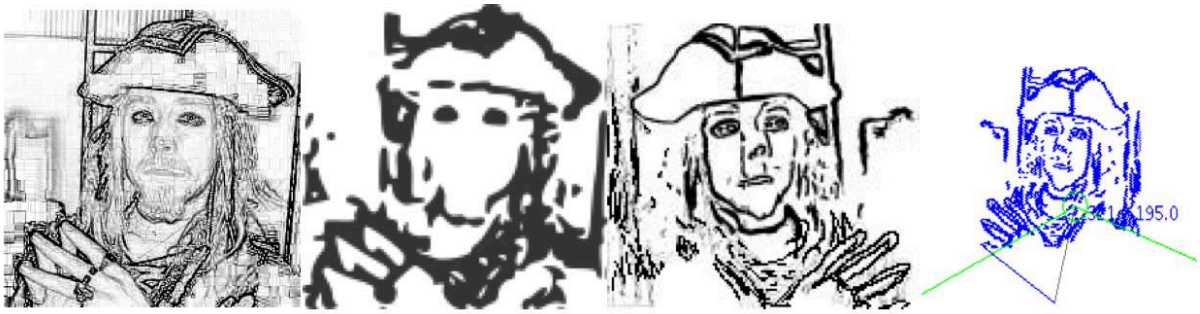
Figure 5.2

Drawing the square was another challenge faced by the team. The perfect square was drawn on the Java simulation, however, by observation, we found two issues that the corners of the square are drawn like a curve and the pen stopped drawing before it completes the square. After several trials, we noticed that the pen was stuck for a few seconds while it was drawing the corner of the square. To prevent the pen from getting stuck, the PWM3 was reset to a decent value, since the PWM3 determines how high the pen is lifted from the paper. Different values of PWM3 was set to obtain the best PWM3 that causes the pen to touch the paper lightly. Once the decent PWM3 was set, the pen did not get stuck at the corner anymore. However, the square was still imperfect due to some unknown issue. To solve this issue, the simulation square (Figure 5.2) was drawn in such a way to counter the drawing made by the SCARA arm.

Next, programming was used to draw a perfect circle and other complicated drawings instead of using a mouse and draw them manually in the java simulation. A new graph would take too much time to draw before every test. Therefore, it was deduced that by storing the coordinate points for a graph, it can be employed by *setMouse()* java method to click on the relevant points of the graph and start drawing. The method to draw a circle was previously mentioned in the results section. However, same as the square, the simulated circle was drawn perfectly but the actual circle was not completed by the SCARA arm. Hence, the "first-loop" was repeated to ensure that the SCARA arm could finish the circle.

For drawing the "Skynet", lifting of the pen is the most important factor. Furthermore, "Skynet" is drawn using the mouse instead of drawing it automatically by using a code. The simulation pen is lifted up and down to prevent a continuous stroke from being drawn since there are gaps between each letter of "Skynet". In the simulation, for the letters that require more than one stroke, it was drawn perfectly. However, the result drawn by the SCARA arm shows that there are gaps within letters 'K', 'Y', 'E' and 'T'. Therefore, we adjusted each letter mentioned in the simulation to improve the version drawn by SCARA arm.

The hardest part of the project is drawing an image by using SCARA arm. To draw the image, the pixels were first read and the black pixels were stored into a 2D array. Next, simulation pen was lifted up and down to draw the white and black pixels respectively. To read image by java, the 2D array was employed, since the team had learnt how to read "ppm" images file by using 2D array in the last trimester. After researching, the team found that *BufferedImage* java class can be used to convert the image from ".png" or ".jpg" to RGB data in the java model so that it can be stored in the 2D array. Therefore, *BufferedImage* and 2D array were used to draw snowman and picture of Elf. Since the snowman image is not made up of pure white and pure black pixels, the threshold needs to be set to a more suitable value. Hence, by printing out all of the colour values from the image, the average value between maximum and minimum colour values is set as the threshold which produce the best image possible.



A B C D

Figure 5.3 Black-white Elf

However, since the picture of Elf is colourful, it was another challenge for the team. Hence, the solution was to use Photoshop to covert the picture into a black and white image (Figure 5.3A). Yet, the black and white picture of Elf is not clear and the sense organs were not visible (Figure 5.3B). This may be due to the brightness of eyes and nose not being dark enough to be distinguished from the face, which results to the first outcome as shown on Figure 4.3. In order to solve this problem, the outline of the black-white image of Elf was redrawn (Figure 5.3C) to look like the image of the snowman. Finally, the clear drawing of Elf was produced successfully (Figure 5.3D).

In terms of teamwork, the team collaborated effectively as every teammate took part in discussing and giving advice when a problem arose. Meanwhile, everyone has different attributes that contributes to the team in different ways. For example, everyone is allocated to three different tasks but help was provided by team members when the need arises. Working as group on this project contributed significantly and gave me great insights into teamwork and workload sharing in a project.

6.Conclusion

In conclusion, the aim of this project was to successfully design, construct and program SCARA arm which could accurately draw a graph or image. The SCARA arm project enabled the team to think critically and constructively in a group environment under time constraints. Also, it introduced the team to what is now commonplace amongst industry and how most tasks these days are not done independently. Furthermore, the communication skills of the team were improved even though at times tension arose between members but professionalism was practiced throughout and problems were solved constructively. In addition, after the project completion, the fundamental knowledge of accomplishing a task under budget, in time and successfully as a group, was learnt as a vital skill applicable in industry and throughout our university life.

7. Reference

1. Eldridge, E. (2016, April 12). (Eldridge, ENGR110 Control Systems: A Robotic Arm, 2016) Retrieved from assignment script: http://ecs.victoria.ac.nz/foswiki/pub/Courses/ENGR110_2016T2/Assignments/RobotArm.pdf

8. Appendix

Load Image JAVA method

```
public void loadImage() {

    try{
        BufferedImage image = ImageIO.read(new File("snowman.jpg"));
        int [][] pixels = new int[image.getWidth()][];

        for (int x = 0; x < image.getWidth(); x++) {
            pixels[x] = new int[image.getHeight()];

            for (int y = 0; y < image.getHeight(); y++) {
                pixels[x][y] = (int) (image.getRGB(x, y) > -12000000 ? 0 : 1);

                UI.println(image.getRGB(x, y));
            }
        }

        double left = 240;
        double top = 100;
        double scale = 1;
        int lastValue = 0;
        int count = 0;
        for (int row = 0; row < pixels.length; row++) {
            if (row %2 == 0){
                for (int col = 0; col < pixels[0].length; col++) {

                    int currentValue = pixels[row][col];
                    if (lastValue != currentValue ) {
                        if (currentValue == 1) {
                            lastState=state;
                            state = 3;
                            state = lastState;
                            arm.inverseKinematic(left+col,top+row);
                            if (arm.valid_state) {
                                drawing.add_point_to_path(left+col,top+row,false); // add point wit pen up
                            }
                        }
                        else {
                            arm.inverseKinematic(left+col,top+row);
                            if (arm.valid_state) {
                                drawing.add_point_to_path(left+col,top+row,true); // add point with pen down
                            }
                        }
                        lastValue = currentValue;
                        count++;
                    }
                }
            }
            else {
                for (int col = pixels[0].length-1; col >= 0; col--) {

                    int currentValue = pixels[row][col];
                    if (lastValue != currentValue ) {
                        if (currentValue == 1) {
                            lastState=state;
                            state = 3;
                            state = lastState;
                            arm.inverseKinematic(left+col,top+row);
                            if (arm.valid_state) {
                                drawing.add_point_to_path(left+col,top+row,false); // add point wit pen up
                            }
                        }
                        else {
                            arm.inverseKinematic(left+col,top+row);
                            if (arm.valid_state) {
                                drawing.add_point_to_path(left+col,top+row,true); // add point with pen down
                            }
                        }
                        lastValue = currentValue;
                        count++;
                    }
                }
            }
        }
        UI.println(count);
        UI.printMessage("DONE");
    }
    catch (IOException e) {UI.println("File loading failed: "+e);}
}
```

Inverse kinematic Java method

```
public void inverseKinematic(double xt_new,double yt_new){

  valid_state = true;
  xt = xt_new;
  yt = yt_new;
  double dx1 = xt - xm1;
  double dy1 = yt - ym1;
  // distance between pen and motor1
  double d1 = Math.sqrt(Math.pow(dx1,2)+Math.pow(dy1,2));
  if (d1>2*r){
    //UI.println("Arm 1 - can not reach");
    valid_state = false;
    return;
  }

  double l1 = d1/2;
  double h1 = Math.sqrt(r*r - l1*l1);

  double beta11=Math.atan2(yt-ym1,xm1-xt);  //the angle is between motor1-Pen line and horizontal line
  double beta12=((Math.PI)/2 - beta11);   // the angle is between "h" line and horizontal line
  double  xa1=xm1+0.5*(xt-xm1);   // the mid-point a1 between  tool and motor2
  double  ya1=ym1+0.5*(yt-ym1);

  // elbows positions
  xj1 = xa1+h1*(Math.cos(beta12));
  yj1 = ya1+h1*(Math.sin(beta12)) ;

  // theta1 = Math.atan((yt-ym1)/(xt-xm1)) ;
  theta1=Math.atan2((yj1-ym1), (xj1-xm1));

  if ((theta1>0)||(theta1<-Math.PI)){
    valid_state = false;
    UI.println("Ange 1 -invalid");
    return;
  }

  // theta12 = atan2(yj12 - ym1,xj12-xm1);
  double dx2 = xt - xm2;
  double dy2 = yt - ym2;
  double d2 =  Math.sqrt(Math.pow(dx2,2)+Math.pow(dy2,2));
  if (d2>2*r){
    // UI.println("Arm 2 - can not reach");
    valid_state = false;
    return;
  }

  double l2 = d2/2;
  double h2 = Math.sqrt(r*r - d2*d2/4);

  //  double beta21=Math.atan((yt-ym2)/(xm2-xt));  //the angle is between motor2-Pen line and horizontal line
  double beta21=Math.atan2((yt-ym2), (xm2-xt));
  double beta22=((Math.PI)/2 - beta21);   // the angle is between "h" line and horizontal line
  double  xa2=xm2+0.5*(xt-xm2);   // the mid-point a2 between  tool and motor2
  double  ya2=ym2+0.5*(yt-ym2);

  // elbows positions
  xj2 = xa2-h2*(Math.cos(beta22));
  yj2 = ya2-h2*(Math.sin(beta22));

  // motor angles for both 1st elbow positions
  // theta2 =Math.atan((yt-ym2)/(xt-xm2)) ;

  theta2 =Math.atan2((yj2-ym2),(xj2-xm2)) ;
  if ((theta2>0)||(theta2<-Math.PI)){
    valid_state = false;
    UI.println("Ange 2 -invalid");
    return;
  }

  // UI.printf("xt:%3.1f, yt:%3.1f\n",xt,yt);
   //UI.printf("theta1:%3.1f, theta2:%3.1f\n",theta1*180/Math.PI,theta2*180/Math.PI);
   //UI.printf("theta1 + theta2 = %3.1f\n",(theta1+theta2)*180/Math.PI);

  //solve angles
  double a = Math.hypot(xj1-xm1, yj1-ym1);
  double b = Math.hypot(xj1-xt, yj1-yt);
  double c = Math.hypot(xm1-xt, ym1-yt);
  double gamma1 = Math.acos((c*c-a*a-b*b)/(-2*a*b))*180/Math.PI;

  a = Math.hypot(xj2-xm2, yj2-ym2);
  b = Math.hypot(xj2-xt, yj2-yt);
  c = Math.hypot(xm2-xt, ym2-yt);
  double gamma2 = Math.acos((c*c-a*a-b*b)/(-2*a*b))*180/Math.PI;

  //UI.printf("gamma1 + gamma2 = %3.1f\n",(gamma1+gamma2));
  if (gamma1+gamma2 <= 90) valid_state = false;
  //else valid_state = true;


  return;
}
```