# RoboVinci

Robbies
Elena Pitta, Myriana Miltiadous, Dimitrios Kourntidis, Katerina Zacharia

June 7, 2023

**Abstract**

This report presents a comprehensive overview of a horizontal plotter's design, functionality, and implementation. In the end, there is a link to a GitHub page with the video and the source code.

## Introduction

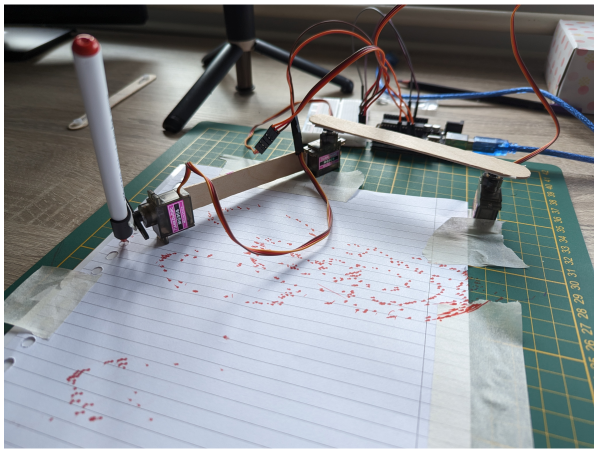
After some research, our team decided to create a robot arm that is able to draw on the horizontal axis, using dots, as we believed it would be both fascinating and challenging.

## Related work and Innovation

The open-source BrachioGraph [1] project offers a complete process and architecture for building a horizontal plotter. Hobbyists, artists, and educators will find the BrachioGraph to be an appealing alternative because it is made to be inexpensive and simple to construct utilizing commonly available materials. Those implementation typically use GCODE to draw, meaning that the image should be vectorised and the GCODE to be created typically with different software. We approach this problem with an E2E solution, meaning that we apply some preprocessing to the picture and then plot it.

## Design

The hardware we used for the implementation was three MG90S servos, an Arduino Uno, some DIY wooden sticks, a marker, double-face tape, and paper. We also used a thin platform to mount the plotter, as well as the paper. It is important to note that the outer servo (the side where the pen is not attached) is placed on the downright corner of the paper.



- **Servo motors:** Electric motors that can precisely control angular or linear position, velocity, and acceleration. The servos are responsible to move our structure. two of the servos are designed to move the robot right or left (x-axis), enabling movements that are approaching those of humans in a single axis. The third servo is responsible for moving the marker up and down.

- **Arduino Uno:** The Arduino Uno is an open-source microcontroller board with many input/output (I/O) ports that is a low-power, high-performance chip. In this particular project, we used the Arduino in order to control our servos using Firmata protocol.

## Implementation

### Image Prepossessing

In order to make the plotter able to draw an image, it is principal to reduce its complexity. Given an image as an input, a canny edge detector is executed and returns only the edges of the inserted image. Those edges (i.e the picture with less complexity) are stored in a multidimensional n x m array called A. Each element of the array corresponds to one pixel of the image. So, we end up with n vectors of length m, where n is for the y-axis and m for the x-axis. In our case, because of canny transformation, the only numbers in the array are 255 for white color (dots taken into account) and 0 for black colour (background). Since the edges are still too many, and the complexity remains high, we decrease the points in reasonable quantity. We choose the number of points considered as optimum, by taking under consideration both the time needed to draw the image and the accuracy of the image (accuracy-time trade-off). Usually, around 1500 points are kept for highly detailed images, like animals or cartoons (around 90 minutes), whereas for simpler images like letters or shapes (circles, triangles etc) 200 points (around 10 minutes) are enough.

### Transformation of dots and Inverse Kinematics

After storing the edges, in the array, we need to translate them into (x,y) Cartesian coordinates(procedure described in section 'Image-to-Plotter Mapping'), which represent the position that the robot should head to and draw a bullet. The step we proceeded to after that conversion, is the implementation of turning the servos correctly so the robot will reach the points (x,y). For doing so, inverse kinematics are used. Inverse Kinematics is the mathematical way to set the robot's end effector at a certain position and orientation. RoboVinci has two servos that need to be turned at the right angles to reach each of the desired (x,y) points. So, we found and modify an algorithm that gets as input a (x,y) point and the lengths of both links and returns two angles, theta1 and theta 2 which corresponds to the degrees that each servo should turn to approach the (x,y) point.

### Image-to-Plotter Mapping

Since the input picture might have any dimensions, we aimed to scale the space where our plotter can write, according to the image's n x m array, A, mentioned earlier, and get as output all the (theta1,theta2) points that the servos should turn. Specifically, the space where RoboVnci is able to write in, has dimensions L x L. L is calculated with this formula: $L = (length1 + length2) * cos(\pi/4)$, with length1 and length2 demonstrating the lengths of the two links. However, our picture might not be in a square format ($m \neq n$); If n is bigger than m ($n>m$) , it means that the image is in portrait format and so the rectangle in which our plotter is able to write is calculated by: L1=L and L2=L*m/n. On the other hand, if m is bigger than n ($m>n$), it depicts a landscape image and hence the rectangle, in which our plotter is able to write, is calculated by: L1=L*m/n and L2=L. So, the rectangle with dimensions L2(y-axis) x L1(x-axis) is where the plotter can draw the image.

Afterward, we need the algorithm to divide that L2 x L1 rectangle into very tiny rectangles(in the size of a dot) and each one of them to represent one element of the image's array(i.e. one pixel). This is done with the following formulas: $y\_axis\_lambda = L2/n$, which is the y-axis length of each tiny rectangle, and $x\_axis\_lambda = L1/m$, which is the x-axis length of each tiny rectangle. So, now each element [i,j] of the matrix A corresponds to the point $(L1 - j * x\_axis\_lambda, L2 - i * y\_axis\_lambda)$ in Cartesian coordinates. With a for loop through all the [i,j] elements of the array, the ones with a value equal to 255 were obtained in Cartesian coordinates as described above. All the x coordinates were then transformed to -x since our plotter is positioned on the right angle of the paper so the x points should be negative in order for the inverse kinematics formulas to give the right angles. After that, having all the x and y coordinates where a dot should be drawn, using the inverse kinematics formulas the algorithm provides as output all the (theta1,theta2) angle points and by giving them to the plotter, it can draw any image. In figure 1 the above procedure is explained with visualization for better understanding.

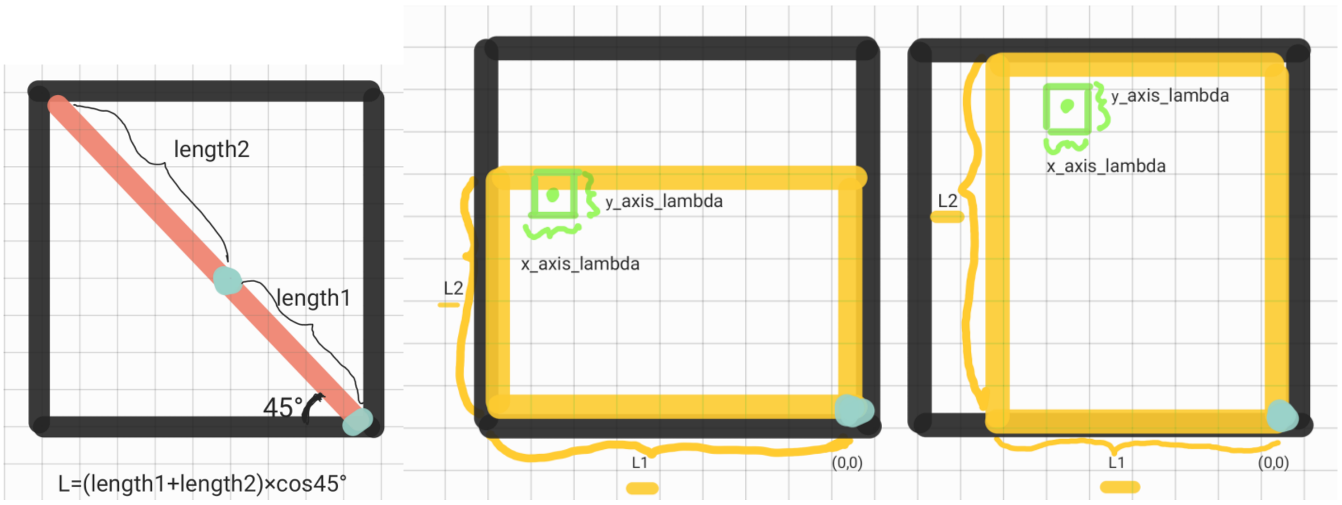


Figure 1: Left figure: Calculation of L. The blue dots represent the servos and the red lines are the two links. The black box is where the plotter is able to write. Middle and Right figure: In both pictures, the black square is where the plotter is able to write. In the middle figure is the case for which the image is in landscape format and in the right picture is the case for which the image is in portrait format. The yellow diagram is where the drawing is going to be on the paper. The green squares are where the dots should be placed if the array element in that position is equal to 255. For example, for the middle figure if the [0,1] element of the array A is equal to 255 then a dot should be placed at the $(-L1 + 1 * x\_axis\_lambda, L2)$, which is represented by the green box. And in the same pattern for the right figure if the [1,1] element of the array A is equal to 255 then a dot should be placed at the $(-L1 + 1 * x\_axis\_lambda, L2 - 1 * y\_axis\_lambda)$, which is again represented by the green box.

## Servos setup

In this section, we will describe how the servos are set up and programmed. As mentioned before, we placed the outer servo on the bottom-right corner, thus we initialize the rotations of the servos to be as follows: Sevro 0 to 90 degrees, servo 1 to 90 degrees, and servo 2 (the one that holds the pen) to 60 degrees, i.e., to keep the pen up. The way servo 1 is set to 180 is by subtracting the corresponding degrees from 180. This is done with the purpose of calibrating the servos.

At this point, the $theta1$ and $theta2$ degrees calculated in the previous stage, are now given to servo 0 and servo 1 respectively, as a pair $(theta1, theta2)$. The pen's servo is only moving down when both theta1 and theta2 are reached, and only for some seconds, i.e., until it marks down the corresponding dot. The degree we set for the pen to touch the paper is 80.

An important factor for optimizing these movements is the accuracy of the rotations. We can determine if the dot is drawn at the correct position by computing the predicted spot and comparing it to the real one. As expected, after experimenting with this, we found out that we should add offsets for servo 0 and 1. Finally, we observed that every time a servo moved in the opposite direction of the one that it moved, a problem occurred. To solve this, the previous theta is taken into consideration, and it is compared to the new theta. According to where it should go, it now moves clockwise or anti-clockwise, by increasing i in a for loop, or by decreasing it.

## Results-Drawings

We first experimented with low-dimensional and low-complex images, such as a rectangular and the letter 'A'. The results were very efficient as the plotter achieved to draw both in a way they were distinct and clearly visible. Then, we have inserted images with higher complexity, such as the character Mickey Mouse, and a dolphin that jumps out of the sea. The dolphin, the letter 'A' and Mickey Mouse drawings are all illustrated in Figure 2. The referenced images can be found in the github repository link we provide below, in appendix, where also all the coding is presented.

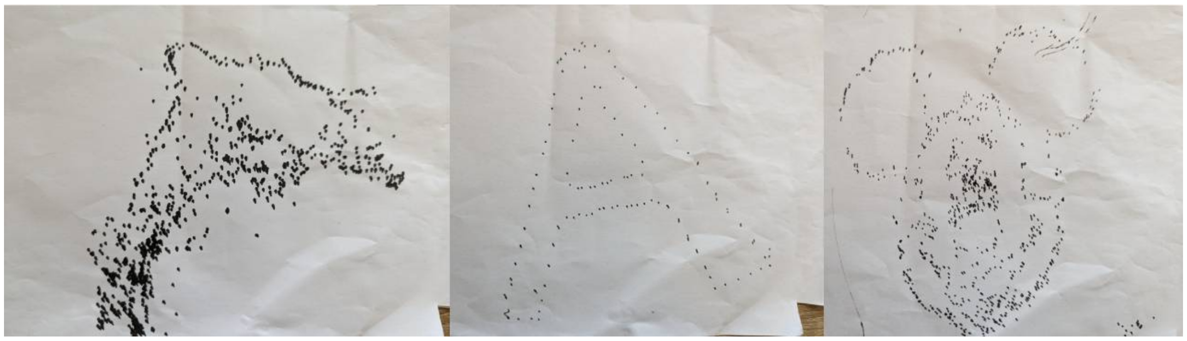


Figure 2: Plotter results

## Conclusion and Discussion

This project has been very interesting and with a lot of challenges. The most challenging part was to optimize the movements so as to increase the accuracy of the drawings. However, we think that we reached a great level of efficiency, given the disposal time and resources, and we definitely learned a lot about servos, Arduino, and Robotics in general. Note: In the link given in Appendix, you can view the whole procedure of the plotter drawing.

## Appendix

https://github.com/j1mb0o/RobbiesProject

## References

[1] Procida, D. Brachiograph. https://www.brachiograph.art/. Accessed: May 28, 2023.