Mr. Drawer: Robot Arm That Draws

Trudy Ani-Asamani, Alan Azargushasb, Alonso Ninalaya, Chedlyne Valmyr

Dept. of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

Abstract –The purpose of this project is to build a robot arm that can paint and draw by recreating images uploaded to it. The robot with the help of a tool such as marker or paint brush would draw and paint desired images. The robot arm through programmed and calculated movements would execute the job of painting and recreating images with motors. The design of the robot allows it to perform strenuous tasks such as rotation and moving horizontally and vertically. It would be able to compare input and output in order to analyze how well the task is being performed. It also has color recognition capabilities.

Index Terms – Microcontrollers, Servo motors, Camera module, Printed Circuit Board (PCB).

Introduction

The objective of the project is to build a robot arm capable of drawing. The raspberry camera takes pictures of the desired input. With the use of servo motors and a marker, the drawing task is completed. This project was a good demonstration of what was learnt at the University. As of right now every member of our group contributed to the project. However, the synthesis of said elements has not occurred. Alan Azargushasb was in charge of building the robot frame and the general testing, and programming of the robot Arm. Alonso was in charge of Robot Vision. Chey was in charge of the communication between the chip from the PCB and the raspberry pi. Trudy was in charge of powering the entire robot arm.

In this document, the different sections that integrated into the final prototype are discussed in much detail. The eyes or input of the robot arm is the raspberry pi camera module. The servo motors are responsible for the movement of the robot arm.The raspberry pi and the ATMEGA2560 serve as the control hub for the raspberry camera and the servo motors respectively. With computer vision, our camera module will be capable of shape and color detection. Additionally, information on inverse kinematics provided context in the programming of the position and direction of the servo motors.

Overview

The robot arm with the use of a marker is able to reproduce shapes and 2-D images. With images uploaded through the camera module, color and shape recognition features of the software allows the raspberry pi to analyze the provided input. Calculations using inverse kinematics allows the servo motors to be programmable in order to achieve the wanted output. Computer vision enables recognition of shapes and colors on the inputted image. The power supply of the robot arm is paramount to having a fully functional robot arm. The PCB which was later implemented using the breadboard enabled the electrical connections of the microcontrollers and other components necessary for the project. The paint brush attachment allows a paint brush or marker to be attached to the robot arm.

Arduino 2560

During this project we used the Arduino 2560 to control the servo motors. In retrospect this was overkill as we could have just gone with the regular Arduino uno. Both the Arduino uno and the Arduino 2560 have 6 pulse width modulation pins. The only difference is that the 2560 has a better processor and 20 digital pins. When I started this project I thought I might use the digital pins. I never did. The Arduino 2560 served us well throughout this project. I created a state machine of my code as seen below.

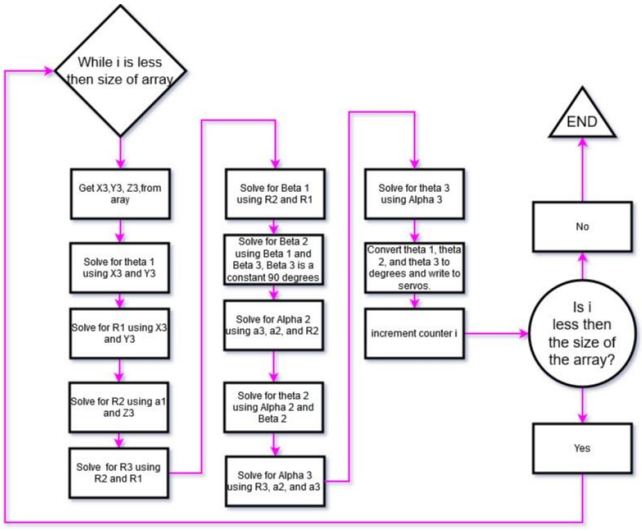


Figure 1. State machine of our code running on the Arduino 2560

Raspberry Pi

A variety of single board computers exist on the market today. The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV. It is a capable little device that enables people of all ages to explore computing and learn how to program in languages Scratch and Python. It is capable of doing everything you would expect a desktop computer to dbeen used in a wide array of digital maker projects, from machined and pareno. It has the ability to interact with the outside world, and has t detectors to weather stations and tweeting bird houses with infra-red cameras. For this design the group needs a board powerful enough to run our code and send the output via a transmitter. It will be in charge of taking input from the camera and send the output to our atmega microcontroller. Moreover, we need to choose a language that is understandable and much less complicated overall. Python programming is the language that the group desired to code in for the raspberry Pi. After doing an extensive research of the different types of single-boards that are available, we decided to use the Raspberry Pi version 4 with 1 Gb of RAM. and it is able to process and transmit data successfully. There are different versions of raspberry pi and one of the most important aspects was the support of the community with the operating system for this version. For the previous versions, there is no much support and help for any troubleshooting happening.

Raspberry Camera Module

With a huge variety of cameras available for the Raspberry Pi, it was hard to decide which one could do a decent job without investing too much money.We used the raspberry pi camera module which is native from the raspberry pi environment. The reason we went for this is mainly for the price, however, the compatibility that we get from this camera is super important for our development; especially, for troubleshotting. However, it is important to mention the limits of this camera. It was hard for us to detect the right shapes and color at night time. Appropriate lighting was needed for better results. However, we still decided to take that tradeoff and pick this camera.

Computer Vision Software

Our single-board will perform the shape and color detection using our camera. We decided to implement OpenCV in our algorithm since it was the most reliable and has been used in many computer vision projects. However, there are a few requirements before we are able to use this library freely. It is important to mention that there is a lot of different operating systems we can download to our Raspberry Pi. However, Raspbian has been the official system and reliability is an important factor for our project. Raspbian is a Linux Distribution and as such, we need to update and set up our environments in order for us to use all the functionalities from OpenCV, which requires running around ten commands to make it work without any future error. This step is the most important phase of the beginning since it saves our team a lot of time when using OpenCV. After this step is done, installing python3 and PiCamera is very intuitive and easy.

Our algorithm with computer vision required studying subjects such as thresholding, contours, data types such as numpy arrays and, of course, color properties which are going to be explained in this section. Computer vision has many subsections and many ways to solve problems with object detection; however, this is the way I, Alonso Ninalaya, thought makes the most sense. After we took a picture with any camera, we received the number of pixels stored in a numpy array (structure used by OpenCV by default). Right after, we need to check what color we get from the image. For this matter, it is important to keep in mind how RGB (red, green and blue) works and design a histogram. Every picture taken comes with set values for their channels. Each channel represents one of the basic colors mentioned before. Our histogram from below is a graphical representation showing how frequently various color values occur in the image. We use 256 since we want to see the pixel count for each of the 256 possible values in the grayscale. Then we pass 0 and 1, which is the value range of our input image after transforming it to grayscale.

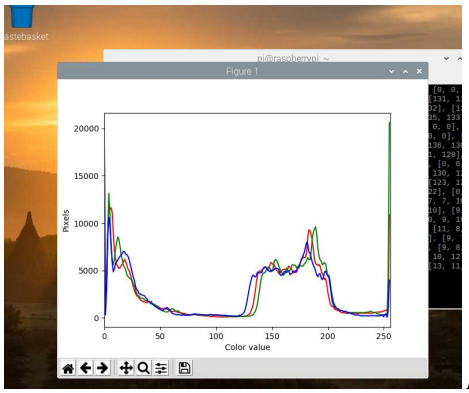


Figure 2. Histogram of one of our pictures taken from our camera

After finding the range values of our picture, we proceed to apply the threshold to our image so that it is easier for us to find objects later on. The process of threshold works like this. What is thresholding? It is a type of image segmentation, where we can change the pixels of an image to make the image easier to analyze. It is related to finding areas of interest of an image. While ignoring the parts we are not concerned with. Because we are using OpenCV, we have access to different functions that allow us to apply the thresholding to our current image. It is important that our image being used needs to be on grayscale since our thresholding transforms our image into a binary image (one that is simply black and whtie).

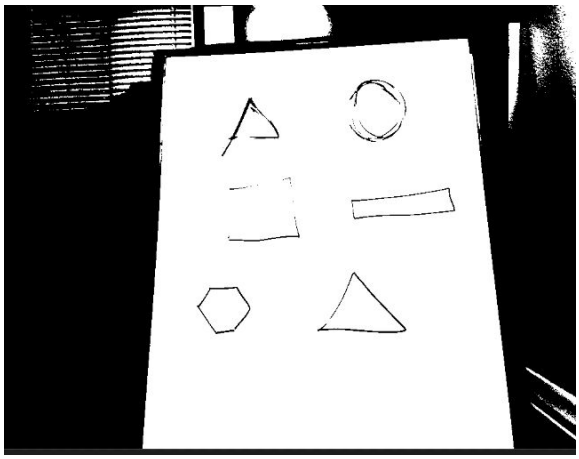


Figure 3: Binary Image after applying thresholding

Continuously, we need to find the contours. What are contours? They can be explained simply as a curve joining all the continuous points, having the same color or intensity. Contours are useful for a tool for shape analysis and object detection and recognition. In OpenCV, finding contours is like finding white object from black background. For our project, this was perfect since our surface is a whiteboard. Contours is represented as a Python list of all the contours in the image. Each individual contour is a numpy array of (x,y) coordinates of boundary points of the object. Contours help us identify shapes present in an image. They are defined as lines joining all the points along a boundary of an image that are having the same intensity.

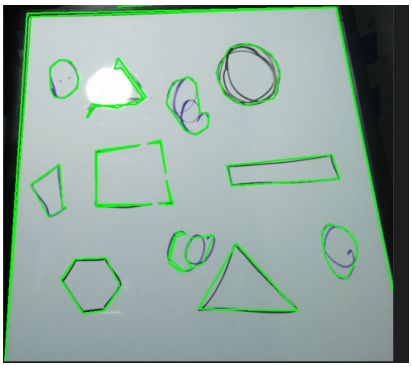
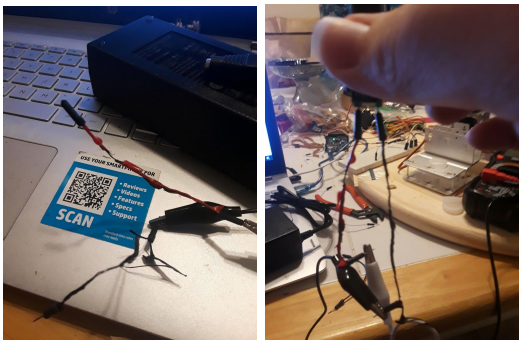


Figure 4: Contours found in our whiteboard

Once we found our desired contours, we proceeded to simply get the list of contours found. As mentioned before, the contours are represented as a lit of (x,y). Hence, the final step is to get all the pixels and send them over to our atmega microcontroller.

Power Supply

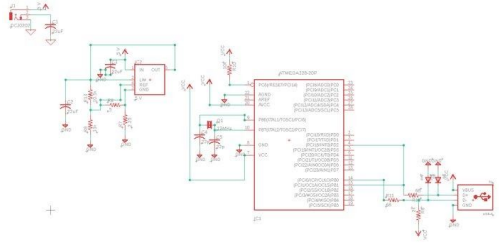
The power supply provides power throughout the entire project. We had two power supply options.The first one is rated at 9 volts and 1 amp. The second power supply is rated at 5 volts and 15 amps. The second power supply was however too powerful for the system and fried the wires as seen in the figure. We went back to the first power supply. With the voltage regulator we get about 5 volts with 700 mA which gives us 3.5 watts of power.



Figures 5 :Wires Fried by Power Supply

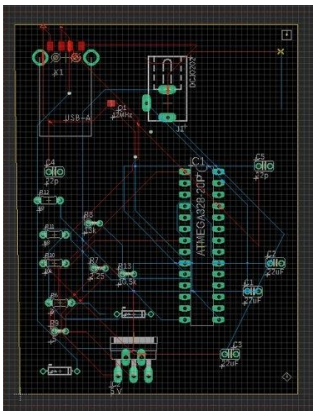
Printed Circuit Board

The PCB is an important part of the project because it allows the connection of multiple devices in one. The PCB controls the communication and the movements of the robot arm. The PCB allows a transition in this project that allows everything to come together and work as one.



Figures 6:PCB schematic

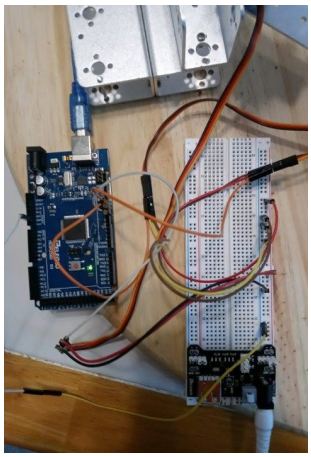
The image above shows the connections in the circuit using the Atmega microcontroller, the Raspberry Pi GPIO pins, the USB port for serial communication, and the DC jack as well as the voltage regulator. These components would allow the PCB to come together. I have included an image below that shows the board layout view of the circuit board we were to use for our robot arm. This PCB allowed all of our parts to be in one place. our original idea for the robot arm allowed for the Atmega and the Raspberry Pi pins to communicate, sending xyz coordinates from the Pi to the Atmega, doing this will allow the servos to draw an images based off of the coordinates they received from the Pi.



Figures 7 :PCB Board

There were a couple of issues that arrived when we made an attempt to order the PCB for our project. We did face a couple challenges when trying to order our PCB, In the beginning of May our original PCB failed due to the changes we had to make on our robot arm in the beginning of the semester. Upon designing a new PCB I attempted to order one from the company called PCBCart after waiting about 3 weeks for the order to arrive it was cancelled by the company, after making an attempt to get it quickly mailed to the United states it was cancelled again because it was being delivered from another country. IN THE beginning of the semester Professor Richie said as a backup we can use the breadboard as an alternative, I was able to order a couple of our parts for the pcb but some parts were too big to fit the breadboard as well a little dangerous to connect with jumper wires while inside of our homes. We don't have the equipment required to extinguish fires quickly. As an alternative I have included an image of our alternative PCB which is shown on figure 8.

Figure 8 shows the connection with the robot arms servos, voltage regulator, power jack that is powering up the robot arm, and the Arduino board which will be programmed to instruct the robot arm to perform certain movements.



Figures 8:Breadboard

Robot Arm

In the beginning our original plan was to build a robot arm that could cook pancakes. We originally thought of the design of the robot arm to be an actual robot arm with a literal robot hand with 4 fingers and a thumb that would pick up various utensils such as spatulas, pan handles, etc. The robot arm would interact with said utensils and a griddle to cook pancakes. That design we learned was superfluous since the added complexity of the fingers and thumbs did not add any functionality and was thus redundant. So we decided to instead use a gripper design as the end effector on the robot arm. Since we were not mechanical engineers we decided to go and look on Amazon and purchased for ourselves a do it

yourself 6 degree of freedom robot arm with a mechanical gripper as the end effector. Later on when the robot arm was built we realized the robot arm would not be strong enough to lift pans or spatulas or utensils. The servos utilized in the robot arm were not powerful enough to do such things and the robot arms slots for the servos were such that new servos would not fit, so we had to use the servos given or start over from scratch. The reason why our servos were not strong enough to do the required tasks of lifting and manipulating cooking utensils is that torque rapidly depreciates the further you get away from the center of gravity. The cost of a robot that would be strong enough to do so was out of our budget. So we decided to change the focus of the project from cooking to painting. We knew that while the Robot Arm was not strong enough to move cooking utensils it could move around painting/drawing utensils. Those things were within the power budget that the servos provided. So we got to work creating a robot that paints/draws. We purchased paintbrushes, and markers, and a whiteboard. The idea was that we would test the drawing capabilities of the robot arm first and then after we had that under wraps we would transition to painting. We never got around to painting, but we did get it to draw.

Inverse Kinematics

Solving the Inverse Kinematics of the robot arm turned out to be a huge hassle. I, Alan Azargushasb, was trained to be an electrical engineer, and all the math related to this was not in electrical engineering but mechanical engineering. In retrospect I would say the vast majority of my work was not in electrical engineering but either mechanical engineering or computer engineering. The only electrical engineering I did was assembling certain wires into the breadboard and Arduino. None of my electrical engineering skills were ever really used in my senior design project. Because of this it took me a while to figure out the equations necessary to program into the Arduino to control the robot arm. I was approaching this completely in the dark. I was lucky to find a robotics course on youtube taught by Angela Sodamann, a mechanical engineer, to help me figure out the math behind it all. Three dimensional robots are generally designated into 5 different types: carestian manipulators, cylindrical manipulator, spherical manipulator, scara manipulator, and articulate manipulator. These manipulators are made out of 2 primary components, a revolving joint, and a prismatic joint. Revolving joints rotate around a central axis and are represented as cylinders in a robotics free body diagram. A prismatic joint moves forward and backward and are represented as squares in a robotic free body diagram. Our Robot arm is made up only of revolving joints and is called an articulated manipulator. I bring this up because for the task at hand, drawing and painting, this is the worst robot you can use for such a task. It is also the most complicated. The math for this kind of robot is much more complicated than the other kinds of manipulators. For the task at hand the simplest and best robot we could use would be a cartesian manipulator, consisting only of prismatic joints. When I bought the robot arm I did not know any of this. I had taken a course in robotics, but it was the introductory course, and did not discuss free body diagrams and solving for the said equations. Luckily I took physics 1-3, as well as Statics and Dynamics at UCF so I was familiar with free body diagrams. If we had used a cartesian manipulator the math would have been very easy. Since cartesian manipulators only consist of prismatic joints I would have a prismatic joint for each axis in a three dimensional space, X-Y-Z. Using a microcontroller I would tell the prismatic joints to expand or contract with respect to their assigned axis. The X-Y coordinates would correlate one to one with the X-axis prismatic joint and the Y-axis prismatic joint and to draw or paint I would expand or contract the Z-axis prismatic joint. My X-Y coordinates would correspond to the X-Y position on the canvas and the paper plus or minus a slight offset. I have zero unknowns. I don't have to solve for any variables. The offsets would come via experimentation and or measurement by hand. The math is simple and it works. The coordinates are my kinematics. With an articulated manipulator it is the exact opposite, I have three revolving joints. Now, the original robot arm had six degrees of freedom. Five of those joints had to do with motion, the last degree of freedom was the servo attached to making the robotic claw and was in charge of opening and closing the claw. Since we were not using the claw but instead our own 3d printed part we got rid of it and the corresponding servo. I got rid of the other two joints and their corresponding servos to reduce the weight as well as the computational complexity of the project. With just 3 joints we have 10 unknowns! All we know is the coordinates of the points we want to reach in 3d space. From those 3 points, in the free body diagram we call them X3, Y3, and Z3 since they are attached to the third joint and are connected to the third servo. We start with two free body diagrams detailing our joints and the relationship they play

with the X-Y-Z coordinates. Let us start off with our constants, our constants are a1, a3, and a3. A1 is the length of base, until the end of servo the center of rotation at servo 2. At least that is how it is supposed to be, we will come back to this point later on. A1 in our measurements was one hundred and thirteen millimeters. A1 rotates around the X and Y planes, from zero to 180 degrees. When the servo, servo 1 is set to angle Zero. The robot is completely in the positive X direction and moves through the X-Z plane. When servo 1 is set to ninety degrees, the robot is completely in the positive Y direction and moves through the Y-Z planes. A2 is the length of the second joint, from the axis of rotation servo 2, to the axis of rotation, servo 3. We find the length of a2 by measurement. With a ruler we found a2 to be one hundred and four millimeters. Joint a2 is controlled by servo 2. The end of joint a2 is pivoted upward in the Z direction when the angle of servo 2 is positive. The lowest angle servo 2 can reach is 60 degrees due to a physical constraint on the arm. There is a metal bracket that impedes the movement of joint a2. At ninety degrees joint a2 is completely horizontal, at one hundred and eighty degrees joint a2 is completely vertical. Joint a3 is from the central axis of rotation to the end of end effector, which is the tip of either the marker, or brush, or 3d printed attachment. Without the marker, the length of joint a3, from axis of rotation to the tip of the 3d printed attachment is one hundred and twenty two millimeters. With the marker it comes out to two hundred and twelfth millimeters. Wherever the tip of a3 is, that is where the coordinate Z3 is as well.

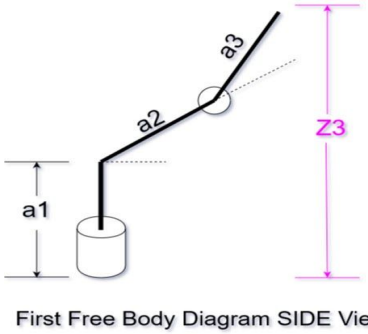


Figure 9: First Free body diagram of Robot Arm, Side view.

As you can see from our first free body diagram. Notice that there is no X3 or Y3 listed in this free body diagram. Why is that? It is because our free body diagram must always be accurate, the position of X3 and Y3 can be affected by servo 1, or rotation by joint a1. We need a second free body diagram to derive the position of X3 and Y3. One that is always accurate no matter what.

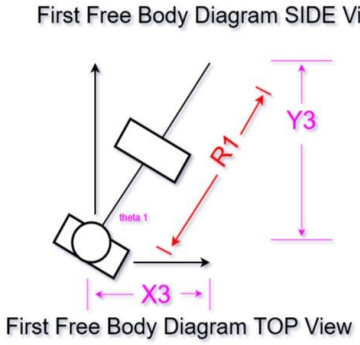


Figure 10:First Free body Diagram Top View

From the top view we now get the X3, and Y3 coordinate. Notice that we set the robot arm in between the X and Y axis. We did this to make sure that this free body diagram is generic and can represent every possibility as a result of servo 1. The angle between the X3 coordinates and the Y3 coordinates is theta 1. From this we create 2 triangles to solve the math for the inverse kinematics. One of the triangles is a right triangle, the other is an oblique triangle.

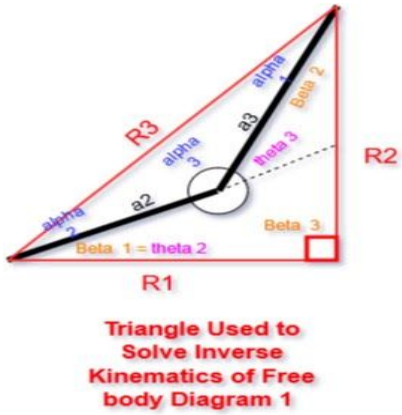


Figure11: Triangle used to solve inverse kinematics of free body diagrams.

The right triangle gives us three unknowns called R1, R2, and R3. The position of the robot arm forms the oblique triangle. They both share the same hypotenuse, R3, and R1 is always the projection of a1 and a2 unto the X-Y plane. From these two triangles we get many unknown angles:alpha 1, alpha 2, alpha 3, beta 1, beta 2, beta 3, theta 1, theta 2, theta 3. We systematically solve for each unknown using trigonometric laws until we know everything. From the knowledge derived from solving these large systems of equations we get theta 1, theta 2, theta 3. These three angles are in radians and will then be converted

to degrees and used as the angles for servo 1, servo 2, servo 3. These free body diagrams show the initial configuration of our robot arm. Originally our whiteboard was vertical, and the robot would draw on the Z-plane , however the whiteboard kept falling backward. So we decided to redesign our robot arm to instead draw on the X-Y plane instead of the Z-plane.

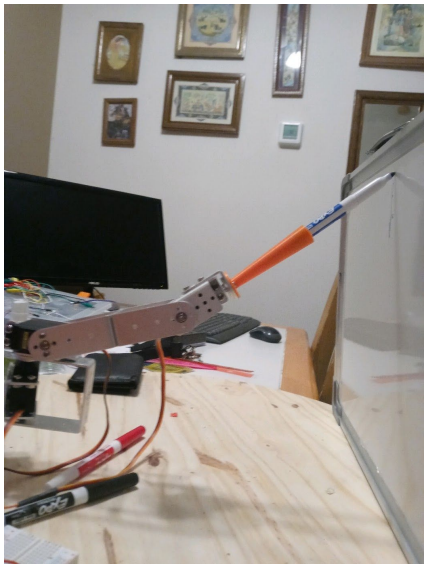
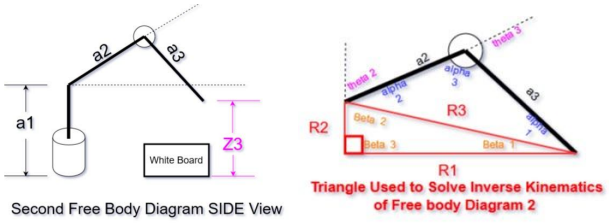


Figure 12:Original Configuration

Since the white bourd now lies flat on the table, we do not have to worry about the white bourd tipping over. However, now we have to redo the math. The triangle formed in our first free body diagram is no longer true for how the robot arm behaves in this new configuration. In the first free body diagram a3 always pointed upward relative to a2. In our new configuration a3 will always point downward relative to a2.



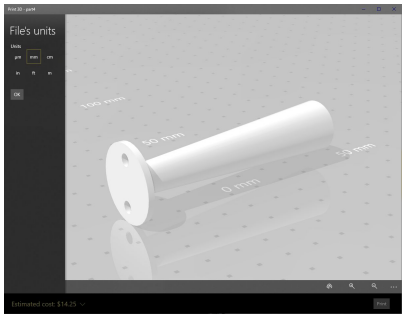
Figures 13 and 14 Second Free body diagram and Second Inverse Kinematic Triangle

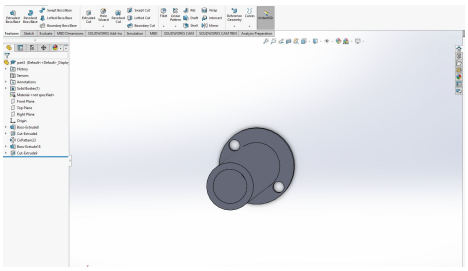
As you can see , in this arrangement Z3 is always below A1.

Paint brush attachment

The paint brush attachment is a crucial part of the robot arm design. This is because the attachment facilitates the accurate execution of painting the canvas or board. Different prototypes of the paint brush attachment were investigated in order to find the best fit for this design. Two main criteria that were essential for the nature of the paint brush attachment were: firstly, the chosen prototype must be able to support our chosen tool (brush or marker). Secondly, the paint brush attachment must be sturdy and robust to facilitate the flawless execution of the artwork. Solid works was the application used to design the paint brush attachment. The images in the figure capture the final design for the paint brush attachment.

Figure 15 & 16: Final design ready for printing with cost included

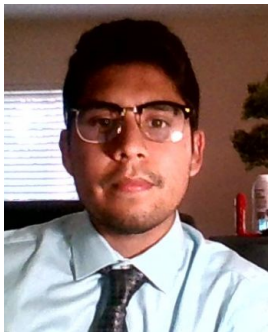




Acknowledgement

The authors wish to acknowledge Dr.Riche and Dr. Lei Wei for their counsel throughout Senior Design 1 and 2. The authors would like to thank Professor Ronald F. DeMara, Professor Zakhia Abichar and Professor Chinwendu Enyioha for agreeing to sit on our project review board. We would also like to thank ourselves for the hardwork and dedication that went into our project.

Biography

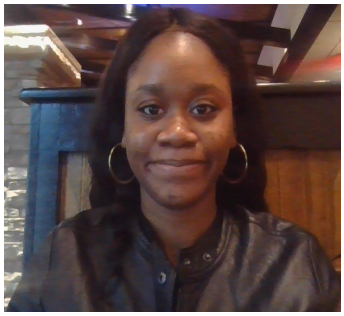


Alonso Ninalaya is a senior studying Computer Engineering at the University of Central Florida. He will graduate on August 1st. His current and previous software experience

includes working with front-end frameworks and hosting applications in the cloud, from an internship helped him with the understanding of this new industry called computer vision. He plans to begin a career in the industry.

Alan Azargushasb is a senior studying Electrical Engineering at the University of Central Florida. Alan will graduate aug 1. He plans on getting a job in the Electrical Engineering industry.

Trudy Ani-Asamani is a senior studying Electrical Engineering at the University of Central Florida. She graduates on August 1st 2020 and hopes to inspire change in the technology industry of Africa.



Chedlyne Valmyr is a senior studying Computer Engineering at the University of Central Florida. With her previous work experience as a Software Engineer at a startup company. She hopes to obtain a position as a Front-end software engineer after graduating.

REFERENCES:

[1]https://www.quora.com/How-can-I-detect-an-object-from-static-image-and-crop-it-from-the-image-using-openCV

[2]https://docs.opencv.org/trunk/d4/d73/tutorial_py_contours_begin.html

[3]https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html

[4]https://datacarpentry.org/image-processing/05-creating-histograms/

[5]https://www.raspberrypi.org/forums/viewtopic.php?t=6986

[6]https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/#:~:text=The%20Raspberry%20Pi%20is%20a,languages%20like%20Scratch%20and%20Python.

[7]https://stackoverflow.com/questions/52179821/python-3-i-am-trying-to-find-find-all-green-pixels-in-an-image-by-traversing-al

[8]https://youtu.be/D93iQVoSScQ