Final Project Report

RBE3001 Team 5

Nathan Rosenberg
Robotics Engineering
Worcester Polytechnic Institute
Worcester, Massachusetts
narosenberg@wpi.edu

Alexandra Wheeler
Robotics Engineering/Computer Science
Worcester Polytechnic Institute
Worcester, Massachusetts
awheeler2@wpi.edu

Walter Gallati
Robotics Engineering
Worcester Polytechnic Institute
Worcester, Massachusetts
wggallati@wpi.edu

***Abstract*—In a modern industrial setting, an automated sorting systems ability to pick up a variety of objects and sort them by their weight could prove to be immensely important. This project aimed to use a thee degrees-of-freedom (DOF) robotics arm to sort objects by weight and color. The goal of this project is to complete the objective of sorting objects in the workspace. The overall program includes PID, force detection, camera vision, trajectory generation, forward and inverse position kinematics and forward and inverse velocity kinematics.**

***Index Terms*—Robot arm, kinematics, design, RBE3001, Matlab, Robotics**

I. Introduction

The system consists of a 3DOF robotic arm. Each link has AS5055A Magnetic Joint Angle Encoder and a TAL220 Parallel Beam Load Cell to allow for closed loop control. To control the arm it has a Nucleo-F746ZG Dev Board. The arm communicates over SPI to the application Matlab to receive commands and send packets. The arm also has a gripper at the tip of the end effector to manipulate objects in its workspace. To determine what is in the workspace there is a webcam that can send live images to Matlab. The arm is shown in figure 1.

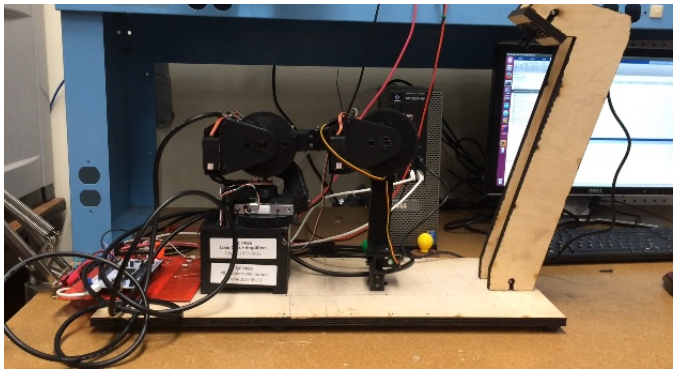

Fig. 1. Final Version of the Arm.

The arms end task is to sort a series of objects. The objects are to be sorted by color (yellow, blue or green) and weight (light or heavy). The arm is to determine the position of an object, determine it's category, pick it up and move it to its designated spot. Afterwards, the arm will repeat until there is no more objects on the board.

II. Methodology

A. Implement Force Sensing

The goal of this step is to get the torque sensors working and communicating properly with the Matlab code. The first step is configure the Nucelo communication to include joint torque sensor data in the status command. This will have the ADC read the three corresponding input channels corresponding to the pre-amplified load cell signals. This data is then averaged to smooth the force readings. Afterwards the sensors are calibrated and include an offset value that is mainly the weight of the arm and interferences from the sensor. A scale is included to convert the adc readings to engineering units.

Force is then calculated from the torque readings using the transposed Jacobian as shown in equation one. These forces are then put onto a live 3D plot of the arm, with vectors that are scaled proportionality to the magnitude of the applied force.

$$F_{Tip} = (jacobian^T)^{-1} * \begin{bmatrix} torque1 \\ torque2 \\ torque3 \end{bmatrix} \quad (1)$$

B. Object Manipulation and Identification

The gripper then needs to be configured. The gripper needs to be mounted onto the end of the end-effector of the arm. The gripper is connected to 6v off the power supply and a has a common ground with the Nucleo board. The code is then configured to be able to open and close the gripper. The arm is then programed to close the gripper on the heavy object and sent to go to several positions recording the joint torques and force vectors. The camera is then modified to detect the centroids of up to three different colors.

C. Sorting System

This part of the lab included getting the arm to sort the objects.The arm has to pick up an object determine the color and weight then sort the object. Overall the arm needs to have a smooth trajectory as it moves about. During the demonstrations, only one of each color can be placed on the sorting area at the same time. So there is only ever one color of each object in the workspace, the camera vision will not need to detect multiple of the same color.

The arm starts with having the gripper open in blank position. The arm can start at any angle but cannot block

the view of the camera. The first step of the program is the camera. The camera will use three color HSV masks to determine the positions of the objects. It will obtain the one closest to the tip of the end-effector and remember the color of the object. Afterwards the object position is passed to the trajectory generation. This will use the trajectory generation function to get the end-effector to move to the object position. Once the arm is in position it will close the gripper.

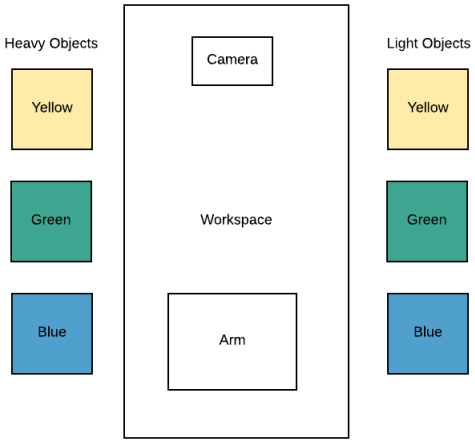

Fig. 2. Diagram of Sorting Methodology.

Once the object has been picked up it will lift the arm to weigh the object. Once the weight is determined it will move the arm to a predetermined location for the objects classification. When the arm has reached its position it will open the gripper to release the object into it's position. The arm will then rerun the program, finding the object closest to it.

III. Results

A. Implement Force Sensing

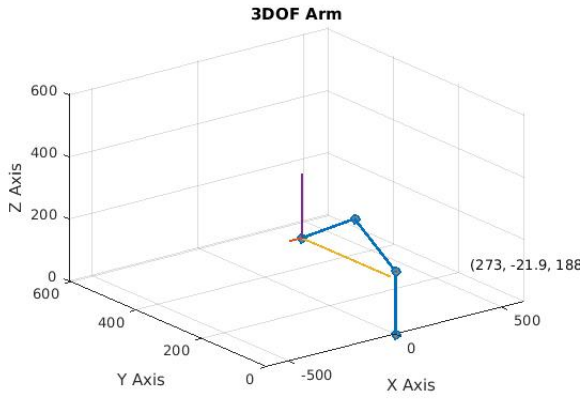

Fig. 3. 3D Plot of Applied Forces on Arm.

The force vector is calculated at the end effector of the arm. To do this, the Jacobian is transposed, and the inverse is multiplied by the instantaneous torques.The torques are of each joint from the strain gauges. Equation one shows how to perform these calculations.

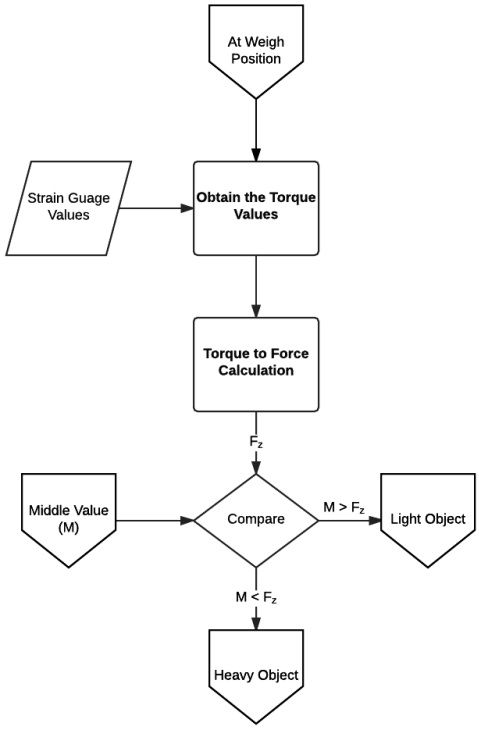

Fig. 4. Flowchart of Object Detection.

Once the force can be calculated from the arm, it can be displayed on the previously made 3D plot. By getting the force calculations in real time the force vectors, of the x, y and z directions, can be applied and shown. The figure below shows these vectors at an arbitrary configuration with an arbitrary amount of force applied. The purple vector is the z direction, red is x and yellow is y. All of these vectors lengths are proportional to the amount of force.

B. Object Manipulation

Once the gripper was able to move around the objects, it was able to pick up the heavy object. The arm was brought to two configurations. The first was Link 1 at home, Link 2 vertical, and Link 3 horizontal, essentially the arm is mostly vertical with the last link at a right angle. The second configuration is Link1 at home, Link 2 horizontal, and Link 3 horizontal, with the arm being straight out. The strain gauge values and forces were collected at both configurations as shown in the tables below. The values are somewhat inaccurate due to variation the force sensors have either due to condition of the wires or jerky motion.

The arm was then set to move through several points recording position, force and magnitude of force. The arm moved through arbitrary points to obtain these calculations.

TABLE I
GAUGE VALUES AT TWO CONFIGURATIONS

	Configuration One	Configuration Two
Link 1	-43.3781	-43.2511
Link 2	5.3549	1.1571
Link 3	5.1595	-0.5402

TABLE II
FORCE VALUES AT TWO CONFIGURATIONS

	Configuration One	Configuration Two
Link 1	-3.5098	-8.4493
Link 2	-12.0139	-6.8225
Link 3	-1.2205	0.1237

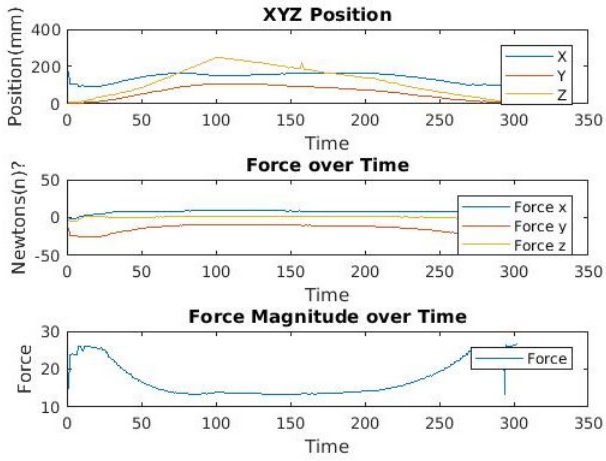

Fig. 5. Plot of Position and Force Vector over Time.

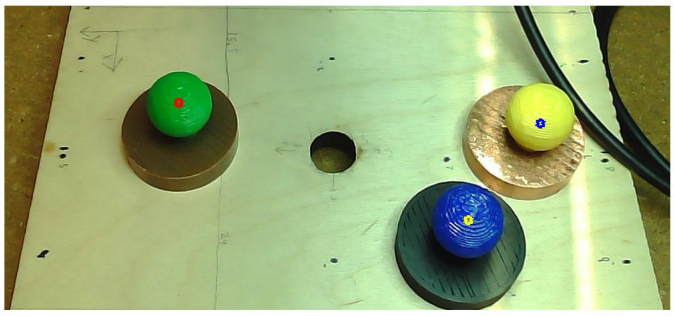

Fig. 6. Image from Camera with Identification of Three Objects.

C. Object Identification

The camera had to be reconfigured from previous labs to allow for three color masks to identify three separate colors. The image prints out to a figure on Matlab and overlays the centroids. It displays a yellow one on blue object, a blue one on a yellow object, and a red centroid on a green object.The arm was then moved to weigh the object. The arm was then moved to hold the heavy weight and light weight, to find out the force vector for each weight. The arm was moved so link one was in home position, and the second and third were put in to approximately forty-five degree angles. Teh valeus were taken from the second link. This during testing was an easy to get to position that was close to where objects are picked up, and a position that gave consistent values. At this position there would be a variance of about .2, the heavy weight was measured at 8.4519 and the light weight is 5.1449. These values every time ended up being stable, with no outliers.

IV. DISCUSSION

A. Implement Force Sensing

Force sensing for each joint was added to the status command. The nucleo code has a rolling average over ten values to try and stabilize the values coming from the sensors. A scale factor is included to get the values to a normal range. The given scale factors had been included but the original data wasn't similar so the values were not put into standard engineering units.

When the raw values after averaging on nucleo were outputted in Matlab there was variation in the results. The values in the nucleo code included offset from the arm standing straight up and then multiplied by a hundred. The values were multiplied to put them into manageable units. Then for the Matlab code, the values were then averaged another 5 times to allow for consistency. When using the combined force values of the Jacobian, the values would be inconsistent, often with changing sign values when parts of the force vector were near zero. As the arm would approach an extreme on any axis the values would get get unstable. This was fixed by only using the second strain gauge to get the force values.

B. Object Manipulation

A new server command is created for the gripper. This will allow for the gripper to open and close with commands from Matlab. The nucleo code will read the first value of a stream of bits, and depending on if it is a one or zero it will open or close the gripper. The program will start off with the gripper in the open position. This will allow for the arm to not have to worry about whether the gripper is open or not when the arm is moving towards the object. Once the arm is at position it will close around the object. The arm will then open when the object is sorted. Then the cycle will repeat since the gripper will be in the open position.

The gripper doesn't have issues with picking up objects, if the motion is too jerky it can drop the heavy objects. To combat this rubber bands were put around the tip of the gripper to provide more surface area fro gripping the objects. This made the arm not drop the heavy objects. Even when there is jerky motion as the arm is moving down.

The values around the heavy object are consistent with motions of torque and forces at different configurations. Between table I and table II, the arm shows the values increase with as the position of the arm moves out. So in configuration two which is at a increased x position the arm reads higher values as there is more weight on the strain gauges. The graph of

the arm values moving through different set points does not provide anything conclusive. Except that force increases when the arm is at a higher acceleration. As well as the force z shows the most variation. This is showing correctly since the arm should have the most variation in z. The inconsistencies in the x and y are likely due to the strain gauges. The stain gauges provides to much variation.

C. Object Identification

To obtain the image, the camera takes a screenshot. Using the Matlab crop tool, it reduces the image to only view the active sorting area. This reduces processing time and mitigates the possibility of false positives from nearby colored objects. From there, the program utilizes one of the three color masks. It goes through the blue mask, green mask, and yellow mask, converting them to black and white images highlighting possible objects. The color mask was determined using the Color Threshold application in Matlab, utilizing the HSV color space. HSV was used due to ease of creating the color mask, it allowed the team to determine a color mask quickly. This can be used to modify the program to sort other colors, increasing versatility.

Taking the black and white images, the built in function $bwareafill()$ is used to fill in the object in case of issues with lighting. $regionprops()$ is then used to reduce the number of observed objects to only objects the general size of the ball target. After going through the three masks, the program calculates the centroids of each item in relation to pixels.

From there, the centroids are converted into xyz coordinates for the arm using the $mn2xy()$ function. Due to issues with the camera offset and scale values are then added to these coordinates. These coordinates are then bundled into an array with the object color and passed to the arm function. The arm will then determine which of objects is closest to the arm and select it to be picked up. To improve the loop rate of the code, the camera is not used until the arm is ready to sort another object. Should the camera not find anything, it returns an empty array, does not pass anything to the arm, and continues searching.

Due to consistency of the force values a threshold value was put in place to detect weights. The better way to due this would be to use euclidean distance from a predetermined weight of heavy and light. This threshold value, placed in the middle of the weights ended up working everytime.

D. Sorting System

The software is designed around an arm class, camera functions and a main script. This allows for concise and easy to read code. Any functions that relate to the arm are in the arm class, everything from setting the PID, getting the angles or any of the position and velocity calculations. The main camera functions has its own script for the conversion of camera coordinates to arm coordinates, detection of objects and the three color masks.

The main script is the code includes the sorting methods. It takes the functions from the arm class and the camera allowing it to perform the tasks. The code was broken up like this to allow for the code to be organized and easier to read. This allows the software to versatile and easily modified without creating a new script for each new function. This encapsulation makes it so the main script can use the functions and data from the arm class without necessarily knowing the actual states and methods the functions use. So one function may use another two or three in the arm class, but the main script only needs to call one function.

The main loop starts with the arm moving to a starting position. Once the arm is out of the way, the program utilizes the camera to find the objects. The object is determeind by getting up to three positions of objects and then finding the magnitude of the vector between each one and the current position of the arm. Of the vectors the minimum is calculated and that is then the object to pick up. Essentially, this is the object closest to the arm at its current position when the camera takes it's snapshot. If two objects end up with the exact same distance, it will prioritize the blue, then green, and then finally the yellow object.

Then using trajectory generation to move the gripper to position. It will go to the XY position above the object then drop down onto the object. Afterwards, it will pick up the object, weigh it in it's configuration. Then depending on its classification move it to the correct spot as shown in figure 2. All movement of the arm will be using trajectory generation, to allow for more precise movement. The overall function is shown in the flowchart of figure 7. Demonstrating how the arm moves from one point of the program to the next.

Part of the goal in this program is simplicity and re-usability. The program should be overall short and easily versatile allowing for this to change if needed. The program should be able to work if another color or weight were to be added without drastic changes to the code. Minimal hardware changes were added such as the rubber bands and securing the camera. This allowed for us to focus on the code and make it easier to add other objects. To add another color to this program, all it would need would be a color mask function and some lines in the camera function as well as adding a spot for the sorting algorithm to place it in. To add another weight would just need to add another threshold value to compare the objects too and more spots for the sorter to place the objects in.

The arm does end up having a few issues. Due to the cropping of the camera, the arm can detect the green and blue objects that had been placed down. Depending on how the object had been placed down, the ball of the object can be seen in the camera frame. This can cause issues leading to there being multiple objects of the same color on the field. The camera code doesn't know how to deal with this and will at times think the object is outside the workspace of the objects due to the identity of two objects. This is fixed by checking if the set point of the camera is in the workspace where the objects should be. If the object is not in the workspace it will move on to the next object or wait until a sufficient object has been determined.

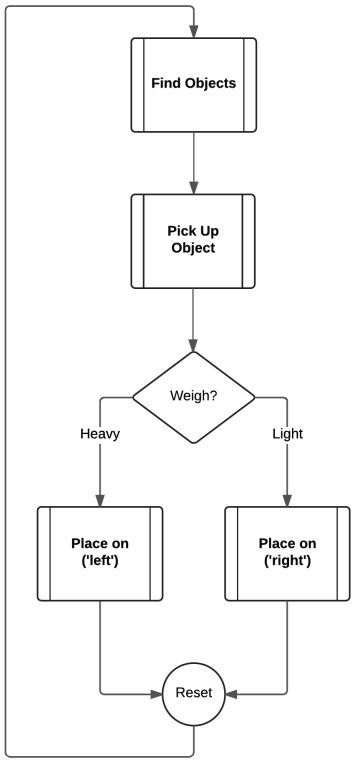

Fig. 7. Flowchart of Overall Program.

V. Conclusion

Overall the robot performed according to specification. It was able to pickup and identify yellow, green, and blue objects, weigh them, and place them in their respective locations. This was done through a combination of kinematics, dynamics, and blob detection using a camera. Each object's position detected with the camera was translated from X, Y pixel coordinates to X, Y, Z coordinates in millimeters in relation to the base of the arm.

This technique, while in theory worked, was flawed in practice. The lens of the camera was warped and not consistent across its field of view and such would distort coordinates of the objects as they moved away from the center of the image. This would cause the end-effector to go to a position not consistent with the actual position of the object. This could be fixed by manually correlating the end-effector position with the position of the centroid from the camera and using an inference transform technique to better calculate the actual position of each object.

Another software architecture improvement would be the standardization of function inputs when it comes to points and vectors and their orientation, i.e. column vs. row. Currently we have this set up for half of our functions and the other half as individual inputs, i.e. (X, Y, Z) vs $(\begin{bmatrix} X \\ Y \\ Z \end{bmatrix})$. Doing this would help shorten the code and make it easier to work with.

Utilizing other measures for the weight detection could be put in place make this system easier. Utilizing other force sensors could help due to the variation. The code could have also averaged more, but resulting in delays in processing. The arm could have also used euclidean distances in weighing the object, but due to simplicity and processing time this was not put in place.

The arm ended up working well with it being simple. The use of the arm class made all the functions neat and organized. The minimal modifications to the hardware allowed for us to focus on keeping everything simple. Not many big changes are needed to add in new objects or weights, and this can be done easily to make this project versatile0 and expandable.

Acknowledgment

We would like to thank the following people for their help: Professor Gregory Fischer, Kevin Harrington, Nathaniel Goldfarb, Gunnar Horve and all the SA's for their support and guidance throughout this project.