Lab 5: Final Project

James Kradjian
Robotics Engineering Major
Worcester Polytechnical Institute
Worcester, MA
jvkradhian@wpi.edu

Jonathan Redus
Electrical & Computer Engineering Major
Worcester Polytechnical Institute
Worcester, MA
jrredus@wpi.edu

Yil Verdeja
Robotics Engineering and Electrical & Computer Engineering Major
Worcester Polytechnical Institute
Worcester, MA
yaverdeja@wpi.edu

This lab demonstrated the advanced concepts of computer vision by tracking and manipulating of objects of various colors and densities. Matlab was used to process images taken by a PlayStation Eye web camera and translate those images into instructions for the robot, as well as interpret data from the strain gauges for force sensing at the end effector. Although vision related decisions were functionally accurate, force related decisions were inconsistent and inaccurate, thus the robot was unable to sort by different weights. Overall, apart from the force measurements, the robot was able to automatically identify an object based on its color, reach out for that object, pick it up, and sort it based on a combination of its color and weight.

Keywords—automation, computer vision, robotic manipulator, sorting by color and weight

I. Introduction (*Heading 1*)

This project accumulates everything that has been implemented so far in the robotic manipulator system. In the previous lab, the Jacobian was calculated, as well as the forward and inverse velocity kinematics of the robotic arm, which allowed to identify the singularities of the arm. Finally, a numeric solution to the inverse position kinematics problem was implemented which utilized Taylor series approximations to reach a certain place in the workspace.

Building from the previous labs, this lab investigates an automated robotic sorting system, where the robot has to localize certain objects within its workspace using a camera, pick them up, classify them based on weight and/or appearance, and release them within a predefined area. The objectives for this lab are as follow:

- Use vision to identify and localize objects
- Control the robot's end effector
- Measure and display applied force vectors at the end effector
- Sort different objects base on their weight and color

This lab has three main sections: (1) creating a force sensing system, (2) adding vision based tracking, and (3) creating an automated sorting system to sort objects in the workspace by weight and color. Finally, using the numeric inverse kinematic algorithm implemented from the previous lab, a dynamic camera tracking was implemented to follow the object in the camera workspace in real-time.

II. Methodology

This section will go through the steps that were taken to complete this lab. It built on previous labs by implementing force sensing at the end effector through use of the strain gauges. It also added vision-based tracking, and motion. Finally, all of these parts were combined to create an autonomous sorting system which could organize objects in the workspace based on the color and weight of the object.

A. Experimental Material

- A computer running Ubuntu
- An ST Nucleo-144 (STM32F746) development board with ARM microcontroller
- An RRR arm, controlled by the Nucleo board
- A webcam, mounted above the workspace
- 2 x banana connector to banana connector cables
- 2 x micro-USB to USB cables

B. Force Sensing

In order to determine the weight of an object picked up by the end effector, the force at the end effector needed to be calculated. To do this, the torque at each joint was measured by getting the status of the robot. Once measured, the torques were multiplied by a manipulated jacobian matrix, to give the force vector.

1) Average Torque Readings

In order to get an accurate torque reading, the values detected by the sensors needed to be averaged to reduce noise. This was done in the firmware, since it was faster to have the firmware read the torque multiple times and then average it rather than having the Matlab call several packets. The averaging was implemented by summing the ADC readings for a hundred readings, then dividing the readings by a hundred, whenever the status server was polled. To increase the accuracy of the torque readings, one hundred data points were taken and averaged. Any readings with significant error would have a marginal effect on the final torque value.

2) Calibrate the Joint Torque Sensors

Next, the torque sensor readings had to be changed from ADC values into actual torque values. Given a calibration

curve, a formula to calculate the torques of each joint was derived as seen below:

$$\tau=(x-y_0)/k \ [Nm] \quad (1)$$

where τ is the torque, x is the ADC reading, y_0 is the offset, and k is the scaling factor. K was set to approximately 178.5, and y_0 was the reading of the ADC when there was no torque on the sensor, which was different for each sensor. To find y_0, the robot was placed in an overhead configuration, and the raw torque, in ADC counts, was determined to be: y_0 = [515; 486; 507] for the first, second and third joint in the robotic manipulator.

3) Calculate the Force at the End Effector

Once the torques were known, the force on the end effector could be calculated. By calculating the statics (statics3001 function) for the system, the following relation was derived:

$$F=J_p^{-T}(q)*\tau \quad (2)$$

Where F is the force vector, $J_p^{-T}(q)$ is the inverted transpose of the Jacobian at joint angles q, and τ is a matrix of the torques at each joint. The jacobian was calculated using the jacob0 function derived from the previous lab, while the torques at each joint were found as described in previous sections. With these values, the force vector was calculated.

C. Vision-Based Tracking

Before this lab, a camera was attached to the workspace, so that objects within the workspace could be identified, located, and eventually grabbed by the robot. The camera was set up, so that locations within the camera frame could be transformed to the workspace of the robot. A function was created to locate certain objects by color, and the robot was made to move to the location of the object. Finally, a dynamic version of object tracking was implemented, which could reach for an object, even as the object moved around the workspace.

1) Camera-Robot Setup

The change from camera coordinates to workspace coordinates largely depended on a provided function called *mn2xy*. First, the camera had to be calibrated using the *calibrate_camera* function. This function took several points from the image that were provided by the user, correlated it with points in the workspace, and created a transformation. By using the mn2xy function with the calibrated pixels of the calibrated coordinate system, it could be given points within the camera coordinates, and would return points in the workspace coordinates of the robot.

However, the results from *mn2xy* were not fully accurate, so another function was created that tweaked the values given by *mn2xy*. It divided the workspace into four quadrants, and depending on what quadrant the object was in, the x and y values were scaled and offset by differing amounts. This provided enough precision to successfully locate objects in the workspace.

Fig. 1. Transforming Centroid Location from Image pixels to Robot Coordinate System

```
B = transMN2XY(centers2);
Y = transMN2XY(centers3);
G = transMN2XY(centers4);

function centerXY = transMN2XY(centerMN)
    % Transforms a Matrix of MN Centers to XY centers
    % CenterMN is the centers of the colors in MN configuration
    dataChange = [];
    sz = size(centerMN);
    if (sz(1) == 0)
       centerXY = centerMN;
       return;
    end
    for i = 1:sz(1) % Goes through every row
        change = mn2xy(centerMN(i, 2), centerMN(i,1)); % Changes the rows MN orienation to XY
        % if xy is in the robot workspace (wood), then add to the matrix
        disp(i);
        if (inWood(change(2)))
            change = mnxyCal(change);
            dataChange = vertcat(dataChange, change); % Adds to the matrix
        end
    end

    centerXY = dataChange; % Returns the whole change

end
```

2) Object Localization

Next, the objects had to be identified within the image. This was achieved by filtering the image by specific color, then identifying the centers of the circles representing the objects.

Filtering was done for each individual color: blue, green, and yellow. Each color was filtered in whatever color space had the most distinct representation of the color of the object. For all cases, blue, green, and yellow, a Lab Color Space was utilized for filtering each color as it provided the best results under the specific environmental conditions. The filter produced a binary image, where the objects were filled as white and the rest of the space was black.

Next, the centers of each object in the workspace were found. This was achieved by going through each of the three binary images produced by color filtering, eroding, and morphing, and then using the *imfindcircles* function the centers and radii of the objects were identified. For each object, a n by 2 matrix was returned, where n is the number of objects of a specific color found in the workspace. These centers were then passed into the program to be used later in moving towards the objects. The centers of each object located within the image was represented using an m by n location, where m and n are pixels in the image of row and column respectively. Using the mn2xy function, the centroids of each object in the workspace was returned in terms of the robot coordinate system.

Since the camera should only find objects within the robots wooden countertop, a boolean function *inWood* was written to return whether a specific coordinate was within the robot platform workspace.

3) Reach for an Object

Next, the robot was made to move to one of the objects found using the object localization function *objLoc*. This was done by passing the x and y values that the localization function returned for a specific object, along with a constant z value, to a movement function. This movement function used

a quintic trajectory to move from the current position to the desired position.

4) Dynamic Object Tracking

As a bonus, a script *DynamicLocObj.m* was created which repeatedly checked the position of the object as the robot moved towards it. This used a modified version of the jacobian-based numeric solution to the inverse kinematics problem implemented in lab 4. It operated by multiplying the inverse Jacobian by the desired change in position, or the difference between the desired position and the current position. This gave the approximate change in joint angles needed to move to the desired position. The difference for this lab was that the desired position was found from the object localization function, so that every iteration of the algorithm updated the location of the object, if it changed, and if it did not move, the arm simply moved to a more accurate position.

D. Automated Sorting System

The final step of the lab was to make a system which could automatically sort objects within the workspace. This required a few additions to the system: gripper control, object weight sensing, and object sorting.

1) Control the Gripper

To control the gripper, an additional server had to be created in the firmware as seen in figure 2. When it received a packet, it would open the gripper if the first term of the packet was a 2, and would close it otherwise. Additionally, the servo motor had to be declared in the code, so that it could be controlled. Then, in Matlab, a function was created to communicate with the firmware server. It would send a packet, telling the servo to open or close based on a boolean input to the function.

2) Weigh Objects

Object weighing used the force detection at the end effector, implemented in section 2.1. It found the z magnitude of the force at the end effector and checked to see if it was over or under a certain threshold. If it was greater, the object was categorized as lightweight (since the force would be in the negative z), and otherwise, the object was categorized as heavy.

Fig. 2. Gripper Server in Firmware to actuate the gripper servo

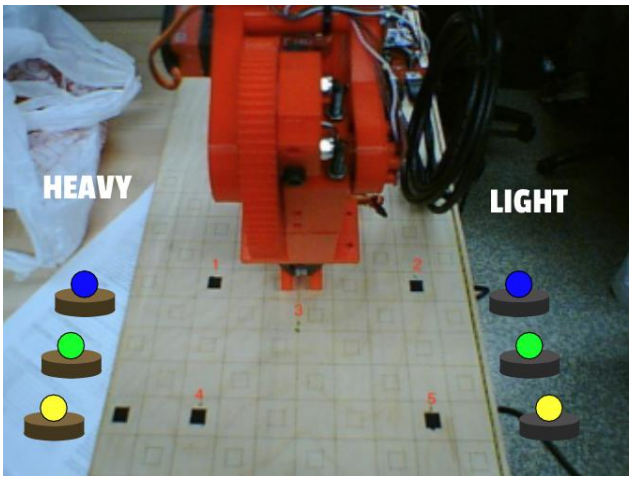
```
void GripperServer::event(float *packet){
        // Close if packet(0) = 0
        // Open if packet(0) = 2

        // Limited range so it doesn't
        // over extend the servo gripper

        if (packet[0] == 2) {
          myGripperServo->write(0.8);
        } else {
          myGripperServo->write(.05);
        }

}
```

Fig. 3. Example Color and Weight Sorting System



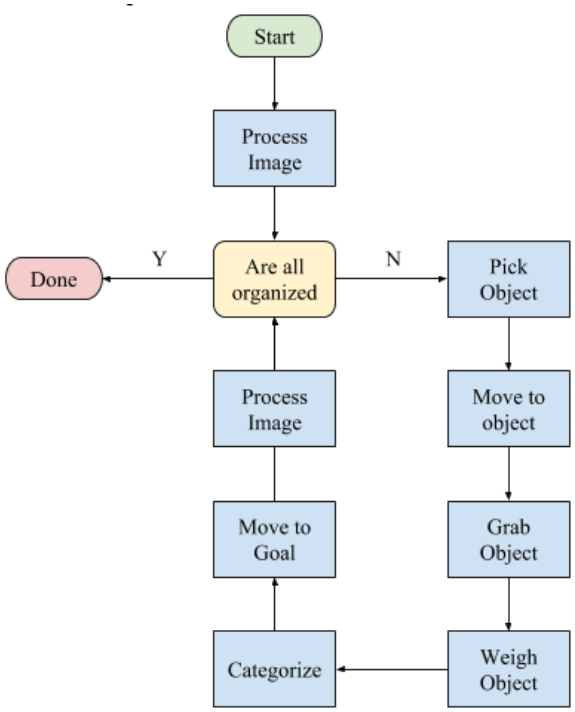
3) Pick and Sort Objects

Next, a sorting function had to be created, so that the robot would know where to move the object in the automated sorting system. This took the color and weight of the object, and assigned each combination of color and weight to a specific location off of the board, but still within the reachable space of the robot. figure 3 shows an example of how the items were sorted in the system implemented.

4) Automated Sorting System

With motion to the location of an object, gripper control, object weight categorization, and object sorting, the final automated sorting system was implemented. Figure 4 shows the general program flow, where blue rectangles are processes, yellow rectangles are decision points, and solid arrows represent the code path.

Fig. 4. Flow Diagram of Automated Sorting System



The "Process Image" procedure was the object localization function, described in section 2.2.2. This passed arrays of the blue, green, and yellow arrays to a function which removed objects that were not on the wood from the arrays. These modified arrays were passed to a function which would exit the loop if there were no objects in the arrays, or would return the position of an object if there were still objects in the arrays. The position was then used to move to a point slightly above the object. Then, the arm descended, the gripper grasped the object, and the arm lifted back up. Next, the object was weighed, using the function described in section 2.3.2, and the weight category, along with the color, was used to find where to put the object. Then the object was moved there, a new image was taken, and the code looped back to organization check.

III. Results

A. Force Sensing

The force sensing for the robot did not work because of reading inaccuracies and a failure to account for the weight of the robot in the statics calculation. Data was gathered to find the errors and their possible causes, and is in Appendix A, along with the procedure used to gather the data.

1) Average Torque Readings

Averaging the torque readings successfully reduced the noise of the torque sensor data. It reduced the standard deviation of the torque readings, taken over a 10 second period, to less than 0.005 Nm in most cases. This is excellent, since this would be less than a 0.05 N standard deviation for a force applied directly at the end effector.

2) Calibrate the Joint Torque Sensors

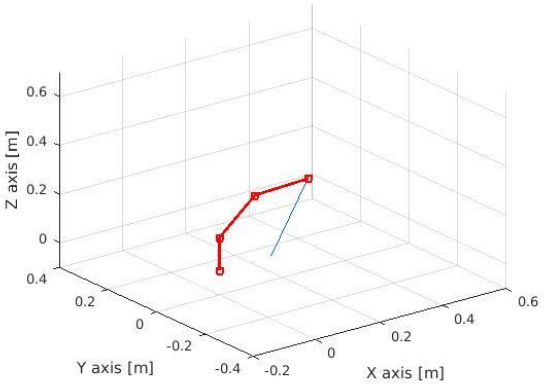
The joint torque sensors were supposed to allow the computer to read the exact torques being applied. However, the calibration of the torque sensors did not account for the weight of the robot, thus it was not properly offset at each point. Also, the calibration process was challenging, as it required moving the robot into a configuration where no torque would be exerted at any joint, and any slight deviance from that configuration could cause errors in the calibration. Thus, the calibration was not very accurate, and the torque reading at some points exceeded 0.5 Nm, even with no force on the end effector.

3) Calculate the Force at the End Effector

The force calculation at the end effector was also inaccurate. The standard deviation of the force calculated in a certain configuration exceeded one newton, while the measured difference between the light and heavy object was frequently less than 0.1 N. Furthermore, the measured forces for the same object differed dramatically depending on the configuration of the robot.

Despite the errors with the readings, a 3D plot was created, which represented the arm as a set of lines, and an arrow representing the force vector positioned at the end effector, as shown in figure 5. The arrow moved and scaled as the calculated force moved and changed magnitude.

Fig. 5. *Three Dimensional Plot of the Arm and Force Vector*



B. Vision Based Tracking

The initial implementation of the vision tracking system was not accurate enough to grab an object successfully, so additional adjustments were made.

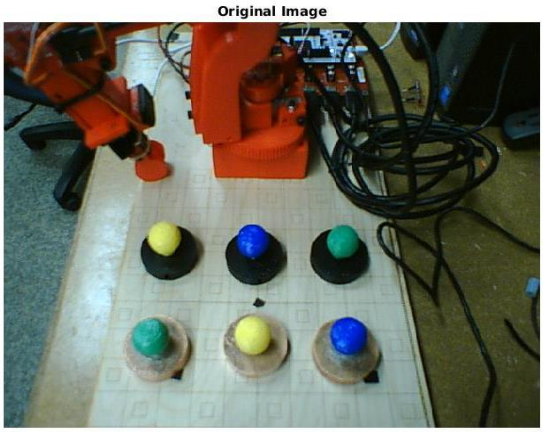
1) Camera-Robot Setup

In the camera setup, an additional function was added, because the provided function did not give a sufficiently accurate position. Though the additional function increased the accuracy, it did occasionally missed objects in the workspace. However, the resulting values were accurate enough to demonstrate the functionality of the system.

2) Object Localization

The *objLoc* function is a function that identifies specific objects based on color (blue, green and yellow) and returns their corresponding centroids in x and y values of the robot coordinate system. The object location function takes in two parameters: a snapshot from the camera *camFrame* and an image figure *imPlot* in order to redraw the image on the figure without creating a new plot each time. To completely understand the results of the object localization function, the steps to filtering, and finding the centroids will be explained, with their results. Figure 6 represents the original image that was sent through to the function. As seen by that snapshot, there are 6 objects, with 3 distinct colors, and 2 different weights (black = light, gold = heavy).

Fig. 6. *Original Image Captured for Object Localization*



Through every iteration of the *objLoc* function, a new snapshot has to be taken from the camera to update the location of the centroids in the workspace. Once completed, the program separately filters out individual colors from the workspace using a function generated by the image segmentation app in Matlab that utilizes the LAB color space. Each type of color space was considered, however, the LAB color space returned the best filtering of the colors due to the light intensity changes in the lab environment. For every individual color filtering, after obtaining the color-filtered image, it then removes all connected components that have fewer than a specified number of pixels from that binary image. The number of pixels used was 400, as it was expected that the whole centroids would be located.

After removing all the noise in the binary image, the second step was to completely fill enclosed objects with white space using the *imfill* function. Sometimes under certain light sources, the filtering of a color could work on most parts of that object. Other parts would not be recognized by the filtering mask as it would be either too dark or too light. Finally, once each circle was filled, due to the effects of noise removing, some circles may be smaller than expected. In order to fix this, using the *bwmorph* function with a *thicken* parameter, it would thicken objects by adding pixels to the exterior of objects.

These three steps in segmenting an image resulted in the binary images seen in figures 7 through 9.

Fig. 7. *Blue Binary Filtered Image*

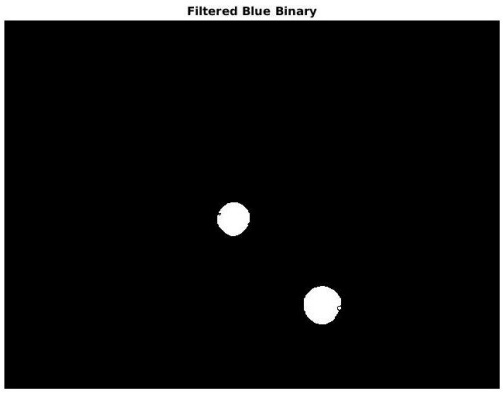


Fig. 8. *Green Binary Filtered Image*

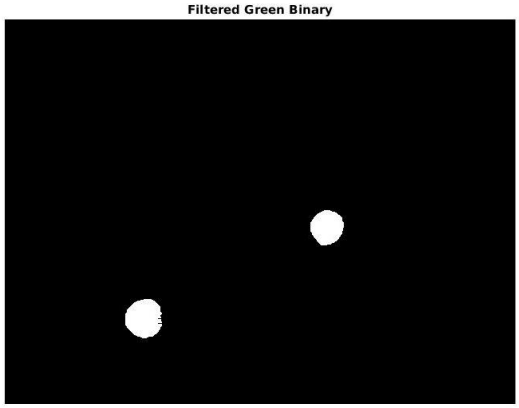


Fig. 9. *Yellow Binary Filtered Image*

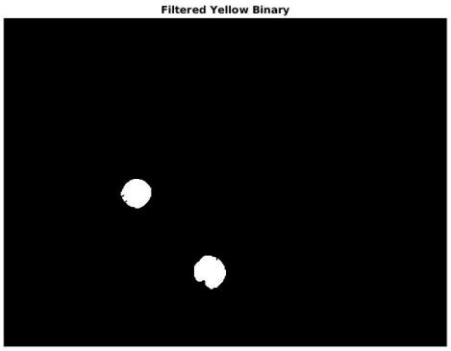
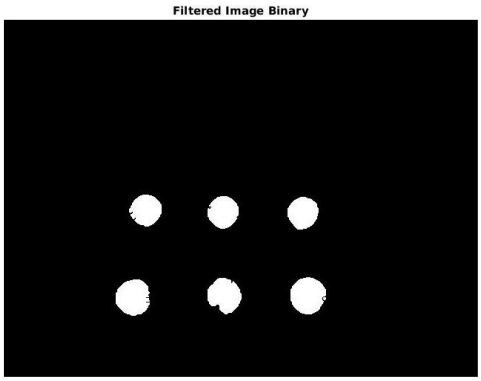


Fig. 10. *Final Binary Filtered Image*



Adding all of those images together results in binary image of the original snapshot. As seen in figure 10, each object closely resembles a circle, and is relatively close to the original image.

Finally, for all three images in figures 7 through 9 that filtered out each color, the centroids of each object were found using the *imfindcircles* function with a parameter that would only search for circles in a specific range. This function returns an array of matrices that contains the centers and the radius of each circle found. Once the centroids are located, and the radiuses are calculated, the colored objects in the original image are overlapped with a distinct colored perimeter to show that the objects were found in the cameras workspace. As seen in figure 11, the original image has each colored object marked with a specific colored-perimeter.

Fig. 11. *Final Binary Filtered Image*

After changing the pixel values of the centroids to x and y values corresponding to the robot workspace, their centroids are outputted. The following points correspond to filtering of the image in figure 6:

B = [(-0.0509, 0.0274) (0.0593, -0.0577)];

Y = [(0.0679, 0.0191) (-0.0446, 0.1026)];

G = [(-0.0303, -0.0507) (0.0616, 0.0911)];

This clearly shows that for each color, two objects were found in a certain position inside the robot's workspace.

3) Reach for an Object

Once the setup was modified, the robot was able to successfully reach for an object most of the time. In the process of accomplishing this, several issues were fixed. First, several functions were not receiving arguments in the right units or frames of references. This was because several functions were not created with the convenience of using the whole system in mind. Also, the PID constants needed to be modified. This was a minor issue in past labs, but since the robot needed to be able to hold objects and move precisely, the issue was fixed. However, these issues were all dealt with by the end of the lab.

4) Dynamic Object Tracking

The dynamic object tracking was also able to successfully reach for an object most of the time. However, the system had two issues. First, the loop was rather slow, which meant that the response time of the robot was not quick enough to be used for many practical applications. The other issues was that the robot would sometimes move between the object and the camera, so that the robot would not know where to move. However, the function successfully demonstrated the basic functionality of the concept.

C. Automated Sorting System

When creating the automated sorting system, no new errors were introduced to the system. However, since this system included work from all the other sections, the errors mentioned previously affected the functionality of the sorting system.

1) Control the Gripper

The gripper code worked adequately for the final version of the lab. When it was initially created, however, it did not, because the example code provided, which was used, had a delay greater than the delay that the packet processor tolerated. Once that was resolved, the physical gripper was turned past the safe limit for it, and thus it could not close fully. However, since this was not noticed for a while, the gripper was damaged and had to be replaced. Once that was done, and the code fixed so that it would not happen again, the gripper worked properly.

2) Weigh Objects

Weighing the objects was not successful, because the force sensing system did not work, as described in section 3.1. This is because the weight categorizing function depended on the force calculation function. In the end, the weight categorization was effectively random.

3) Pick and Sort Objects

The pick and sort system worked most of the time by the end of the lab. It was not completely accurate, since it inherited the errors that the reach for an object function had. However, it was sufficient to demonstrate the concepts of the lab, and could be further refined if needed and time was available.

4) Automate Sorting System

The automated sorting system inherited the errors of the weight categorization and pick and sort functions. Thus, it could do everything it intended to do most of the time, except for weigh the objects, which it never did accurately.

IV. Discusion

A. Force Sensing

The force sensing portion of this lab did not work. This is because the sensor readings did not provide accurate enough data to correctly calculate the force, and because the weight of the robot was not taken into account when calculating the statics equation.

The torque sensor readings generally had a standard deviation of less than 0.005 Nm. While this seems sufficient for the purposes of this lab, it was not. The standard deviation of the force was frequently less than 0.05 N, but in some cases exceeded 0.1 N. This was problematic, because the measured difference between the light and heavy objects was less than 0.1 N in many cases. Thus, more accurate torque sensing would have needed to precisely measure the weight of the robot.

Also, the statics calculation did not account for the weight of the robot. This meant that the robot would calculate very different forces for the same object, depending on the configuration of the robot. Therefore, for accurate readings throughout the workspace, the statics equations would have to be recalculated.

B. Vision-Based Tracking

There were two issues common to all systems which used vision tracking. The first was improper conversion from the camera to the reference frame of the robot. A function was provided which was supposed to do this transformation. However, it did not work properly, so another function was added to compensate for the remaining error. However, it would have been better to modify the provided function to correctly do all conversions, since it would have been easier to follow. This was not done because the initial function was rather difficult to understand, so it seemed safer not to modify that which was not understood.

The function created to compensate for this error could have been improved as well. It divided the space into quadrants, and modified the positions depending on what quadrant it was in. However, this did not deal with the underlying issue of why it was distorted. It may have been that perspective was not accounted for correctly, or that the lens of the camera created a distortion. Since the cause was not known, though, quadrants dealt well enough with the issue. Thus, in future work, it may be helpful to find the source of

the distortion, so that more accurate equations can be used to change the position values.

The second issue was that the conversion from the camera to the workspace created an origin in front of the robot, while the reference frame for controlling the robot had an origin at the base of the robot. This lead to a great deal of confusion when trying to plan the motion of the robot. Functions were created to deal with this issue, but were not put in the best place in the code. Had the function for translating from the camera to the workspace also moved the origin to the base of the robot, the whole problem would have been dealt with in one location, rather than needing to insert the transformation functions throughout the code.

1) Dynamic Object Tracking

The dynamic object tracking system had two additional issues: it operated slowly, and sometimes the arm would move between the camera and object, confusing the system.

The first issue was probably caused by inefficient code, most likely the vision processing system. However, this is not surprising, since vision processing requires many calculations. Thus, a possible solution would be to reduce the amount of code that will run. This means can be two things: removing unnecessary code, and cropping the image, so that fewer values are processed. Either, or both, should help speed the system up.

The other issue is more complicated, because the system needs to be able to distinguish between the arm obscuring an object and an object being removed. This could be solved by adding another camera, to see the blind spots created by the robot motion. However, this would be more expensive and complicated, and could make the code run slower. Thus, this solution should only be implemented if absolutely necessary.

C. Automated Sorting System

The final automated sorting system introduced no new errors. However, it inherited errors from previous sections, which meant that it did not always work. It still operated consistently enough to demonstrate the functionality of the system.

V. Conclusion

This lab demonstrated knowledge of statics, computer vision systems, and robotic systems by implementing a force sensing system, vision-based object tracking, and an autonomous object sorting system. Throughout the lab, concepts from older labs were used as well, such as motion planning and trajectory generation, position and differential kinematics, and coding practices.

Even though there were a issues through the course of this lab that set us back (inconsistent and inaccurate readings from the load cell and force sensing), it wasn't enough to affect the results of the lab. Nevertheless, we were able to get everything complete, and our hardships helped us to further understand and think about new ways to get around the problem. A suggested solution would have been to use centripetal forces and angular velocities to calculate the forces at the end effector. However, there wasn't enough time to implement this feature to the robot.

Appendix A: Force Testing Results

Due to problems with the force testing results, tests were run to see where the issues arose. To do this, force readings were taken in multiple positions while holding either nothing, a light object, or a heavy object. For each position and weight, the torque and force readings were recorded in a loop for ten seconds. After the loop ran, statistics were calculated from the data: the minimum, mean, maximum, and the standard deviation. All of the statistics are recorded below, along with images of the robot configurations:

Fig. 12. *Position 1*

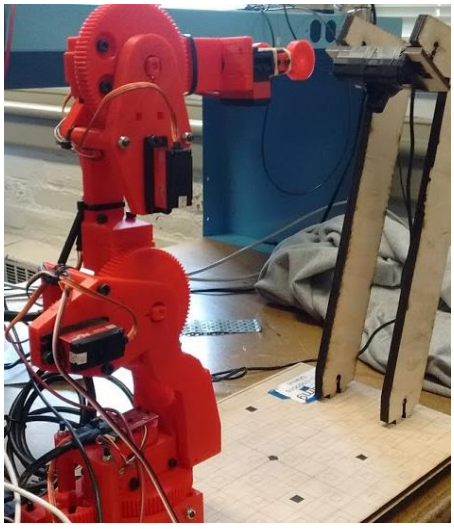


TABLE I. NO OBJECT, POSITION 1

	Torque Readings		
	J0 Torque	***J1 Torque***	***J2 Torque***
Min	0.1737	0.1849	0.0392
Mean	0.1799	0.199	0.0448
Max	0.1849	0.2073	0.0504
Std	0.0021	0.0037	0.0023
	Force Readings		
	X Force	***Y Force***	***Z Force***
Min	-0.3072	1.0505	-0.9282
Mean	-0.2703	1.0883	-0.8807
Max	-0.2327	1.1206	-0.7686
Std	0.0143	0.0127	0.0278

TABLE II. LIGHT OBJECT, POSITION 1

	Torque Readings		
	J0 Torque	***J1 Torque***	***J2 Torque***
Min	0.1681	0.1961	0.0224
Mean	0.179	0.2061	0.0285
Max	0.1849	0.2129	0.0448
Std	0.0023	0.0038	0.0039

	Force Readings		
	X Force	***Y Force***	***Z Force***
Min	-0.2728	1.0149	-1.0861
Mean	-0.177	1.0811	-1.0129
Max	-0.1393	1.1171	-0.8645
Std	0.0226	0.0139	0.0366

TABLE III. HEAVY OBJECT, POSITION 1

	Torque Readings		
	J0 Torque	***J1 Torque***	***J2 Torque***
Min	0.1625	0.1961	0.0448
Mean	0.1706	0.207	0.1022
Max	0.2185	0.2185	0.112
Std	0.0075	0.0053	0.0086
	Force Readings		
	X Force	***Y Force***	***Z Force***
Min	-0.6796	0.9679	-0.8615
Mean	-0.6233	1.0152	-0.5955
Max	-0.2914	1.2958	-0.5076
Std	0.0492	0.0438	0.0555

Fig. 13. *Position 2*

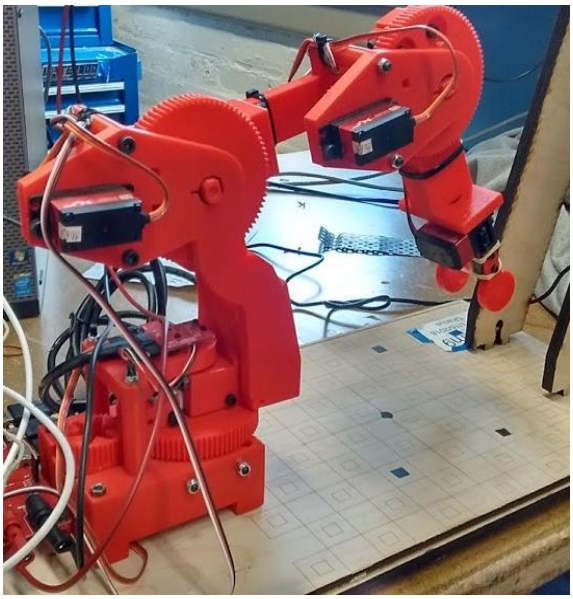


TABLE IV. NO OBJECT, POSITION 2

	Torque Readings		
	J0 Torque	***J1 Torque***	***J2 Torque***
Min	0.1344	-0.028	0
Mean	0.14	0.4003	0.0069
Max	0.1681	0.4146	0.0896
Std	0.0043	0.0573	0.0112

	Force Readings		
	X Force	***Y Force***	***Z Force***
Min	-2.0683	0.5424	-1.1944
Mean	-1.9891	0.5662	-1.1061
Max	0.4109	0.6682	0.8534
Std	0.321	0.0159	0.2623

TABLE V. LIGHT OBJECT, POSITION 2

	Torque Readings		
	J0 Torque	***J1 Torque***	***J2 Torque***
Min	0.1344	0.3978	0
Mean	0.1376	0.4014	0.0051
Max	0.1401	0.4034	0.0112
Std	0.0028	0.0027	0.002
	Force Readings		
	X Force	***Y Force***	***Z Force***
Min	-2.0257	0.5427	-1.1841
Mean	-1.9995	0.5568	-1.124
Max	-1.9648	0.5698	-1.0585
Std	0.0157	0.0113	0.0223

TABLE VI. HEAVY OBJECT, POSITION 2

	Torque Readings		
	J0 Torque	***J1 Torque***	***J2 Torque***
Min	0.1232	0.3641	0
Mean	0.1238	0.3685	0.0035
Max	0.1401	0.3922	0.0112
Std	0.0025	0.0048	0.0033
	Force Readings		
	X Force	***Y Force***	***Z Force***
Min	-1.959	0.4988	-1.0742
Mean	-1.8493	0.5046	-1.0197
Max	-1.8015	0.5706	-0.9464
Std	0.0278	0.0102	0.0341

Fig. 14. *Position 3*

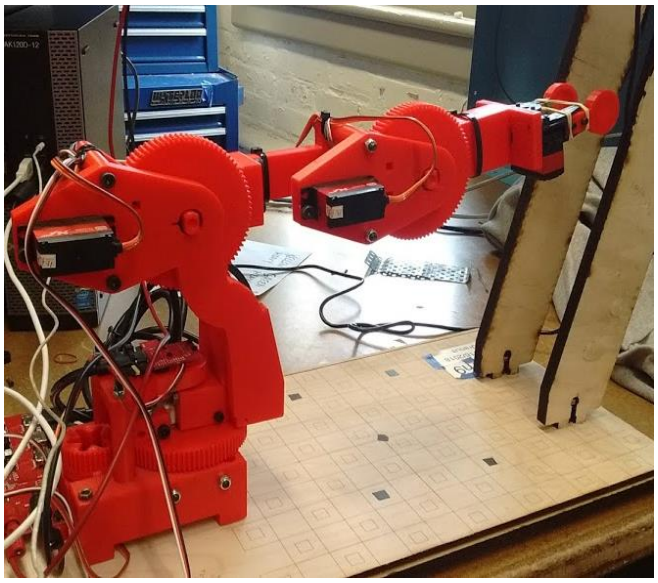


TABLE VII. NO OBJECT, POSITION 3

	Torque Readings		
	J0 Torque	***J1 Torque***	***J2 Torque***
Min	0.1232	0.5714	0.1008
Mean	0.1285	0.5757	0.1028
Max	0.1288	0.5826	0.1064
Std	0.0014	0.0028	0.0027
	Force Readings		
	X Force	***Y Force***	***Z Force***
Min	1.1453	0.4181	-32.37
Mean	1.4639	0.4992	-26.642
Max	2.3523	0.606	-24.453
Std	0.1763	0.0417	1.1317

TABLE VIII. LIGHT OBJECT, POSITION 3

	Torque Readings		
	J0 Torque	***J1 Torque***	***J2 Torque***
Min	0.1232	0.5714	0.0896
Mean	0.1263	0.5722	0.0966
Max	0.1288	0.577	0.1177
Std	0.0028	0.0019	0.0037
	Force Readings		
	X Force	***Y Force***	***Z Force***
Min	0.9412	0.4101	-28.849
Mean	1.4098	0.5115	-26.762
Max	1.7413	0.6045	-23.171
Std	0.1386	0.0423	0.9363

TABLE IX. HEAVY OBJECT, POSITION 3

	Torque Readings		
	J0 Torque	***J1 Torque***	***J2 Torque***
Min	0.112	0.5546	0.0784
Mean	0.1128	0.5605	0.0893
Max	0.1232	0.5938	0.1232
Std	0.0025	0.0054	0.0056
	Force Readings		
	X Force	***Y Force***	***Z Force***
Min	0.5493	0.393	-27.181
Mean	0.8713	0.466	-24.77
Max	1.1222	0.5402	-22.199
Std	0.1055	0.0335	0.8232