Force Sensing, Camera Vision, and Item Sorting

Gabriel Entov
Department of Electrical and Computer Engineering, Robotics Engineering
Worcester Polytechnic Institute
Worcester, MA, USA
gentov@wpi.edu

Parker Grant
Department of Electrical and Computer Engineering, Robotics Engineering
Worcester Polytechnic Institute
Worcester, MA, USA
pagrant@wpi.edu

R. Ryan O'Brien
Department of Electrical and Computer Engineering, Robotics Engineering
Worcester Polytechnic Institute
Worcester, MA, USA
rrobrien@wpi.edu

***Abstract*—This experiment involved the use of a 3 DOF arm to pick up, weigh, and sort colored balls.**

I. Introduction

This experiment is the fifth and final part of the RBE 3001 curriculum at WPI. A 3 DOF arm was used, controlled by a Nucleo board. Firmware was coded in C, while control software was written in MATLAB. Communication servers were implemented allowing for data packets to be sent back and forth in real time. Prior experiments involved incorporating forward and inverse kinematics both algebraically and geometrically to control the arm. Additionally, Jacobians were used to calculate differential kinematics, allowing for more precise motion control. This final lab looked at both force sensing using static analysis and vision based tracking.

Statics are a fundamental aspect of robotic mechanisms and can be used in various applications. One such application is force sensing at the end effector of a mechanism. The relationship between joint torques, $\boldsymbol{\tau}$ and the Jacobian matrix $J_P^T(q)$, shown in (1), can be used to find forces on the end effector, F_{tip}. The arm used in this experiment was equipped with strain gauges at each joint to measure torque, which allowed for dynamic weight measurement in any orientation.

$$\tau = J_p^T(q)F_{tip} \tag{1}$$

Once force sensing was implemented, the remainder of the lab was related to computer vision and vision based tracking. A mounted camera was used to take photos used for image processing and object location. Various MATLAB functions were used to translate pixel coordinates from the pictures to position in the task space.

The final task in this lab was to locate and sort blue, green, and yellow balls with either metal or plastic bases according to both weight and color. This was accomplished by locating balls with camera vision, then weighing them using force sensing. Joint applications of these techniques are used widely in robotics. Their usefulness is shown in this experiment.

II. Methods

This experiment was based on three main components; force sensing, vision based tracking, and automated sorting. The methodology used to achieve each are described in the following section which were

A. Force Sensing

Each of the joints on the arm were equipped with a strain gauge for sensing torques applied on them. The STATUS server implemented in a prior lab was able to send the position, velocity, and torque readings, although the torque values obtained were initially insufficient. In order to improve accuracy, the torque gauges were oversampled by a factor of 100, and multiplied by a provided constant, 178.5, to convert them to Newton meters, in the firmware before being sent to MATLAB.

Once torques were calibrated, the force present on the end effector could be calculated using (1). This relationship between joint torques and the jacobian of the system allowed for simple calculation of forces, which was performed in the sttics3001 function. In order to demonstrate these forces, a live stick model of the arm was modeled with a force vector scaled by magnitude on the end effector in the function statics3001Test. This was used to dynamically test and show forces detected.

B. Vision Based Tracking

Vision based tracking was implemented using a webcam connected directly to the control software in MATLAB. to accomplish this, the webcam was positioned so that it could see almost all of the robots work area. The reference frames of the camera and the robots workspace were not aligned, and so a conversion between these two reference frames was created. This conversion was created by manually selecting the fiducian points based on an image of the empty workspace taken by the webcam. The function which did that was named camera_calibration.

The calibration data from the camera_calibration function is later used by mn2xy, a function which converts between the reference frames of the camera and robot. Once this was done, several functions were created to process the images taken by the webcam. The first step of this image processing was

separating the image into several different colors. To do this, a function was made to test which parts of the image fell within a certain color range range, then exported a binary mask which held this information. Three of these functions were actually made: one for checking the blue, green and yellow color ranges. These were named createBlueMaskUCL, createGreenMaskUCL, and createYellowMaskUCL, respectively. Each of these masks were then cleaned using a series of different operations.

These masks were cleaned of noise by running the edge detection, fill, erode and dilate functions on these masks. A single function called procIMG was created to handle all four of those functions conveniently at once. Once the noise filtering is complete, the centroids of any blobs are calculated for each color's mask. The function needed to do this actually already exists in MATLAB, and it is called regionprops. This provides us with the xy coordinate of the colored objects within the camera's reference frame, so by converting these points using the mn2xy function, the position of the tracked object relative to the robot's base reference frame was found. To make this process easier to run, all of these commands were put into a script called CameraVision.

C. Automated Sorting System

Force sensing and vision based tracking were combined to create an automated ball sorting system. The script moveToObject endlessly searched for and sorted objects using a number of functions. All motion while sorting was performed using linearInter, a function that took two points and moved the arm along a linearly interpolated path of 50 points. This function incorporated a .05 second delay between each point to slow down and smooth motion.

Two functions, gripperOpen and gripperClose, were created to allow consistent control of the gripper. Due to limited resources, a motor was used to control the gripper instead of a servo, so these functions were time dependant. To ensure the gripper was never opened or closed erroneously, the functions returned a 1 or 0 indicating whether the gripper was closed or not. This value was stored in a variable, isClosed, that was checked before each loop of the function. One last function, weighObject, was also created to allow sorting by weight. This function simply read the forces on the end effector when called, and returned a value, isHeavy, telling whether a weight was detected.

MoveToObject consisted of three nested try/catch statements that checked global variables containing positions of detected balls. CameraVision was called outside of the statement to update global position variables. The script then attempted to locate a green object by scaling the coordinates, if present, in the mn2xy function. At this point, if no green coordinates existed, an assert statement threw an exception and moved into the catch, which contained another try/catch looking for blue objects. This behaved the same way, and had a yellow try/catch in the catch.

If any number of objects were detected in any stage, a for loop would execute once per object of that color before moving on. At this point, the arm moved to an object and closed the gripper, then lifted it to a specific position, [0 95 0], at which weight values were calibrated to ensure consistency and accuracy. Once weighed, balls were placed in locations dependant on both color and weight. Heavy objects were placed the the 'right' of the robot, chosen arbitrarily and indicated physically on the board, while light objects were moved to the 'left'. Green objects were placed at a 50 degree angle, respective to joint 1, blue objects at a 70 degree angle, and yellow at a 90 degree angle.

D. Additional Functionalities

Two additional functionalities were implemented on the arm, a 3D live-updated stick model, and live tracking of colored objects. The 3D live-updated stick model is a 3 dimensional representation of the robotic manipulator, that can show its position in real time. The use of simplified 3D representations rather than detailed CAD models allowed MATLAB to draw the robot very quickly and efficiently. The live tracking was done in a script called LiveTrack. This script very simply located objects and immediately moved to a position above them. All other functionalities were cut out to increase speed.

III. Results

There were many hardware and software limitations that prevented consistent behavior in some functionalities in this experiment. Due to the heavy reliance between tasks, this in some cases compounded and required specific compensation. Regardless, the arm was able to complete all required tasks sufficiently.

A. Force Sensing

It is important to note that the force sensors did not behave consistently. When moving from one position to another, the voltage did not change on the amplified output of the strain gauge. Figures 1 and 2 show the robot in one position with the oscilloscope reading of the strain gauge voltage. As the arm is moved into another position, the voltage on the strain gauge does not change. Although the strain gauge voltage should be miniscule, the amplified output is being measured, and therefore there should be a noticeable difference between the two positions. This should be especially true as the two positions chosen are two of very different torques.

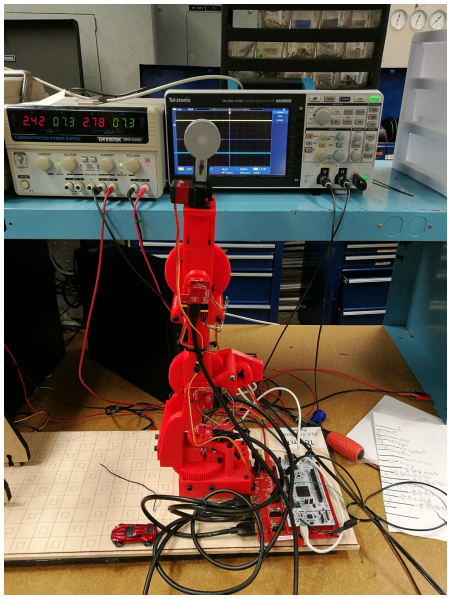

Figure 1: Arm in Upright Position

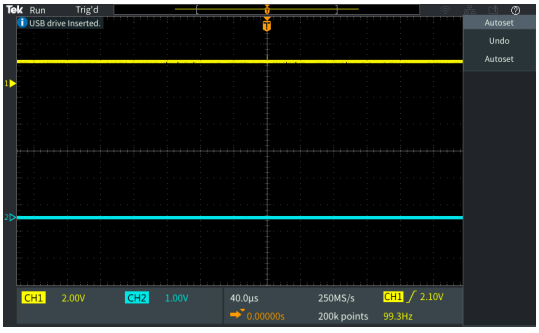

Figure 2: Strain Gauge Amplified Voltage

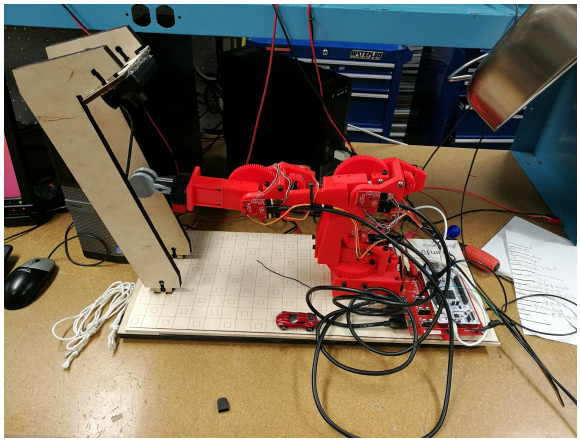

Figure 3: Arm in Extended Position

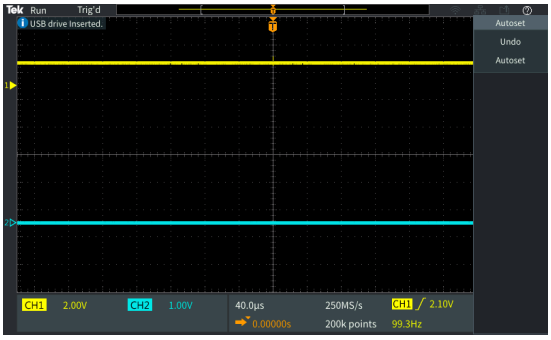

Figure 4: Strain Gauge Amplified Voltage

Regardless of the inconsistency, the arm was effectively able to weigh objects held, provided the weighObject function was calibrated since the last board reset. Larger forces did not always result in larger detected values, but there was a measurable difference.

B. Vision Based Tracking

Vision tracking came with its own difficulties. Originally, our image cleanup function would work very intermittently. The main malfunction observed would be the disappearance of blobs. For some reason blobs would have a seemingly random chance of actually surviving through the image processing function. It turned out that this issue was due to inconsistent edge detection. Sometimes, the edge detection would miss some parts of the blob's edge, resulting in gaps in the outline. These gaps prevented the blob from being refilled later on, hence causing their mysterious disappearance. This was resolved by changing the edge detection mode from the default "Sobel" detection method to "Canny", which does a better job of maintaining a continuous edge.

The mn2xy function had incorrect coefficients for scaling, and used centimeters as opposed to millimeters as the rest of the calculations did. Additionally, the x and y coordinates returned by mn2xy were centered at the base joint, whereas the home position was centered directly below Joint 1. Once this was accounted for, the robot was able to track relatively well close to the base and in the middle. The problem came when moving closer to the edges and further away from the base. This is due to the fact that the camera is at an angle, meaning that the further the object is from the camera, the less accurate the camera can track it. In other words, the camera sees a trapezoid instead of a rectangle due to its orientation.

To compensate for this, the object was placed in two different positions along one of the edges of the platform. The camera's approximation was read, and then the real distance was measured. Then a mathematical relationship was established to help account for variance in the y direction due to variances in the x direction. Additionally, the camera was not perfectly centered, and so an offset had to be applied to the values to allow for the mathematical relationship to work on the other side of the platform. After all of these relationships were programmed, new grippers were 3D printed

in order to account for the very minor offsets to small for the previously aforementioned linear approximation to handle.

C. Automated Sorting System

Once all the mentioned accomodations were made, sorting was able to be consistently performed despite inconsistencies. The arm occasionally missed balls, but was usually able to correctly locate and pick them up on the second or third try in these cases.

D. Additional Functionalities

In order to achieve live tracking at decent speeds, many operations had to be cut out of object locating. Applying binary masks to images and filtering for centroids of objects was a lengthy procedure, taking over a second each time. In order to speed this up, only one color, green, was searched for. Applying less filtering in the procIMG function was tested, but resulted in inconsistent and inaccurate tracking. Rather than interpolating, direct movement was used to further speed up reactions. These measures resulted in position updates 2-3 times a second, which was sufficient provided the hardware limitations.

IV. Discussion

This experiment covered a large breadth of topics, from forward and inverse kinematics to statics to computer vision, and integrated them all into a single task. Any task in the field of robotics will similarly require several several different techniques to accomplish, given the nature of robotics.

Each part of the lab built on top of the last in order to achieve the final result. The strain gauges performed admirably in the beginning of the lab, but then became very unreliably. In fact, it is possible that what was thought to be working properly, in fact was not working the way it was intended to.

Moving on from the strain gauges, using the color thresholder in MATLAB proved to be incredibly useful for setting up color thresholds to make distinguishing between colors in the workspace much easier.

Once the colors were reliably found, moving the arm to the object was less intuitive than expected. It was important to remember that because the camera was mounted at an angle, the image that the camera sees is at an angle and forms a trapezoid instead of a rectangle, which makes it much more difficult to convert from pixels to xy location on the platform.

For future iterations of this course, it would be extremely beneficial to mount the camera directly above the task space, and not at an angle. This would save the students a lot of time on the lab. If the space is an issue, then correctly programming mn2xy to account for the trapezoidal shape would also help. Leaving it as it is now is a good programming exercise for the student, but can be frustrating if one does not realize why it is happening.

V. Conclusion

The object of this lab was to develop a robotic system capable of sorting objects by weight and color. To do this, the robot needed to be capable of using computer vision technology to locate objects and convert coordinates from the image to coordinates in the robot's joint space. Additionally, it used the camera vision to determine the colors of each object. After locating and picking up objects, the robot then weighed them and sorted based on weight and color.