Sorting Objects Based on Color and Weight Using a 3 DOF Robotic Arm

Michael Abadjiev
Robotics dept.
Worcester Polytechnic Insitute
Worcester, MA, USA
msabadjiev@wpi.edu

Qingyuan Chen
Robotics dept.
Worcester Polytechnic Insitute
Worcester, MA, USA
qchen3@wpi.edu

Nicholas Johnson
Robotics dept.
Worcester Polytechnic Institute
Worcester, MA, USA
nbjohnson@wpi.edu

***Abstract*—This lab gave us an opportunity to apply what we have spent the past 7 weeks learning about in class to implement a robot that can sort objects by weight and color, using trajectory planning techniques and robot kinematics to follow smooth paths. We chose to additionally implement remote control of the robot using a Raspberry Pi to allow control via a smartphone and a 3D space mouse.**

I. Introduction

In this project, we implemented functionality for a 3 degree of freedom robotic arm to be able to sort objects based on weight and color. In doing so, we applied all of the material covered in class, including forward and inverse position and velocity kinematics, trajectory generation, and some basic computer vision. Our system included a Matlab script running on a lab computer, communicating via USB with a robot control board on a 3 degree of freedom robotic arm. Additionally, our system required the use of a Raspberry Pi to emulate an Apple home device that was used to implement voice control of the robot. Finally, we also used a 3D connexion SpaceNavigator space mouse to be able to directly control the motion of the robots tip.

II. Background

For this project, we used the RBE 3001 robotic arm that was provided to us. This is a 3 degree of freedom arm which uses a STM Nucleo as a robot controller, as well as a USB webcam for computer vision. The arm features 3 servo motors to actuate each rotary joint, and a fourth one to open and close the gripper. Each joint also has a 12 bit, 4096 tick encoder on the spur gear used to determine joint position. In order for the control board to communicate with the encoders, they are wired as SPI slave devices that send information back to the Nucleo. Additionally, each link contains a load cell that can be used to measure the torque on the link at any given time. The load cells are each connected as a full wheatstone bridge, and a Analog to Digital Converter on board the nucleo interprets the signal into useable values for computations.

A AS5055A 12-bit magnetic position encoder is used for sensing the joint angle. It communicates with the robot through a 4-wire SPI interface. The Nucleo uses SPI mode 1 and each transmission is a 16 bit packet with the most significant bit first, 1 error flag, and 1 parity bit. The load cell is connected to a differential amplifier, INA826AIDR, with a gain of 411 to magnify the signal to a useable value.

A. Robot Kinematics

The most important aspect of controlling the robot was being able to know its location in the workspace at any given time. This was done using robot kinematics to determine the tip location from joint positions, as well as desired joint positions for a given tip location. Fig. 1 shows the intermediate frames defined following DH-standards in order to calculate the position kinematics.

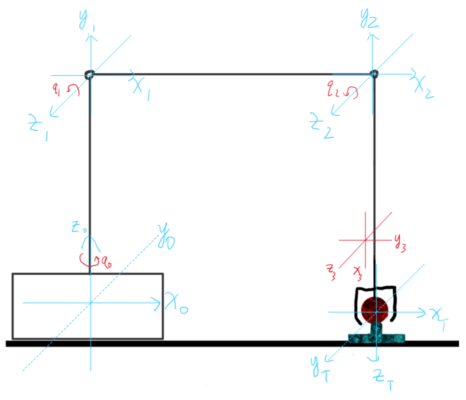

Fig. 1. Frame and Axis definition in X,Z view

Forward Kinematics were used to determine the tip position of the robot based on the joint angles. In our case, the forward kinematics are calculated using a transformation matrix. The matrix, which is constructed following the DH-convention, will transform tip location into the base frame. Table 1 shows the DH parameters.

TABLE I
DH-PARAMETER

Link	Θ	***d***	α	***a***
L0-L1	q_0	L1	90^o	0
L1-L2	q_1	0	0	L2
L2-L3	q_2 - 90^o	0	0	L3
L3-Ltip	90^o	0	90^o	0

The individual transformation function between links is calculated using DH-parameters from Table I and the overall transformation is found by using equation (1)

$$T_0^{tip} = T_0^1 * T_1^2 * T_2^3 * T_3^{tip} \tag{1}$$

The full T_0^{tip} transformation is as follows [1]

$$\begin{bmatrix} C_{1+2}\,C_0 & S_0 & S_{1+2}\,C_0 & C_0\,\left(l_3\,S_{1+2}+l_2\,C_1\right) \\ C_{1+2}\,S_0 & -C_0 & S_{1+2}\,S_0 & S_0\,\left(l_3\,S_{1+2}+l_2\,C_1\right) \\ S_{1+2} & 0 & -C_{1+2} & l_1-l_3\,C_{1+2}+l_2\,S_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

Inverse kinematics are used to determine the joint position given a specific tip position. This can be done through either basic geometry or by using the jacobian matrix. Since our robot arm is quite simple, we used geometric inverse kinematics.

Fig 2 and 3 show the necessary shapes to geometrically determine the inverse kinematics, and Eq 3-8 show the resulting joint positions as functions of the link lengths and desired task space position.

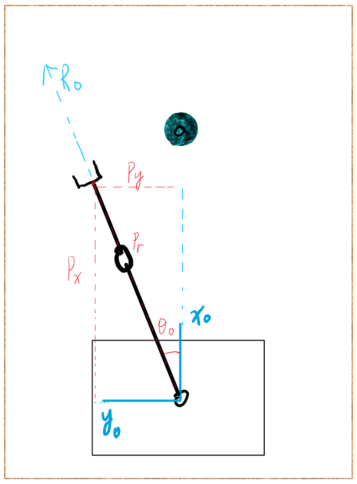

Fig. 2. Inverse Kinematics parameter in X,Y view

$$q_0 = atan2(P_x, P_y) \tag{3}$$

$$P_r = sqrt P_x^2 + P_y^2 \tag{4}$$

$$H = l_1 - P_z \tag{5}$$

$$D = sqrt H^2 + P_r^2 \tag{6}$$

$$q_2 = asin(\frac{l_2^2 + l_3^2 + D^2}{2 * l_2 * l_3}) \tag{7}$$

$$q_1 = acos(\frac{l_2^2 + D^2 - l_3^2}{2 * l_2 * D}) - atan2(P_r, H) \tag{8}$$

[1]in order to shorten eqautions, C_0 and S_0 are used in place of $cos(q_0)$ and $sin(q_0)$ etc. for this document

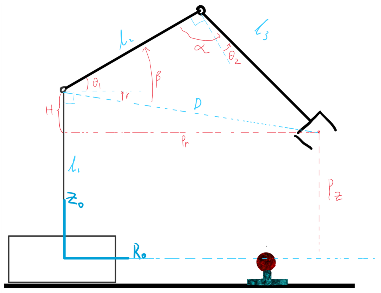

Fig. 3. Inverse Kinematics parameter in R,Z view

The Jacobian is a matrix used to transform tip velocities into joint velocities. It is found by taking the derivative of different parts of the base to tip transformation matrix with respect to time. It is also used to compute the torque relationship between the joints and tip. Eqns 9-10 show the jacobian for our robot arm as well as the relationship between joint velocities and tip velocities.

$$\begin{bmatrix} -S_0\,\left(l_3\,S_{1+2}+l_2\,C_1\right) & C_0\,\left(l_3\,C_{1+2}-l_2\,S_1\right) & l_3\,C_{1+2}\,C_0 \\ C_0\,\left(l_3\,S_{1+2}+l_2\,C_1\right) & S_0\,\left(l_3\,C_{1+2}-l_2\,S_1\right) & l_3\,C_{1+2}\,S_0 \\ 0 & l_3\,S_{1+2}+l_2\,C_1 & l_3\,S_{1+2} \\ 0 & S_0 & S_0 \\ 0 & -C_0 & -C_0 \\ 1 & 0 & 0 \end{bmatrix} \tag{9}$$

The tip velocity can be calculated by $\dot{\vec{x}} = J(q) * \dot{\vec{q}}$

$$\begin{bmatrix} \dot{q_1}\,C_0\,\left(l_3\,C_{1+2}-l_2\,S_1\right)-\dot{q_0}\,S_0\,\left(l_3\,S_{1+2}+l_2\,C_1\right)+l_3\,\dot{q_2}\,C_{1+2}\,C_0 \\ \dot{q_0}\,C_0\,\left(l_3\,S_{1+2}+l_2\,C_1\right)+\dot{q_1}\,S_0\,\left(l_3\,C_{1+2}-l_2\,S_1\right)+l_3\,\dot{q_2}\,C_{1+2}\,S_0 \\ \dot{q_1}\,\left(l_3\,S_{1+2}+l_2\,C_1\right)+l_3\,\dot{q_2}\,S_{1+2} \\ S_0\,\left(\dot{q_1}+\dot{q_2}\right) \\ -C_0\,\left(\dot{q_1}+\dot{q_2}\right) \\ \dot{q_0} \end{bmatrix} \tag{10}$$

The tip force is calculated by $\vec{F} = J(q)^{T^{-1}} * \vec{\tau}$ where $\vec{\tau}$ is a vector of the three joint torques. The resulting symbolic vector is too complicated to include in this document, but can be easily calculated using the jacobian found in Eq 9.

III. Methodology

A. Program Logic

Fig. 4 shows the program logic used to sort objects by color and weight. The robot starts off in a paused state, awaiting commands over Matlab's serial connection with the Rasberry Pi. Once a command has been issued, the first step is to determine the locations of all objects to be sorted using the camera to filter by color. Upon identifying the centroids of objects to sort, the robot then lifts each ball and weighs them one at a time, consequently sorting them into their proper locations.

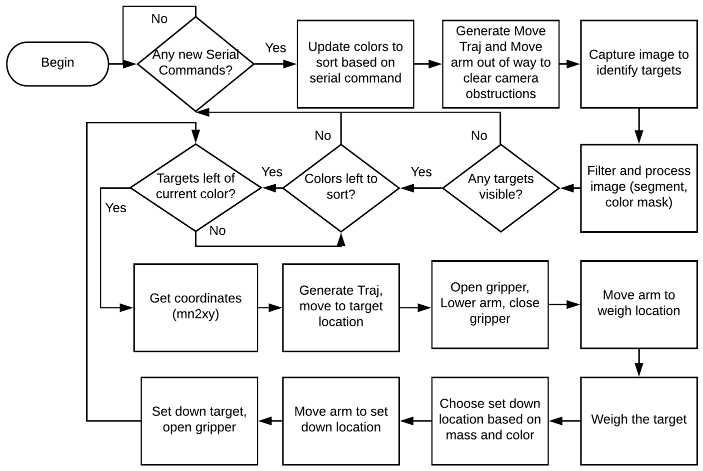
Begin
Any new Serial Commands?
No
Yes
Update colors to sort based on serial command
Generate Move Traj and Move arm out of way to clear camera obstructions
Capture image to identify targets
Filter and process image (segment, color mask)
Any targets visible?
No
Yes
Colors left to sort?
No
Yes
Targets left of current color?
Yes
No
Get coordinates (mn2xy)
Generate Traj, move to target location
Open gripper, Lower arm, close gripper
Move arm to weigh location
Weigh the target
Choose set down location based on mass and color
Move arm to set down location
Set down target, open gripper

Fig. 4. Program Flowchart

B. Computer Vision

A major aspect of this project was using a USB webcam mounted on the frame of the robot to implement some basic computer vision algorithms. The robot had to be able to distinguish between three contrasting colors of ball, and identify the centroids of each ball in order to be able to pick them up. To do this, four image masks were developed in Matlab to first crop the image to contain only the workspace, and then create three new images, one for each color. Once the masks were applied and centroids of the objects were located, it was then necessary to transform the positions from image pixel locations to workspace coordinates. This was done by using five known calibration points in the workspace, and interpolating between those points to determine the location of the balls.

C. Weighing Objects

One of the challenges of this lab was to distinguish between heavy and light objects. The robot arm is equipped with load cells in each link of the arm, however, the readings from the load cells were found to be very unreliable and could not be used to consistently weigh objects. Instead, the torque on each motor was used. Since the arm uses digital DC motors, the PWM value output to the motor will be a relative motor torque. This reading is consistent and has obvious changes for different weights.

D. Trajectory Generation

In this project, it was desirable for the motion of the tip to follow smooth trajectories between any two points. Trajectory generation using a quintic polynomial was implemented to accomplish this. Each time the robot was to move to a new point, a trajectory was generated by setting the desired start and end velocities and accelerations as zero. The desired number of segments between points, and the time to move were determined as functions of the absolute distance between them. This created smooth trajectories that moved the robot at an approximately constant speed no matter the distance to travel. Eqs. 11, 12 and 13 show the polynomial used to calculate smooth trajectories. The coefficients a_0 - a_6 were calculated using Eq. 14.

$$P(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 + a_6 t^6 \tag{11}$$

$$V(t) = a_1 + 2a_2 t + 3a_3 t^2 + 4a_4 t^3 + 5a_5 t^4 + 6a_6 t^5 \tag{12}$$

$$A(t) = 2a_2 + 6a_3 t + 12a_4 t^2 + 20a_5 t^3 + 30a_6 t^4 \tag{13}$$

$$\begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 \\ 0 & 1 & 2t_0 & 3t_0^2 & 4t_0^3 & 5t_0^4 \\ 0 & 0 & 2 & 6t_0 & 12t_0^2 & 20t_0^3 \\ 1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 \\ 0 & 1 & 2t_f & 3t_f^2 & 4t_f^3 & 5t_f^4 \\ 0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} q_0 \\ v_0 \\ \alpha_0 \\ q_f \\ v_f \\ \alpha_f \end{bmatrix} \tag{14}$$

E. Motion Control

To ensure smooth motion of the robot arm, a PID control loop was used. A feed forward value was set to account for the arm weight and then PID values were added on top. Proportional gain (P) was added to get a quicker response time when a move was commanded, Integral gain (I) was added to ensure the target was reached by integrating the error over time. Differential gain (D) was added to reduce overshoot. The initial PID tuning was done in Lab 1 using the Ziegler-Nichols method for initial values and then we proceeded to fine tune the values by analyzing the motion of the arm and adjusting the terms accordingly. This produced quite a smooth motion for the arm without any mass on the end.

F. Raspberry Pi

To add more functionality to our robot a Raspberry Pi (the Pi) was used. The addition of the Pi allowed for remote control of the robot by both button and voice, as well as manual control using a CAD spacemouse. To communicate with Matlab a serial connection was established between the Pi and Matlab. As serial is very rudimentary, a packet system was developed to transfer information between the two devices. Each string of serial commands contained an identifier at the beginning, the information to read, and and end of message character in order to inform the Matlab script what information a specific message contained. Fig. 5 shows this structure, with an example command from the space mouse.

$$\begin{bmatrix} 'm' & 175 & 0 & 0 & 'e' \\ mouseID & xpos & ypos & zpos & endofmessage \end{bmatrix}$$

Fig. 5. Serial packet structure

To have remote and voice control from a phone, software was needed on the Pi to communicate with the phone and interpret its commands. This was accomplished by an open source piece of software called Homebridge. Homebridge is a server that is designed to run on a Pi that emulates an Apple HomeKit device that can communicate with Apple products, in this case, an iPhone. As this software only handles pairing with a device but doesn't actually command anything a plugin was used called cmdswitch. Cmdswitch runs a specified command whenever the home device is triggered. For our use case it called a Python script that then communicated across the serial connection to Matlab to inform the robot of the move it was supposed to execute. For our robot there were four options, sort all of the balls on the field, all blue, all green, or all yellow. Once it finished its task it awaited another command.

G. *Space Mouse*

In order to use the space mouse with the robot, it was necessary to have a driver that capable of interpreting the data from it. As the device is designed almost exclusively for Windows users, the company does not provide an up to date driver for linux machines, like the ones used in the robotics lab. For this reason, the space mouse was instead connected to the same Pi used for the remote control. It was much easier to implement a simple USB monitor using a python script on the Pi that tracked the movement of the mouse. The data that the mouse sends to the computer is in the form of velocities in each direction, or around each axis, so the Pi was used to integrate these velocities with respect to time to determine the position of a virtual cursor in 3 dimensions. This position was then sent through the already established serial connection to Matlab, which moved the robot accordingly

IV. Results and Discussion

Our robot was able to successfully and consistently sort balls by both color and weight. Fig 3 shows the monitoring interface that was displayed by matlab in order to give us an understanding of how the robot was moving and track past motion.

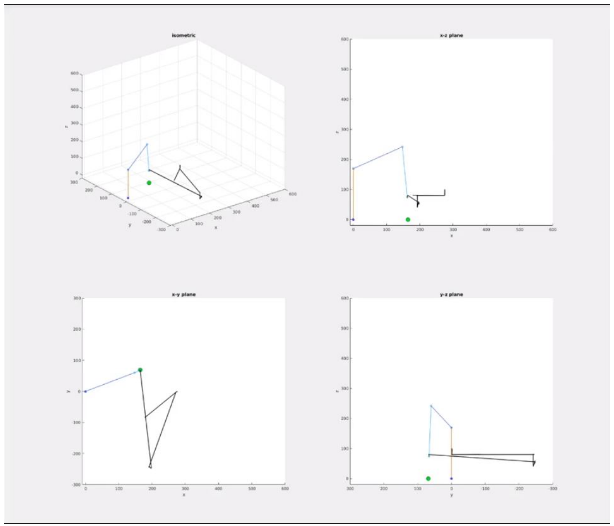

Fig. 6. Robot Monitoring Interface

A. *Computer Vision*

Our main issue when attempting to process images came with the variability in lighting conditions present in the lab. For example, on a sunny day, if the robot was by the window, the camera might process an image differently than at 1 am in outer room with only artificial lighting. In order to mitigate this, we attempted to consistently use the same work bench, and always placed an extra lamp directly above to illuminate the workspace.

B. *Weighing*

When attempting to use the load cells included in the robot arm, we experienced difficulty obtaining consistent enough results to be able to reliably sort balls based on weight for a few reasons. The biggest issue is that the signal output by the load cell is very small, even after being amplified by the INA826AIDR differential amplifier on the with a gain of 411. Such a low signal is extremely susceptible to interference from the environment, like the magnetic fields around each motor and this is also compounded with the length of the wires to make the signal unreliable. Additionally, because of the mechanical hysteresis in the arm, the unloaded reading of the load cell changes dynamically as the robot moves around. This could be solved by implementing some form of live calibration of the load cells, however we decided to instead try a different approach to determining the joint torques using the motor readings, and performing our weight calculations based on these numbers instead.

C. *Mechanical*

There were two main sources of mechanical error on this robot: slop in the gears, and hysteresis caused by friction. In the original design of the robot provided to us, the 3D printed gears had some slop in the connection between the gear and the motor. This made tuning the PID control loop difficult, as the slop caused variability and oscillations. This issue was resolved when new gears were printed that created a better press fit with the servo horn mounted on the motor, however this solution introduced a new issue: friction. With the reduced slop came increased friction in the whole system, which made it difficult to ensure that the robot would always wind up in the same orientation when traveling to the same point from different start locations. For example, when moving to the weighing location, it would sometimes stop short to one side or the other, depending on where it had started. In order to circumvent this issue, the algorithm was modified to attempt to follow the same path to the weigh location each time, so the robot would first move to a point directly below it, then slightly upwards. This successfully eliminated most of the error and the robot was able to accurately weigh and sort items.

D. *Raspberry Pi and Space Mouse*

The serial connection between the Raspberry Pi and the Matlab script was another problematic part of this robot. The main issue was the inherent speed problems with any serial connection, which was further amplified by the fact Matlab on linux cannot exceed a baud rate 9600 bits per second. Coupled with the required matlab computational time, and the faster, but still not perfect, HID connection with the robot, this caused commands issued to or on the Pi to take some time to be executed. This lag was especially noticeable when using the space mouse, which had nearly a 100ms delay before movement would be executed.

E. *PID Tuning*

The motion of the robot was quite smooth with the gains chosen, however, adding weight to the end when lifting a ball caused the motion to become unstable. It was impossible to use the same values for motion while holding a heavy ball and while holding a light ball, as the different weights created different power requirements. To solve this issue, an additional server was implemented on the Nucleo which allowed for on-the-fly changes to the PID values of each joint. This was used to adjust the gain values depending on what mass the robot should be holding. This allowed smoother movement as gains were tuned individually for heavy and light objects.

V. Conclusion

This project required us to apply all of the concepts that were covered in class in order to be able to implement all of the required functionality for our robot. We used what we learned about forward kinematics to be able to track the robot's position. Inverse kinematics and trajectory generation based on a polynomial were used in order to create smooth trajectories between points, and to convert those trajectories from task-space positions to joint-space orientations. Basic computer vision algorithms, along with jacobian-based torque calculations were also used to sort the balls by color and weight. Additionally, we extended the project to implement remote control of the robot via a smart phone or a 3D spacemouse.

VI. Additional Links

1) *Demonstration video:* Youtube Video
2) *Matlab code:* Github repository
3) *Nucleo code:* Github repository
4) *Raspberry Pi code:* Github repository

VII. Team Member Contributions

A. *Primary Code Contributions*

Each of us contributed to all of the code, however listed below are the sections we specialized in.

1) *Michael:*

- Space mouse
- Trajectory planning
- Color sorting
- Image Filters

2) *Qingyuan:*

- Robot Kinematics
- Robot plotting
- Weighing objects

3) *Nicholas:*

- Raspberry Pi remote control
- Robot PID control
- Nucleo communication servers
- Trajectory generation
- Overall program structure

B. *Document Contributions*

1) *Michael:*

- Introduction
- Background
- Methodology
- Results
- Conclusion
- General Formatting

2) *Qingyuan:*

- Background
- Methodology
- Results
- Robot action video

3) *Nicholas:*

- Background
- Methodology
- Results

VIII. Acknowledgments

Additionally, we would like to thank the lab staff for their support and availability. There was nearly always someone on hand who was ready and able to answer any questions we had. We would also like to thank the robot, as, without it's existence we could not have completed this project.