ROB 550 Robotic Control Report - Team 10

Sarvesh Mayilvahanan, Pou-Chun Kung, Manav Prabhakar
{smayil, pckung, prmanav}@umich.edu

I. Introduction

This report details how our group tackled the problems of forward kinematics, inverse kinematics, and path planning, with additional information of how each of these pieces were tied together for the arm's functionality in the competition. The forward kinematics allows us to have knowledge of the end effector's position in 3D space with only the known information of the joint angles, which was accomplished by using screw vectors. Additionally, a geometric inverse kinematics solution was implemented that provides a set of joint angles that will cause the end effector to have a desired position and orientation in 3D space. Path planning was used to combine these elements and have fluid motions without risk of interfering with other objects on the board. These elements were key to the movement of the arm and how it interacted with the objects around it.

II. Teach and Repeat

Prior to developing the kinematics of the arm, we built a function to store a set of joint angles and execute them to conduct a desired action. The teaching process involved moving the arm around with no torque and using a button to save the joint angles at the moment the button was clicked. A series of these joint angles were saved in a file, which could then be loaded and executed.

We created a set of actions which would swap the position of two blocks at locations $(-100, 225)$ and $(100, 225)$ using an intermediate location of $(250, 75)$. Because the blocks simply swap locations, we are able to continuously run this swapping motion as many times as we would like given that the blocks return to similar locations each time.

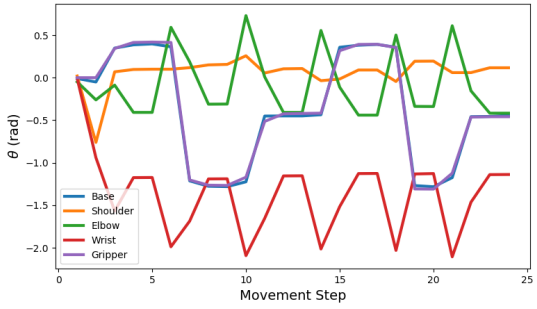

Fig. 1. Joint angles over time for a single swapping cycle

In Figure 1, we see a plot of each of the 5 joint angles over time for one cycle. We see that the gripper angle mirrors the base angle, which is due to the gripper staying square with the workspace. The elbow and wrist have opposite motions in order to keep the gripper straight as it goes up/down.

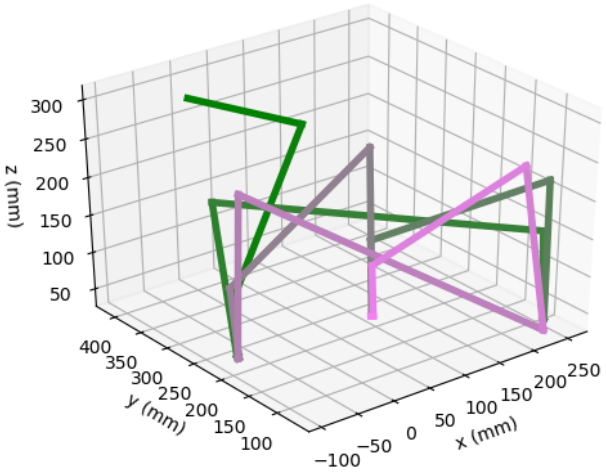

Fig. 2. 3-dimensional position of gripper during cycle using forward kinematics

We see the movement of the gripper in 3D space over time, with green being the initial point and pink being the final point. We see the arm initially moves to a point above the block and then moves straight down to pick up or place the block. This movement was later coded into the actual motion planning of the arm when performing autonomous actions.

We were able to cycle the blocks 10 times, and stopped after then tenth as it should be able to swap blocks indefinitely.

III. Forward Kinematics

Forward kinematics is key to identifying the current position of the end effector in the world coordinates given only the known information of the individual servo joint angles. Two approaches can be taken to evaluate the end effector position: Denavit–Hartenberg parameters or screw vectors. For the forward kinematics of this robotic arm, we chose to proceed with the screw vector method.

In order to use this method, we must first identify the end effector's position and rotation at its initial position

(all joint angles are 0), which we call the M matrix. The M matrix that we used, which is shown below, indicates that in the initialized position, there is no rotation and the end effector is offset in the y and z axes.

$$M = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0.408 \\ 0 & 0 & 1 & 0.305 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We also identify a screw vector S_i for each joint that describes the normalized angular and linear velocity of the joint i in the base frame.

$$S_i = \begin{bmatrix} \omega_i \\ v_i \end{bmatrix}$$

Constructing the screw vectors for each of the n joints, we can construct a screw matrix by joining the screw vectors together $S = \begin{bmatrix} S_1 & \dots & S_n \end{bmatrix}$:

$$S = \begin{bmatrix} 0 & -1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.305 \\ 0 & -0.105 & 0.305 & 0.305 & 0 \\ 0 & 0 & -0.05 & -0.25 & 0 \end{bmatrix}$$

We can then calculate the position and rotation of the end effector as a matrix T defined by the equation below:

$$T(q) = \left[\prod_{i=1}^{n} e^{[S_i]\theta_i} \right] M, \quad q = \begin{bmatrix} \theta_1 & \dots & \theta_n \end{bmatrix}^T$$

Where we can evaluate $e^{[S_i]\theta_i}$ as follows:

$$\mathrm{e}^{[S_i]\theta_i} = \begin{bmatrix} e^{[\omega_i]\theta_i} & (I\theta_i + (1 - cos\theta_i)[\omega_i] + (\theta_i - sin\theta_i)[\omega_i]^2)v_i \\ 0 & 1 \end{bmatrix}$$

Where we define $e^{[\omega]\theta}$ using the Rodrigues' formula.

$$[\omega] = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}$$

$$e^{[\omega]\theta} = I + sin\theta[\omega] + (1 - cos\theta)[\omega]^2$$

From the resultant matrix T, we can pull the world coordinates of the end effector as well as the Euler angles. We decided to go with ZXY Euler angles because it naturally made sense to rotate about the Z axis first (θ), then rotate about the horizontal (ψ), and finally rotate the end effector (ϕ).

We verified the accuracy of the forward kinematics solution by moving the arm to known locations across the board and empirically determined that it was producing accurate results.

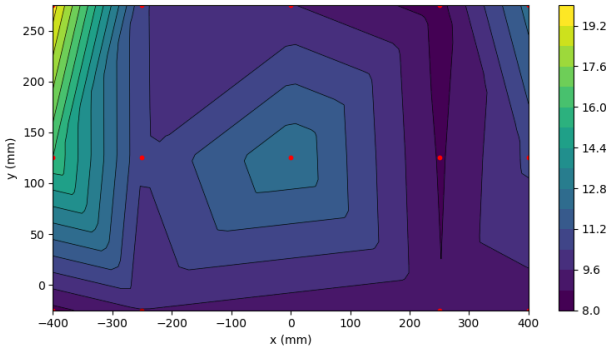

Fig. 3. L2 Norm between true end effector positions and forward kinematics predictions

Figure 3 above shows the error of the forward kinematics in predicting the end effector position. The red dots represent where measurements were taken, and the contour of the error is interpolated based on the data. While there isn't enough data to conclusively make statements on the forward kinematics accuracy, we can generally see that positions near the arm's base at $(0, 0)$ have lower error than positions further away at the edges of the board. This is likely due to the smaller errors in the forward kinematics calculation getting compounded in the individual joints.

Overall, the forward kinematics serves its purpose well, and because it wasn't truly used for any of the autonomous motions, the slight inaccuracies around 10 mm didn't make a difference in our end performance.

IV. Inverse Kinematics

Inverse kinematics is a crucial component of our robotic control system, allowing us to provide an orientation and location in 3-dimensional space that we would like the end effector to be at and determine the joint angles that will result in such a position.

We begin by looking at how we can simplify our problem from a complex 3D problem into a more manageable 2D problem. We do this by looking at the arm in a 2D plane characterized by rotation about the z axis. We can then decompose the problem into a 3 link arm with 3 unknown angles to solve for.

Before we analyze how our inverse kinematics solves for the correct joint angles, we need to take a step back and understand its purpose in the overall operation of the robotic arm. The goal of the inverse kinematics function is to provide joint angles that can be used to move the arm to an intended location with an intended angle ψ from the horizontal.

Initially, our thought was to only consider situations where the gripper was completely horizontal or vertically pointing downwards. This was because other intermediate angles would present awkward situations for the arm in the handling of the blocks. However, we noticed in testing that the arm could only pick up blocks horizontally outside of a certain range, which was much less accurate and sometimes dangerous if the arm was obstructed by a block.

The solution we put forth was to consider an inputted set of angles ψ that we could compute IK solutions for. These angles would be sorted by preference and would return the joint angles for the most preferred, achievable angle ψ.

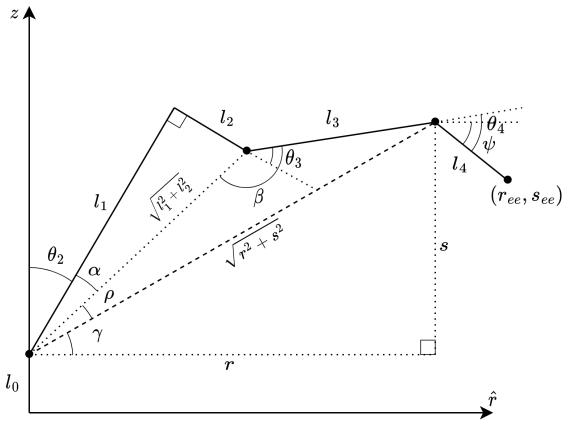

Fig. 4. Geometric inverse kinematics diagram for finding elbow angle

In the figure above, the joint angles $\theta_2, \theta_3, \theta_4$ represent the shoulder, elbow, and wrist angle correspondingly. The distance l_0 is the height of the shoulder from the base, l_1, l_2 are the distances from the shoulder to the elbow joint, and l_3 is the distance from the elbow to the wrist. We represent the angle of the wrist to the horizontal as ψ.

In order to initially calculate the joint angles using this frame, we must find the rotation of the arm about the z axis θ_1 such that we can define a new plane with the radial direction $\hat{r}$. (`kinematics.py - 249`)

$$\theta_1 = -\tan^{-1}(x/y)$$

We then find the location of the wrist in the new plane (r, s). We first find the end effector position (r_{ee}, s_{ee}) as follows (`kinematics.py - 247`):

$$r_{ee} = \sqrt{x^2 + y^2} \qquad s_{ee} = z - l_0$$

Based on the given wrist angle ψ, we can find the wrist position as follows given the wrist length l_4 (`kinematics.py - 275`):

$$r = r_{ee} - l_4\cos(-\psi) \qquad s = s_{ee} + l_4\sin(-\psi)$$

We additionally adjust the wrist location for some of the gravitational effects as the arm attempts to move to further locations by moving the arm back proportional to the radial distance r squared. Here, η is a tunable parameter to determine how drastic the gravitational adjustment is. (`kinematics.py - 278`)

$$r_{adj} = r - \eta r^2\cos(-\psi) \qquad s_{adj} = s + \eta r^2\sin(-\psi)$$

We can also find the angle from the horizontal γ to the wrist position (r, s). (`kinematics.py - 281`)

$$\gamma = \tan^{-1}(s/r)$$

Before proceeding with the joint angle calculations, we check to make sure that this position (r, s) is reachable by the arm. If $\sqrt{r^2 + s^2} > \sqrt{l_1^2 + l_2^2} + l_3$, then we output a flag for the corresponding angle ψ and position (x, y, z) marking the configuration as unreachable. (`kinematics.py - 282`)

We can then find the angle between the elbow and wrist joints β by using the law of cosines. We get two different angles corresponding to an elbow up and elbow down configuration. (`kinematics.py - 293`)

$$\beta = \pm\cos^{-1}\left(\frac{l_1^2 + l_2^2 + l_3^2 - (r^2 + s^2)}{2\sqrt{l_1^2 + l_2^2}l_3}\right)$$

We can also find the angle between the two perpendicular links from the shoulder to the elbow joint α (`kinematics.py - 259`):

$$\alpha = \tan^{-1}(l_2/l_1)$$

By simply combining the known angles with geometry, we can find a formula for the elbow joint angle θ_3. We get two of these angles because of the two different β angles. (`kinematics.py - 294`)

$$\theta_3 = \beta - \alpha - \frac{\pi}{2}$$

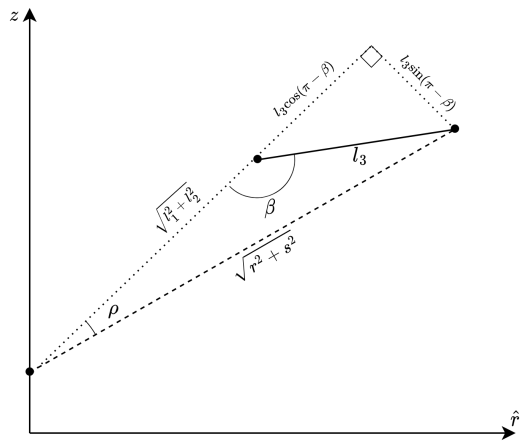

Fig. 5. Geometric inverse kinematics diagram for finding shoulder angle

Next, we find the angle ρ by extending the triangle bounded by the three joints. (`kinematics.py - 296`)

$$\rho = \tan^{-1}\left(\frac{l_3\sin(\pi-\beta)}{\sqrt{l_1^2+l_2^2}+l_3\cos(\pi-\beta)}\right)$$

The shoulder angle θ_2 can then be found through a combination of the other known angles. (`kinematics.py - 297`)

$$\theta_2 = \frac{\pi}{2} - \alpha - \rho - \gamma$$

The wrist joint angle can simply be computed by using the shoulder and elbow angles along with the given angle to the horizontal. (`kinematics.py - 303`)

$$\theta_4 = \theta_2 - \theta_3 + \psi$$

We opted to only rotate the gripper in the case of picking up or placing blocks vertically ($\psi = -\frac{\pi}{2}$) for a given block rotation θ_B which is determined by the computer vision implementation. (`kinematics.py - 251`)

$$\theta_5 = \begin{cases} \theta_1 + \theta_B - \frac{\pi}{2}\cdot\text{sign}(\theta_1) & \text{if } \psi = -\frac{\pi}{2} \\ 0 & \text{otherwise} \end{cases}$$

These five joint angles $\Theta = \begin{bmatrix}\theta_1 & \theta_2 & \theta_3 & \theta_4 & \theta_5\end{bmatrix}$ represent the output of the inverse kinematics function, where there are $2n$ different Θ values passed for n values of ψ. There are also $2n$ different flags $(0, 1)$ passed that correspond to the function's success in finding a viable inverse kinematics solution for that particular configuration.

V. Path Planning

Given a working inverse kinematics function, we can now proceed with simple path planning for specific actions such as pickup up blocks, placing blocks, and unstacking blocks.

A. Block Pickup

In order to pick up a block, we initially rotate the base such that the arm is in line with the object radially. We then proceed to move the arm to a position 300 mm above the board where the block is located. We then attempt to move to a position 100 mm above the block at angles ranging from $\psi = (-\frac{\pi}{2}\ldots 0)$. If we are able to successfully find an inverse kinematics solution for $\psi = -\frac{\pi}{2}$, we proceed to pick up the block at this angle.

If one of the non-vertical angles ψ had a successful solution, we proceed to move the arm back to a position 100 mm away from the block at the specified angle ψ to a staging position.

$$\begin{bmatrix}x\\y\\z\end{bmatrix}_{\text{stage}} = \begin{bmatrix}x\\y\\z\end{bmatrix}_{\text{block}} + \begin{bmatrix}0.1\cos(-\psi)\sin(\theta_1)\\-0.1\cos(-\psi)\cos(\theta_1)\\0.1\sin(-\psi)\end{bmatrix}$$

We can then move the arm to the block position and close the gripper to secure the block. After the block has been grasped, we can once again move to the position 300 mm above the board where the block is located and then return to a position directly above the base, which allows for quick movement to the next position for placement.

B. Block Placing

Block placement follows a similar order of events, with the arm initially rotating towards the new position and the moving to a point 300 mm above the board and placement location. We then move once again to a point 100 mm above the placement location and testing placement angles $\psi = -\frac{\pi}{2}, 0$. This is because we would only like to place blocks in a vertical or horizontal orientation in order to ensure the block sits squarely. If a successful inverse kinematics solution is found for a vertical configuration, we proceed to place the block at this location, with the gripper rotating to ensure the block is square to the coordinate frame.

If it is unable to find a vertical solution, we move the arm to the placing location with a horizontal configuration and open the gripper. After the block has been placed, we once again return the arm to its staging positions and finally the position above the arm's base.

VI. State Machine

In this section, we describe our logical flow for Event 1 and how the individual modules of our computer vision track and the control track aid us in completing the different events.

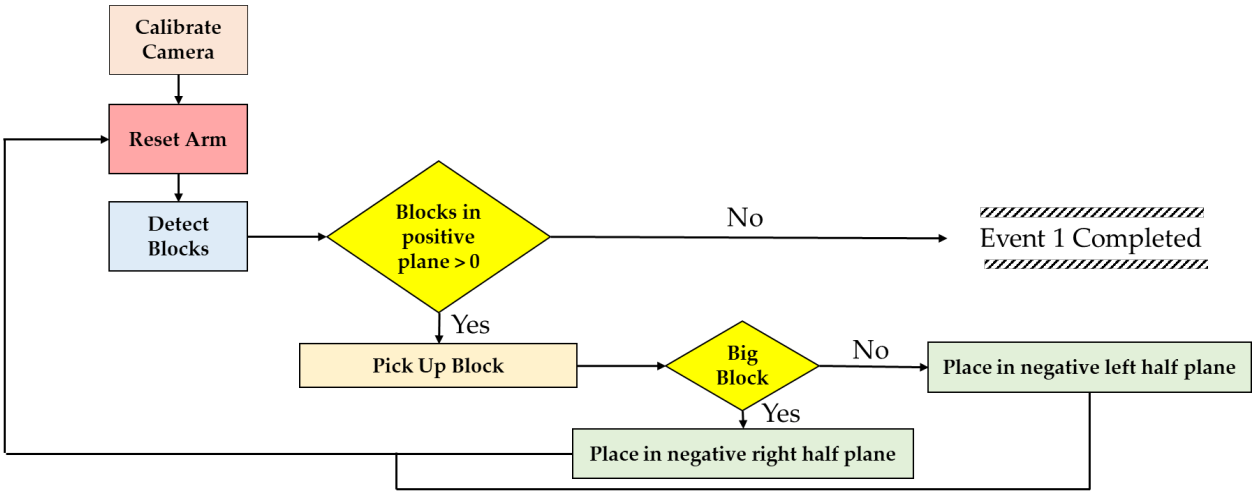

Fig. 6. Logical flow for event 1. We use the existing modules for detection, picking up blocks, and placing blocks

We have six primary action modules: $BlockDetector()$, $clean\_neg\_plane()$, $unstack\_all()$, $place\_block()$, $pickup\_block()$, and $do\_ik()$. All events use a combination of these modules in different logic to accomplish the specified task. Fig. 6 shows how these modules are involved in completing event 1.

Each step in the event 1 logic performs a set of actions utilizing the modules:

- **Calibration -** The first step is calibrating the camera using an April Tag bundle. Next, we perform template matching on the board to grab the region of interest. Once the template matching is finished, we extract the boundaries of our board, cropping out anything outside the board and check if our block detector is able to detect all blocks properly. After completing these two steps, our calibration is complete and we can begin the event.
- **Reset Arm -** The arm is set in upright position above the base using $do\_ik()$. This is done to avoid any occlusions in the camera's view of the board.
- **Detect Blocks -** This invokes our $BlockDetector()$ module to detect and store the attributes of all blocks, including the 3D pose of the block, orientation, size (big/small), and color. For this event, the blocks are sorted by distance from the base.
- **Pick Up -** Once the blocks are detected, the nearest block is the target, and based on its attributes, an inverse kinematics configuration is calculated and the block pickup (see section V-A) is arranged using $pickup\_block()$.
- **Placing in Negative Half Plane -** Based on whether the block is large or small, its placement position in the negative plane is determined. Once the position is determined, the block placement (see section V-B) is arranged using $place\_block()$.

This event flow is looped until there are no blocks left in the positive plane, after which the event is completed.

We can similarly construct logical flows for the other events, which have been omitted for the brevity of this report.

VII. Conclusion

The system performed fairly well during the competition as we were able to complete all the four events and ended up with the highest points in our section. There were however, slight tweaks that were required for every task keeping in mind the trade-off between precision and speed of the robotic arm. We forced the arm to move faster and wait for less time in the timed events and allowed the arm to spend extra time getting the best possible grip of a block for the other events.

We do have scope for improvement with respect to our state and logical flows as well as the calculations for our inverse kinematics configurations. In order to accommodate all contingencies, our arm movements were slightly complex. As a result, even though it didn't fail to perform any tasks, the performance time with simpler block configurations increased as opposed to simply using straightforward inverse kinematic configurations and path planning in such situations.

We can thus improve on making our algorithms intelligent enough to determine the level of complexity of the arm motion that is required to perform a given task. This improvement will be directly dependent on the vision algorithms and on the amount of movements we can provide to the arm. Additionally, adding intelligence to understand when the arm may need to regrasp the block would speed up the performance time.