ES 159 Final Project
Caleb Moore

Manipulating a 4 DOF Robot to Launch a Projectile

Objective

The goal of this project was to write a collection of useful functions for a unique 4 DOF robot (5 DOF including claw), and use these functions to manipulate the robot in an interesting way. The optimistic target of the interesting movement was to pick up a small projectile from a specified location, and launch it such that it landed on a specified target. The more realistic target was to pick up a small projectile from a specified location and launch it in a specified direction.

Figure 1: Mark II Robot

Materials

For this project I used the following:

- Mark II Robot (designed and assembled prior to this course)
- 5.1V, 5A Constant Voltage Power Supply
- Monitor with HDMI cable
- USB Mouse
- USB Keyboard
- Raspberry Pi with Python
- Ping Pong Ball

Procedure

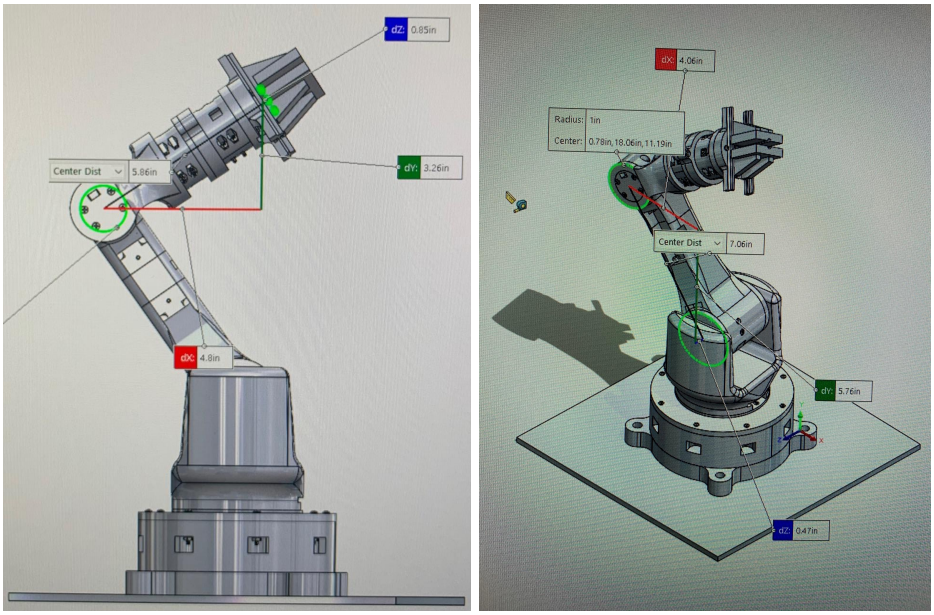
I broke down my project into several parts. First, I identified which functions from the course I would need to write for my robot. The majority of the work for my project would be in writing these functions in a new language, with new robot geometry. Additionally, I needed to identify functions that my application would require that weren't used in the course. Once I knew what I wanted to tell the robot to do, I needed a function that communicated desired positions to the robot. With all of the necessary functions written, I then needed to test the grip strength necessary to hold the projectile, and plan out the correct sequence of movements required to launch it.

Necessary Functions

The functions I required for my robot to accomplish my interesting movement included a Forward Kinematics Function, an Inverse Kinematics Function (analytic or numerical solution), and a Waypoints Function (using position control). In addition, I needed a function to check whether or not my robot could achieve a desired end-effector position. My last helper function was one that could actually make the robot move to the desired location. Putting all of these functions into a 'catapult()' function allowed me to accomplish my interesting task with one command.

Forward Kinematics Function *FK()*

This function uses the geometry of the arm (which I took directly from the CAD model in SolidWorks) to calculate where the end effector will be, and in what orientation, given 4 joint positions.



Figures 2,3: CAD Model of Robot with Essential Dimensions

The function is very similar to the FK lab of the course, which means I used the Denavit-Hartenberg Convention to find the screw axes. The table and labeled screw axes are in Figure 4. Writing this lab in Python was fairly straightforward once I had these screw axes.

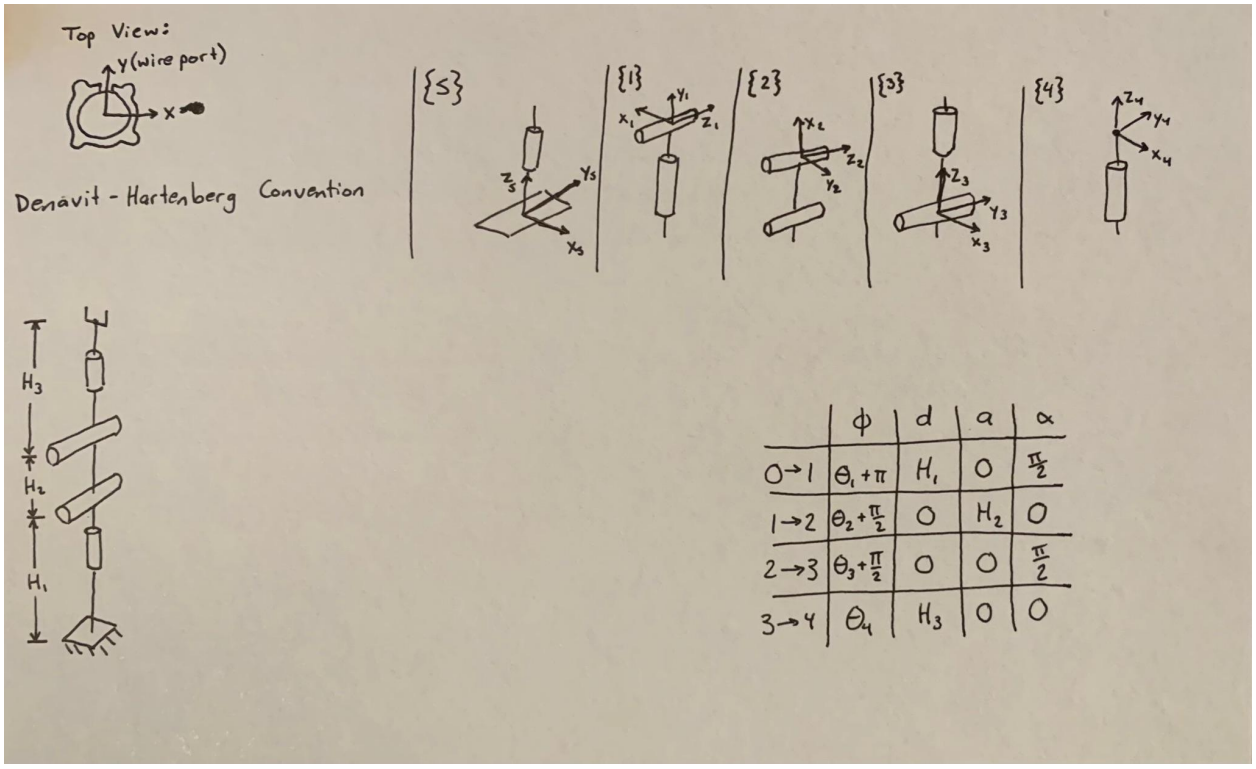


Figure 4: Denavit-Hartenberg Convention applied to the robot

The FK function checks to see if the inputs are outside of the joint limits, and it calls workspacetest() to check if the final configuration is in the workspace (it might be in the table). Once I had written the function, I decided to plot the workspace and learned that it consisted of one sphere, where each point on the surface of the sphere had a loop around it in the vertical plane containing the origin. With this mental model, I was able to simplify how I checked whether a position was in the workspace or not with workspacetest().

Inverse Kinematics Function *FindFastAngles()*

The original function that I wrote to calculate the inverse kinematics was using the Newton-Raphson method of approximating a numerical solution. Due to complications with translating the function to Python, I began to wonder whether I was overlooking an obvious alternative to approximating the solution. Upon realizing that the taskspace of the motion I was looking to achieve (launching a projectile) did not require me to rotate the end effector about the wrist), I decided to calculate the analytic Inverse Kinematics. This was better for me for two reasons: Firstly, I was having difficulty getting the numerical IK function to work for me, and secondly, I could easily calculate the analytic inverse kinematics for my taskspace using only the position of the end effector (x,y,z), without worrying about the basis vectors of the end effector. Calculating the analytic solution involved drawing a triangle where one point was the center of the shoulder joint, one point was the desired end effector position, and the third point was determined by knowing two of the three side lengths (H2 and H3). This triangle gave me the value of the shoulder joint and the elbow joint. The base joint was found using the end effector's x and y position. The claw joint never needed to change because of my specific taskspace.

Waypoints Function *MoveSlowly()*

This function was also modified for my purposes to make it faster to deal with. Instead of sending it a sequence of waypoints, it receives the start and end position, and the length of time to expend in getting there. This simplification was used to guarantee that the joints would not be asked to achieve an end effector orientation that was unattainable (recall that 4 DOF robots can achieve many [x,y,z] positions for the end effector, but not as many [u,v,w] orientations).

Workspace Check Function *workspacetest()*

This function takes an input end effector matrix T and determines whether the robot can achieve it or not. It does this by using the end effector position and orientation (travel backward by H3 along the z-axis of the body frame) to determine where the elbow of the robot would need to be. If the elbow lies on the surface of the sphere of radius H2 around the shoulder joint (see Figure 4), then the position and orientation are attainable. The function also checks whether any part of the claw will make contact with the table (it cannot contact the base of the robot).

Catapult Function *catapult()*

The catapult function takes a start position of the projectile and an end position to launch the projectile at. See Testing and Implementation below for a discussion of how this function changed over time. It first implements some checks to see if the robot will need to turn around to reach the starting point or the end point (base angle range is 270 degrees). Next, it calculates the launch speed required to fire a projectile at a 45 degree angle (to maximize distance) such that it hits a specified target location. These calculations (See Figure 5) do not factor into the current catapult function, but they are calculated to show that with faster motors the desired goal could be accomplished. Starting from the home position (standing vertically), the robot moves to the location of the projectile. It closes its claws around the projectile and slowly lifts it about an inch off the ground. Next it slowly maneuvers into the initial launch position, fully extended away from the target. Eventually it rotates the shoulder and elbow simultaneously, much like the wind-up of a pitcher, and quickly extends both joints to launch the ball as it passes through the home position at a fixed but unknown velocity (See Testing and Implementation below for discussion as to why).

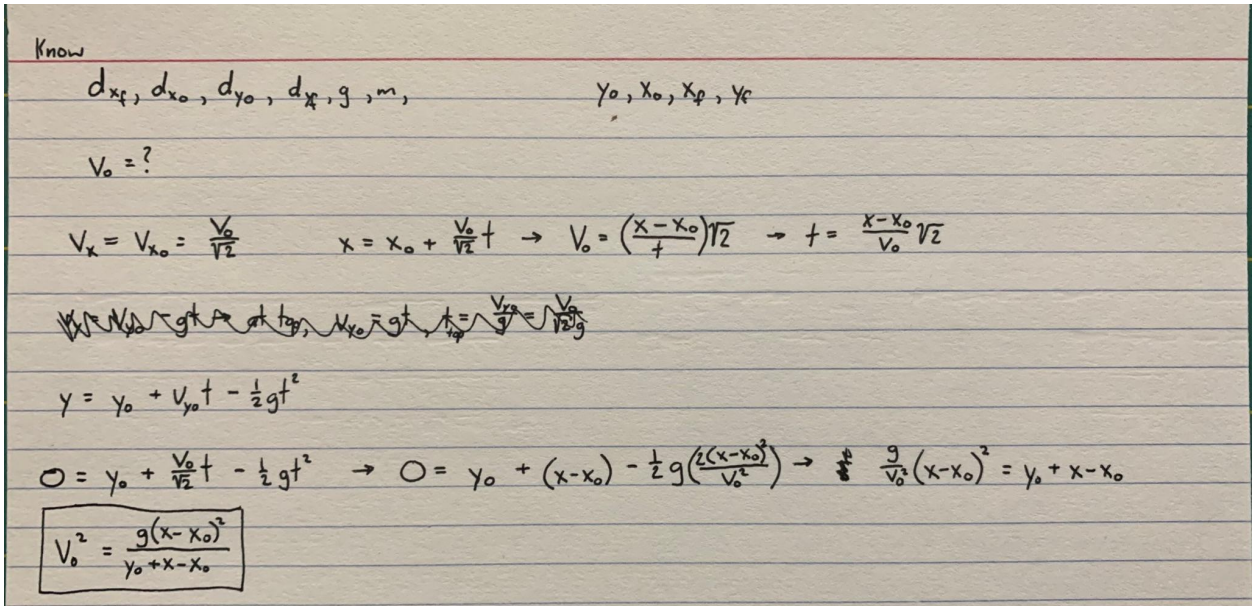


Figure 5: Calculations for launch speed

Movement Function *MoveTo()*

This function is adapted from previous code I had written prior to this course. It uses the ServoKit library from Adafruit to communicate with the hardware of the robot. After running initial checks for collisions and converting desired angles into the numbers that the joints will understand (most range from 0 to 270, not -135 to 135), it sets the pulse width for each PWM signal to each motor and sends the necessary pulses.

Testing and Implementation

After several dry runs of picking up at position (11,0) and launching at (11,-5) without a projectile to prove that the sequence of movements was what I was expecting, I next had to choose a projectile and calibrate the claw's grip around it. At the suggestion of Professor Werfel, I settled on a ping pong ball for its low mass. While keeping the ball in between the claws, I progressively closed them until the ball's weight was supported. Using this new grip position knowledge, I updated my catapult function and tested the movement sequence with a projectile.

The results were underwhelming. By sending too many waypoints to the robot during the launch, I was slowing its maximum speed significantly. I opted to rewrite part of the launch code to only send the end waypoint, so that the ball would be launched at a 45 degree angle. This meant that I could no longer control the speed (and thus distance) at which the ball was launched, and only the direction. My optimistic objective was thus discarded in favor of my practical objective.

Again, the results were underwhelming. However, I knew that my robot had the potential to move quickly enough because whenever I sent it to its home position, it would move so fast that it could fall over. The difference was which angles I was trying to move quickly over. At extreme angles, away from home, the necessary torques are very high for the robot to overcome. When the robot is almost vertical, the necessary torque required to move is much lower, so the robot speeds up. Using this knowledge, I decided to launch the ball not from the ideal angle of 45 degrees, but from the point at which the velocity of the end-effector is highest. By releasing the ball when the robot is fully extended in the home position, I can maximize the torque of the shoulder motor.

Trying this new launch angle helped, but the robot did not release the ball with as high a velocity as I had hoped. Finally, I decided to launch the ball as a baseball pitcher would, by rotating around two parallel axes. As the shoulder joint rotated from an extreme angle to home, I had the elbow angle do the same. This would maximize the linear velocity of the ball, because it was now rotating around two parallel axes in ranges that required low torque, just like when a human arm throws a baseball (as opposed to a softball). I also added a second waypoint after the release of the ball so the arm wouldn't stop or slow down at or before release (just as a pitcher follows through their motion).

Video Demonstration:
https://youtu.be/UbziBq38yJE

Summary of Results

By the end of my project, the robot was able to pick up a ping pong ball from a specified location (x,y) and launch it in the direction of a specified target (x,y). It did this with the help of an analytic Inverse Kinematics Function, a Forward Kinematics Function, a Waypoints Function, a Workspace Checker Function, and a Movement Function.