CS225A Final Project: DartBot

Kyle Casey, Victor Yin, Sidney Stevens, Ethan Woods
Stanford University
kcasey2@, vhsy@, sidneyas@, ewoods24@ - stanford.edu

Abstract

Our project, "DartBot," tackles the topic and application of simulated robotic game play. "DartBot" utilizes the versatile seven degree of freedom Panda Arm robot with a one degree of freedom end-effector gripper. We implement a simulated darts game. In our designed game, a user can select a difficulty corresponding to the distance between the robot and a dartboard. Once the user chooses the difficulty, the Panda Arm picks up a dart from a table and throws it at a dartboard, which sits at the selected distance. Our robot can accurately hit the bullseye at any of the given distances.

1 Introduction

Project DartBot is the culmination of our team's work in CS225A: Experimental Robotics. Throughout the quarter, we learned to control robots to execute precise motions within a finite workplace. For our final, we sought to expand our knowledge by taking on a new challenge within the field of robotic control. We aimed to apply our skills to a challenge that we would further find fun and meaningful.

This project was inspired by one of our favorite local bars, The Rose & Crown. This bar has an area for people to play darts and regularly hosts friendly competitions for its patrons. We wondered what it would be like if a robot were to play the game, so "DartBot" was born.

While the initial goal was to have our robot play an actual game of darts, this proved to be infeasible for the scope of this class because training the robot to hit all of the different point values on the board would be extremely difficult. It would require different trajectories for each spot on the board, some training for it to play for an optimal scoring sequence, and a complex user interface to input everything and make it interactive. As we iterated through ideas for how to implement our project, we landed on our own "dart game" that we tailored to the robot's capabilities and fit the class's scope. Our game allows a user to select different difficulty levels corresponding to different distances between the robot and the dartboard. The novel part of our project is that the robot can throw a dart and hit the bullseye at any given distance.

2 Final Implementation

2.1 Overall Structure

Users interact directly with our graphical user interface to select the robot's level of challenge. The GUI then communicates the user's selection to the Redis server and launches our simulation visualization and robot control scripts. The simviz opens with a dartboard in the location specified by the user, and the robot moves through a pre-specified trajectory to reach a final velocity based on the position of its target.

Our project is implemented entirely in simulation in the SAI 2.0 environment. On a real robot, we could simply enact a pre-specified control sequence. However, visualizing our control strategy in simulation involves multiple collisions between the dart and the end-effector grippers, and the

,

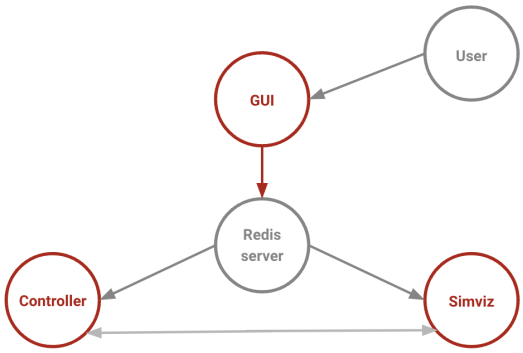

Figure 1: Structure

dart and the dartboard. In order to update the simulation realistically, our simviz and controller communicate through the Redis server with information about the panda arm's and dart's position. At points, the dart is controlled by our inputs to the simviz rather than responding to simulation outputs that do not perfectly match real-world physics.

2.2 User Interface and Redis Server

For our user interface control, we created a GUI in python which allows the user to pick from four levels of difficulty (representing 1m, 2m, 3m, and 4m distance between the robot to the dartboard). Before the simulation begins, the user selects the level, which will place the dartboard at the corresponding distance.

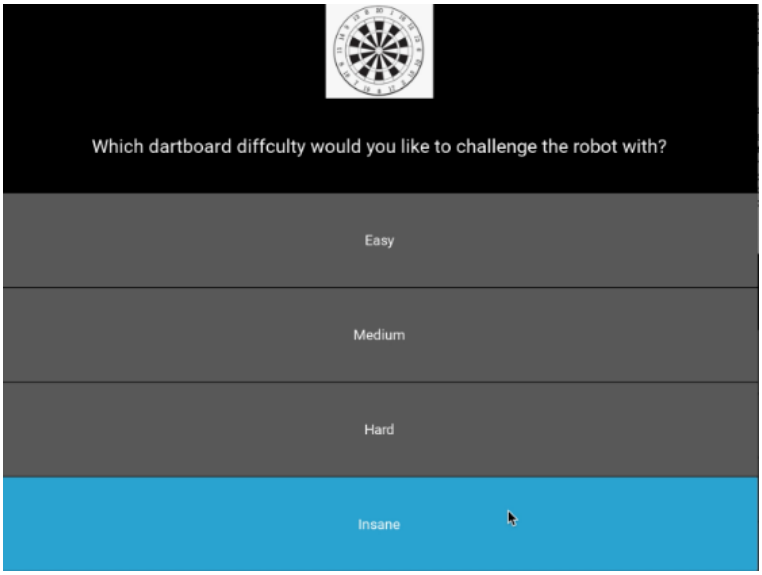

Figure 2: User Interface (GUI)

2.3 Environment

The virtual environment for our dart-playing arena is pictured below. It incorporates the panda arm along with custom objects that our team designed and built in blender. The key components that have collisions include the dart, dartboard, robot arm, and the table on which the dart is placed (See Appendix for a figure of the collision boxes visualized). In the background, we added a bar table and bartender along with additional decorations.

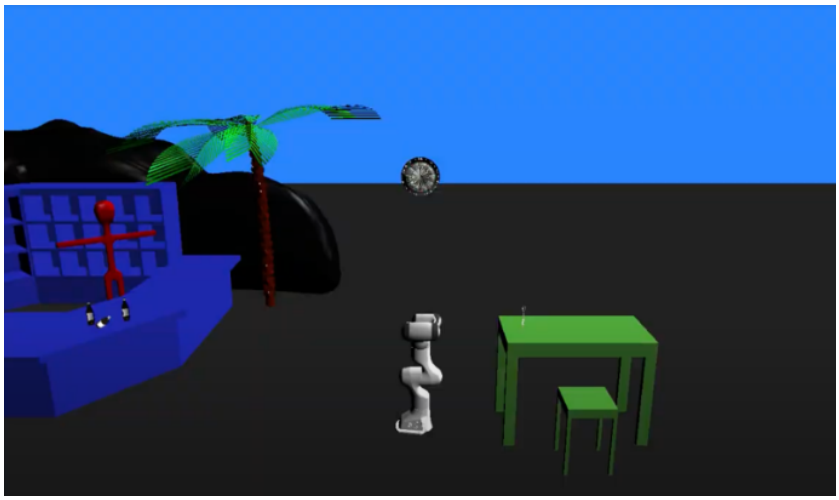

Figure 3: Overall Environment

The dart is modeled according to standard dimensions: measuring 13.5cm length and weighing 50 grams. This was created as a dynamic object to allow for collisions with the robot gripper and table. We set the dart on the table for added difficulty. Our simulated dartboard has a standard diameter of 40cm. This was created as a static object and placed at a distance determined by the user.

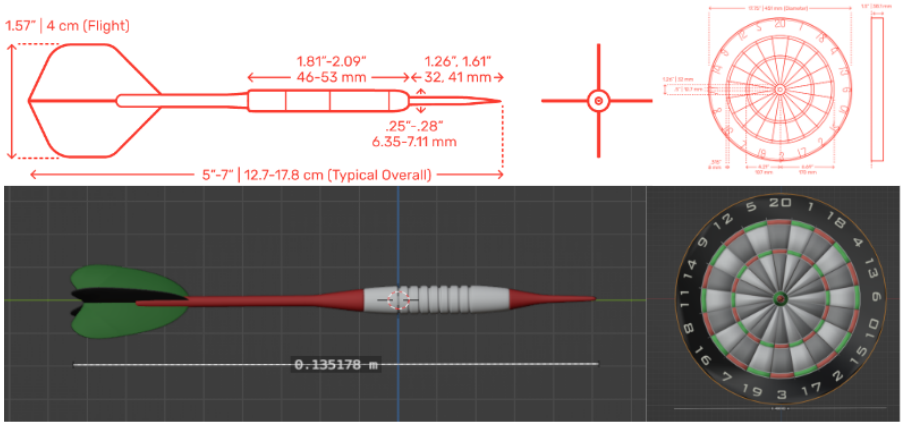

Figure 4: Dart and Dartboard

For the robot arm, we used the Sai2 panda_arm_hand.urdf model provided by the teaching team. This came with the standard panda end-effector for the gripper. In addition, we modeled supplemental collision boxes for the robot end-effector. See the appendix figures for more detail.

2.4 Controller and Trajectory

The state machine pictured below (Figure 5) summarizes the steps taken by our robot to accomplish its desired task. When the simulation is launched, the robot loads in a preset non-singular configuration. It then moves to pick up the dart, get into its pre-launch configuration, move to its launch position and velocity, and finally release the dart at its target.

In order to achieve critical damping as the robot moves throughout its pick up operation and launch trajectory, we tuned kp and kv values for the position and joint task as well as the grippers. Given the high torque commands we give to the robot during trajectory, the kp and kv values for position control were extremely high in order to follow the trajectory more closely.

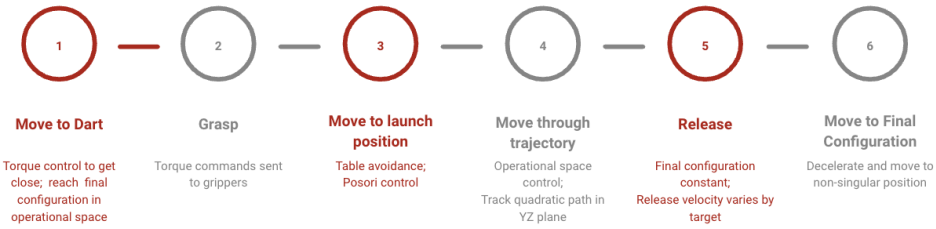

Figure 5: High Level State Machine

While the robot is in position and joint space, We set velocity saturation and the interpolation flag to limit the robot's velocity to prevent the dart from slipping. We also set the dynamic decoupling to full. This will become important for the throwing trajectory. Most importantly, we set the control position for the end-effector to be 0.22 meters from the base of the last link. This is the location where we will grab the dart. Below are the robot torque commands.

Joint Space Control:

$$\Gamma = -K_p(q - q_d) - K_v\dot{q} \tag{1}$$

Gripper Control:

$$\Gamma_{grip} = -k_{v_{grip}}\dot{q}_g - k_{p_{grip}}(q_g - q_{g,d}) \tag{2}$$

Operational Space Control:

$$F = \Lambda(-k_p(x - x_d) - k_v\dot{x}) \tag{3}$$

$$\Gamma = J_v^T F + g \tag{4}$$

Throughout our control sequence we alternate between commanding the robot in joint space and in operational space. Joint space control allows us to limit the degrees of freedom of the panda arm as we get the end effector within a desired region to perform a specific task. The robot transitions between states when its joints or end effector position are within a specific tolerance from the desired orientation or position.

Our first torque command occurs in joint space. Through multiple rounds of iteration we were able to find a joint position that got the end effector close to the dart, but not quite touching it. Once we got the gripper to within a certain critical epsilon (varied from 0.01 to 0.001) from the dart, we transition to position control. To assess proximity to the desired orientation, we calculate the norm of the difference between the joint values and the desired joint values. We used this technique of calculating the error in our position or orientation to transition between all of our control states. In operational space control, we take the difference between the end effector's position and desired position.

Once the robot is in its desired orientation, we use operational space control in order to move the end effector to the dart. In this state, the end effector is oriented to be parallel to the ground and perpendicular to the dart. We set a low tolerance on final position and velocity prior to grasping to ensure that the robot can pick up the dart at its center of mass without knowing it over. Once a still and accurate position is reached, the robot transitions to a grasping state. Gripper desired positions are set to +/-0.019m from the prismatic origin. These values were crucial because the edge of the dart is located at +/-0.02m relative to the prismatic origin. Setting the desired location be at a position between the dart's outer edges (a location the grippers could never reach) ensures that the grippers apply a constant force on the dart. This, along with parameter tuning for a high coefficient of friction in the simviz allowed our grippers to securely hold onto the dart while moving it.

Once the robot successfully grasps the dart, we use joint space control to move the dart and panda arm safely out of the way of the table. Once the robot reaches this configuration, we use operational space control to reach the desired pre-launch start location for the trajectory.

Now the robot is in the trajectory state. First, we set the velocity saturation and interpolation flag in the operational space to false. We update the k_i value for the PID controller depending on the distance to the user-selected dartboard. Then for each loop it sets a desired position, velocity and acceleration. Our PID operational space control equation (3) is thus replaced by:

$$F = \Lambda(-k_p(x - x_d) - k_v\dot{x} + \int_0^t (x - x_d)dt) \tag{5}$$

For our trajectory, we set a constant acceleration quadratic trajectory in the Y-Z plane. We initialized a new time variable at the start of the trajectory state. This time variable was used to create the following equations.

$$\begin{gathered} a_y(t) = a_y(0) \\ \mathrm{v}_y(t) = a_y(0) * t + \alpha_y \\ \mathrm{d}_y(t) = \tfrac{1}{2} * a_y(0) * t^2 + \beta_y * t + d_y(0) \end{gathered} \tag{6}$$

$$\begin{gathered} a_z(t) = a_z(0) \\ \mathrm{v}_z(t) = a_z(0) * t + \alpha_z \\ \mathrm{d}_z(t) = \tfrac{1}{2} * a_z(0) * t^2 + \beta_z * t + d_z(0) \end{gathered} \tag{7}$$

The velocity and position equations have additional hyperparameter terms (alpha, beta) due to inconsistency in the joint space. After multiple iterations of tuning, we found good hyperparameter values for each dartboard that consistently bring the end-effector to the correct position and velocity at the end of its trajectory. Unlike for the other states, we set the exit conditions for the trajectory state in terms of time (not proximity to a target value). Once the time reaches t-end for the trajectory, we set the gripper position back to its initial configuration (-0.01 and 0.01). Then the controller communicates to the simviz via Redis to set the friction equal to zero to release the dart. This helps minimize the potential velocity that the dart can lose. We then allow the controller to complete one more iteration and then set the kv value in position space to 400 to dampen the speed of the robot. The robot then transitions to it final state, which enacts control in joint space to move the robot into a final non-singular orientation.

Figures 6 and 7 display plots of our end effector position and velocity when aiming for the dartboard 4 meters away. The first two plots show how inaccurate our control strategy was without integral control and hyperparameters.

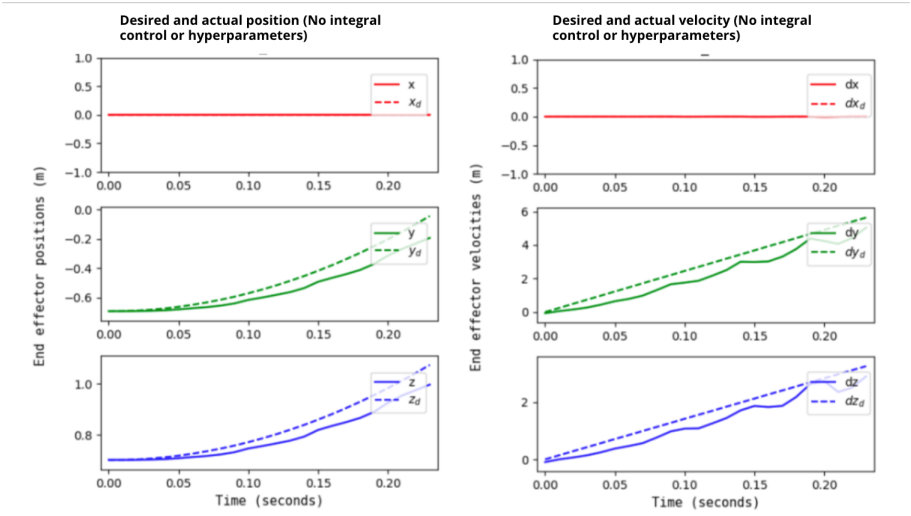

Figure 6: No Integral Control or Hyperparameters

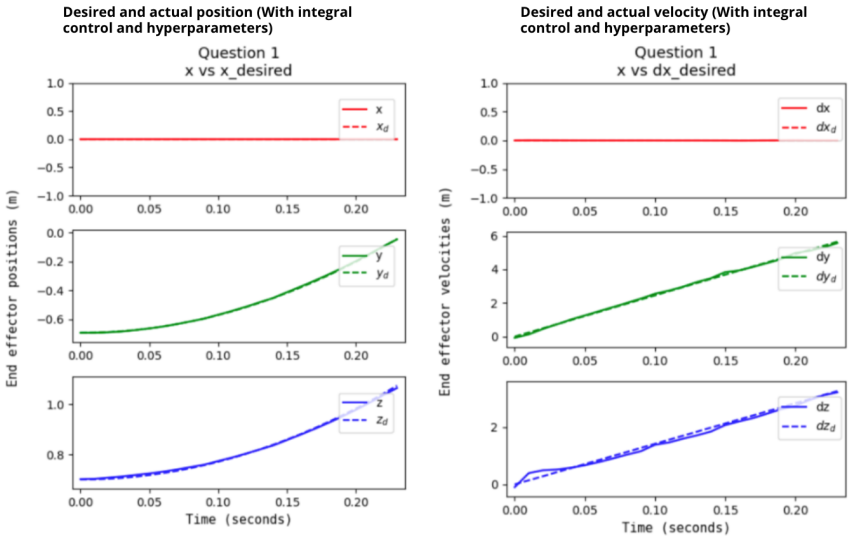

Figure 7: Integral Control and Hyperparameters

Rather than simulate collisions between the board and dart, we instead freeze the dart when it reaches the board's position as we were unable to fully resolve challenges associated with friction in the simulation.

3 Challenges

3.1 Collisions

During this project, we faced many issues with the controller, but our most impactful issue was collisions. At first, we were concerned about the darts' collision with the dartboard. After some initial tests, we realized that the dart was not going to be able to stay on the dartboard. However, we were able to come up with a very simple solution. In our simviz cpp, we simply set the dart as a dynamic object and when the dart reached its desired position in the y direction, we saved that position and set that as its position each loop so it essentially froze in place.

Another collision issue which frustrated our group was the collision between the dart and the grippers. At first, we set each gripper as a collision box and the dart as a collision cylinder. We thought that a cylinder would create a line of contact and this minimal contact would lower the randomness and therefore lower the slipping of the dart. However, this was not true. We ultimately chose to have each gripper be a pair of collision boxes and the dart itself was also a box. We also set the friction in the simviz to be around 10. Although this number was too high (above 1), after many iterations it was a number that could consistently minimize slipping. However, we could never eliminate slipping and this was a problem that greatly affected our controls and accuracy of throws.

3.2 Controls

As we stated in our implementation, our robot follows a desired trajectory employing both operational space control and dynamic decoupling. Finding a trajectory that worked consistently took a great amount of strategy iteration and parameter tuning. We first tried to implement a trajectory generation method with certain constraints. For example, our trajectory was in the y-z plane. We wanted to get the robot into an initial desired position with zero velocity and zero acceleration. We also wanted a final position and velocity for the release of the dart. This totaled to 10 constraints and 10 equations. Therefore, we created two quartic polynomials for the y and z directions, respectively. Unfortunately, no matter what constraints were applied, the robot end-effector was not able to consistently follow any trajectory. So we then simplified our trajectory to a straight-line, constant velocity trajectory. This trajectory was an improvement, but still failed because it didn't have any second order following

(acceleration tracking). Our final solution was thus to follow a quadratic trajectory with constant acceleration. We pass parameter-based equations (6 7) to the controller, and iterate within a loop to update the desired position, velocity and acceleration based on the amount of time that has passed.

With this improved trajectory we were able to hit the bullseye on some attempts, but did not have a 100% success rate. This was due to two factors. First, the controller command torques have a slight time delay, so our position and velocity had a steady state error. To solve this problem we implemented a PID controller, adding the ki term shown in equation (5). This improved our trajectory, but there was still a slight error. Though we are uncertain as to the exact source of this error, we believe it is due to the location of our trajectory in the robot's workspace. Our trajectory is purely in the operational space and doesn't take joint space into account. We believe the extra degrees of freedom and flexibility in the joint space caused an inconsistency in trajectory following. Our solution was to add hyperparameters in the velocity and position control. This allowed us to tune our trajectory for each of the four dartboard locations. However, it limited our project because we we unable to find a system that could work for any random dartboard distance.

4 Conclusion / Future Work

Our project wouldn't have been possible outside of simulation, but certain aspects of simulation limited and hindered our progress. Simulation allowed us to test torque commands and algorithms without damaging the robot and the world around it. However, some of our greatest struggles were with collisions and working with the physics of the environment. We were able to get our controller to consistently follow our trajectory, but we could never get the dart and gripper collisions to consistently work. For future work, the trajectory can definitely be improved and optimized in multiple different facets. A more improved trajectory would be in three dimensions (xyz). This would allow a user to select any dartboard location within a certain range. The trajectory also would be improved if it took into consideration torque limits and the robot's reachable workspace for each joint.

In the future, creating a dart throwing robot in the real world would be a great challenge. The panda robot has extremely small max velocities and torques, so ultimately throwing a dart any reasonable distance might be impossible. This is an interesting area to explore and going forward, this task could be completed by a robot with higher torque capabilities.

5 Video Link

Link: https://youtu.be/cSNVPkkFV4w

Nomenclature

α_y	Hyperparameter for velocity in y direction
α_z	Hyperparameter for velocity in z direction
β_y	Hyperparameter for position in y direction
β_z	Hyperparameter for Position in z Direction
$\dot{q}$	Joint Velocities
$\dot{q}_g$	Gripper Velocity
$\dot{x}$	End-effector Velocity
Γ	Torques
Γ_{grip}	Gripper Force
Λ	Operational Space Matrix
a_y	Acceleration in y Direction
a_z	Acceleration in z direction
d_y, d_z	Displacement in y and z Directions
F	Force
g	Gravity
GUI	Graphical User Interface
J_v^T	Transpose of End-effector Jacobian Matrix
k_i	Integral Gain in Operational Space
$k_{p_{grip}}$	Proportional Gain for Gripper
K_p	Proportional gain in joint space
k_p	Proportional Gain for Gripper
k_p	Proportional Gain in Operational Space
$k_{v_{grip}}$	Derivative Gain for Gripper
K_v	Derivative Gain in Joint Space
k_v	Derivative Gain in Operational Space
PID	Proportional–Integral–Derivative
q	Joint Positions
q_d	Desired Joint Position
$q_{g,d}$	Desired Gripper Position
q_g	Gripper Position
t	Time
v_y, v_z	Velocity in y and z Directions
x, y, z	Positional Coordinates/Trajectories of End-effector
x_d, y_d, z_d	Desired Positional Coordinates/Trajectories of End-effector

6 Appendix Figures

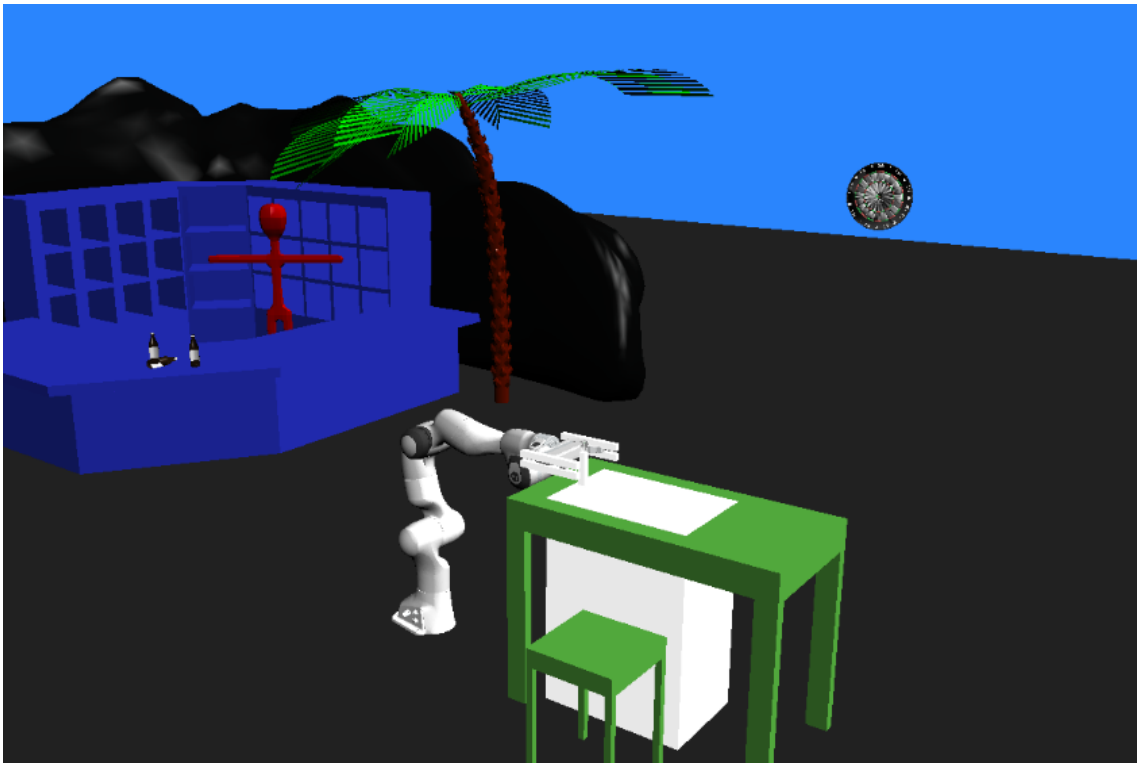

Figure 8: Environment (Collision Boxes)

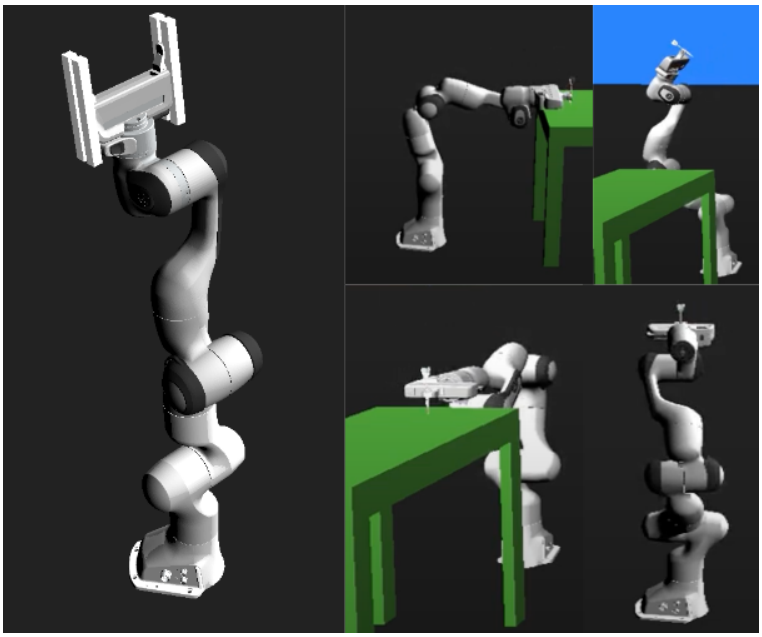

Figure 9: Robot Arm

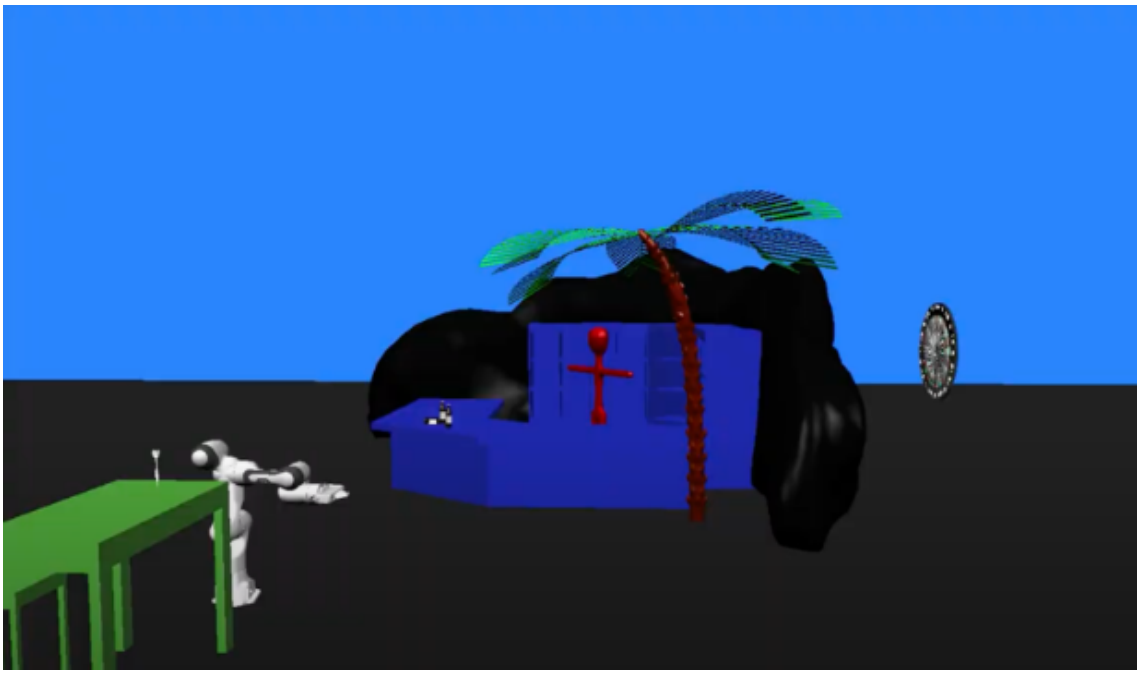

Figure 10: Overall Environment (Angle 2)