Project Update 3

Team: Frat Daddy

Members: Monika Spytek (mspyte3) & Seung Hyun Shin (sshin14)

Links: https://github-dev.cs.illinois.edu/mspyte3/ece470

https://youtu.be/nqU8xzZwBuU

Update:

Our goal for this update was to manually tune the system to get one ball into one cup and have a sensor inside the cup return if the ball triggered it. As usual, a lot of our time was spent trying to manage the API's functions and learning its quirks. However, we also spent a notable number of hours trying to find just the right base angle and initial velocity to give the ball so that it would land in the cup.

The first issue we tackled was trying to give the ball an initial velocity. At the end of the last update, we didn't manage to get the ball to move anywhere. Even trying to use the simxSetObjectFloatParameter() function (with the velocity parameter) for other objects in the scene wasn't making them move. An extensive amount of googling later led us to one forum where somebody said that an object's position might need to be reset before assigning any velocity. We manually reset the ball's position to the coordinates just before we wanted to detach it from the parent and launch it, and this worked well in terms of finally getting our ball to move in the right direction. The next issue we encountered was that those coordinates would change depending on the base angle of the robot for each throw.

Using the simxGetObjectPosition() function also gave us trouble for a while because it kept returning the origin of the world instead of the location of the ping pong in the world frame's reference. This was fixed by waiting for half a second between moving the arm to the position and reading the ball's position and then by setting simxGetObjectPosition() with the blocking operation mode. We learned a lot about the different operation modes when trying to work through this portion of the project. The blocking mode needed to be used because it stops the python thread from executing until the simulation returns a value. Previously, it was returning the origin because our python code wasn't waiting for a calculated response from the simulation. Once we got the correct position, we disconnected the ball from its parent (the robot), reset the ball's position, and gave the ball an initial velocity in the x direction all while pausing the communication with the simulation. We paused the communication so that all those commands would be sent simultaneously.

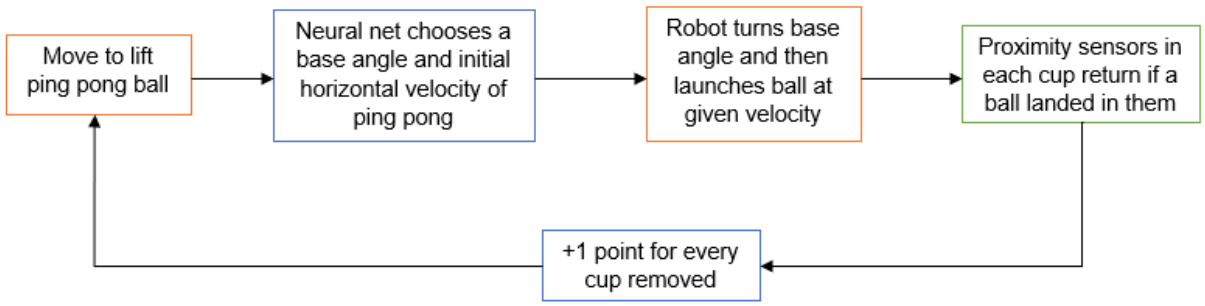
Several hours were spent using a guess-and-check method to find the right base angle and initial velocity to land the ball in the cup. Some basic projectile kinematics were used to estimate about what velocity was needed to get the ball to travel from the height it was let go to the cup, but using kinematics equations would have ended up being more complicated than

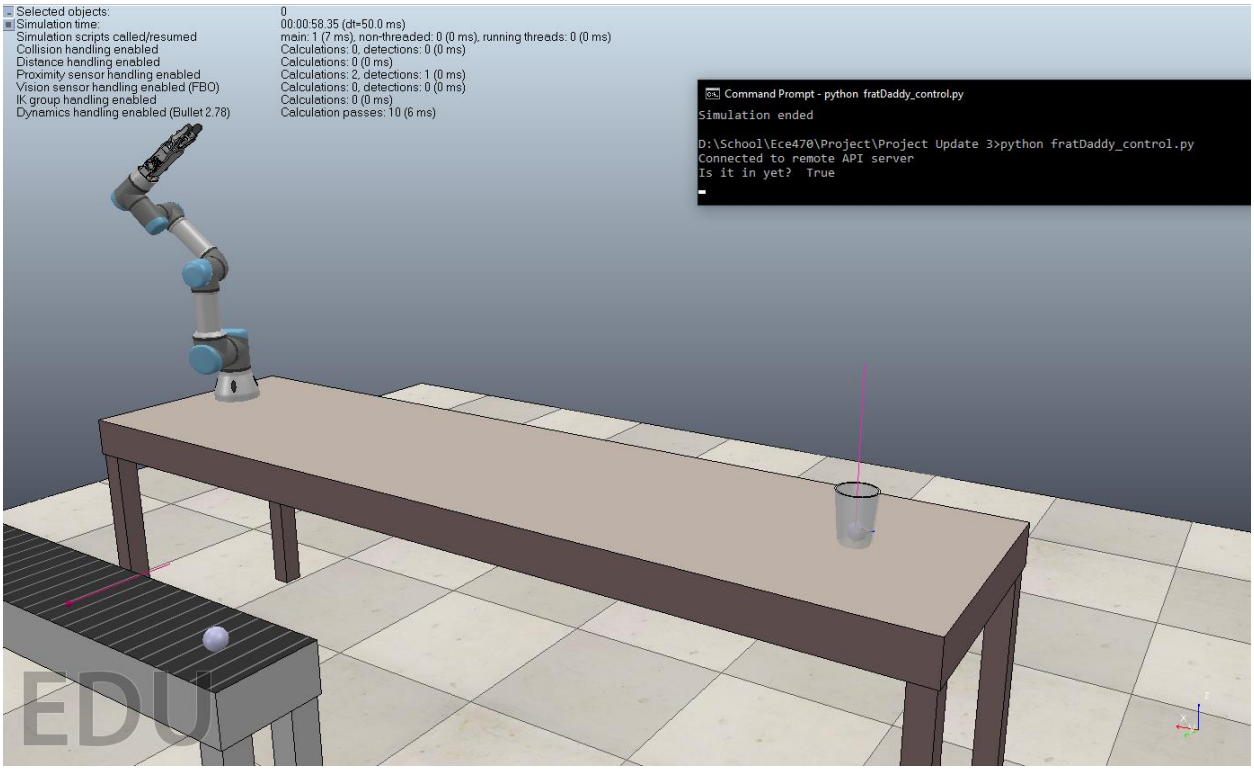
making small manual adjustments. There are just too many variables to consider to quickly calculate the velocity and angle we wanted because we would also have to spend a lot of time accounting for the base angle lining up perfectly and for irregularities in the simulation.

We still have trouble where every so often the ball will collide with the gripper and not be centered in the correct location. However, a lot of the irregularities were mitigated by changing some of how we simulated picking up the ball. Previously, we were grabbing the ball and then setting joint velocities to make the gripper tighten on the ball. Doing this would sometimes push the ball out of position and ruin that run of the simulation. Instead, now we set the joint velocities to 0 before we attach the ball to the gripper. This keeps the gripper from pushing the ball out of position. This improvement on our previous code made our simulation much more reliable and made it much easier to find a base angle and velocity that would consistently land the ball in the cup.

After finding the right values to make the shot, we started working with a proximity sensor inside the cup. The proximity sensor needed to be set in the CoppeliaSim file to only detect the ping pong ball, otherwise it would sense the table and the cup. We initially assigned the handle for the proximity sensor at the beginning of the code along with all the other handle declarations, but we were very confused as to why suddenly our simulation's reliability was significantly worse. Without changing anything else in the simulation, now our ball was almost never landing in the cup. Simply commenting out the code relating to the sensor would make our program run correctly again. We still aren't sure why, but on a hunch, we moved the sensor's initialization to right after the ball has been thrown and our simulation was reliable again and able to return that the ball had landed in the cup.

The last improvement we made for this update was to organize our code better into functions for each task. Doing this is preparing for our next goal of introducing a neural network to repeatedly run the program and learn the base angles and velocities to land the ball in multiple cups. Our next steps include giving our simulation 10 cups with force sensors and building a simple neural network that will randomize the base angle and ball velocity try to and land the ball in each cup. We aren't sure yet if it would work better to train the neural net individually for each cup location (since the cups will always be in the same orientation) or if we should train it on the game as a whole, so making a shot in the same cup can only reward the arm once. The first approach would mean that we would need to separately train each cup and then save those values, and then train the robot on the game as a whole for which cup to get out first. The second approach would train the parameters for each cup while also training for the game as a whole, but it might not be very reliable because throwing to the same location twice won't reward the network, potentially reinforcing a random throw each time. Ultimately we want to reinforce throwing to one of 10 locations in a sequence.


Move to lift ping pong ball
Neural net chooses a base angle and initial horizontal velocity of ping pong
Robot turns base angle and then launches ball at given velocity
Proximity sensors in each cup return if a ball landed in them
+1 point for every cup removed


Selected objects: 0
Simulation time: 00:00:58.35 (dt=50.0 ms)
Simulation scripts called/resumed main: 1 (7 ms), non-threaded: 0 (0 ms), running threads: 0 (0 ms)
Collision handling enabled Calculations: 0, detections: 0 (0 ms)
Distance handling enabled Calculations: 0 (0 ms)
Proximity sensor handling enabled Calculations: 2, detections: 1 (0 ms)
Vision sensor handling enabled (FBO) Calculations: 0, detections: 0 (0 ms)
IK group handling enabled Calculations: 0 (0 ms)
Dynamics handling enabled (Bullet 2.78) Calculation passes: 10 (6 ms)
Command Prompt - python fratDaddy_control.py
Simulation ended
D:\School\Ece470\Project\Project Update 3>python fratDaddy_control.py
Connected to remote API server
Is it in yet? True
EDU