Challenge Problem 3

Sarina Cusumano, Jackson Fairborn, Radhika Patel

For Challenge Problem 3, our group worked with the PyBullet simulator and the Franka Emika Panda robot to implement inverse kinematics for drawing on a virtual blackboard.The provided starter code provided the core environment setup, including a fixed base Franka Emika Panda robot arm and a blackboard positioned in an *y-z* plane at $x = 0.5$. Our task was to compute a sequence of end-effector positions and orientations that allows the robot to draw the letters "CU", as well as an outline of the CU Buffalo logo for an additional challenge.

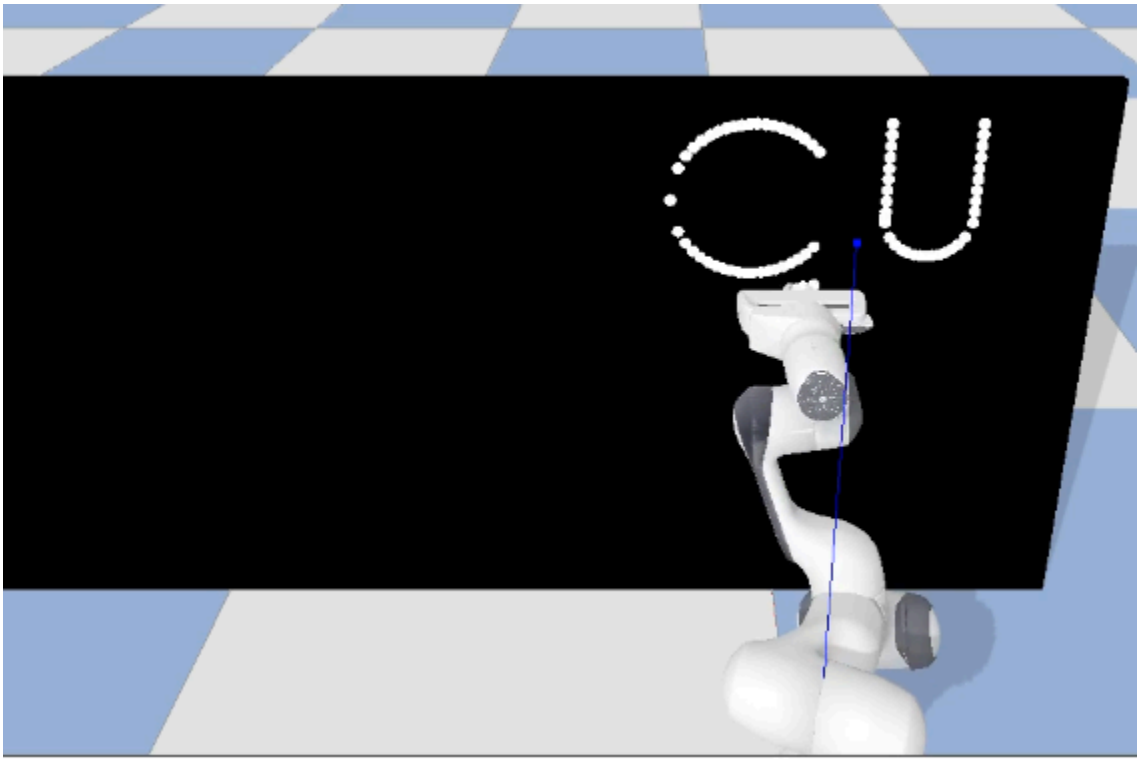
In addition to the provided functions *draw_on_blackboard* and other setup functions, we relied on PyBullet's inverse kinematics solver, *getInverseKinematics*, to convert the workspace coordinates into feasible joint configurations, instead of explicitly generating joint trajectories.

Our team encountered two issues with *getInverseKinematics*. Initially, it was obvious that either the function was limited in how far from its current orientation it could move per function run, or that *stepSimulation*, a PyBullet function used to advance time in the simulation space, did not allow the robot to fully complete a move to a given joint position. Especially when making large movements, like from its initial resting position to the goal position, the robot would draw one or more intermediate dots in space before it finally made it to the new destination. To remedy this, the function *find_loc_difference* was written and utilized to find the distance between the current robot arm position and the goal position fed to *getInverseKinematics*, and a loop consisting of a *getInverseKinematics* call followed by *resetJointState* and *stepSimulation* would be run as long as there was a significant distance between the current position and the goal. This functionality was combined in the function *move_to_point*.

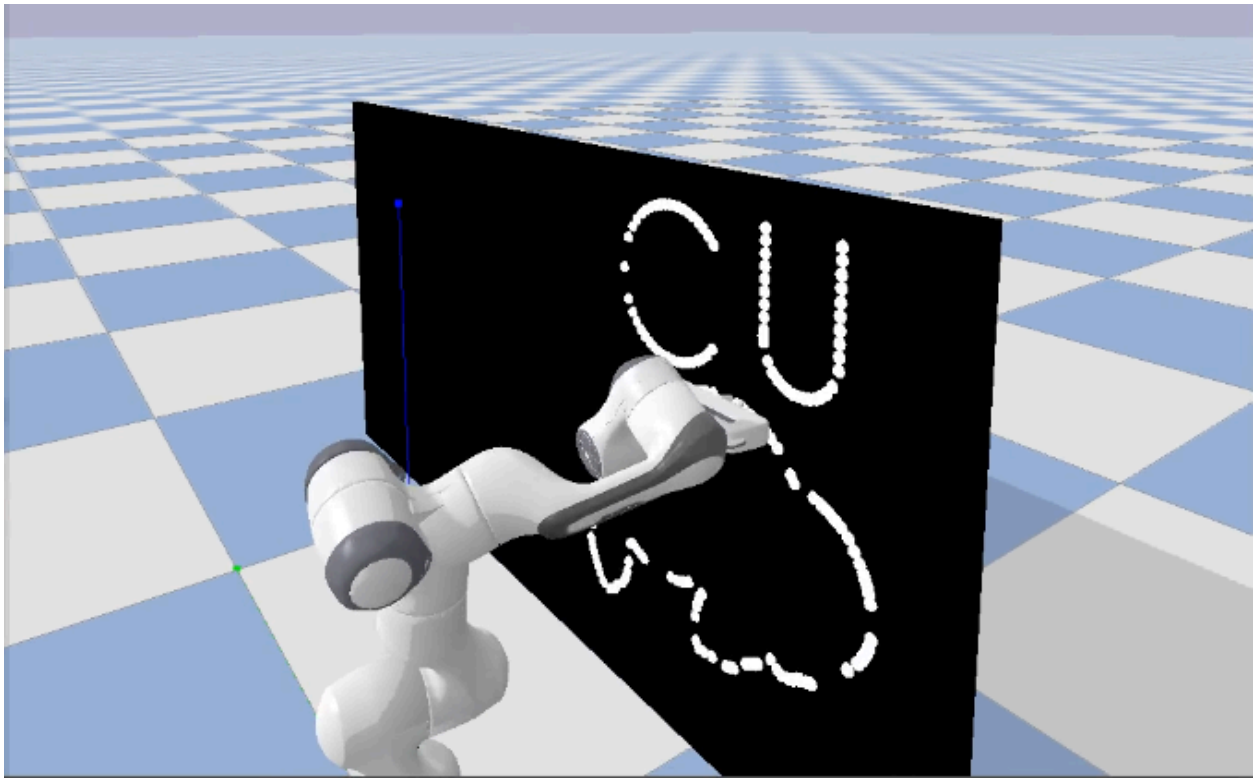
Additionally, *getInverseKinematics* had a tendency to generate angles the robot was unable to reach due to joint limits, resulting in timing out or other weird bugs. As *getInverseKinematics* does not automatically look in the robot ID to determine if there are joint limits to avoid, these had to be queried from the robot during startup using *getJointInfo*. Confusingly, this type of *getInverseKinematics* call requires the minimum and maximum positions in addition to the range of movement, which apparently cannot be calculated from the minimum and maximum positions within the function, as well as a preferred joint orientation. It also requires the joint limits of every single joint, including unmoving joints, unlike its own output and contrary to information implied in the documentation. Unlike *getInverseKinematics* input, *resetJointState* helpfully skipped immobile joints. To simplify inputs, *move_to_point* had a mode where only a y and z coordinate needed to be specified. The orientation would automatically be set to pointing at the blackboard with a quaternion, and the x coordinate would be set to 0.5, the location of the blackboard.

To form the letter "C" we created a parametric representation consisting of two semicircular arcs centered around a chosen base point. Through the Numpy function *linspace*, a continuous list of y points were created, which could be fed into an equation for a circle to get a z coordinate, each corresponding to positions on the blackboard. The "U" was constructed using

two vertical lines and a semicircle at the bottom, similarly sampled densely enough to produce a curve. These point sequences were fed into *move_to_point*, whose call was followed by *draw_on_blackboard* for the entire list of generated points. Executing this routine sequentially for all points produced readable “C” and “U” letters on the blackboard, as shown below.



The bonus task required extending our method to trace the CU Buffalo logo. For this, we imported a reference image and used OpenCV to convert it to grayscale, threshold it and extract its largest external contour. The contour was then scaled and shifted to correspond to positions on the blackboard. Since the raw contour consists of hundreds of points and would have slowed the simulation significantly, the contour was downsampled by selecting every other point. The resulting list was smooth enough to form a recognizable silhouette. The final image of the CU Buffalo logo, as well as the letters “CU” from the initial task can be seen below.



In the final simulation, the robot accurately draws both the “CU” lettering and the buffalo outline. Overall, this challenge problem demonstrated the effectiveness of combining geometric point generation with a constrained inverse kinematic solver to produce precise end-effector motion in a simulated robotic environment.

Link to Inverse Kinematics implementation: https://github.com/radzp16/CP3
Link to video: https://github.com/radzp16/CP3/blob/main/final_submission.mp4