Final Project Report

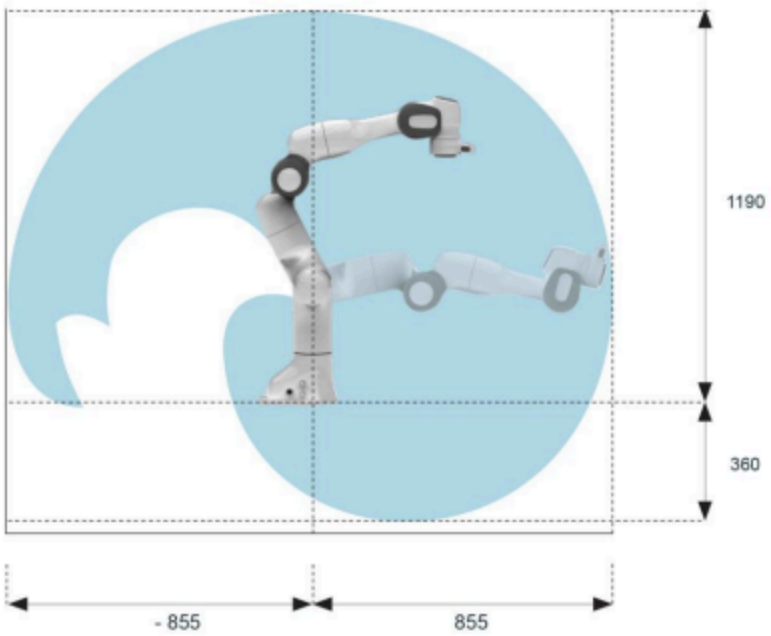
Clare Wu, Ryan Rudes, Eloise Zeng
https://drive.google.com/file/d/1-6IaItlRzracQgOu8Va9XeoPzmuFSt7W/view?usp=drive_link

Introduction

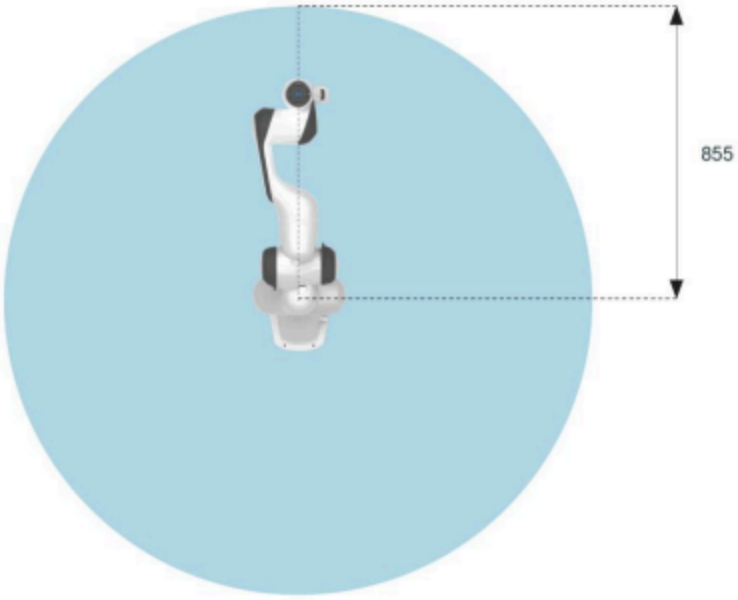
Our goal is to program the Franka Emika Panda robot, a 7-DOF robot arm with a paddle attachment, to smoothly intercept and redirect a tennis ball thrown with an arbitrary trajectory. We implemented a simple physics engine incorporating gravitational forces and collisions. Given the ball's initial position and velocity, we project its path in discrete time steps of 1 millisecond. Then, for each point along the ball's trajectory, the robot, starting from its non-singular zero configuration, simulates how it would intercept the ball at that location, matching the ball's velocity at the point of interception. We prevent the robot from attempting to catch the ball below the ground plane (z = 0). We select the ideal location to intercept the ball, using the condition number as a heuristic. After contacting the ball, the robot orients the paddle's circular face such that its normal vector opposes the velocity of the ball while gradually decelerating it to zero velocity. The duration of deceleration is a parameter that can be tuned to achieve desirable behavior such as catching the ball, then throwing it to the side, rolling it on the ground, balancing the ball, or directly dropping the ball.

Robot and System Description

To display the robot in Rviz, we used the franka_panda package, which was provided by the course. We used the panda_arm.xacro file, which has seven degrees of freedom. We fix the base at the origin. Then we modified the hand.xacro file to fix the position of the hand's two fingers/grippers. We constructed a URDF file for the paddle. We connected the arm, hand, and paddle in a urdf.xacro file such that the robotic arm has a hand that grips the paddle. The robot is able to pass through itself and has a maximum reach of 0.855 meters. The workspace of the robot is depicted below.



1. Arm workspace side view



2. Arm workspace top view

Source:
https://www.generationrobots.com/media/panda-franka-emika-datasheet.pdf

Task

We designate the tip of the robot as the center of the cylindrical paddle. The robot's primary task is to match the desired position, velocity, and orientation of the tip, without regard for its rotation about the z-axis in the tip frame. Thus, the task space has five degrees of freedom. Three DOFs are dedicated to positioning the paddle in xyz coordinates. Two DOFs are dedicated to orienting the vector that is normal to the paddle's circular face. Since the paddle has cylindrical symmetry, it doesn't matter how the paddle is rotated about this normal vector when catching the ball. The robot uses its two extra degrees of freedom to perform its secondary task: moving its joints toward the robot's initial configuration, which is nonsingular and comfortable for the arm to move freely in. This hinders the robot from locking up or swinging its limbs at high velocities.

The task isn't achievable when the ball doesn't cross the workspace of the arm. We eliminate this scenario by generating an initial ball position and velocity that ensures the ball passes through the robot's workspace. If the ball exits the robot's workspace, we freeze the arm so it doesn't twitch as it attempts to interact with the ball.

Algorithm and Implementation

Our robot's behavior is divided into three stages.

In stage zero, we run a planning algorithm to determine the ideal location at which to intercept the ball, which has a randomized trajectory crossing the robot's workspace.

In stage one, the robot intercepts the ball at the ideal location with the visualizer enabled. The paddle moves at the same velocity of the ball at the instant of first contact. The z-axis of the tip frame opposes the velocity vector of the ball, so the paddle is normal to the ball's motion (refer to the pseudocode below).

In stage two, the robot decelerates the ball to zero velocity. This is an unstable equilibrium, so we are concerned with keeping the ball on the paddle. Hence, the paddle follows the ball as it rolls. Once the allotted deceleration time is up, the arm stops moving, thus letting the ball go.

```
stage_0():
   Get Jacobian at given position
   Get average link_length of arm

   Scale first three rows of Jacobian by link_length
   Compute condition number

   return condition number, joint values, tip position and velocity
```

```
generate_trajectory():
   acceleration = [0, 0, gravity]
   for t over given time:
      # integrate ball velocity and position
      velocity += dt * acceleration
      position += dt * velocity
      add velocity and position to list
```

We choose the point of interception at which the arm's condition number is minimized. This ensures the arm can move freely after the point of interception.

```
interception_points = [points along ball's trajectory equally spaced
through time]

best_condition_number = infinity (initially)
```

```
plan_interception():
   for point in interception_points:

      Check in range:
         If yes, continue
      Check if can reach in time:
         If yes, continue

      cond = stage_0()
      if cond < best_condition_number:
         intercept at this point
```

Then we move the arm to match identified point of interception and ball's velocity at this point

```
ik_intercept_ball():
   # desired position and velocity
   pd, vd = spline(t, intercept_time, initial_position, intercept_position,
initial_velocity, intercept_velocity)

   # Find appropriate angle and rotational velocity
   intercept_angle = angle between initial paddle norm and paddle norm when
catching ball
   alpha, alphadot = goto(t, intercept_time, 0, intercept_angle)

   # find axis to rotate around
   n = cross(original_nz, intercept_norm)

   # desired direction of normal vector
   nzd = Rotn(n, alpha) @ intercept_norm
   wd = n * alphadot # desired rotational_vel

   # find current tip position, orientation, Jacobians for velocity and
rotational velocity
   ptip, Rtip, Jv, Jw = fkin(qdlast) #current joint positions
   wd_wrt_tip= Rtip.T @ wd then get rid of last row
   Jw_wrt_tip = Rtip.T @ Jw then get rid of last row
   # don't care about last row because don't care about rotation around
tip's z-axis

   J = combine Jv and Jw_wrt_tip
   calculate errors
   vr = vd + (lambda * error_p)
   wr = wd_wrt_tip + (lambda * error_nzd)
```

```
   xdot = combine vr and wr
   take weighted inverse of J (J_winv)
   qdot_secondary = lambda2 * (initial q - current q)

   qddot = J_winv @ xdot + (I - J_winv @ J) @ qdot_secondary

   qd = current qd + (dt * qddot)

   return (qd, qddot, pd, vd, nzd, wd)
```

We use the weighted inverse of the Jacobian to help the robot avoid a singular configuration. The paddle decelerates the ball such that the ball doesn't bounce off the paddle.

```
SLOWDOWN_TIME = time assigned to balance ball
# time can't be too short because the ball will hit the ground
#also changes with stickiness of paddle

ik_decelerate_ball():
   deceleration = -(ball_velocity / (SLOWDOWN_TIME - (t -
self.intercept_time))
   ball_direction = ball_velocity / norm(ball_velocity)

   calculate desired velocity (vd) and position (pd) based off ball
   vd = ball_velocity + deceleration * dt
   pd = ball_position - ball_direction * (TENNIS_BALL_RADIUS + PADDLE_THICKNESS
/ 2) # the paddle is opposing the ball_direction. Thus, we desire the paddle to
move in the opposite direction of the ball. We also ensure the paddle remains
tangent to the ball.

   nzd = (ball_position - pd) / norm(ball_position - pd)
   # change orientation of paddle to follow ball as it rolls on the paddle

   # similar to previous function
   ptip, Rtip, Jv, Jw = fkin(qdlast) #current joint positions
   calculate wd_wrt_tip
   calculate Jw_wrt_tip

   J = combine Jv and Jw_wrt_tip
   calculate errors
   find vr and wr
   xddot = combine vr and wr
```

```
    J_winv = weighted inverse of J
    qddot = J_winv @ xddot
    qd = current q + (dt * qddot)

    return (qd, qddot, pd, vd, nzd, wd)
```

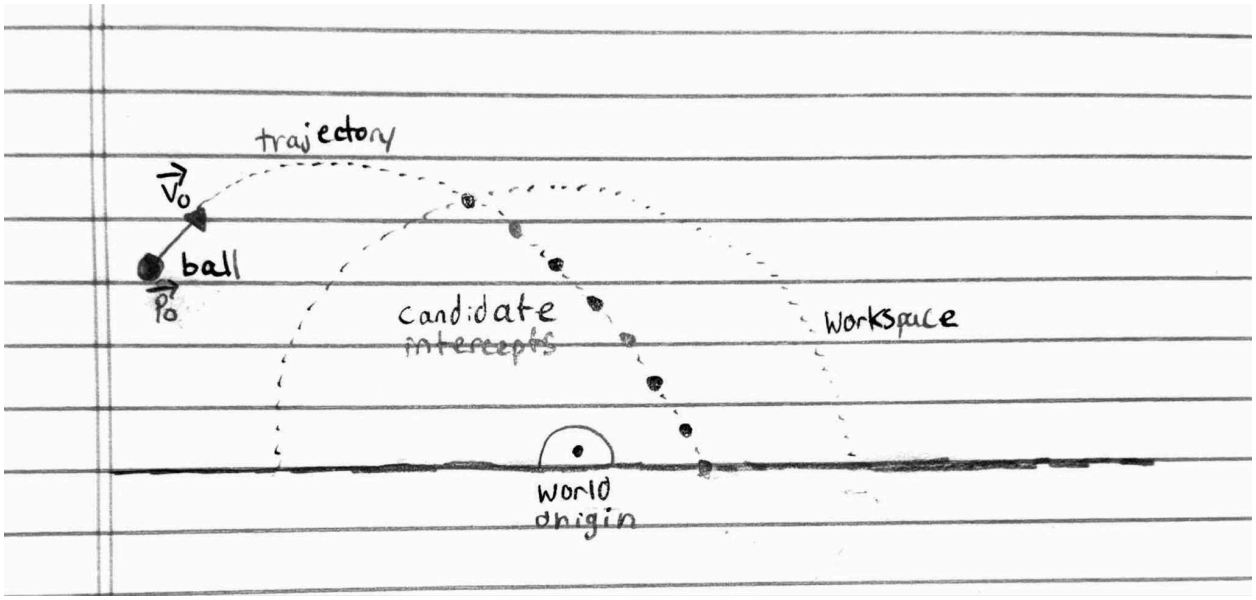
The ball and the paddle will always contact each other as long as their velocities match after interception. If they don't, the ball is prone to bouncing. Since the robot is unable to match the desired velocity perfectly, we introduce a STICKINESS constant in the physics engine. If the magnitude of the difference between the paddle and ball velocity (in the direction of the paddle's normal vector?) is less than the STICKINESS constant, we adjust the ball velocity to match that of the paddle in the direction of the paddle's normal vector. The ball is free to roll relative to the paddle. This simulates how in real life, the ball can squish or stretch a little and still remain in contact with the paddle.

Particular Features

One way we deal with singularities is by avoiding them with our planning in stage 0. Assuming total information regarding the dynamics of the ball's motion, we forecast its trajectory ahead of time based on the initial conditions (position and velocity) and a discrete-time physics update. We approximate the workspace of the robot arm as a hemisphere above the ground plane of radius .855 m centered at the world origin and consider the section of the trajectory that intersects the hemisphere. We subdivide that parabolic curve section into points ("candidate intercepts"), equally spaced throughout time by 1 millisecond.

For each candidate intercept, we (without visualization) unroll our stage one arm controller (see stage_0) as if we were to intercept the ball at that location, then record the condition number of the robot arm's configuration at the moment of interception (see plan_interception). After testing each candidate intercept, we choose the intercept that yields the optimal configuration, which occurs when the condition number is minimized.

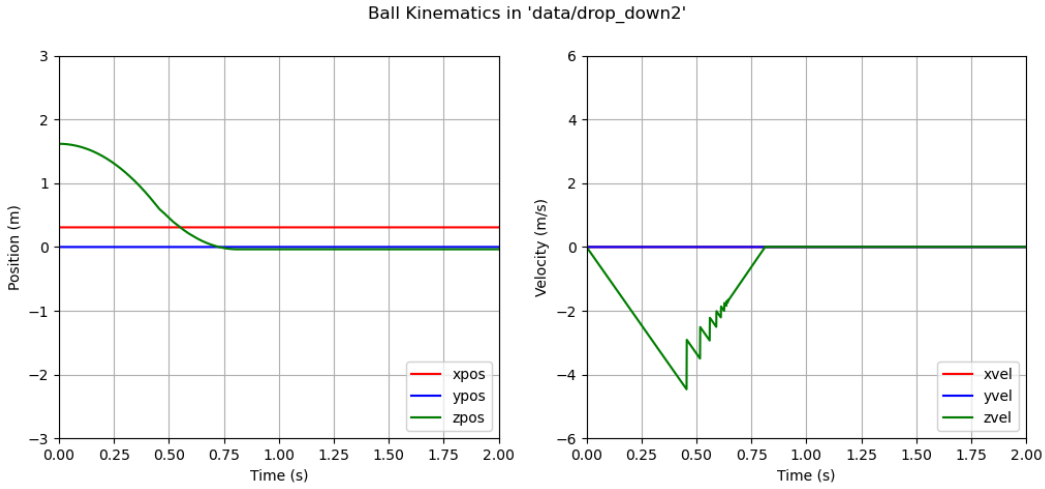
The motivation for this planning is that after intercepting the ball, the robot must react quickly and smoothly to decelerate the ball without allowing it to bounce off of the paddle. Therefore, we intercept the ball where the robot has the most freedom to move thereafter. The robot tends to catch in the middle of its workspace, about half the arm's length from the origin.


trajectory
$\vec{V}_0$
ball
$\vec{p}_0$
candidate intercepts
workspace
world origin

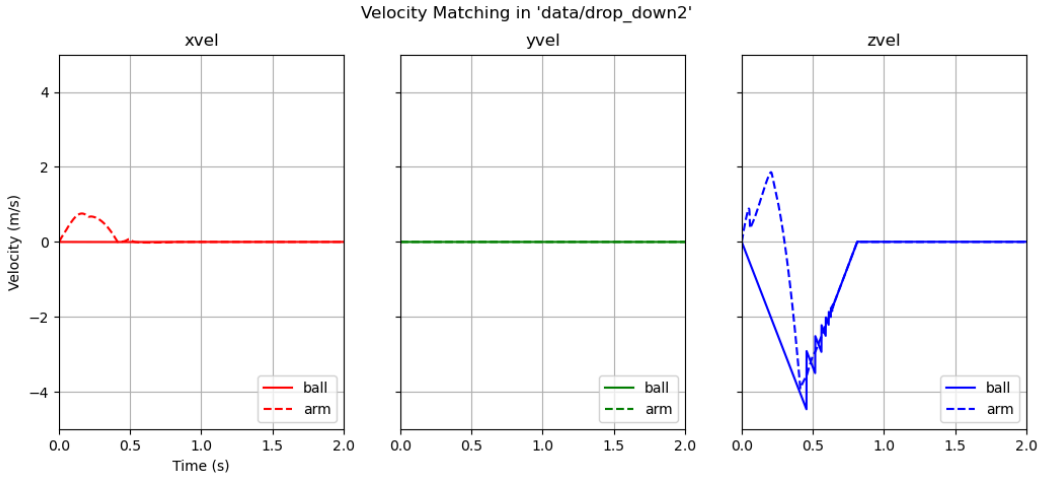
Analysis, Plots, Other Materials

Drop Down

In this demonstration, the ball is dropped from rest 1.0 meter above and 0.2 meters in front of the initial tip position. First contact occurs at roughly t=0.45s, at which point tiny discontinuities in the ball velocity can be observed each time it bounces. The smaller these discontinuities, the "smoother" our interception and subsequent slowdown.



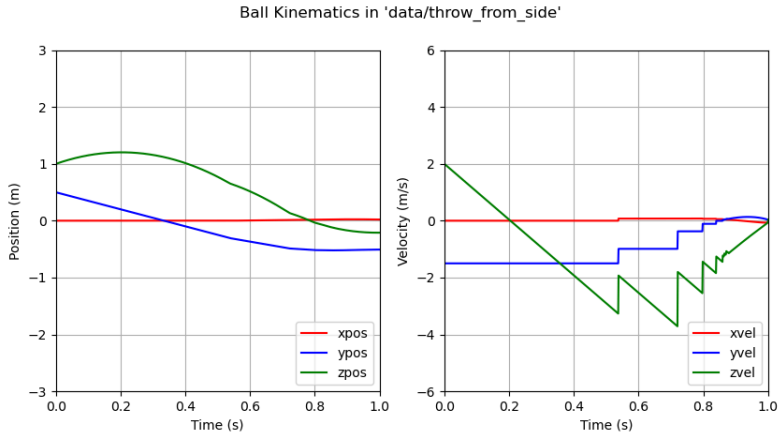
The ball is smoothly brought to rest at roughly t=0.8s.



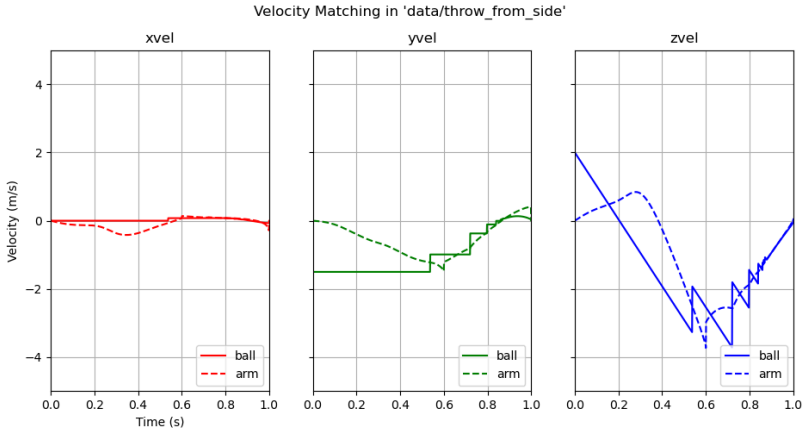
Comparing ball velocity to tip velocity; these should be identical at the instant of first contact.

Throw From Side

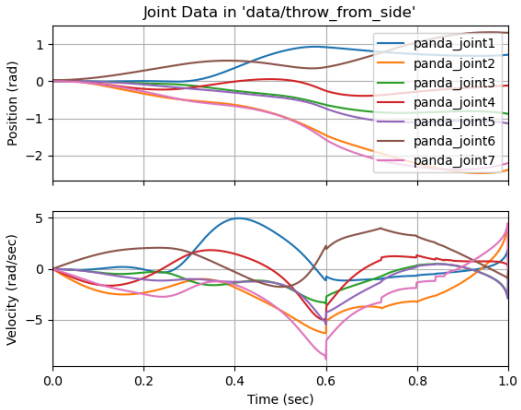
In this demo, the ball is tossed “underhand” from the side of the robot.



First contact is made at roughly t=0.55s and the ball is brought to rest at t=1s.



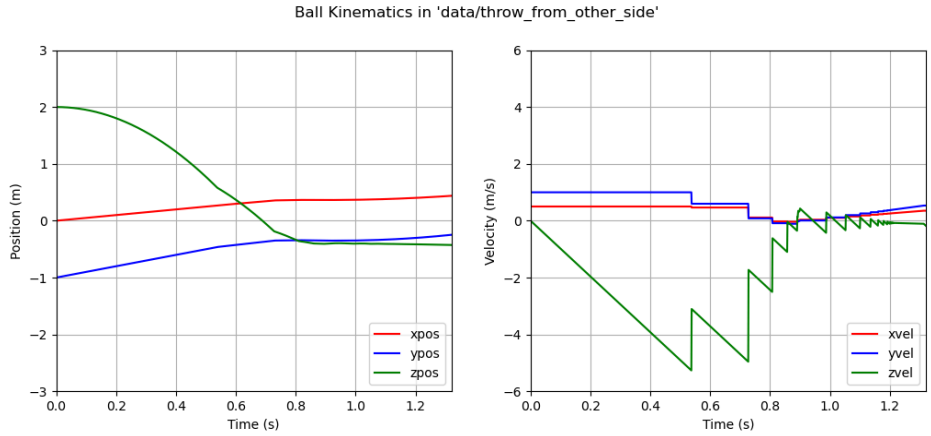
From the moment of first contact onward, the paddle moves with nearly the same velocity of the ball, but with just enough gradual deceleration to eventually bring it to rest.



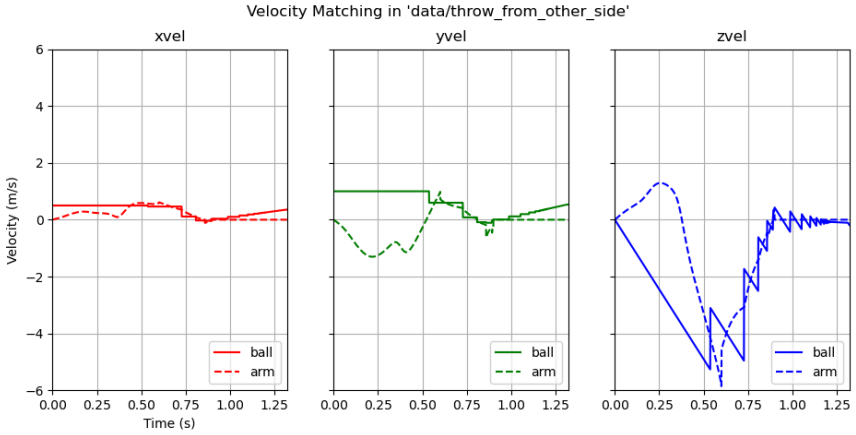
Joint velocities remain fairly smooth over the duration of this complicated interception, demonstrating little to no instantaneous accelerations.

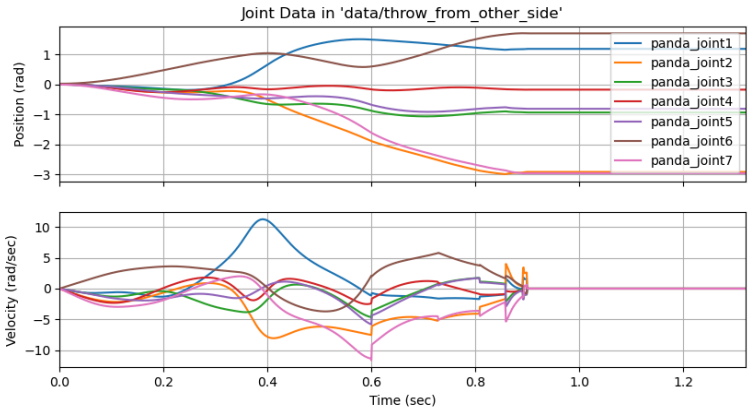
Throw From Other Side

In this demo, the ball is tossed “underhand” from the other side of the robot.



First contact is made at roughly t=0.55s and the ball is brought to rest at t=1.2s.





Joint velocities remain fairly smooth over the duration of this complicated interception, demonstrating little to no instantaneous accelerations.