Risk of ROS 2 Presents

The Amazing SPOT!

By Julian Millan and Lucas Ancieta

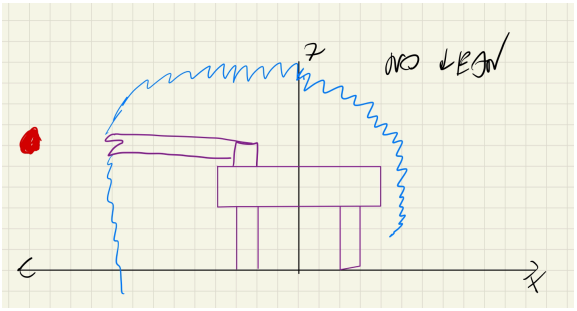
1. **Overall Description / Introduction**

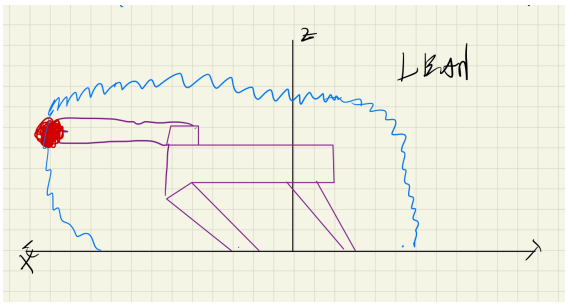
   Our main goal with this project was to have a four legged SPOT robot with a URDF modified to have a 7 degree of freedom arm on top of its body, stand with its feet planted still and follow/catch a bouncing ball. The planted feet mean that the robot has to coordinate its two hip joints and one knee joint (three joints per leg) to move its body closer to the bouncing ball. The trajectory of the ball is just outside of its standing reach, so the lean created by the leg joints allow the arm to move close enough to “grab” the ball. This means a big requirement for this project was no walking cycle, and that the feet would stay in the same place no matter what. This project involved trigonometry, trajectory calculations for the ball, gravity, and time steps.

2. **Robot/System Description**

   Each of the four legs had three joints to consider (all revolute): The knee joint (rotation about the y-axis), a hip joint about the x-axis, and a hip joint about the y-axis. The added arm we included via an edited URDF has an additional 7 DOFs to consider, bringing the total up to 19 DOFs. Our workspace looks like a shifted elliptical, limited by the length of the arm. However, we can coordinate the legs to have the robot lean certain directions to increase the reach of this arm. The feet must be fixed to their points on the ground, so this expanded workspace is limited by the length of the legs.

   Below is a rough workspace sketch:





3. **Task Description**

   The task is primarily to catch the ball while keeping the robot’s feet planted. We can express our task space in the simple x,y,z coordinates of the world. We have

a goal position symbolized by the center of the moving ball. We can track how close we are to achieving the task by comparing the distance from the end effector of SPOT's arm to the ball position. We consider the task "achieved" during the catch and release phase of the project when the hand gets within a certain radius of the ball (visually, the hand looks like it's actually inside the ball). This is when we pause the program until it gets restarted, and SPOT must wait 20 seconds before it can catch the ball again. The ball catching task is what the entire body works together to accomplish. The arm always seeks the ball position, the body shifts slowly towards where the ball is going, and the legs bend and shift to allow the body to move while not moving the feet. The two tasks of keeping the feet planted and chasing the ball are of equal importance, unless the ball is outside of the leaning reach. If the ball is outside of the leaning reach, keeping the feet planted takes priority and the robot will simply follow the ball with its arm/body as best as it can. Keeping the feet planted is always achievable (unless we send the body flying in the code, of course). Most of the time, the ball is not catchable, but it is always followable, which is still executing the main part of the task effectively. The arm and body subsystems work together to follow the ball, while the four legs work together to all keep their feet planted and bend to follow the body.

4. Algorithm and Implementation

We have multiple unique aspects as part of our project. First, we create a ball object and generate an elliptical motion around the robot (world space). The ball is affected by gravity and when it hits the ground, it bounces back up again. We have a simple boolean that when positive allows the ball to be caught. When the ball is caught (end effector within the ball's radius by a certain margin, found using forward kinematics on the arm chain), the program stops for five seconds before running again. The ball will then go back to bouncing, but for twenty seconds the robot cannot catch the ball (boolean is set to negative). This makes sure the system isn't just always sleeping once the ball is caught, and allows the ball to be in a different position when it can be caught again (not just one spot). To track the ball, we build up five kinematic chains, one for the arm and four for the legs. We then specify the position of each of the feet and create T-matrices to show the feet relative to the world.

Below is a code snippet showing our code for the body position and movement:

```
Rbody = Reye()
vbody = pxyz(self.v[0,0], self.v[1,0] * 1.5, (0.3205 * sin(self.q[0]) - 0.3205 * cos(self.q[1])) * 0.3) * 0.01
pbody = self.pbody + vbody * 0.15
Tbody = T_from_Rp(Reye(), pbody).astype(np.float64)
self.pbody = pbody
```

We keep the body level and have its velocity follow that of the ball (represented with self.v). For its height, we know that bending at the hip should cause it to be

shorter. Taking both hip angles into account, we lower the body when needed in order to keep the feet on the ground. We then integrate the velocity to move the position. After that, we find the transforms for each of the feet relative to the body and also find the position and velocity of the ball relative to the body. We then find the linear and angular Jacobian for the arm and the linear Jacobians for the legs (all done using the fkin() function). These are all placed inside one big 18x19 Jacobian.
Below is the end of the algorithm:

```
# combine into one vector
xrdot_all = np.vstack((vr_fl, vr_fr, vr_rl, vr_rr, xrdot_arm))

# full calculation via weighted inverse
J_winv = np.transpose(J_all) @ np.linalg.inv(J_all @ np.transpose(J_all) + self.gam**2 * np.identity(18))
qdot = J_winv @ xrdot_all
qdot = qdot.astype(np.float64)

q = self.q + qdot * self.dt
q = q.astype(np.float64)
self.q = q # update stored q value

if self.catch == False and np.linalg.norm(parm_fkin - self.p) < 0.55: # check to see if we catch the ball
    time.sleep(5)
    self.catch = True
    self.catch_time = self.t
```

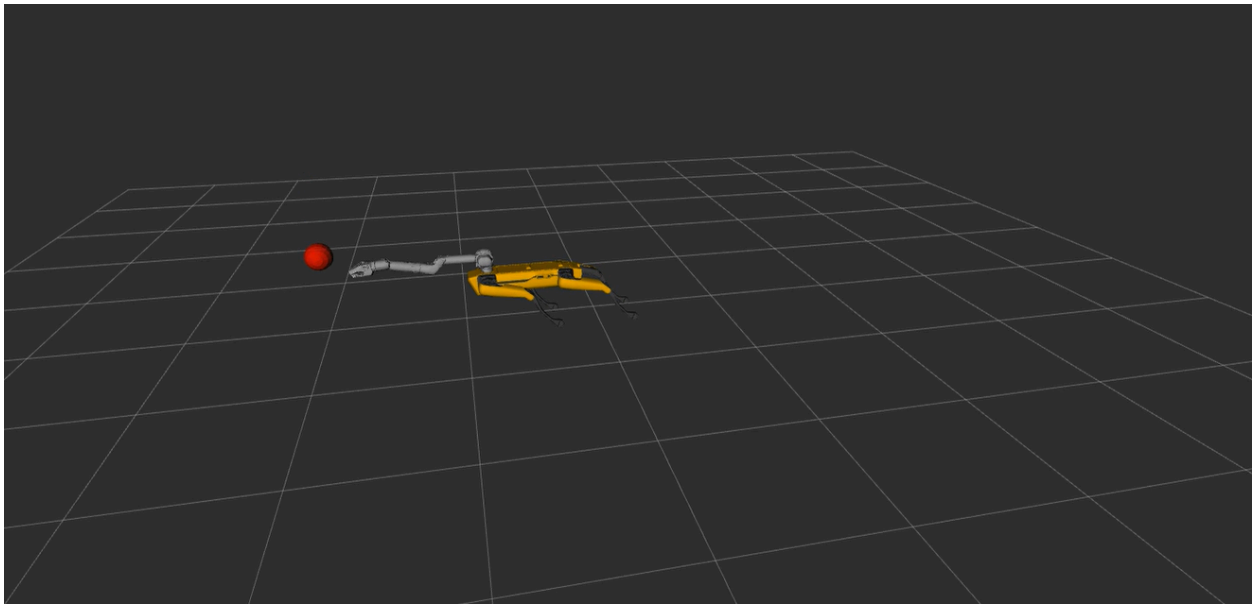
We use a weighted inverse with a gamma of 0.095. This allows us to keep the arm stable and accurate as it spends a lot of time stretched almost completely out (a prime spot for a singularity). We find the angle velocities needed by multiplying the weighted inverse Jacobian by the aforementioned linear aspects of the legs and linear/rotational of the arm. We integrate this and save it. At the bottom, we check the previously mentioned catch boolean and the distance to the ball's center from the end of the arm. When this if statement is triggered, we have a successful "catch", resetting the boolean and starting the catch cooldown timer.

5. Particular Features

As mentioned before, there is a near constant singularity we have to watch out for with the arm, since the ball path is such that the arm is mostly straight the entire way. The weighted inverse and also error vectors we consider during the forward kinematics allow us to balance things properly. Below is an example of our fkin() calculations.

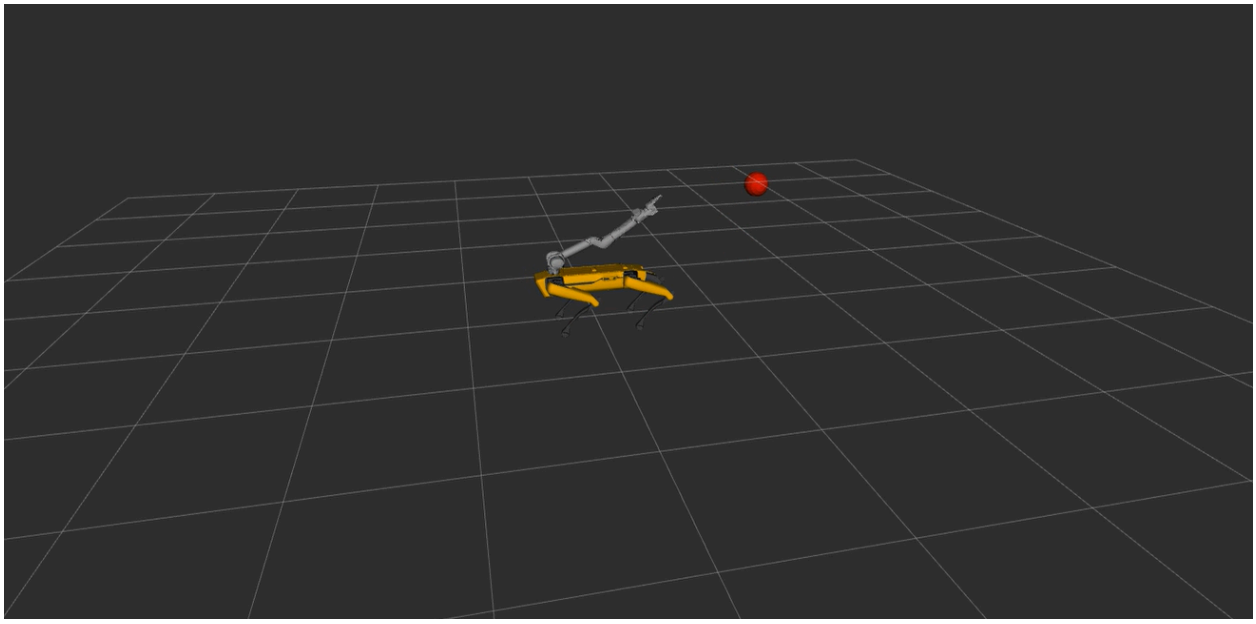
```
# Compute the front left leg joints
(pfl_fkin, Rfl_fkin, Jv_fl, Jw_fl) = self.fl_chain.fkin(self.q[0:3]) # old forward kinematics for arm
vr_fl = np.transpose(Rbody) @ -vbody + self.lam * ep(p_from_T(T_FL_foot_body.astype(np.float64)), pfl_fkin)
```

We see the robot handle being in the limit of its workspace quite well in the below picture:



The robot is stretched at its horizontal limit in the above image. We see the arm is mostly straight with a maximal lean from the legs. Still, the ball is just outside of reach. Note how the knees are bent and angle to compensate for the lower robot height to get it closer to the ball, and that the feet are still planted. Our most important task of keeping the feet planted is preserved while still doing our absolute best to reach for the ball.

When the ball gets directly behind SPOT, the workspace is very limited.

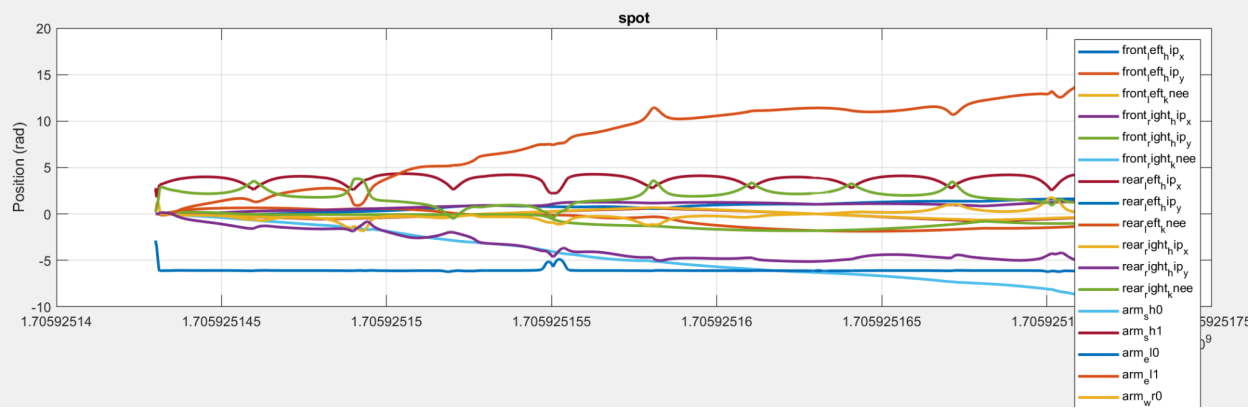


You can see the degree of lean is much less than in front of SPOT. This is because SPOT's knees have physical limitations. To achieve the same lean as before, it would need to completely invert its knees, which would have very high

joint velocity, physical issues/incapabilities, and is just generally impractical. So, we simply use the knees for height here and do our best to shift to the ball. Again, prioritizing the feet and stability over grabbing the ball.

6. Further Analysis

Below is a plot of the joints during 1–2 cycles of the ball moving (no catch and release, only following).



We see the legs generally have a very smooth and fluid trajectory (especially the knees), while the arm has some spikes now and then. This makes sense as the arm has the most movement in the task space and is constantly subject to singularities and being unable to reach its task. Under the catch and release condition, we would see something similar, except big pauses for five seconds during the sleep state. Parameters tweaked include lambda (used for the error vector calculations during the fkin() phase), gamma (used in the weighted inverse), the required distance between the ball and end effector for a catch, and the body velocity trajectory. While we had a good idea of where to start with these parameters, tweaking them involved a lot of trial and error and to determine what looked the best/smoothest.

```
vbody = pxyz(self.v[0,0], self.v[1,0] * 1.5, (0.3205 * sin(self.q[0]) - 0.3205 * cos(self.q[1])) * 0.3) * 0.01
pbody = self.pbody + vbody * 0.15
```

For the above lines concerning the body, notice how everything is on the magnitude of about 1/100th the speed of the ball. This ensures that the body doesn't stretch too far and the feet come off the ground. During testing, we noticed that very minimal movement happened in the y-direction, yet the x-direction worked great. This is likely because the x-direction has both a hip and knee joint moving it for each leg, while the y-direction only has one hip joint. Scaling this up a bit compensated for this and allowed more effective movement. The movement in the z-direction uses the length of the upper leg found from the URDF (hip to knee) and the angles of both hip joints. A downward movement is always expected (no floating off the ground). We scaled this down a bit because too much speed wouldn't allow for the knees to bend in time and poor SPOT

would fall through the floor. Lastly, we see our velocity integration to be considerably greater than our typical 0.01 time step. This is because we wanted noticeable and big changes in body position so that the joints could have more movement to work with. Too small and the body would be barely moving at all, letting the ball get away and just generally lagging behind it. This integration allowed SPOT to keep up with the ball.