CS/ME 133a Final Report

video, github

By Allen Yang, Alex Gogola, Isabela Ceniceros

Introduction

The purpose of our project was to visualize a robot juggling a soccer ball with its feet in RVIZ. For this project Atlas was chosen as the robot since we had prior experience with it. Our goal was to give Altas the task of keeping up a ball with his foot, and let Atlas determine how to utilize its DOF's to do it. The soccer ball was represented as a simple orange ball which followed a simple trajectory formula.

ATLAS Description

Atlas is a humanoid robot with a total of 30 degrees of freedom, and we utilized 23 of them (we didn't use the left arm). Atlas' workspace is determined by the reach and mobility of its limbs which allows it to perform a variety of tasks. In RVIZ, Atlas is capable of both fixed-base and moving-base operation. While we implemented our code in a way so that Atlas has a moving base, we fixed the right leg to the ground for our project. To do this, we calculated the position of the pelvis with respect to the world frame so that the right foot was in the desired position each iteration, as the pelvis was chained to the world frame. We felt this was best because for our given task, we only would've needed a moving base if we wanted to alternate legs juggling. However, we decided we want to focus more on implementing more tasks with one leg rather than the same tasks for both legs. While there were no built in constraints for the joint values, we implemented a secondary task to keep the joints at human like angles.

Task

The objective of our code was to control Atlas' trajectory based on the trajectory of our ball and the collision of the ball and Atlas' left foot. The primary task was for the left foot of Atlas to track and meet up with the ball at the bottom of its trajectory. The ball's trajectory after collision as well as Atlas's foot trajectory was not predetermined and was calculated dynamically. The height of each collision was randomized to be between 0.1m and 0.5m, and the angle of contact was randomized to be between -0.05rad and 0.05rad, giving some variance in movement between collisions. Further, we assumed close to perfectly elastic collisions, and thus had the final velocity of Atlas's foot be 0 at each collision. This primary task had 6 dimensions (3 translational and 3 rotational). Aside from kicking the soccer ball, we also gave Atlas the ability to hold a plate (or anything) in its hand as another primary task by fixing its right hand's position and rotation. This task also had 6 dimensions (3 translational and 3 rotational). Finally, we also fixed the x-y position as well as the rotation of Atlas's head in order to better simulate balance. This task had 5 dimensions (2 translational and 3 rotational). Even with three primary tasks with a total of 17 dimensions, the higher dimensionality of Atlas's joint space allowed for all these tasks to be completed, as long as they were within Atlas's task space (which we insured). We also implemented a secondary task that restricted the angle of each joint we utilized. While this is a 23 dimensional task, these restrictions were only enforced if the execution of the primary tasks allowed it.

Algorithm and Implementation

Nodes.py

In order to have Atlas perform the task 3 nodes were created, one for the ball, one for the robot, and one for the GUI slider. The ball's movement is governed by a constant gravitational acceleration, and its position is constantly computed with numerical integration of its velocity. The Robot node controls the movement of the robot, storing information required for computation of its trajectory, like the ball's movement. The GUI slider gives users the ability to manipulate the height of the "plate" in the robot's right hand. These three nodes are controlled by a "pseudo/wrapper node" that is responsible for running all three nodes concurrently as well as passing data between the nodes correctly. This helped with efficiency, as communication between nodes didn't rely on publish/subscribe messages but was directly controlled by this wrapper node, and we were still able to have some kind of object oriented design.

Main.py

The main script contains the Trajectory class, which computes both task trajectories and determines the robot's joint states. After each collision, the trajectory class randomly generates the next height and angle it wants to collide at. Then, using the kinematic equations for constant acceleration, we can solve for the time it takes to go up and the down and final position of the ball at the desired final height. Using this information, we used a simple cubic spline with initial and final velocities of 0 to determine the trajectory of the robot's left leg.

For each iteration, the trajectory of the right arm was determined by how far it is from the desired position. We simply determined xdot of this task to be directly proportional to the error by a factor of 0.001. The desired z position was controlled by a GUI slider that was running concurrently with the robot and ball nodes. Everytime the GUI slider was adjusted, the wrapper node automatically passed the new desired position to the robot node. Even though these nodes were in different threads, this implementation is still thread safe because the desired position was controlled only by the GUI slider. We implemented the fixing of Atlas's head in a similar way, but instead using a xdot of 0 and solely adjusting the position using the positional and rotational error. The above three tasks were all computed in the world frame.

For the secondary task of restraining the joints, we drove the joint positions toward the center of each restraint interval by a qdot directly proportional to the distance from the center of the interval and inversely proportional to the size of the interval. We implemented it this way so that joints with more constraints were driven harder towards these constraints. Each interval was also smaller than 2pi, which helps prevent too much twisting. Furthermore, for joints like the knee, restricting the angle to be greater than 0 helps with switching between "elbow up" and"elbow down" solutions, which helps deal with singularities. This task was directly computed in joint space.

To solve the inverse kinematic problem, we constructed one single 17x23 Jacobian and concatenated all of these tasks to be solved by one Jacobian that was with respect to the right foot, which allows for less "implicit" constraints. We had to convert all positions and orientations to positions and orientations relative to the right foot and in the right foot's frame.

$$
{}^{\mathrm{f}}|p_{r-f} = {}^{\circ}|R_f^T \;\left({}^{\circ}|p_{r-o} - {}^{\circ}|p_{f-o}\right)
$$

$$
{}^{\mathrm{f}}|R_r^T = {}^{\circ}|R_f^T \;{}^{\circ}|R_r
$$

Note that the trajectory of our foot is computed in the world frame, so we needed the position and orientation of the right foot in the world frame. This was a matter of simple 3d frame transformation.

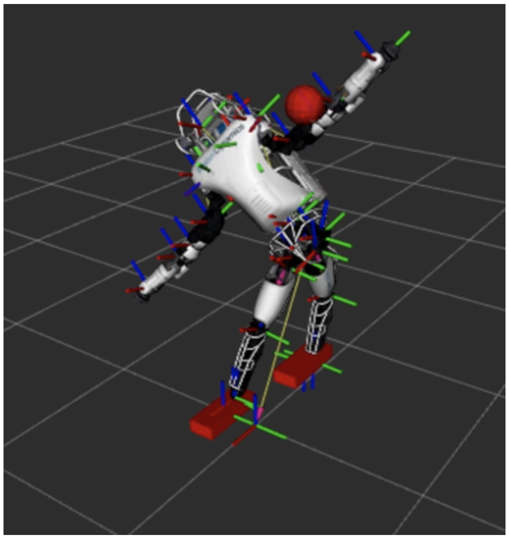
We also had to convert the Jacobian. Since the positions, orientations, and Jacobians were all computed directly with respect to the pelvis using forward kinematics, we used these values directly to compute the Jacobian.

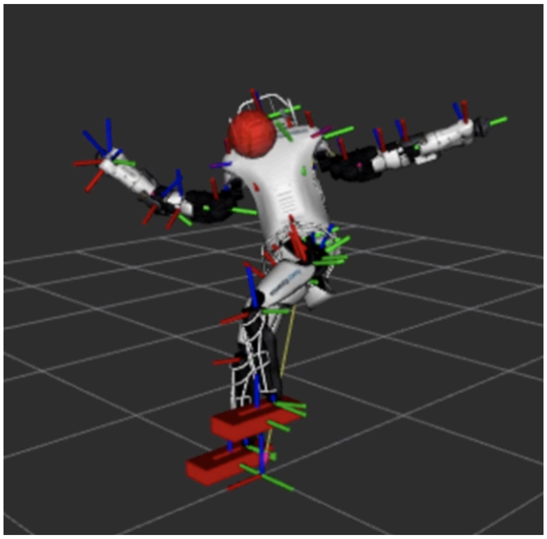
$$
{}^{\mathrm{f}}|J_{r-f}^{v} = {}^{\circ}|R_f^T \;\left({}^{\circ}|J_{r-o}^{v} - {}^{\circ}|J_{f-o}^{v} + \left[{}^{\circ}|p_{r-o} - {}^{\circ}|p_{f-o}\right]_{\times} {}^{\circ}|J_f^{\omega}\right)
$$

$$
{}^{\mathrm{f}}|J_r^{\omega} = {}^{\circ}|R_f^T \;\left({}^{\circ}|J_r^{\omega} - {}^{\circ}|J_f^{\omega}\right)
$$

$$\begin{array}{cc} & 23 \\ 17 & \begin{bmatrix} & & \\ & J & \\ & & \end{bmatrix}\end{array} \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix}\begin{array}{l} \}\ \text{l-leg } 6\times1 \\ = \\ \}\ \text{r-leg } 6\times1 \\ \}\ \text{r-arm } 10\times1 \\ \}\ \text{neck } 1\times1 \end{array}\begin{bmatrix} \\ \\ \\ \end{bmatrix}\begin{array}{l} \}\ \text{l-leg - } 6\times1 \\ \}\ \text{r-arm - } 6\times1 \\ \}\ \text{head - } 5\times1 \end{array}$$

Furthermore, finding the pseudo inverse solution of this redundant system automatically minimizes the norm of the joint velocities, allowing for better movement. To calculate the pseudo inverse, we used the weighted SVD inverse with gamma=0.01 for some basic singularity handling.





The first picture demonstrates the robot before joint restraints, and notice that the left knee is bent inwards. By implementing the secondary task of having the knee at a positive angle, we were able to minimize the amount of movement done near the singularities. Further, we can see that the robot is oriented in a way such that movement between collisions across all joints in the chain are minimized, which is a direct byproduct of computing the pseudoinverse.

URDF:

The URDF of Atlas is already made, however Atlas' feet are not flat which would not allow for the collision. We altered the URDF to include a paddle-like box onto the foot. This box was then linked to the foot as a fixed joint. This allowed the collision to be simulated similarly to that of a paddle and ball.

Particular Features

To model collisions, the robot node had to first check whether the ball was touching the box at each iteration. To do so, the robot first converted the ball's position (with respect to the world) to the position with respect to the left foot frame. Then, checking if the ball collided was a simple matter of checking if its coordinates with respect to the left foot were within the bounds of the box. If a collision was detected, then the orientation matrix of the left foot during the collision is passed to the ball, and the ball's velocity is both reversed and rotated according to the left foot's orientation. This adds a degree of freedom to the ball bounding, as it is now not just constrained to a straight line up and down. Further, implemented in this way, the ball is able to bounce off both the bottom and top of the foot.