ME/CS/EE 133A Project Final Report
Right Footed Atlas

Sophia Steven, Lauren Garriques, Kaylor Cruz

December 2023

1 Introduction

In our final project, Boston Dynamics' dynamic humanoid robot, Atlas, is modeled as reacting to an incoming ball and "kicking" it, reaching out its leg to meet the ball on a predetermined trajectory, and changing the sign of the ball's velocity vectors. Kinematic equations that determine the place and time of intersection between Atlas' right foot and the ball are utilized to determine the ball's trajectory. This collision location and elapsed time are used to determine Atlas' behavior in kicking the ball, as they are used as arguments for the kinematic equations driving Atlas' joint motions.

Atlas was originally modeled with a simple 6 degree of freedom (DOF) kinematic chain. Out of the 30 joints Atlas is supplied with, we locked all but 6 to isolate the joints in the right leg. Here, the pelvis was used as the world frame, and the right foot as the tip frame of the chain. The final position of the foot was determined directly in the code using the kinematic chain function from the homework sets, along with a Jacobian matrix to determine the joint behavior based on these constraints.

At first, Atlas would bend its joints the wrong way in order to move its foot to the desired position. Atlas' joints behaved strangely at its original straight-legged position, due to the original position being a singularity. We adjusted Atlas' initial stance into a slight athletic stance, and it raised and lowered its right leg smoothly to reach the kick position.

As a 6 DOF robot is fully constrained in a 3D task space, there was no weighting on the joint speeds we could introduce to the kinematic calculations that could affect the mechanics of the kick. In order to create a more interesting and dynamic kick, we shifted Atlas' constraints to a 12 DOF system. This kinematic chain had the left foot as the world frame, and the right foot as the tip frame.

After the shift to the 12 DOF kinematic chain, the robot's path of its right foot to the kick position did not immediately change. We introduced weighting on the joints to change its behavior based on what joints were freed, allowing the left knee, ankle, and hip joints to contribute motion towards getting the right foot to its target. Slight changes in the allowed max joint velocities in the 12 DOF configuration created massive changes in the mechanics of Atlas' kick, due to redundancy of degrees of freedom in the task space.

All code can be found at the following GitHub page:
`https://github.com/smsteven22/ME-133A-Final-Project-Right-Footed_Atlas`

2 Implementation of Ball Kinematics and Impact with Atlas

In order to simulate the movement of the ball, we wrote a Node class, `SoccerNode()`, that sets up the trajectory of the ball, creates the sphere marker and moves it along the trajectory with every time step dt, and continually checks the ball's position for hitting the ground or the robot's foot as it moves toward the collision point.

The ball's initial position, velocity, acceleration, and size in RVIZ are determined in the initialization function for `SoccerNode()`. The velocity magnitude of the ball is set in the range of $0\frac{m}{s}$ to $2\frac{m}{s}$ in the x and z directions, partly to achieve a reasonable motion window for Atlas, but more importantly to assist the visualizer. As we discovered with Professor Günter Niemeyer, the display fps is dramatically less than the 100 Hz rate. The initial x position of the ball is also constrained to ensure the x coordinate of the ball is within a reasonable distance of Atlas at the kick time. The only non-impact acceleration on the ball is gravity, valued at $-9.81\frac{m}{s^2}$ in the z direction. The trajectory of the ball is determined by a kinematic equation. We then use this manual ball kinematic equation to generate `pkick`, the position at which the ball will collide with the right foot of Atlas. We calculate the z position of `pkick` as follows, and the other coordinates similarly:

$$z_p = z_v t + \frac{1}{2} z_a t^2$$

where z_p is the z coordinate of the ball's position, z_v is the z coordinate of the ball's velocity, and z_a is the z coordinate of the ball's acceleration.

Next the sphere marker is created and initialized. The ball will keep traveling along its trajectory as governed by the spin() function, terminating the ball visualization once the ball has reached its target. While the ball is moving along its trajectory, the update function monitors if the ball hits the ground or the right foot every time step. It is important to note that the only information Atlas "receives" about the ball trajectory occurs in initialization in order to generate a target for the inverse kinematics; after this, the ball is only tracked as a physics object.

The ball making "contact" with the ground is defined as when the z position of the center of the ball travels below the z position of $z_p = radius$, where radius is the radius of the ball. If this occurs, the z position of the center of the ball is set to the $diameter - z_p$, where diameter is the diameter of the ball and z_p is the current z position coordinate of the ball. The z velocity of the ball is then reversed, "bouncing" the ball. The ball makes contact with the right foot once the x, y, and z coordinates of the ball are all within a radius length of Atlas' right foot, tracked as a point. Once the ball is within this collision sphere, the x position is increased by one radius length and the velocity is flipped in the x direction. These changes send the ball in the direction it approached from, effectively approximating a direct collision with the additional assumption that Atlas' mass is much greater than that of the ball.

3 Implementation of the 12 DOF Chain

Inside the `Trajectory()` class, the `jointnames()` function returns a list of all the joints found in the Atlas URDF. Within the total list returned by `jointnames()`, we created two subsection lists containing each leg's knee, ankle, and hip joints, called `right_leg_intermediate_joints()` and

`left_leg_intermediate_joints()`. In order to allow meaningful weighting of joints, introduce hip movement into the kicking motion, and explore different ways the robot can move to reach the same collision position with the ball, we introduced 12 degrees of freedom into the robot. To achieve 12 DOFs, we needed a 12 joint kinematic chain, which we created by defining two separate kinematic chains, `right_leg_chain` and `left_leg_chain`. Using the outputs of each respective chain's `.fkin()` function call, we were able to determine the robot's q and $\dot{q}$ values.

To achieve the 12 DOF robot, we expressed the right foot as the tip frame with respect to the left foot as the world frame. As a result, a transformation of the position, Rotation matrix, and Jacobian components extracted from each chain's `.fkin()` function call was implemented. The position transformation is given by the following expression:

$$^{L}|p_R = (^{P}|R_L)^T(^{P}|p_R - ^{P}|p_L)$$

where P represents the pelvis, R represents the right foot, and L represents the left foot. The pelvis is used as an intermediate frame to find the position of the right foot with respect to the left foot.

Similar to the position transformation, the orientation transformation is given by the expression

$$^{L}|R_R = (^{P}|R_L)^{T\,P}|R_R$$

To transform the Jacobian components from the `.fkin()` calls, we created a half Jacobian template of 3 rows of 30 joint values. The calculated values from the function call were appropriately placed into these templates, and used to form the Jacobian component transformations for J_v and J_ω, respectively:

$$^{L}|J_R^v = (^{P}|R_L)^T(^{P}|J_R^v - ^{P}|J_L^v + [^{P}|p_R - ^{P}|p_L]_\times \; ^{P}|J_L^\omega)$$
$$^{L}|J_R^\omega = (^{P}|R_L)^T(^{P}|J_R^\omega - ^{P}|J_L^\omega)$$

To form the full 6×30 Jacobian matrix, $^{L}|J_R^v$ and $^{L}|J_R^\omega$ were stacked vertically.

Once the full Jacobian has been calculated, it can be used to return a q and $\dot{q}$ that successfully controls the kick motion, leading Atlas' right foot to the position of the kick. To experiment with the mechanics of Atlas' foot reaching the `pkick` position, we introduced a joint weighting called $\dot{q}_{max}$ that favored some joints over others, changing the way Atlas kicks. The calculation for $\dot{q}$, given $\dot{q}_{max}$ is as follows

$$\dot{q} = \text{diag}(\dot{q}_{max}) \left[\begin{pmatrix} ^{L}|J_R^v \\ ^{L}|J_R^\omega \end{pmatrix} \text{diag}(\dot{q}_{max}) \right]^{+} \left[\begin{pmatrix} v_d \\ \omega_d \end{pmatrix} + \lambda \begin{pmatrix} \texttt{ep}(p_d, ^{L}|p_R) \\ \texttt{eR}(R_d, ^{L}|R_R) \end{pmatrix} \right]$$

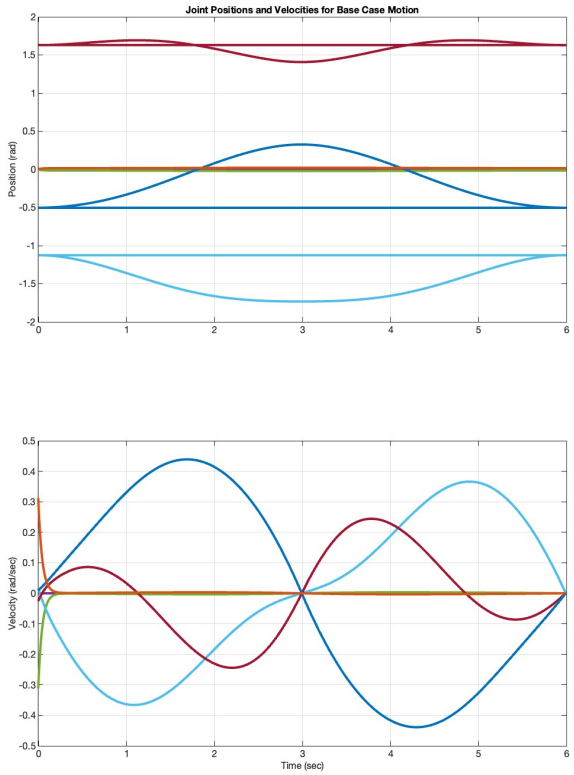
p_d and v_d are the desired position and velocity vectors calculated through a cubic spline. The `goto()` function was called on the previous q values of the robot and uses a cubic spline to calculate the desired position and velocity, depending on where in the kicking motion Atlas is. R_d is set to the matrix and ωd is set to a zero vector.

The values for q are calculated by multiplying the calculated $\dot{q}$ value by the time step dt and adding that to the previous value of q, q_{last}.

$$q = q_{last} + dt \cdot \dot{q}$$

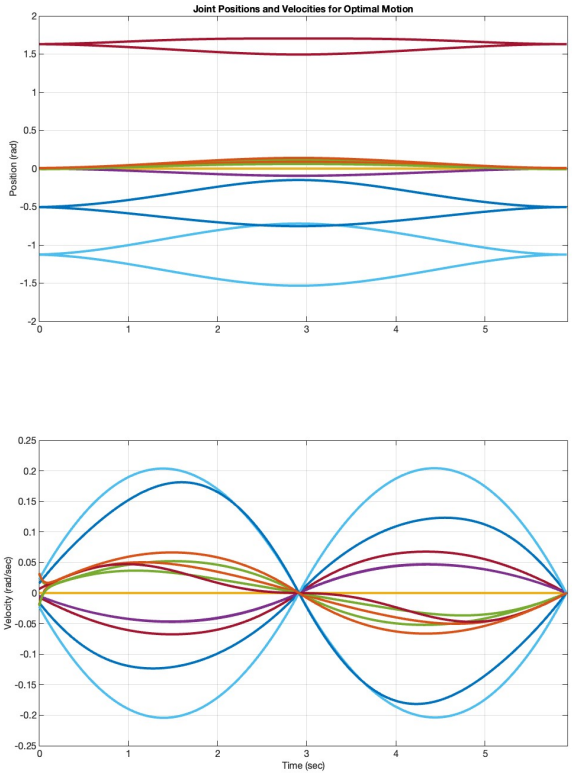
4 Testing Results

For our first test of Atlas, we locked all but it's 6 right joints, referred to as the base case. The motion of the robot was identical to the motion of the 6 DOF robot with the pelvis as the world frame. However, the kinematic chain in this case stretched to the opposite foot. A plot of the joint positions and velocities for this task with these strict constraints on Atlas can be seen below.



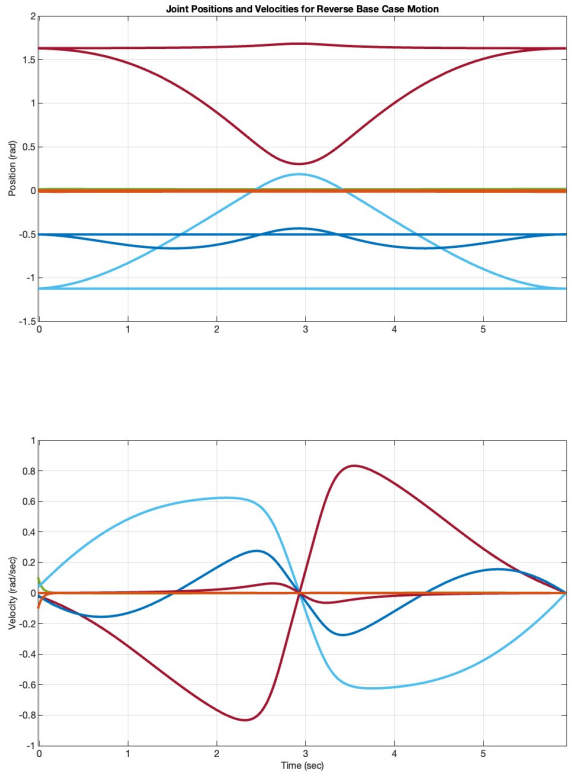
As seen in the plot and video, virtually all of the position changes of the leg are driven by the motion of the left ankle, knee, and hip about the y-axis. In the 6 DOF configuration, the knee is raised vertically along with the bending of the hip and the ankle to allow all three joints to contribute fairly equally for the foot tip of the kinematic chain to reach its target. The motion is a knee drive and the extension of the ankle to meet the ball bouncing towards it.

For our second trial, we set all $\dot{q}_{max}$ values for the 12 joints in our kinematic chain to 1. The goal was to see how Atlas would optimally kick if the entire kinematic chain was free.In this case, a far larger amount of joints contributed to the motion of the robot, recruiting joints from each hip and ankle to allow Atlas to smoothly move all of its joints towards the target point.

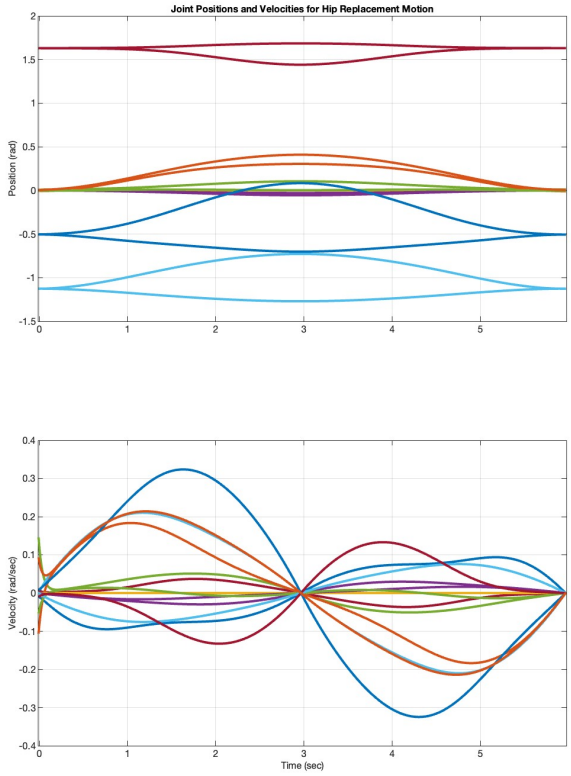


The hip joints from both the left and right legs were the largest contributor to both the change in position and velocity of the robot, followed by the left and right ankles. The position graph was more evenly parabolic and the velocity graph more evenly sinusoidal in this case than in the base case, as joints can steadily move through their range of motion with the freedom allowed by 12 DOFs.

In the third case we tested, the reverse of the base case, we locked all of the joints in the right leg, allowing only 6 DOFs in the left leg. There was a very slight weighting that had to be placed on the right knee in order to avoid singularity. The locking of the other right leg joints created a much more uneven motion in Atlas in trying to reach the kicking target – the robot essentially had to do a one-legged squat with its left leg to reach the target without moving its kicking leg. The result is a less smooth motion and an uneven graph, with intense changes in velocity and position for the joints that can move, especially in the knee in the left leg. The locking of the right joints puts undue strain on the left leg, which could cause faster injury and wearing of joints in Atlas long-term.

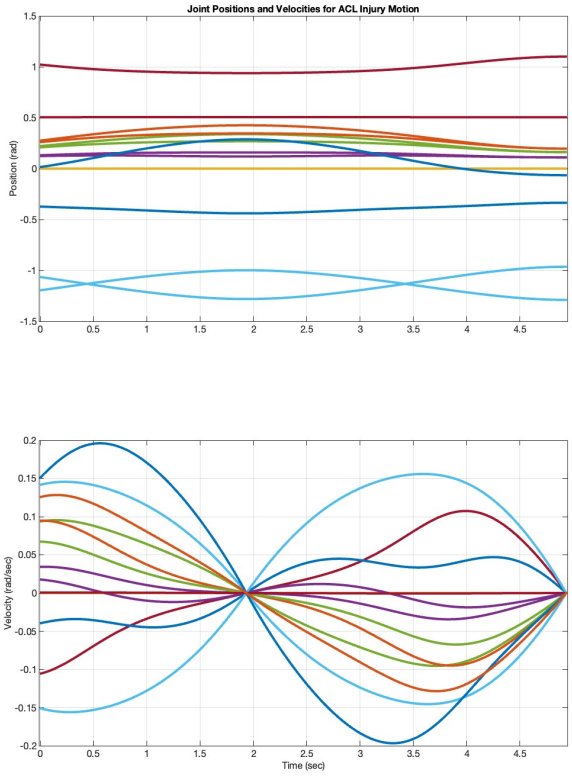


In our fourth test case, Atlas is aging, and their hips aren't what they used to be! Despite their age, Atlas still loves to play soccer. Atlas was lucky enough to have a left hip replacement, so their left hip is slightly more functional than their right hip, but they still favor a kicking motion that uses the least amount of hip motion. To simulate this case, we placed a weighting of 0.3 on the right hip, limiting the motion compared to the left hip, which was weighted at 0.5. The $\dot{q}_{max}$ of both hips was weighted less than the knee and ankle joints in this case, which were all weighted at 1.



The result was an interesting sideways motion that we had not previously seen in Atlas' motion – it moved to its left side in order to favor its right hip, avoiding a sharp driving motion in the more injured joint. The graph again was not as smooth as the optimal case, as uneven weighing of the $\dot{q}_{max}$ forced Atlas to come up with a different solution to complete its task and play soccer again, despite its hip injury.

In our fifth, and final test case, unfortunately, Atlas tore their right ACL going into a tackle. With their doctor's approval, Atlas can continue to kick a soccer ball, but only if their right knee remains straight (q value for the `r_leg_kny` joint is set to 0 in `q0`). Atlas needs to readjust its joint positions slightly when initialized due to the `r_leg_kny` joint being set to 0, resulting in slight movement of the robot before the evaluate() function from the Trajectory class begins movement.



Once the trajectory class begins movement, Atlas once again leans to the left in order for its right leg to reach its target. The most pronounced movement was sideways movement of the left hip and ankle to compensate for the right knee's inability to bend.

5 Conclusion

The Atlas robot can reach its kicking point target through a variety of different configurations, depending upon the weighting employed. When given limited constraints, it incorporates balanced motion across the joints to smoothly reach the target point, while when heavily constrained it takes more dramatic action across the available actuators. This kicking point target can itself vary, adjusting to different ball trajectories within a reasonable window of Atlas' physical reach, as well as velocities that cooperate with the visualization tools. While we certainly cannot anticipate all edge cases in a project of this scope, we can constrain Atlas in a number of ways and it will still find a way to get to any reachable ball.