Bot Dylan: The Guitar-Playing Robot

Camilo Garrido, Trey Scott, Kenadi Waymire

ME/CS/EE 133a

13 December 2024

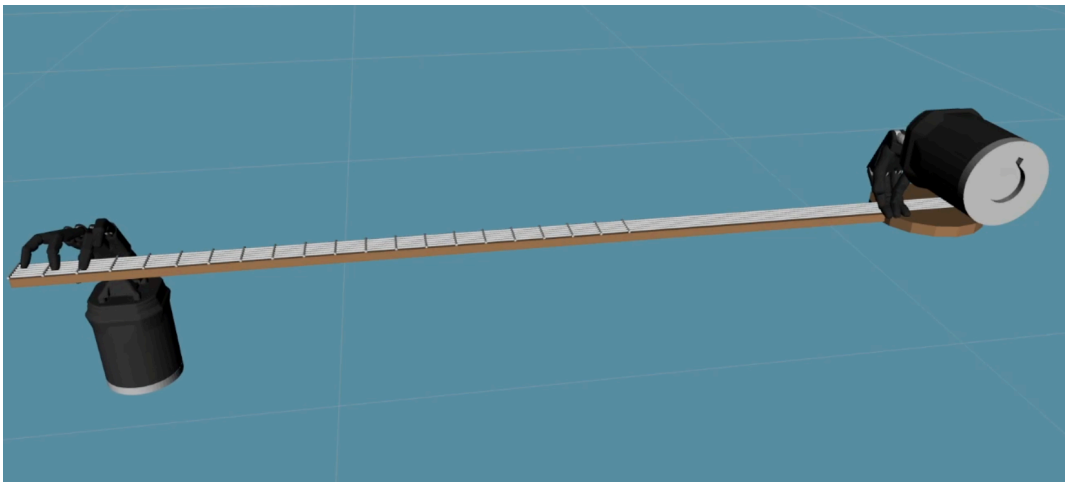

Figure 1. BotDylan in action.

Music has always been intertwined with human creativity, but robotics offers an intriguing avenue to simulate and replicate some aspects of musical performance. For this project, we chose to design and model a robotic system capable of playing the guitar, providing an opportunity to apply and deepen our understanding of the kinematic principles covered in ME 133a. Specifically, we focused on bimanual coordination and the precise, small-scale movements required to emulate human guitar-playing motions

Playing the guitar requires two distinct but synchronized motions; one hand serves as the fretting hand, pressing down the strings along the guitar's fretboard to specify a chord. The other hand acts as the strumming hand, sweeping its fingers across the strings at the lower end of the guitar to generate sound. For this project, we use the Shadow Robot Dextrous Hand (pictured below). This model features coupled hands in one URDF—perfect for our goal of strumming and fretting the guitar simultaneously. We chose to utilize this hand due to the wide range of motion provided by having 24 joints per hand (before any modifications), allowing us to achieve the precise, intricate movements necessary to play the guitar. We chose to play the guitar the way a right-handed person typically plays — using the left hand as the fretting hand and the right hand as the strumming hand — but the choice of hand is entirely arbitrary. As such, we will refer to these as the fretting or strumming hand throughout the report.

Throughout this project, we explored the technical challenges of programming bimanual human-like hand coordination. All of the fingertips share some joints (e.g., at the wrists), the dimensions of our primary and secondary tasks change dynamically depending on the number of fingers involved in playing a chord as will be explained in more detail, and we had to carefully fine-tune our system to achieve natural-looking hand-motions (including incorporating a tertiary task). These challenges required us to develop a deeper understanding of the material covered in ME 133, particularly regarding inverse kinematics for multi-task systems. The work underscored the importance of integrating theory with practice and highlighted the potential for robotics to contribute to unconventional fields like music.

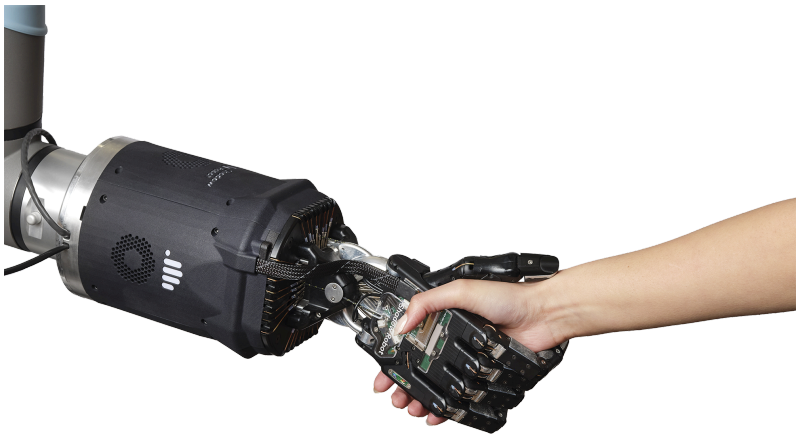

Figure 2. Shadow Robot Hands. https://shadow-robot-company-dexterous-hand.readthedocs-hosted.com/en/2.1.5/index.htm

Robot/System Description

Our robot consists of two humanoid hands — specifically, the Shadow Robot Hands (both left and right) [1]. It has a fixed base, located at the strumming forearm because we assumed that we could make the strumming hand strum the guitar naturally without having to move its forearm; having a moving base would just add unnecessary complexity. That said, we modified the URDF to have the fretting arm slide along a prismatic joint along the x-axis, rather than being fixed relative to the strumming forearm, to facilitate fretting along the guitar neck with the fretting hand. In addition to this prismatic joint, there are 44 revolute joints, giving the robot a total of 45 DOFs.

Right Hand				Left Hand				
Wrist (Side-to-Side)				Right-to-Left Hand Prismatic Joint				
Wrist (Forward and Back)				Wrist (Roll)				
Pointer (Side-to-Side)	Middle (Side-to-Side)	Ring (Side-to-Side)	Pinky Base (In/Out)	Wrist (Side-to-Side)				
Pointer Knuckle (Up/Down)	Middle Knuckle (Up/Down)	Ring Knuckle (Up/Down)	Pinky (Side-to-Side)	Wrist (Forward and Back)				
Pointer Distal (Up/Down)	Middle Distal (Up/Down)	Ring Distal (Up/Down)	Pinky Knuckle (Up/Down)	Pointer (Side-to-Side)	Middle (Side-to-Side)	Ring (Side-to-Side)	Pinky Base (In/Out)	Thumb Base (In/Out)
Pointer Proximal (Up/Down)	Middle Proximal (Up/Down)	Ring Proximal (Up/Down)	Pinky Distal (Up/Down)	Pointer Knuckle (Up/Down)	Middle Knuckle (Up/Down)	Ring Knuckle (Up/Down)	Pinky (Side-to-Side)	Thumb (Up/Down)
			Pinky Proximal (Up/Down)	Pointer Distal (Up/Down)	Middle Distal (Up/Down)	Ring Distal (Up/Down)	Pinky Knuckle (Up/Down)	Thumb Knuckle (Side-to-Side)
				Pointer Proximal (Up/Down	Middle Proximal (Up/Down)	Ring Proximal (Up/Down)	Pinky Distal (Up/Down)	Thumb Distal (Up/Down)
							Pinky Proximal (Up/Down)	Thumb Proximal (Up/Down)

Table 1. Table of Joints for BotDylan.

Task Description

Playing the guitar involves, first and foremost, positioning the tips of the fretting hand's fingers onto the guitar strings between specific frets, while having the strumming hand gently strum the tips of its fingers through the guitar strings at the base of the guitar. To that end, we start by defining a primary task consisting of target y and z positions for each finger of the fretting hand that is involved in playing the chord — corresponding to the y-position of the string that the finger needs to press down on, and the z-height of the guitar strings (relative to the world origin) respectively. We also specify the z-position of the fretting hand thumb as the z-position of the underside of the guitar neck in the primary task the way humans do when we play. The y and z positions of the tips of the strumming hand fingers are also included in the primary task, specifying the path through the strings, and the depth into the strings that we want the strumming hands to achieve — so that the strumming hand will follow the strumming trajectory that we specify.

We moved the x-coordinates of the fretting hand fingertips because although we do want fingers to position themselves within the correct frets for a given chord, the frets are about 3 times wider than the fingers themselves. Relaxing these x-coordinates by letting them be handled in the secondary tasks ensures that the fretting fingers first reach the desired strings (which are much thinner than the frets and thus require more positional accuracy), and then move toward the centers of the frets as much as possible. Additionally, for the fretting hand fingers that are not involved in playing a chord, we want to have some clearance between their fingertips and the guitar strings; otherwise, if this robot were actually playing the guitar, it would mess up the sound of the chord. However, we don't need this clearance to be super exact, so we pass the desired z-coordinates of these fingers into the secondary task. Similarly, we do not need the fretting hand thumb to be at any particular x or y along the underside of the guitar neck, but we felt that it does look more natural for it to be centered between the x coordinates of the fingers involved in playing the current chord and near the middle of the guitar neck (along its width/in the y direction). This is because when people play guitar, they use their thumb to support the guitar neck beneath the fingers pressing down on the chord. If we don't specify this in the secondary task, the thumb can end up somewhat awkwardly off to the side.

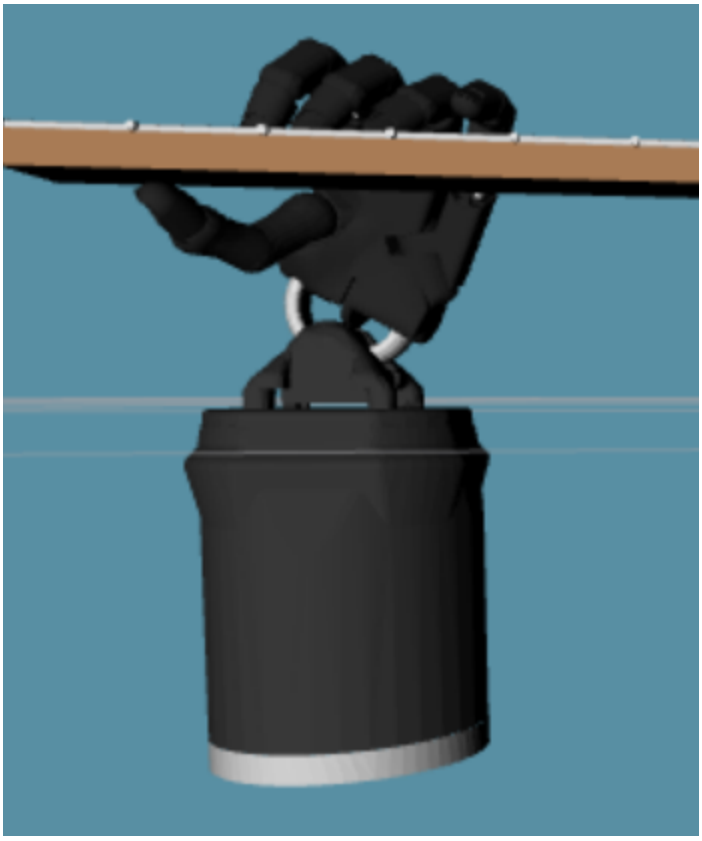

Figure 3. Demonstration of the thumb position off to the side when its x and y positions are not included in the secondary task.

Lastly, we have a tertiary task that uses any remaining DOFs to push the joints toward the “natural” joint positions (shown below) they are initialized at — essentially joint positions that give the hand a relaxed/resting pose — without interfering with the more critical primary and secondary tasks.

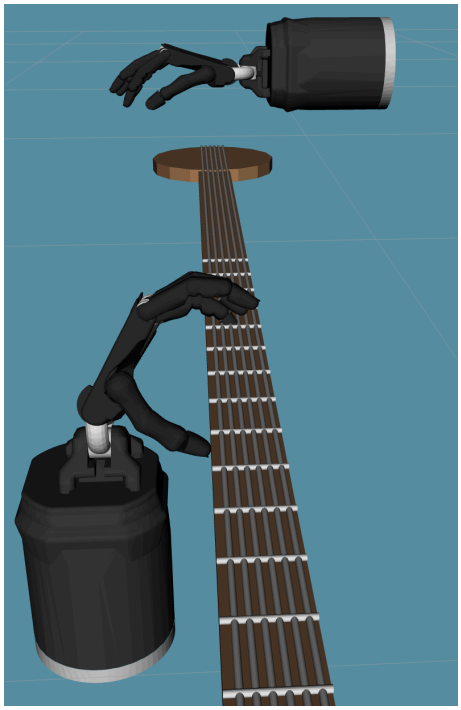

Figure 4. Hands with joints positioned in a ‘relaxed’ starting position.

The overall dimensionality of the task space is thus 27: the x, y, and z positions of the fingertips of the 9 fingers that are not fixed. However, the dimensionality of the primary and secondary tasks can vary, since some chords involve more fingers than others.

The task is usually achievable with the dimensions of the guitar (0.05 x 1.5 x 0.015 m^3) we selected and the chords we have defined (C, E & G), however, with so many tasks and subsystems (fingers) that share joints, minor conflicts, and singularities occasionally arise. We handle these by using a weighted inverse for our primary and secondary tasks. We also have weights for the goal joint positions of the tertiary task — proportional to the inverse of the range of motion of the corresponding joint. The idea here is that a joint with a wide range of motion can be displaced more from middle/natural position while still being within its range of motion, compared to a joint with a small range of motion — so to achieve a natural looking hand pose, we prioritize the goal positions of joints with smaller ranges of motion. Fortunately, the URDF already provided the upper and lower bounds of the joint positions for the actual robot hand which mimic the range of motion of human hands, which saved us the time of estimating these ourselves. For the prismatic joint that we added ourselves, we set the range as the length of the guitar neck since the wrist shouldn’t have to move beyond the neck, and for the fretting-hand’s rolling wrist joint, we estimated the range of motion to be about $\frac{\pi}{2}$ based on our own arms’ range of rolling.

Algorithms/Implementation

We begin our trajectory class by setting up kinematic chains for each of the 9 movable fingertips. It's worth noting that because each hand's fingers share the same wrist (and, in the fretting hand, the prismatic slider) joints, we had to index these wrist joints and join them with the finger-specific joints (from the output of the jointnames method) for each finger. Next, the initial joint positions are defined, alongside other necessary parameters: lambdas for the primary, secondary, and tertiary tasks and a gamma for the weighted inverse. The initial positions of the fingertips are computed using the kinematic chains to establish the robot's starting configuration. The starting positions for each joint are roughly in the middle of their ranges to avoid starting at a singularity.

To facilitate generating trajectories, we developed several utilities. A "song_info" function returns the user-input chord progressions (target positions of the strumming hand), strumming pattern, and total move time T (in practice, this could be determined from chord values and a song's time signature for any given song, but for now we just choose it). A "get_ptips" function computes the x-y-z coordinates of each of the 9 movable fingertips using the "fkin" method of the kinematic chains, and "get_Jv" calculates the linear velocity Jacobian, which relates joint velocities to the fingertip velocities. Jv is first initialized as a 27x40 array of zeros. For each fingertip, we calculate the Jacobian using the fkin method of its kinematic chain. This has 3 rows (x,y,z), but different numbers of columns/joints depending on the specific finger (refer to the joints table for reference). So for each of the 27 possible task coordinates, we index every other 3 rows of Jv as well as the columns corresponding to the joints of that finger, and assign their values based on the Jv of that specific finger. We also created a chord class to quickly and consistently define new chords to facilitate our fretting trajectory, and a fretboard class which is instantiated with the dimensions and x-y-z position of the guitar — and has an inbuilt function for mapping the conventional (fret, string) coordinates of guitar chords to (x, y, z) coordinates in our world frame. In the process of getting these coordinates, the in-built function also identifies which coordinates should be assigned to the primary task, versus which should be assigned to the secondary task depending on which of the fretting hand's fingers are involved in playing the chord — identified by having (np.nap, np.nan) values for their (fret, string) positions.

While the trajectory is running, we track how many chords have already been played by dividing the time, t, by the total move time, and flooring the result. This way we can create a trajectory for the fretting hand between the chord that was played last, and the chord we currently want to play; if the song is just beginning, the fretting hand's initial position is treated as the previous chord. We store the modulus of t mod T, and use this as our time of interest

variable for our movement so that the time of interest resets to zero when we start playing a new chord. Furthermore, to have the guitar playing look somewhat convincing, we have the trajectory move to the chord in the first half of the total move time, T, and then hold the chord position for the remainder of the move time — as a consequence of this, our initial and final velocities and accelerations should be zero when moving between chords. We then pass these time arguments and the previous and next chord positions into the goto() utility function.

For the strumming hand trajectory, we have 3 different patterns that users can choose between: “strum,” “downstroke,” and “upstroke.” In the “strum” pattern, motion is divided into four phases of length T/4, corresponding to movements from the starting position to the midpoint, from the midpoint to the maximum extension, and back to the starting position. At each time step, the phase of the motion is determined by calculating t mod T, as done in the fretting trajectory. Within each of the four phases, the desired position and velocity of the strumming hand are calculated, as well as the depth and length offsets along the fretboard. The strumming hand uses splines for trajectory calculations to ensure that we end up with a non-zero final velocity.

The “downstroke” and “upstroke” patterns are handled similarly. In these patterns, the strumming motion is instead divided into three parts within T. In “downstroke”, the hand moves downward across the fretboard, whereas “upstroke” has the hand moving upwards across the fretboard. Each pattern determines unique start and end configurations while adhering to the constraints defined by the strum_length and strum_depth parameters.

We then concatenate the desired positions for each hand, as well as the desired velocities for each hand. This yields a desired position and velocity (and consequently a positional error) with 27 dimensions, and a 27x45 Jacobian. Since we are using a weighted inverse to deal with singularities, we get the added benefit of not having to worry about the shape of the Jacobian. Next, the program calculates the inverse kinematics, separating the primary and secondary tasks by indexing only the dimensions that should be involved in said tasks. The desired joint velocities of the secondary task are added to the nullspace of the desired joint velocities of the primary task.

We then calculate the joint velocities for the tertiary task of returning the joints toward a natural position. For all of our joints, the goal position is the initial “relaxed” position at which we initiated them (i.e., q0), except for the right-to-left-hand prismatic joint, whose goal position is calculated as the mean of the x-positions of the fretting-hand fingertips — since a human playing guitar would altogether move the forearm of the hand they’re fretting with to be close to the region of the guitar neck where they’re playing a chord for comfort. We multiply the

difference between these goal joint positions and the actual joint positions by the weight matrix we defined based on the inverse of the joint ranges, and a lambda we tuned for the tertiary task specifically. Lastly, we add these tertiary task joint velocities to the nullspace of the secondary task, which is itself already in the null-space of the primary task. Thus, these tasks are executed in the order of importance for achieving the core function of playing the guitar (ideally in a "natural"-looking way).

First, compute the following for all 27 fingertip dimensions (and 45 joint angles for the Jacobian). Note that for fingers not involved in playing a chord, the desired x and y coordinates — which we entirely don't care about — are simply set to their previous values so that the sizes of these arrays do not change, but it doesn't matter because these coordinates are not indexed in neither our primary nor secondary tasks (and have no bearing on the tertiary task).

$$\dot{\mathbf{x}}_d = \mathbf{v_d}$$

$$\dot{\mathbf{J}} = \mathbf{J_v}$$

$$\mathbf{err} = \mathbf{errp} = \mathbf{x_d} - \mathbf{x_r}$$

$$\mathbf{J_{winv}} = \mathbf{J^T}(\mathbf{JJ^T} + \gamma^{\mathbf{2}}\mathbf{I_M})^{-\mathbf{1}}$$

Next, calculate the desired joint velocities for the primary and secondary tasks as follows. For each of the arrays on the right-hand side of the equation, deliberately index only the dimensions that are relevant to that task — as outlined in our task description. Also, choose the appropriate lamba (which gets smaller for each subsequent task to prioritize the convergence of more important tasks).

$$\dot{\mathbf{q}} = \mathbf{J_{winv,task}}(\dot{\mathbf{x}}_{\mathbf{d,task}} + \lambda_{\mathbf{task}}\mathbf{err_{task}})$$

Make sure to add the secondary task joint velocities to the null space of the primary task joint velocities.

$$\dot{\mathbf{q}}+ = (\mathbf{I_{N,P}} - \mathbf{J_{winv,P}}\mathbf{J_p})\,\dot{\mathbf{q}}_{\text{secondary}}$$

Next, with W being the weight matrix based on the inverse of joint ranges and $\mathbf{q_{goal}}$ being the "natural" joint positions with which we initialized the robot, we calculate the tertiary task joint velocities:

$$\dot{\mathbf{q}}_{\text{tertiary}} = \lambda_{tertiary} * \mathbf{W}(\mathbf{q_{goal}} - \mathbf{q_d})$$

and add these to the null-space of the secondary task:

$$\dot{\mathbf{q}}+ = (\mathbf{I_{N,S}} - \mathbf{J_{winv,S}}\mathbf{J_s})\,\dot{\mathbf{q}}_{\text{tertiary}}$$

Figure 4. Brief inverse kinematics math overview (key points).

The final resultant joint velocities are then multiplied over time-step dt (10 milliseconds) and added to the previous joint positions — to numerically integrate the joint positions of the

robot. The desired positions are stored for calculating the next positional error. The main function creates an instance of the trajectory class and spins a generator node to continuously evaluate the desired robot motion.

Particular Features

The bimanual robotic hands are equipped with the ability to handle singularities. Singularities occur when the robot's joints reach configurations that cause a loss of control in certain directions of movement, which can severely impact task performance. To address this, the hands utilize a weighted inverse kinematics approach when the fingers are fully stretched out. This method assigns weights to the joints, prioritizing movements in configurations that avoid or minimize singularities while maintaining precise control.

The hands can also be programmed to play a variety of songs by feeding them predefined patterns of built-in chord positions and strum patterns. Each chord is defined within a class structure and is assigned a name (such as E, C, or G) and the positions each finger will hit on the strings and frets of the guitar. Each chord also returns exactly which finger will play each note depending on the current position of the hand.

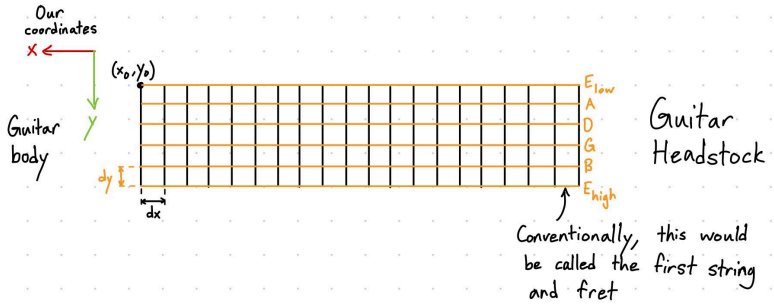

Figure 5. Fretboard coordinate system.

Analysis, Plots/Images, or other Helpful Material

The bimanual robotic hands already had detailed CAD models integrated into their URDF files. That being said, we did add a revolute joint at the base of the fretting hand's wrist to be able to roll the fretting hand in order to reach chord positions more easily.

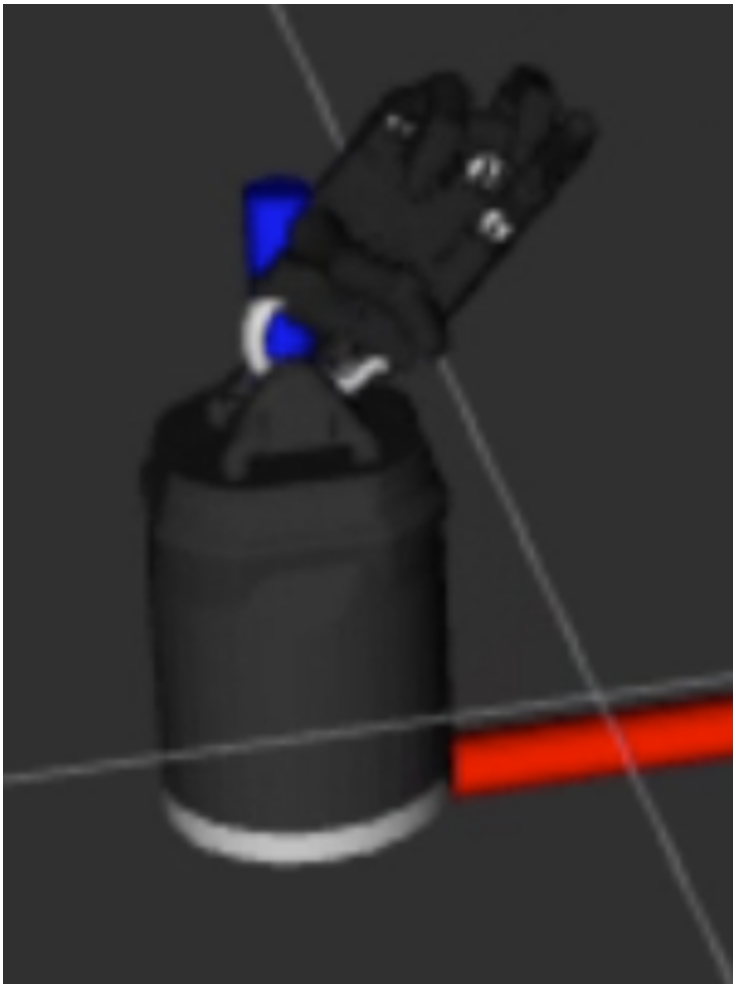

Figure 6. Photo of joint movement before adding the additional revolute joint for rolling the wrist. Please forgive the low quality (we only had a screenshot of an old video before adding in this joint). You can hopefully make out that the middle and ring fingers intersect each other due to the constrained motion. After adding the revolute joint, the overlapping drastically reduced.

We also had to tune the gamma for our weighted inverse so that it wouldn't needlessly "wash out" the joint velocities — since we need very precise precision accuracy for the fretting hand. We found that a gamma of 0.00005 best accomplished this, while still smoothing out movements near singularities (which again, are rare with our tasks and robot setup). The lambdas for the primary, secondary, and tertiary tasks were also tuned, with the magnitudes decreasing for the more pressing tasks so that they would converge more quickly.

For the guitar model, we created a simple URDF using the built-in shape primitives in ROS rather than dealing with a more complex, premade guitar CAD. Strings and frets are modeled as cylinders, with the neck being a box. Using a simple URDF had the added benefit of

being able to directly grab the position and rotation data for each fret and string within ROS to find the desired fret position to play chords.

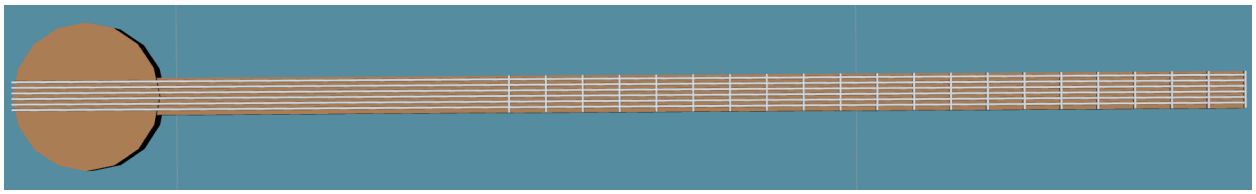

Figure 7. URDF model for guitar.

We approximated the fretboard as a grid of equally sized rectangles to facilitate generating chords and our fretting hand trajectories, so that we could spend more time focusing on the crux of this project: the kinematics, inverse kinematics, and task specification.

All in all, this project has been an incredible learning experience. We have accomplished our key objectives of precisely manipulating a complex bimanual robot, and in doing so, profoundly deepened our understanding of the material covered throughout this course. There is always room for further fine-tuning the system, to reduce finger collisions, but we are remarkably proud to have gotten the system working to this level in just the past few weeks.