Team Volleybot Final Report

Bryce Huerta, Courtney Moran, Rohan Punnoose, Sergio Licon
CS 225A - Spring 2021

Abstract

The goal of this project was to design a set of volleyball playing robots using SAI2 and simulate a full game. First, a world was constructed as a urdf. An official outdoor volleyball court was modeled with lines for designated front zones, back zones and out-of-bounds area. A net with two posts was modelled and wrapped with a collision mesh, and a camera was set up along the side of the court as well as at the back. The ball was added as an object with only translational degrees of freedom and a collision mesh as well. Two robots were added to either side of the court; the two robots featured legs from the Toro robot and the arm from the Panda. A flat plate was used as the end effector to make contact with the ball. The state machine featured three states: IDLE, BALL_TRACKING, and HIT; these states corresponded to the three actions our robots would take depending on the trajectory of the ball. Our controller scheme featured separate controls for the base - a joint space controller with no dynamic decoupling - the end effector - operational space control with joint space regularization in task null space - and the legs and feet - Jacobian-based force and moment control. State information about the velocity of the ball at impact was used to project the ball's landing spot, and this information was fed to the controller for each active robot. Ultimately, two robots, one per side, were able to play a continuous game of volleyball where both robots rallied for several shots.

Final Implementation

This section is broken down into further subsections labeled “World”, “State Machine”, “Controllers”, and “Planning” where the final implementation for each is described in full detail.

World

The first step in getting our volleybots simulated was to create the world they would live in. Luckily for our group, the teaching team was gracious enough to provide us with the world and supporting urdf files used by the tennisbot team which served as an example to model for our simulation world.

We began by modeling all of our static objects in the simulation. This included creating a regulation 9m×18m volleyball court with corresponding lines to delineate the front zone, back zone, and out-of-bounds region. Additionally, we modeled a surrounding region to serve as “grass” that would be outside our “sand” court. All of these were modeled as thin surfaces in SolidWorks and then exported to Blender to add textures, colors and position all of the regions properly with respect to each other. We attempted to then model a net of our own for the simulation, but the file size was absurdly large which slowed down the simulation. Since there were more important tasks at hand we decided to recycle the net from the tennis team and add some posts to raise it to an appropriate height for volleyball. These posts were also modeled and colored in Blender before being added to the world urdf. Of these static objects only the netted portion and the ground were covered by a collision mesh. Both of these collision meshes were modeled using appropriately sized box primitives.

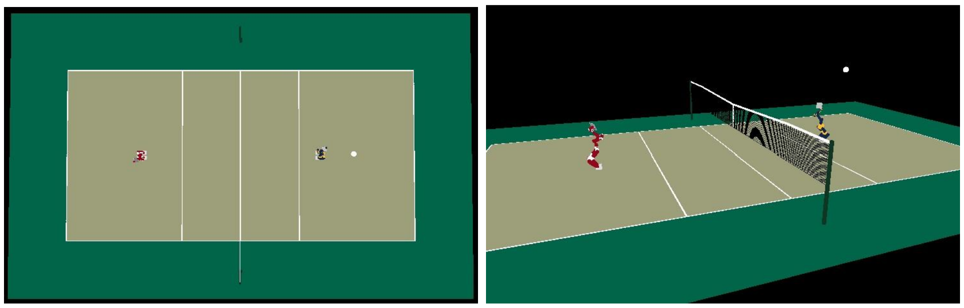
Figure 1: Overhead view of court (left) and side view of net (right).

Next we were tasked with adding our robots to the world. We had originally planned to use the provided mobile base Panda arm robot, but after assessing the height of the net and the Panda arm, we decided to combine the Panda arm with Toro legs to give an additional joint for height. Additionally, we decided to add a paddle to the end effector of the Panda arm to serve as a paddle for hitting the ball. This paddle end effector is the only part of the robot with a collision mesh, and similar to the court and net it is created using an appropriately sized box mesh. With our robot configured, Volleybot was born! We entered Volleybot into the simulation and our next task was to add the volleyball itself! The volleyball needed to be added as a robot because it would be moving around during the course of the simulated game. Additionally, the volleyball was wrapped in a primitive spherical collision mesh to bounce off objects as needed. As a robot, the volleyball has six degrees of freedom, but in our simulation we are only using translational degrees of freedom and this ignoring the effects of spin. Having the volleyball simulated as a robot also helped later in tracking the state variables of the volleyball, but this is discussed further in a later section. The model for the volleyball was actually pulled from GrabCad after an unsuccessful attempt in Solidworks and it was colored in Blender. The team started off with just one Volleybot and the volleyball in the simulation for simplicity in tuning the Volleybot to hit the ball over the net. Once the controller was tuned with one Volleybot working, we added a second one to the opposite side to work on implementing the state machine and get a 1 vs. 1 game of volleyball simulated. Per Professor Khatib's advice, we added team colors to our Volleybots in the form of cardinal and white to represent Stanford, and blue and gold to represent UC Berkeley. In the end, our team was only able to simulate a 1 vs. 1 volleyball match, but the setup of the world urdf and simviz files would allow for an easy insertion of more robots to achieve the 2 vs. 2 match that we had originally aimed for, but is now categorized as future work. This change would simply require the addition of robots to the world urdf, additions to the robot and redis vectors in simviz, and correction of robot locations in both the world and simviz.

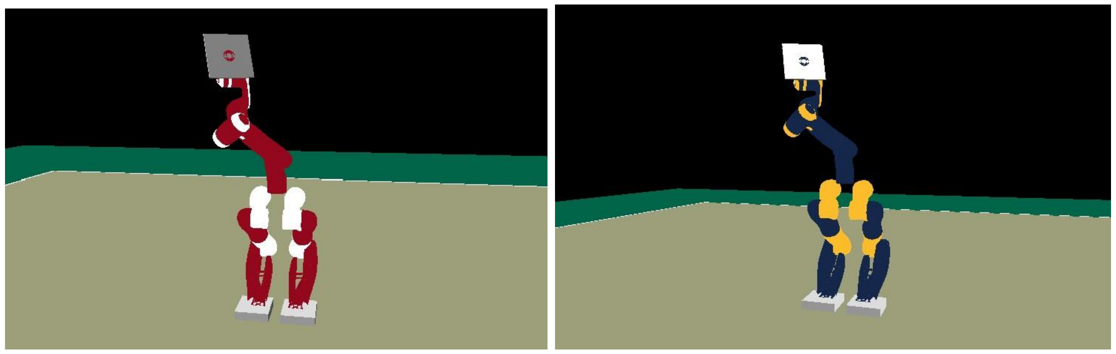

Figure 2: Stanford Volleybot (left) and Cal Volleybot (right)

Other useful details of our world and simviz include that we created a key in the simviz to toss the ball whenever the user presses the "t" key. This tosses the ball in a general direction but with some randomness as to the exact initial velocities which proved useful in testing and tuning the trials with one Volleybot. Later in the project we added a camera to the backcourt to serve as an addition to the camera already placed in the centercourt. The user can switch to these cameras by holding the "c" key and pressing "1" or "2" for the centercourt or backcourt camera, respectively. These were very useful in recording our video for the demo. On the physics side of things we used a coefficient of restitution of 1.0 and also used 0.0 as the coefficients of static and dynamic friction. None of these values are realistic to an actual game of volleyball and integrating more realistic values falls into the category of future work for this project. Other aspects that fall into future work are the aforementioned addition of robots to simulate 2 vs. 2 volleyball, the resistance of a sand court, and more ball dynamics such as the incorporation of the effects of spin and air resistance.

State Machine

The state machine has three states, IDLE, BALL_TRACKING, and HIT. In the IDLE state, the robot will hold the end effector straight up and keep its previous position. If the ball is moving toward the robot, it will transition to the BALL_TRACKING state, and it will begin predicting where the ball will land and moving to that location. When the ball is within 0.1m of the end effector, the robot will transition to the HIT state. In the HIT state, the robot will execute a small swing action to hit the ball up and over the net. After the robot has hit the ball, and it is moving away from it, it will transition back to the IDLE state.

The state transition from IDLE to BALL_TRACKING is based on the predicted ball position at a certain z-intersection being within a specified radius horizontally from the robot. The state transition from BALL_TRACKING to HIT is based on the predicted time to contact being below a certain threshold, an output of the predictor. The state transition from HIT to IDLE is based on the sensed ball velocity being in the direction of the end effector (the ball has bounced).

Controllers

We began our controller design with a simple goal in mind: move the end effector to desired position and velocity in order to hit the ball. To accomplish this goal, we needed to control each of the following three components of each robot:

1. Base
2. Legs and feet
3. End effector

To control the base control, we used a joint space controller with no dynamic decoupling. This positions the “hip” of the robot translationally. The joint space controller is a simple PD controller, and additionally contains a velocity saturation component. A nullspace matrix is used in order to ensure torques are only generated in the base translational degrees of freedom, and not the arms or the legs. Since the full robot model is used in the controller, the joint space controller implementation we used (from SAI2) did dynamic decoupling before projecting onto the non-nullspace degrees of freedom, which causes issues since the dynamic decoupling includes torques in the arms and legs that will be zeroed out. Therefore, dynamic decoupling was not used, and instead the gains were manually tuned based on the actual mass matrix.

To control the end effector, we used an operational space controller with joint space posture regularization in the task null space. The end effector position is commanded to be a certain position relative to the robot hip, which is typically fixed but can be changed by the outer loop planner. The end effector desired orientation is set by the planner, based on the ball prediction and desired hit configuration.

To control the legs and feet, we used Jacobian-based force and moment control. This controller is used to drive the Toro legs into the ground and keep the feet horizontal. Essentially, a desired leg downforce is computed based on a PD control loop around hip height, and is mapped via the Jacobian transpose to leg joint torques. The feet represent another 2 degrees of freedom, and simple orientation PD controllers are used to ensure they remain horizontal and in contact with the ground. Similar to the base control, a nullspace matrix is used to ensure no torques are generated for the arm or the base degrees of freedom, and no dynamic decoupling is used.

Just before contacting the ball, the legs and base work together to impart a slight “swing” force into the ball, in order to impart energy into the ball’s trajectory and prevent the ball from losing all of its energy as it bounces from robot to robot. This is implemented in the HIT state, which sets the desired end effector position such that position error terms cancels and the closed loop (joint space) system becomes a first order velocity tracking loop, tracking the desired swing hit velocity. Note that the arm controller’s end effector desired position is set to be fixed in the hip frame, so the swing torque comes from the base and legs joint space controllers.

As the base controller moves the base around rapidly to track and swing into the ball, this base acceleration causes a disturbance in the leg and arm controllers. To mitigate this, this acceleration effect is predicted and cancelled out via a feedforward term in both the arm and leg controllers.

Planning

In order to implement the control, our system needed to predict where the ball was going to land. To make these predictions, we assumed that the ball would obey simple projectile motion, neglecting forces caused by air resistance and the Magnus effect due to spin. Using state information of the ball, we were able to extract the ball’s position and velocity in the world frame.

$$pos_{landing} =- 1/2gt^{2} + v_{ball}t + p_{ball}$$

s

From this equation, we could find the ball’s final position in the Z-axis depending on our target height. We could then take the time to reach this height and calculate the ball’s coordinates in X and Y in the world frame. The robots’ base controllers would use this data to move the robot into a proper hitting position.

$$t_{f} = \frac{-v_{ball,z} - \sqrt{v^{2}_{bal,zl} + 2g(p_{ball,z} - hit\ height)}}{-g}$$

To compute the desired end effector orientation, we needed to compute the normal vector to the end effector surface, which would result in the ball landing at a desired location. Assuming perfect elasticity, this normal vector was simply the average of the incident and desired reflected velocities. The desired reflected velocity was initially computed using the projectile motion equations above, and setting the landing position to be center court. By solving for t_{f}using only the z coordinate, we assumed that the z component of the velocity was perfectly reflected. This turned out to be not very accurate, so instead we simply set the desired reflected velocity to be the incident velocity negated.

Challenges

Our first significant challenge began when we tried to get our robots to hit the ball to the other side. We had a fairly easy time getting the collision meshes to work, so contact itself was not an issue. However, our end-effector initially would only be able to hit the ball to the other robot some of the time, often missing the target well out of bounds. This issue was attributed to the aiming scheme not being very accurate. This was because we made many assumptions to simplify the desired end effector orientation computation. To remedy this, we switched the aiming scheme to instead just reflect the incident velocity of the ball. Since the ball was coming from the other side of the net, reflecting its velocity would send it back (approximately) to where it originated on the other side. However, once we started getting consistent hits back and forth, the ball would lose energy and go straight into the net after a few hits. This miss came from our ball coefficient of restitution being set below 1, so we had to adjust our end effector control to add energy to the ball on each hit instead of simply reflecting the incoming ball.

The next challenge we faced was simulating two robots. Having both robots in our world significantly slowed down the simulation, causing our game to be played significantly slower than in real time. The slowness of the simulation made development hard at times as the actuation and motion of the robots and ball that we could see was reduced by several orders of magnitude.

Finally, developing the controller and state machine was the most difficult aspect of the project. After some iteration, we arrived at our state machine with the three states: IDLE, BALL_TRACKING, and HIT. However, we found it difficult to achieve robust state transitions. At times, these seemingly simple changes could result in unintended stuck loops that were difficult to debug. On the controller side, we ran into the problem of dynamic decoupling when segmenting joints into different tasks, which yielded out of control robot behavior during testing. This issue may have occurred in part due to the fact that we were incorporating parts of both the Toro (legs) and Panda (arm), and the implementation-specific order in which dynamic decoupling and nullspace projection are done.

Conclusion

Through this project, we accomplished our goal of simulating a full game of volleyball between two robots using SAI2. From this project, we learned how to build aspects of an environment in SAI2. Previously, we had only worked with the robot and end effector in empty space. Now, we were able to model a full volleyball court with lines, a net, and multiple camera angles. We also learned how to build extensions of pre-defined robots; as we mentioned, we combined the legs from the Toro to the Panda arm with a flat plate end effector. We even learned how other objects, like our ball, could be modelled as “robots” (our ball was a 6-DOF robot in its definition). We also learned how to implement a state machine for multiple robots and switch between actions to make our game work.

Overall, this project was incredibly enjoyable for the members of our team. We started with a goal to simulate a game that we each have played on the outdoor courts at Stanford. We finished this project with two robots that can consistently hit the volleyball across the net, something that we cannot even do in real life. We overcame the difficulties of modelling the world and building the controller, and we produced some memorable highlights of our robots locked in competition.

Future Work

In future implementations, it would be interesting to add two more robots and have them play as teams. This would involve a slightly more complicated state machine that would handle coordinating hits between teammates. The system would need to assign one robot to make the first hit, then hit on the same side of the court, with up to three hits per side.

Another improvement that could be made would be to the aiming scheme. Our original planning computations for the aim location and desired end effector orientation were highly simplified, essentially reflecting the ball on its inbound path. With a switch to team play, a more accurate aiming scheme would be necessary. More realistic dynamics to the simulation such as modifying the coefficient of restitution, coefficients of static and dynamic friction, adding air resistance to the ball, adding spin to the ball, and even the resistance of trying to move in sand.