The Spikeball Woes or the Spikeball Woohoos? Only Time Will Tell...

Sam Beaulieu, Kristina Beck, Aubrey Kingston, Eric So
CS225A: Experimental Robotics
Spring 2020
Stanford University

***Abstract*— This paper is a review of using SAI2 simulation software to create four spikeball- playing robot arms. We use ball tracking and prediction, position and orientation control, and game play logic to simulate a realistic game of spikeball between four stationary, seven-degree-of-freedom robot arms. The following paper will discuss the final implementation of this simulation along with its challenges, such as controlling all four separate robot arms with one controller, reducing computational complexity, and creating an effective state machine. Overall, this project was challenging yet rewarding once we were able to get all the various parts of our simulation working together.**

I. Final Implementation

A. Background

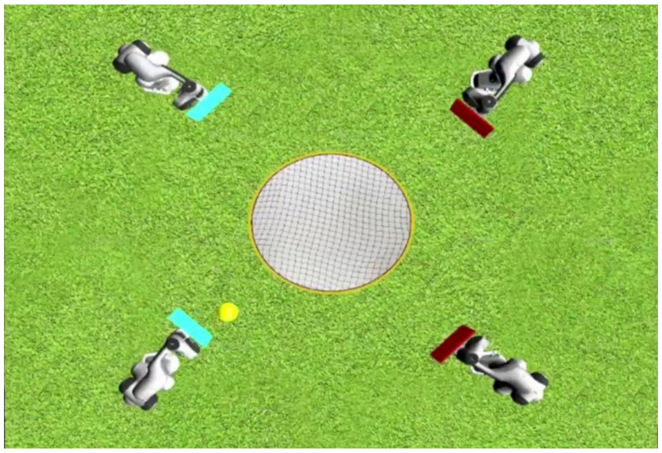

Fig. 1. Overhead view of Spikeball game play set up with ball initialized in the bottom left then sent diagonally across to Robot 0 to start the game.

The final implementation of this project incorporated four "Panda" robotic arms, each with a stationary base, seven rotational joints, and seven degrees of freedom. The four arms were located equidistant from a net. Each arm was located in one of the four quadrants around the net, positioned 1.0 meters away in both the x- and y- directions. For the rest of this paper, the robot arm in the positive x, positive y quadrant will be referred to as robot 0, with the rest labeled as robot 1, robot 2, and robot 3 traveling counter-clockwise from robot 0. The goal of this project was to have the four robotic arms play a game of Spikeball. Robot 0 and robot 3 are one team while robots 1 and 2 are another team, shown in Figure 1 in red and blue, respectively.

The following list summarizes the key components of our final implementation.

- Setting up the simulation interface
- Establishing the redis server client and position/ orientation control
- Game play logic

B. Simulation and Environment

Other than the four robot arms pulled in to act as players for the game, the environment included a standard sized spikeball net at the center with a diameter of approximately 1 meter. The net object was made using BLENDER software that led to it having a complex mesh. Thus, in this simulation, the net is shown visually while the collision mesh is a simple box. The ground and walls are also box meshes with the grass and sky just there for visuals. Finally, the ball was originally a sphere made in BLENDER but then changed later on to a sphere mesh in the .urdf file. This was done to simplify the object from the high-resolution mesh made from BLENDER. The ball is 0.12 meters in diameter with a low mass to limit the momentum transfer to the robots.

During game play, there only needs to be one robot in action as it responds to the ball. In order to do this, we put commands in place such that the robots will remain in their current position after hitting the ball. This reduction had to be done in both the controller code as well as simulation in order to achieve the desired cutback in computation and processing costs.

C. Game Play Logic

The overall rules of Spikeball are as follows.

- The four players work in teams of two
- A player hitting the ball may either return it to the other team or pass to its partner
- If the player chooses to return the ball to the other team, it must bounce off the central net
- If the player decides to pass, the team can have up to three total touches (or two passes) before the ball must be returned to the other team

In order to maintain these rules, we used several booleans to act as flags in order to track if we want a robot to pass or spike, if the ball is coming from a pass or spike, if we want to initiate ball tracking, and if we want to predict final ball position. We also used a function to use the general direction of the movement of the ball to determine which robot to activate next. When a robot receives a ball off the net, we call a function to randomize whether a robot immediately spikes the ball back to the opposing team or passes the ball to its teammate. If the robot passes to its teammate, we then use another randomizing function to determine the number of passes the team will perform (one or two) before returning the ball. These functions use the default random engine class. Meanwhile, if we determined that a robot should be moved into position to receive a ball, we used two functions to determine the desired end-effector position and orientation.

D. Desired Robot

To determine the next robot to control, we took into account the ball's velocity as well as whether or not the ball was being passed or had just been spiked. Depending on these flags, we would use the sign of the ball velocity to know know quadrant the ball would travel to, and thus, the desired robot. Otherwise, we would switch the robot based on the team. For example, if Robot 0 had just hit the ball and it was known to be passing, the desired robot would be Robot 3, as both are on red team.

E. *Desired Position and Orientation*

In order to find what position and orientation a robot needs to be at in order to either pass or spike the ball, we wrote two functions that take in the ball's initial position and velocity as well as the ball's target position after it is hit. We first wrote these in MATLAB in order to visualize its trajectories, as seen in Figure 2. As discussed in the Challenges section, we had to ignore gravity

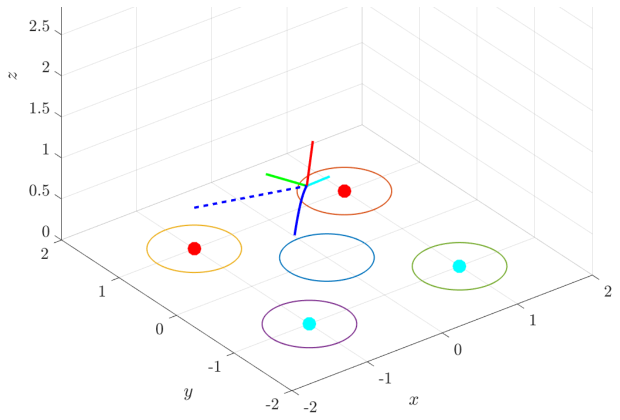

Fig. 2. Matlab visualization of ball's trajectory in order to confirm the accuracy of the desired position and orientation functions. The middle circle represents the net and the other four circles represent the reachable radius around each robot.

for our simulation. Thus, we changed these functions to assume straight line trajectory of the ball, decreasing the computation power necessary to solve the system of equations.

To calculate the desired end effector of the robot, we first projected the trajectory onto the X-Y plane. Since no forces are acting on the ball in this plane the ball will track a straight-line trajectory. Since each robot has a limited workspace, we cannot arbitrarily send the end effector to meet the ball. We thus draw a reachable radius around the robot and find the intersection of this circle with the incoming linear trajectory in XY. The intersection points in x is given by:

$$x=\frac{a+bM-BM\pm\sqrt{-a^2M^2+2abM-2aBM-b^2+2bB-B^2+M^2r^2+r^2}}{M^2+1}$$

Where a is the global x-coordinate of the center of the robot, b is the global y-coordinate of the robot, M is the X-Y slope of the incoming ball, B is the Y-intercept of the incoming ball, and finally r is the reachable radius of the robot. This quadratic expression has two solutions, we select the real solution that minimizes the L2 norm between the current ball position and the predicted intersection with the reachable radius. We can then select y via $y = Mx + B$.

Given a real solution exists (the ball may not ever pass through the workspace of the robot, thus a solution does exist) we can then find the z intercept. To find the z position of the end effector we derived the equations of motion.

$$z = -\frac{1}{2}gt^2 + \dot{z}_0 t + z_0; \qquad t = \frac{x - x_0}{\dot{x}_0}$$

To find the orientation of the end effect, we require the inbound velocity vector at the end effector and the required outbound velocity vector to get the ball to the target. We first perform a rotation about the world frame z axis to align x' with the average of the incoming and outgoing xy trajectory. Then we rotate about the y' axis to align the z' orientation of the end effector. The first rotation about the z axis is given by:

$$R_\theta = \begin{bmatrix} c\theta & -s\theta & 0 \\ s\theta & c\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\theta = \frac{1}{2}(\theta_{in} + \theta_{out})$$

Where $\theta_{in} = \text{atan2}(\dot{y}, \dot{x})$ and $\theta_{out} = \text{atan2}(y_{ee} - y_{target}, x_{ee} - x_{target})$. The rotation about the y' axis is slightly more involved. We first calculate the inbound rotation using:

$$\phi_{in} = \text{atan2}(V'_y, V'_x)$$

$$V' = \begin{bmatrix} c\theta_{in} & s\theta_{in} & 0 \\ -s\theta_{in} & c\theta_{in} & 0 \\ 0 & 0 & 1 \end{bmatrix} V_{ee}$$

Where V_{ee} is the predicted velocity at the end effector. We then find the angle out, ϕ_{out}. This is given by $\phi_{out} = \text{atan2}(h, d)$ where h is the difference in absolute z height between the target position of the ball and the end effector position and d is the euclidean XY distance to the target. As an aside, the above expression for ϕ_{out} is a dramatic simplification from when we include the presence of gravity. The true expression required to find ϕ_{out} is given by:

$$d = \frac{v_0\cos(\phi_{out})v_0\sin(\phi_{out}) + \sqrt{(v_0\sin(\phi_{out}))^2 + 2gh}}{g}$$

We solve this iteratively by incriminating up guesses of ϕ_{out} until we are within some threshold of the desired distance to the target. This is computationally expensive compared to the simplified expression without gravity that we presented previously. We then use the average of these two ϕ angles. $\phi = \frac{1}{2}(\phi_{in} - \phi_{out})$. It should be noted that the sign here is intentional as the ϕ's have different signs.

$$R_\phi = \begin{bmatrix} c\phi & 0 & -s\phi \\ 0 & 1 & 0 \\ s\phi & 0 & c\phi \end{bmatrix}$$

Finally, we multiply the two rotation matrices together to find the required orientation of the end effector. It is important to note that due to the initialization of the robot in our world, we have to apply an additional rotation matrix which we omit here for brevity. The final expression for the rotation matrix for the robot end effector is given by $R_{desired} = R_\theta R_\phi$.

II. CHALLENGES

One of the main challenges of this project was controlling all four robot arms with one controller. We needed to make sure that we were able to properly predict which robot was needed next so that we could send control torques to only one robot at a time. This was done as mentioned previously and was planned from the start. However, in our original implementation, computation was still being done for the other robots and greatly reduced the responsiveness of the system. This proved to be a great challenge in the early stage of our project that was resolved with the guidance of the teaching team. With their help, our simulation and controller code only made movements and computation for one robot at a time.

Another challenge occurred with our available computing power. Our functions for getting the correct position and orientation were in particular very computationally expensive, with multiple square roots. Therefore, we found it extremely important to keep track of when we need to predict final position so that we only call that function once, getting rid of unnecessary computations.

We also needed to slow down the overall speed of the ball and increase the speed of the robot arms so that the program had time to compute the ideal end-effector position and orientation and then move the appropriate robot arm into this configuration. The first step in increasing the robot speed turned out to be turning off the OTG flag, as adjusting the gains for more speed quickly made an unstable system. This change allowed the end-effector to move in a more direct path to its target. However, this was not enough, and so we reduced the ball's velocity. For this to occur, gravity was removed from the system and not accounted for in controls or prediction. This meant the ball would move in straight trajectories and so the magnitude of velocity could be reduced while ensuring it moved in the correct direction for any robot to respond to it.

Finally, the overall logic and randomness of game play was complicated. As stated earlier, we needed several different boolean variables to determine the various states of the game and to make sure we didn't call computationally expensive functions more than necessary. This took some time initially to ensure it ran properly as we started. Towards the end, once passing could be included, we would have to walkthrough the state machine thoroughly for debugging issues that came up.

III. Results and Next Steps

Overall, we found this project to be challenging yet worthwhile. A video of our simulation can be found at the link below. Due to the complexity of the system for our computers, the simulation was intentionally coded to include a slowdown factor. This along with the decreased ball velocity, led to an approximate 15x- speed up on the video.

https://tinyurl.com/SpikeballRobot

In the end, our robot controller is capable of playing a full game of Spikeball with spiking and passing. Potential improvements for the projects can range from game logic and decision making to computation and complexity. By improving the efficiency of the program, it would be possible to incorporate gravity back into the system. Once gravity is included, the existing position and orientation prediction functions can still work. With the robots correctly playing Spikeball with gravity implemented, it would then be possible to incorporate more decision making and competitive play for the robots.

IV. Acknowledgements

We want to thank Professor Oussama Khatib and our two TAs William Chong and Adrian Piedra for their immense and consistent help and support throughout this quarter.