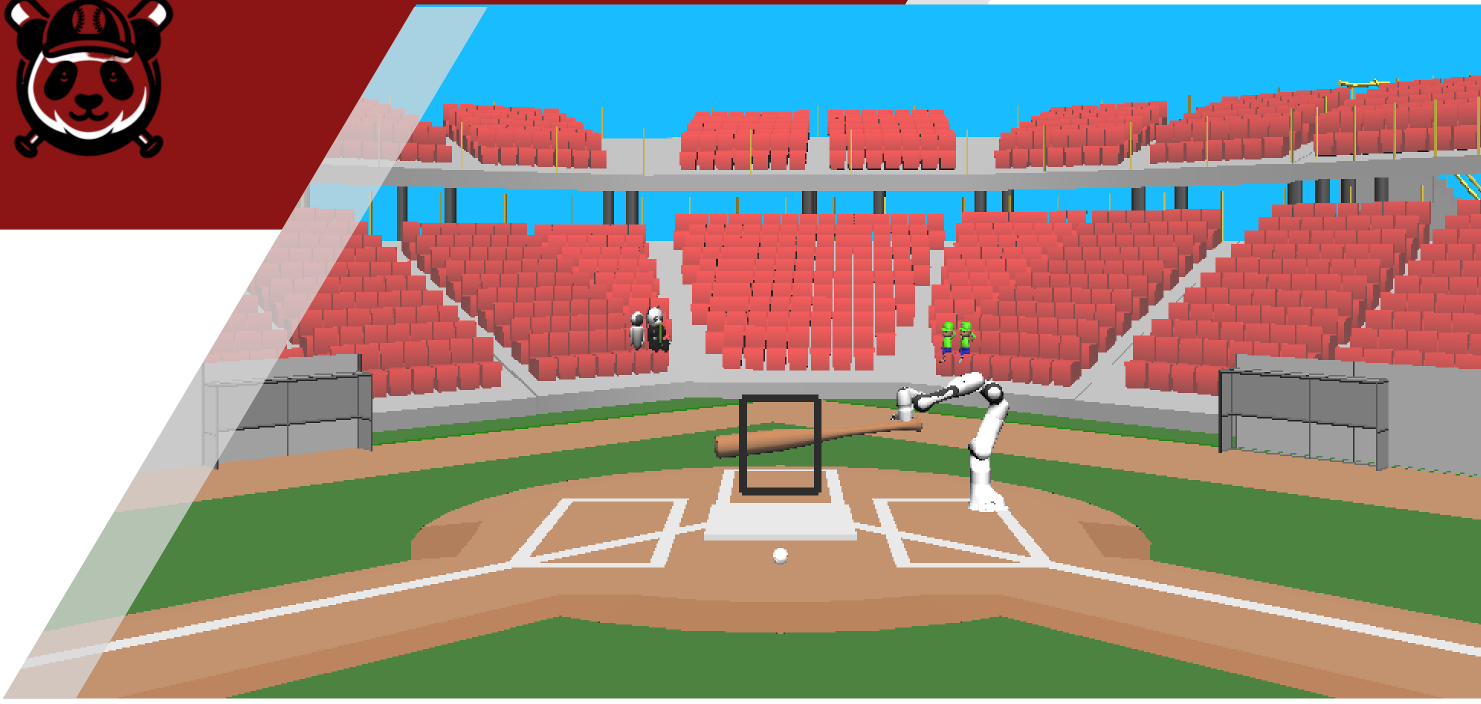
CS225A: Experimental Robotics

Spring 2024

BASEBALL BAT EMULATOR



Aby Jose

Charles Joyner

Ryan Nguyen

Omar Ramos Escoto

Final Project Report

Ryan Nguyen, Aby Jose, Charles Joyner & Omar Ramos Escoto

Introduction

Our project, inspired by interactive games like Wii baseball, aims to create an engaging and realistic baseball simulation. The core idea is to enable a human player to pitch a virtual ball to a robot, which then attempts to hit it. This involves sophisticated tracking, data transformation, and robot control mechanisms to achieve a seamless and immersive experience.

The code repository for the project can be found: https://github.com/aby25jose/cs225a-baseball

And the final video for the project can be found: Baseball_video.mp4

Project Description

The primary goal of our project is to develop an interactive baseball simulation where a human player pitches a virtual ball to a robot batter. The system is designed to accurately capture the human's hand movements using Optitrack cameras, transform this data into a usable format for the simulation, and control a robotic arm to hit the incoming virtual ball.

Key Components

1. Human Interface
2. Robot Batter
3. Simulation environment

Project Objectives

Objective 1: Design a Reliable Robot Controller

The first objective is to design a robot controller capable of reliably hitting a quickly moving target within its strike zone. This involves:

- **Trajectory Generation:** Creating a cubic polynomial trajectory for the bat based on the initial position, velocity, and desired end position.
- **Orientation Tracking:** Calculating the rotation matrix to transform the initial orientation to the desired end orientation, and implementing time-stepped angle-axis rotation.
- **Controller Update:** Continuously updating the robot controller with the desired position, velocity, and acceleration at each time step to ensure accurate motion tracking.

Objective 2: Develop an Interactive Human Component

The second objective is to create an interactive component that allows a human to throw a virtual ball. This requires:

- **Accurate Position Tracking:** Using the Optitrack system to record the user's hand position and orientation in real-time.
- **Data Processing:** Implementing numerical differentiation to find the hand velocity, which is crucial for modeling the ball's trajectory.
- **Simulation Integration:** Outputting the calculated ball velocity to the simulation for kinematic calculations when the user releases the ball.

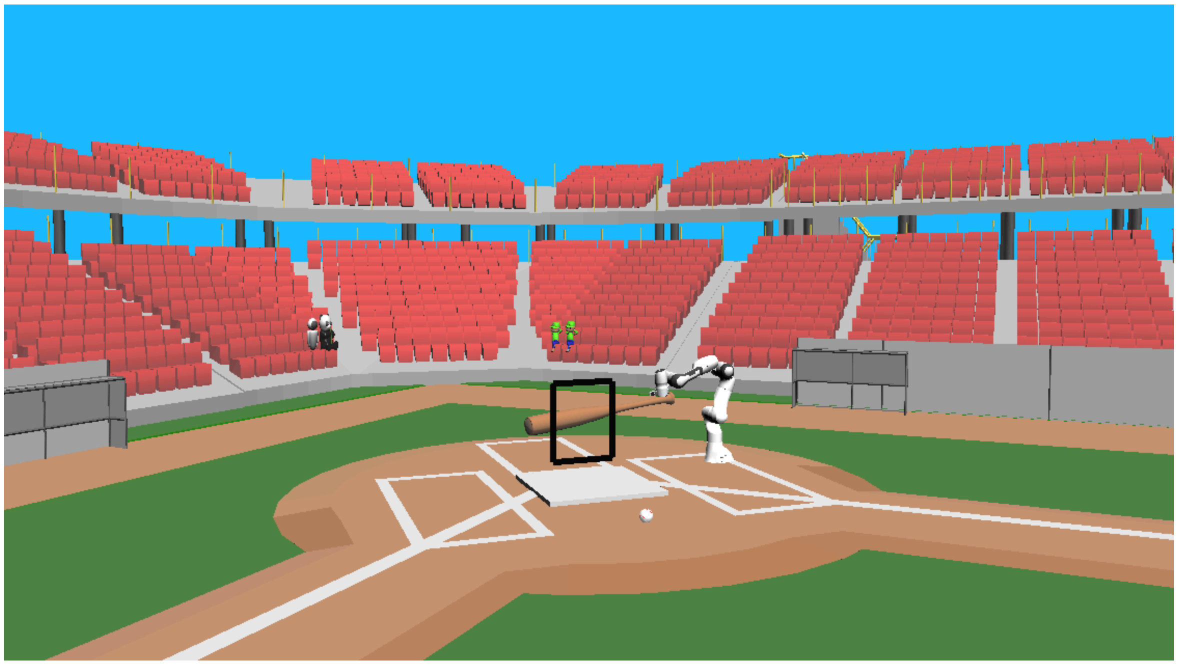
Robot Description

Our robot is a modified version of the Panda robotic arm, customized for the simulation of a baseball batting robot. The arm's seven links and revolute joints provide a range of motion essential for simulating dynamic baseball swings.

The distinctive feature of this robot is the baseball bat attached to its end-effector via a fixed joint, allowing for authentic batting actions. The bat's visual representation is derived from a scaled-down 3D model of a typical baseball bat, ensuring a realistic appearance within the simulation. More importantly, the collision mesh is configured to align with the bat's physical orientation and the typical angle of a bat during swing. This setup not only enhances the realism of the bat's interaction with the baseball but also fine-tunes the simulation to reflect the actual impact dynamics found in baseball.

Simulation World Environment Description

The simulation environment is designed to provide a realistic baseball batting experience. Set within a virtual stadium, the Panda arm equipped with a baseball bat interacts dynamically with pitched baseballs on a simulated baseball field. This field, complete with collision properties, allows the ball to roll on the ground and stop behind the robot if necessary. A visible strike zone aids the pitcher in targeting, and the surrounding stadium enhances the immersive experience of the simulation.



The simulation is powered by Sai2Graphics for visual rendering and Sai2Simulation for physics-based interactions, structured across multiple threads to ensure smooth motion and timely responsiveness. One thread is dedicated to simulating physical dynamics, including the robot's movements and the baseball's trajectory, while another handles user inputs and external updates. A Redis server facilitates communication between different components, like the controller and the graphical interface, enabling asynchronous updates and cohesive state management across the system.

To ensure consistency and manage concurrency, mutexes protect shared resources, preventing race conditions during updates. These are vital when updating positions and velocities within the simulation, ensuring atomic and consistent changes, which is especially important for the baseball dynamic object. Redis also plays a crucial role by storing and broadcasting real-time data like the baseball’s position and velocity, and event notifications such as the ball crossing the strike zone. Additionally, velocities are mapped and scaled within the simulation to reflect realistic physical behavior, ensuring the baseball behaves intuitively for users.

Human interface

Our Human interface uses two main components to allow the user to interact with the virtual environment. The Optitrack system gives us the pose information about the pitcher hand and the glove provides data about the ball's state.

Optitrack:

In our interactive baseball simulation project, we aimed to create a realistic and engaging experience where a human pitches a virtual ball, and a robot hits it. To achieve this, we needed precise tracking of the human's hand movements to control the ball's trajectory. For this purpose, we employed the Optitrack motion capture system to obtain accurate position and orientation data of the user's glove. This data was crucial for:

- Accurately simulating the ball's release.
- Transforming the motion capture data into the virtual simulation's coordinate system (OpenSai world frame).
- Calculating the velocity of the ball based on the positional data.

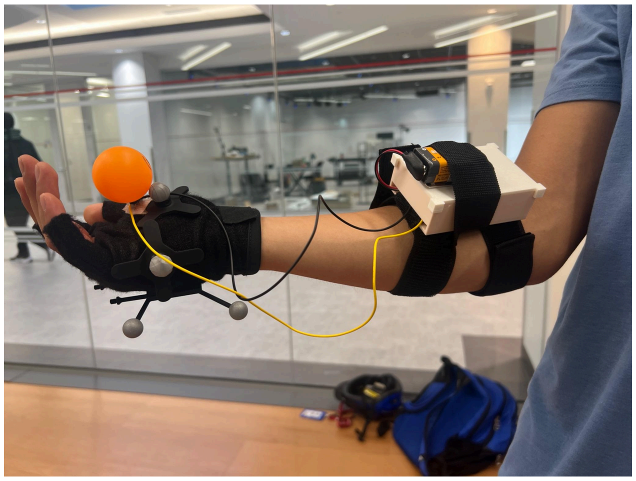
Reflective markers were attached to the glove worn by the user to track its movements precisely.
The marker data was streamed to the Optitrack system, which processed the coordinates of the glove in real-time. The position and orientation data was streamed from the Optitrack computer to our system via Redis. However, the position and orientation of the glove was in the Optitrack world frame, and we had to ensure the right transformations to get it into the OpenSAI frame.

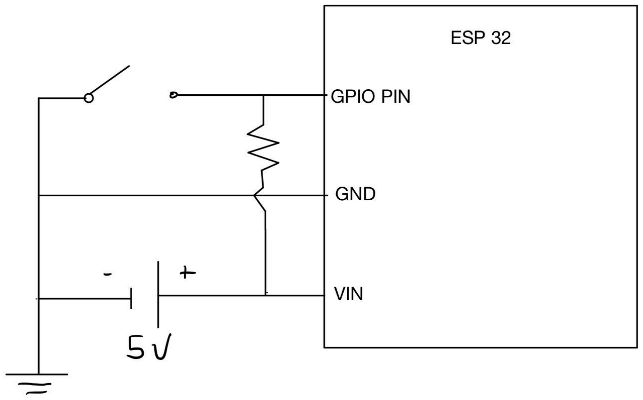
The Optitrack frame origin was taken to be the initial position and orientation of the glove when the simulation starts up. The OpenSAI frame origin was hardcoded to be the position of the pitcher mound, whose coordinates were fixed with respect to the base frame of the Panda robot. The rotation matrix from the OpenSAI frame to the Optitrack frame was a negative 90 degree rotation about the x-Axis.

We then check the relative motions of the current position and orientations of the glove with respect to the Optitrack origin defined earlier. We then convert these relative motions to the OpenSAI frame to obtain the relevant positions and orientation of the glove in the simulation environment. This accurately captures the relevant information of the ball required for the simulation.

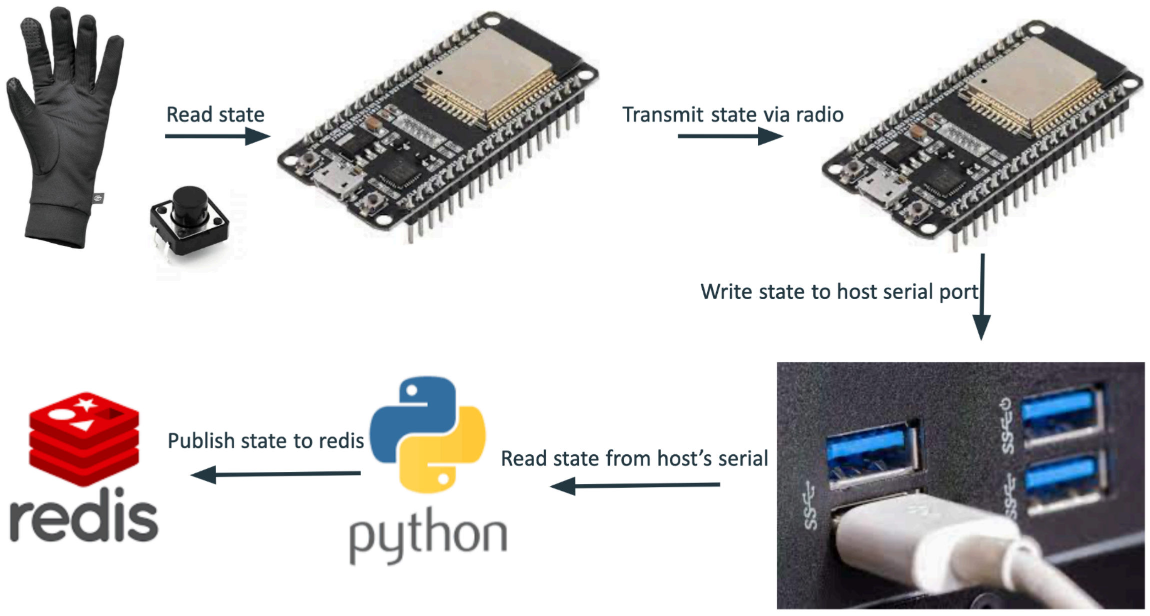
The position data was then numerically differentiated at the rate of 120 Hz (which was the camera capture rate). This gave us the velocity of the glove at each timestep in the simulation environment.

Glove:

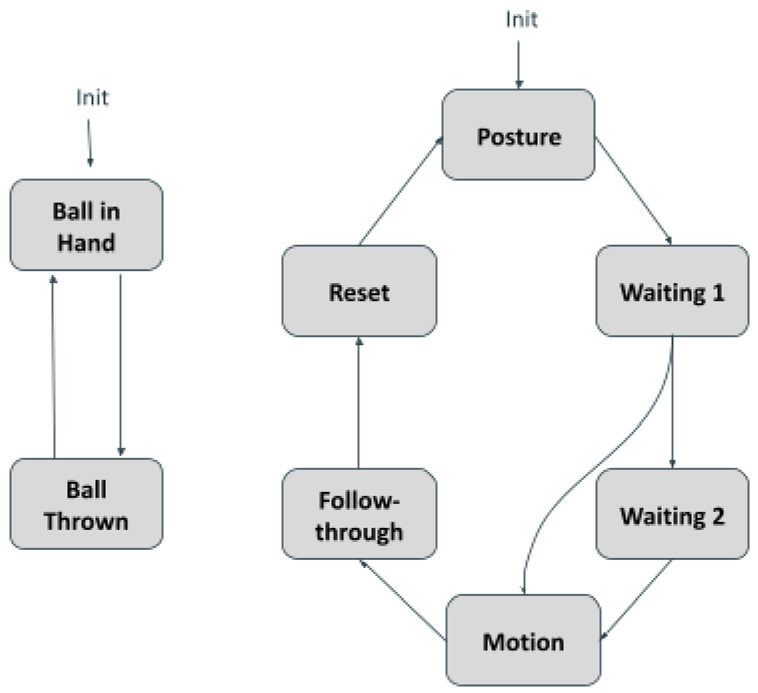




The glove consists of an esp32 Wifi module, a pull up circuit connected to a button, and a power source. The button is positioned in such a way that when the user grips their hand in a fist– similar to how a person would grip a baseball– the button is forced into a depressed state. When the person opens their hand–releasing the virtual baseball– the button returns to its base state. The ESP 32 wifi module has GPIO pins that can be used to read digital signals. The GPIO pins read high when the button is open and low when the button is depressed. The wifi module then writes over radio to a companion receiver ESP 32 device. The receiver displays the button's state via a led(on for button pressed otherwise off) and then writes the button state over serial to the computer running the simulation. The host computer then runs a python script to pull the data from its serial ports and publish it to the redis server. The C++ code then can read the button's state from redis at will. Below is a visual depiction of the entire system.



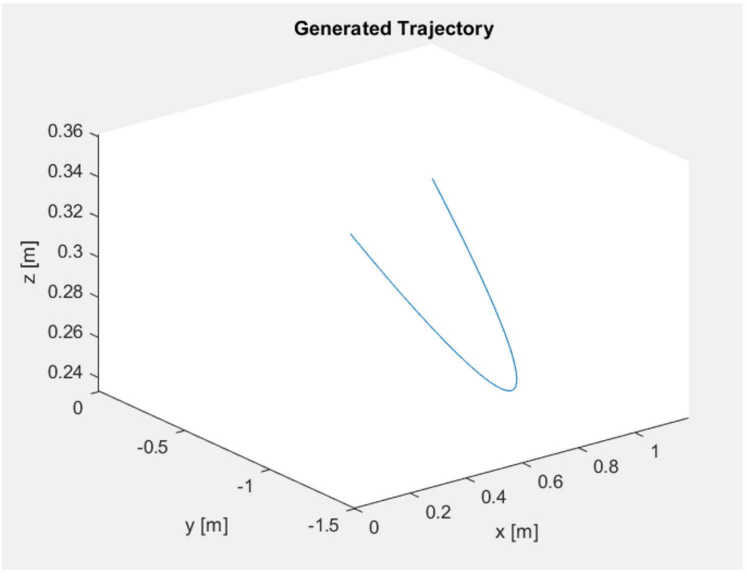
State Machine and Controllers



There are two state machines for controlling the ball and the robot which are separate but in communication with each using redis.

Robot: Posture - Upon startup, the robot is controlled by a joint task controller to the starting position of the robot ready for a swing. This is done with a joint task controller to keep the robot away from joint limits and singularities. Because the initial position of the bat before the swing is not critical to the performance of the robot, we do not need to control it in task space. The robot changes state to Waiting 1 once the joint angles of the robot are close to our desired configuration.

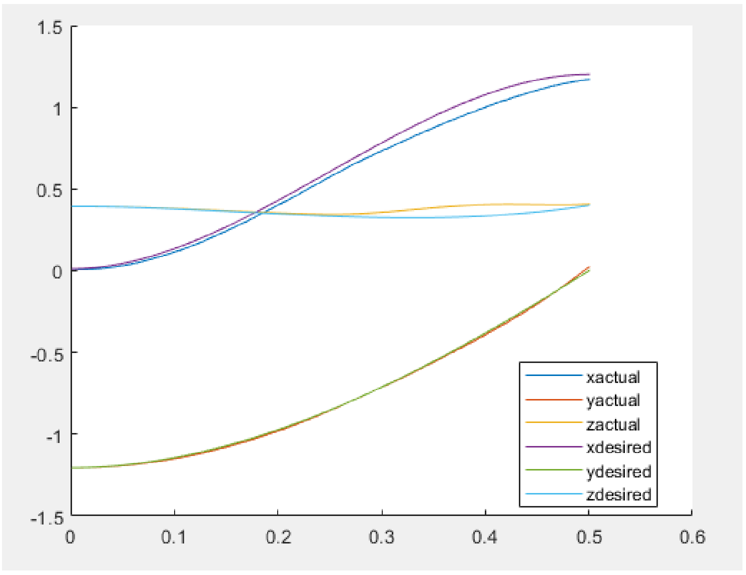
Robot: Waiting 1 - In the Waiting 1 state, the robot simply holds its position until a signal is sent via redis that a ball will be entering the strike zone. The information sent via redis includes the position where the ball will enter the strike zone (x,y,z) and the time it will take for the ball to reach it. These values are calculated when the state of the ball transitions from Ball in Hand to Ball Thrown. If the time it takes for the ball to reach the strike zone is less than 0.5 seconds, a predetermined swing time, the controller will



calculate a trajectory such that the bat will impact the ball with a predetermined desired velocity in the time it takes the ball to reach the zone, transitioning to Motion state. This trajectory is a cubic polynomial based on the initial starting and end positions and velocities of the bat. Additionally the end effector orientation is calculated for all points along the swing path by finding the axis of rotation and angle of rotation from the start orientation to the desired end orientation. The orientation "trajectory" is simply the start orientation rotated by the angle of rotation about the axis of rotation. If the time it takes for the ball to reach the strike zone is greater than 0.5 seconds, the robot will wait until the ball is 0.5 seconds away from the strike zone in Waiting 2 before calculating the trajectory and switching to Motion state.

Robot: Waiting 2 - In the Waiting 2 state, the robot holds its state until the ball is 0.5 seconds away from the strike zone.

Robot: Motion - In the Motion state, the robot follows the calculated trajectory and orientation. This is done by calculating the desired position, velocity, acceleration, and orientation of the end effector in each loop based on the elapsed time since the swing began. The velocity and acceleration trajectory are simply derivatives of the position data. The robot then is controlled in task space by setting the desired position, velocity, acceleration, and orientation. In the nullspace of the task space, we set the desired joint angle trajectory to avoid joint limits and singularities. After the calculated swing time, the robot transitions to the Followthrough state.



Robot: Followthrough state - In the Followthrough state, the robot is controlled identically as in the Motion state, except the calculated cubic trajectory and orientation uses the start position velocity and position of the bat when it is transitioning to the Followthrough state and brings it to rest at a predetermined end position. After 2 seconds passes, the robot transitions to the Reset state.

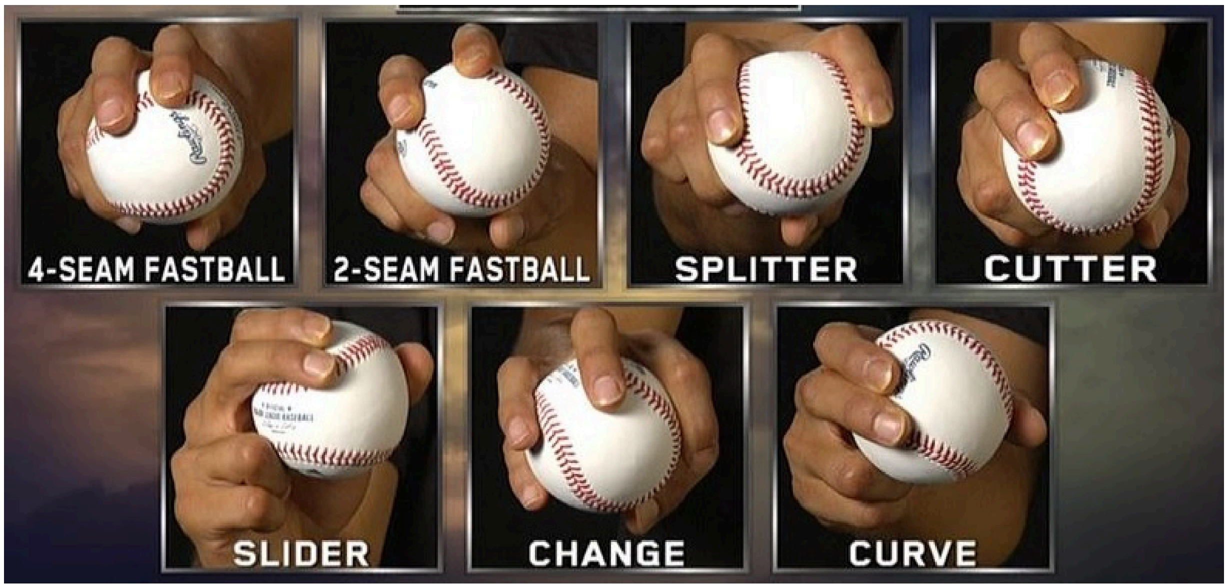
Robot: Reset state - In the Reset state, the robot is controlled identically as in the Motion state, except there is not a cubic trajectory generated, but rather a predetermined trajectory in the path of a circle in the x-y plane. After 5 seconds passes, the robot transitions to the back Posture state to position itself ready for a new swing.

Ball: Ball in Hand - Upon startup, once the robot reaches Waiting 1 state, the ball starts in the Ball in Hand state, meaning that its position is controlled by Optitrack readings. When the user presses and releases the button, the ball transitions to the Ball Thrown State. The velocity at the instant in time is calculated, and the ball's initial position and velocity data can be used to determine if the ball will enter the strike zone with simple kinematics. If so, the ball's position and time to reach the strike zone data is sent via redis to the robot such that it can transition out of its Waiting 1 state.

Ball: Ball Thrown - The Ball Thrown state means that the ball's position is completely determined by simulation and collisions. We do not take into account any Optitrack data. If the ball is not going to enter the strike zone, we transition to Ball in Hand state after the ball reaches a position of y = -2m. If the ball does enter the strike zone, we transition to Ball in Hand state after the robot returns to the Waiting 1 state - in which a full cycle of a swing has already occurred.

Challenges and future work

Throwing Intuition, Glove State, and Haptics: One issue we faced after pivoting from throwing a real ball with Optitrack tracking to using a completely virtual baseball was making pitching the virtual ball feel intuitive. We used a button on the Optitrack glove's palm to indicate whether the ball is gripped or released, pressing it when making a fist. However, the small contact pad required the pitcher to change their grip and focus on keeping the button pressed, detracting from the system's intuitiveness. The button worked well as a proof of concept, but we plan to replace it with a flex sensor-based system on each finger. This will allow us to measure the glove's open or closed state without a specific grip, providing analog outputs for better calibration to individual users. Additionally, the way the pitcher grips the ball affects its angular acceleration and spin dynamics during flight (see reference below).



With a flex sensor on each finger we can guess the type of grip the pitcher is utilizing and add a spin component to the glove's state. Allowing us to make a more realistic simulation that changes the ball's dynamics with respect to how the pitcher grips and spins the ball. One large benefit of the button is that it provides really good tactile and auditory feedback every time it changes state. To replace this helpful feedback we could potentially use haptic feedback in the form of vibrations to indicate that the ball is being gripped. With the aforementioned changes we can greatly increase the ease of use and realism of our simulation.