CueTips

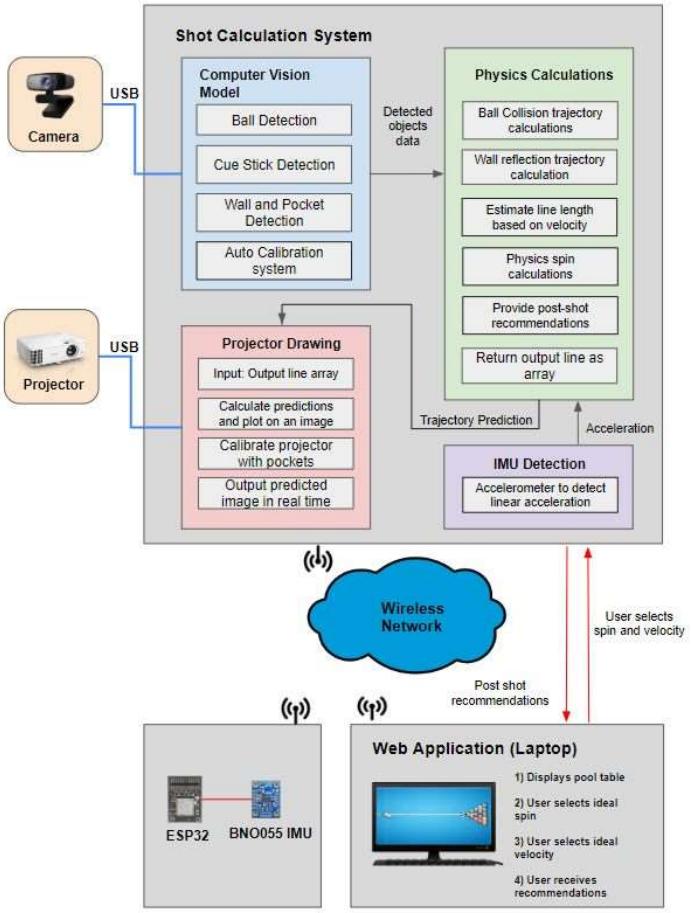
C0: Tjun Jet Ong, Andrew Gao, Debrina Angelica
18-500 Capstone Design, Spring 2024
Electrical and Computer Engineering Department
Carnegie Mellon University

Product Pitch

There is a steep learning curve when it comes to learning how to play billiards. Without proper guidance from professionals or experienced friends, it can often lead to frustration or discouragements. CueTips uses Computer Vision to enhance the pool learning experience by predicting shot trajectories in real time, acting as a personal coach and guiding your shots with precision.

Our goal is to create an immersive, responsive experience that's precise and engaging. Important key metrics we wanted to achieve were: accurate ball detection **(<0.2in),** accurate shot calculations (**<2 degrees of the margin of error),** and a full system end-to-end latency of **less than 100ms.** Throughout the semester, we managed to achieve an average ball prediction accuracy of **0.05inches,** an average shot prediction accuracy of **1.13 degrees margin of error**, and an end-to-end latency of **about 22ms.** To interact with our system, there will also be a web application available for users to select their desired metrics.

System Architecture



Conclusions & Additional Information

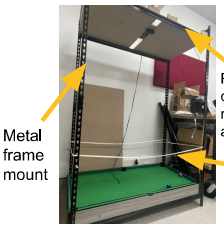
We were able to achieve most of the goals we set for ourselves, and built a product that provides instantaneous trajectory predictions for pool players of all levels. Some next steps would be to migrate the software to an Nvidia Jetson Nano, create a more personalized training experience for users (play recommendations, technique adjustments specific to each user), and tracking ball spin automatically with a better IMU. We'd also like to test on a life-sized pool table if possible. We learned a lot about applied computer vision in a real-world product, as well as integration between multiple sensors and devices.

SCAN ME to view our website!

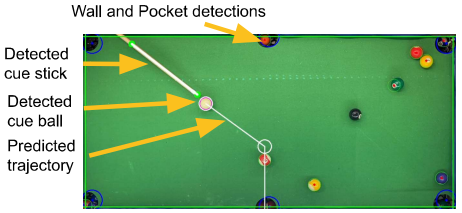
System Description

Our product is composed of (6) primary systems:

- **Ball Detection System**
  - This is responsible for detecting all balls currently on the pool table, excluding the ones that have already been pocketed. It differentiates between the cue ball and the other balls. We used OpenCV's HoughCircles to detect the balls.
- **Cue Stick Detection System**
  - This will find the cue stick on the pool table and detect its trajectory line - where the user is aiming. This system also includes an accelerometer attached to the cue stick that detects how fast the user is hitting. This is done using color masking and OpenCV's BoundingRect functions.
- **Wall and Pocket Calibration System**
  - The calibration system detects where the pool table walls are as well as the six pockets using Canny Edge Detection and OpenCV's HoughLines. These are calibrated and set in the beginning and kept constant while the software runs.
- **Physics Engine System**
  - The physics engine takes in data from all the other systems and computes the predicted trajectory of the user's aim. As the user moves the cue stick around, the physics engine computes the changes in trajectory in real-time.
- **Web Application System**
  - This consists of a frontend (React) and backend (Flask), receives user data about where they want to strike the ball, and sends it to the backend which is integrated with the physics engine and other systems.
- **Projection System**
  - This is responsible for taking the predicted trajectories and outputting them to the projector for instantaneous feedback for the user's aim.



Full Product, Side View



Ball, Physics, Cue Stick, Calibration

System Evaluation

We evaluated our system's performance based on its processing speed and accuracy. A tradeoff that was made in order to achieve low latency was to forgo the use of machine learning. We were able to achieve a latency of 22ms, far exceeding expectations, by using OpenCV for image processing. Accuracy of our system was evaluated based on the accuracy of the ball detections and the trajectory predictions. We found that our ball detection accuracy performed very well, with an average error of 0.05in. Ball detections also performed well in potential edge cases where balls are placed adjacent to each other or nearby the table walls and pockets. The performance of our trajectory prediction accuracy was very high, especially for regular shots (aiming target ball to pocket). Though there were more unsuccessful shots for bank shots (bouncing target ball of a wall and into pocket).

Test Results

Test Metric	Testing Methodology	Target	Measured Performance
Ball Detection Accuracy	Project ball detection. Measure distance between real and projected balls.	Detect pool balls within **0.2in** from actual ball location	**0.05in** average error *sample:* *5 frames, 15 balls per frame*
Trajectory Prediction Accuracy	*Success Ratio* Project predicted trajectory on the table. Take 20 shots to hit a target ball to the pocket by following our trajectory guidance. Determine percentage of successful shots.	**95%** of shots must successfully pocket a ball	Normal shots: **100%** Kiss shots: **90%** Bank shots: **75%** *sample:* *20 shots per shot type*
	Angle Deviation: Project the predicted trajectory on table. Take 20 shots to hit a target ball to the wall. Measure average angle deviation from actual to projected coordinate of the wall collision.	Predicted trajectory must be within **2°** of actual trajectory	**1.13°** error on average *sample: 20 shots*
End-to-end Latency	Time the code execution from the point when it receives a particular frame to the point when it outputs a predicted trajectory based on that frame.	Achieve per-frame processing time of at most **100ms**	**22ms** on average *sample: 10 runs*