# C0: CueTips

Andrew Gao, Debrina Angelica, and Tjun Jet Ong

Department of Electrical and Computer Engineering,
Carnegie Mellon University

***Abstract*— CueTips is an eight-ball pool training system that caters to both beginners and professionals, aiding players in developing an intuition on how to play the game of pool. By processing real-time video feed from the pool table, our system provides users with visual projections of shot trajectories as they move their cue stick around. Unlike existing technologies that rely on cue stick attachment for aim assistance or force control, our system minimally alters the cue stick, preserving the authenticity of the game experience. Furthermore, our system presents users with a comprehensive view of all potential shot trajectories based on where the user aims, offering unparalleled training opportunities for pool enthusiasts of all skill levels.**

***Index Terms*—billiards, eight-ball, pool, trajectory prediction, computer vision**

## I. Introduction

Our project, CueTips, is an assistive system that helps beginner and advanced Eight-Ball Pool players improve their skills. Eight ball pool is a difficult game to learn, primarily due to the high degree of accuracy needed to make a successful shot to pocket a ball. Very slight differences in the angle of the player's aim can have a profound impact on the trajectory of the ball they hit. Without expert guidance to correct players' aim, it can take a very long time to develop an intuition for where the ball will go depending on the player's aim. While it is possible for players to hire coaches to improve their aim, doing so may not be feasible for the average player who wants to play recreationally.

Our product is an assistive feedback system that provides real-time predicted trajectories based on the player's cue stick position. As the player moves their cue stick around, they will be able to see a projection line showing them the direction and angle in which the ball they are aiming at will go. This will help players know whether they will be able to pocket the ball and subsequently give them insight on the correct aim location.

A competition of our project is a similar project that was done in the Spring 2023 semester for Carnegie Mellon University's (CMU) Electrical and Computer Engineering (ECE) Capstone course. This project looked at the current stationary state of the pool table and suggested the most optimal shot to take. The advantage that our project has compared to this one is that our project provides real-time feedback to users, based on where the user positions the cue stick. This allows players to make the best decision on the shot to take and allows users to improve their intuition for aiming. If the user's aim is off, they will be able to notice that from our trajectory projection, which allows for them to correct themselves. The goal is that over time, this is more beneficial for helping users learn as it provides more personalized feedback based on the user's movement and position of the cue stick, relative to the pool table. Furthermore, our system gives users the flexibility to choose which ball to hit, which would allow them to improve their decision-making skills in the game of pool as they get to observe the trajectories of the different balls on the table through trial and error.

Our physics model also has some differences that differ from the prior project. Firstly, our system allows users to practice bank shots since we will also display the trajectory of the cue ball if it is aimed towards the walls of the table. Moreover, our physics model also allows for kiss shots, where users can execute a shot with multiple ball collisions. In addition, our physics model can also take into account the user's desired spin and show users the cue ball's deflection angle after a collision with a wall or target ball. This allows users to get an idea of where the cue ball will land after the shot is executed with the spin that they selected.

## II. Use-Case Requirements

The primary goal of our project is to serve as a tool that effectively helps users learn to play pool.

One of the features required to facilitate fast learning is to ensure that our system's performance is fast. We want our system to be able to react instantaneously to user movement as they play the game. Hence, we must be able to achieve a latency of at most 100ms. This latency is measured from the time the user shifts their position to the time that our system provides and updates a projection for the trajectory of the user's aim. We chose a threshold of 100ms as research has shown that this duration is the threshold required to create the illusion of an instantaneous response [1]. This immediate feedback is not only important in creating a responsive user interface, but also crucial in helping users learn instantly from their mistakes if their aim is not accurate.

The accuracy of the predicted trajectory line is another crucial use case requirement for our project. In order to effectively help users to learn how to improve their aim, we must be able to provide them with accurate predictions on the ball trajectory based on the cue stick's position. We aim to have our predictions accurate to at most 2 degrees of error. 2 degrees of error is the measured angle between the line of our predicted trajectory and the line of the ball's actual path after the shot is executed. We chose to select an error of two degrees because the pockets are of a width that would allow a ball to still enter even if it were 2 degrees to the left or right of its predicted trajectory [2]. This was measured from the full diagonal length of the table, assuming that the trajectory of the ball was directed to the center of the pocket.

In order to facilitate the high accuracy our model aims for, we must also ensure that our object detection models are accurate. We aim to be able to detect ball position within 0.2 inches of their actual position. If our model's perceived location of a ball skews too far away from the ball's true position, this would cause inaccuracies in our physics calculations that drive the trajectory prediction.

CueTips addresses some crucial aspects of public health, safety, and welfare by promoting mental well-being, physical activity, and cognitive health. Our product aims to alleviate the stress associated with the learning process of pool, fostering a positive and supportive environment for individuals to learn pool. Furthermore, the interactive nature of the game serves as a form of recreational exercise, allowing individuals to move away from their sedentary lifestyles. In addition, it could also be useful for older individuals to engage in activities that stimulate cognitive functions and maintain mental acuity. In addition, when building the product, we made sure to build a steady shelf as a structure to ensure that users were safe when using the tool. As items such as a projector and camera were placed overhead, we made sure that the structure could support the weight and maintained safety of users.

In terms of social factors, our product aims to be accessible and reach a wide range of users. This motivates our choice of outputting visual trajectory predictions, as it would inherently be community-building by serving as a massive social interest in many parts of the world. By having a simple, intuitive, and visual interface, CueTips makes no language assumptions and has an intuitive user interface that is understandable to everyone. In terms of environmental factors, we also made sure that biodegradable, reusable material was used as the shelf so that it can easily and quickly be dismantled and reused. Being accessible to everyone, CueTips creates room for vast economic potential. We could strategically partner with entertainment venues, gaming centers, and sports bars to target the demographic who are not only passionate about the game of pool, but also seeking interactive and technologically advanced gaming experiences. Through upfront hardware sales, subscription models, and potential collaborations with game developers for exclusive content, there is a lot of potential for our product to contribute significantly to the economy.

## III. Architecture and/or Principle of Operation

Our project consists of several physical components. It consists of a pool table mounted onto a shelf to house, a camera for object detection, and a projector to project recommended trajectories. It also includes a web application to display the user's shot and provide personalized recommendations.

### *A. Physical Structure*

The physical structure of our set-up consists of a RayChee pool table mounted onto a rack. The pool table is directly mounted onto one of the racks of the frame. This structure is 48 inches wide, 24 inches long, and 72 inches tall. There will also be a second rack that is 48 inches above the pool table. After some testing and calibration, we determined that the optimal distance from the projector and the camera to achieve the best field of view is when they are both placed 44 inches above the pool table. We raised that by four inches in order to ensure some additional leeway in case we needed a greater field of view when performing our actual detection model. Figure 1 shows a picture of our final structural setup.

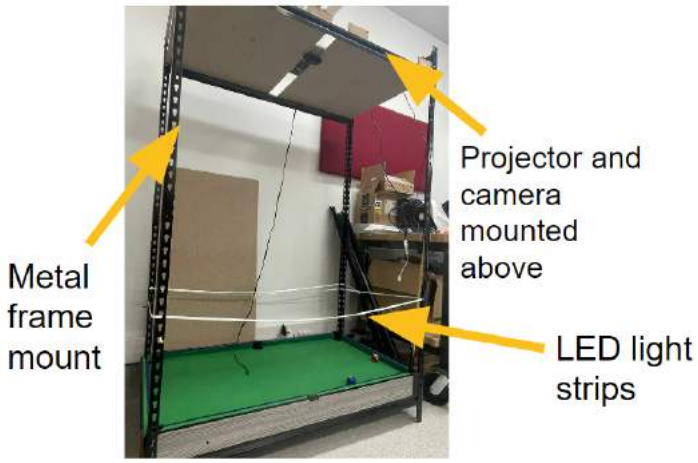



Fig. 1. Pool table incorporated into a shelving unit, mounted on a rack

### *B. Camera and Computer Vision*

The camera we are using to perform the computer vision model is the Logitech C922 Pro Stream 1080p webcam. We will cut out the appropriate portion of our plywood rack and mount the camera nicely into the cut-out, in order to achieve a good image capture of the table. The camera will be connected directly to our computer via a USB Cable. The camera gives a 78-degree Field of View, meaning that if we want to get a full capture of the pool table, we will at least have to mount it 33 inches above the pool table.

We will run the backend computer vision model on our own computers. This will include the detection of the current state of balls on the table, detecting which ball is the cue ball, balls, the cue stick, pockets, and walls. This information will then be used as input to our physics model, which we will use to predict the trajectory of the shot the user will take. Figure 2 shows the full process of what happens from the moment the camera inputs are detected, to the trajectory calculations, and finally to the projection of the predicted trajectory.

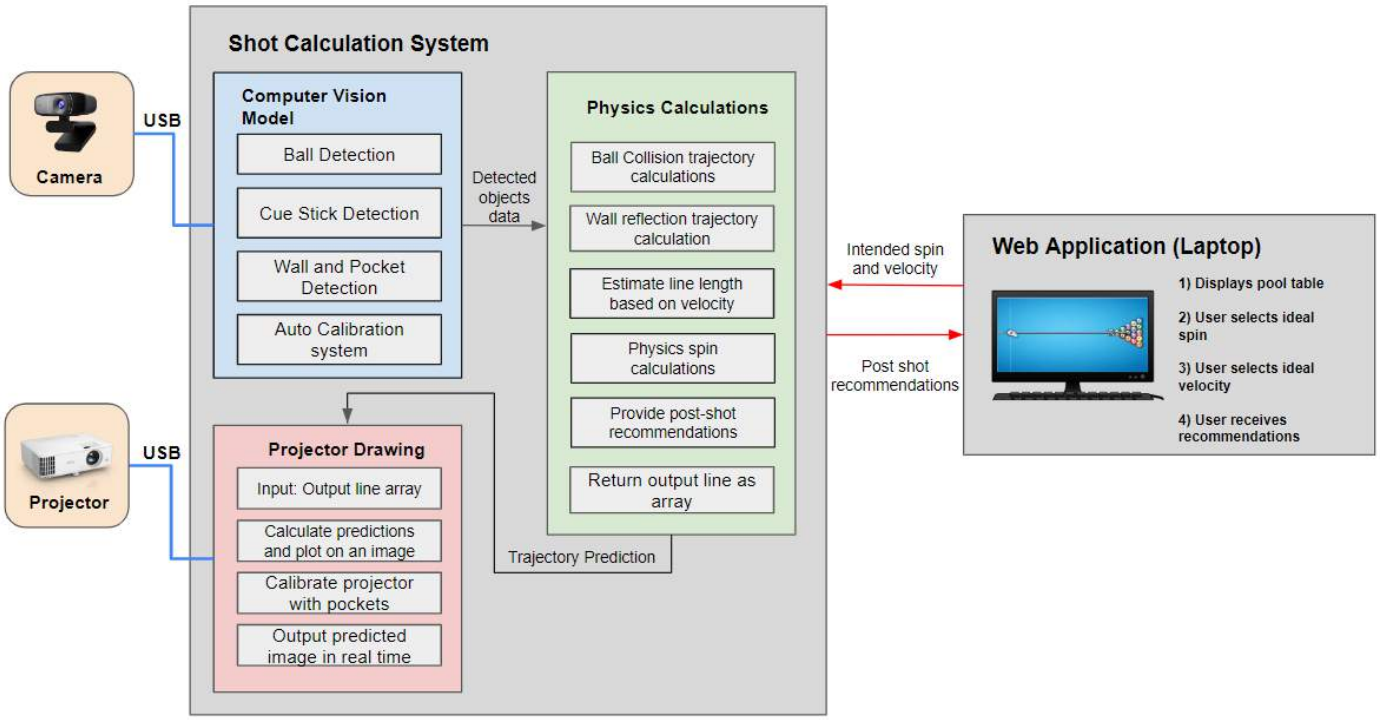



Fig. 2. *BLOCK DIAGRAM OF FULL SYSTEM ARCHITECTURE*.

### *C. Physics Model*

There are five key outputs of the computer vision model that we will use: An array that contains the coordinates of the centers and radius of all colored balls, the center and radius of the cue ball, the location of the cue stick encoded as two points - the cue tip and the back of the cue, an array of line equations of the four walls, and an array that contains the coordinates of the six pockets. Our physics subsystem contains three main parts, a function to calculate reflections against the walls, a function that calculates ball collisions, and a function that detects when the balls enter the pockets. In order to implement the trajectory predictions, we first identify the locations of the cue ball's center and the two points of the cue stick. Whenever a user aims at the cue ball, these three points should be almost collinear. By calculating the Euclidean distance between the cue ball and the cue tip, as well as the perpendicular distance between the cue ball's center and the extrapolated line of the cue stick, we only project the line if the cue stick is held close to the cue ball.

In order to implement the wall reflections and the trajectory predictions, we first identify the locations of the cue ball's center and two points on the cue stick. Whenever a user aims at a cue ball, these three points should be collinear. We will then extrapolate a line across these three points to estimate the trajectory that the cue ball will go. As we extend this line further, we will meet either one of the four cases: There is no obstacle, there is a wall that intersects the line, or there is a ball that intersects the line, the ball enters a pocket. After taking into account all of these, we will output the final predicted trajectory that the cue ball will take given the current aim. The predicted trajectory line is then sent as input into the projector subsystem.

### *D. Projector*

The projector we are using for this project is the VOPLLS 1080P Full HD Supported Video Projector. This projector takes in the coordinates of the predicted trajectory line and plots it as a white line on a black background. This system will also make sure that if the line coincides with any of the pockets, it will stop the trajectory of the line to indicate that the ball will go in. The projector will also be connected to the laptop via HDMI Cable.

### *E. Web Application*

Our web application provides a nice user interface to livestream the video output and displays the ball trajectories onto the table. The application also allows for a method for users to select their desired spin, which they intend to execute. A distinctive feature of this web application lies in the provision of actionable feedback, promptly communicating errors to users and offering strategic insights for improvement based on individual playstyles. After they execute their shot with the intended spin, our system will look at the location that the user hit the ball and provide recommendations on whether the spin was successfully executed or not. We will use React to create our front-end web application. For the backend, we will deploy a Flask server to receive video input from the camera and stream it into the front-end.

## IV. Design Requirements

To ensure that object detection was as accurate as possible, we defined some requirements for the optimal performance of the computer vision system. A lot of subsystems fundamentally depend on the accuracy of the detection of objects in our frame. These objects include the walls, pockets, cue ball, balls on the table, and the cue stick. Thus, we needed to ensure that we selected a camera that was precise enough to cater for this. We minimally needed to choose a camera that could give us 30 frames per second, and a resolution of 1080 pixels.

Our design requirements for CueTips are a translation of our use case requirements defined in section II.

### *A. Computer Vision Subsystem*

To ensure that object detection is as accurate as possible, we defined some requirements for the optimal performance of the computer vision system. A lot of subsystems fundamentally depend on the accuracy of the detection of objects in our frame. These objects include the walls, pockets, cue ball, balls on the table, and the cue stick. Thus, we needed to ensure that we selected a camera that was precise enough to cater for this. We minimally needed to choose a camera that could give us 30 frames per second, and a resolution of 1080 pixels.

This level of camera quality would facilitate more accurate image processing using computer vision, ultimately allowing us to achieve the main requirement for this subsystem--a maximum ball detection error of no more than 0.2 inches, which is measured from the center of the ball detection to the center of the ball's actual location. High accuracy in this subsystem would mitigate error propagation that may lead to inaccuracies in other dependent subsystems.

### *B. End to End Latency*

We aim to output feedback to the user with a latency of at most 100ms. End-to-end latency is defined as the time taken from the detection of the first frame, to the projection of the output frame. In order to ensure that users have a seamless experience (smooth experience), we must ensure that end-to-end latency falls below 100ms as this is the threshold beyond which a response is no longer perceived as instantaneous. To achieve this requirement, we decided to implement an approach that uses zero machine learning. Our system purely uses highly optimized functions for image processing in the OpenCV, making it possible for us to achieve lower latencies.

### *C. Projection & Feedback Subsystem*

The projection & feedback subsystem must be able to provide an intuitive user interface so that it is easy for the user to follow the feedback provided by our system. We will use a projector mounted overhead to display the ball trajectories predicted by our model onto the pool table. This allows the users to clearly see the trajectories as lines which they can then use to inform them of whether a ball will successfully go into the pocket.

Furthermore, this subsystem must display accurate trajectory predictions to users to accelerate their learning. Providing inaccurate feedback to users would deter their learning as users would have misconceptions on the outcome of their aim. Hence, it is crucial that our trajectory outputs have no more than a 2-degree error from the trajectory executed by the user in real life. In other words, the actual path followed by the target ball after it is hit should be no more than 2 degrees to the left or right of the predicted trajectory. In addition to this maximum error requirement, this subsystem must be able to calculate trajectory predictions that are 95% successful. This means that the user should be able to successfully pocket a ball 95% of the time when following the trajectory feedback provided by our system.

## V. Design Trade Studies

In our previous Design Review, we mentioned four main tradeoff decisions involving latency requirements, mount construction, foregoing machine learning, and building the physics engine. Since then, we made several mode design tradeoffs: cue stick detection improvements, handling ball spin, and calibration.

### *A. Power vs efficiency for latency and usability*

Previously discussed in our design review, one of the larger hardware design tradeoffs we made was choosing the hardware for our web application, computer vision model, and physics engine. At the beginning of our project, we wanted to run our software on an FPGA or Nvidia Jetson Nano for efficiency, but quickly realized that this detracts from the core use case requirements of the project. Whether the system is run on an FPGA, Nvidia Jetson Nano, or laptop does not change the core functionality of the project. If we opted to run this on an FPGA or Jetson Nano, it would have distracted us from improving the various computer vision subsystems or developing the web application - core parts of the user experience. Furthermore, there are two more technical roadblocks we would have run into had we run this on an FPGA or Jetson: latency and projector compatibility. For latency, the clock speed of both an FPGA and Nvidia Jetson Nano are orders of magnitude slower than that of a laptop (GHz vs MHz). In our preliminary testing, we found that running most computer vision algorithms (HoughLines, contour detection, Gaussian blurring, etc) take a few milliseconds at most; not only would there be technical challenges in writing CV code for Jetson Nano, FPGA (writing in C/C++ vs Python), but these algorithms would run significantly slower. Our final system has an end-to-end latency of roughly 22ms on average, with the highest peaks being around the 30-40 ms range. This is well within our latency requirements but would not be the case if we ran our system on an FPGA or Jetson Nano. Regarding projector compatibility, there is an added difficulty of getting the projector to work with either an FPGA or Jetson Nano. Since neither of these two have a graphical user interface, we would have to find a more complicated workaround to display the predicted trajectories onto the pool table. However, using a laptop lets us simply

mirror the screen while running the system in order to display the predicted trajectories.

### *B. Mount construction for the camera and projector*

One of the best decisions we made constructing the system was opting for an out-of-the-box solution for our camera and projector mount. We were faced with two decisions: construct a custom mount for the camera and projector, or buy/adapt an existing solution for our needs. If we custom constructed a mount, it would be cheaper than buying but take more time to build. If we bought/adapted an existing structure to our needs, it would cost more money but take less time to get working. We ultimately decided to adapt an existing solution for our needs; we bought a metal shelf, partly deconstructed it, and used the topmost shelf to hold our projector and camera. The assembly only took us a few days, compared to what likely would have been at least a week or two. Since we only have 12-13 weeks for the project, saving time was much more important than budget. Additionally, we had not cut into our budget too much, so it made sense to buy the shelf and adapt it for our purposes.

### *C. Foregoing machine learning in CV subsystems*

The largest software tradeoff we made was deciding to forego machine learning from the start of our project. Machine learning is effective for object detection and would remove much of the algorithmic complexity of our computer vision subsystems. However, it adds a significant amount of latency for each frame processed. From initial testing with several machine learning object detection models (R-CNN, YOLOV8, etc) and measured the time it took for each frame to be processed. In order to meet our latency requirements, we decided to opt out of machine learning for this project. This made the latency requirements easy to meet, but also added much more algorithmic complexity to the computer vision subsystems. Because our detection had to now be deterministic and done manually, we had to have several layers of masking, filtering, edge/contour detection, color detection to account for inconsistencies in our detection.

### *D. Open-source vs physics engine from scratch*

In developing the physics engine, a big tradeoff we made involved deciding between leveraging open source pool physics libraries and game engines versus constructing ours from scratch. Deciding to leverage existing work - whether it be from research papers, game engines, etc - would give us a well-tested, robust, and verified physics engine for our system; however, it would be possible that adapting it to our use case would be difficult. On the other hand, writing our physics engine from scratch would let us have full control over the functionality and let us keep it both lightweight and highly specialized for our system. However, this would mean getting deeply involved in the theoretical aspects of ball/wall collisions, spin physics, and changes in momentum. Ultimately, our team decided to write the physics engine from scratch. We initially decided to try to adapt existing game engines, research papers, and physics libraries for our use case. However, much of the existing work was too heavy for our system. They involved a lot of extra functionality that we did not need, and this made it more difficult to adapt them to our use case. The core of our engine focuses on trajectory prediction, and trying to account for other variables like mass or friction coefficient proved to be very time-consuming with little return on investment. Thus, we decided to develop our own custom physics engine from scratch, which we accomplished in 1-2 weeks.

### *E. Cue stick detection improvements*

This tradeoff involves the technical details of the cue stick detection subsystem, and what we did to improve both the stability and sensitivity of its detection. The biggest issue with the cue stick is being able to both detect subtle movements from the user moving the cue stick while also keeping the trajectory displayed stable if the user does not move it. Initially, our cue stick subsystem relied on detecting a small portion of the front of the cue stick. It detected the contours and approximated the front as a small rectangle. We then drew a line between the edges of the rectangle and used it as the trajectory of the cue stick. This method resulted in a fairly accurate trajectory line for the cue stick. However, at times it was unstable, and the trajectory would randomly jump significantly. To improve this, we had to add additional layers of computer vision algorithms to stabilize the stick's trajectory. The crucial difference was masking the pool table with a range of green in order to filter out noise. With just this change, the cue stick trajectory was much more stable than before. Then, we replaced the polygon approximation with contour detection, filtering for an area range, finding the minimum enclosing rectangle around the cue stick. We then took the midpoints of the short sides of the rectangle, and took the connecting line as the stick's trajectory. With these changes, we achieved a good balance between stability and sensitivity for the cue stick. These improvements did come at a cost - the latency increased, and the sensitivity decreased compared to the previous method. However, this was an acceptable tradeoff - because of our other decisions, we had a good amount of room to work with regarding latency; there was also a decrease in sensitivity, but not by an amount that would significantly affect the user experience negatively.

### *F. Handling ball spin*

Ball spin and its implementation was both another tradeoff and important design decision we made. We decided between two options to incorporate spin into our system. The first method was incorporating ball spin by mounting an inertial measurement unit (IMU) onto the cue stick and collecting gyroscope data from it. This would then be streamed real-time to the physics engine which would incorporate spin by detecting the angle at which the cue stick strikes the cue ball (z-axis). The second method was delegating spin input to the user. They would have an option on the web application to select the part of the cue ball they were going to strike. This input would be sent to the physics engine, which would then account for the desired spin and change the predicted trajectory accordingly.

The first method involves very little input from the user - they would just have to aim at the cue ball and strike. The second method, however, involves the user a little more at the cost of a slightly clunkier user experience. This decision primarily trades off technical complexity for user experience quality. Ultimately, our team decided to go with the second method. Although the user experience is slightly clunkier, it results in a more accurate implementation of spin. We initially tried to implement the first method, but we ran into an issue with the IMU data drifting, especially with the gyroscope. After a few minutes, the gyroscope drifts by several dozens of degrees. The data becomes extremely unreliable, thus, we decided to forgo the IMU altogether. Implementing the first method with spin would result in a completely inaccurate trajectory, so we decided to implement spin with the second method. The user experience is a bit clunkier, but it is fundamentally more accurate - which is more important at the end of the day.

### *G. Calibration*

Calibrating our entire system was a minor part of our project, yet had significant impact. The crucial tradeoff with the calibration was increasing software complexity and setup time in exchange for consistent, lasting detections as well as an insightful optimization. Our calibration subsystem is run before the actual application starts. It takes a set number of frames in the beginning, runs our wall and pocket detection algorithms, then sets permanent state for the duration of the application's run. Previously, we had been trying to run the wall and pocket detection algorithms for each frame, but we realized that this was both unnecessary and wasteful. The only part of the game that changes are the balls and cue stick; the pool table walls and pockets stay constant. As such, we can pre-compute the detected walls and pockets, and fix them in the same positions for the application's entire run. This not only reduced our latency, but also resulted in more stable detections. Previously, when we ran the wall and pocket detection algorithms on each frame, other objects (like moving balls, cue stick) would interfere in the detection and produce inaccurate results. By fixing them from the start for the entire run, we were able to avoid such errors. Furthermore, since we now only compute the walls and pockets once, we can leave it out of processing each frame, making it significantly faster.

## VI. System Implementation

Our overall system consists of six different subsystems: An initial calibration system, ball detection subsystem, cue stick detection subsystem, physics shot calculation subsystem, a web application, and a projector subsystem.

### *A. Initial Calibration Subsystem*

The initialization of our system includes a crucial 5-second calibration step aimed at ensuring the accuracy of subsequent wall and pocket detection for shot calculations later on. Our WallDetection module has its primary 'getWalls()' function that facilitates the extraction of valid wall data. Initially, an image is loaded via a specified file path, undergoing a series of transformations which include green colored masking (cv2.bitwise_and with the image and mask) followed by edge detection (cv2.Canny). We subsequently use heuristic algorithms to classify these lines into top, bottom, left, and right walls based on their spatial relationships within the image and their angular orientation. Conversion from polar to rectangular coordinates further refines this data for subsequent processing and analysis. Additionally, the PocketDetection module, through the implementation of the 'getPockets()' function employs cv2.HoughCircles() with custom parameters to detect potential pockets. These detections are then validated against previously identified walls to ensure adherence to size and distance criteria, thereby confirming their classification as pockets. Acknowledging the variability inherent in individual frame detections, our system employs a strategy of aggregating multiple frame detections to establish the most frequent and accurate values for both walls and pockets, thereby facilitating precise calibration of the walls and pockets. Figure 3 shows an example of the calibrated walls and pockets. The green lines represent the detected walls, and the blue circle represents the location of the detected pockets.

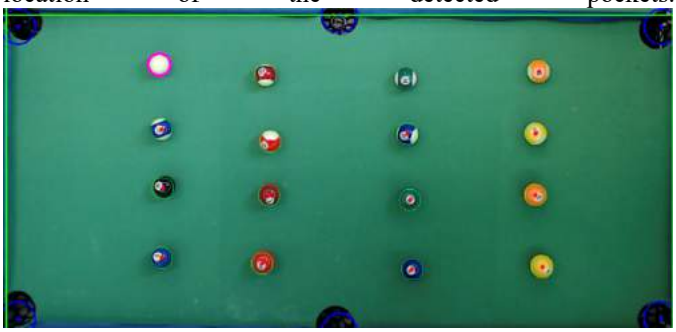


Fig. 3. Ball detections and outlines drawn.

### *B. Cue Stick Detection Subsystem*

The cue stick detection subsystem is an essential component of our pool analysis framework, serving to accurately identify and track the movement of the cue stick during gameplay. This subsystem begins by applying green colored masking to isolate the cue stick from the background, enhancing subsequent detection accuracy. Following this, Gaussian blur is applied to reduce noise and refine the edges of the masked image, facilitating more precise contour detection of the cue stick. Through contour detection, the subsystem identifies the outline of the cue stick, laying the foundation for precise bounding and classification. As the balls have not been filtered out of the picture, we cross check the contour locations with the ball locations, as well as use shape and radius size detections to differentiate the cue stick from the cue ball. Subsequently, we use the 'minEnclosingRectangle()' function to create a bounding rectangle around the detected contour, effectively encapsulating the cue stick within a defined region of interest. Within this rectangle, two points are drawn based on the shorter sides of the rectangle, serving as reference markers for subsequent classification. Through a distance-based classification process, one point is identified as the cue tip,

while the other is classified as the opposite end of the cue stick. This ensures accurate tracking and analysis of cue stick movements. Figure 4 shows an example of the cue stick being detected in the picture.

Fig. 4. Cue stick detection shown on the image.

### C. *Shot Calculation Subsystem*

Our shot calculation system is integrated within the Physics class that utilizes OpenCV and NumPy libraries, serving as a fundamental component for predicting ball trajectories and collision points on the pool table. At its core, the system comprises a function to detect ball collisions, a wall reflection function to calculate reflections off table walls, and a pocket collision function for identifying the points of intersections with pockets. The ball collision function calculates the precise point of the collision between two balls on the table. Collision is determined to occur if the distance between the centers of the cue ball and another ball approximates twice the radius of the balls. Leveraging quadratic equations, this function computes the intersection points of the cue ball trajectory with other balls, selecting the closest point as the collision point. Next, the wall reflection function accounts for reflection physics and ball cushion dynamics, implementing rigorous equations that model reflected ball shots, including factors such as angle of incidence and ball cushion interaction. This allows users to execute strategic bank shots with precision. Lastly, the pocket collision function detects instances where the trajectory line intersects with a pocket, thereby halting the trace of the line. By accurately identifying these points of intersection, the system provides insights into shot outcomes, when the ball enters the pocket. The run_physics() function traces a trajectory line starting from the cue stick, which serves as an initial projection for the cue ball's movement. Subsequently, if this trajectory intersects with any balls or walls on the table, the ball and wall collision functions are executed to calculate the ensuing trajectory adjustments. This iterative process continues until the trajectory line intersects with a pocket, or reaches its pre-coded maximum line length, at which point the line is halted, signifying the completion of the shot prediction process. It is also worth noting that the trajectory line only appears if the cue tip is brought close enough to the cue ball and doesn't work when it is brought close to any other ball. Figure 5 shows the example of the trajectory line being projected onto the table.

Fig. 5. Calculated trajectory presented on the table.

### D. *Web Application subsystem Subsystem*

The Web Application subsystem represents a user-friendly interface that enables users to see a livestream of the video footage, simultaneously observing the real-time detection of crucial game elements including walls, pockets, balls, cue stick, and predicted trajectories. This feature-rich functionality allows users to have comprehensive insights into gameplay dynamics, aiding strategic decision-making and enhancing gameplay experience. Additionally, users can select their intended spin directly on the website. Upon confirming the selection, updated spin trajectories are dynamically generated and displayed, providing users with immediate feedback on the potential outcomes of their shots. The front end of the application was coded using React, ensuring a responsive and intuitive user interface. The backend employs Flask to handle video streaming and perform complex computations necessary for real-time detection and trajectory prediction. This combination of technologies allowed us to deliver a seamless and immersive user experience. Figure 6 shows the completed web application.

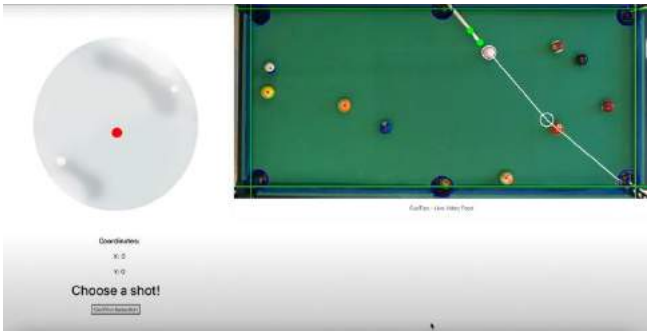



Fig. 6. Web application.

### E. *Projector Subsystem*

The projector subsystem consists of both a hardware and software component for visualizing predicted trajectories in our trajectory prediction system. The hardware component consists of a camera, shelf mount, and a projector. These components work together to capture the original frame of the pool table, process it, and project the predicted trajectories onto the table surface. The software component is the final stage of the trajectory prediction process. It receives the original frame processed by preceding subsystems, along with a python array consisting of a list of vectors representing the points of

collisions. The software first creates an equal-dimension black background using 'np.zeros_like', providing a canvas for trajectory prediction. It then iterates through the list of vectors, plotting them as white lines onto the black background. This approach ensures maximum contrast, guaranteeing clear visibility of the trajectory lines even in well-lit environments.

## VII. Testing, Verification and Validation

We employed four testing methods to evaluate the performance of our product and validate the accuracy of our shot calculations. We ran multiple tests to ensure we satisfied the three metrics that are central to our use case requirements: 1) latency, 2) trajectory prediction accuracy, and 3) object detection accuracy.

### *A. Methodology for Testing Object Detection Accuracy*

Our first use case requirement specifies a target metric to detect pool balls within 0.2 inches from their actual locations. In order to verify this design requirement, we employed a test that involved executing our object detection model on a set of carefully selected test suites. Four different test suites were devised which covered the following scenarios: 1) normal balls on the table, 2) balls adjacent to each other, 3) balls close to walls, and 4) balls close to pockets. These different test suites were designed to ensure that we comprehensively tested our system's ability to accurately detect pool balls across various table configurations and positions. We manually set up five different randomized ball layouts for each of the four test suites in order to have a larger sample size of test cases, which would increase the reliability of our measurements. Figure 7 below shows the test suites that were used for ball detection.

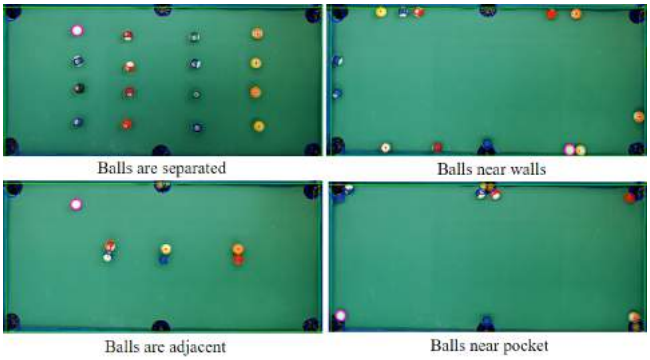



Fig. 7. Test suites we used to test ball detection accuracy.

After the object detection model had been run on each table setting with a unique ball layout, we proceeded to take the measurement of the ball detection error. The process of taking this measurement involved manually identifying the center and radius of each pool ball in the table setting, then comparing it with the corresponding center and radius returned by our ball detection model. Next, the Euclidean distance between the detected center and the true center of each ball in the setting was calculated. This distance is what we utilize as the detection error of our ball detection model.

### *B. Methodology for Testing Trajectory Accuracy*

To validate the accuracy of shot calculations, two distinct tests were performed. One measured the success rate of the trajectory accuracy, while the other sought to measure the average error of our predicted trajectories.

The first test we performed sought to calculate the success rate of our shot calculations. To do this, we had a user execute 20 shots using our system, then calculated the percentage of shots out of these attempts that were successfully pocketed. For each shot, the user was prompted to aim a target ball into a pocket while following the trajectory guidance provided by our system. We chose to test three different types of shots in order to evaluate our product's efficacy in different gameplay scenarios. For each type of shot, the user begins by aiming the cue stick at the cue ball, and the objective is to pocket a target ball. These shots included normal shots, bank shots, and kiss shots. We define a normal shot to be a scenario in which the user aims the cue ball to hit another target ball directly. Bank shots are when the user aims the cue ball to bounce off the wall prior to the cue ball colliding with a target ball. Lastly, a kiss shot is where there are cascading collisions involving two target balls. The cue ball is aimed toward a target ball, which will hit another target ball into a pocket. The sequence of collisions for each of these types of shots are detailed in Table I.

Table I. Collision Sequences For Each Shot Type

| **Shot Type** | **Sequence of Collisions** |
|---|---|
| Normal Shot | Cue Ball - Target Ball - Pocket |
| Bank Shot | Cue Ball - Wall - Target Ball - Pocket |
| Kiss Shot | Cue Ball - Target Ball 1 - Target Ball 2 - Pocket |

The notation "A-B" signifies that a collision occurs that starts from object A and goes on to object B.

The next test we performed sought to measure the average error of our predicted trajectories. This test was meant to validate our use-case requirement of having a predicted shot trajectory accuracy of less than 2 degrees. This angle is a measure between the trajectory line predicted by our system and the actual trajectory line followed by the target ball. In order to take this measurement, we aimed a cue ball to hit a target ball to the wall and compared the coordinate of the target ball's actual collision location with the location predicted by our model. We first took note of the starting coordinate of the target ball. We then took a shot to hit the target ball towards the wall. Next, we obtained the predicted coordinate of collision by running our physics model stopping it when it predicts a wall collision. Finally, to get the true coordinate of collision, we programmed a separate test bench that was designed to detect when a target ball collides with a wall. When this collision occurs, the program would stop itself and print out the coordinate of the target ball upon its collision with the wall. After the starting point and the two collision points were taken, we measured the angle between the two paths. This measured

angle is what we utilize as the shot calculation error of our trajectory prediction. This method allowed us to quantitatively determine the accuracy of our trajectory prediction.


Fig. 8. Different types of shots executed to test trajectory accuracy.

### C. *Methodology for Testing Latency*

Finally, to test the end-to-end latency of our project, we timed the code execution for each frame programmatically. As soon as a frame comes in from the camera via OpenCV's *imread* function, we include with the frame the marked input time as additional metadata. When the frame finishes processing, we again take the time when the predictions are generated and find the difference (in milliseconds) between the starting time and the ending time. Since the software system is entirely contained within the laptop, we do not need to account for transmission latency via Wi-Fi. Our use case requirement regarding latency was that end-to-end processing would be within 100ms. By ensuring that each individual frame takes less than 100ms to be processed through the entire software pipeline, we ensure that the user perceives changes to the projected predictions within 100ms which has the appearance of the trajectory outputs being instantaneous.

### D. *Results for Object Detection Accuracy*

The data we collected from our object detection accuracy tests showed that we were able to exceed our target metric. We took an average of all the errors we had sampled in our tests and found that the average error in our ball detections was only 0.05 inches. This measure is much lower than our original target error of at most 0.2 inches. These accurate results can be attributed to the use of color masking and detection averaging, both of which were additional design iterations we implemented on top of our principal ball detection model. With this method, we were able to generate accurate and stable object detections, as portrayed by the low error we have measured.

### E. *Results for Trajectory Accuracy*

For our trajectory accuracy test evaluating the success rate of our predictions, we were able to achieve satisfactory results for each of the different types of shots. Our use case requirement called for an accuracy rate of 95% for all types of shots. We were able to achieve this target metric for normal shots and come close for bank shots and kiss shots. Our trajectory calculation system was most accurate for normal shots, with a success rate of 100%. The success rate for kiss shots was 90%, and for bank shots it was 75%. The lower success rate in kiss shots and bank shots can be attributed to greater losses in energy as a result of a greater number of collisions. We addressed this issue by iterating upon our design to take into account the cushioning of the walls and losses due to friction in order to minimize the inaccuracies present in these types of shots.

### F. *Results for Latency*

The results we collected from timing our code showed that our system has an average latency per frame processing of 22ms. This is a statistic that far exceeds our expected target latency of 100ms. We were able to achieve such high performing results by solely using OpenCV for image processing. A tradeoff that was made in order to achieve this low latency was to forgo the use of machine learning.

TABLE II. SUMMARY OF TEST RESULTS

| **Use Case Requirement** | **Specification** | **Performance** |
|---|---|---|
| Ball Detection [PASS] | Detect pool balls within 0.2in from actual ball location | Overall Average Error: 0.05in |
| Shot Calculation [PARTIAL] | Predicted trajectory must be 95% accurate | Normal shots: 100%<br>Kiss shots: 90%<br>Bank shots: 75% |
| | Predicted trajectory must be within 2° of actual trajectory | Error of 1.13° on average |
| Latency [PASS] | Achieve latency of less than 100ms | 22ms on average (far exceeded expectations) |

## VIII. Project Management

### A. Schedule

Our schedule for the project is depicted in Fig. 9 on page 14 of this report. The blocks in blue represent the sections that are being done by Andrew, green represents the sections being done by Tjun Jet, and the red block represents the sections being done by Debrina. The yellow blocks indicate the blocks that will be done by everyone. Our schedule includes weekly milestones and allows us to keep track of what tasks should be done by a given week.

Throughout the semester, our schedule underwent changes involving the expected completion date of certain tasks. Tasks related to trajectory output on the projector were delayed as we put our focus to first creating a robust system backend in the early stages of our project. Tasks involving testing, verification, and error checking tended to be prolonged. We decided to extend the timeline for these tasks as we realized that there were many more iterations of testing that needed to be done during the integration stages. Lastly, tasks related to spin implementation and the web application were also delayed and prolonged. Spin proved to require more complex implementations; hence we needed more time to complete this. Furthermore, since the spin selection step was executed through our web application, we also extended our timeline for the web application development to work on it in parallel with the spin implementation. Changes to our schedule have been made in yellow and are shown in Fig. 9 on page 14.

### B. Team Member Responsibilities

Debrina's primary responsibility is the Computer Vision subsystem. She is responsible for implementing object detection and segmentation algorithms. The objects that will have to be clearly segmented are the array of pool balls on the table, distinguishing between the cue ball and the target balls, cue pockets, and the four walls surrounding the table. She must ensure the detections are accurate enough to meet our use case requirements. Debrina's secondary responsibility is supporting in the implementation of the physics model to conduct tests and feature improvements as further requirements emerge throughout the development process.

Tjun Jet's primary role is to work on the physics engine. The physics engine takes in the outputs from Debrina's computer vision model and calculates the trajectory that the cue ball will follow if it's being hit in a certain manner. He will focus on two main detection models - wall detection and ball collision mechanics. He must consider the reflections of the wall and the mechanics when the cue ball collides with the ball. Tjun Jet's secondary role is to support the spin implementation by conducting research on the physics behind spin and assisting in its implementation.

Andrew will play a primary role in the projector subsystem and the spin subsystem, web application development, and cue stick detection. He must ensure that the cue stick detection is accurate and stable to minimize flickering in the trajectory predictions. Furthermore, he is in charge of implementing the spin mechanics and integrating this to the web application to allow users to interface with our system.

### C. Bill of Materials and Budget

Our bill of materials and budget can be found in Table I on page 10 of this report. At the end of the semester, we no longer needed the IMU since we determined that it did not contribute much value to our system due to frequent inaccuracies it faced. Since we no longer used the IMU, we no longer needed the ESP32 and the 9V batteries that were meant to be used with the IMU.

### D. Risk Management

To handle our project risks from the standpoint of scheduling, we initially left plenty of buffer time in the last few weeks of the semester. We started off with a plan that would allow us to complete the project a few weeks early. This would give us plenty of time to spare in case we encountered roadblocks that would lead to delays in our schedule.

From the standpoint of resources, since our project did not require many expensive components, we felt comfortable in allocating more of our budget to make purchases that would ensure the quality of our components. We prioritized selecting items that had good quality (as opposed to selecting the most affordable option) to avoid having to repurchase a component due to it falling short of expectations. An example of this is our selection of the shelving unit to hold the pool table and projector. Even though the shelf was on the pricier end, it was the most sturdy option available and fit our use case very well.

We handled risks related to design by developing a well-defined API early on to ensure that the interfacing between our different subsystems remained consistent. This allowed for tolerance in design changes since we could easily adapt to them. Each subsystem would be able to continue to run correctly as long as we ensure strict invariants related to the inputs received by each subsystem.

At the start of the semester, we anticipated that one risk we might encounter is that the latency of our detection becomes very high, making the user interface not as good as we want it to be, as the output trajectories will be too slow. We addressed this risk by structuring our implementation to rely only on OpenCV's object detection library. Our risk reduction measure was to choose not to use any machine learning in our system as this would lead to slowdowns.

When risks did develop into issues, we turned to some risk mitigation strategies. A notable instance involves our cue stick detection subsystem. At the start of the semester we had identified detection inaccuracy to be a potential risk. We anticipated that inaccurate object detections, primarily cue stick detection, would lead to incorrect trajectory calculations. While we initially thought to use April Tags to address this risk, we decided to move away from April Tags as we learned that it did not allow for a smooth user interface. We initially used April Tags to detect the cue stick by mounting it onto the cue stick. However, this restricted the user's ability to orient the cue stick in any way they want. Hence, we decided to experiment with different approaches using OpenCV to effectively detect and isolate the cue stick.

## IX. Ethical Issues

There are a few ethical issues that may relate to our product. To name a few: structural integrity of the camera/projector mount, privacy concerns with camera usage, accessibility for sight-impaired users, and skin tones affecting computer vision subsystems.

A major potential ethical issue health-wise has to do with the structural integrity of the mount that the camera and projector is fixed on. Our team opted for a heavily modified metal shelf frame to mount our equipment; the shelf is made of metal, and both the camera and projector are placed a decent height above where the player would be playing. If the structure were to collapse or be damaged, the user could be seriously injured by the equipment mounted on top or the metal from the shelf itself. Everyone who uses this product would be affected by this issue. To mitigate this, we could rigorously test and perform structural analysis to ensure that the structure does not collapse, and that it can withstand collisions, shaking, general wear-and-tear, etc.

Another concern would be users' concerns about our system's camera usage. The camera mounted on top could lead our users to believe that we are collecting data about them, or storing the frame data we collect of them when using our product. We do not store any data, and once a frame is processed and trajectory predictions are shown, it is gone. However, many users could still hold suspicions, and this is a concern that can affect all users. To mitigate this, we can open-source our code and put disclaimers in the web application that we do not store frame data. By doing this, those concerned can verify rigorously that we do not store camera footage.

Accessibility is yet another potential concern for sight-impaired users of our product. From the beginning, our product was designed with visual feedback in mind; however, sight-impaired users would not be able to use our product in its current state since the trajectory predictions are visual - projected on top of the pool table. To mitigate this, we could extend our system to give audio feedback. We could implement another computer vision subsystem that guides the user to where the cue ball is, or how to move the stick to aim.

Lastly, skin tones is another ethical concern that may affect our project. Since we utilize color detection in our system, users of different skin tones may see varying results. For instance, our cue ball detection algorithms rely on using HoughCircles to identify circular objects with roughly a set radius and comparing the percentage of white (a range of white) pixels to all other pixels. If someone with a very pale skin tone were to use the system, it is possible that our ball detection system would confuse their hand or fist as the cue ball. This difference in usability depending on the user's skin tone is a big ethical consideration to be addressed. To mitigate this issue, we could change the functionality of the system. One such solution would be to show trajectory predictions for any ball the cue stick is pointed at - doing so would eliminate the need to identify the cue ball from the rest via color detection. Another solution would be to do some sort of contour counting or area filtering, making sure that "hand-like" features are disqualified from being cue ball candidates.

## X. Related Work

The most similar project to ours is a 18-500 capstone project last year - S23 Team C7's "8-ball lifeguard" [3]. The project was designed for beginners to learn how to play pool. It does this by providing the user with the most optimal shot to take given the current pool table state. They take in the game state via a camera, compute the best shot to take, then output it onto the pool table via a projector. This project is similar to ours in functionality and system/hardware. However, the use case requirements and software systems are very different. We wanted to heavily prioritize the responsiveness and interactivity of the system by allowing the user more control over the shot, helping them build intuition by experimenting with various shots and seeing the predicted trajectories change in real-time.

## XI. Summary

Our system was able to meet the design specifications. We set out to create an assistive product that helps people learn how to play pool more effectively. Our design specifications involved three components: low latency, accurate object/game state detection, and accurate trajectory predictions. For latency, our target metric was 100ms, but we far exceeded expectations by reducing the end-to-end latency to 22ms on average. For game state detection, we wanted to distinguish all different objects in the game (cue stick, balls, pockets, walls) with 100% accuracy, which we were able to achieve. Furthermore, as a quantitative metric, we aimed for <0.2in error in our ball detection; our project met this metric with 0.05in of error. Lastly, for trajectory prediction accuracy, our target was 2 degrees of error from the predicted line to the actual shot line. Our system also met this metric, with 1.13 degrees of error from testing various shots.

From this project, we learned much about addressing the technical challenges within our project. However, more importantly, there were several high-level lessons learned that are applicable to all engineering projects: 1) users first, 2) allocate time for the unexpected, 3) build to have it work - not perfect. One of the most important lessons we learned is to understand that engineering does not exist in a vacuum; we are always building for an end user. It is very easy to get wrapped up in the technical details and make design decisions that make sense technically, but produce a poorer user experience. For instance, when coming up with our feedback implementation, we initially thought of just displaying the trajectories via our web application. This would be the most straightforward way to display the feedback, as we would only need to stream the video via Flask to our React frontend. We realized, however, that it would be a very poor user experience to have to constantly look back and forth between web application and pool table to take each shot. Ultimately, we decided to have the projector display the predicted trajectories on top of the pool table. Many people who used our system far preferred this display to the web application's due to its intuitive interface and responsiveness.

The second lesson we learned was to allocate time for the unexpected. Throughout our project, we often ran into technical issues that we had not accounted for. As a result, the development of some components took much longer than expected. Some subsystems with a timeline of 1-2 weeks took as long as 3-4 weeks to fully complete. If we were to redo the project again, we would try to allocate extra time for building the subsystems that we did not have a full implementation plan for. These often were the places where we ran into unexpected technical issues that cost more time to resolve.

Lastly, we learned that it is more important to have things work rather than to make it perfect. Our team had 12-13 weeks to complete this project, and oftentimes we had to move fast at the expense of code quality or cleanliness. Spending extra time over-optimizing each function or formatting/documenting everything could result in not being able to finish other critical tasks. Throughout the duration of our project, we employed hacky shortcuts that enabled us to move faster to complete the actual critical parts of our system. For instance, initially we tried to integrate auto-cropping into our calibration subsystem; it was annoying to keep having to crop our video every time the camera moved. However, we realized that building this functionality out would not contribute meaningfully to the actual user experience - it was just a quality-of-life improvement for our own sake. It would take at least a day or two to develop, and we decided instead to keep a file with all the different camera crops we used, labeled each, and just remembered which to use. We didn't have to build the cropping system, and we also saved 10 minutes each work session not having to manually recalibrate the crop every time.

Regarding future work, we do not plan to continue this project after this semester, that some of us either pursuing graduate school or entering the industry. CueTips was a great experience for all of us in terms of embracing the technical challenges in engineering, working collaboratively as a team, and most importantly, allowing us to also experience the process of learning the game of pool.

## XII. Glossary of Acronyms.

IMU – Inertial Measurement Unit
FPGA – Field-Programmable Gate Array
GHz – Gigahertz
MHz – Megahertz
CV – computer vision
R-CNN – Region-based Convolutional Neural Network
YOLO – You Only Look Once
RGB – Red, Green, Blue
USB – Universal Serial Bus
ms – milliseconds

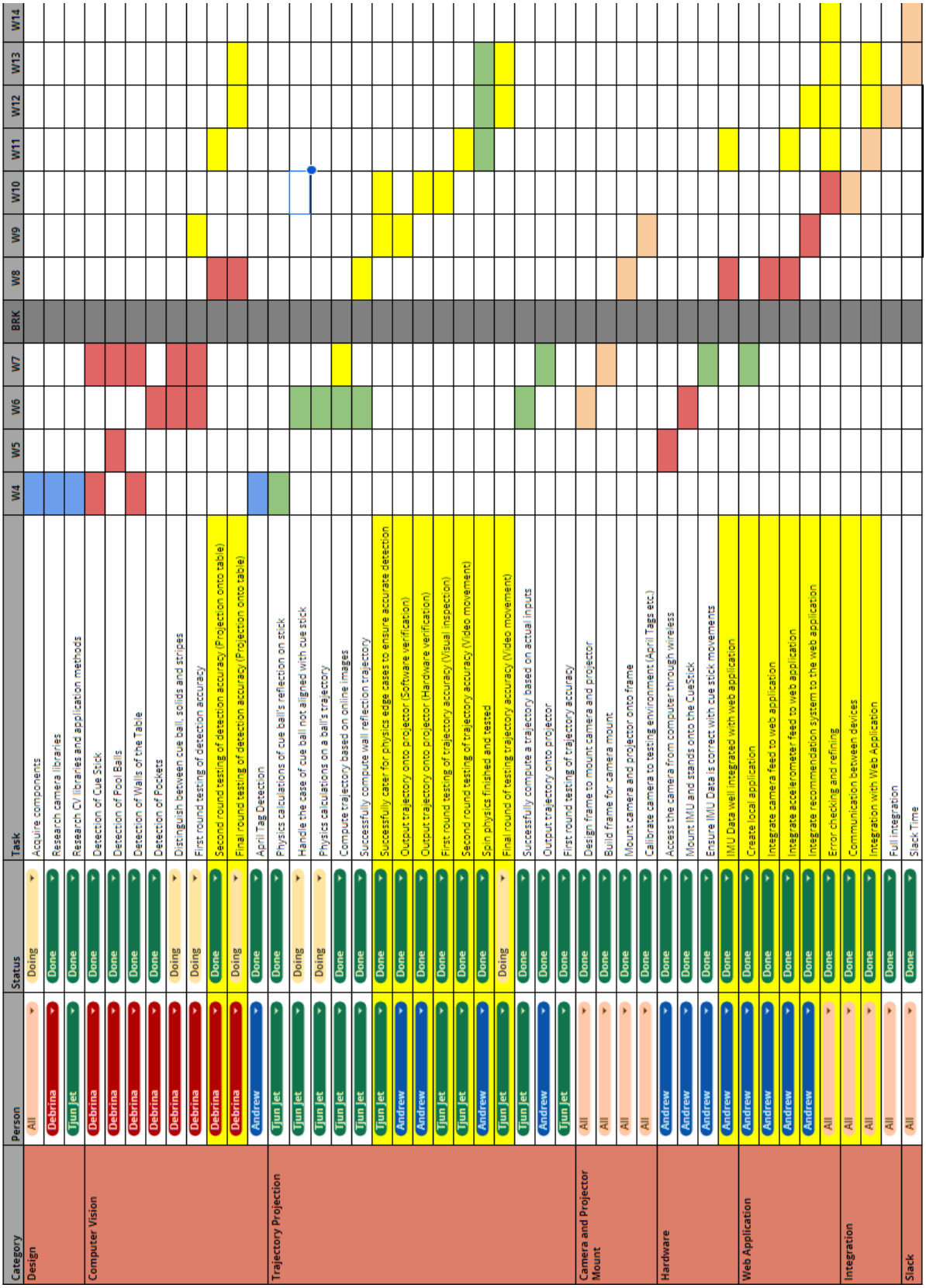
## XIII. References

[1] R. B. Miller, "Response time in man-computer conversational transactions," in Proc. AFIPS Fall Joint Computer Conference, vol. 33, pp. 267-277, 1968.
[2] AZBilliards Forums, "Fractional Aiming and Required Accuracy," Available: https://forums.azbilliards.com/threads/fractional-aiming-and-required-accuracy.522183/.
[3] Agarwal, Rager, Ray "Team C7: 8-Ball Lifeguard" Carnegie Mellon University. Available: https://course.ece.cmu.edu/~ece500/projects/s23-teamc7/.

TABLE III. BILL OF MATERIALS

| Item | Part Name | Manufacturer | Quantity | Cost @ | Total |
|---|---|---|---|---|---|
| Pool Table | RayChee Portable Mini Billiard Table | RayChee | 1 | $129.99 | $129.99 |
| Rack | Muscle Rack 5-Shelf Steel Freestanding Shelving Unit, Black | Muscle Rack | 1 | $109.00 | $109.00 |
| WiFi-enabled Microcontroller | ESP-WROOM-32 ESP32 ESP-32S Development Board | AITRIP | 1 | $15.99 | $15.99 |
| Inertial Measurement Unit | Adafruit 9-DOF Absolute Orientation IMU Fusion Breakout - BNO055 | Bosch | 1 | $34.95 | $34.95 |
| Laptop | Macbook Air | Apple | 1 | $0 | $0 |
| Camera | Logitech C922 Pro Stream | Logitech | 1 | $0 | $0 |
| Battery | 9V Batteries | Amazon Basics | 1 | $0 | $0 |
| Projector | VOPLLS 1080P Full HD Mini Projector | VOPLLS | 1 | $49.99 | $49.99 |
| LED Light Strips | Tenmiro 65.6ft Led Strip Lights | Tenmiro | 1 | $9.99 | $9.99 |
| | | | | **Grand Total** | **$349.91** |

FIG. 9. PROJECT SCHEDULE



| Category | Person | Status | Task | W4 | W5 | W6 | W7 | BRK | W8 | W9 | W10 | W11 | W12 | W13 | W14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Design | All | Doing | Acquire components | | | | | | | | | | | | |
| | Debrina | Done | Research camera libraries | | | | | | | | | | | | |
| | Tjun Jet | Done | Research CV libraries and application methods | | | | | | | | | | | | |
| Computer Vision | Debrina | Done | Detection of Cue Stick | | | | | | | | | | | | |
| | Debrina | Done | Detection of Pool Balls | | | | | | | | | | | | |
| | Debrina | Done | Detection of Walls of the Table | | | | | | | | | | | | |
| | Debrina | Done | Detection of Pockets | | | | | | | | | | | | |
| | Debrina | Doing | Distinguish between cue ball, solids and stripes | | | | | | | | | | | | |
| | Debrina | Doing | First round testing of detection accuracy | | | | | | | | | | | | |
| | Debrina | Done | Second round testing of detection accuracy (Projection onto table) | | | | | | | | | | | | |
| | Debrina | Doing | Final round testing of detection accuracy (Projection onto table) | | | | | | | | | | | | |
| | Andrew | Done | April Tag Detection | | | | | | | | | | | | |
| Trajectory Projection | Tjun Jet | Done | Physics calculations of cue ball's reflection on stick | | | | | | | | | | | | |
| | Tjun Jet | Doing | Handle the case of cue ball not aligned with cue stick | | | | | | | | | | | | |
| | Tjun Jet | Doing | Physics calculations on a ball's trajectory | | | | | | | | | | | | |
| | Tjun Jet | Done | Compute trajectory based on online images | | | | | | | | | | | | |
| | Tjun Jet | Done | Successfully compute wall reflection trajectory | | | | | | | | | | | | |
| | Tjun Jet | Done | Successfully cater for physics edge cases to ensure accurate detection | | | | | | | | | | | | |
| | Andrew | Done | Output trajectory onto projector (Software verification) | | | | | | | | | | | | |
| | Andrew | Done | Output trajectory onto projector (Hardware verification) | | | | | | | | | | | | |
| | Tjun Jet | Done | First round testing of trajectory accuracy (Visual inspection) | | | | | | | | | | | | |
| | Tjun Jet | Done | Second round testing of trajectory accuracy (Video movement) | | | | | | | | | | | | |
| | Andrew | Done | Spin physics finished and tested | | | | | | | | | | | | |
| | Tjun Jet | Doing | Final round of testing trajectory accuracy (Video movement) | | | | | | | | | | | | |
| | Tjun Jet | Done | Successfully compute a trajectory based on actual inputs | | | | | | | | | | | | |
| | Andrew | Done | Output trajectory onto projector | | | | | | | | | | | | |
| | Tjun Jet | Done | First round testing of trajectory accuracy | | | | | | | | | | | | |
| Camera and Projector Mount | All | Done | Design frame to mount camera and projector | | | | | | | | | | | | |
| | All | Done | Build frame for camera mount | | | | | | | | | | | | |
| | All | Done | Mount camera and projector onto frame | | | | | | | | | | | | |
| | All | Done | Calibrate camera to testing environment (April Tags etc.) | | | | | | | | | | | | |
| Hardware | Andrew | Done | Access the camera from computer through wireless | | | | | | | | | | | | |
| | Andrew | Done | Mount IMU and stands onto the CueStick | | | | | | | | | | | | |
| | Andrew | Done | Ensure IMU Data is correct with cue stick movements | | | | | | | | | | | | |
| | Andrew | Done | IMU Data well integrated with web application | | | | | | | | | | | | |
| Web Application | Andrew | Done | Create local application | | | | | | | | | | | | |
| | Andrew | Done | Integrate camera feed to web application | | | | | | | | | | | | |
| | Andrew | Done | Integrate accelerometer feed to web application | | | | | | | | | | | | |
| | Andrew | Done | Integrate recommendation system to the web application | | | | | | | | | | | | |
| | All | Done | Error checking and refining | | | | | | | | | | | | |
| Integration | All | Done | Communication between devices | | | | | | | | | | | | |
| | All | Done | Integration with Web Application | | | | | | | | | | | | |
| | All | Done | Full integration | | | | | | | | | | | | |
| Slack | All | Done | Slack Time | | | | | | | | | | | | |