Team B9: BreakTime

Augmented Reality Pool Guidance System

Christina Ou: Electrical and Computer Engineering, Carnegie Mellon University

Harry Xu: Electrical and Computer Engineering, Carnegie Mellon University

Samuel Kim: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A system that helps beginning pool players practice and improve. As the user lines up their shot at the table, computer vision is used to read ball positions, cue stick location, and cue stick speed. A software backend processes the data and predicts the outcome. Finally, a GUI is projected back onto the table for a user to see.**

***Index Terms*— billiards (pool), camera, computer vision, geometry, image processing, physics, prediction, projector, PVC frame, user feedback**

I. Introduction

BreakTime is a practice aid for the game of pool. Shooting straight and visualizing the correct paths for the pool balls are important techniques to mastering pool. BreakTime is a system to help beginners develop these two critical skills: aim and stroke.

Our system is friendly for casual players because our system is inexpensive, user-friendly, and real-time. Other competing solutions do not achieve these characteristics. Competing solutions include: hiring a professional coach (expensive), inventing our own pool table with embedded technology (not scalable), or displaying feedback on a separate monitor (poor user experience).

BreakTime's goal is to be an accurate training tool that is user-friendly for beginners. To achieve this goal, we summarize our system requirements as follows:

- Functionality: 50% improvement in shot-making ability
- Accuracy: 2° margin of error from intended to resultant shot
- Performance: 1 second end-to-end latency

Functionality will be measured with a 9-shot skills test (Figure 1), where users will be measured by how many more shots they can make with BreakTime. Accuracy will be measured by how many degrees, at most, the object ball will deviate from the predicted line output by our system. Performance will be measured by the total time a user must wait for visual feedback, from the time they lay the cue stick on the table to the time that feedback is projected onto the pool table.

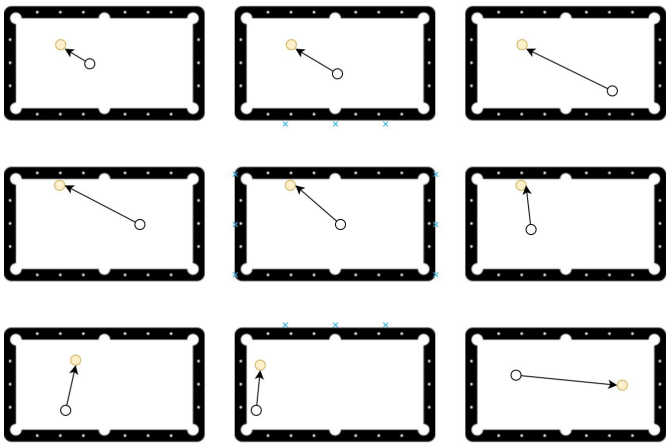

Figure 1: User skills test for testing system functionality. With our system, users should be able to make more of these shots.

II. Design Requirements

In order to achieve our goal of creating a tool that will help beginning pool players improve their skills, we must define design requirements for our system. These requirements are derived from the user perspective and what will most benefit the user while using our product. We have previously stated our overall requirements, and these are geared towards our stakeholder: the user. To help fulfill these overall system requirements, we have defined quantitative requirements for each of the components in our system.

Our system must have an accurate perception of the real game state; we are utilizing computer vision for this, and so we must ensure that our computational intake of the current game state is accurate and able to handle real-time plays. Without accurate ball location data, it becomes difficult for the software to give a viable shot. In addition, image processing can be time-consuming. In order to adhere to our overall system performance requirement, we will optimize our image size and algorithms as much as possible. Below are the requirements for our computer vision component:

- Accuracy: 1 cm distance between the actual and perceived pool ball and cue stick locations
- Performance: 500 ms object detection latency

Our software backend will compute the ball path projections based on the positions of balls and how fast the user is moving the cue stick. The outputs of the software will be projected onto the pool table to help players visualize shots. This projection determines how the player shoots their ball and is thus the most important user-facing part of the system. Inaccuracies here will lead to missed shots, regardless of a player's skill level. In accordance with our overall

performance, we will minimize latency. As such, these are our software backend requirements:

- Accuracy: visual suggestions projected onto the table are accurate within 5 mm
- Prediction outcomes: at most 2 degrees deviation between predicted and actual paths
- Performance: 500 ms latency for computing predictions

Hardware in our project encapsulates the different physical objects in our system setup. A camera is used to take in the table game state, and our projector will display our computational results back onto the table. As these items process the input and output of information, they need to be accurate to support computer vision and our backend. Some distortion can be accounted for on the software side. Our frame will be holding our camera and projector. As one of the physically bulkier additions to our project, it must be unobtrusive to the player and remain stable during a game of pool. We present the following requirements for our hardware pieces.

- Camera:
  - Field of view covers 100% of pool table
  - Captures at least 1 image per second
  - Color contrast and resolution supports distinguishing physical objects
  - Supports the colors of 9 pool balls
- Projector:
  - Projected image covers 100% of pool table
  - Image has enough brightness and contrast to be visible on table surface
- Frame:
  - 90% of pool shots are achievable even with frame setup
  - Body-weight shifts on pool table setup do not affect projector and camera position

III. Architecture and/or Principle of Operation

Our system consists of the physical components, the Computer Vision, and the Software Backend/Prediction & Feedback subsystems. These components work together to parse the player action and provide accurate and timely response to the player. We delve into more detail on each of the components in sections IV and V.

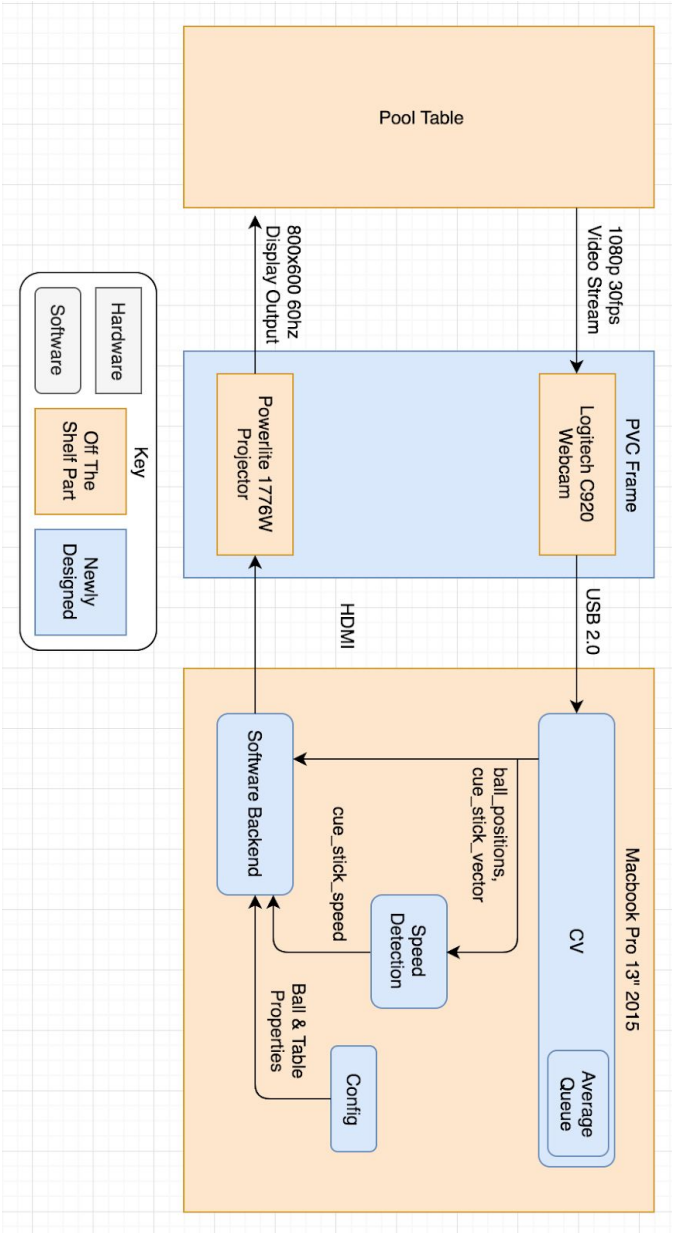

Figure 2: Our block diagram describes the high level architecture of our system and the interfaces between our parts.

IV. Design Trade Studies

A. *Physical components*

In order to assemble a testing configuration, we had to purchase and acquire a table, camera, projector, and construct a frame to hold our components together. For each component, we considered alternatives and chose the best option.

i. *Pool table*

For purchasing a pool table to use, we measured the ball size, the pocket size, and the table's diagonal length. We wanted a table that allowed for a large margin of error such that our system has some flexibility and the player can make easier long distance shots. Our motivation for this was to

accomodate for small deviations in a player's form as they practiced on a small demo table as we are developing a training tool.

We were able to model this problem with the cosine law geometry formula. With the hardest and farthest shot on the table, we determine how far off in inches the ball can deviate from the center of the pocket. From this, we compute the angle margin of error, which is the number of degrees a player's shot can be off by and still make the shot. It is important to note that across the 20", 40", and 96" tables that there are differing ratios of ball size to pocket size to table length, so the margin of error does not scale proportionately to table length. Ultimately, we chose the 40" table due to its large margin of error.

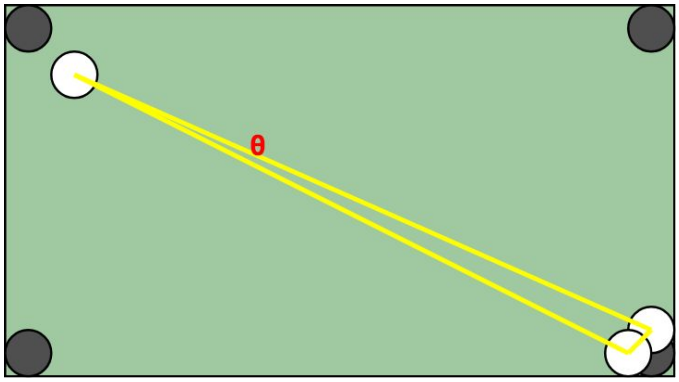

Figure 3: Angle margin of error visualization

$$Shot\ deviation = pocket\_size - ball\_diameter$$

	Pocket size (inch)	Ball diameter (inch)	Shot deviation (inch)
20" table	1.6	1	.6
40" table	3	1.5	1.5
96" table	5	2.125	2.875

$$Margin\ of\ error = \cos^{-1}\left(\frac{table\_length^2 + table\_length^2 - shot\_deviation^2}{2 * table\_length * table\_length}\right)$$

	Shot deviation (inch)	**Table diagonal length (inch)**	**Margin of error (°)**
20" table	.6	21.65	1.59°
40" table	1.5	37.5	2.29°
96" table	2.875	100.438	1.64°

ii. Camera

For the camera, we initially chose to use a 1080p 30fps Logitech c615. This camera was well within budget compared to better performing cameras. We had tested our Computer Vision subsystem with test photos taken with a 1080p cell phone camera. These test photos were found to be within our system requirements of 0.5cm.

However, we soon noticed that the Logitech camera output was much blurrier than our cell phone's camera, and this was due to a misunderstanding of camera quality specifications. We were relying on the video mode and selecting 1080p our cell phone also shot images with 1080p. Instead, we should have looked more into the megapixel count, which helps determine the resolution of our resulting image. Our original Logitech C615 had 8 MP, and we eventually used a Logitech C920 that has 15 MP and was similar to our cell phone camera specifications. The images from the C920 were much sharper and more viable for our CV input.

In addition, the C920 supported videos at a rate of 30 frames per second, which falls well within our initial requirement of 1s of latency. For this camera to become the bottleneck, we would need our latency to be less than 33ms.

iii. Projector

For the projector, we tested the Hunt library projector, the Epson VS250, in the pool table of the UC basement. We held the projector 6' above the table and projected a color matrix with additional light from a flashlight. We measured for brightness, resolution, and table coverage. We found that brightness was sufficient, resolution was well within our 0.5cm requirement, and the projection covered the whole table. We expect that at a smaller scale, the same projector can deliver comparable brightness, resolution, and coverage. Thus, the Epson Powerlite 1776W available in the ECE lab - which is superior in every way - is more than sufficient for our requirements. It surpasses the VS250 in every metric (the 1776W is a $1100 model while the VS250 is a $300 model) and thus meets our requirements.

iv. Frame

For the frame to hold our parts together, we chose to prototype a PVC frame as opposed to another material such as wood, specifically 1.5" Schedule 40 PVC. 1.5" Schedule 40 PVC is shown to have a tensile strength of nearly 1000 pounds for 8" of pipe[1]. From a structural standpoint, this number can assure us that an overhead frame constructed of this PVC will not bend under weight. One advantage of PVC over wood is the speed of prototyping and ease of assembly of PVC. Pairing

[1] https://www.pvcfittingsonline.com/resource-center/strength-of-pvc-pipe-with-strength-chart/

that with its affordability, we choose PVC as the material and construction of our frame. Additionally, we were considering an overhead tent frame versus an overhanging crane design. We chose to go with the crane design due to the minimal obstruction to the player, and the ability for the crane design to be mounted separately from the table. Ultimately, we unable to fully prevent any vibration or movement of the frame due to the ground mounted nature (if a player knocks into the frame no change of material will protect against that), and we will implement redundancies for re-calibration into our software.

v. Computer

To run our hardware, we will be using a 2015 Macbook Pro with a 2 core 2.7 ghz processor. OpenCV and the rest of our computation utilize only the processor. With this hardware, we are currently able to achieve around 34ms of latency for our Computer Vision subsystem, and 20ms of latency for our Software Prediction subsystem.

B. Computer Vision

In order to detect the different objects on our pool table, we decided to use computer vision techniques. We knew that our input would be from a camera, and CV's image processing and techniques are the best for giving our pipeline the location data needed. For our computer vision component, we made design decisions along the following areas: (i) technologies used, (ii) choosing our ball detection algorithm, (iii) stabilizing our detection results, and (iv) determining our cue stick detection algorithm.

i. Technologies Used

We are using the OpenCV 4.0 as our computer vision library. It is one of the most popular open-source image processing libraries with many functionalities for image processing and computer vision techniques. OpenCV is very computationally efficient, as it is built upon C/C++. At the same time, it supports fast development as there is a Python wrapper around the core C/C++ library.

ii. Choosing Ball Detection Algorithm

In our work of determining the best algorithm, we have considered two options: (1) HSV filtering + contour extraction and (2) edge detection.

HSV filtering and contour extraction utilizes the fact that all our pool ball objects have distinct colors and patterns. HSV stands for hue, saturation, and value. We can filter our image by a range of hues, or the shades of color. With an image filtered for only a certain color of ball, we can perform contour extraction to compute the minimum enclosing circle of a ball object. This is useful in averaging out the ball position and accounting for imperfect input data.

Edge detection finds the boundaries of objects by detecting discontinuities in the brightness of an image. There are varying filters that help determine the edges in an image such as the sobel filter. From this, we use the hough transform to extract circular contours from the image. In order to determine the color of each circular outline, we extracted 200 points from the contour of the ball to determine the color the ball was closest to.

After tuning both of these algorithms, we compared their accuracies to our ground-truth measurements of the pool ball locations. We used a 4-ball setup and ran both algorithms for 20 frames, capturing the CV's computation of each ball's location. In figure 4, we show the sum squared error of the difference of the physical and computed ball locations for both algorithms. HSV filtering outperforms edge detection in both the location difference and computational time. However, with the addition of our stabilization algorithm, HSV filtering and edge detection start to both perform well and at a similar level.

	Location Diff Values (cm)	Location Diff StdDev (cm)	Comp Time (ms)
HSV	0.097	0.0060	42
EDGE	2.507	0.0853	277
HSV - AVG Q	0.117	0.0006	41.2
EDGE - AVG Q	0.074	0.0121	278

Figure 4: CV Testing Results

iii. Stabilizing Detection Results

In order to stabilize and reduce variance in our ball detection results, we implemented an Average Queue (AVG Q) data structure that averages the past 10 ball locations. This greatly helped reduce variance as shown in figure 4 for both HSV filtering and edge detection algorithms.

iv. Determining Cue Stick Detection Algorithm

Determining the position and angle of the cue stick is vital for accurate feedback to the user. We experimented with several different methods of detecting the cue stick.

First, we used RGB filtering for the white, tan color of the cue stick. From this mask, we drew a line of best fit between the points in order to determine the angle. This method was not the best as the cue stick is not completely white and has wood patterns on it. Additionally, it did not allow us to have consistent points for the tip and middle of the cue stick.

Next, as HSV filtering worked well for pool ball detection, we decided to apply HSV filtering to the cue stick. We placed a single red tape on the tip of the cue stick, and this was well detected by our CV algorithm. Having a consistent relative

location for the tip of the cue stick was very useful for our speed module. We continued to use RGB filtering to detect any midpoint on the cue stick.

Lastly, we wanted to try detecting all parts of the cue stick (position, angle) through HSV filtering. We used red and blue tape for the tip and a midpoint on the cue stick, respectively. This led to detection and accuracy issues if a player's hand covered part of the middle blue point. Additionally, the midpoint on the blue point was not as accurate as the midpoint found through RGB filtering on the cue stick body. Thus, we decided to stick with our single red tape method.

C. Software Backend

For the software backend, several design decisions were made with project requirements in mind (e.g. timeframe, system requirements, user experience). These decisions include: (i) technologies used, (ii) simulation vs prediction, (iii) choice of pool game, (iv) choice of user feedback, and (v) user interface display.

i. Technologies Used

For the code, we decided to use Python 3.7.0 - the latest version of Python. The alternative was to use C++. C++ is a compiled language, so it is considerably faster than Python and is often used for programs that have high performance requirements. Despite these advantages of C++, we ended up choosing Python because of developer constraints -- it was the language everyone on the team felt comfortable with and Python also encourages more rapid development over C++, which we felt was suitable for the timeframe of this project. Additionally, we found our computations to run on the order-of-magnitude of nanoseconds -- that is to say, the performance improvement of C++ becomes unnecessary with the computations we're running.

ii. Simulation vs Prediction

A major design choice was deciding between (a) simulating the outcome of where *all* balls would end up, or (b) just computing the path of the cue ball and object ball.

Figure 5 illustrates these 2 options. Option A requires simulating all events after a ball is hit, waiting for equilibrium to be reached, and finally presenting the outcome to the user. Option B requires computing where the cue ball would strike the object ball, the line where the cue ball would deflect, and the line where the object ball would deflect.

Option A's advantage is that it fully predicts where all balls will end up for the user. However, the major drawback of Option A is its computational complexity (it requires a time step-by-step simulation). One of our main system requirements is low latency, so this prompted us to go with Option B over A.

Additionally, Option B is better than Option A for the purposes of helping a beginner train. Option A might be *too much* information for a beginner who is focused on getting one ball in at a time. Option B trains the user to focus on the 'ghost ball' technique[2] -- a very common aiming technique that many pool players use.

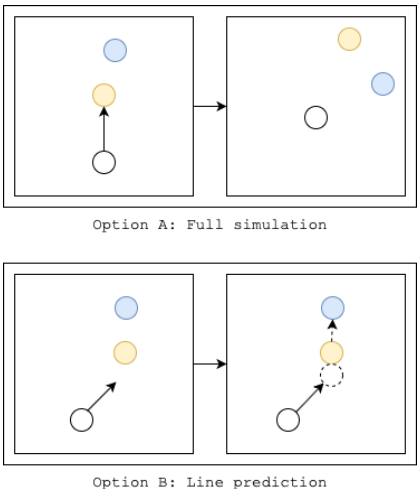

Figure 5: Simulation vs prediction

iii. Choice of pool game

Another assumption to be made before development was deciding which game would be played. The two most widely known games of pool are 8-ball and 9-ball. Although 8-ball might be the more popular game for beginners[3], we ultimately went with the decision of developing primarily for 9-ball to simplify the development process.

In 9-ball, there is only 1 object ball at any point in time (i.e. the player must hit the 1-ball if it is the lowest ball on the table). This is in contrast to 8-ball, where the user can have up to 7 balls available to hit. Figure 6 shows a 9-ball and 8-ball layout with all balls on the table. In 9-ball, the user must hit the solid yellow 1-ball. In 8-ball, if the user is solids (or stripes), they have the option to hit balls 1-7 (or 9-15).

By only having 1 object ball instead of 7, the 'feedback' portion of the software backend is greatly simplified and requires less computation (and, thus, less latency in our system).

---

[2] http://www.easypooltutor.com/articles/29-aiming-techniques-a-execution/31-how-to-aim.html

[3] As of 3/4/19, an Apple App Store search of 'pool' shows results for pool games that are all focused on 8-ball, not 9-ball

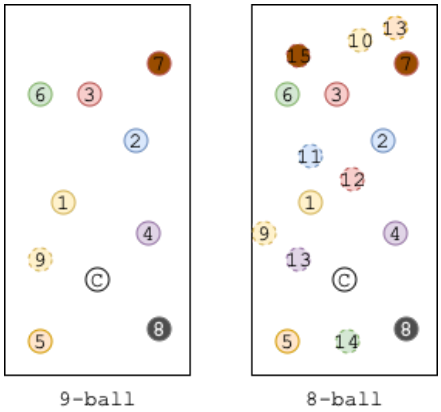

Figure 6: 9-ball vs 8-ball target ball options

iv. Choice of user feedback

In 9-ball, the goal of the game is to hit the current object ball into a pocket in such a way that the cue ball is positioned well for the next object ball. This is referred to as 'cue ball control'.

With the choice of the 40" table -- a toy version of an actual pool table, there is not enough ball mass or surface area for a player to apply 'english' or spin onto the cue ball, which makes 'cue ball control' come down to simply how much force is applied to the cue ball.

Figure 7 (below) shows how different speeds can affect positional play for the next shot. In the scenario shown, the yellow 1-ball is hit into the pocket, but must be hit at the correct speed to get into good position to hit the blue 2-ball. By taking spin out of the equation, the only parameter to work with is cue ball speed. Controlling cue ball speed is also an important skill for beginner pool players to develop[4].

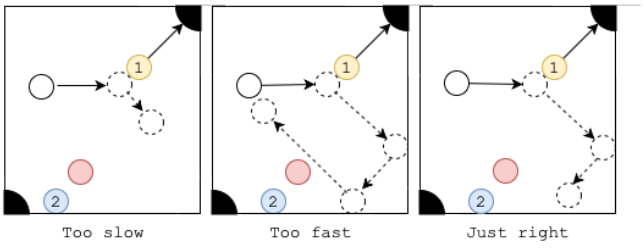

Figure 7: Cue ball speed for positional play

v. User Interface Display

We use PyGame as our graphics library, and the projection of the PyGame display onto the pool table is what the user sees. We initially visualized our pool balls as a solid ball color to emulate the actual pool ball. However, when pairing this projector output with CV, if the circles were not perfectly placed or if there were any lag, the CV algorithms would start detecting these projected circles as additional balls.

[4] https://billiards.colostate.edu/bd_articles/2004/nov04.pdf

To mitigate the side effects of the projector output on CV ball object detection, we experimented lowering the projector brightness, changing the PyGame ball colors, and only visualizing the ball outlines. Only outputting the circular ball outlines worked with the best with CV (Figure 8, right), and that is what we ultimately went with.

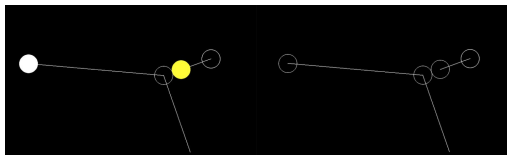

Figure 8: Different projector visualizations of pool ball locations

Lastly, we knew that were was some standard deviation in the CV perception of our pool balls, so we considered adding in a visualization of the standard deviation. We tested drawing a cone of where the the pool balls would end up if shot along a certain path (Figure 9). However, when projected, we noted that the cone was inhibiting to the user experience as it introduces more visual clutter.

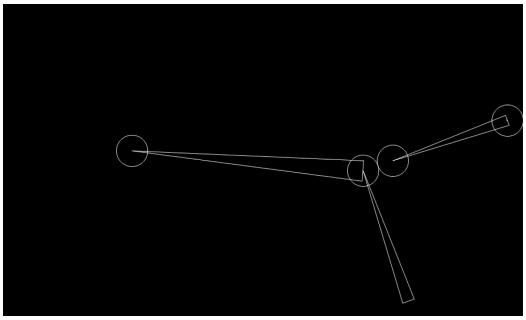

Figure 9: Path standard deviation visualization

V. System Description

A. Computer Vision

The Computer Vision subsystem is our machine perception of the game state. It is responsible for 4 major functions: (1) Receiving and preprocessing images from the camera, (2) Determining the locations of all pool balls on the table surface, (3) Determining the location and angle of the cue stick, (4) Computing the speed the player shoots the cue stick with.

i. Preprocessing camera input

We are taking input from a video camera and will need to process each frame. Depending on our computer vision accuracy, we may downsize the image for faster processing.

ii. Pool ball locations

There are 10 balls of various colors in a game of 9-ball pool. As the balls have distinct outlines, we can use edge detection and hough transforms to detect the balls. We find the edges of an image using a sobel filter. From this, we use the

hough transform to extract circular contours from the image. In order to determine the color of each circular outline, we compare 200 points from the contour of the ball to determine the color the ball is closest to. From the circular contour, we use the center point as our machine's perceived location of the pool ball.

We take these center points and add them into our Average Queue data structure. The average queue averages the past 10 locations of the ball, which greatly reduces the ball location variance.

iii. Cue stick location and angle

The cue stick is the user's main interaction with the game and our system. To detect the cue stick's positioning, we employ filtering techniques.

To detect the cue stick tip location, we filter for the red tape on the cue stick. Then, we perform some dilation on the red tape mask. Then, we draw a minimum enclosing circle on the red contour, and use the center point as the tip location.

For a midpoint on the cue stick, we employ RGB filtering. We perform image smoothing with dilation and erosion of the filtering mask. Instead of extracting contours, we compute the best fitting line out of the cue stick mask. We take the midpoint of the best fitting line to determine a middle point on the cue stick. With a tip and a midpoint, we can now compute a precise position and angle of the cue stick.

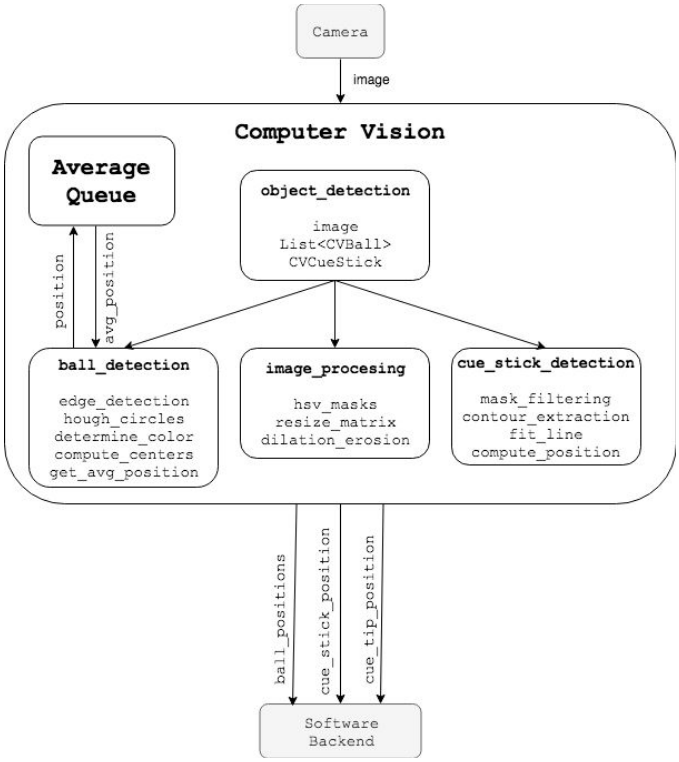

Figure 10: Computer vision subsystem architecture

B. Speed Detection

In our overall system, we wanted a couple features to aid user experience:

1) Continually showing the lines to a user *after* they hit the cue ball (so the user can compare predicted to actual result).
2) Showing the user how far their ball(s) would go if the user started doing practice strokes with the cue stick (a common practice in pool).

The Speed Detection module is a small module that helps achieve these 2 features. This module interfaces between the CV and software backend. It takes in every updated ball and cue stick position from the CV module. It keeps some state of the most recent positions of the balls and cue stick positions, and produces 2 outputs for the software backend: "cue_ball_moving" and "cue_stick_speed".

If the software backend receives "cue_ball_moving" to be True, it means the user has struck the ball, and the software backend should pause all computation so that the previously projected lines remain on the table for the user to visualize even after she struck the ball.

The software backend uses "cue_stick_speed" and predicts how *far* balls will travel for the given speed, using classic 2-D kinematics equations (e.g. $v_f = v_o^2 + 2ad$) to solve for distance traveled (if the ball came to a complete stop without colliding with another ball) or to solve for final speed (if the ball collided with another ball, d distance away).

C. Software Backend

The software backend subsystem is responsible for processing all data in the system. The entire subsystem is laid out in Figure 11. This subsystem is responsible for 4 major functions: (1) maintaining state of the pool game, (2) predicticting where the cue ball and object ball will go, (3) suggesting the best speed to hit the cue ball, and (4) outputting the computed information as a GUI. In this section, we discuss how these functions are implemented as separate modules.

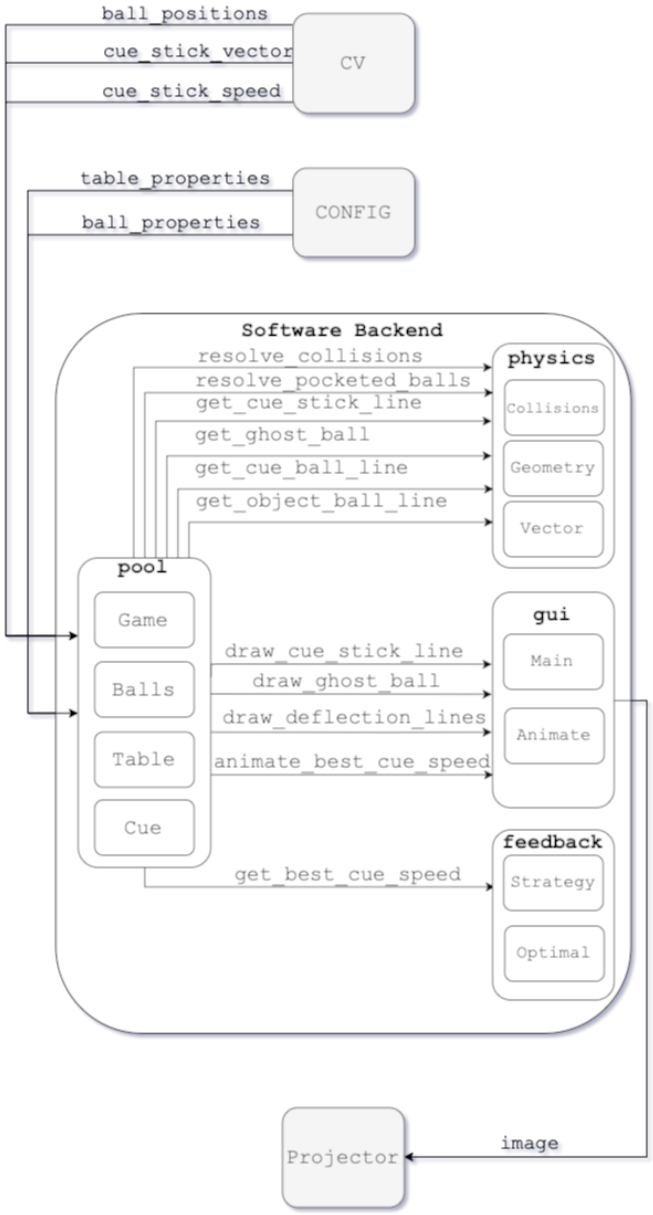

Figure 11: Software backend subsystem architecture

i. Maintaining game state (`pool` module)

The 'pool' module is static and straightforward - it simply holds the current state of the pool table at any particular point in time. Basic object-oriented principles were applied to model the real-world game of pool closely by creating classes: Game, Balls, Table, Cue. In an MVC framework, the 'pool' module would serve as the 'model'

ii. Computation and prediction (`physics` module)

As discussed in Section IV, most problems require the use of one or more equations. To provide support for these equations, the ''physics' module can be further broken down into the following classes: Collisions, Geometry, Vector. Each class holds utility equations to support the wide array of formulas needed to support ball path prediction. We now discuss how the game of real-life pool was modeled for this software implementation. In an MVC framework, the 'physics' module would serve as a 'controller'.

With some basic assumptions, the game of pool can be abstracted as a 2-D Cartesian plane and all interactions can be modeled as simple geometry problems. The following assumptions are currently being made:

- Ball-to-ball collisions can be treated as 2-D point collisions
- Friction between balls and table cloth is known and constant
- Pool table cushions can be treated as perfect walls in the case of ball-to-cushion collisions
- The cue ball will be struck with no spin

With these assumptions and abstractions in mind, most problems can be solved with one or more simple equations. Below are the most significant problems and how they were solved with one or more equations:

Detecting collisions:

- Finding distance between two points:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \leq r$$

- Assuming balls have the same radius

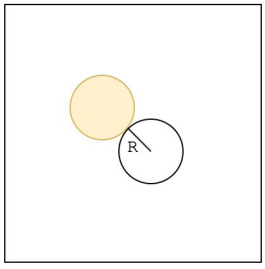

Figure 12: Ball collisions

Resolving collisions:

- Two-dimensional collisions with two moving objects[5]

$$v_1' = v_1 - \frac{2m_2}{m_1+m_2} \frac{\langle v_1 - v_2, x_1 - x_2 \rangle}{\|x_1 - x_2\|^2} (x_1 - x_2)$$

$$v_2' = v_2 - \frac{2m_1}{m_1+m_2} \frac{\langle v_2 - v_1, x_2 - x_1 \rangle}{\|x_2 - x_1\|^2} (x_2 - x_1)$$

- Where $< a, b >$ indicates a dot product of 2 vectors and $\|a\|^2$ is the magnitude of a vector.

[5] https://en.wikipedia.org/wiki/Elastic_collision#Two-dimensional_collision_with_two_moving_objects

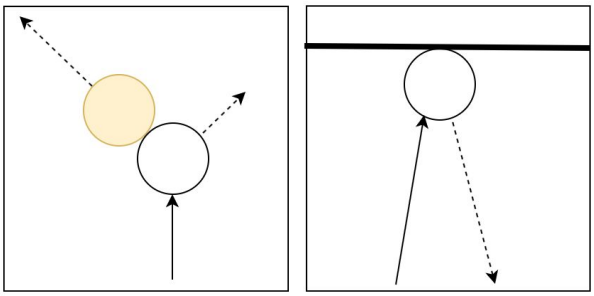

Figure 13: New velocities after collisions

Finding 'ghost ball':

- Finding intersection between line and circle:

For a line equation $y = mx + c$ and for a circle equation $(x-p)^2 + (y-q)^2 = r^2$.

Substitute the line equation into the circle equation:

$$(m^2+1)x^2 + 2(mc - mq - p)x + (q^2 - r^2 + p^2 - 2cq + c^2) = 0$$

Treating this equation as a quadratic of the form $Ax^2 + Bx + c = 0$,

- If $B^2 - 4AC < 0$, the line misses the circle
- Else if $B^2 - 4AC = 0$, the line is tangent to the circle
- Else if $B^2 - 4AC > 0$, the line intersects the circle at 2 points.

To get the intersection point(s):

$$x = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

$$y = m\left(\frac{-B \pm \sqrt{B^2 - 4AC}}{2A}\right) + c$$

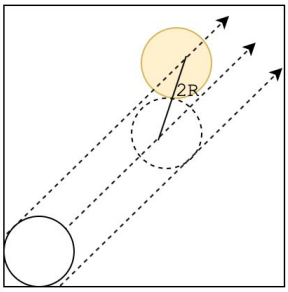

Figure 14: Finding cue ball contact point

iii. User suggestion (`feedback` module)

The 'feedback' module is a stretch goal (post-MVP) part of the system that provides suggestions and feedback to the user for how they should hit their shot. Currently, the following assumptions are being made (to reiterate what was stated in Section IV):

- The game being played is 9-ball
- The cue ball will be hit with stun (no follow or draw)[6]

As stated in Section IV, these assumptions imply a much simpler implementation of the 'feedback' module. In 9-ball, there is only 1 object ball every time (compared to 8-ball, where there can be up to 7 object balls). This means that the user's goal is to always hit 1 object ball into a pocket and setup position for the next, sequential object ball. (i.e. Hit the 1 ball in and setup position for the 2 ball).

Additionally, without follow or draw, we can assume the cue ball deflection path will follow the tangent-line rule[6], so the only remaining factor is the cue ball speed coming out of the collision.

From Section IV, it was established that cue stick speed is the best feedback to give back to a user. This module will work backwards to find the range of ideal speeds to strike the cue ball to end up in an ideal position for the next shot. Figure 15 (below), shows a range of acceptable cue ball positions as a large circle. This range can be found by seeing if the resultant cue ball position would be able to see enough of the object ball to hit the next object ball into a pocket.

Once this ideal position range is found, we can work backwards to find the best range of cue ball speeds.

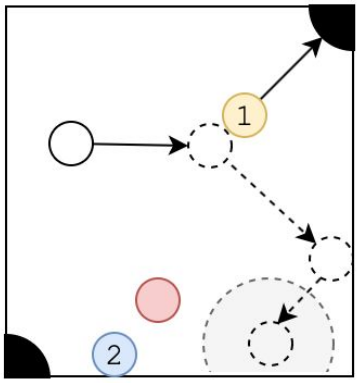

Figure 15: Start with an allowed range for position. Then, work backwards.

iv. Projector output (`gui` module)

The GUI is straightforward and outputs the computed data and results into a user-friendly format, to be projected back onto the pool table. Figure 16 (below) shows a proof-of-concept from a similar project. In an MVC framework, the 'gui' module would serve as the 'views'.

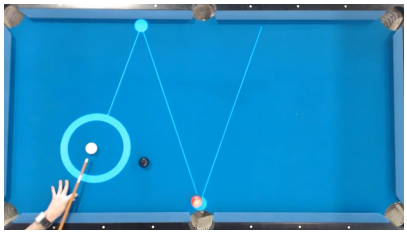

Figure 16: GUI proof of concept

[6] https://www.billiards.com/article/the-tangent-line

D. Hardware Components

The physical hardware components of the project consist of a camera, a projector, a pool table, a frame, and a computer.

The pool table is the central component that the user plays on. The camera takes in real-world data and this information will be sent to our computer's computer vision algorithms. The projector displays the output of our path prediction onto the pool table.

To hold our camera and projector, we are building a frame so that our input data is accurate and our output is stable for the user. The camera and projector are mounted above the center of the table, with the camera being above the exact center and the projector mounted besides. Our software accounts for this slight bias. The frame holds these components roughly three feet over our 40" demo table, which is a generous field of view for our purposes.

The computer takes raw data from the camera, runs CV, predicts ball paths and user feedback, and finally generates a GUI to be projected onto the table.

E. End-to-end Results

After our final product was constructed, we ran some tests with the entire system. The motivation was to determine what the end user experience would be like. Our testing process was as follows:

1. Place balls for the specific scenario
2. User lines up a shot with the cue stick
3. Mark where the system *predicts* where the ball will go
4. User actually hits the shot
5. Mark where the ball *actually* ended up
6. Compare the *prediction* from (3) to the *result* from (5)

For a clearer illustration, refer to Figure 17, where two of our testing scenarios are shown.

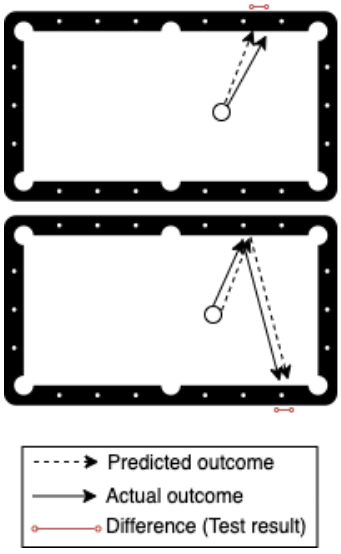

Figure 17: Example of end-to-end testing procedure

Our results were as follows:

Scenario	**Results (Δcm)**
cue ball → wall (15cm)	0.275cm
cue ball → wall (30cm)	0.340cm
cue ball → wall (15cm) → wall (50cm)	0.315cm
cue ball → ball (15cm) → wall (15cm)	0.353cm
cue ball → ball (15cm) → wall (30cm)	0.412cm
cue ball → ball (30cm) → wall (15cm)	0.456cm

These results were achieved by getting the average over 10 trials. Note that the last 3 scenarios refers to the cue ball striking an object ball, and measuring where the object ball ends up on the wall. The higher result deviations (i.e. > 0.4 cm) were expected due to longer distances and more human error. Overall, our results were not perfect, but we were satisfied with our results (See Summary section for further discussion).

VI. Project Management

A. Schedule

Our schedule separates our project into its main components, the Computer Vision, the Prediction and Feedback software, and our physical parts. In addition, we budget additional time for testing and integration. We do not budget for project documentation and other deliverables, instead we allocate generous deadlines as means of distributing the impact of these deliverables. We removed the robotics from our schedule and replaced it with more detailed hardware analysis. Our schedule is at the back of this document at Fig 14.

B. Team Member Responsibilities

As we have three main areas in our system, we have split our roles accordingly.

Christina worked on the computer vision subsystem and handled the integration between the camera and CV software. She worked with the team on CV tuning and accuracy adjustments as the data is passed into the backend.

Sam handled the software backend, working on maintaining the game data and computing path prediction. He also worked on developing all submodules for the software backend, taking care of the primary functionality, discussed further in Section V.

Harry oversaw our physical components and testing. He designed and built our frame and mounted our camera and projector over the table. He assisted Christina with development of computer vision assistance modules such as

the average queue module, and helped tune aspects of the CV such as ball detection.

Overall, all the team members worked together for integration and demo related tuning and development tasks.

C. Budget

Item	Amount	Description	Status
Pool table	62.22	40" Green Pool Table	Arrived
Projector	0	Borrowed from ECE Lab	Secured
MVP Frame	45.41	PVC Purchase from Home Depot	Arrived
Poster	0	Subsidized en masse	Secured
Camera	34.32	Logitech C615 Camera	Arrived
USB Extender	6.49	Extends the short camera cord	Arrived
Frame Extension	25.75	More PVC parts for the frame	Arrived
Camera 2	72.28	Camera with more megapixel accuracy	Arrived
15' HDMI Cable	11.77	HDMI Cable	Arrived
Camera Arm	21.18	Holds the camera	Arrived
Total	279.42		

D. Risk Management

One of our major risk elements throughout the system is accuracy and jitter. Accuracy is very important to the end user and there are multiple areas where accuracy is affected. Jitter causes our system to seem unstable, and is unusable to the end user. These errors compound, so we must minimize these errors as much as possible at each step of the system flow.

With computer vision, the ball positioning is essential for the software prediction module to present accurate feedback to the user. As computer vision is dealing with images, there can be inaccuracies in the image processing and eventual detection. To help the system best perceive the physical world locations, we are working with two computer vision methods of object detection. We tested two approaches, and chose the more accurate of the two. In addition, we implemented an aggregate bucket system to reduce random outliers and provide a stable and accurate ball position. This aggregate bucket system does not significantly increase latency, and helps increase both accuracy and reduce jitter. There still exists a potential for inaccuracies and jitter, and we try to reduce with that with tuning for edge detection thresholds and color ranges.

On the software backend side, a major concern is being able to model the real-world well enough to accurately predict where pool balls will go. We created a config file with physical properties of the pool table. We ran experiments where we rolled the ball and measured the time to stop. We were able to extract different coefficients for different regions of the pool table. Despite this, there is still the risk that the manufactured pool table and balls are imperfect and deviate from the expected as we are able to eliminate most of the major cases, but there still exists edge cases.

On the physical components side, there may be issues with the camera input, pixelation, and environment lighting. The camera and projector alignment and orthogonality to the table surface may also affect the measurements. Because the frame holds both the camera and projector, its stability is essential for our other components to receive and output visually accurate data. We have aligned the pool table to the frame to reduce variance, and fixed the camera and projector at marked intervals.

Ultimately, our end-to-end system will inevitably suffer from accuracy and jitter issues. We reduced these risks through accurate tuning and experimental calculations.

VII. Related Work

'This AR Projector System Acts Like A Billiards Coach': The following video showcases a similar project that utilizes computer vision to project the expected path of balls onto a pool table. This is our motivation, to expand on the capabilities and features of such a system.
https://www.youtube.com/watch?v=zHVW2_lH9vs

Pool Live AR:
This product is an augmented reality system that is our motivation to provide an interactive and engaging environment for the user. It uses a strong projector system to provide an augmented reality.
http://www.poollivear.com

8 Ball Pool (Apple App Store):
An interactive game that showcases lines from a top down view. This is our motivation behind the display for our system.
http://itunes.apple.com/us/app/8-ball-pool/id543186831?mt=8

VIII. Summary

We are working methodically and efficiently towards meeting our design specifications. As we are in the intermediate phases of our implementation, we are placing priority on functional correctness as this impacts a player's

training the most. Once we achieve the desired functionality as stated in our requirements, then we will proceed to improve system performance for a better user experience.

A. Fulfillment of Requirements

In terms of our computer vision requirements, we fully fulfilled our goals. For our final demo, we used the Edge Detection method with Average Queue.

Requirement	**Result**
Accuracy: 1 cm distance between the actual and perceived pool ball and cue stick locations	0.074 cm
Performance: 500 ms object detection latency	278 ms, per frame

For our software backend requirements, we achieved:

Requirement	**Result**
Accuracy: visual suggestions projected onto the table are accurate within 5 mm	3.585 mm, on average
Prediction outcomes: at most 2 degrees deviation between predicted and actual paths	2.2 degree deviation, on average
Performance: 500 ms latency for computing predictions	74ms, per iteration

For our end to end requirements, we achieved:

Requirement	**Result**
Functionality: 50% improvement in shot-making ability	40%
Accuracy: 2° margin of error from intended to resultant shot	2.2 degree deviation, on average
Performance: 1 second end-to-end latency	393 ms, camera to projector output

Overall, we were satisfied with our results. We were able to satisfy a majority of our requirements, and work around the requirements that were not met exactly.

B. Future work

To improve our project's extensibility to different settings, we would like to implement an auto-calibration system. This would be helpful in reducing our tuning time as we test our system in different lighting conditions and environments.

Additionally, we would like to configure our system for a standard-size pool table. With some upgrades to our hardware components (e.g. better camera, larger frame), many of our system components should scale up to a large pool table.

C. Lessons Learned

We learned many lessons throughout the different phases of our project. In the beginning during our concept phase, we could have explored more project ideas. We followed our interests and areas of expertise, but it would have been useful to explore more unfamiliar domains.

As we were refining our project idea, we could have done a better job matching our project scope with the course expectations. We were overly ambitious in wanting to include a robotics component to our project, so understanding our abilities and time constraints would have helped us formulate an appropriate scope.

For the realization and implementation of our project, we ran into many issues. Sometimes, we were fixing these issues at the wrong step in our pipeline, whereas we should have been fixing compounding errors at the root. For example, when the projector projects an incorrect pool ball location, instead of fixing it on the software or computer vision side, there may have been an inaccuracy with the camera placement.

Additionally, we ended up spending a lot of time tuning our system for different lighting conditions and environments. It would have been worthwhile to invest making an auto-calibration system to speed up development time.

When people build projects, they place a lot of emphasis on the building phase and tend to push risks aside. Evaluating these beforehand allows us to be more aware of our risk factors and create backup plans.

If our system were to be put into production, we really want to make sure that we are factoring the user experience into account; the system needs to be usable and offer practical value. The main barrier between our system and production at the moment is the need to tune for inconsistent lighting and the replicability of our frame setup.

For scheduling, we realized that finer granularity would be helpful. Adequate planning helps us determine our schedule and allows us more room to be flexible.

References

[1] Billiards and Pool Physics Resources from Colorado State University: https://billiards.colostate.edu/physics/
[2] Rules of 9-ball pool: http://www.vnea.com/111111new-page.aspx
[3] Cue ball follow effect (stun, draw, follow): https://billiards.colostate.edu/FAQ/stun/90-degree-rule/
[4] Ball Tracking with OpenCV: https://www.pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/
[5] OpenCV Contours: https://docs.opencv.org/3.4.2/d4/d73/tutorial_py_contours_begin.html

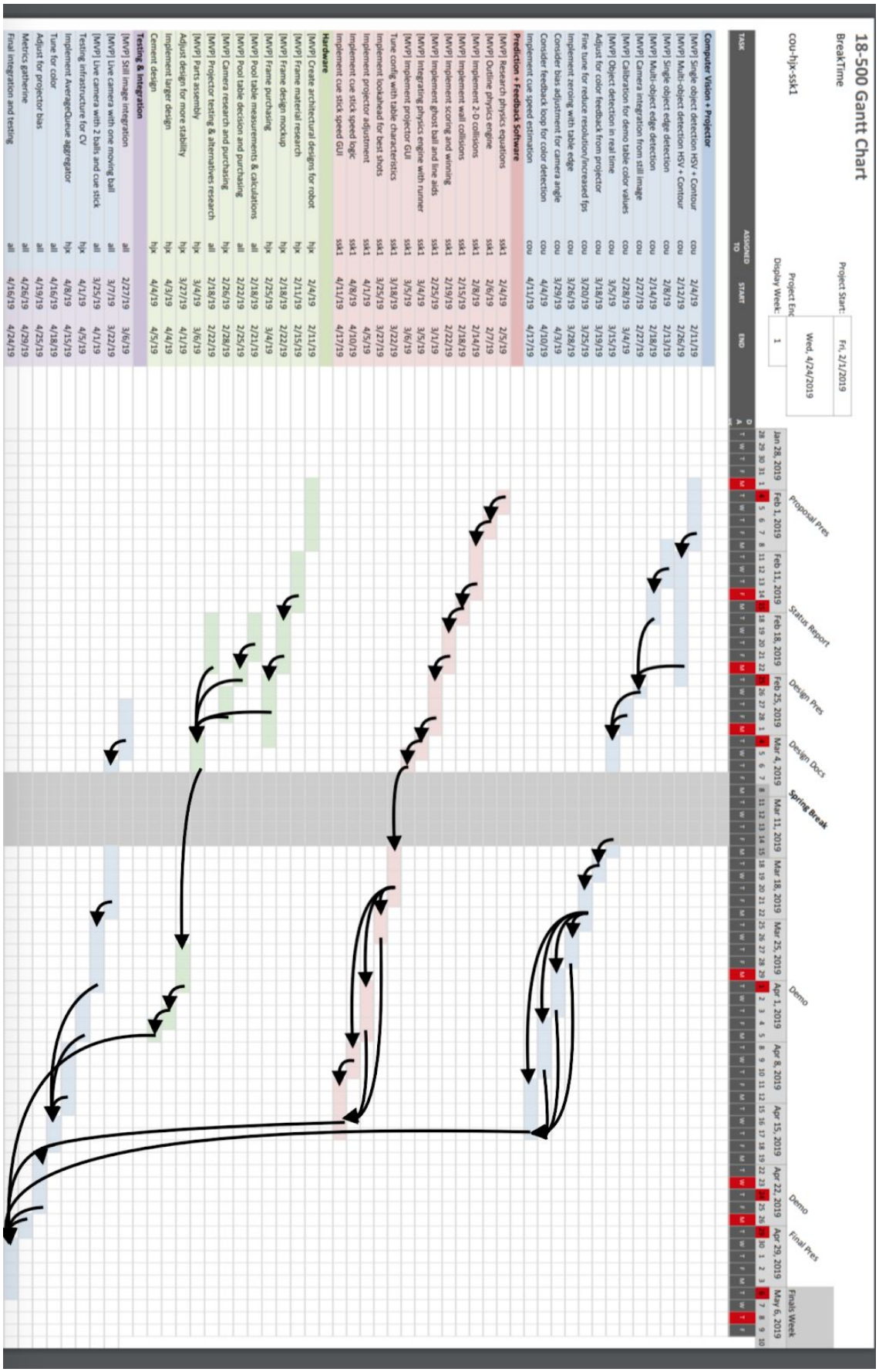

Fig. 18 Gantt Chart of Current Progress