Matt Basile and Zareen Choudhury
6.111 Final Project: "La PC-na"
Project Proposal

1 Introduction

The luxury of purchasing separate pool tables, foosball tables, and air hockey tables is beyond the budget of many, particularly college students. We propose a more cost effective solution that could synthesize a multitude tabletop entertainment options into a single system. Our project is based on the concept of projecting virtual versions of tabletop games to provide a combination of virtual and physical entertainment experience.

In our project, we will implement one of these classic tabletop games -- pool -- by using physical cues but virtual balls and pockets. The pool table, including the pockets, balls, and rack, will be projected onto a surface. Players will use physical cues to "hit" the virtual balls, and the projected game will update real-time to mimic the physics of pool table collisions.

2 Design

The virtual pool table, balls, and pockets will be displayed on a TV screen that is oriented horizontally on a tabletop. A camera with an infrared (IR) filter will be mounted with a view of the TV screen to capture real-world motion. The physical cues will be outfitted to emit IR light so that the camera can track them. An accelerometer and gyroscope will also be attached to each cue in order to determine its orientation and speed of motion. A high level diagram of our design is displayed in Figure 2.1.

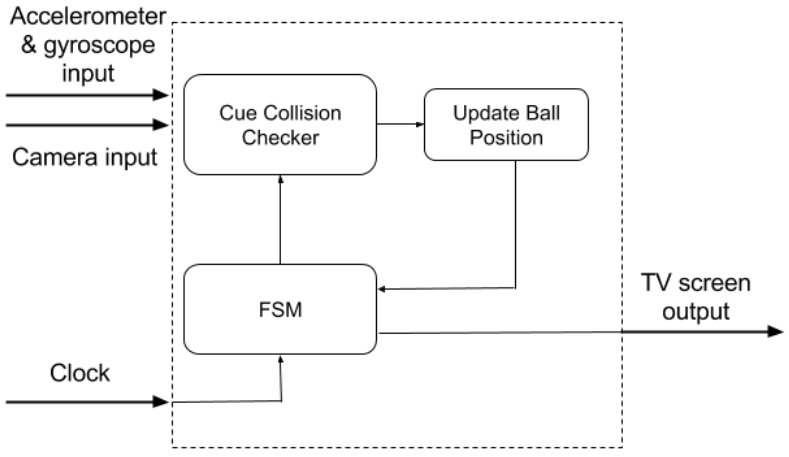

Figure 2.1: High level diagram of system.

During a given player's turn, if the player's cue comes in contact with a virtual ball, this event will be detect by a cue collision checker module that interprets the camera stream and the processed accelerometer & gyroscope data. Internal modules will then update the positions of

the balls based on the angle and speed of the cue collision and any collisions between balls. As the ball positions are updated in real time, they will be outputted over VGA on the TV screen.

3 Implementation & Block Diagram

We divided our implementation into external components and internal components. External components include setting up IR tracking, camera processor module, IMU processor module, and cue collision checker module. Internal components include the game's finite state machine (FSM), modules for updating ball positions, and a pocket checker module. Zareen Choudhury will work on external components, and Matt Basile will work on internal components. Each module is described in greater detail in the block diagram (Figure 3.1) and the subsections below.

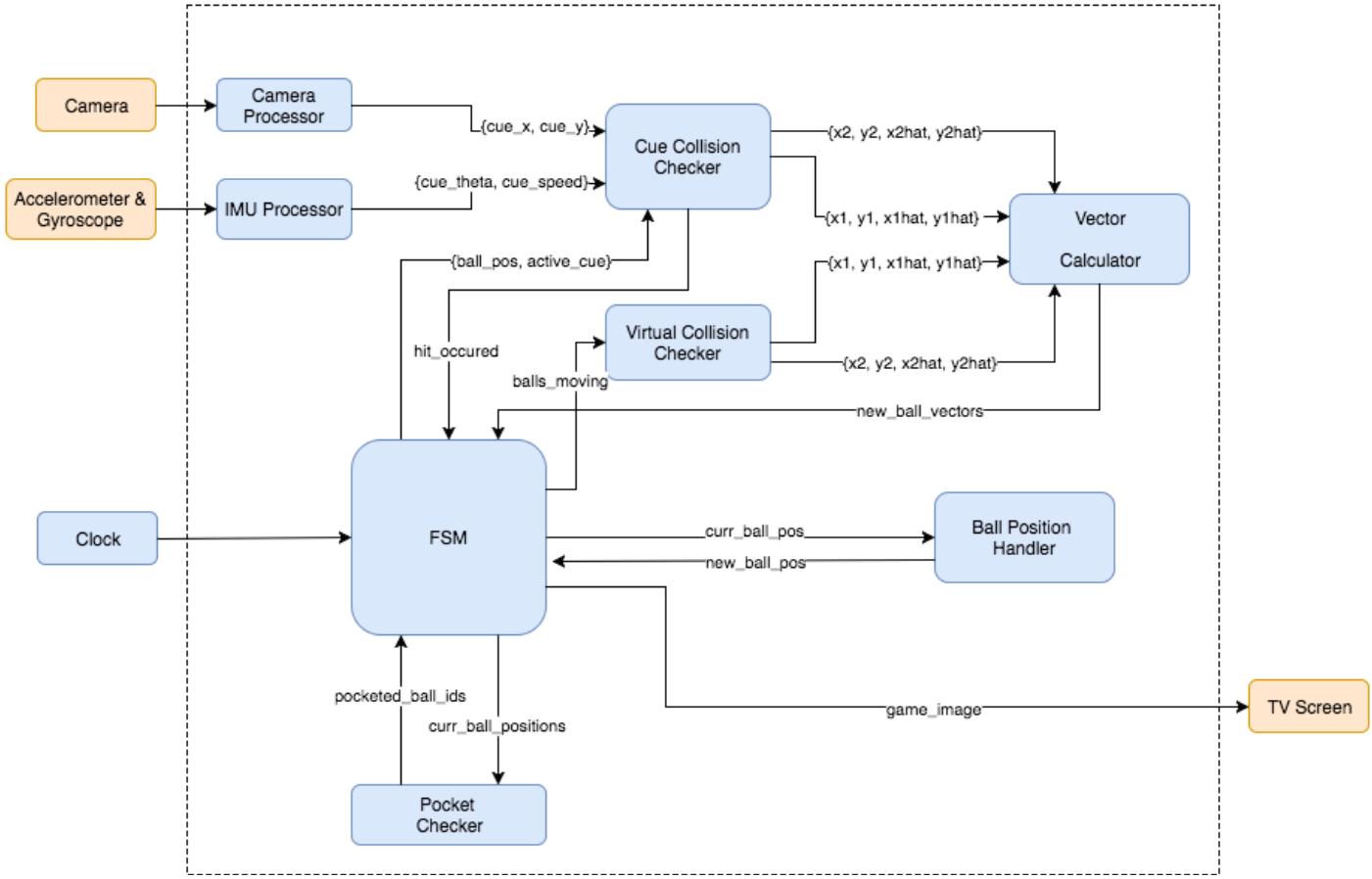

Figure 3.1: Detailed block diagram

3.1 IR Tracking & Camera Processor (Zareen)

IR object tracking will be used to determine the position of the cue. We plan to form two rings of IR LED's around two positions along the cue about 1 foot apart (distance will be determined

through experimentation). The NTSC camera will be mounted with a full view of the TV screen. Since the NTSC does not come with an IR filter, we will use a floppy disk as an IR filter.

The NTSC will be connected to the labkit, and the Camera Processor module will process incoming camera data. The module will identify the tip of the cue and output the (x,y) position of cue with respect to the coordinate system of the board. Since the TV screen has a resolution of 1024x768 pixels, the outputs `cue_x` and `cue_y` will be represented by 10-bit numbers.

3.2 IMU Processor (Zareen)

The IMU Processor will convert raw accelerometer and gyroscope data into the angle and speed of the cue. We plan to use the MPU-9250, which contains a 3-axis accelerometer and 3-axis gyroscope and uses digital SPI communication. The board will be physically attached to the back of the cue stick, and its output will be directly connected to the labkit. The module will process the raw data and output `cue_speed` and `cue_theta`, the magnitude of the cue's speed and its angle with respect to the board's coordination system.

3.3 Cue Collision Checker (Zareen)

The Cue Collision Checker (CCC) module will determine whether the cue has collided with a virtual ball. The module's inputs include the current (x,y) coordinates of all the balls provided by the FSM, the (x,y) coordinates of the cue in question provided by the Camera Processor (Section 3.1), and the speed and angle of the cue provided by the IMU Processor (3.2). Only the current player's cue coordinates are provided; if other players' cues collide with balls, they are ignored. The cue is treated like a ball of a very small radius. If the cue's coordinates approach the radius around any of the balls' coordinates, the module outputs a high signal, `hit_occurred`, to the FSM to indicate a collision. It also outputs to the Vector Calculator (3.5.2) the (x,y) coordinates and velocity vectors of the cue and the ball it collided with.

3.4 Game FSM (Matt)

The Game FSM (which will be referred to as just the FSM) is used to update the current state of the game, process inputs from a variety of modules, and output the correct video stream to the TV. Inputs for the FSM are the processed IMU data (3.2), the Cue Collision Checker (3.3), the Hole Checker (3.9), and the Ball Position Handler (3.5). The FSM outputs to the Ball Position Handler (3.5), Ball Collision Checker (3.5.1), and TV screen. States for the FSM include waiting for the game to start, player 1's turn, player 2's turn, and a victory (and defeat) screen. Other intermediate states may include waiting for a cue strike, calculating ball's positions, and a special state when the 8-ball is the only remaining ball. The FSM determines which state to transition to by waiting for inputs such as cue strikes, no balls moving, a ball rolling in a pocket, and the 8-ball rolling in a pocket. Because constantly checking the position of each ball for collisions will be computationally expensive, the FSM can turn off the computations in states when the balls will not move using its outputs to the Ball Position Updater and Ball Collision

Checker. Sending outputs from multiple modules through the FSM helps ensure consistent clocking throughout the system.

3.5 Ball Position Updating (Matt)

3.5.1 Ball Collision Checker

The Ball Collision Checker (BCC) is responsible for checking for collisions between all possible balls (which may vary from 2 to 12 depending on the goals reached). The BCC takes in as its inputs the location of every ball (from the ball's center) from the FSM (which receives them from the Ball Position Handler) and returns the x and y coordinates and x and y components of the motion vector for each of the two balls in the collision. The BCC can efficiently check for collisions by calculating the absolute distance between every pair of balls. If this distance equals twice the radius of a pool ball, the balls must be colliding since every ball is a circle. The BCC then sends the (x,y) coordinates and velocity vector of each ball involved in the collision to the Vector Calculator.

This will likely be the most computationally intensive part of the system, as the number of collisions to detect increases by $O(n^2)$ with the number of balls *n*. Furthermore, the realism of the pool balls movement is reliant on constant, real-time updates - this module will most likely be the limiting factor on the clock cycle. Therefore, the system will have to be highly parallelized to detect all possible collisions in a reasonable amount of time. The number of balls can be adjusted to increase or decrease performance if necessary.

There will also be a separate set of collision checkers to determine collisions with walls - however, because walls are stationary and exist in only one axis, the method of collision detection is much simpler - if the position of the center plus or minus the radius of the ball equals the wall position, a collision has occurred.

3.5.2 Vector Calculator

Every ball holds four pieces of information about its position and movement: the x and y coordinates of its center, and its velocity vector, with an x and y component. When two balls collide, the VC receives the position and velocity vector for each ball from either the BCC or the CCC. In an elastic collision between two pool balls, the tangential component (to the collision) of each ball's motion remains constant, while the normal components are exchanged between the two balls. The VC calculates the normal vector (n1 and n2) for each ball by calculating the vector between the center of ball 1 (c1) and the center of ball 2 (c2) and dividing by 2. The normal vector and initial motion vector of each ball are used to calculate the tangential vector by subtracting their squares (since the three vectors must form a right triangle). Then, the new motion vector for each ball is calculated by combining the square of the ball's tangential vector with the square of the normal vector of the other ball. The new motion vector for each ball is outputted to the FSM, which stores the position and movement information for each ball.

3.5.4 Ball Position Handler

The Ball Position Handler (BPH) takes in the current position, the speed, and direction of motion of each ball, and returns the updated position of each ball (tracked by its center point) in the next clock cycle. For each X and Y coordinate, BPH takes in the current point, and adds the length of the time interval multiplied by the magnitude of the component for that axis. This will update the location of the ball depending on its speed and angle at every time step, and return these updated coordinates to the FSM for display on the TV. The BPH also accounts for friction. The X and Y component of the ball's movement vector, if it is greater than 0 (regardless of collision), is decreased by a small amount to represent friction between the ball and the table - this friction coefficient can be tweaked after testing to simulate a more realistic table environment.

3.9 Pocket Checker (Matt)

The pocket checker is used to determine if a ball has fallen into a hole, thus removing it from play. Much like the other collision checkers, the pocket checker simply checks if a ball's center has become sufficiently close to the center of the pocket (represented by an X-Y coordinate), representing the ball rolling into the pocket. If the center of the ball becomes closer than the radius of the pocket, it will roll in. The Pocket Checker then returns the list of balls currently in pockets to the FSM, which removes these balls from future collision computations, in order to improve performance at later stages of the game.

4 Timeline

The division of tasks and project timeline are outlined in Table 4.1.

	Week 10/31	**Week 11/7**	**Week 11/14**	**Week 11/21**	**Week 11/28**	**Week 12/5**
IR Setup & Camera Processor	Zareen	Zareen				
IMU Processor		Zareen	Zareen			
Cue Collision Checker				Zareen	Zareen	
Integration				Both	Both	

Ball Sprite & Collisions	Matt					
Ball Collisions (moving balls)	Matt	Matt				
Speed/ friction updates		Matt	Matt			
Pocket Checker			Matt			
Game FSM			Matt	Matt		
Testing & Buffer					Both	Both

Table 4.1: Project timeline with deadlines and assignments of tasks.

5 Testing

5.1 IR Tracking

The IR tracking will be tested by visualizing the tracked cue and examining the outputted coordinates. The camera feed will be displayed via VGA output on the monitor connected to the labkit, with the identified cue highlighted in a distinctive color. As the cue is moved around in the real world, this real-time visualization will allow us to see if the camera's IR filter is tracking the targeted object. If it is unable to pick up the cue, the IR source may not be strong enough, may not be at the right angle, or the filter may not be good enough.

Once the object tracking has been solidified, we will overlay the (x,y) coordinates of the cue's tip on the visualization. The coordinate system of the board could also be overlaid. As the cue is moved around, the (x,y) coordinate values should change to match the corresponding position in the coordinate system.

5.2 IMU Processing

The IMU Processor module will be tested by examining the outputted angle and speed values. The cue can be physically rotated to specific angles, using a protractor for reference, and the outputted angles should match within a margin of error. Similarly, the cue will be moved at

different constant speeds and compared with the outputted speed values. The expected speeds will be determined by dividing the distance traveled by the time elapsed. We will also test the response time of the speed and angle calculations by abruptly changing these two factors independently, and measuring how quickly the values change accordingly.

5.3 Cue Collision Checker

The Cue Collision Checker will be tested on various static configurations of pool balls on the table. Under different configurations, we will test hitting balls hitting various balls with the cue at different angles and speeds. We will check that the module outputs a high signal after each cue-ball collision. Additionally, we will test moving the cue around the board region but not in the vicinity of balls, to ensure that the module maintains a low output when there is no expected collision.

5.3 Game FSM

Once the FSM has been established, we can associate a number of actions with switches and buttons on the labkit - each input could move a particular ball into a pocket or represent a cue strike. This will allow us to test transitions between states without relying on other inputs or modules, and test that the game properly transitions between each player's turn after all balls have stopped moving, and that the game properly recognizes win conditions (8-ball in pocket) and restarts the game.

5.5 Ball Collision Checker

A simple simulation of 2 balls, with random starting positions and velocities, bouncing around an empty screen can be used as an initial visual test of the collision checker - collisions should appear smooth, and the ball's resulting angles and speed should appear consistent with real-world physics. This can easily be scaled up by adding more balls to the simulation, testing the scalability of adding additional balls to the collision checker, a serious concern since the number of possible collisions scales with $O(n^2)$.

5.6 Pocket Checker

By populating the screen with a number of random test pockets and test balls, we can test balls rolling by pockets in a variety of scenarios - namely, that balls can roll close to pockets (overlap) without rolling into the pocket until the center of the ball overlaps with the pocket. Furthermore, we can test that balls which roll into pockets are removed from play by checking if other balls will collide with pockets instead of rolling into them (which would signify an invisible ball just sitting in the pocket).

6 Resources

The resources we will use beyond the 6.111 Labkit are described in Table 6.1.

Item	Quantity	Cost
TV screen	1	Free (borrowing from lab)
NTSC camera	1	Free (borrowing from lab)
MPU-9250	2	Free (borrowing from Joe)
IR LED's	~10	$10 (ordering online)
Cues	2	Free (borrowing)

Table 6.1: Project resources, quantities, and costs.

The TV screen and NTSC camera are being borrowed from the 6.111 lab for free, the MPU-9250 boards are being borrowed from our mentor Joe Steinmeyer, and the cues are being borrowed for free from outside of lab. Each MPU-9250 board contains a 3-axis accelerometer and 3-axis gyroscope. We plan to order either individual IR LED's or IR LED strips, which are available online for under $15. There is a possibility that we may have to order another MPU-9250, which is available for under $10. Overall, we estimate our project's budget to be no more than $25.

7 Challenges

The largest uncertainty in our project revolves around the speed of processing game display updates. The most challenging aspect will be rendering a real-time response immediately after a cue collision with a virtual ball. This is particularly difficult because of the compounded latency of processing the camera data, processing the IMU data, and rendering numerous round balls.

We have devised a minimal, reasonable, and stretch goal based on these anticipated challenges. The minimal goal is to render 1-2 square balls that do not animate in real time, but rather move after a delayed number of clock cycles. The reasonable goal is to have 5-6 round balls that move real-time, immediately following a cue collision. The stretch goal is to play with 12 round balls in real time, along with sound effects, haptic feedback on the cues, scoring, and full pool rules.

This project is challenging and complex, and it will require us to learn things that we do not have experience with, such as processing IMU data and rendering real-time physics-based simulations. We are excited to take on this challenge and building a unique pool experience that blends the physical world with the virtual world!