6.111 Final Project Proposal
2-Player 3D Air Hockey

Alex Chen, Abraham Shin, Yuqing Zhang

October 30, 2014

Contents

1 Overview

1.1 Summary

We will build a 2-player virtual air hockey game with a 3D graphical interface. In physical air hockey, two players hold mallets that are used to strike a puck, which slides across a rectangular table with limited friction. The edges of the table are raised so that they form walls that the puck bounces off of. The objective of each player is to hit the puck into the opponent's goal and defend his own. When a goal is scored, the scoring player wins a point. Air hockey is a very popular game, and our goal with our virtual implementation is to provide an exciting and easy-to-play game.

In our virtual implementation of the game, each player will wear a glove whose movement will control his mallet. The table, along the mallets and puck on it, will be rendered on the monitor. We will split the monitor in half and produce an image of the table from the perspective from both players. The table view is from a height above the table, looking down at an angle. We will use a camera to track the positions of the gloves, storing state to calculate velocity and other potential attributes. We will build a physics engine to handle collision detection and the calculation of proper post-collision velocity vectors.

1.2 Design

Our system will consist of four major components: object recognition, physics engine, graphics rendering, and game logic. The modules were designed as such in order to maximally separate independent parts of the code and minimize the number of inputs and outputs required between modules. With our current module designation, it is possible to work on all four modules in parallel with minimal dependence on other modules.

The object recognition module will find the location of the users' gloves and send the coordinates to the game logic. Then, the game logic will send the coordinates for the mallets and the puck as well as the velocity of the objects to the physics engine module, which will use the information to calculate the next location and velocity for the puck and returns the values to the game logic. Depending on the mode(state) of the game, the game logic will send the location of all three objects to the graphics module (game mode), send the locations of the mallets to the module with the puck's fixed (pause mode), or send the saved locations of the objects (replay mode). Then, the graphic rendering module will use the location of the objects and map them onto the 3D display of the game.

In terms of work distribution, Yuqing will be responsible for the object recognition module, Abraham will be in charge of the physics module, and Alex will be taking care of the graphics rendering module. Whoever finishes his module will start working on the game logic module.

2 Block Diagram

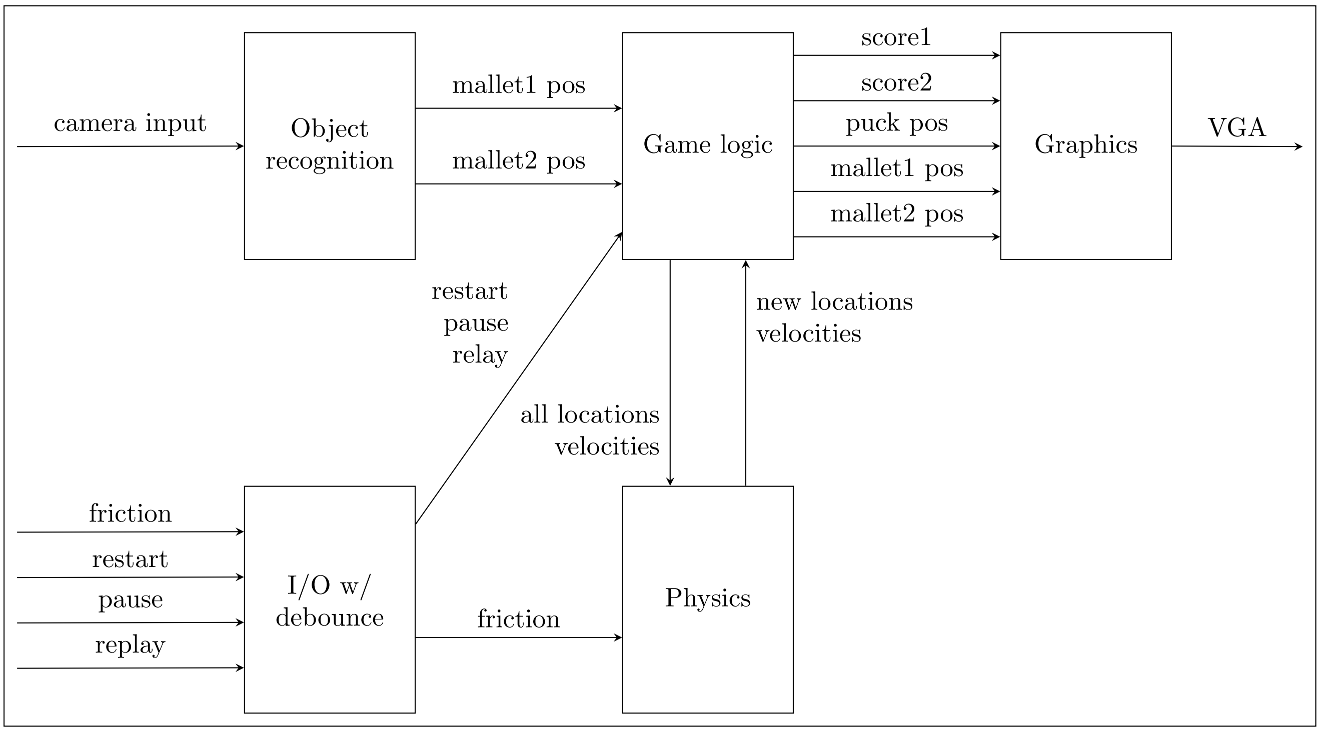

Figure 1: The overall block diagram features four modules. The object recognition module converts a camera input into mallet positions, which are then fed into the game logic module. The game logic module keeps track of the object positions and player scores. Object positions are updated with help from the physics module, and the game is displayed using the graphics module. The fifth module is just for debouncing input and output and is relatively minor.

Figure 1 contains the block diagram for the overall system. Note the four main modules. We have included a fifth one that is very simple and is responsible for debouncing input and output.

3 Modules

3.1 Object Recognition

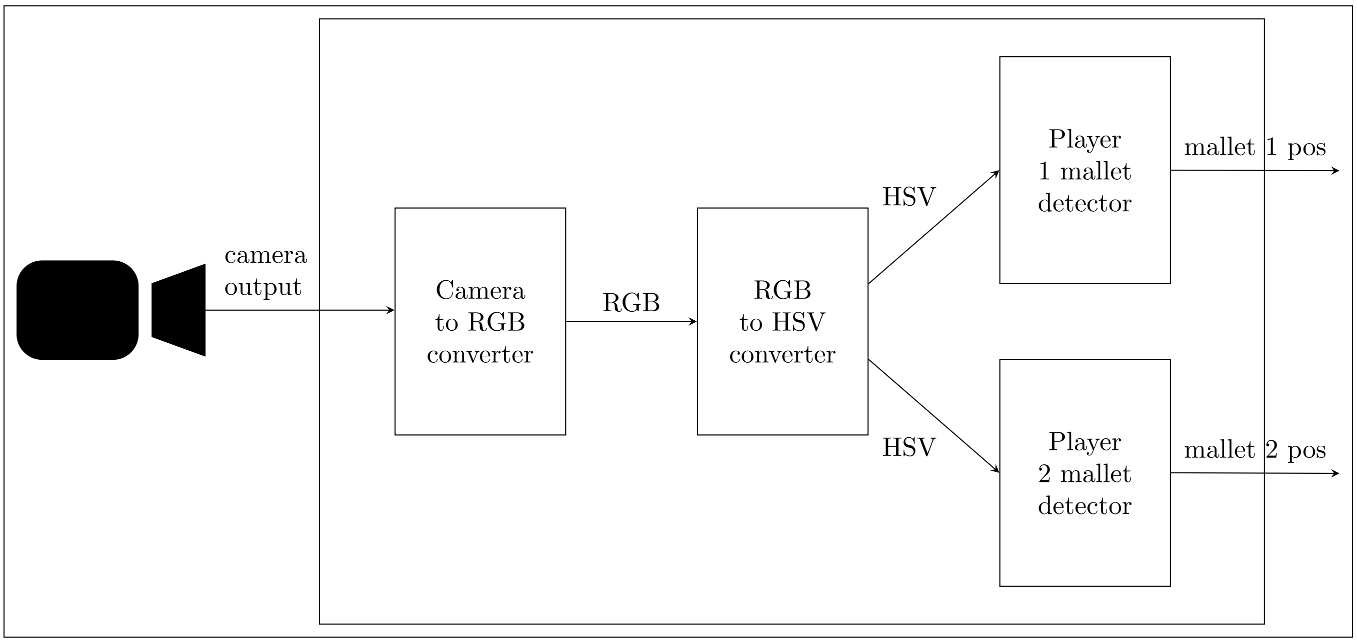

Figure 2: The object recognition module handles converting camera input into two mallet positions.

The object recognition module, shown in Figure 2 will take data from the camera as input and determine the position of the two mallets on the table. First the module reads in NTSC video data from the camera and converts the data into RGB format, which then allows conversion into HSV format. Video data in HSV format more readily facilitates object tracking. We then have mallet detector modules to translate the sensor data into a position, in terms of x and y coordinates, on our virtual table. These coordinates are then outputted and will be sent to the game logic module.

3.2 Physics Engine

The physics engine module, shown in Figure 3 will be responsible for calculating the projected velocity and location of the puck based on the current velocity and locations of the objects. While calculating the change in speed, three main factors will contribute to the change in velocity and locations of the puck: the collision against the wall, the collision against the mallet, and the friction between the puck and the table.

Calculating the change in velocity based on the collision against the wall will be the simplest of the three factors. This part's implementation will be similar to that in our Lab 3 in that the puck will change the sign of a velocity axis when it hits the wall, but we will be considering the impact friction in this calculation to mimic the dissipation of momentum when some energy is lost through the collision. Since we are only using one multiplication and sign change in the velocity for this collision, the implementation will not require any pipelining.

The collision against the mallet, on the other hand, will have many complications. When a player's mallet and the puck collides, we need to consider not only the angle at which the objects collide, but also the momentum of the two objects at the moment of impact. This calculation

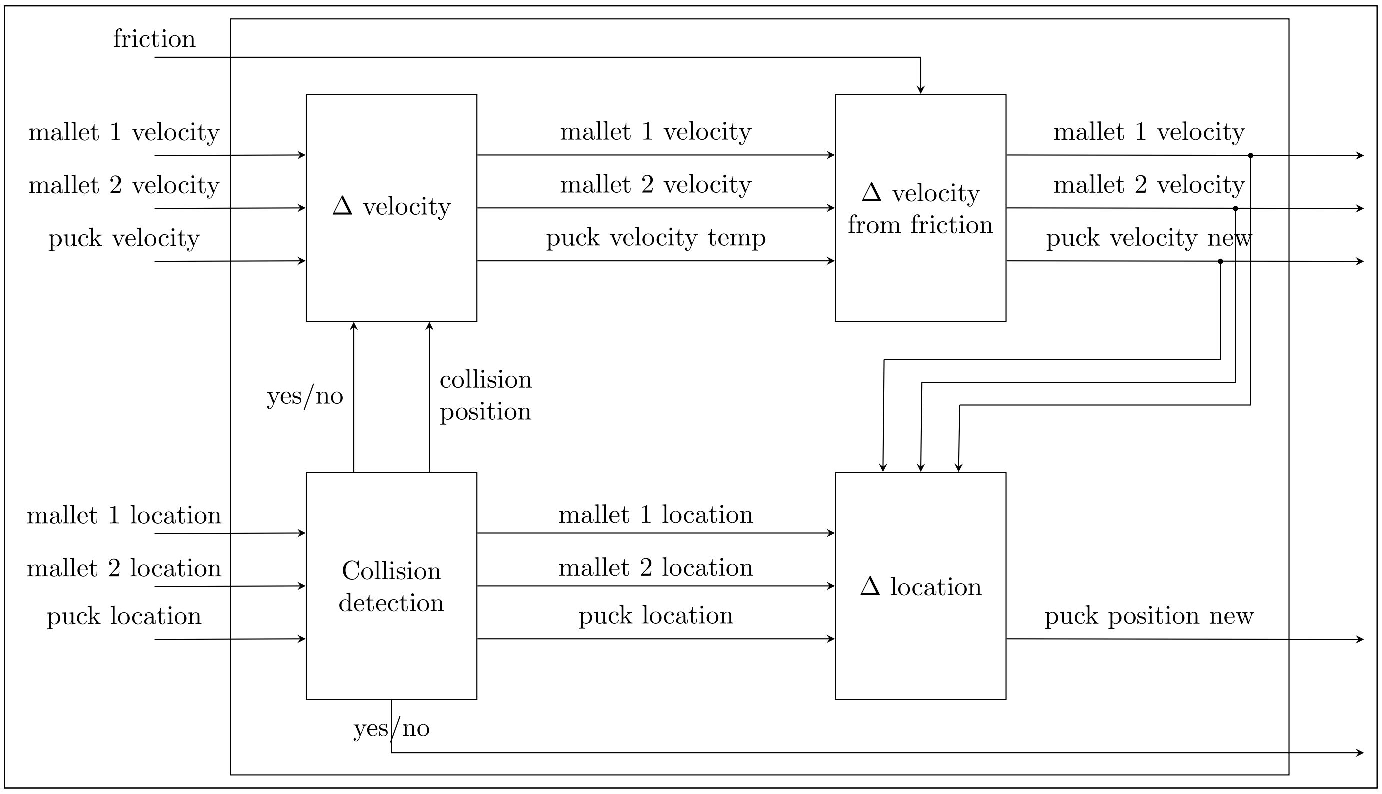

Figure 3: The physics module handles calculating new object locations and velocities given old ones. In some cases the input variable name equals the output variable name. These represent cases where the variable is not modified and is fed directly out of the module. One of the outputs of the collision detection module is a boolean denoting whether a collision occurred.

will be implemented by the vector sum method of two resulting sets of velocities. Since figuring out the angle of collision by itself will require two different multiplications (by checking if the distance between the center of the mallet and the puck are less than the sum of their radii), we will need to pipeline the module.

Last, the friction force will be calculated to mimic the real-life motion of a puck sliding over the surface of the table. Since we are modeling an air hockey game, the players will have the option of turning on or off the friction component of this game. If the friction is off, the velocity values that enter this module will exit unchanged, but the velocity will decrease accordingly if the users choose to have the friction on with a switch. The calculation itself only takes one multiplication, so pipelining will not be necessary.

When a puck is left on the surface of a working air hockey table, the puck traces a seemingly random walk around the table in a motion that replicates Brownian motion. Although this motion is unnecessary when the puck is moving at a high speed, this random motion will inevitably affect the trajectory of a slow puck. We will not be attempting to produce a small random motion for the puck from the beginning: this is only an extension of the project that may be implemented depending on whether we can finish all other parts of the design.

3.3 Graphical Interface

The graphical interface module (shown in Figure 4) converts data from the game logic into something displayed on a monitor through VGA. The main module accepts the mallet locations, puck locations, and scores. There are two states: game mode and goal mode.

In game mode, the graphical interface renders a divided screen, in which the left side shows the game from one player's point of view and the right side shows the game from the other

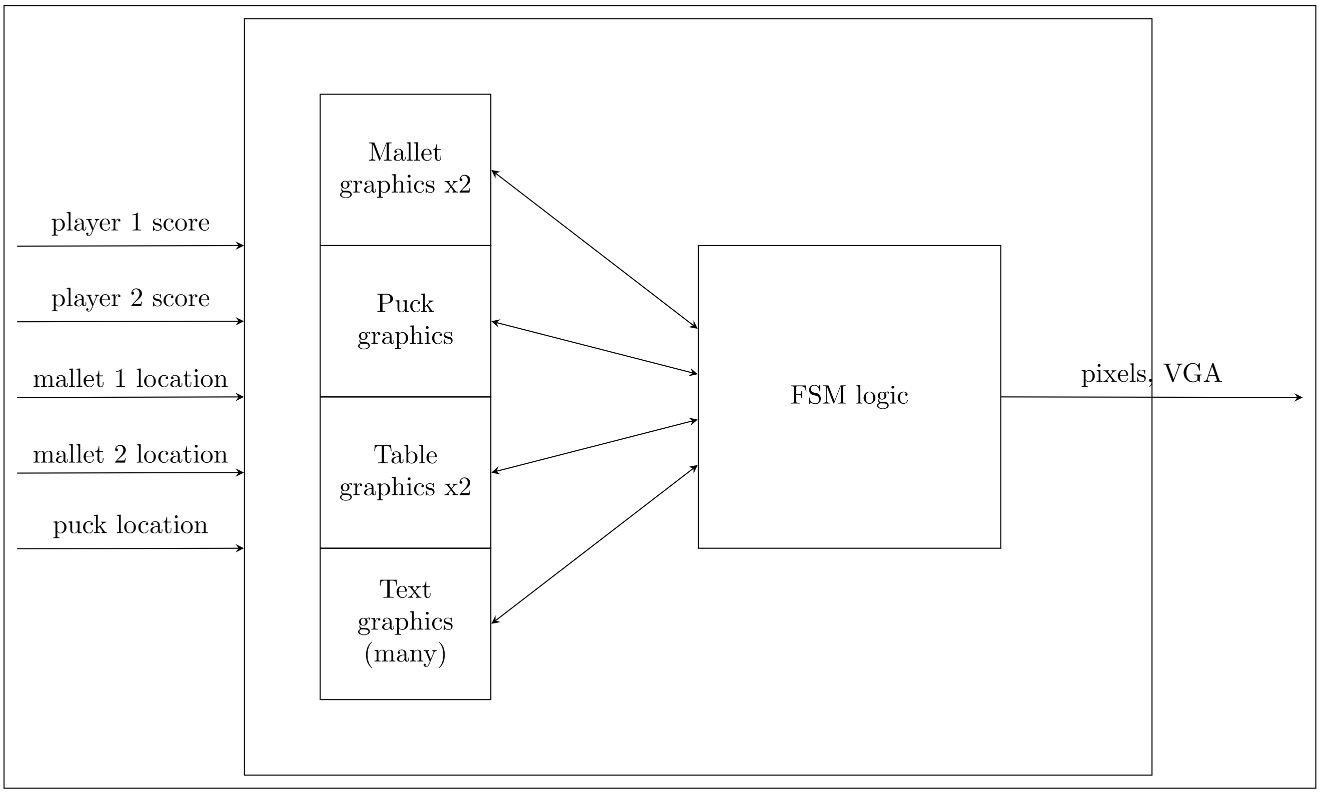

Figure 4: The graphical interface module accepts player scores and object positions as input and outputs to VGA a display of the game. Its FSM features two states, one for displaying the game and replays, and one for notifying players when a goal is scored. The FSM logic transmits pixel locations and object locations to the graphics modules (the four on the left), which return the pixel colors. The FSM logic combines the pixel colors and determines a proper output to VGA.

player's point of view. The view features an air hockey table shown from the perspective of someone as if they were actually playing air hockey. Each air hockey table depiction will feature two mallets and a puck. To calculate the locations of these objects on the screen, the graphics module will perform mathematical calculations to map coordinates from the physics world to the screen world. In addition to the air hockey table depictions, there will also be title text "Air Hockey" as well as the scores of the individual players displayed. For all this, the main graphics module will be interfacing with other modules. Each of these helper graphics modules will accept a pixel and the graphics object (the locations and dimensions of the puck, text, or mallet) and return what color the pixel should be. Using this information, the graphical interface will be able to render an output in realtime that will incorporate all the moving and static objects on the screen. For text and mallets, we may choose to use images stored in memory, but for memory conservation purposes we are currently planning to render them as geometric objects (rendering each letter and number individually).

Next, there is goal mode. When the graphical interface module detects that one of the scores of the players changed, it will display a huge "GOAL SCORED" on the screen, perhaps with other animations. It will be a short splash screen that happens every time a goal is scored, and then the game will revert back to its original state.

The biggest challenge for this module will be rendering everything from a 3D perspective. 2D is much simpler because everything is either rectangles or circles, but in 3D, the objects are not as simple: we have cylindrical pucks as well as oddly-shaped mallets. The table will not even be displayed as a rectangle, but instead as a trapezoid due to the perspective. Every other object will also have to be scaled in size depending on the distance from the viewer, and objects that are circles from above become ellipses on the screen. This will take careful mathematical calculation, and we will have to come up with efficient methods for mapping pixels from the physics world to the display world without being too slow.

If there is sufficient time, we may try to implement ray tracing in order to add a light source and shadows to the table to make the game appear more realistic. This would also have a chance of exceeding the clock cycle time and would be challenging to implement as a result. It does help, however, that the materials we would be using to render are relatively simple and there are not too many objects in the scene.

3.4 Game Logic

The game logic module is the main brain component of the design project. The module takes mallets location coordinates from the object recognition module and compares it with the last location to calculate the velocity. It then relays these information to the physics engine module and receives the updated locations and velocities of the puck. The locations of all the objects (mallets and puck) will be stored in a BRAM for the purpose of relay, which will be explained further in a later paragraph.

Along with the newly updated locations, the game logic receives a bit of information from the physics engine whether a collision had occurred and another bit for whether the collision occurred with the wall or the mallet. The main module then retrieves the appropriate sound effect from another BRAM (which contains the memory of pre-selected sound effects) to drive a speaker, and thereby producing the collision sound. To control different options for the game, we will be using several buttons as the input through the debounce module: replay, restart, and unpause. Once the puck reaches the goal area, the game pauses and the logic module will signal the graphical interface to increment the score. During the paused state, one can press the replay button to replay the last 100 location steps until the goal, and keep repeating until unpause button is pressed, at which point the next round of game will resume. At any time, if one wants to restart the game and reset the current scores to zeroes, one can press the restart button.

Due to the above implementation, FSM is essential in controlling the outputs to the graphical interface. In the paused state(1), the module will send the locations of the puck when it first enters the goal as well as the location of the mallet directly from the object recognition module so that the users can still move the mallets around while the puck is stuck in the goal. When the game is in the replay state(2), the module will take the last 100 location sets for the mallets and puck and sent them to the graphical interface so that the screen can show the last few seconds of the gameplay leading up to the goal repeatedly. If the game is in the play state(3), the output to the graphics module will be the locations of the mallets as above as well as the newly updated location of the puck from the physics engine module.

The biggest challenge for this module will be the communication with other modules. As the central control system, it has to receive and send information from all other modules in a controlled manner, possibly at different frequencies, since exchanging the information about the pucks location with the physics module may take longer than it takes to prepare other information to assemble and send to the graphics module. Therefore, the module may benefit from extra inputs and outputs to communicate with other modules about when this module is ready to receive or send information.

4 Materials List

Our only needed materials are two differently colored gloves, a camera, and speakers, which are all either available in the lab or able to be purchased for a low cost.

5 Timeline

5.1 Milestones and Deadlines

- Week of 10/27
  - Design system modules
  - Proposal conferences
  - Complete proposal (Thursday 10/30)
- Week of 11/3
  - Start implementing respective modules
  - Project design presentation
  - Block diagram conference (Thursday 11/6, 5pm)
- Week of 11/10
  - Have functional independent modules by end of week (fully tested graphics, object recognition, and physics)
  - Proposal revision (Friday 11/14, 5pm)
  - Proposal checkoff checklist meeting (Friday 11/14, 5pm)
- Week of 11/17
  - Finish integration and debugging of all modules
  - Finish game logic
- Week of 11/24 (Thanksgiving)
  - More time for integration and debugging of all modules
  - Presentable by end of this week
- Week of 12/1
  - Work on final report
  - Finishing touches (user interface, additional features)
- Week of 12/8
  - Checkoff (Monday 12/8, 4pm)
  - Demos, and video (Tuesday 12/9, 6pm)
  - Finish final report (Wednesday 12/10, 5pm)