Autonomous Pool Playing Robot

Software Design

Eric Le Fort
leforte@mcmaster.ca
1308609

March 20, 2017

Contents

List of Tables

List of Figures

Date	Revision #	Comments	Authors
14/11/2016	0	- High Level Design document creation	Eric Le Fort
28/11/2016	0	- High Level Design first draft completion	Eric Le Fort
16/12/2016	0	- Added System Block Diagram	Eric Le Fort
25/12/2016	0	- Low Level Design document creation	Eric Le Fort
14/01/2017	0	- Low Level Design first draft completion	Eric Le Fort
17/01/2017	0	- Added Physics and Communication sections	Eric Le Fort
20/03/2017	1	- Consolidated Design Document Creation	Eric Le Fort

Table 1: Revision History

1 Introduction

This document's purpose is to describe the architectural and low-level design of the software controlling the Autonomous Pool Playing Robot. Both the architecture for the encompassing system as well as architectures for distinct subsystems will be discussed. The purpose of this document will be to document the decisions made concerning the system's design as well as provide enough detail so that the programming of the system can be as trivial as possible.

1.1 System Description

This system will provide a control system for an autonomous pool playing robot. It will include three separate computational units and four separate programs.

The first computational unit will be a camera (likely a camera phone). This device will be responsible for reacting to a request, taking an image and then relaying that image.

The next computational unit will be the PC. This device will have two separate programs that must execute for the system to be successful. One program will handle the VR and the other will handle shot selection as well as message passing between devices. The VR program will handle processing the image from the camera into a table state that can be used by the shot selection algorithm. The other program on the PC will handle performing an algorithm to determine which shot should be taken, where to move the cue in order to take that shot, instructing the camera to take pictures, receiving the image from the camera, receiving a signal to take a shot from the μC and communicating the shot that must be taken back to the μC.

The last computational unit, the μC, will be responsible for interpreting the shot instructions provided by the PC into signals to control the machine accordingly, receiving control signals from the system, providing signals to move the machine out of the way to predetermined locations when requested, and sending the signal to take a shot back to the PC.

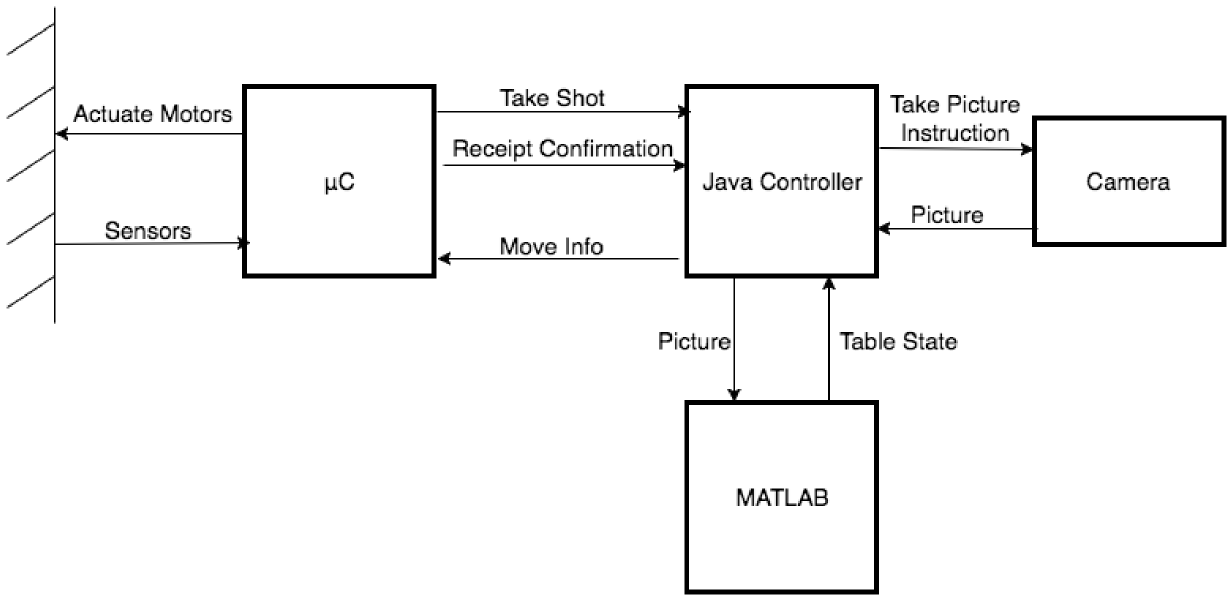


Figure 1: System Block Diagram

1.2 Overview

This document has nine sections not including this one. Each section contains either design diagrams or further explanations to further describe the architecture of this system and is intended to prepare the software team to implement the design.

- **Use Cases**: Describes possible user interactions with the system as well as the intended results of those interactions through the use of simple diagrams. A Use Case Diagram is also provided for reference.
- **Analysis Class Diagram**: Defines the various classes in the system, how they will be connected, and their type (boundary, controller, or entity).
- **Architectural Design**: This section defines the overall system architecture as well as the architectures of all sub-programs.
- **Detailed Class Diagram**: The section will provide a detailed class diagram to define the contents of each class in the program.
- **CRC Cards**: Each program will be broken up into their specific classes. The responsibilities of each class as well as any collaboration required with other classes to fulfill each responsibility (if any) will be listed.
- **Module Guide**: This section will outline the responsibilities and secrets of each module as well as each module's MIS and MID.
- **Physics Simulations**: This section will describe the method by which the pool table physics simulations will be performed as well as the various equations and constants involved.
- **Communication Protocols**: This section will describe inter-process communication protocols necessary in this system.
- **Scheduling of Tasks**: This section defines various timing constraints for various operations. It also provides state charts and sequence diagrams to illustrate the flow of the system.

1.3 Naming Conventions & Definitions

This section outlines the various definitions, acronyms and abbreviations that will be used throughout this document in order to familiarize the reader prior to reading.

1.3.1 Definitions

Table 2 lists the definitions used in this document. The definitions given below are specific to this document and may not be identical to definitions of these terms in common use. The purpose of this section is to assist the user in understanding the requirements for the system.

Table 2: Definitions

Term	**Meaning**
X-axis	Distance along the length of the pool table
Y-axis	Distance across the width of the pool table
Z-axis	Height above the pool table
End-effector	The end of the arm that will strike the cue ball
θ	Rotational angle of end-effector
Cue	End-effector
Personal Computer	A laptop that will be used to run the more involved computational tasks such as visual recognition and the shot selection algorithm
Camera	Some form of image capture device (e.g. a digital camera, smartphone with a camera, etc.)
Table State	The current positions of all the balls on the table
Entity	Classes that have a state, behaviour and identity (e.g. Book, Car, Person, etc.)
Boundary	Classes that interact with users or external systems
Double	Double-precision floating point numbers

1.3.2 Acronyms & Abbreviations

Table 3 lists the acronyms and abbreviations used in this document.

Table 3: Acronyms and Abbreviations

Acronym/Abbreviation	**Meaning**
VR	Visual Recognition
PC	Personal Computer
μC	Micro-Controller
CRC	Class Responsibility Collaboration

2 Use Cases

This section will outline the use cases that this system will be expected to handle. A Use Case Diagram will also be provided to help illustrate these cases. All use cases are initiated by the user interacting with a physical interface that will send a signal to the μC. Between operations, the various programs will revert to a dormant state to await the next instruction.

2.1 Use Case Diagram

The following diagram depicts the use cases of this system.

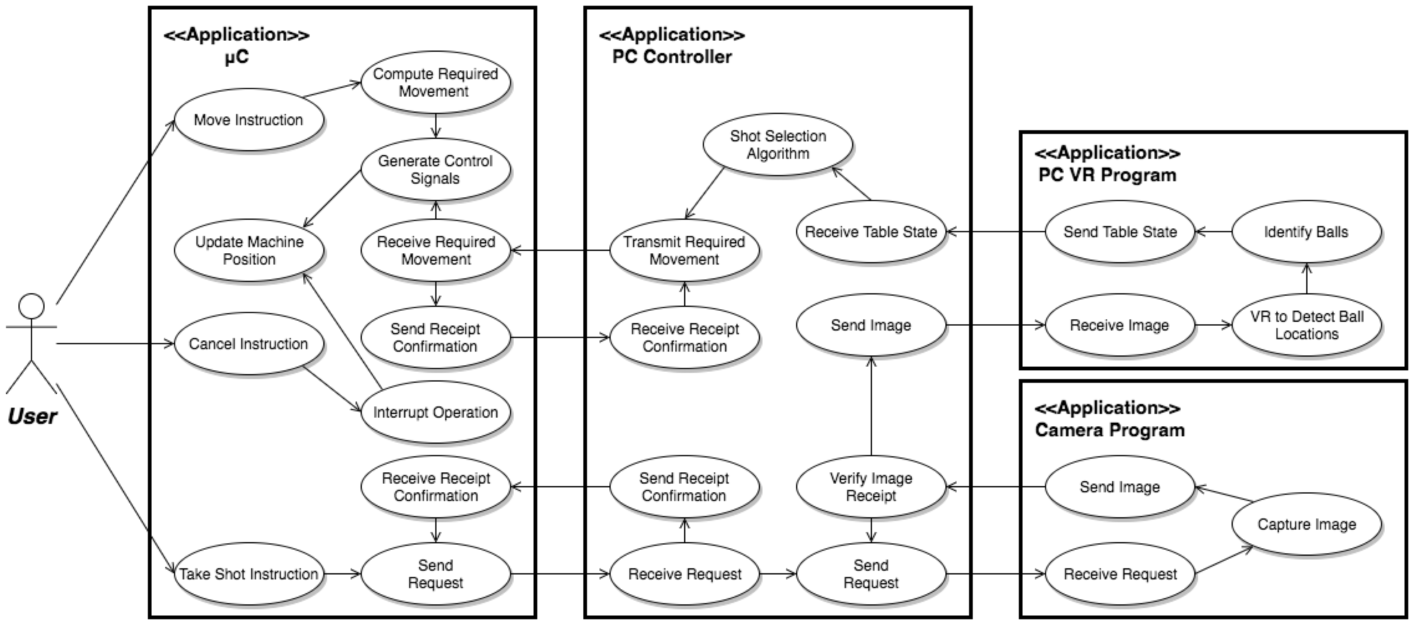


Figure 2: Use Case Diagram

2.2 Move Instruction

This use case involves only the μC. The user presses the *move* button. Once the signal has been received, the μC will decide which predetermined set point to move to based on the current position of the machine that is stored in its memory. The μC will then determine the signals required to relocate the machine to the appropriate location and then send them to the machine. Once those signals are sent, the μC will update its memory to store the new position of the machine.

2.3 Cancel Instruction

This use case involves only the μC. The user presses the *cancel* button. Once the signal has been received, the μC will immediately stop the machine's motion. It will then update its memory to store the current position of the machine.

2.4 Take a Shot Instruction

This use case involves all four programs: the μC, the PC Controller Program, the Camera Program, and the PC VR Program in that order. The user presses the *take a shot* button. Once the signal has been received, the μC will send a request to the PC to begin its process and wait a small amount of time for the PC to send a receipt response. If the μC does not receive a response within the specified timeframe, it will resend its request. This process will be repeated until a response is received. The PC Controller Program will then send a signal to the camera to take a photo. Upon receiving this signal, the camera will take a picture of the table and send this back to the PC. If the picture was received correctly, the PC Controller Program will then start the PC VR Program and provide the picture to be used. Otherwise, the PC

Controller Program will continue to resend the request to take a picture as necessary. The PC VR Program will utilize object detection to determine where the balls are on the table and which ball is which. Once collecting this information, it is provided to the PC Controller Program through an intermediate file. The PC Controller Program will then use its shot selection algorithm to determine how to strike the pool ball to take based on the table state provided and transmit that shot to the μC. After that, the μC will send a receipt signal back to the PC Controller Program. The μC will interpret the shot information to create an instruction set of signals to move the machine appropriately and take the shot.

2.5 Take a Shot Operation Interrupted During PC Computation

This use case involves all four programs: the μC, the PC Controller Program, the Camera Program, and the PC VR Program in that order. The user presses the *take a shot* button. Once the signal has been received, the μC will send a request to the PC to begin its process and wait a small amount of time for the PC to send a receipt response. If the μC does not receive a response within the specified timeframe, it will resend its request. This process will be repeated until a response is received. Sometime between this point and getting the selected shot communicated back to the μC, the user presses the *cancel* button. The PC will complete its normal operation but once the shot is communicated back to the μC, it will simply be ignored.

3 Analysis Class Diagrams

The following section contains the analysis class diagrams for the μC and the PC Controller program. The PC VR program and Camera are not complicated enough to require the inclusion of these diagrams.

3.1 μC Analysis Class Diagram

The following is an analysis class diagram depicting the relationships between classes in the μC.

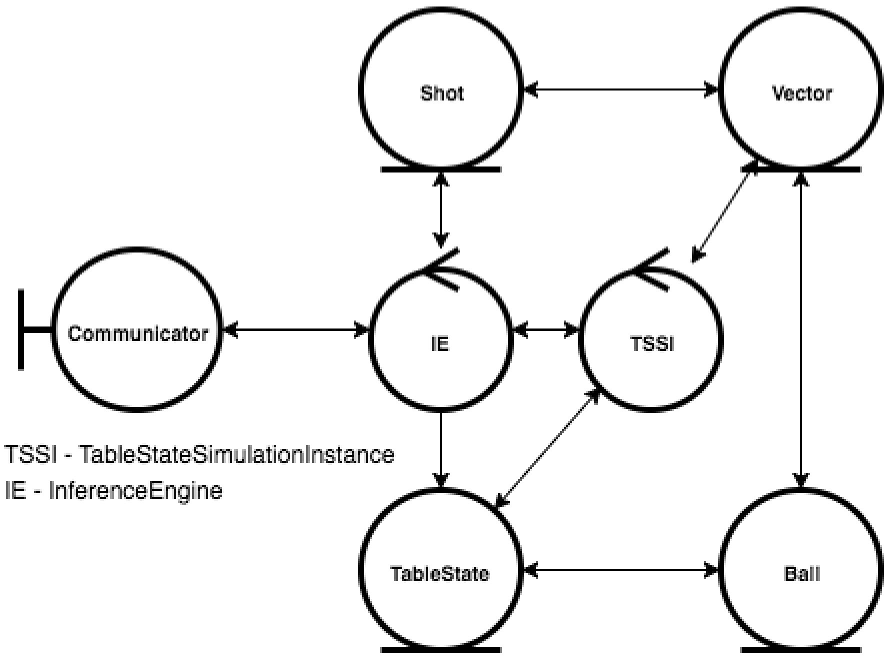


Figure 3: μC Analysis Class Diagram

3.2 PC Controller Analysis Class Diagram

The following is an analysis class diagram depicting the relationships between classes in the PC Controller program.

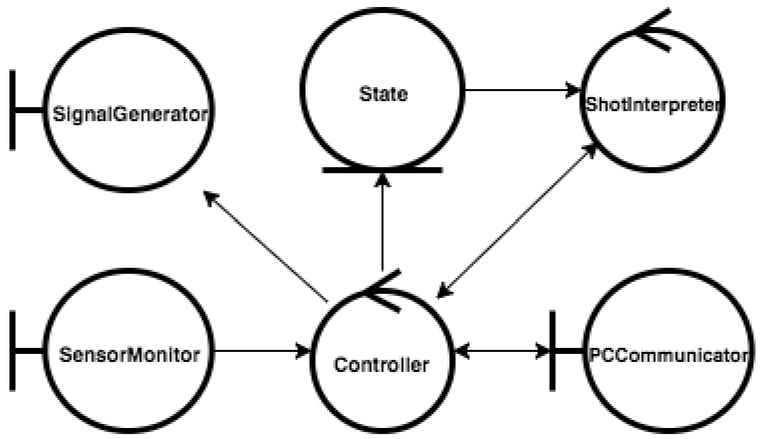


Figure 4: PC Controller Analysis Class Diagram

4 Architectural Design

This section will discuss the architectures that are to be used while designing this system. The goal of these architectures is to promote the overall quality of the software for this system. In particular, the software should be more efficient, robust, and maintainable as a result of this architectural design.

4.1 System Architecture

The architecture of this system as a whole will be modelled using the *Batch Sequential* style. This architecture involves sending chunks of data after each program is completely done processing and promotes simple division of subsystems that can operate as independent programs. This creates a strong separation of concerns within the larger system.

The shortcomings of this system are that it does not allow for concurrency and it has low throughput. Luckily, in this situation, the throughput is not intended to be very high. Also, each subsystem must wait for the previous system to fully complete its task in order to proceed regardless. Therefore, since this system does not even allow for concurrent operation, there is no loss in potential efficiency.

4.2 Subsystems

The subsystems will follow the batch sequential architecture on a system level but that architecture does not make sense for the subsystems themselves. This section will discuss the architectures that will be utilized on the subsystem level.

4.2.1 μC Architecture

This subsystem will utilize a *Sensor-Controller-Actuator* architecture style. This style is well-suited to embedded systems that deal with hardware which is optimal for this subsystem. The style breaks the subsystem up into four types of components: interfaces, sensors, actuators, and a controller. An illustration of the architecture is provided below:

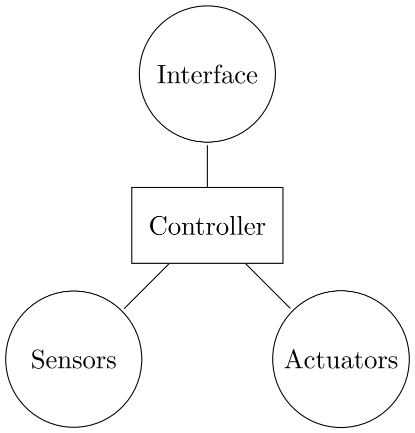


Figure 5: Sensor-Controller-Actuator Architecture

In the case of this subsystem, the breakdown of the components into these types will be as follows:

- Sensors - System State
  - Calibration sensors (quantity yet to be determined)
- Sensors - Interface
  - Stop buttons (multiple located throughout machine, quantity yet to be determined)
  - Move button
  - Take shot button
- Logic
  - Motion logic
  - Current state info
- Actuators
  - Translational motors ($\times 2$)
  - Rotational motor
  - Pneumatic end-effector

4.2.2 PC Controller Architecture

This subsystem will utilize a *Rule-Based* architecture style. This style is well suited to making strategic moves while playing a game with rigid rules constraining actions. In other words, for deciding which shot is the best to take given the current state of table and the rules of the game of pool. This style breaks down the subsystem into four types of modules: an interface, a rule-set, working memory, and an inference engine. The interface will be where the system gets its input and sends its output. The rule-set is a particular sort of knowledge base that contains the rules constraining the acceptable actions. The working memory is temporary memory used by the subsystem. Lastly, the inference engine is what ties the subsystem together. Its job is to apply the rules appropriately and, using the working memory as a tool, generate the appropriate result to send as output.

In the case of this subsystem, the breakdown of these modules into these types will be as follows:

- Interface
  - Input – table state
  - Output – specification of shot (position, angle, and force)
- Rule-Set
  - Rules of pool
  - Mechanical constraints
  - Physical constants (i.e. ball weight, table friction, spring constant of table cushions, etc.)
- Working Memory
  - Table state
  - Current optimal shot
- Inference Engine
  - Simulate results of shots
  - Check rules against shot simulations at some incremental angle
  - Score shots based on weighted application of rules

4.2.3 Camera Architecture

This subsystem will utilize an *Event-Driven* architectural style. This will likely be the simplest subsystem present in the system. The role of this program is to sit idle waiting for a request to take a picture. Once it receives that request, it should take a picture and then communicate that image back to the PC Controller program. This request is the only event this program must respond to.

4.2.4 PC VR Program Architecture

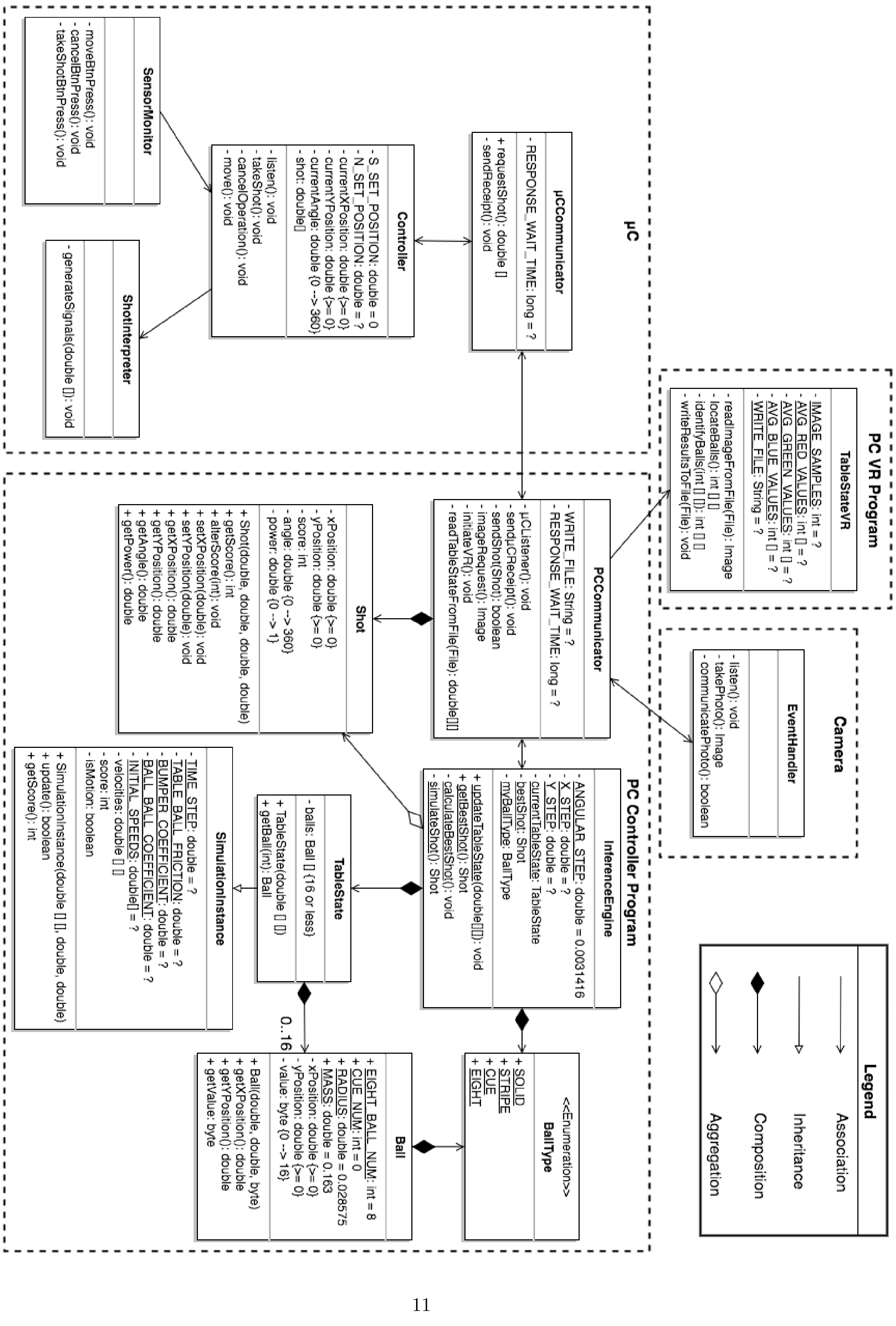
This subsystem will also utilize a *Batch Sequential* architecture style. This style has already been discussed in the system architecture and so will not be discussed again here. In this subsystem, the process is begun upon receipt of the request to start operation. The subsystem will then utilize a MATLAB VR library to detect the balls as objects on the table using the image provided. The results of this will be passed to the next step which will use the positions of the identified objects to determine which specific ball is which using colour comparison. Once this step completes, the table state has been computed and is then sent back to the PC Controller program. The issues of low throughput and lack of concurrency are not an issue in this case as the system would have to wait for the previous step to complete in order to begin the next step anyway.

An illustration of the specific implementation of this architecture is provided below:

$$\begin{tikzcd}
\text{in} \arrow[r, "\text{image}"] & \text{object detection} \arrow[r, "\substack{\text{image \&} \\ \text{object positions}}"] & \text{ball identification} \arrow[r, "\text{table state}"] & \text{out}
\end{tikzcd}$$

Figure 6: PC VR Architecture

5 Detailed Class Diagram



μC
μCCommunicator
- RESPONSE_WAIT_TIME: long = ?
+ requestShot(): double []
- sendReceipt(): void
Controller
- S_SET_POSITION: double = 0
- N_SET_POSITION: double = ?
- currentXPosition: double {>= 0}
- currentYPosition: double {>= 0}
- currentAngle: double {0 --> 360}
- shot: double[]
- listen(): void
- takeShot(): void
- cancelOperation(): void
- move(): void
SensorMonitor
- moveBtnPress(): void
- cancelBtnPress(): void
- takeShotBtnPress(): void
ShotInterpreter
- generateSignals(double []): void
PC VR Program
TableStateVR
- IMAGE_SAMPLES: int = ?
- AVG_RED_VALUES: int [] = ?
- AVG_GREEN_VALUES: int [] = ?
- AVG_BLUE_VALUES: int [] = ?
- WRITE_FILE: String = ?
- readImageFromFile(File): Image
- locateBalls(): int [] []
- identifyBalls(int [] []): int [] []
- writeResultsToFile(File): void
Camera
EventHandler
- listen(): void
- takePhoto(): Image
- communicatePhoto(): boolean
PC Controller Program
PCCommunicator
- WRITE_FILE: String = ?
- RESPONSE_WAIT_TIME: long = ?
- μCListener(): void
- senduCReceipt(): void
- sendShot(Shot): boolean
- imageRequest(): Image
- initiateVR(): void
- readTableStateFromFile(File): double[][]
InferenceEngine
- ANGULAR_STEP: double = 0.0031416
- X_STEP: double = ?
- Y_STEP: double = ?
- currentTableState: TableState
- bestShot: Shot
- myBallType: BallType
+ updateTableState(double[][]): void
+ getBestShot(): Shot
- calculateBestShot(): void
- simulateShot(): Shot
Shot
- xPosition: double {>= 0}
- yPosition: double {>= 0}
- score: int
- angle: double {0 --> 360}
- power: double {0 --> 1}
+ Shot(double, double, double, double)
+ getScore(): int
+ alterScore(int): void
+ setXPosition(double): void
+ setYPosition(double): void
+ getXPosition(): double
+ getYPosition(): double
+ getAngle(): double
+ getPower(): double
TableState
- balls: Ball [] {16 or less}
+ TableState(double [] [])
+ getBall(int): Ball
SimulationInstance
- TIME_STEP: double = ?
- TABLE_BALL_FRICTION: double = ?
- BUMPER_COEFFICIENT: double = ?
- BALL_BALL_COEFFICIENT: double = ?
- INITIAL_SPEEDS: double[] = ?
- velocities: double [] []
- score: int
- isMotion: boolean
+ SimulationInstance(double [] [], double, double)
+ update(): boolean
+ getScore(): int
<<Enumeration>>
BallType
+ SOLID
+ STRIPE
+ CUE
+ EIGHT
Ball
+ EIGHT_BALL_NUM: int = 8
+ CUE_NUM: int = 0
+ RADIUS: double = 0.028575
+ MASS: double = 0.163
- xPosition: double {>= 0}
- yPosition: double {>= 0}
- value: byte {0 --> 16}
+ Ball(double, double, byte)
+ getXPosition(): double
+ getYPosition(): double
+ getValue(): byte
0..16
Legend
Association
Inheritance
Composition
Aggregation

Figure 7: The system's detailed class diagram.

6 Class Responsibility Collaboration (CRC) Cards

This section provides CRC cards for all classes in each of the programs. Each card will list the responsibilities as well as any classes the must be collaborated with to perform that responsibility.

6.1 μC CRC Cards

The following are the CRC cards for the μC program:

SensorMonitor	
Responsibilities	Collaborators
- Sensing signals from buttons	(None)
- Sensing signals from calibration sensors	(None)

Communicator	
Responsibilities	Collaborators
- Send shot calculation request to the PC Controller Program	Communicator (PC)
- Receive request receipt confirmation from PC Controller Program	Communicator (PC)
- Send shot receipt confirmation to PC Controller Program	Communicator (PC)
- Receive shot specification from PC Controller Program	Communicator (PC)

SignalGenerator	
Responsibilities	Collaborators
- Control translational motors	Shot Interpreter
- Control rotational motor	Shot Interpreter
- Control pneumatic end-effector	Shot Interpreter

ShotInterpreter	
Responsibilities	Collaborators
Translate shot instruction into list of appropriate signals	State, Communicator (μC)

State	
Responsibilities	Collaborators
Store and allow access to data regarding the current physical position of the machine	Communicator (μC)

Controller	
Responsibilities	Collaborators
Interrupt operation if that instruction is received	SensorMonitor
Control flow of program operation	(None)

6.2 PC Controller CRC Cards

The following are the CRC cards for the PC Controller program:

Communicator	
Responsibilities	Collaborators
- Listens for request from μC	Communicator (μC)
- Sends confirmation of receipt to μC	Communicator (μC)
- Sends shot specification to μC	Communicator (μC), Shot, Vector
- Listen for receipt confirmation from μC	Communicator (μC)
- Sends image capture request to camera	EventHandler
- Listens for response from camera	EventHandler
- Initiate the PC VR program	(None)
- Read table state from file	(None)

Vector	
Responsibilities	Collaborators
- Stores and allows access to an x- and y-component of a vector	(None)

Ball	
Responsibilities	Collaborators
- Store and allow access to its position	Vector
- Store and allow access to the value representing its identity	(None)

Shot	
Responsibilities	Collaborators
- Store and allow access to the position from which to take this shot	Vector
- Store and allow access to the angle from which to take this shot	(None)
- Store and allow access to the amount of power to use to take this shot	(None)
- Store and allow access to the scoring assigned to this shot	(None)

TableState	
Responsibilities	Collaborators
- Store and allow access to the positions of the balls present on the table	Ball, Vector
- Store and allow access to the number of both types of ball on the table	Ball

TableStateSimulationInstance	
Responsibilities	Collaborators
- Maintain the positions and velocities of balls on the table at the current time step	TableState, Vector
- Update the positions and velocities of balls after one time step	TableState, Vector
- Keep track of whether there is still movement occurring	Vector
- Keep track of the scoring of the simulation (of a shot)	TableState

InferenceEngine	
Responsibilities	Collaborators
- Calculate the shot to be made	TableStateSimulationInstance, TableState, Shot, Vector

6.3 Camera CRC Cards

The following is the CRC card for the Camera program:

Event Handler	
Responsibilities	Collaborators
- Listen for request from PC Controller	Communicator (PC)
- Take a photo	(None)
- Communicate the photo	Communicator (PC)

6.4 PC VR Program CRC Cards

The following is the CRC card for the PC VR program:

TableStateVR	
Responsibilities	Collaborators
- Read image from a file	(None)
- Locate balls using VR	(None)
- Determine ball identities	(None)
- Write table state to a file	(None)

7 Module Guide

This section discusses the various modules that this system is comprised of. The modules are divided based on which program they belong to. For each module, its responsibilities, secrets, MIS, and MID will be outlined.

7.1 Camera Modules

The following is the module contained within the Camera subsystem.

7.1.1 EventHandler

Responsibilities

- Listen for request from PC Communicator
- Take a photo
- Communicate photo to PC Communicator

Secrets

- Picture-taking process

MIS

This module is an always-running program which executes the function of taking a picture and communicating it back to the requesting program. While this module is not performing the previous function, it will be listening for a request to be made.

MID

The state charts below provide a succinct depiction of this module's internal design.

7.2 PC VR Program Modules

The following is the module contained within the PC VR subsystem.

7.2.1 TableStateVR

Responsibilities

- Read image from a file
- Locate balls using VR
- Determine ball identities
- Write table state to a file

Secrets

- Object detection algorithm
- Ball identification algorithm

MIS

This module takes in an image of a pool table and analyzes it in order to return the locations and identities of each of the pool balls on the table.

MID

This module has 4 main steps. First it must read in the image from a predetermined file location. Then it locates the pool balls in that picture using a Visual Recognition object detection algorithm (supplied by a MATLAB library). Next, it will identify which ball is which according to comparing pixel colours within the detected objects to the colours of different pool balls using a LAB colour space. Lastly it will write these results to a file.

The following pseudocode better describes this process:

```
read image from file (path to file);
locate balls (image);

for every detected object:
        sample pixels within;

        while (pixels not similar to ball archetype AND arcge);
                increment archetype being checked;

        if (index out of range):
                raise unidentifiable ball error;
        else:
                results[index for this ball] = this object's location;

return results;
```

7.3 PC Controller Modules

The following are the modules contained within the PC Controller subsystem.

7.3.1 InferenceEngine

Responsibilities

- Calculate the best shot to be made

Secrets

- Algorithm to choose which shots to simulate
- The computer's ball type (i.e. stripes or solids)
- The rules of pool

MIS

This module allows for specification of a TableState through the use of a 2-D array of doubles. Using that TableState, the module will simulate various potential shots in order to determine an optimal one which is then accessible to other classes.

MID

This module iterates through shots that are to be simulated by SimulationInstances. In order to minimize computation, it only looks at shots that have a hope of directly hitting another legal ball (e.g. the eight ball cannot be struck first unless all of the player's other balls are already sunk). At the end of this computation, this module returns the specification of the optimal shot.

The following pseudocode better describes this process:

```
determine balls to shoot;

while (legal balls left to check){
        calculate smallest angle from cue ball to target ball;
        calculate largest angle from cue ball to target ball;

        for (all angles from smallest to largest angle){
                for (every power option){
                        create SimulationInstance for this shot;

                        while (simulation not finished){
                                simulation.update();
                        }

                        update current shot's score;

                        if (score of new shot > optimal shot score):
                                update optimal shot;
                }
                current angle += ANGULAR_STEP;
        }
}

return new Shot(calculated X, calculated Y, optimal shot's angle, optimal shot's power);
```

7.3.2 PCCommunicator

Responsibilities

- Listens for request from μC
- Sends confirmation of receipt to μC
- Sends shot specification to μC
- Listens for confirmation of receipt from μC
- Sends image capture request to camera
- Listens for response from camera
- Initiate the PC VR program
- Read table state from file

Secrets

- Receipt confirmation message contents
- Maximum time awaiting a response

MIS

This module is an always-running process which communicates with the various programs in this system while also providing control flow for the PC Controller program.

MID

The state charts below provide a succinct depiction of this module's internal design.

7.3.3 SimulationInstance

Responsibilities

- Maintain the positions and velocities of the balls on the table at the current time step
- Update the positions and velocities of the balls on the table after a time step
- Keep track of whether there is still movement happening
- Keep track of the scoring of the simulation (of a shot)

Secrets

- The method of calculating a shot's score
- Shot simulation algorithm
- Physical constants
- Simulation time step

MIS

This module handles the physics simulation involved with taking a shot while also scoring the shot according to various criteria. The state is updated for a new time step by calling the appropriate command. Once it returns *false*, there is no further motion and the simulation is complete.

MID

This module handles performing a discrete time step simulation of a shot according to a shot and initial state of the table. In order to achieve this, it updates to the next snapshot of the simulation until every ball remaining on the table is stationary.

First, the following pseudocode describes how this object is created:

```
Given:
        An array of balls
        The x and y components of the cue ball's initial velocity

inMotion = true;
score = 0;

Initialize the velocities array;

velocities[Ball.CUE_NUM][0] = initial x component;
velocities[Ball.CUE_NUM][1] = initial y component;
```

The following pseudocode assumes a very fine time step but may not not be feasible given the imposed time constraints (which will be tested empirically):

```
for (all balls on table){
        update position according to velocity;
        update velocity according to friction;
}

for (all balls on table){
        if (in pocket AND sufficiently slow){
                if (cue ball):
                        score reduced;
                else if (8 ball AND not shooting 8):
                        score reduced;
                else if (wrong type):
                        score reduced;
                else:
                        score increased;

                set position to be off table;
                set velocity to 0;
        }else{
                for (current ball to last ball){
                        if (collision):
                                compute resulting velocities;
                }
}

for (all balls on table){
        if (velocity != 0){
                inMotion = true;
                return inMotion;
        }
}
```

7.4 μC Modules

The following are the modules contained within the μC subsystem.

7.4.1 Controller

Responsibilities

- Control flow of program operation
- Interrupt operation if cancel instruction is received

Secrets

- Set movement positions (for “move” commands)
- Instruction dispatch process

MIS

This module handles the control flow for the μC Program including determining appropriate movement and creation of interrupts when necessary.

MID

The state charts below provide a succinct depiction of most of this module’s internal design. The only other notable component in this module is how it selects where to move when a “move” command is received. In that event, it compares the current location to the two set locations at either end of the table. Whichever one is furthest is the one which the machine is moved towards. This is designed in a way such that the machine moves as far away as possible from where it was in the way of the user.

7.4.2 SensorMonitor

Responsibilities

- Sensing signals from buttons
- Sensing signals from calibration sensors

Secrets

- The method of noticing signals

MIS

This module monitors the control signals coming from the buttons and calibration sensors and notifies the Controller when one of these sensors are activated.

MID

The state charts below provide a succinct depiction of this module's internal design.

7.4.3 ShotInterpreter

Responsibilities

- Translate shot instruction into list of appropriate signals
- Control translational motors
- Control rotational motor
- Control pneumatic end-effector

Secrets

- Algorithm to determine appropriate movement
- Method of transmitting signals to machine

MIS

This module uses a movement specification that it is provided in order to compute and generate the signals necessary to have the machine perform the required motion.

MID

The state charts below provide a succinct depiction of this module's internal design.

7.4.4 μCCommunicator

Responsibilities

- Send shot calculation request to the PC Controller program
- Receive confirmation of receipt from PC Controller program
- Receive shot specification from PC Controller program
- Send confirmation of receipt to PC Controller program

Secrets

- Receipt confirmation message contents
- Maximum time awaiting a response

MIS

This module handles communicating with the PC Controller Program in order to compute the optimal shot to take.

MID

The state charts below provide a succinct depiction of this module's internal design.

8 Physics Simulations

A substantial component of this system will be the ability to simulate and score a shot. Scoring will be determined by which balls are sunk (if any).

8.1 Constants

The following constants will be used by this system's simulation engine. Certain values must be measured empirically and will be marked To Be Measured (TBM). All values will be in standard SI units.

Constant	Description	Value
C_{R_b}	Coefficient of Restitution, ball - ball	0.96
C_{R_w}	Coefficient of Restitution, ball - wall	0.866
g	Gravitational Acceleration	9.807
v_{i_1}	Initial Speed, soft hit	TBM
v_{i_2}	Initial Speed, medium hit	TBM
v_{i_3}	Initial Speed, hard hit	TBM
μ_k	Kinetic Friction, ball - cloth	TBM
m_b	Mass, ball	0.163
v_{sink}	Maximum Speed to Sink Ball	TBM
x_{max}	Maximum x-coordinate	1.848
y_{max}	Maximum y-coordinate	0.921
r_{ball}	Radius (ball)	0.0286
μ_s	Static Friction, ball - cloth	TBM
T	Time Step	0.01
w_c	Width, mouth of corner pocket	TBM
w_s	Width, mouth of side pocket	TBM

8.2 Equations

The program will utilize the following equations to predict the evolution of the simulation.

Ball - Ball Collision

$$v_{a_f} = \frac{C_{R_b} m_b (v_{a_i} - v_{b_i}) + m_a v_{a_i} + m_b v_{b_i}}{m_a + m_b}$$

Since the masses are equivalent, this simplifies to..

$$v_{a_f} = \frac{1}{2}(C_{R_b}(v_{b_i} - v_{a_i}) + v_{a_i} + v_{b_i}) \tag{1}$$

Ball - Wall Collision (perpendicular component)

$$v_f = -C_{R_w} v_i \tag{2}$$

Ball - Wall Collision (parallel component)

$$v_f = ? \tag{3}$$

Ball Velocity Update

$$v_f = ? \tag{4}$$

Ball Position Update

$$d_f = d_i + vT \tag{5}$$

9 Communication Protocols

This section will outline the protocols used for communicating between the various devices and programs within this system. This is crucial as the different programs will rely on standard formats in order to understand the contents of messages.

9.1 μC - PC Controller

These programs must both be able to communicate message receipts, the μC must be able to make a shot request to the PC Controller, and the PC Controller program must be able to send shot specifications back to the μC. The receipts will be very simple messages, just containing an unsigned integer byte with the value 200. The shot request will also be an unsigned integer byte holding the value 55. Lastly, the shot specification will be sent as follows:

170, x, y, angle, power

where the number 170 is stored as an unsigned integer byte, x and y are the coordinates for the shot in double-precision floating point (8 bytes), angle is the angle of the shot in double-precision floating point, and power is the power of the shot stored in a single byte (either 1, 2, or 3).

The various numbers at the beginning of messages act as both checksums and indicators of the message's contents. The numbers were chosen purposefully in a manner such that they have very different one byte binary representations (they share at most 4 common digits).

Communication will occur wirelessly using either bluetooth or Wi-Fi as determined by availability of hardware and software resources.

9.2 PC VR - PC Controller

The PC VR program must be able to communicate the table state to the PC Controller. This will be accomplished by writing results to a file with the following format:

Cue Ball x-coordinate, Cue Ball y-coordinate
1 Ball x-coordinate, 1 Ball y-coordinate
2 Ball x-coordinate, 2 Ball y-coordinate
⋮
15 Ball x-coordinate, 15 Ball y-coordinate

where each set of coordinates is separated by a newline character and there are exactly 16 lines.

9.3 Camera - PC Controller

The PC Controller must be able to request a picture be taken and the camera must be able to communicate the image back to the PC Controller. Similar to the protocol between the μC and the PC Controller, the request will be a single unsigned integer byte holding the value 55 and the image data being sent will be preceded by a single unsigned integer byte holding the value 170. As mentioned previously, these values will act as checksums.

10 Scheduling of Tasks

The goal of this section is to outline the ordering and maximum allowable time frames of tasks in this program.

10.1 Allocation of Time

From the requirements document, there is only 90 seconds allowed between pressing a button and a shot being made. The most difficult computational step will be the shot simulations and so this section will deduce how much time the machine will have for that step.

Firstly we must account for how long the physical machine would need in the worst case (moving all the way across both axes and rotating 180 degrees). To be fair, we will allocate 20 seconds to this operation.

From here we can divvy up the remaining time among the various computational blocks. The smaller tasks such as communication, recognizing the button press, and other such operations will be allocated a total of 5 seconds. The process of object detection and identification will be given 15 seconds. This leaves 50 seconds to process the necessary simulations.

10.2 State Charts

The following charts illustrate the lifecycle of all relevant classes in this system. This section is meant to depict a more isolated picture of how each class will operate.

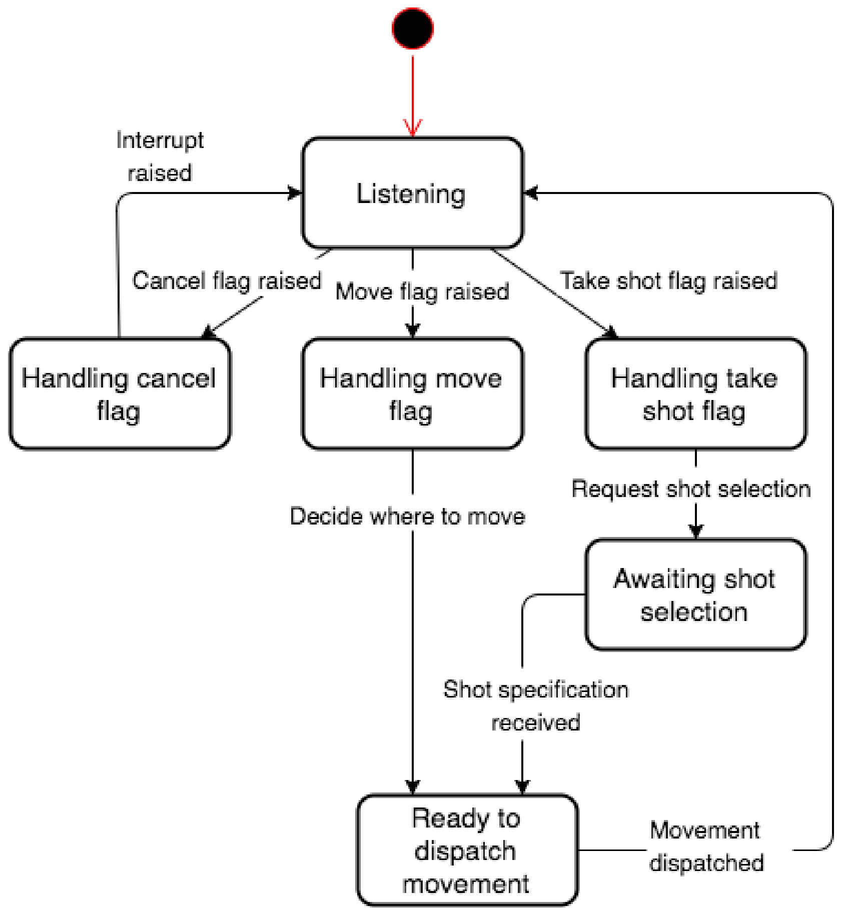


Figure 8: A state chart for the Controller class.

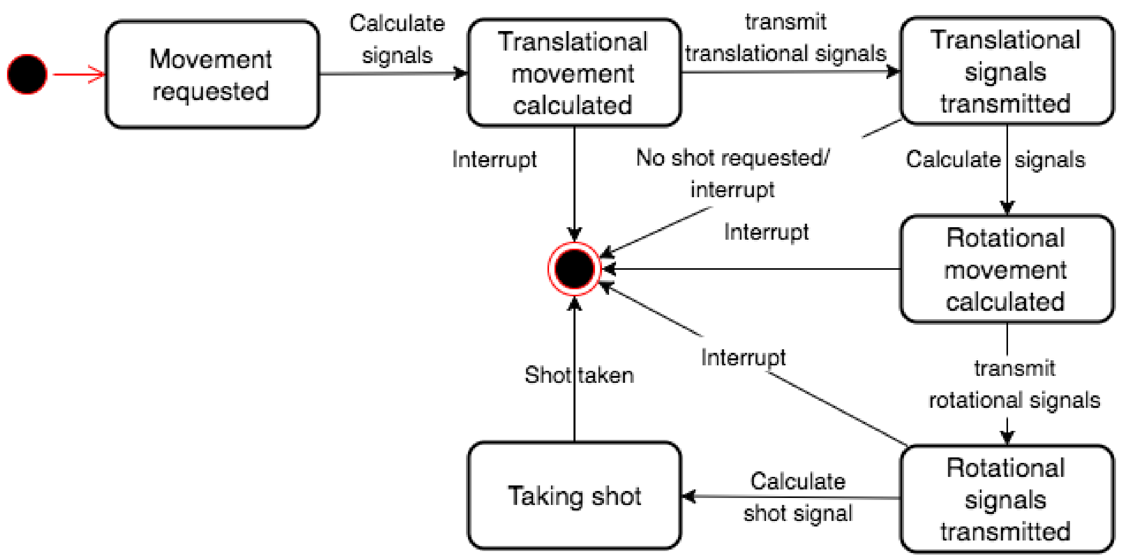

Movement requested
Calculate signals
Translational movement calculated
transmit translational signals
Translational signals transmitted
Interrupt
No shot requested/ interrupt
Calculate signals
Interrupt
Rotational movement calculated
Shot taken
Interrupt
transmit rotational signals
Taking shot
Calculate shot signal
Rotational signals transmitted

Figure 9: A state chart for the ShotInterpreter class.

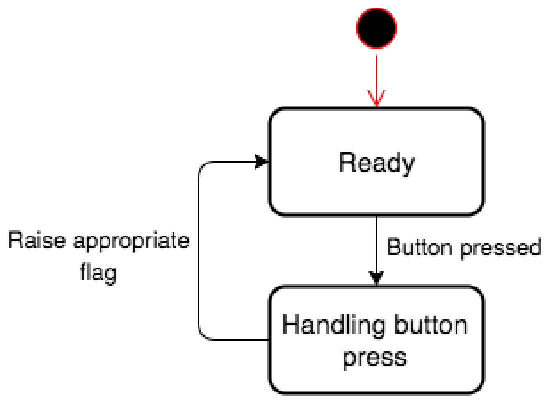


Figure 10: A state chart for the SensorMonitor class.

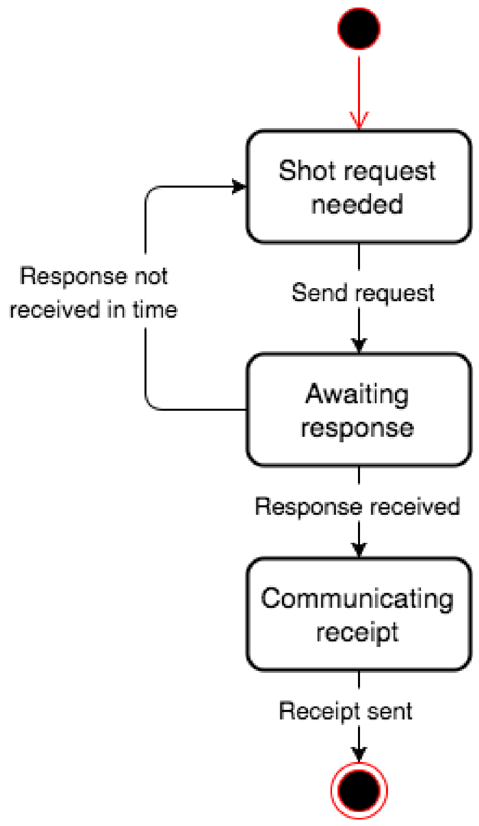


Figure 11: A state chart for the μCCommunicator class.

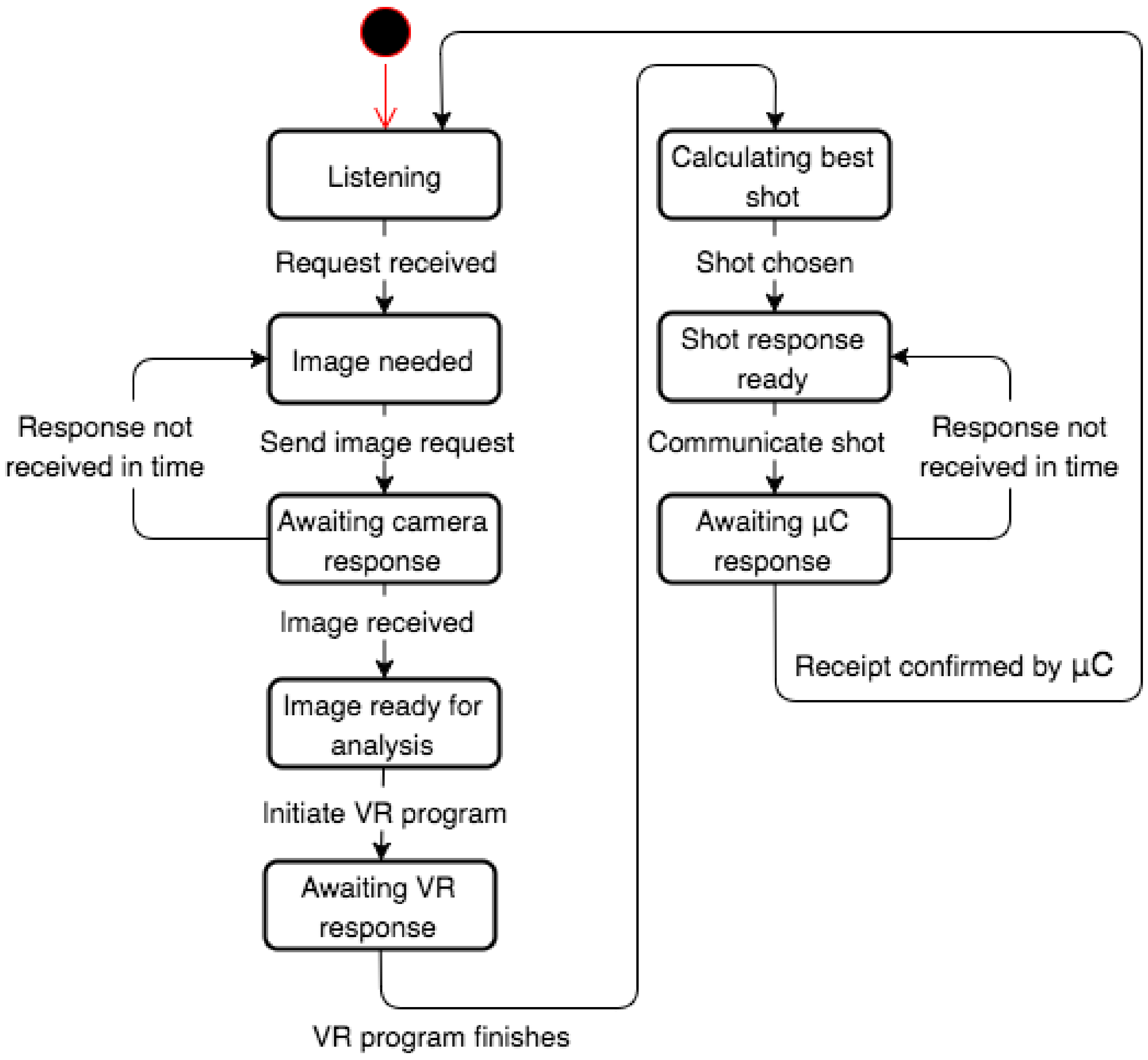


Figure 12: A state chart for the PCCommunicator class.

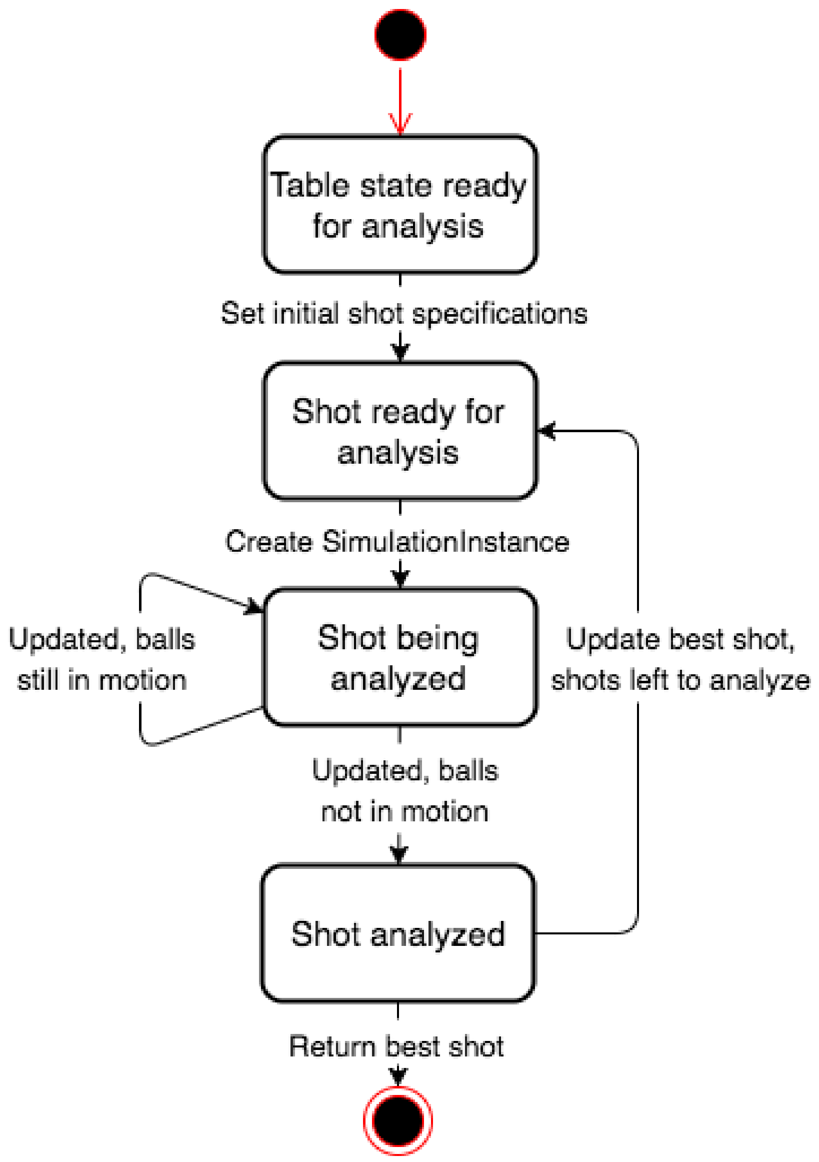


Table state ready for analysis
Set initial shot specifications
Shot ready for analysis
Create SimulationInstance
Updated, balls still in motion
Shot being analyzed
Update best shot, shots left to analyze
Updated, balls not in motion
Shot analyzed
Return best shot

Figure 13: A state chart for the InferenceEngine class.

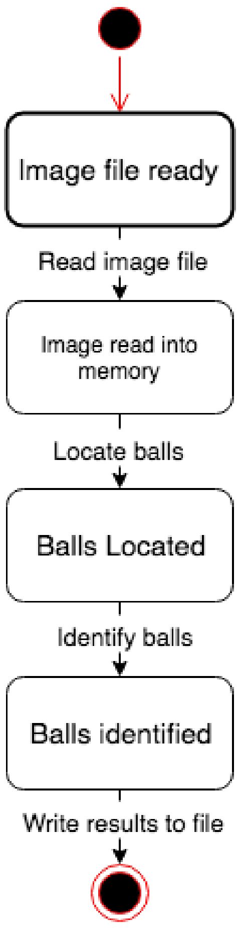


Figure 14: A state chart for the TableStateVR class.

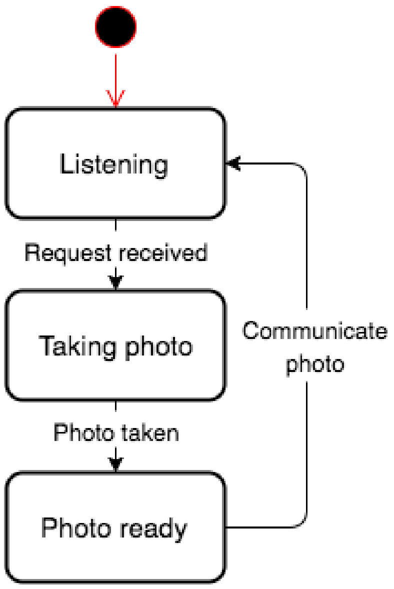


Figure 15: A state chart for the EventHandler class.

10.3 Sequence Diagrams

The following are various sequence diagrams for different actions the system is required to perform. These diagrams are meant to provide better context for how the classes interact with each other to perform certain tasks.

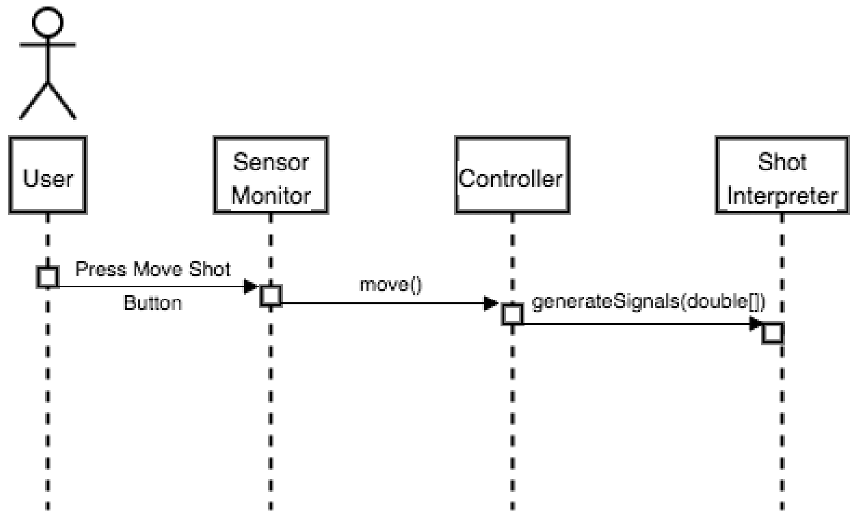


Figure 16: A sequence diagram for the "move" operation.

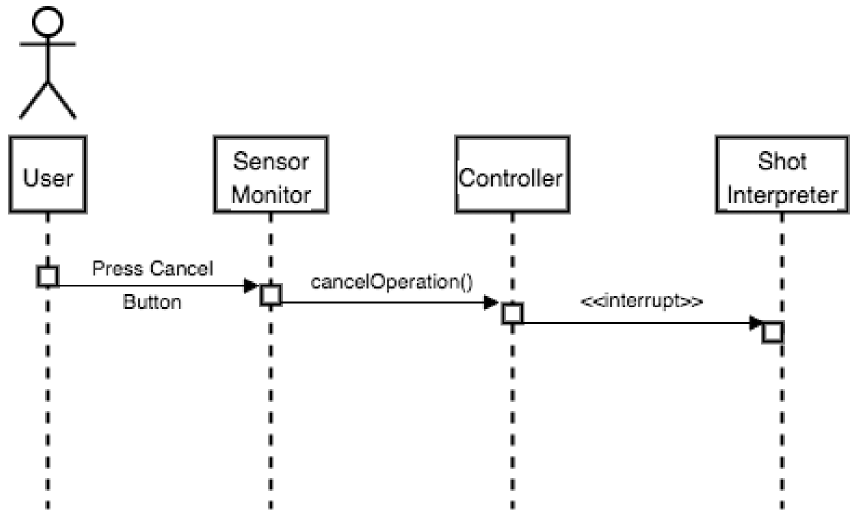


Figure 17: A sequence diagram for the "cancel" operation.

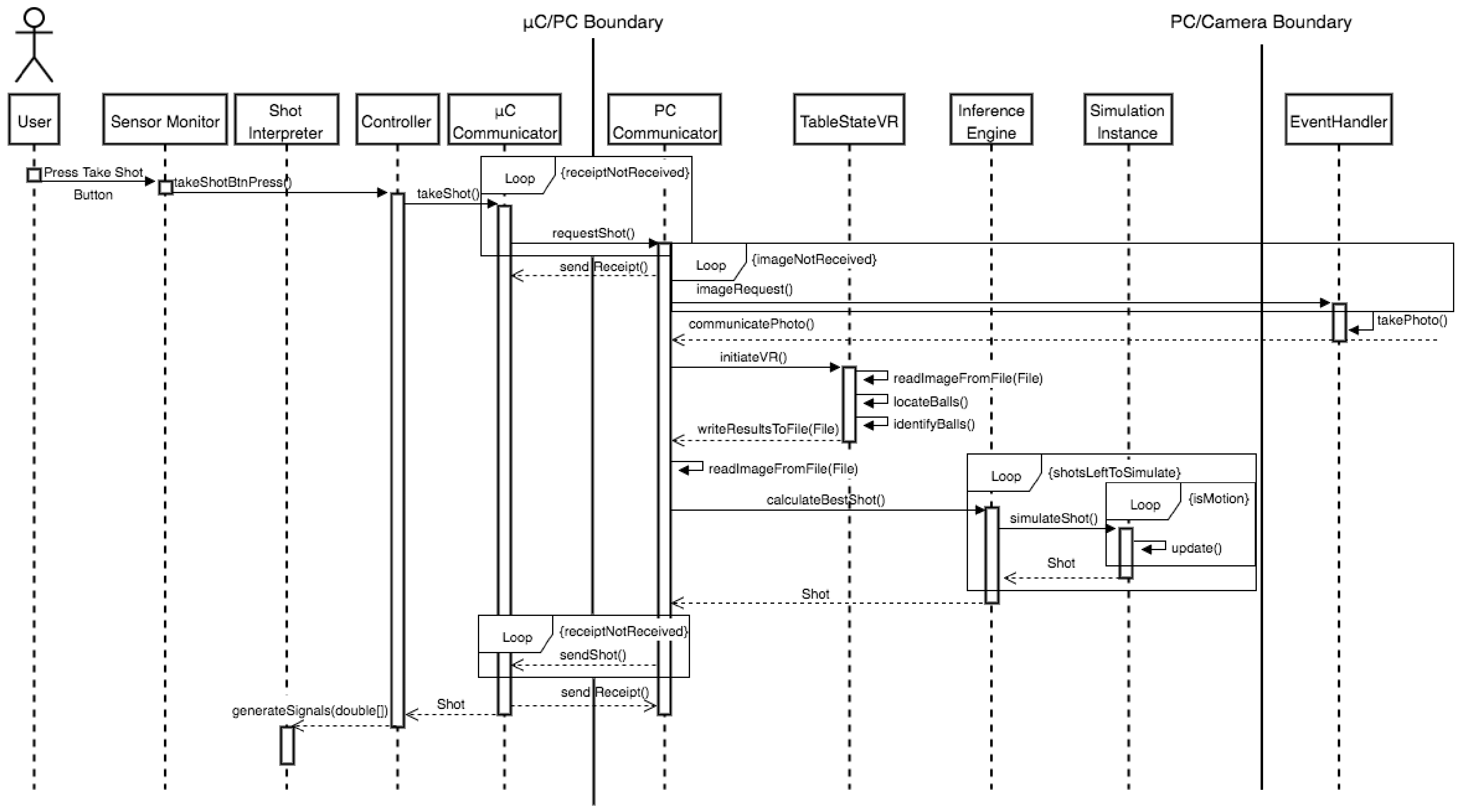


Figure 18: A sequence diagram for the “take shot” operation.