Autonomous Pool Playing Robot

Low-Level Software Design

Eric Le Fort
leforte@mcmaster.ca
1308609

January 21, 2017

Contents

List of Tables

List of Figures

Date	Revision #	Comments	Authors
25/12/2016	0	- Initial document creation	Eric Le Fort
14/01/2017	0	- First draft completion	Eric Le Fort
17/01/2017	0	- Added Physics and Communication sections	Eric Le Fort

Table 1: Revision History

1 Introduction

This document will outline the low-level software design for an autonomous pool-playing robot. 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

A system description can be found in the *High-Level Software Design* document for this system.

1.2 Overview

This document will begin by providing a detailed class diagram of the classes in the system. Then, each module will be covered in more detail such as the module's responsibilities, secrets, Interface Specification (MIS), and Internal Design (MID). Lastly, the document will discuss the scheduling of tasks and provide state charts and sequence diagrams to help illustrate the scheduling.

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 Detailed Class Diagram

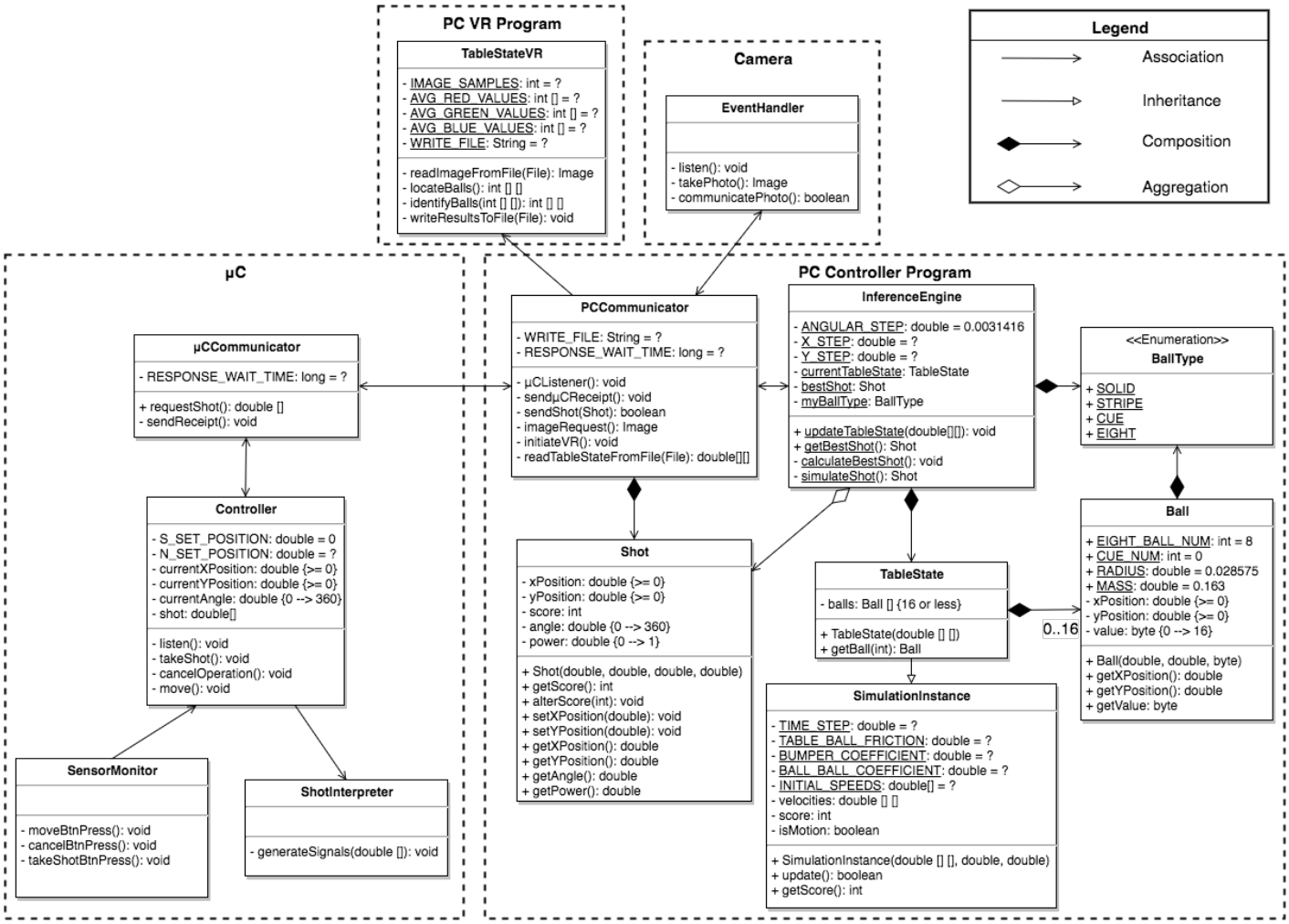

Figure 1: The system's detailed class diagram.

3 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.

3.1 Camera Modules

The following is the module contained within the Camera subsystem.

3.1.1 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.

3.2 PC VR Program Modules

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

3.2.1 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;
```

3.3 PC Controller Modules

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

3.3.1 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);
```

3.3.2 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.

3.3.3 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;
        }
}
```

3.4 μC Modules

The following are the modules contained within the μC subsystem.

3.4.1 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.

3.4.2 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.

3.4.3 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.

3.4.4 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.

4 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).

4.1 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

4.2 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

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

5 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.

5.1 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.

5.2 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.

5.3 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.

6 Scheduling of Tasks

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

6.1 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.

6.2 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.

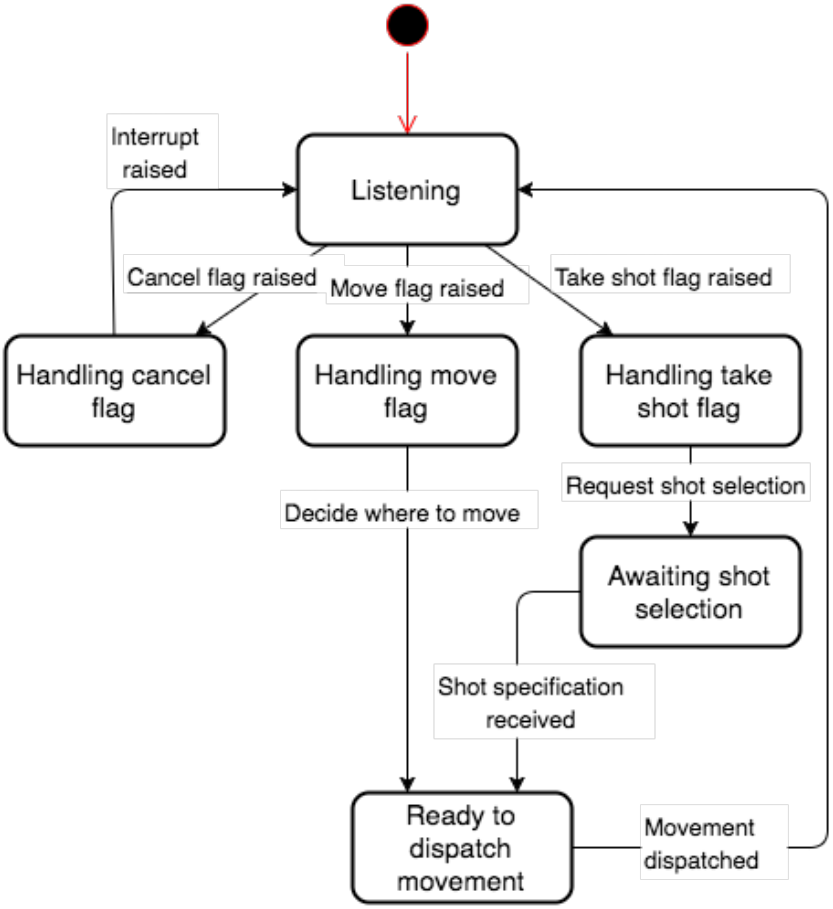

Figure 2: A state chart for the Controller class.

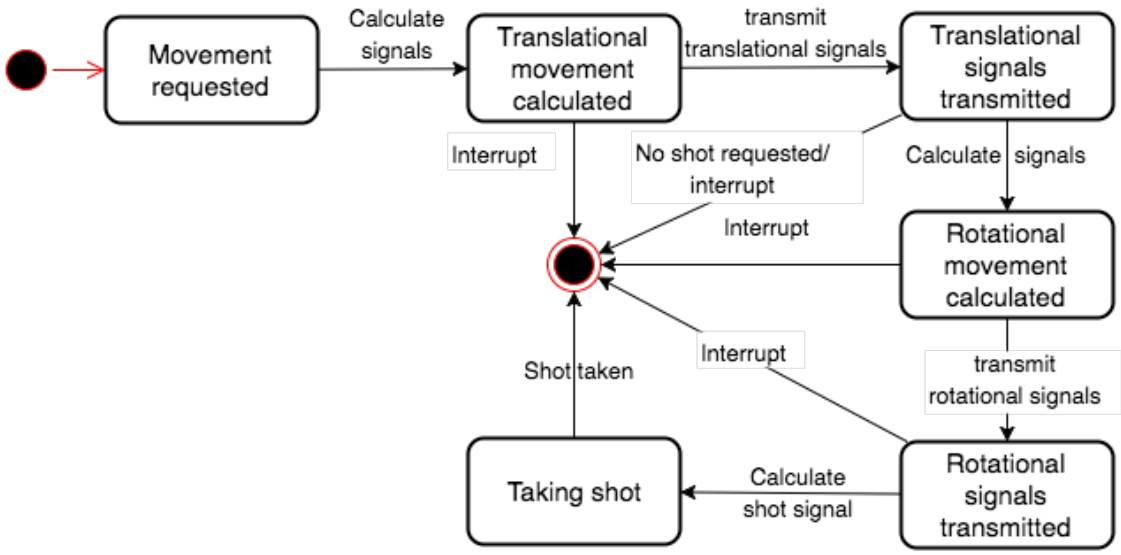

Figure 3: A state chart for the ShotInterpreter class.

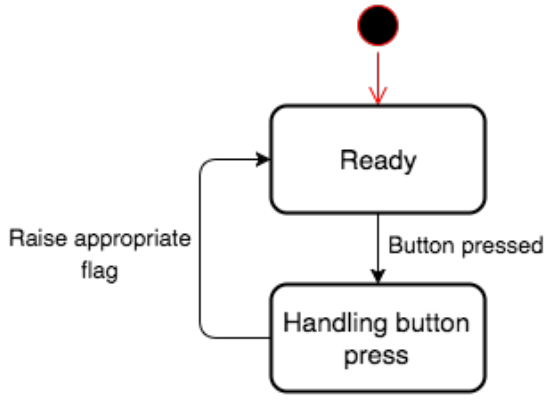

Figure 4: A state chart for the SensorMonitor class.

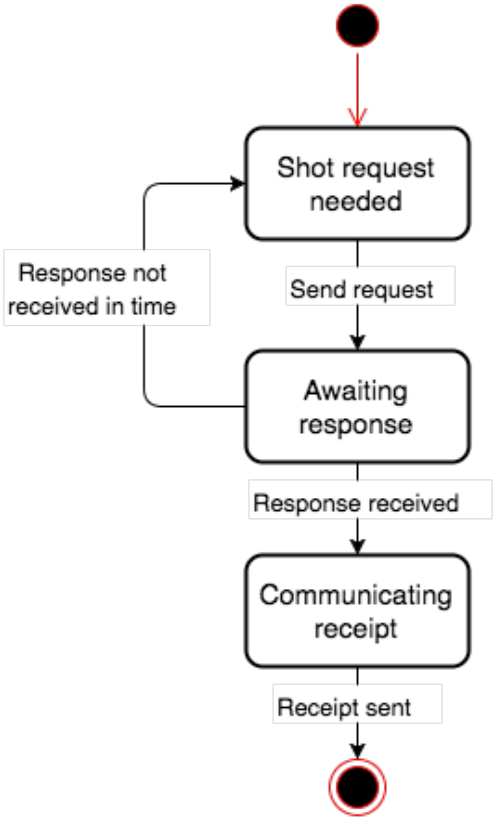

Figure 5: A state chart for the μCCommunicator class.

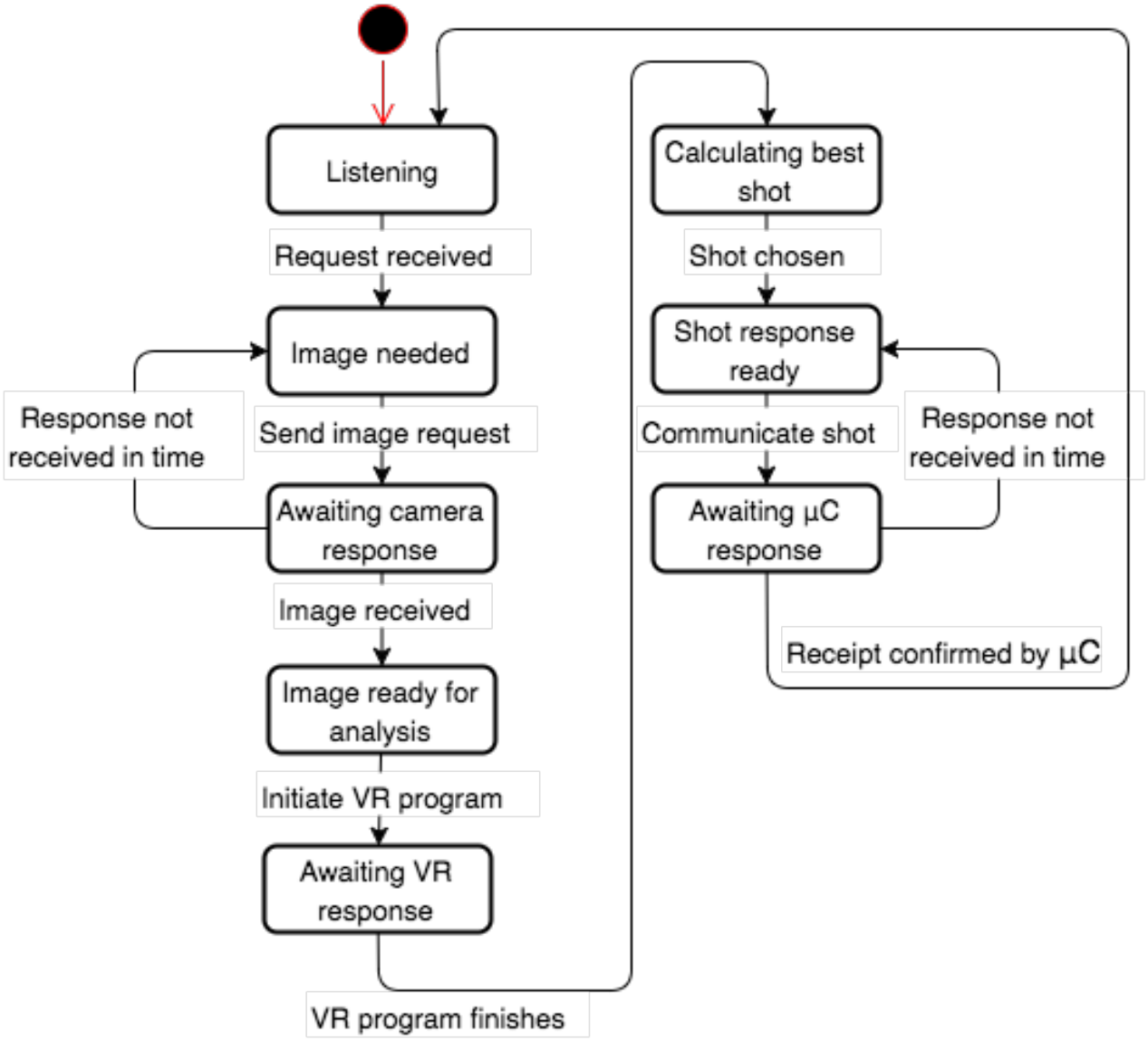

Figure 6: A state chart for the PCCommunicator class.

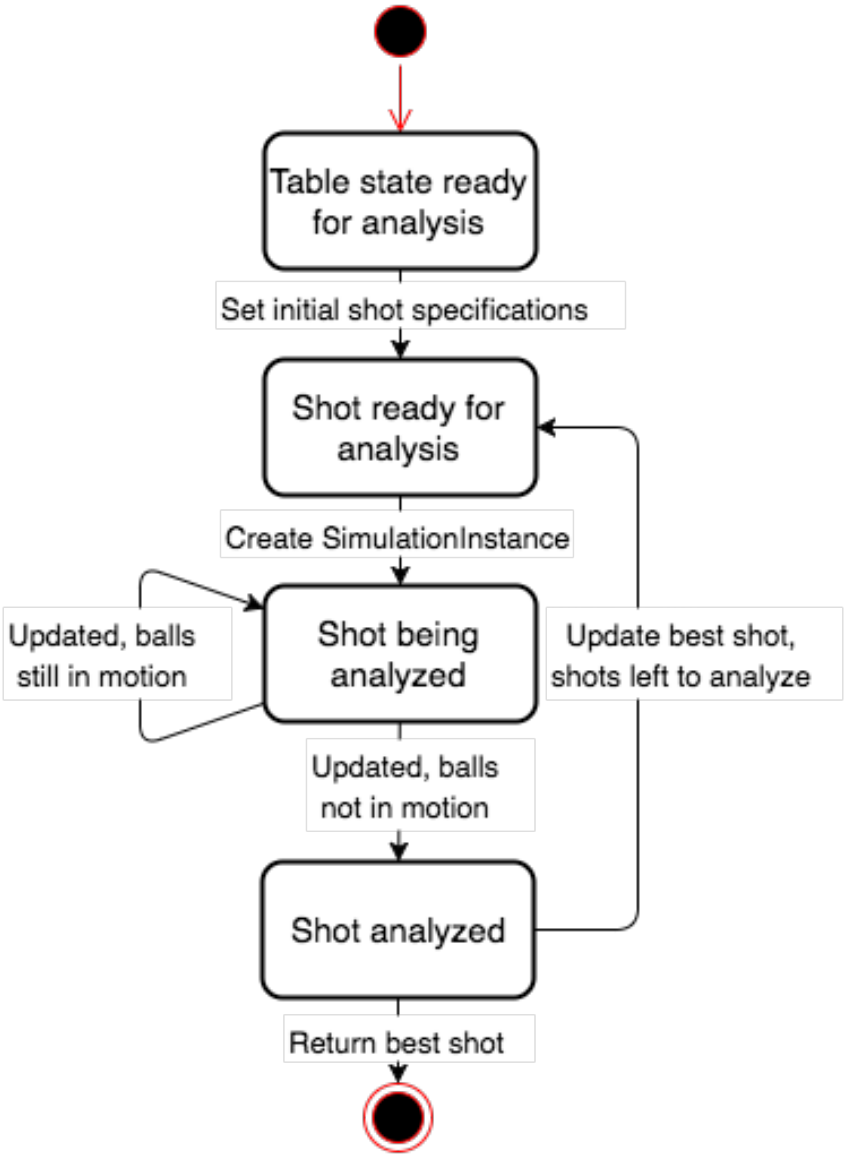

Figure 7: A state chart for the InferenceEngine class.

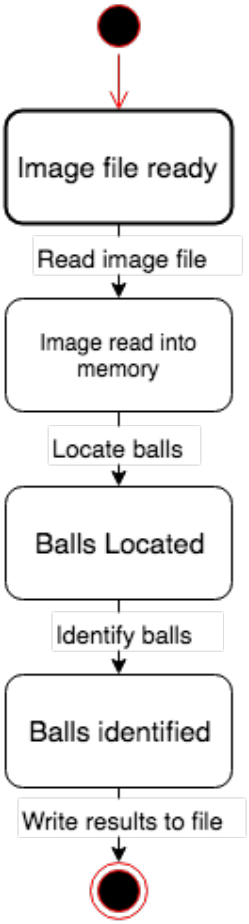

Figure 8: A state chart for the TableStateVR class.

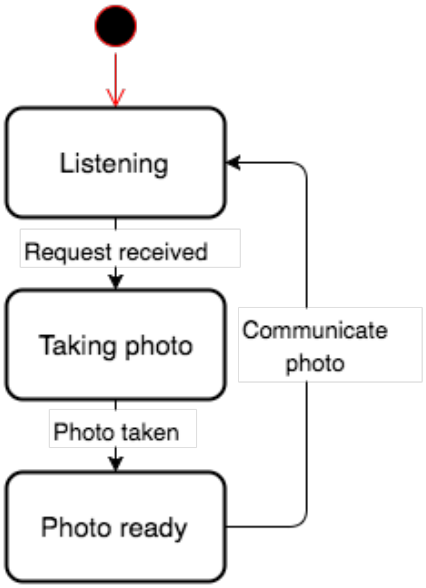

Figure 9: A state chart for the EventHandler class.

6.3 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.

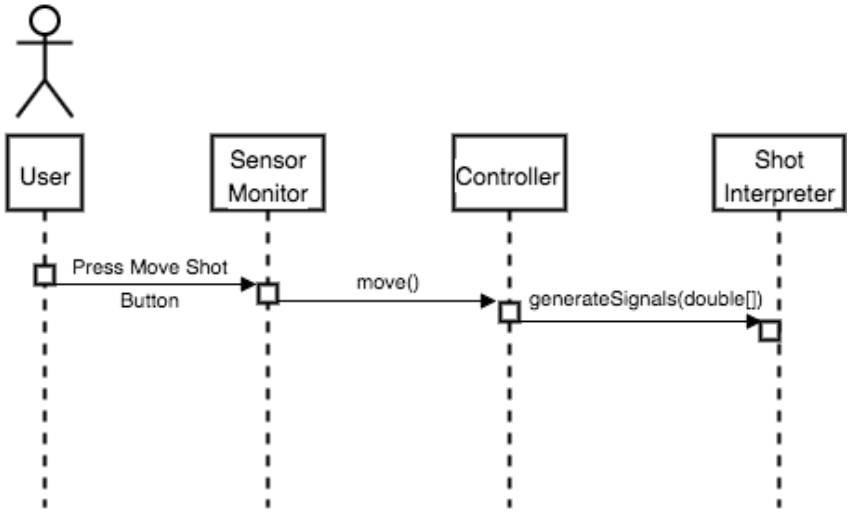

Figure 10: A sequence diagram for the "move" operation.

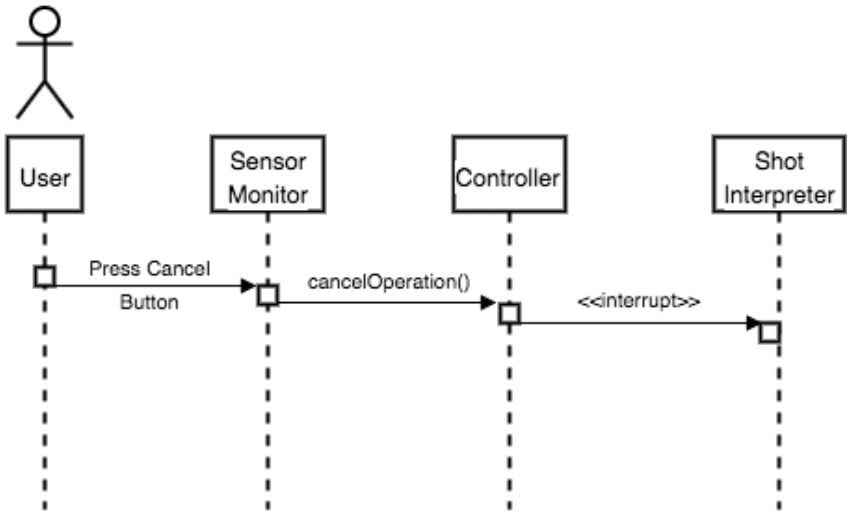

Figure 11: A sequence diagram for the "cancel" operation.

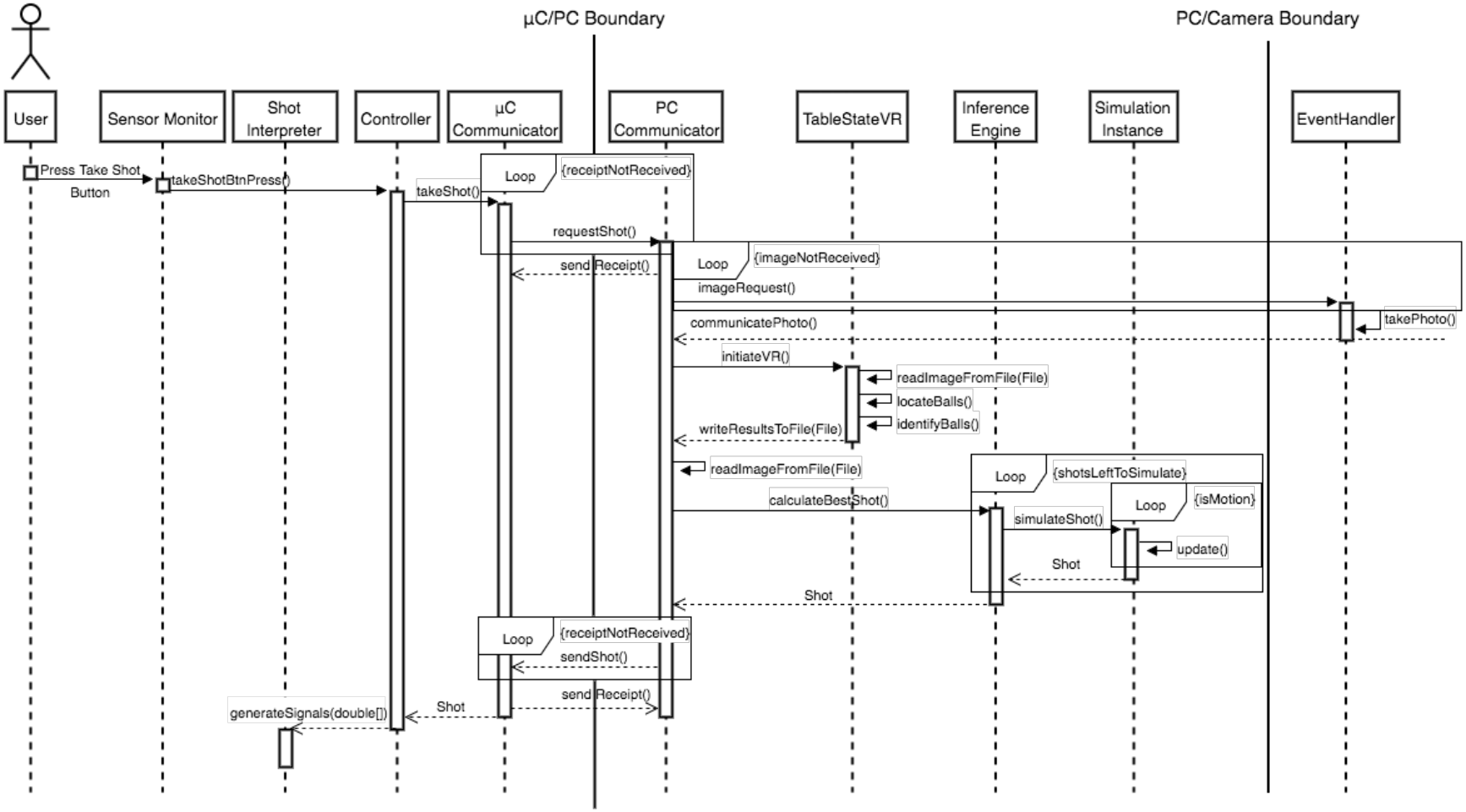

Figure 12: A sequence diagram for the "take shot" operation.