Autonomous Pool Playing Robot

Low-Level Software Design

Eric Le Fort
leforte@mcmaster.ca
1308609

Max Moore
moorem8@mcmaster.ca
1320009

January 12, 2017

Contents

List of Tables

List of Figures

Date	Revision #	Comments	Authors
25/12/2016	0	- Initial document creation	Eric Le Fort

Table 1: Revision History

1 Introduction

This document will outline the low-level software design for a 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 /textitHigh-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

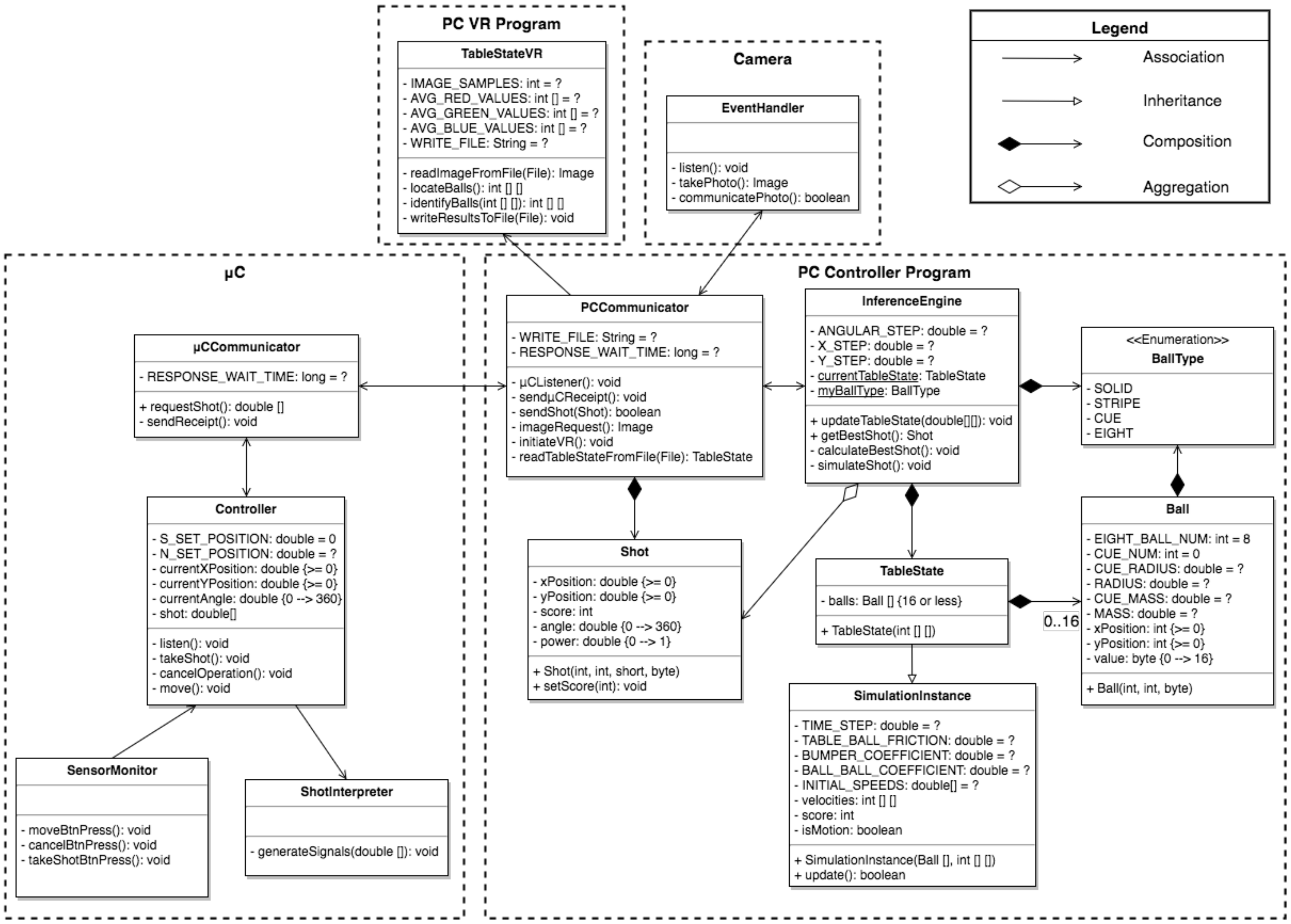
PC VR Program
TableStateVR
- IMAGE_SAMPLES: int = ?
- AVG_RED_VALUES: int [] = ?
- AVG_GREEN_VALUES: int [] = ?
- AVG_BLUE_VALUES: int [] = ?
- WRITE_FILE: String = ?
- readImageFromFile(File): Image
- locateBalls(): int [] []
- identifyBalls(int [] []): int [] []
- writeResultsToFile(File): void
Camera
EventHandler
- listen(): void
- takePhoto(): Image
- communicatePhoto(): boolean
Legend
Association
Inheritance
Composition
Aggregation
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 Controller Program
PCCommunicator
- WRITE_FILE: String = ?
- RESPONSE_WAIT_TIME: long = ?
- µCListener(): void
- sendµCReceipt(): void
- sendShot(Shot): boolean
- imageRequest(): Image
- initiateVR(): void
- readTableStateFromFile(File): TableState
InferenceEngine
- ANGULAR_STEP: double = ?
- X_STEP: double = ?
- Y_STEP: double = ?
- currentTableState: TableState
- myBallType: BallType
+ updateTableState(double[][]): void
+ getBestShot(): Shot
- calculateBestShot(): void
- simulateShot(): void
<<Enumeration>>
BallType
- SOLID
- STRIPE
- CUE
- EIGHT
Shot
- xPosition: double {>= 0}
- yPosition: double {>= 0}
- score: int
- angle: double {0 --> 360}
- power: double {0 --> 1}
+ Shot(int, int, short, byte)
+ setScore(int): void
TableState
- balls: Ball [] {16 or less}
+ TableState(int [] [])
0..16
Ball
- EIGHT_BALL_NUM: int = 8
- CUE_NUM: int = 0
- CUE_RADIUS: double = ?
- RADIUS: double = ?
- CUE_MASS: double = ?
- MASS: double = ?
- xPosition: int {>= 0}
- yPosition: int {>= 0}
- value: byte {0 --> 16}
+ Ball(int, int, byte)
SimulationInstance
- TIME_STEP: double = ?
- TABLE_BALL_FRICTION: double = ?
- BUMPER_COEFFICIENT: double = ?
- BALL_BALL_COEFFICIENT: double = ?
- INITIAL_SPEEDS: double[] = ?
- velocities: int [] []
- score: int
- isMotion: boolean
+ SimulationInstance(Ball [], int [] [])
+ update(): boolean

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)

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. This shot is accessible to other classes.

MID

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

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

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

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 Scheduling of Tasks

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

4.1 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.

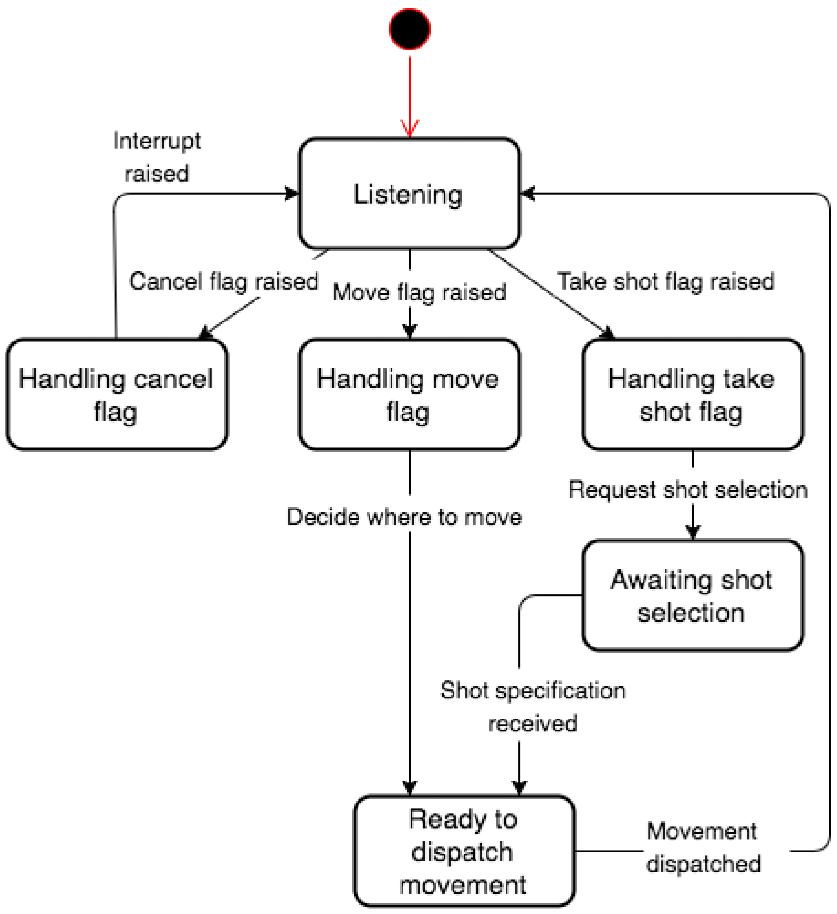

Figure 2: A state chart for the Controller class.

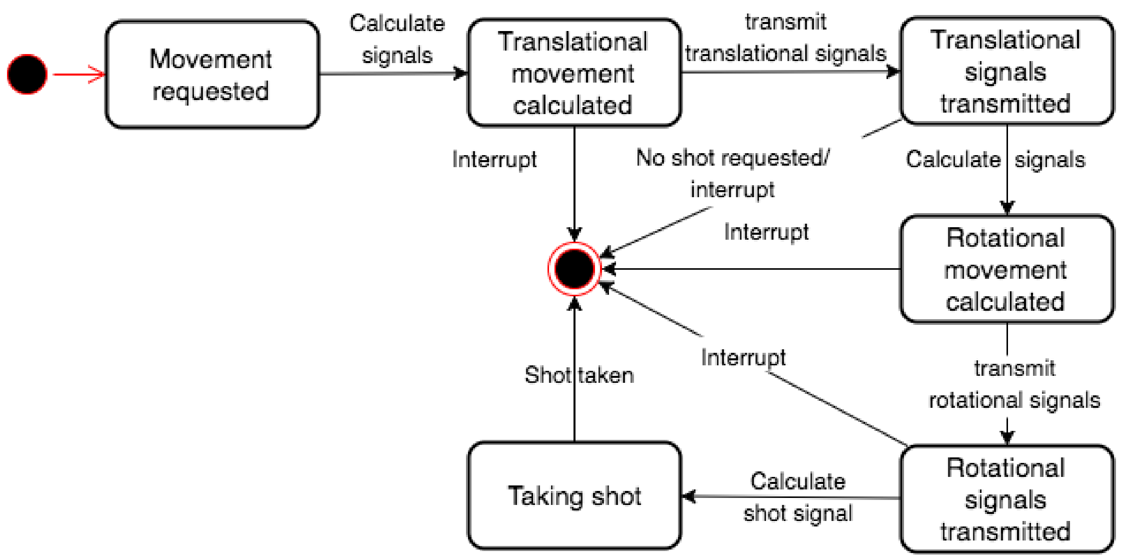
Movement requested
Calculate signals
Translational movement calculated
transmit translational signals
Translational signals transmitted
Interrupt
No shot requested/ interrupt
Calculate signals
Interrupt
Rotational movement calculated
Interrupt
Shot taken
transmit rotational signals
Taking shot
Calculate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

Figure 3: A state chart for the ShotInterpreter class.

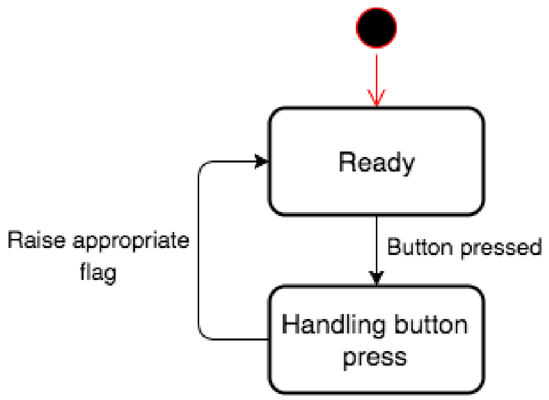

Figure 4: A state chart for the SensorMonitor class.

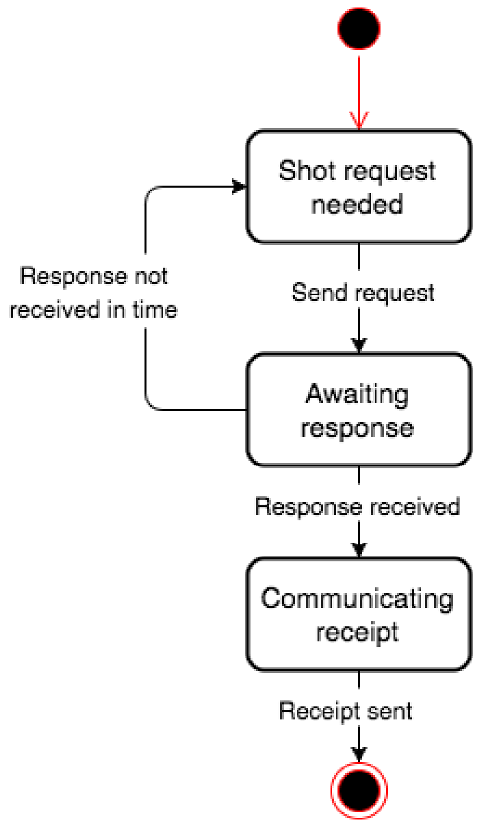

Figure 5: A state chart for the μCCommunicator class.

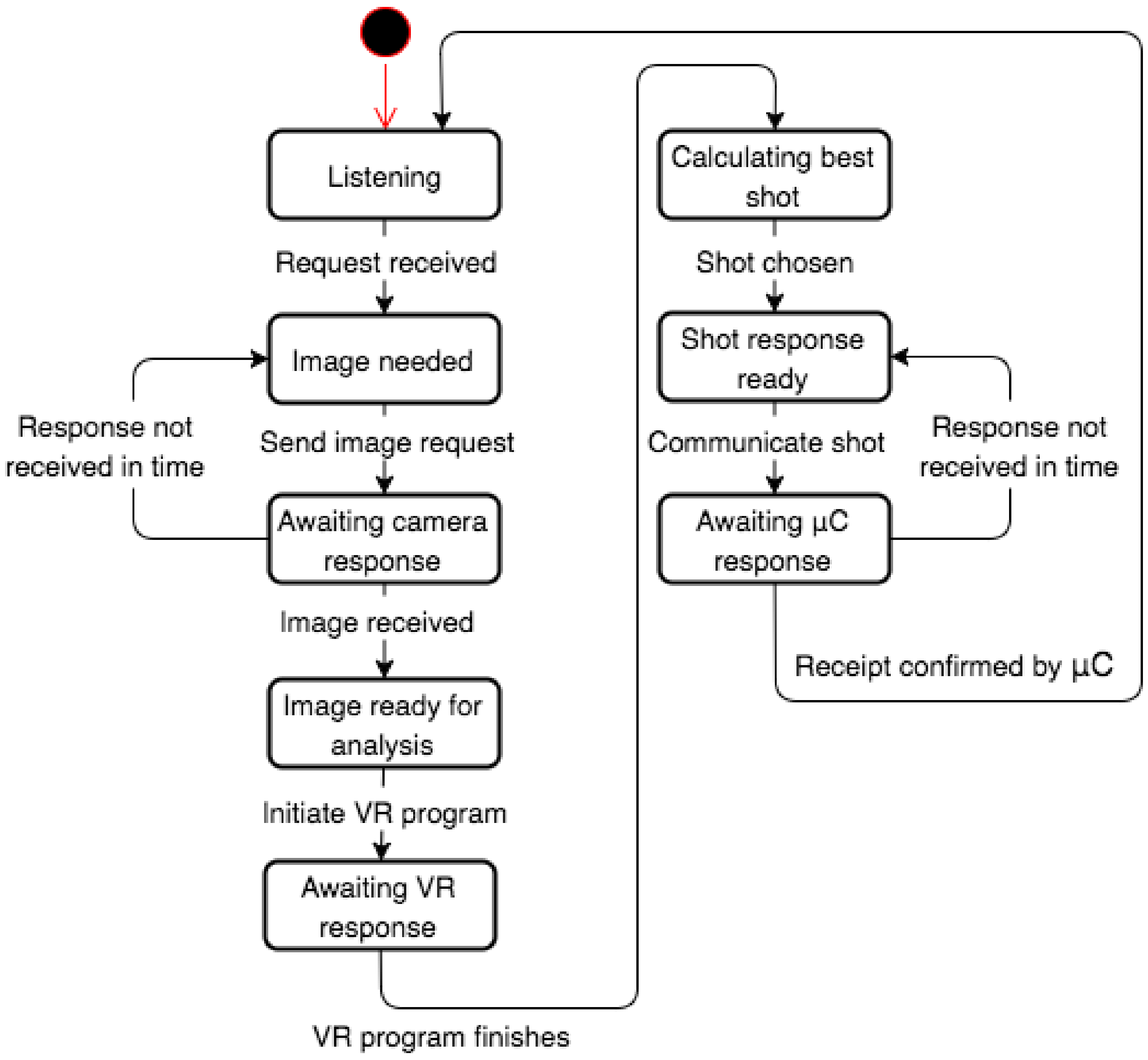

Figure 6: A state chart for the PCCommunicator class.

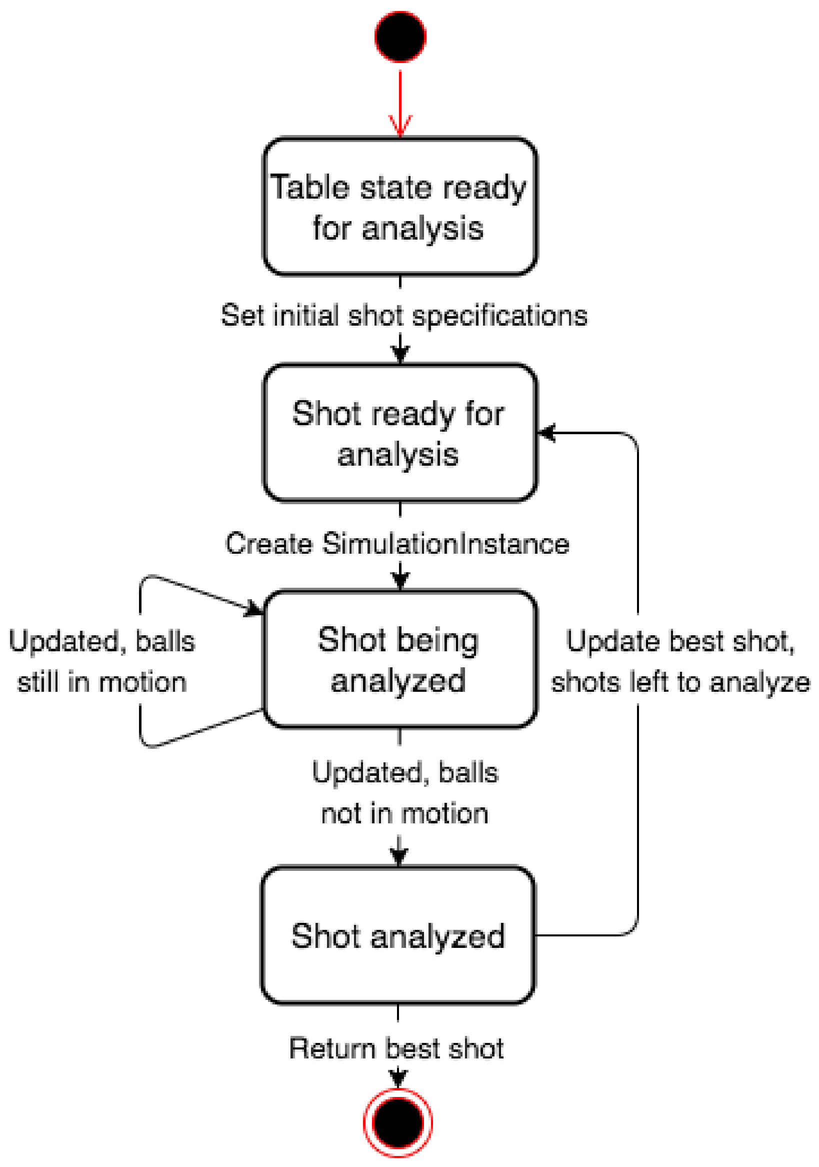
Table state ready for analysis
Set initial shot specifications
Shot ready for analysis
Create SimulationInstance
Updated, balls still in motion
Shot being analyzed
Update best shot, shots left to analyze
Updated, balls not in motion
Shot analyzed
Return best shot

Figure 7: A state chart for the InferenceEngine class.

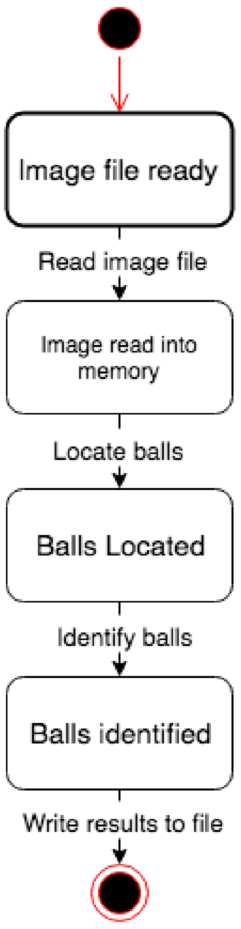

Figure 8: A state chart for the TableStateVR class.

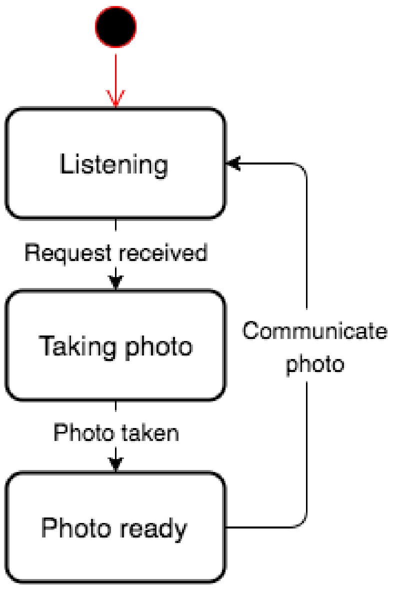

Figure 9: A state chart for the EventHandler class.

4.2 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.

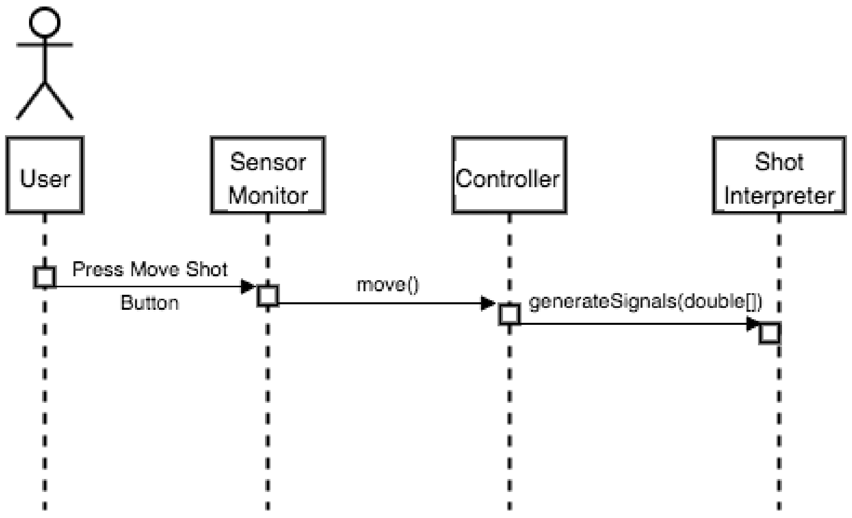

Figure 10: A sequence diagram for the "move" operation.

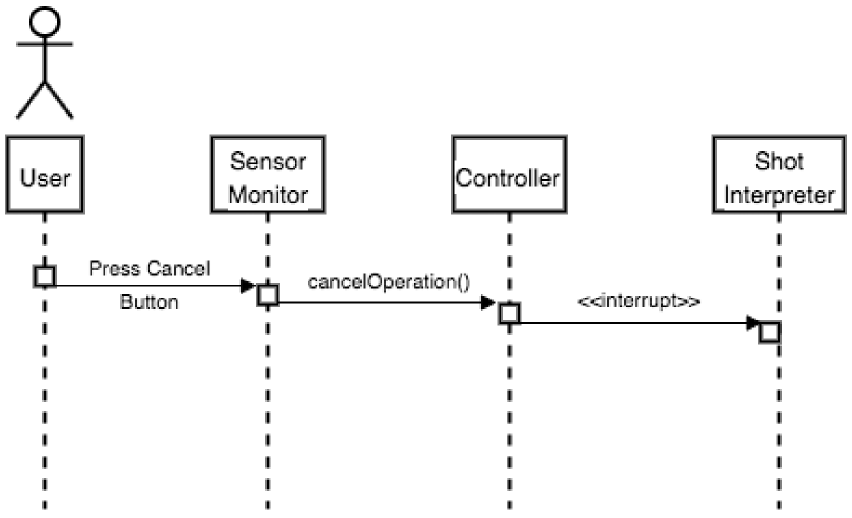

Figure 11: A sequence diagram for the "cancel" operation.

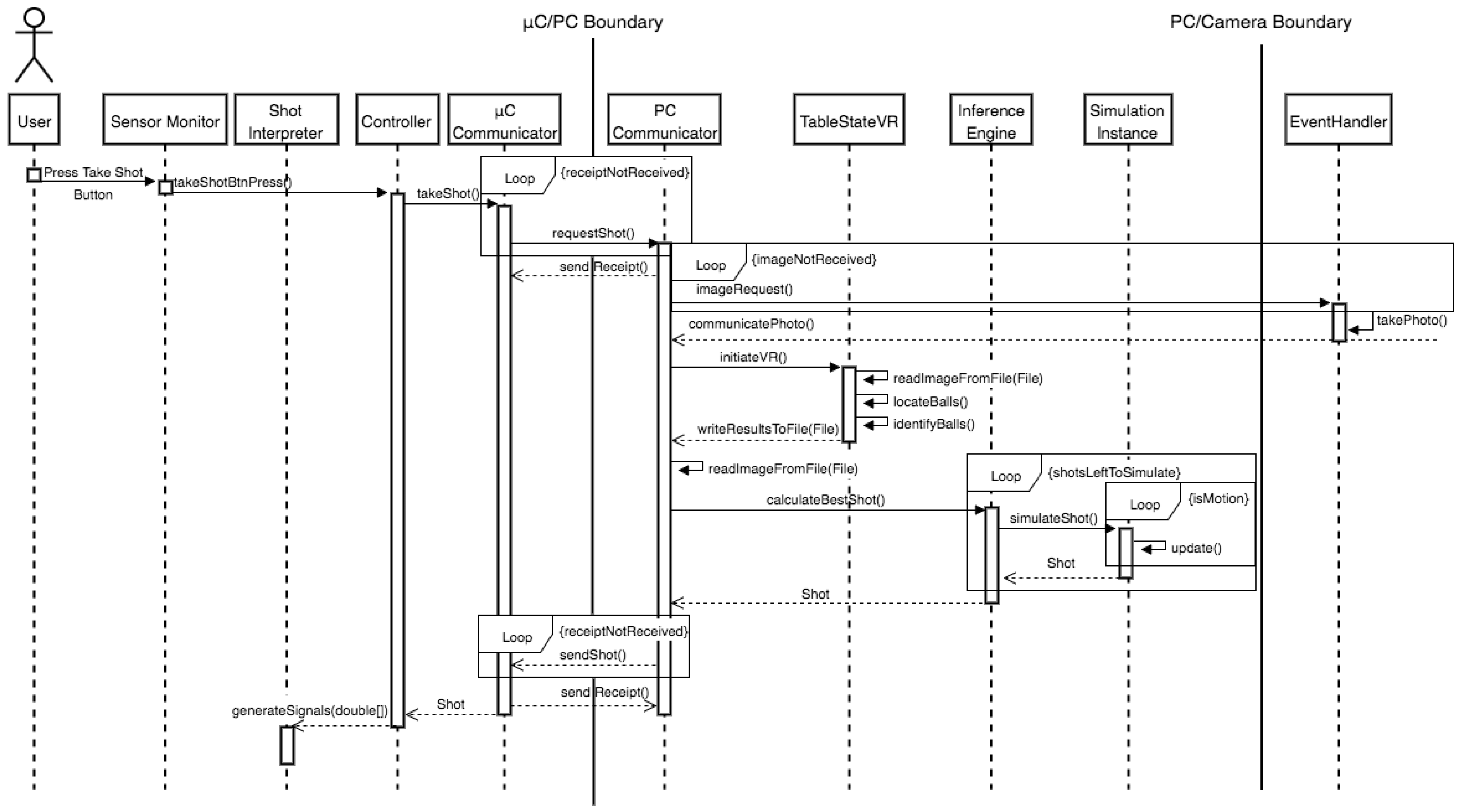

Figure 12: A sequence diagram for the “take shot” operation.