Autonomous Pool Playing Robot

High-Level Architectural Design

Eric Le Fort
leforte@mcmaster.ca
1308609

Max Moore
moorem8@mcmaster.ca
1320009

November 28, 2016

Contents

List of Tables

List of Figures

Date	Revision #	Comments	Authors
14/11/2016	0	- Initial document creation	Eric Le Fort Max Moore

Table 1: Revision History

1 Introduction

This document's purpose is to describe the architecture 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.

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.

1.2 Overview

This document has four 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 subprograms.

- **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.

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

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.

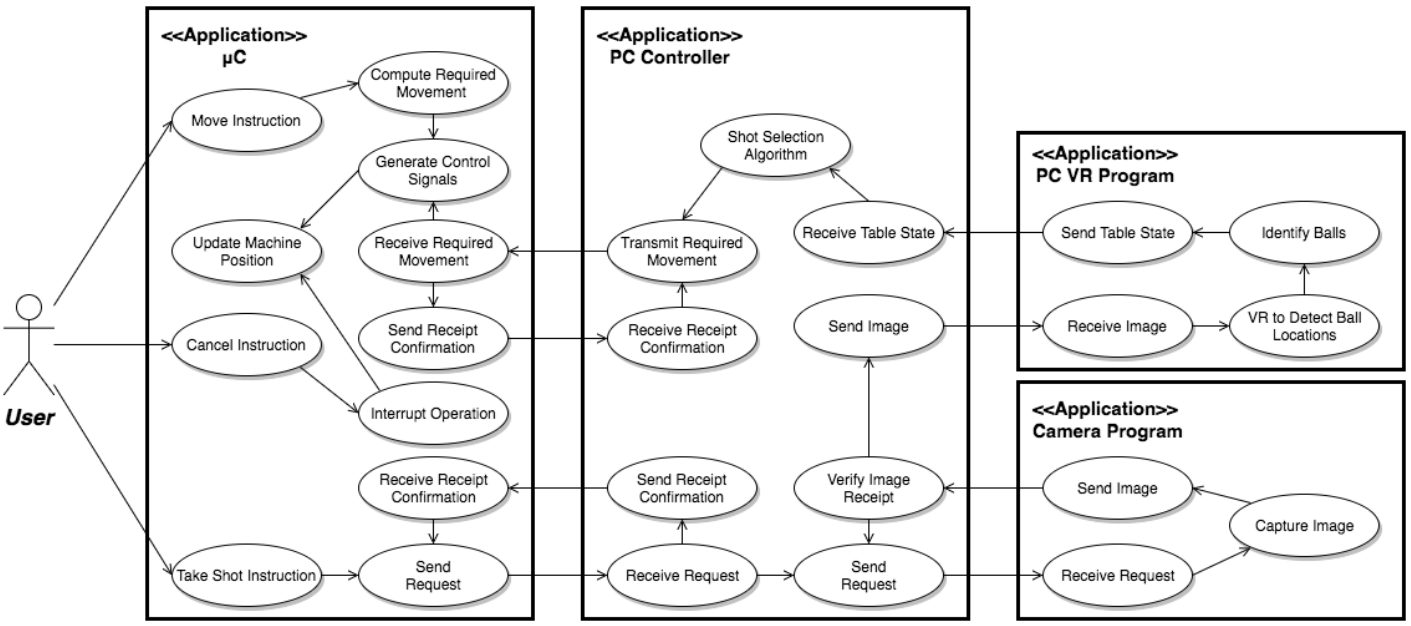

Figure 1: Use Case Diagram

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

3.1 μC Analysis Class Diagram

3.2 PC Controller Analysis Class Diagram

3.3 Camera Analysis Class Diagram

3.4 PC VR Program 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:

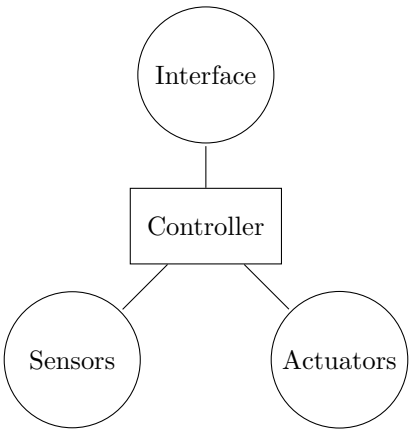

Figure 2: Sensor-Controller-Actuator Architecture

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:

$$\text{in} \xrightarrow{\text{image}} \text{object detection} \xrightarrow{\substack{\text{image \&} \\ \text{object positions}}} \text{ball identification} \xrightarrow{\text{table state}} \text{out}$$

Figure 3: PC VR Architecture

5 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.

5.1 μC CRC Cards

The following are the CRC cards for the μC program:

5.2 PC Controller CRC Cards

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

Communicator	
Responsibilities	Collaborators
- Listens for request from μC	-
- Sends confirmation of receipt to μC	-
- Sends shot specification to μC	-
- Listen for receipt confirmation from 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)

5.3 Camera CRC Cards

The following is the CRC card for the Camera program:

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

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

5.4 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)