McMaster University

SmartServe

Software & Mechatronics Capstone

High Level System Design

Authors:
Christopher McDonald - *001312456*
Harit Patel - *001317372*
Janak Patel - *001307060*
Jared Rayner - *001311702*
Nisarg Patel - *001322805*
Sam Hamel - *001321692*
Sharon Platkin - *001316625*

Professor:
Dr. Alan Wassyng

Teaching Assistants:
Bennett Mackenzie
Nicholas Annable
Stephen Wynn-Williams
Viktor Smirnov

Last compiled on March 18, 2018

Contents

List of Figures

Date	Revision	Comments	Author(s)
Dec 1, 2017	1.0	Main content done for all sections	Christopher McDonald
Dec 13, 2017	1.1	Corrected Document Overview	Nisarg Patel
Dec 14, 2017	1.2	Refined Project Scope & System Description (Section 2)	Nisarg Patel
Dec 17, 2017	1.3	Added CRC card intro and reviewed	Christopher McDonald
Dec 21, 2017	1.4	Edited entire document	Sharon Platkin
Dec 21, 2017	1.5	Added Arduino Diagrams and Shooter Information	Nisarg Patel
Mar 9, 2018	2.0	Added Likely / Unlikely Changes and Assumptions, Removed Shot Optimizer	Christopher McDonald
Mar 9, 2018	2.1	Added motor specifications	Janak Patel
Mar 10, 2018	2.2	Added details on CV and SR, fixed Start Sequence Diagram	Christopher McDonald
Mar 10, 2018	2.3	Fixed CRC cards for changes to requirements	Christopher McDonald
Mar 18, 2018	2.4	Converted to overall design, added in low level / component details	Christopher McDonald

Figure 1: Revision History

1 Introduction

1.1 Project Overview

SmartServe is an autonomous table tennis training system for table tennis players with various skill levels. SmartServe aids in diagnosing and improving a player's performance over time. The system trains table tennis players by shooting table tennis balls towards the player and detects successful returns from the player. The system can further adapt to the player's weaknesses and help them overcome it through further training. Importantly, SmartServe alleviates the problems of finding and working with a coach for players, as well as coaches trying to train multiple players simultaneously. The system will be deemed a success if the table tennis players and coaches can enjoy and see some value added by using SmartServe.

The project started at the beginning of the Fall 2017 academic term and will conclude at the end of the Winter 2018 term. In addition, the core project team consists of final year Software and Mechatronics Engineering students who are enrolled in the MECHTRON 4TB6/SFWRENG 4G06 capstone project course.

1.2 Document Overview

The purpose of this document is to provide an overview of the system design which meets the system requirements as specified in the Requirements Document. The system will be decomposed into subsystems, where each has responsibilities and is designed to fulfill certain requirements. The subsystems will have their intended input, expected output and description of how the module will be used. In further documentation, each will be designed in a way which is abstracted from this document's perspective. The expected use cases will also be detailed to understand the expectations of the user and how each subsystem interacts with one another. A more detailed view of this including timing as a factor will be detailed in the Sequence Diagram section.

For each subsystem, its purpose will be defined with respect to the overall system. After doing so, detailed input and output parameters will be defined. How each subsystem is architected is out of scope for this document and will be defined in further documentation.

1.3 Naming Conventions and Terminology

The following terms and definitions will be used throughout this document:

- **ACID**: a database transaction which is atomic, consistent, isolated and durable

- **CV**: computer vision
- **FPS**: frames per second
- **FSM**: finite state machine, shows transitions between states
- **GUI**: graphical user interface
- **IPO**: input process output
- **Pitch**: rotation along the y-axis; this rotation angle primarily dictates the range of the ball from the net to the edge of the table on the user side
- **Roll**: rotation along the x-axis
- **Shooting Mechanism**: refers to the part of the system that shoots the table tennis balls towards the user side (player) Please refer to Figure 2 for visual illustration
- **System**: encompasses both the hardware and software parts of SmartServe
- **System Side**: the side of the table where the electromechanical system is placed; it is the opposite side of the User Side Please refer to Figure 2 for visual illustration
- **TCP:** transmission control protocol
- **Team**: all team members of the core capstone project, as noted in the list of Authors
- **User Side**: the side of the table where the user (player) is standing
- **Yaw**: rotation along the z-axis; this rotation angle primarily dictates the panning functionality of the shooting mechanism from the right side to the left side of the table

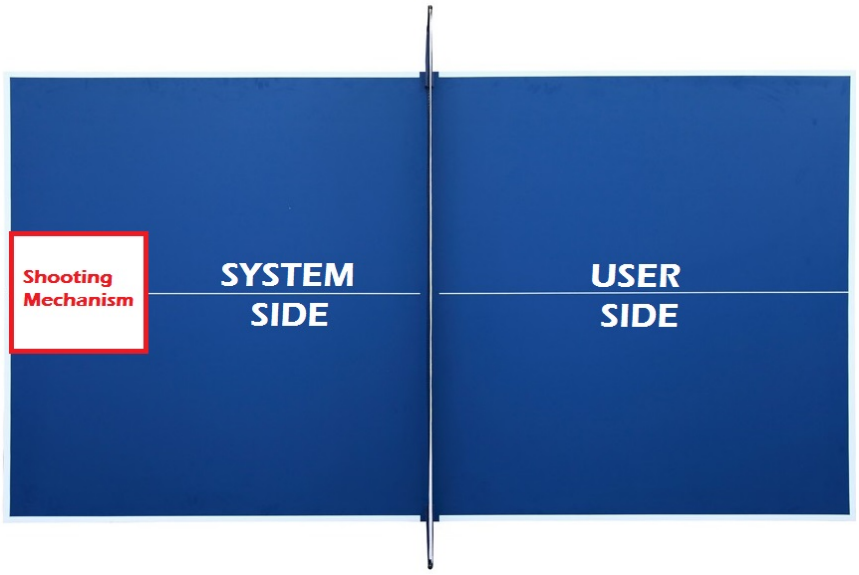

Figure 2: Top View of the Tennis Table

1.4 Project Scope

The system will only attempt to shoot balls from the shooting mechanism straight towards the user side. Notably, the system will not attempt to return any shots from the user. After the user has returned a shot, the Computer Vision (CV) subsystem will be utilized to determine if the user's shot lands on the table or not. Additionally, the characteristics of the shot following any return will be determined by the system's mode and the proficiency of the player if available.

1.5 Assumptions

This design assumes the user supplies a functioning webcam, a capable computer to run the software and can connect to the SmartServe system via USB. The computer should have proficient computing power, as the CV subsystem will suffer in proficiency as a result. It is also assumed the use cases described here cover all interactions the user will attempt to have with the system. We assume the user has a ITTF-approved table, an orange table tennis ball and a paddle to use.

2 System Description

2.1 System Architecture

The system will follow a service-oriented architecture. This means that a central subsystem will interface with several services that serve a single purpose. Some subsystems will be simply told what needs to be done and others will be asked for some return value. This is done to implement separation of concerns where one subsystem doesn't know everything about the system and only what is necessary to satisfy their requirements. It also allows for easy increments of versioning, where one subsystem can be used as long as it is functional and easily swapped out for a newer version with extra features or increased performance. Moreover, this system has heavy timing constraints and an unpredictable environment so some actions must be taken in absence of a service's response. For example, the computer vision may take longer to track a ball depending on its trajectory where the system must shoot another ball in order to keep the user engaged.

2.2 Subsystems

The system will be broken down into subsystems with the following purposes:

- **SmartServe**: general management of subsystems
- **Computer Vision**: detection of returns
- **Shooting Mechanism**: shooting the ball toward the user
- **Shot Recommendation**: provides best shot
- **Data Storage**: storing the data
- **Shooting Model**: modelling the shot's trajectory
- **User Interface**: taking input from the user and showing output

The diagram for this breakdown can be found in Figure 3.

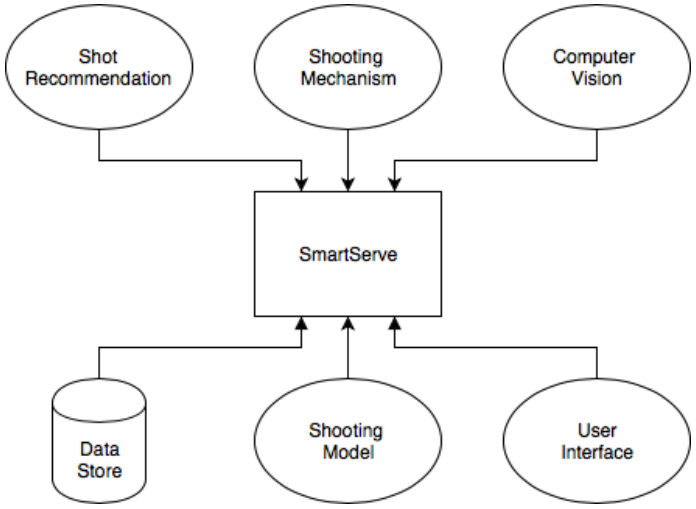

Figure 3: Subsystem Breakdown

2.3 Use Cases

The diagram including all use cases can be found in Figure 4. The user-instantiated ones will be described in detail below.

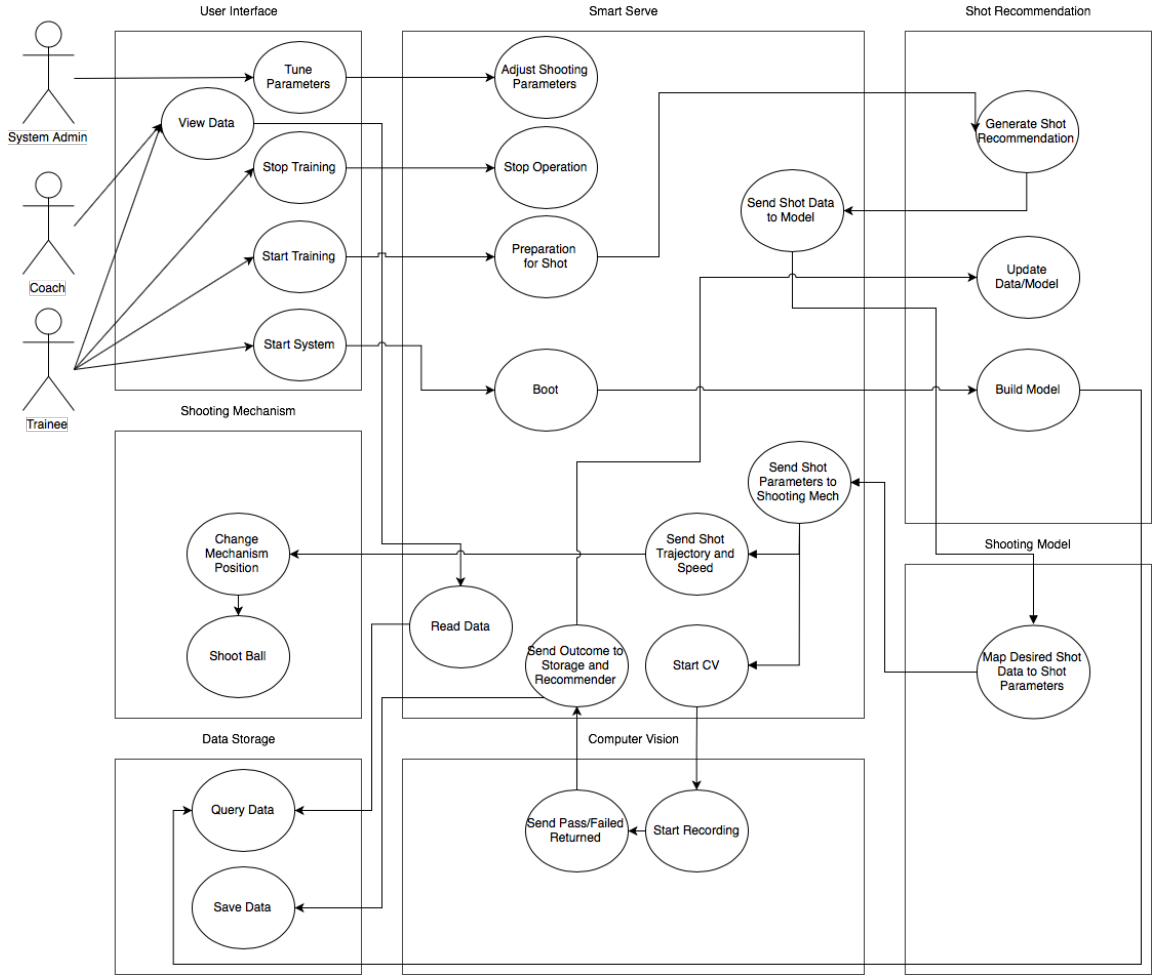

Figure 4: Use Case Diagram

2.3.1 Start Training

The user interface will have ways to allow the user to start this action which makes the Smart Serve subsystem prepare for a shot to shoot towards the user. To do this, it needs to request a shot to use from the Shot Recommendation subsystem. The shot includes the following parameters: desired location on the table, the speed of the shot and the angular velocity of the ball. The Smart Serve subsystem can then use the Shooting Model to translate this information into pitch, yaw and angular velocity to shoot the ball so it matches the desired shot. After doing so, it will instruct the shooting mechanism to shoot the ball with the appropriate parameters and start the CV subsystem to begin tracking for a successful return. Only until the CV subsystem returns the pass or failed return

data, can it update the Data Storage and Shot Recommendation subsystems with this new information. The former will store the result and the data for the shot together, where the latter will update the model used to generate shot recommendations. Upon completion, it can begin preparing for a new shot.

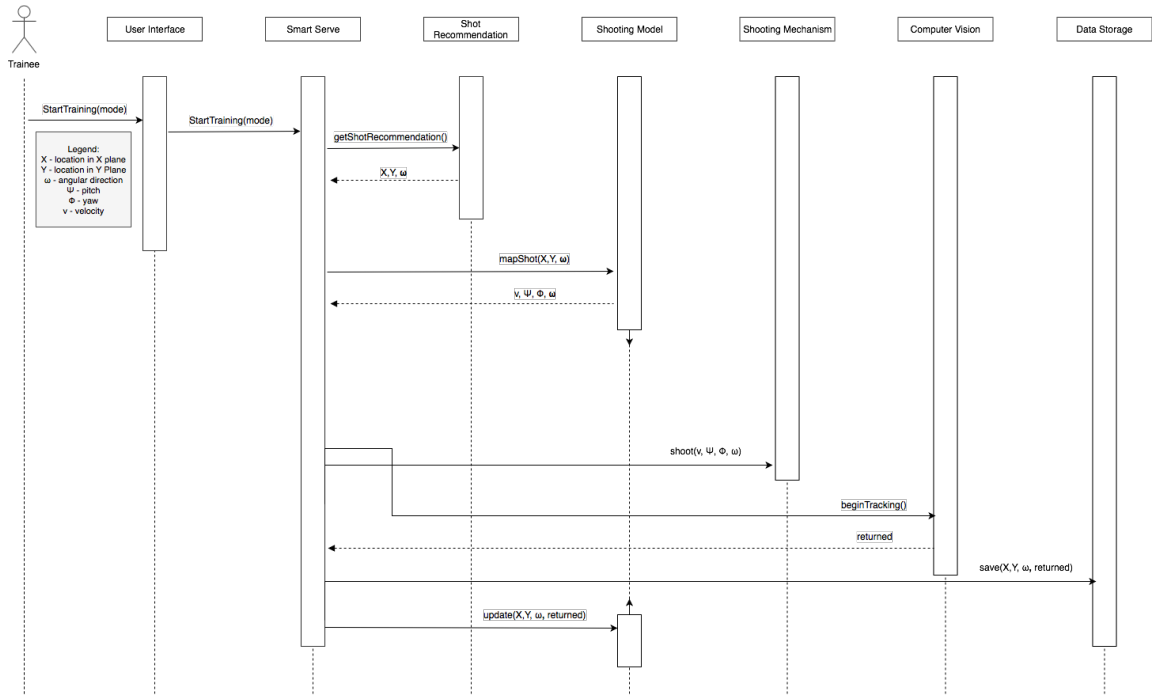

Figure 5: Sequence Diagram for Starting Training

2.3.2 Stop Training

In the event the user wants to cease training, the user interface will allow this and will halt the system from shooting balls. No data should be written nor shots requested for the shooting mechanism to shoot in order to preserve the integrity of the data being gathered.

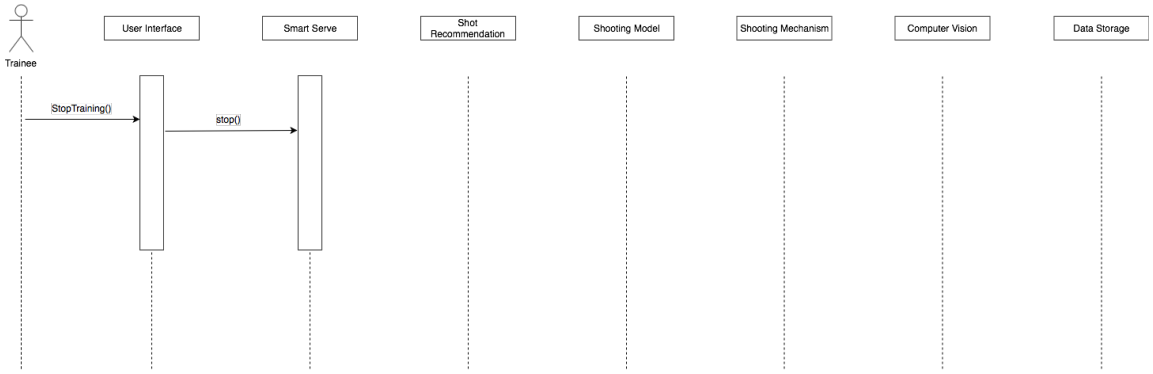

Figure 6: Sequence Diagram for Ceasing Training

2.3.3 View Results

When the user wants to visualize the results of their performance, the user interface should be used. It will use the Smart Serve subsystem to query data from Data Storage and present it in some meaningful way.

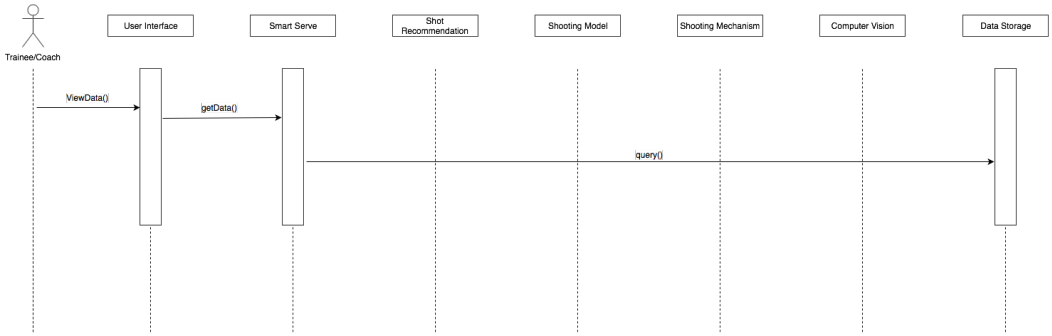

Figure 7: Sequence Diagram for Viewing Training Results

2.3.4 Tune Parameters

The system may need to be adjusted for various lighting, table sizes or environments. The user interface will allow a system administrator to do this in order to directly change values associated for these variables.

Figure 8: Sequence Diagram for Tuning Parameters

2.3.5 Start System

The user interface will allow the system to be booted which will start the Smart Serve subsystem. This will allow the Shot Recommendation system to build its model based on previous data for the user, if it exists.

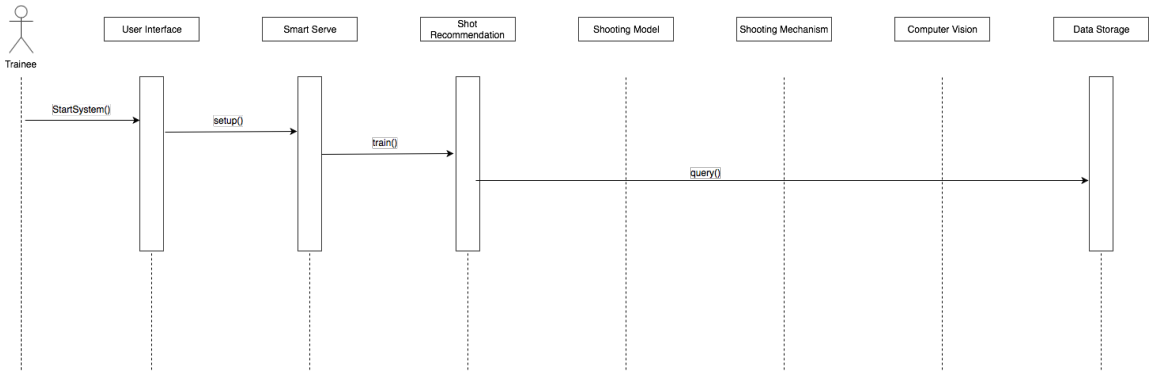

Figure 9: Sequence Diagram for Booting the System

2.4 Behaviour Description

As the behaviour of the system has been discussed previously, this section will describe the expected user behaviour and how they will interact with the system.

2.4.1 Normal Operation

The trainee would begin by starting the system from its off state. This will trigger the Start System use case. The user will then trigger the Start Training use case and specify a training mode. This creates the loop of Smart Serve prepping and serving shots for

the user. Only until the user triggers the Stop System use case will this loop stop during normal operation. During or after training a user can start the View Data use case to get details on their performance.

2.4.2 Abnormal Operation

Although possible, there are some actions a user could take which would be considered abnormal. If a user hits the ball and it immediately leaves the viewable area of the computer vision, whether it never returns or does so after 1.5 seconds, it should be considered a failed return. A user could also replace the table with a different one without calibrating the system to the new table. Furthermore, the user could misplace the system such that it is not centre with the table or at the edge of the table. The proper placement can be found in Figure 2. The system will require the user to load the balls into the shooting mechanism's hopper which will need to be a particular size and colour to work optimally.

2.4.3 Error Handling

If the system should encounter an error, it should display the details of it through the User Interface subsystem. In addition to this, it must take measures to ensure the safety of the user and integrity of the system. This includes stopping all training and pending actions for the system to perform. Examples of such errors include a jammed table tennis ball, and an empty mechanism ball hopper.

2.5 Likely Changes

Some aspects of the system design will likely change as the project progresses which will be excluded within this document. This is done so the design is considered complete and the next stages of design and implementation can be done. The first likely change would be to the computer vision subsystem, as the introduction of another camera would change the high level design of this subsystem. The additional camera would track a 3rd dimension, which is along the short side of the table. This change would propagate through the system since the Shot Recommendation subsystem would need to consider this new data for recommending new shots. The Data Storage subsystem will also be changed to hold the new data.

2.6 Unlikely Changes

The overall architecture of the system will not change, although additional subsystems could be added or existing ones could be changed. The degrees of freedom for a shot will not change. The Smart Serve subsystem will likely not change, although the data being processed between states may change.

3 Subsystems Overview

3.1 Smart Serve

The Smart Serve subsystem will provide the means of interfacing with all the subsystems and enforcing timing constraints. For all intents and purposes, it can be considered the main process and hub of the system. In the event a subsystem is taking too long to respond, this subsystem must be able to continue operation to meet timing requirements. The various states and transitions can be found in Figure 10. The FSM (Finite State Machine) diagram shows each state should have an *exit option* if a service takes longer than allowed to preform an action.

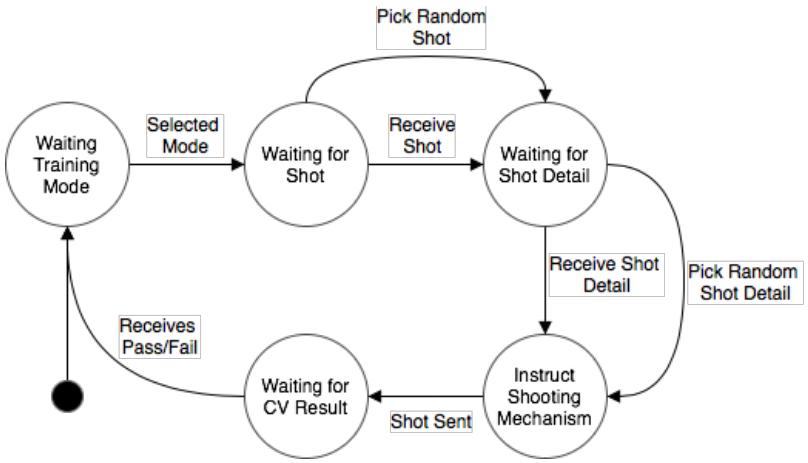

Figure 10: Smart Serve FSM

3.2 Computer Vision

The Computer Vision subsystem will be a service to the Smart Serve subsystem to determine if a shot is successfully returned. When sent a request to begin tracking the ball, it will do so and return a true when the ball bounces off the table or false when it doesn't. In the event the ball never enters frame, it will assume the return was failed after a fixed amount of time.

The FSM will be updated every frame captured within the CV subsystem. Once a frame is captured, the position of the ball will be compared against the previous position and a difference in position can be calculated. Based on how it moved, the state of the FSM will or will not change. Since the ball is being viewed on the System's side, a valid return is when it is moving towards the system, travels downward and then upward while still in frame. The states and transitions of the FSM represent this.

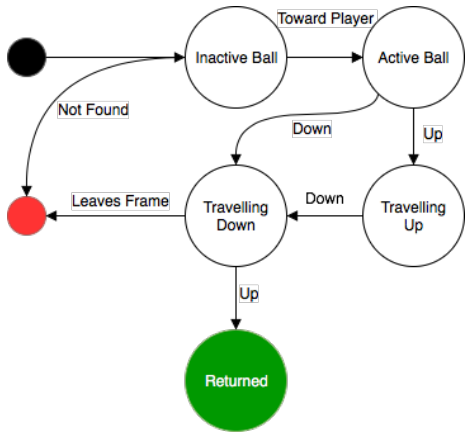

Figure 11: CV Finite State Machine

3.3 Shot Recommendation

The Shot Recommendation subsystem will be a service to the Smart Serve subsystem to determine which shot should be taken next. The way the subsystem decides on the shot will be determined by the training mode the user selects. These will include shooting the same shot every time, pseudo-randomly picking a shot or using reinforcement learning algorithms to decide the next shot. The last mode will be based on previous performance by the user.

The training mode will utilize a variation of a Upper Confidence Bound (UCB) algorithm. The subsystem will attempt to predict the actual return rate for each type of shot for a particular player by holding a range of values the actual return rate exists within. The initial state is from 0 - 100% and the range for a individual shot closes when the shot is attempted. The minimum will rise if it is returned successfully and the maximum is lowered if the player misses it. When a shot is needed, the subsystem will select one shot from the shots with the lowest minimum bound since they have the highest chance of being the player's weakest points. This provides the important benefit of exploring different shots at the beginning and exploiting some later in the training phase.

3.4 Shooting Model

The Shooting Model subsystem will be a service to the Smart Serve subsystem which provides the means of mapping a desired shot to the details needed to take the shot. The input and output for the system can be found in Figure 12. The output will be an array of many combinations of pitch, yaw, speed and angular velocity that satisfy the input shot details.

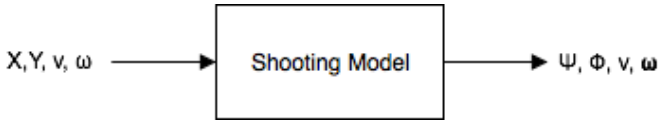

Figure 12: Shooting Model I/O

3.5 Data Storage

The Data Storage subsystem will store all the details for each shot, the user's profile and the return rates for each user. It will need to have interfaces for saving data and querying data out of the system in a variety of formats. Ideally, it will require minimal detail and configuration for the subsystem which uses it due to the variety of implementations of these subsystems.

3.6 User Interface

The User Interface subsystem will be the means of translating user requests into actionable requests for the Smart Serve subsystem. The inputs will include a username, password, mode selection and some means of starting and stopping the training. The UI will also need to accept custom parameters for performance data including time ranges and ways to cluster shot types. The output will include custom and useful error messages as well as graphs and tables for performance data.

3.7 Shooting Mechanism

The Shooting Mechanism will be the subsystem which fires the ball towards the user. A microcontroller will be used as the point of contact and set any actuators required to fire a specified shot. It must complete this action within 1.5 seconds for the Computer Vision to detect the ball returned by the user.

The Shooting Mechanism will consist of three main sub parts: Automatic Feeder, Automatic Panning, Ball Shooter. The functionality of the Automatic Feeder part will be to automatically control the feeding rate of the table tennis balls into the shooting barrel and feed the balls into the barrel as initiated by the program. The functionality of the Automatic Panning will be to rotate the entire shooting mechanism or the shooting barrel about the z-axis so the mechanism is able to cover all of the shooting zones as specified in the requirements. Additionally, the functionality of the ball shooter is actually shoot out table tennis balls from the shooting barrel. Please refer to figure 13 for visual representation.

In terms of electrical components, Arduino UNO microcontroller will be used to control

all of the IPO (Input Process Output) processes between the sensors, actuators and Smart Serve. For the ball shooter, a DC Motor will be used as the system will require high speeds and high torques to shoot out table tennis balls. The details on the DC motor can be found below in Table 1 (SAYAL Electronics SKU No. 230437). The DC Motor will be controlled using the PWM method for speed control. Additionally, a Stepper Motor will be utilized for the automatic feeding of the balls into shooting barrel functionality as the Stepper motor would not require any position encoder as well as the stepper motor will have predictable movements. Details on the stepper motor can be found below in Table 2 (Grobotronics SKU No. 19-00012858). For the automatic panning part of the shooting mechanism, a servo or a stepper motor will be utilized however that would be further refined after rigorous prototyping and simulations. Please refer to Figure 14, Figure 15, and Figure 16 for an overview of Electrical Bread Board, Schematic and PCB layout diagrams of how the Shooting Mechanism system outline.

Motor Type	DC
Operating Voltage	3V - 6V DC
Max. Speed	17000 rpm
Shaft Diameter	2 mm
Torque	20.72 g-cm

Table 1: Shooting Motor Specifications

Motor Type	Uni-polar Gear Stepper with Driver
Operating Voltage	5V DC
Max. Speed	80rpm over-driving with 9V
Holding Torque	350 g-cm
Step Angle (output shaft)	5.625°/64
Gear Ratio	1/64 reduction
Steps per Rev.	512
Shaft	Flattened for set-screw attachment

Table 2: Feeder Motor Specifications

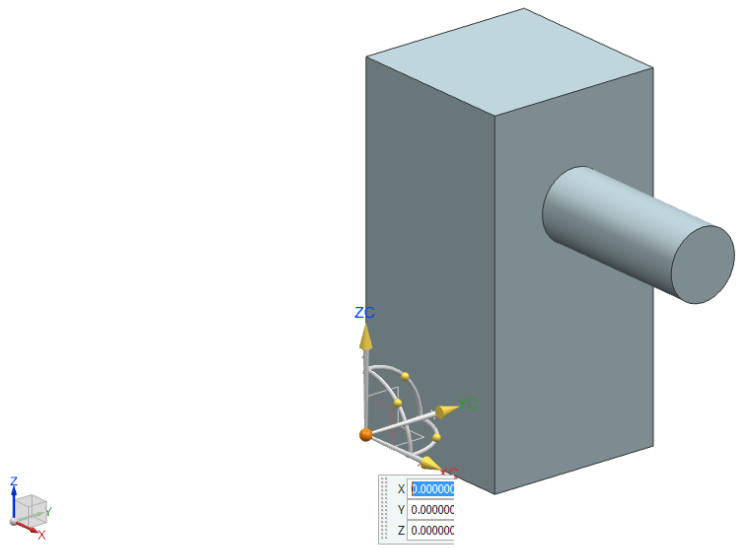

Figure 13: Shooting Mechanism Rough Outline

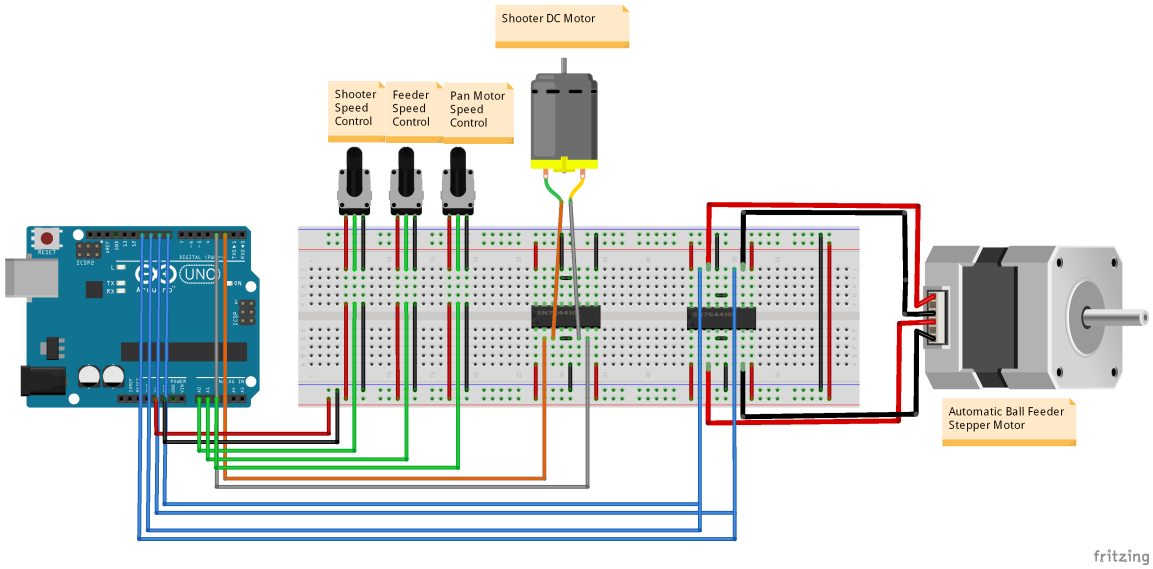

Figure 14: Electrical Wiring Diagram - Bread Board

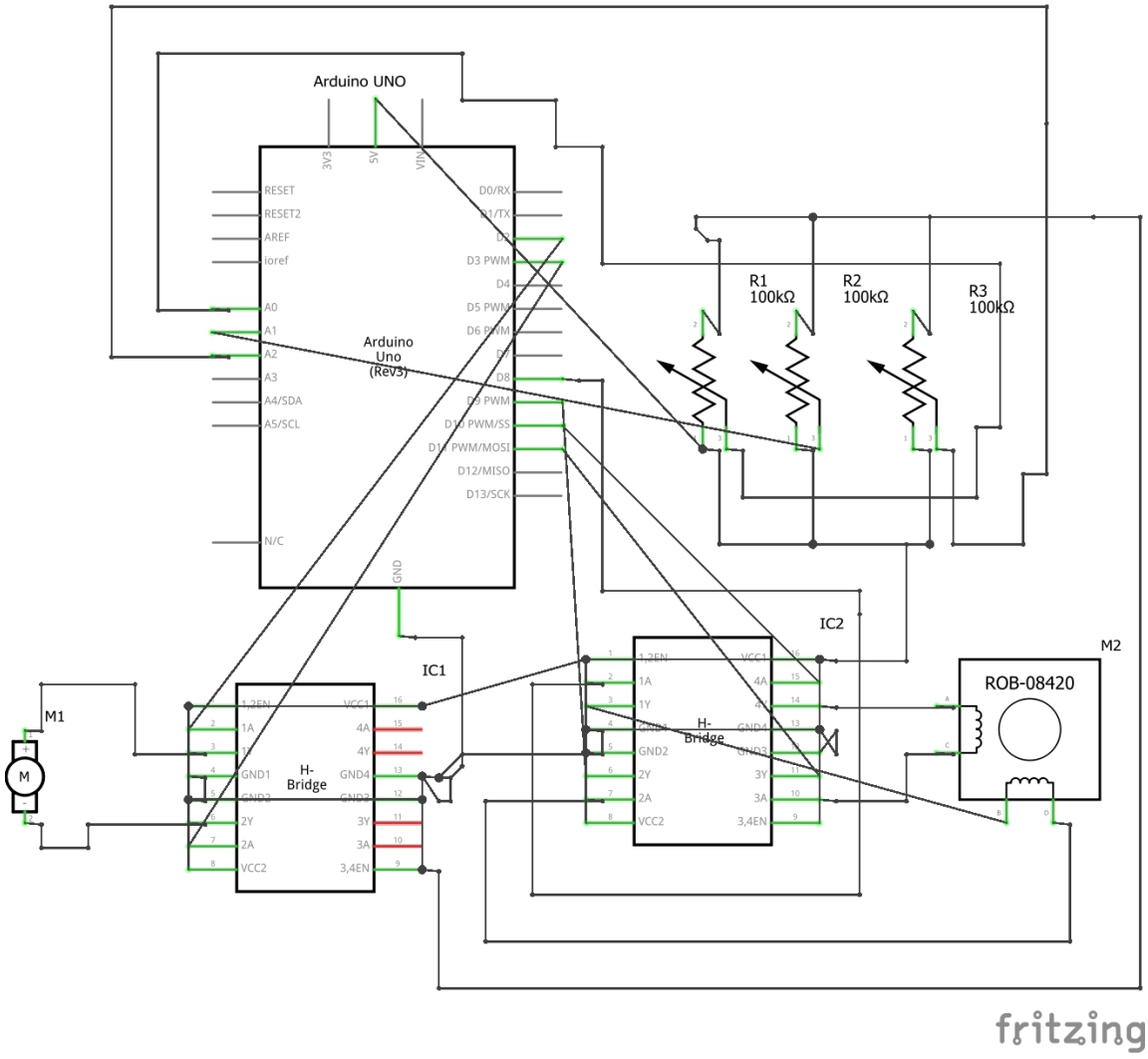

Figure 15: Electrical Wiring Diagram - Schematic

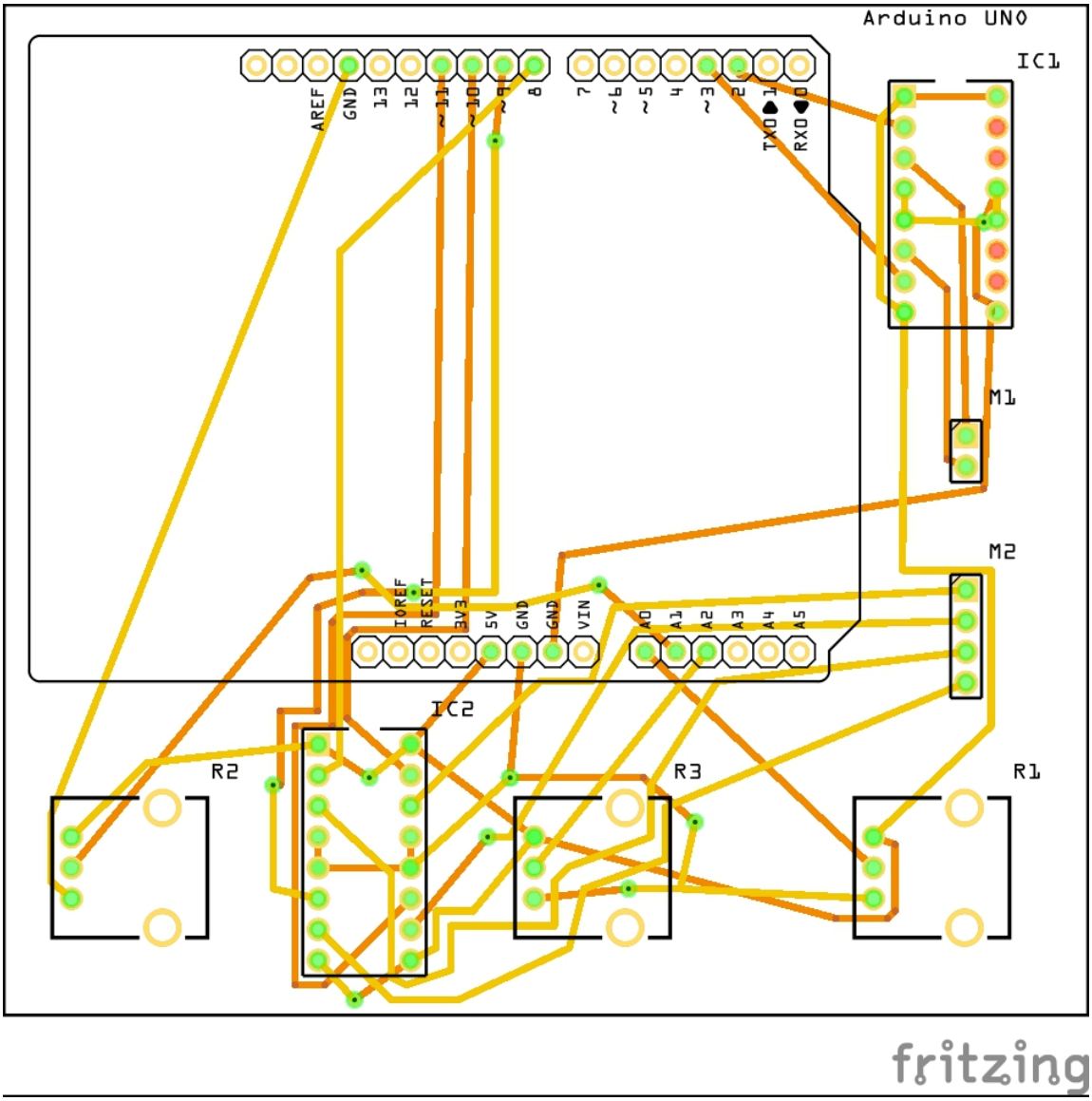

Figure 16: Electrical Wiring Diagram - Printer Circuit Board Trace

4 Class Responsibility Collaboration (CRC) Cards

The following sections will include one CRC card for each subsystem. A CRC card contains information pertaining to which requirements a subsystem is responsible for and which subsystems one would collaborate with. This will be used for testing purposes and to

track where a given requirement is satisfied. A requirement does not have to be satisfied completely by one subsystem, but a majority of the work must be done by that subsystem.

4.1 SmartServe

Smart Serve	
Responsibilities:	**Collaborators:**
• **F10**: pause the shooting mechanism • **F18**: shoot a ball once the previous has been returned to the system side or 1.5 seconds after the previous shot, whichever is shorter • **P2**: must include all but previous 3 shots in performance data • **P5**: must support only one user playing at one time	• Computer Vision • Shot Recommendation • Shooting Model • Data Storage • User Interface • Shooting Mechanism

Table 3: Smart Serve CRC Card

4.2 Computer Vision

Smart Serve	
Responsibilities:	**Collaborators:**
• **F5**: detect a successful return by the user • **OE2**: functional in indoor settings with bright florescent lighting	• Smart Serve

Table 4: Computer Vision CRC Card

4.3 Shot Recommendation

Smart Serve	
Responsibilities:	**Collaborators:**
• **F7**: load a previously saved state • **F14**: implements a training mode	• Smart Serve • Data Storage

Table 5: Shot Recommendation CRC Card

4.4 Shooting Model

Smart Serve	
Responsibilities:	**Collaborators:**
• **F14**: implements a training mode • **F15**: implements a one-shot mode	• Smart Serve

Table 6: Shooting Model CRC Card

4.5 Data Storage

Smart Serve	
Responsibilities:	**Collaborators:**
• **F6**: saves details for each shot taken by the shooting mechanism • **F8**: allows creation of a new user • **F9**: authenticate users • **P4**: must support 1000 users • **MS2**: able to add new metrics to analyze performance • **S1**: hash all passwords for user profiles • **S2**: encrypt all performance data for each user • **P1**: allow read access for coaches	• Shot Recommendation • Smart Serve

Table 7: Data Storage CRC Card

4.6 User Interface

Smart Serve	
Responsibilities:	**Collaborators:**
• **F11**: end the training session • **F12**: resume training session from a paused state • **F13**: display user's performance over a custom time range • **F16**: allows user to adjust training parameters during an active or paused session • **F17**: can be calibrated for a specific table size • **LF1**: have a minimalist design that is easy to navigate through • **UH1**: is intuitive to use • **UH2**: operable using the English language • **P1**: response time for user input must be less than or equal to 100ms	• Smart Serve

Table 8: User Interface CRC Card

4.7 Shooting Mechanism

Smart Serve	
Responsibilities:	**Collaborators:**
• **F1**: shoots the table tennis ball towards the user at various locations • **F2**: shoots the table tennis ball towards the user at various speeds • **F3**: shoots the table tennis ball towards the user at various degrees of Yaw • **F4**: shoots the table tennis ball towards the user with various degrees of roll • **HS1**: always hit the table at least once per shot • **HS2**: not shoot the ball faster than 22 m/s • **HS3**: have no exposed electrical wiring or components • **HS4**: carries warnings around moving parts • **HS5**: has a button to cease all power to system	• Smart Serve

Table 9: Shooting Mechanism CRC Card

5 Detailed Class Diagram

The Detailed Class Diagram is shown in Figure 37, found in the Appendix.

6 Module Guide

6.1 SmartServe Modules

Controller

Responsibilities
The controller handles all the timing constraints and sequential events for shooting balls towards the player. It is the interface for the UI to allow the user to preform any and all actions.

Secrets
The sequence and timing constraints of the shooting procedure.

MID

- **boot** - none
  returns: *boolean*
  description: instantiates all dependancies and ensures services are working as expected
- **startTraining** - Mode m
  returns: *boolean*
  description: starts the shooting procedure given a certain training Mode
- **stopTraining** - none
  returns: *boolean*
  description: stops the training procedure
- **setShootingParameters** - ShootingParameters sp
  returns: *boolean*
  description: sets the shooting parameters for certain table sizes

ArduinoConnector

Responsibilities
This module is responsible for facilitating the communication with the Arduino which is part of the Shooting Mechanism subsystem (SM). This includes sending and receiving messages as well as ensuring proper testing of the connection is preformed.

Secrets
The connection to the Arduino.

MID

- **test** - int port
  returns: *boolean*
  description: tests connection to the Arduino

- **shoot** - float pitch, float yaw, float angularVelocity
  returns: *none*
  description: instructs Arduino to shoot the ball in a certain way

- **position** - none
  returns: *Position*
  description: returns the position of the mechanism

ShotRecommendationConnector

Responsibilities
This module is responsible for facilitating the communication with the ShotRecommendation subsystem (SR). This includes sending and receiving messages as well as ensuring proper testing of the connection is preformed.

Secrets
The connection to the SR.

MID

- **connect** - int port, [optional] String ip
  returns: *boolean*
  description: instantiates all dependancies and ensures services are working as expected

- **getRecommendation** - none
  returns: *Shot*
  description: returns the shot data to shoot towards the player

- **updateModel** - Shot shot, boolean returned
  returns: *none*
  description: sends data to SR on whether a R was returned or not

CVConnector

Responsibilities
This module is responsible for facilitating the communication with the Computer Vision subsystem (CV). This includes sending and receiving messages as well as ensuring proper testing of the connection is preformed.

Secrets
The connection to the CV system.

MID

- **connect** - int port
  returns: *boolean*
  description: tests connection to CV subsystem

- **start** - none
  returns: *boolean*
  description: instructs CV to begin tracking and return data for shot

SQLConnector

Responsibilities
This module is responsible for facilitating the communication with the Data Storage subsystem (DS). This includes sending and receiving messages as well as ensuring proper testing of the connection is preformed.

Secrets
The connection to the DS.

MID

- **connect** - int port, [optional] String ip
  returns: *boolean*
  description: tests connection to DS on port *port* at the IP Address *ip* or *localhost* if ip is unavailable

- **query** - String procedure, Map<String, String>values
  returns: *ResultSet*
  description: returns data from database based on procedure ran and values given

- **save** - String procedure, Map<String, String>values
  returns: *boolean*
  description: returns success information on write to database based on procedure ran and values given

ShootingParameters

Responsibilities
None.

Secrets
None.

MID

- **ShootingParameters** - double tableWidth, double tableLength
  description: constructs a ShootingParameters object

6.2 Shot Recommendation Modules

Controller

Responsibilities
This module will act as the API interface for the SR. As such, it will accept requests and return the appropriate data. It will also communicate will other modules should a user of this system need to do so.

Secrets
The process for handling SR requests.

MID

- **listen** - none
  returns: *none*
  description: waits for a request made for a shot
- **query** - String procedure, Map<String, String>values, [optional] int port, [optional] String ip
  returns: *Cursor*
  description: gets data from a stored procedure using some set of values for a MySQL instance on port *port* at the IP Address *ip*

Model

Responsibilities
This module will hold the data for each user in such a way information can be extracted. It will contain an internal model which is built from the data and use it to recommend a new shot.

Secrets
The algorithm to build the model.

MID

- Model **model** - representation of shot performance data for extracting information
- **Model** - none
  description: constructs a new Model
- **train** - Cursor cur
  returns: *none*
  description: using some data from Cursor, this will train the model
- **next** - none
  returns: *Shot*
  description: returns a shot based on the user's past performance

6.3 Shooting Model Modules

Shot

Responsibilities
Specific details (such as yaw and speed) that are required to take the desired shot are stored within this abstract data type module.

Secrets
None

MID

- **Shot** - Orientation orientation, double velocity
  description: constructor to store the shooting model details in an abstract data type.
- **toString** - none
  returns: *String*
  description: Returns shooting details in a printable string format.

ShootingModel

Responsibilities
Responsible for mapping a desired shot to the details needed to take the shot.

Secrets
Methods and formulas being used to identify the required details needed to take the desired shot.

MID

- **ShootingModel** - double initialHeight, double xInitialHeight, double pitch
  description: instantiates the ShootingModel subsystem (SModel).

- **getShotDetails** - double landingXCoord, double landingYCoord
  returns: *Shot*
  description: calculates details to take the desired shot and stores them within the ADT.

6.4 Computer Vision Modules

Detect

Responsibilities
This module detects successful returns from the user, and sends that data to the SmartServe subsystem (SS).
Secrets
Success criteria of returns and how the camera feed is analyzed.
MID

- **detect** - CameraCapture cap
  returns: *none*
  description: Detects if the ball was succesfully returned by the user and calls the Send function

- **send** - none
  returns: *none*
  description: Sends a signal to SS indicating that a successful return was made

CameraCapture

Responsibilities
None.

Secrets
None.

MID

- **CameraCapture** - none
  description: constructs a CameraCapture object

6.5 Data Storage Modules

Global

Responsibilities
Communicates with the database to create, delete and update rows in various tables.

Secrets
All connection information between the sub-systems, table contents.

MID

- User **user** - String userName, String password, int userId
  description: representation of a user
- Shot **shotType** - int zoneId, int omegaId, int shotId
  description: representation of a shot type
- Zone **zone** - int zoneId, double xLoc, double yLoc
  description: representation of a zone on the table, used as a look up table
- Omega **omega** - int omegaId, double angle, double velocity
  description: representation of angular velocity of the ball, used as a look up table
- ReturnRate **returnRate** - int userId, int shotId, Timestamp timeStamp, boolean returned
  description: return statistics for each user
- **signUp** - String userName, String password
  returns: *none*
  description: adds a user row to the *User* table
- **nextShot** - int zone
  returns: *Shot*
  description: determines which shot type to perform
- **returned** - Timestamp timeStamp, User user, Shot shot
  returns: *none*
  description: updates returnRate table for user for performance statistics
- **signIn** - User user
  returns: *boolean*
  description: validates and authenticates the user

6.6 Shooting Mechanism Modules

This module will be specified in the hardware component design of the SM.

6.7 User Interface Modules

Controller

Responsibilities
Responsible for translating the desired action of the user from the View module to the SS.

Secrets
Connection between modules and sub-systems.

MID

- **main** - String[] args
  returns: *none*
  description: calls the View module to initialize the UI
- **setTrainingMode** - Mode m
  returns: *none*
  description: sets mode from View module and sends it to SS
- **setShootingParameters** - ShootingParameters
  returns: *none*
  description: sets shooting parameters from View module and sends it to SS
- **setStatistics** - String statParam
  returns: *none*
  description: sets statistics parameters and sends them to View module
- **signUp** - User user
  returns: *none*
  description: gets user information from view module and sends it to SS
- **login** - User user
  returns: *boolean*
  description: gets user name and password from the View module and authenticates by calling SS

View

Responsibilities
The view module will contain all the actual visual aspects of the system that the user will

interact with. This includes text, pictures, buttons, etc.

Secrets
The structure and implementation of the view.

MID

- **start** - none
  returns: *none*
  description: general user interface code, called to initiate UI
- **selectMode** - MouseEvent me
  returns: *Mode*
  description: listens for a mouse click on mode types, and returns that mode
- **calibrate** - none
  returns: *ShootingParameters*
  description: collects user input for table size in order to calibrate the system
- **startTrainingBtn** - MouseEvent me
  returns: *boolean*
  description: listens for mouse click on start training button
- **stopTrainingBtn** - MouseEvent me
  returns: *boolean*
  description: listens for mouse click on stop training button
- **viewStatsBtn** - MouseEvent me
  returns: *none*
  description: listens for mouse click on statistics button and displays statistics to the user
- **signUpBtn** - MouseEvent me
  returns: *User*
  description: listens for mouse click on sign up button and collects user information
- **loginBtn** - MouseEvent me
  returns: *User*
  description: listens for mouse click on sign in button and displays profile if authenticated properly
- **displayError** - Exception exceptionLog
  returns: *none*
  description: allows user to read an error or download a log file to send to development team

7 Communication Protocols

7.1 SmartServe to Shot Recommendation

The SR will use Python to leverage machine learning libraries like SciKit Learn and the SS will be implemented in Java. In order for the SS to communicate to the SR, it will make an HTTP request with some data and receive an HTTP response encoded using JSON. This allows the use of a reliable means of communication and flexibility to host the SR remotely if need be.

The SS will make a GET request to the SR for requesting a shot to use and can make POST request to give data regarding whether a shot was returned or not. In the event the HTTP request takes too long, the SS should handle it accordingly by timing out and using random shots or continuing the program.

HTTP libraries are standard in Java and a microframework for handling HTTP requests can be used for Python like flask.

7.2 SmartServe to Shooting Mechanism

The SM will implement an interface using an Arduino. The SS will use libraries for communicating to the Arduino to communicate to the SM which can be found here for 64-bit Windows and Linux installations and here for 64-bit macOS installations. The SS does not need a response from the system, as it will tell the SM where to shoot and how to do so but does not need a response.

7.3 SmartServe to Computer Vision

The SS will communicate to the CV via sockets over the TCP protocol using the Java Networking libraries and the Python socket libraries. The SS will initiate the communication to start tracking and expect a return value based on whether it was returned or not. The SS will timeout in the event the CV does not return a value after 1.5 seconds.

7.4 SmartServe to User Interface

The SS and the User Interface subsystem (UI) will both be programmed using Java. The UI system can interface with the SS by calling exposed public methods based on user input.

7.5 SmartServe to ShootingModel

The SS and the SModel will both be programmed using Java. The SS system can interface with the SModel by calling exposed public methods based on user input.

7.6 SmartServe to Data Storage

The SS is programmed in Java and the DS will be implemented using Stored Procedures held in a MySQL database instance. In order for the SS to use the DS, a SQLConnector can be used to do so.

7.7 Shot Recommendation to Data Storage

The SR is programmed in Python and the DS will be implemented using Stored Procedures held in a MySQL database instance. In order for the SR to use the DS, a SQLConnector can be used to do so.

8 Hardware Design

8.1 Mechanical Components

Shift Adapter

The shaft adapter is designed to fit snugly on the stepper motor shaft and provides a more robust connection with the *spinning feeder cutout*. The shaft adapter is designed to allow the feeder cutout to sit at a precise height where it could rotate without any obstructions and push the table tennis balls into the *Inner Tube*. The shaft adapter will bring the spinning feeder cutout to a height such that the contant point of the ball is in the middle (biggest diameter of the ball).

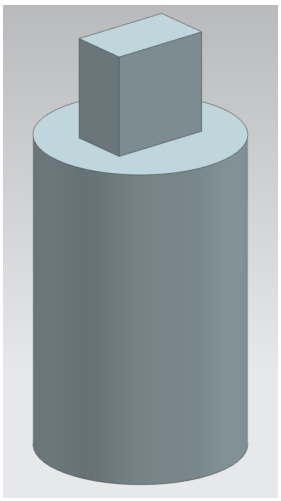

Figure 17: Shaft Adapter 3D

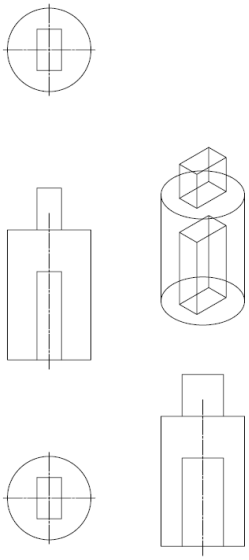

Figure 18: 2D Engineering Drawing of Shaft Adapter

Panning Control

The functionality of the Automatic Panning will be to rotate the entire shooting mechanism or the shooting barrel about the z-axis so the mechanism is able to cover all of the shooting zones as specified in the requirements.

The components designed for the panning stage include an acrylic base plate, a rapid prototyped worm wheel gear ring, the worm, and a motor bracket. The base plate clearance mounting holes were made using a workshop drill press / milling machine. The rapid prototyped stepper motor bracket is mounted to the base plate and adjusted so that correct gear engagement is achieved. The Lazy Susan bearing is positioned 10 mm above the plate using spacers and longer #6?32 screws, washers, and nuts. The worm gear is designed as a ring shape to reduce rapid prototyping material consumption. A corresponding hole pattern mates to the top plate of the Lazy Susan bearing. The optical switch is used to home the system at power up and align the system in the direction required with a reference and from there on the system will always be aware of its position. Optical switches allow for high repeatability making it a very applicable sensor to use in the prototype.

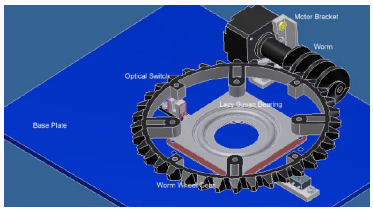

Figure 19: Base Design and Assembly

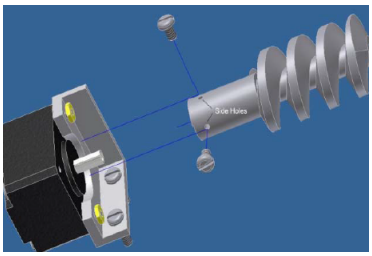

Figure 20: Worm Gear Coupler

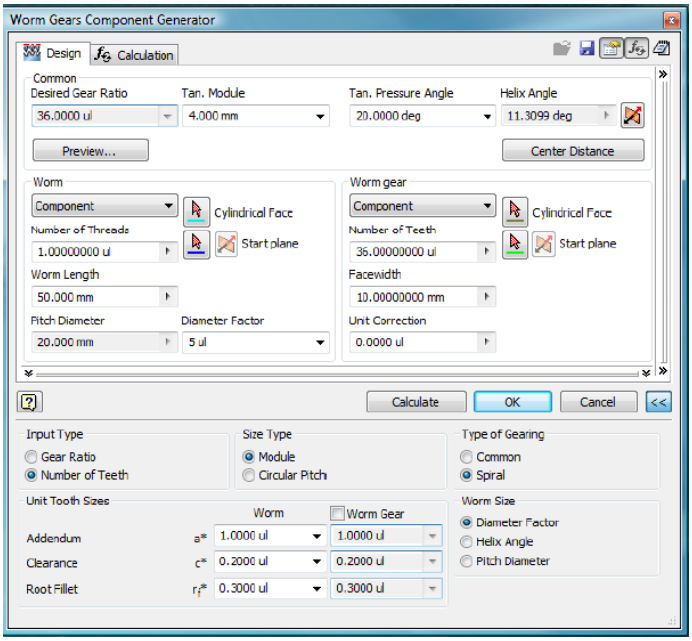

Figure 21: Azimuth Stage Autodesk Inventor Design Accelerator Worm Gear Parameters

Spinning Feeder Cutout

The spinning feeder cutout is fixed to the shaft of the stepper motor inside the bucket to guide the table tennis balls into the *Inner Tube*. The cutout is laser cut from a thin and sturdy sheet of wood so it has enough thickness (contact surface area) to easily push the table tennis balls around. There will be a lot of balls going in and out of the feeder cutout continuously so it has the sharp edges to hold the ball in once its captured and 45mm diameter semi circles to easily allow the 40mm table tennis balls to slide into the pockets.

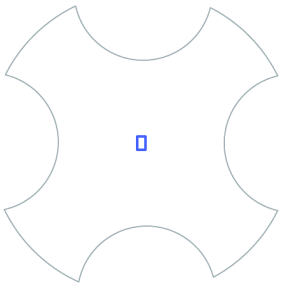

Figure 22: Feeder Cutout from topview for laser cutting

Inner Tube

The inner tube is designed to hold table tennis balls to be shot out by the shooting mechanism, as well as have solid support hold the shooting mechanism pipe structure. The diameter of the inner tube is 45mm to allow the ball to be easily fed into the tube but restrict jittering when traveling through the pipe.

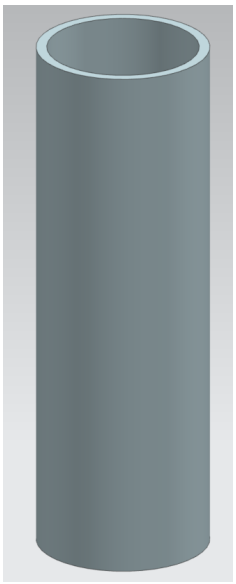

Figure 23: CAD design of a Inner Tube

Motor Bracket and Outer Tube

A custom designed motor bracket is used to mount the DC motor onto the outer PVC tube to create our preliminary shooting mechanism. The motor has a cylindrical housing thus making it difficult to fix onto anything especially onto another cylindrical object, the Outer Tube. The motor has to be mounted securely and tightly to minimize the vibrations into the shooting mechanism which could translate onto the entire system.

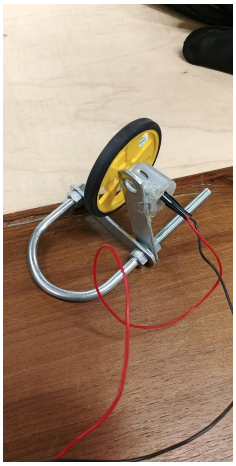

Figure 24: Motor Bracket

Figure 25: Outer Tube

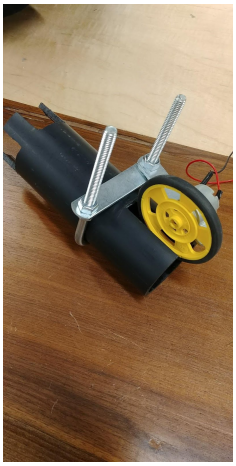

Figure 26: Motor with Motor Bracket attached to the Outer Tube

Roll Control

The roll control part will act as a manual control for the user to decide the type of spin to give the table tennis ball for the current session. The roll control part acts as the male end of a locking mechanism, which will fix the shooter in either a 0, 90, 180, or 270 degree angle. With the roll control settings, the user will be able to manually change the roll setting to experience a topspin, backspin, and left or right spin.The roll control part will be fixed on the base end of the inner shaft of the shooter. The outer shaft of the shooter will then be able to lock into the desired position.

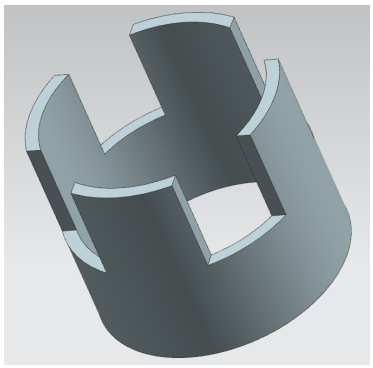

Figure 27: CAD design for Roll Control component

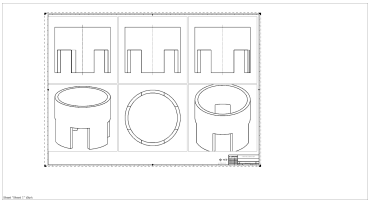

Figure 28: 2D Engineering Drawing of Roll Control Component

Base, Support Brackets and Bucket

The base is made out of a hardwood and laser cut to an appropriate size. The base acts as a fixture to attach the bucket supports to the azimuth stage. There are two metal support brackets which are made out of stainless steel, and lift the bucket to an appropriate height above the base. The bottom of the supports are screwed into the hardwood base, and the top of the supports are bolted to the bucket. The bucket is 8in x 7in and is large enough to hold at least 20 balls without overfilling.

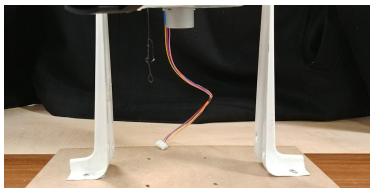

Figure 29: Base with Support Brackets attached

Figure 30: Bucket with Inner Tune and Roll Control attached

Pitch Control

The pitch control consists of a screw that pierces through the support beam and bucket, to hold the shooter at a certain angle. The range of angles are as followed: -15,0,15,30,45, and are set by pre drilled holes in the bucket. This design needs to be optimized with a bolt in order to further secure the pitch and eliminate error.

Figure 31: Pitch Control slots

Figure 32: Pitch Control Holding Pin from Inside

8.2 Electrical Components

Stepper Motors

We chose a Japan Servo KP35FM2-035 stepper motor for the panning assembly because of its torque and step accuracy. To make the panning smooth as possible the motor provides 200 steps per revolution (1.8 degrees per step) from this stepper is sufficient for our purposes. The max torque is 700 g/cm at 24 VDC / 500mA but due to microcontroller restrictions and we are only using a torque of approximately 250g/cm to operate at a around 9VDC. At this lower voltage the stepper motor combined with the worm gear train provides enough torque to move the system (panning) at a sufficient speed. A gear stepper motor (model 28BYJ48) with a driver is utilized for the automatic feeding of the balls into shooting barrel functionality as the stepper motor would not require any position encoder as well as the stepper motor will have predictable movements. This motor generates 300 g/cm torque at a voltage of 5VDC. This motor will be continuously running during the operation of the system at varying speeds and a ball will enter the Inner Tube at every quarter turn of the Feeder Cutout. The stepper has 64 steps per revolution (5.625 degrees per step) so every 16 steps the stepper moves, a ball will be fed into the feeder. This low-step count motor is good for our purposes because it allows the stepper to function at higher RPMs than other higher step count similar frame size motors.

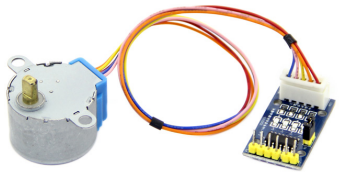

Figure 33: Gear Stepper Motor (Model 28BYJ48) with a Driver

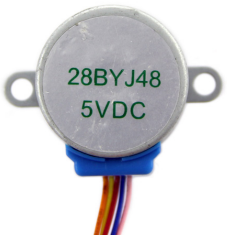

Figure 34: Stepper Motor 28BYJ48 Mounting Holes

DC Motors

For the ball shooter, a DC Motor purchased from Sayal Electronics and Hobbies is used as the system will require high speeds and high torques to shoot out ping pong balls. The DC motor operates between 3-6 volts, can reach a max RPM of 17,000, has a 2mm shaft diameter and max torque of 20.72 g/cm. Although the table tennis balls don't account for much weight the motor torque needs to be sufficient enough so that the wheel attached to the shaft can push on the ball, make contact and toss it out as required speeds. The DC Motor will be controlled using the PWM method for speed control.

Figure 35: MOTOR DC 3-6VOLTS 17000RPM 2MM SHAFT 20.72G/CM TORQUE

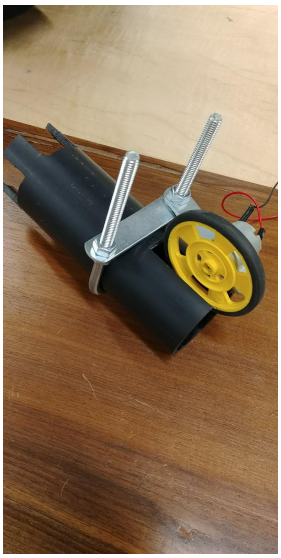

Figure 36: DC Motor attached to the Motor Bracket onto the Outer Tube

Arduino

Arduino UNO microcontroller is used to control all of the IPO (Input Process Output) processes between the sensors, actuators and Smart Serve. Arduino is chosen because it has a hardware platform setup and allows programming and serial communication over USB so designing so a PCB is not necessary. It is fast for our purposes, there is a lot of libraries available to use and lots of hardware modules designed for Arduino UNO. Also, this microcontroller easily interfaces with the hardware used in this prototype.

9 Module-Requirement Traceability Matrix

Table 10: Module-Functional Requirement Traceability Matrix

Requirement ID	SS	SR	SM	CV	DS	SMech	UI
F1			X			X	
F2						X	
F3						X	
F4						X	
F5				X			
F6					X		
F7					X		
F8					X		
F9					X		
F10	X						X
F11	X						X
F12	X						X
F13					X		X
F14		X					
F15		X					
F16	X				X		X
F17	X				X		X
F18	X						

Table 11: Module Non-Functional Requirement Traceability Matrix

Requirement ID	SS	SR	SM	CV	DS	SMech	UI
LF1							X
UH1							X
UH2							X
P1							X
P2	X				X		
P4					X		
P5	X						
OE2				X			
MS2	X				X		X
S1					X		
S2					X		
P1	X				X		
LC1							
HS1			X			X	
HS2						X	
HS3						X	
HS4						X	
HS5						X	

10 Appendix

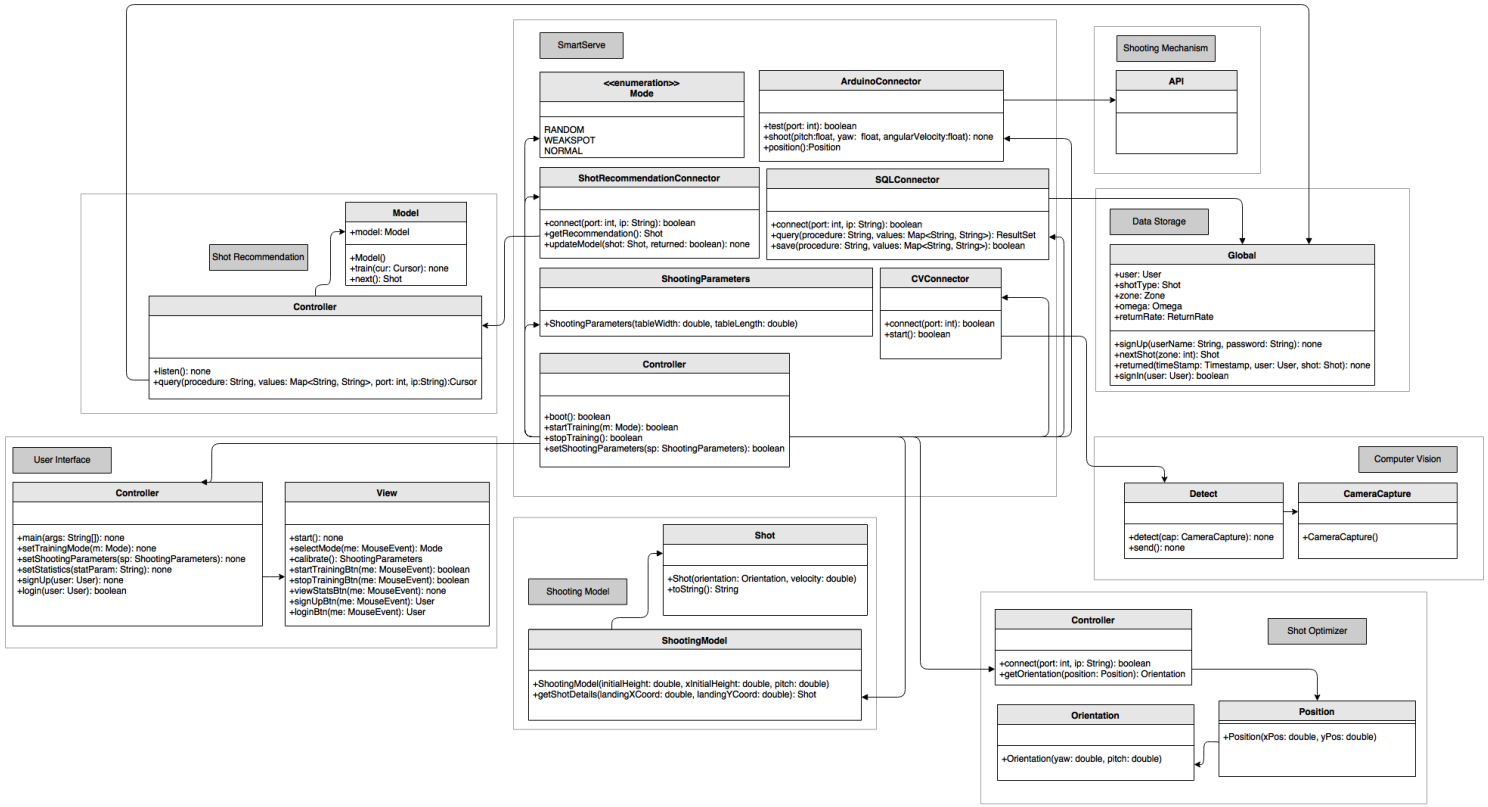

Figure 37: Detailed Class Diagram