McMaster University

SmartServe

Software & Mechatronics Capstone

Low Level System Design

Authors:
Christopher McDonald
Harit Patel
Janak Patel
Jared Rayner
Nisarg Patel
Sam Hamel
Sharon Platkin

Professor:
Dr. Alan Wassyng

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

Last compiled on January 11, 2018

Contents

List of Figures

Date	Revision	Comments	Author(s)
Dec 29, 2017	1.0	Structure made for document including headings	Christopher McDonald
Jan 11, 2018	1.1	Added content for ShotRecommendation and SmartServe systems, and Communication Protocols Sections	Christopher McDonald
Jan 11, 2018	1.2	Added content for Document Overview	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

This document will add more detail into the subsystems introduced in the High-Level System Design document (HLSD) found here. The HLSD omitted details such as how the subsystems would work, how they are built and how communication will be handled between them. This document will cover all of those details and introduce how each subsystem will be programmed by defining the language and libraries required.

All software developers on the development team should read this document when building the subsystems to ensure the correct data is being sent and communication portals are set up correctly. Each subsystem will hold one to many modules and each module will contain one to many methods. A detailed class diagram will organize all the subsystems and their modules with arrows indicating which module uses another one.

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

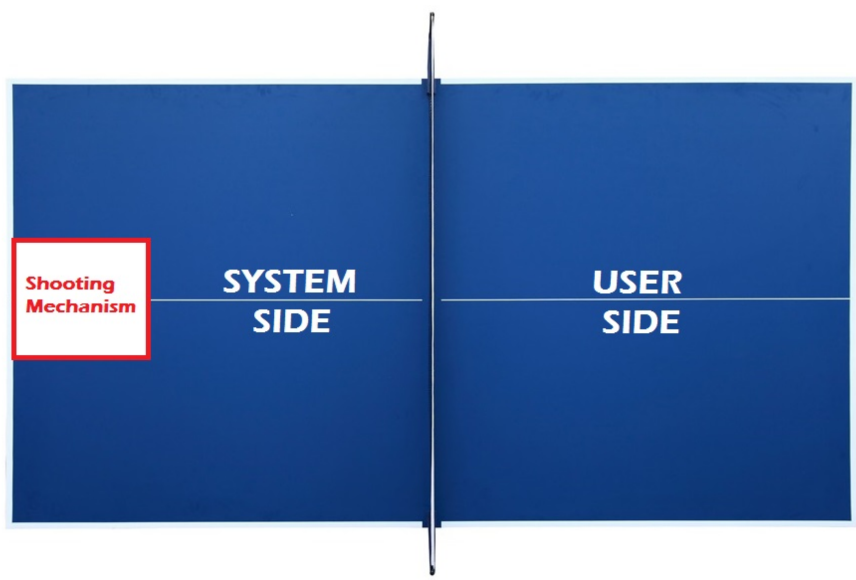

Figure 2: Top View of the Tennis Table

2 Detailed Class Diagram

3 Module Guide

3.1 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. 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 Shot Recommender. This includes sending and receiving messages as well as ensuring proper testing of the connection is preformed.

Secrets
The connection to the Shot Recommender.

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 ShotRecommender on whether a shot was returned or not

CVConnector

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

Secrets
The connection to the Data Storage system.

MID

- **connect** - int port, [optional] String ip
  returns: *boolean*
  description: tests connection to Data Storage subsystem 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

3.2 Shot Recommendation Modules

Controller

Responsibilities
This module will act as the API interface for the Shot Recommendation system. 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 Shot Recommendation requests.

MID

- **listen** - none
  returns: *none*
  description: waits for a request made for a shot
- **query** - String procedure, dict 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
- **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

3.3 Shooting Model Modules

Module 1

Responsibilities
some text
Secrets
some text
MID
some text
MIS
some text

3.4 Shot Optimizer Modules

Module 1

Responsibilities
some text
Secrets
some text
MID
some text
MIS
some text

3.5 Computer Vision Modules

Module 1

Responsibilities
some text
Secrets
some text
MID
some text
MIS
some text

3.6 Data Storage Modules

Module 1

Responsibilities
some text
Secrets
some text
MID
some text
MIS
some text

3.7 Shooting Mechanism Modules

Module 1

Responsibilities
some text
Secrets
some text
MID
some text
MIS
some text

3.8 User Interface Modules

Module 1

Responsibilities
some text
Secrets
some text
MID
some text
MIS
some text

4 Communication Protocols

4.1 Smart Serve to Shot Recommendation

The Shot Recommendation system (SR) will use Python to leverage machine learning libraries like SciKit Learn and the Smart Serve system (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.

4.2 Smart Serve to Shooting Mechanism

The Shooting Mechanism system (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.

4.3 Smart Serve to Computer Vision

The SS will communicate to the Computer Vision subsystem (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.

4.4 Smart Serve to User Interface

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

4.5 Smart Serve to Shot Optimizer

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

4.6 Smart Serve to Shooting Model

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

4.7 Smart Serve to Data Storage

The SS is programmed in Java and the Data Storage system (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.

4.8 Shot Recommendation to Data Storage

The SR is programmed in Python and the Data Storage system (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.