

McMASTER UNIVERSITY

SMARTSERVE

SOFTWARE & MECHATRONICS CAPSTONE

Component 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 14, 2018

Contents

1	Introduction	4
1.1	Project Overview	4
1.2	Document Overview	4
1.3	Naming Conventions and Terminology	4
2	Detailed Class Diagram	6
3	Module Guide	6
3.1	SmartServe Modules	6
3.2	Shot Recommendation Modules	9
3.3	Shooting Model Modules	11
3.4	Shot Optimizer Modules	11
3.5	Computer Vision Modules	13
3.6	Data Storage Modules	13
3.7	Shooting Mechanism Modules	14
3.8	User Interface Modules	14
4	Communication Protocols	16
4.1	SmartServe to Shot Recommendation	16
4.2	SmartServe to Shooting Mechanism	17
4.3	SmartServe to Computer Vision	17
4.4	SmartServe to User Interface	17
4.5	SmartServe to Shot Optimizer	17
4.6	SmartServe to ShootingModel	17
4.7	SmartServe to Data Storage	18
4.8	Shot Recommendation to Data Storage	18
5	Hardware Design	18
5.1	Mechanical Components	18
5.2	Electrical Components	24
6	Appendix	26

List of Figures

1	Revision History	3
2	Top View of the Tennis Table	6
3	Shaft Adapter 3D	19
4	2D Engineering Drawing of Shaft Adapter	19

5	Base Design and Assembly	20
6	Worm Gear Coupler	20
7	Azimuth Stage Autodesk Inventor Design Accelerator Worm Gear Parameters	20
8	Feeder Cutout from topview for laser cutting	21
9	CAD design of a Inner Tube	21
10	Motor Bracket	22
11	Outer Tube	22
12	Motor with Motor Bracket attached to the Outer Tube	22
13	CAD design for Roll Control component	23
14	2D Engineering Drawing of Roll Control Component	23
15	Base with Support Brackets attached	23
16	Bucket with Inner Tune and Roll Control attached	23
17	Pitch Control slots	24
18	Pitch Control Holding Pin from Inside	24
19	Gear Stepper Motor (Model 28BYJ48) with a Driver	25
20	Stepper Motor 28BYJ48 Mounting Holes	25
21	MOTOR DC 3-6VOLTS 17000RPM 2MM SHAFT 20.72G/CM TORQUE .	26
22	DC Motor attached to the Motor Bracket onto the Outer Tube	26
23	Detailed Class Diagram	27

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
Jan 12, 2018	1.3	Added Data Storage and User Interface modules	Sharon Platkin
Jan 13, 2018	1.4	Added details for Shooting Mech, CV, Shot Optimizer, Shooting Model subsystems	Harit Patel & Sam Hamel
Jan 15, 2018	1.5	Edited Module Guide, Added Detailed Class Diagram	Sharon Platkin
Jan 16, 2018	1.6	Edited Document, refined and defragged definitions	Sharon Platkin & Christopher McDonald
Jan 19, 2018	1.6	Added work done by Hardware Team	Nisarg Patel & Jared Rayner & Janak Patel & Christopher McDonald
Mar 13, 2018	2.0	Added how to handle errors and exceptions	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 or variables. 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
- **ADT**: abstract data type
- **CV**: computer vision subsystem
- **DS**: data storage subsystem

- **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
- **SM:** shooting mechanism subsystem
- **SModel:** shooting model subsystem
- **SO:** shot optimizer subsystem
- **SR:** shot recommendation subsystem
- **SS:** smartserve subsystem
- **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
- **UI:** user interface subsystem
- **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

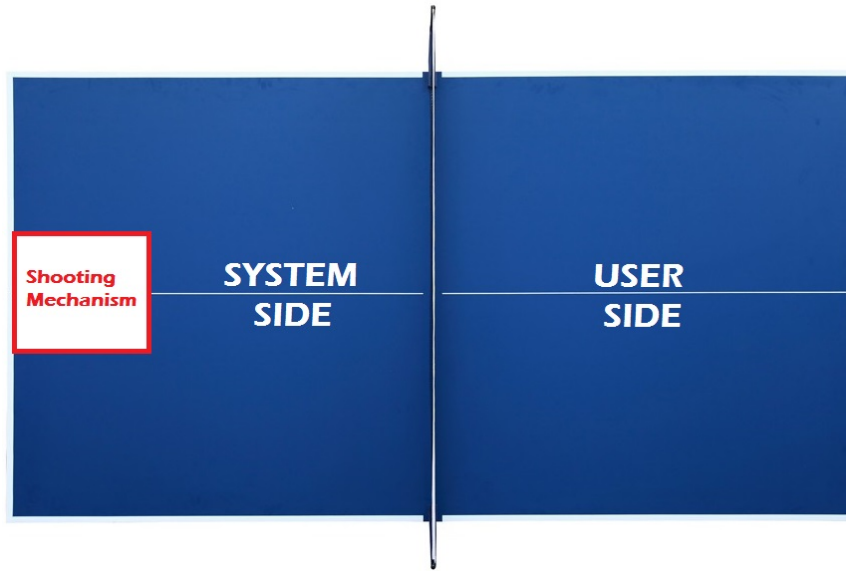


Figure 2: Top View of the Tennis Table

2 Detailed Class Diagram

The Detailed Class Diagram is shown in Figure 23, found in the Appendix.

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 perform 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 sub-

system (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

3.2 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 with 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

3.3 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.

3.4 Shot Optimizer Modules

Controller

Responsibilities

This module is responsible for facilitating the communication with the Shot Optimizer

subsystem (SO). This includes sending and receiving messages as well as ensuring proper testing of the connection is performed.

Secrets

The connection to the SO.

MID

- **connect** - int port, [optional] String ip
returns: *boolean*
description: instantiates all dependancies and ensures services are working as expected.
- **getOrientation** - Position position
returns: *Orientation*
description: returns the optimal orientation data to orient the SM.

Orientation

Responsibilities

None.

Secrets

None.

MID

- **Orientation** - double yaw, double pitch
description: constructs a Orientation object

Position

Responsibilities

None.

Secrets

None.

MID

- **Position** - double xPos, double yPos
description: constructs a Position object

3.5 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 successfully 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

3.6 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

3.7 Shooting Mechanism Modules

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

3.8 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

4 Communication Protocols

4.1 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.

4.2 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.

4.3 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.

4.4 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.

4.5 SmartServe to Shot Optimizer

The SS and the 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 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.

4.7 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.

4.8 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.

5 Hardware Design

5.1 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 contact point of the ball is in the middle (biggest diameter of the ball).

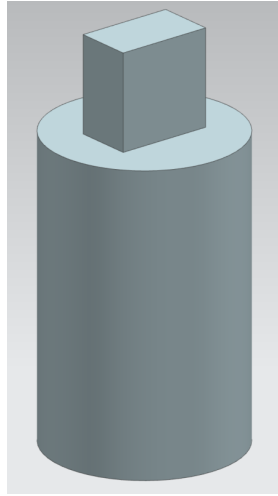


Figure 3: Shaft Adapter 3D

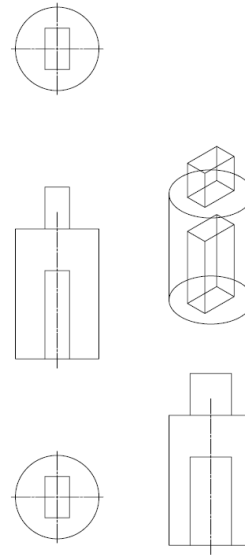


Figure 4: 2D Engineering Drawing of Shaft Adapter

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.

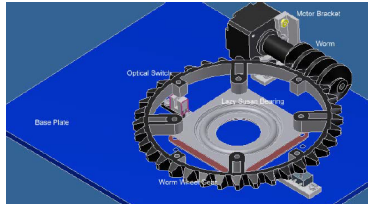


Figure 5: Base Design and Assembly

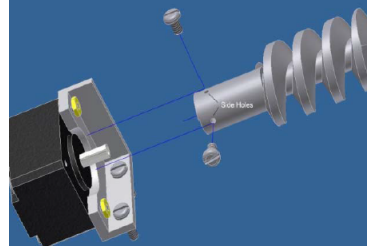


Figure 6: Worm Gear Coupler

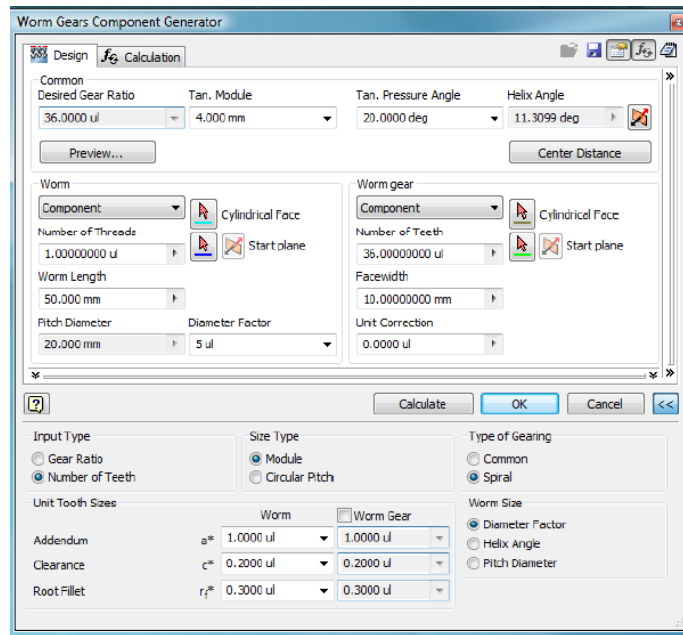


Figure 7: Azimuth Stage Autodesk Inventor Design Accelerator Worm Gear Parameters

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.

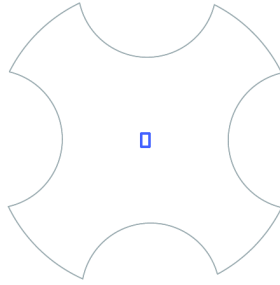


Figure 8: Feeder Cutout from topview for laser cutting

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.

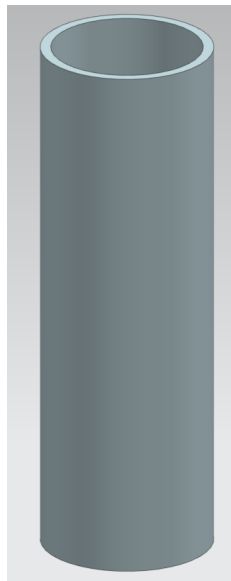


Figure 9: CAD design of a Inner Tube

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.

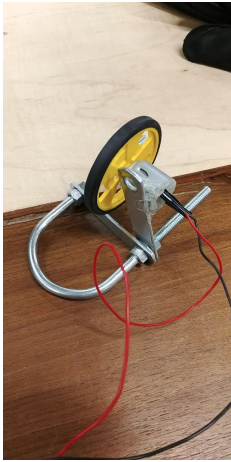


Figure 10: Motor Bracket



Figure 11: Outer Tube

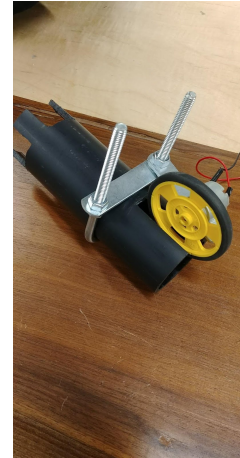


Figure 12: Motor with Motor Bracket attached to the Outer Tube

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.

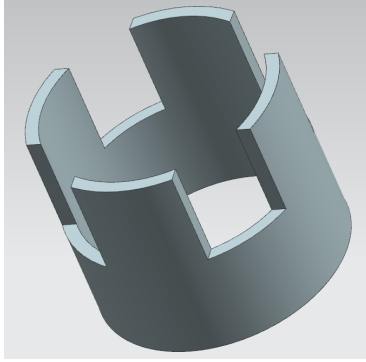


Figure 13: CAD design for Roll Control component

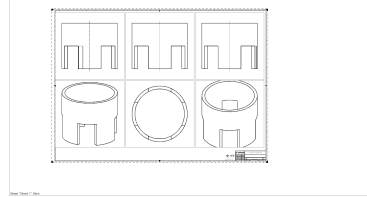


Figure 14: 2D Engineering Drawing of Roll Control Component

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.

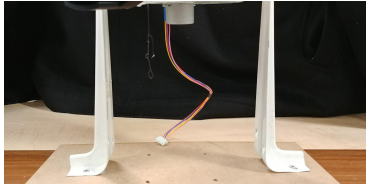


Figure 15: Base with Support Brackets attached



Figure 16: Bucket with Inner Tube 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 17: Pitch Control slots



Figure 18: Pitch Control Holding Pin from Inside

5.2 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.

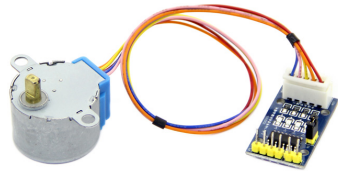


Figure 19: Gear Stepper Motor (Model 28BYJ48) with a Driver

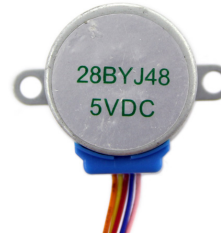


Figure 20: Stepper Motor 28BYJ48 Mounting Holes

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



Figure 22: DC Motor
attached to the Mo-
tor 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.

6 Appendix

