Robot League

Team: sdmay22-26

Client: Dr. Diane Rover

Advisor: Dr. Diane Rover

Team Members

Brogden Worcester	Client Interaction
Cheyenne Smith	Controller Designer
Dalton Holdredge	Machine Learning Engineer
David Quan	Frontend/Database Designer
Jordan Suby	Frontend Engineer
Joseph Holtkamp	Full Stack Engineer
Noah Brooks	Robot Designer
Tejas Agarwal	Structural Engineer

Team email: sdmay22-26@iastate.edu

Website: https://sdmay22-26.sd.ece.iastate.edu/

Revised: April 28, 2022

Executive Summary

Development Standards & Practices Used

Software Development Practices

- Agile/Waterfall project management process
- SOLID Design Principles

Engineering Standards

- 7007-2021 Ontological Standard for Ethically Driven Robotics and Automation Systems
- IEEE 802.11 wifi standard
- IEEE 1725-2011 rechargeable battery standard

Summary of Requirements

- Robot cars are controlled over WiFi to play a game of soccer
- Machine learning is used to determine the locations of each robot and the ball on the arena
- Game can be built/collapsed with minimal difficulty with provided instructions

Applicable Courses from Iowa State University Curriculum

- E E 185
- E E 201
- E E 230
- E E 333
- E E 425
- E E 466
- CPR E 288
- CPR E 308
- CPR E 329
- CPR E 430
- CPR E 489
- COM S 106
- COM S 309
- COM S 311
- S E 319
- S E 329
- S E 339

New Skills/Knowledge acquired that was not taught in courses

- Embedded machine learning training/implementation
- Embedded system design
- Raspberry Pi development
- Application development with Flutter
- Video streaming technologies
- Better understanding of software development practices
- 3D printing and modeling

Table of Contents

1 Team

1.1 Team Members

Joseph Holtkamp	Dalton Holdredge	Noah Brooks	David Quan
Brogden Worcester	Cheyenne Smith	Tejas Agarwal	Jordan Suby

1.2 Project Management Style Adopted by the team

Agile is meant for long term or perpetual projects, waterfall is designed for finite projects. Our project adopted a hybrid approach where we worked iteratively but where iterations prioritized what was useful in the scope of time for our project. Each week, we gathered what we had accomplished and decided what was the best step(s) to take in the next week or iteration of the project.

2 Introduction

2.1 Problem Statement

Our team will create two robots to compete in a soccer-inspired competition. They will be controlled by an application that communicates with the Raspberry Pi microprocessor on the robots. The game will be able to support human vs human playing and be responsive when users control the robots, and machine learning will be used to add features to the game using object detection.

2.2 Requirements & Constraints

Functional Requirements:

- Object detection is able to track gameplay objects in real time
- Object detection is accurate enough to have consistent and reliable locations of the objects
- Robot, arena camera and computer, and user interface device communicate over WiFi connection
- Robots must be robust enough to withstand collision
- Robots are light enough to carry around
- User can control the robot driving with intuitive control
- Easy to understand and navigate the application
- Ability to set up profiles
- Cross-platform compatibility
- Low latency video streaming
- Robots can maneuver adequately in the small arena
- Robots can have battery life long enough for sustained gameplay

Non-functional Requirements:

- Uses camera feed to provide information to the machine learning model
- Object detection latency is ≤ 250 milliseconds

- Machine learning model will have mAP@0.5 ≥ 0.75
- Uses Websocket/handshake protocol
- Haptic sensor on Android application allows user to drag to precisely control the robot's driving commands
- Server from ISU for hosting socket server and SQL database
- MJPEG video streaming latency is ≤ 250 milliseconds
- Robots can turn with a 0-inch turning radius (**constraint)**
- Robots can travel at a minimum speed of 3 mph (**constraint)**
- Robot battery packs have ≥ 2 amp-hours of capacity

Resource Requirements:

- 2 geared DC motors per bot (for a total of 4)
- 3 microprocessors: 2 Raspberry Pi 3b+ and 1 Raspberry Pi 4B
- Jumpers and wires for connecting the hardware
- 3D printer for printing the robot body and chassis
- Rechargeable power supplies for each bot and 120V AC power supply for the arena camera
- 2 Raspberry Pi Camera v2 modules (one camera each) for the robots to have first-person view within the application, and 1 camera above to utilize for object detection. Each camera requires at least 8MP resolution
- $600 budget for entire project (**constraint**)

2.3 Engineering Standards

The following are a few of the engineering standards that we will be implementing in this project. The first being 7007-2021 Ontological Standard for Ethically Driven Robotics and Automation Systems. For our project, our bots will support an autonomous mode, and a driver mode. This code needs to provide clear and precise communication between the bots' different systems and users in order to provide the most satisfactory game play for the player. The other standard we are implementing is 802.11 wifi standard: Our bots will be communicating over a wifi signal using a socket connection to communicate between bot and application in order to control the bots during play. Both these standards are needed to make sure we create an application that will allow the users to have the best experience.

2.4 Intended Users and Uses

We benefit from learning different skills and new languages for this project as something we can take to future employers. Also, this benefits others interested in the robotics and/or embedded systems fields by providing an example of what could be taught to future students. This project helps to show an application of Machine Learning and object detection. Our intended users are future engineering students who have an interest in robotics and computer programming.

2.5 Security concerns

For this project we considered security considerations to be a low priority. As the main purposes are education and entertainment, there's not a lot of incentive for a malicious actor to put in the effort to find and exploit any potential software vulnerabilities, as even if they were successful they shouldn't be able to do anything more nefarious than ruin the game. Physically speaking it is also possible for someone to steal or otherwise damage the vehicles themselves, but again there shouldn't be a lot of incentive to do so as they aren't much good without the arena. If for whatever reason you don't feel you can trust the people who have unsupervised access to where the arena is set up, the bots can always be kept separately in a locker when not in use.

3 Project Plan

3.1 Project Management/Tracking Procedures

Our project management style is a hybrid of agile and waterfall. We have adapted an agile-like task breakdown and storyboard to keep track of what stage each task is in and who is working on it. Due to the short timeframe of our project, the idea of success by incremental, continuous improvement would not be sufficient. To accommodate our limited time to implement the project, we have organized the implementation of our project in a waterfall model. Combining these styles offers us a structure for success in our time frame, with weekly "sprints" and agile-like task decomposition.

For version control, we are using Git. Our repository is hosted on Gitlab, as this was provided to us for this course. For communication and collaboration, our team utilizes Discord. Our communication with Dr. Rover is carried out via Slack primarily. Lastly, Jira is where our tasks, stories, and epics are tracked.

3.2 Project Milestones

Key Milestones:

- Create a single prototype for our robot league robots
- Interface with our robots over WiFi to be able to send and receive data between the application and the robots
- Create an embedded machine-learning model to recognize the ball at 75% precision with an IoU threshold of 0.5, with a maximum of 250 milliseconds for each detection
- Create a second prototype for a robot that can move more quickly, has all of the hardware components encased, is smaller, and has rechargeable batteries.
- Create the arena prototype
- Create a connection with the bots and retain connection 100% of the time
- Have the bots last 60 minutes of playtime
- Write python function to allow the robots to autonomously play the game

Software Milestones:

- Create application interface
- Create socket server for communication between robots and application
- Display video from the robot in the app
- Stream controls from app to robot

- Create database structure to represent users and game statistics

3.3 Resource Requirements

Part No.	Description	Quantity	Rate	Cost
1	PVC Pipes	13	$ 4.41	$ 57.33
2	T Connectors	14	$ 0.39	$ 5.46
3	Elbow with Side Outlet	8	$ 1.49	$ 11.92
4	Cross 4 way connectors	2	$ 1.69	$ 3.38
5	4 way T connector	1	$ 12.29	$ 12.29
6	Elbow Connectors	4	$ 0.39	$ 1.56
7	DC motors with wheels	4	$17.95	$71.80
8	Raspberry Pi 4 4GB	1	$ 55.00	$ 55.00
9	Raspberry Pi 3B+	2	$35.00	$ 70.00
10	Raspberry Pi Cameras	2	$ 24.99	$ 49.98
11	8 mp camera	1	$ 30.00	$ 30.00
12	Homemade 18V battery pack	2	$ 18.00	$ 36.00
13	PiSugar battery pack	2	15.00	$30.00
14	Touchscreen Monitor	1	$69.99	$69.99
15	HDMI Cable	1	$6.68	$6.68
16	6 Foot USB-C Cable	1	$13.99	$13.99
17	L298N Motor Driver Shield	2	$17.05	$34.10
18	ISU socket server	1	$0.00	$0.00
19	Database	1	$0.00	$0.00
			Total Cost	$559.48

4 Design

4.1 Design Context

4.1.1 Broader Context

Area	Description	Examples
Public health, safety, and welfare	The product must provide entertainment and possibly an outlet for stress for participants and audiences.	Reducing stress impacts the wellbeing of individuals in a wealth of ways.
Global, cultural, and social	Competitions and games play a role in most communities and provide a realm in which unrelated groups can be introduced to each other.	Games can bring people together from anywhere, especially when available over the internet.
Environmental	Our project should have little to no environmental effect. If any, slightly increasing the demand for rare earth metals used in computer hardware and electricity.	Increasing/decreasing energy usage from nonrenewable sources, increasing/decreasing usage/production of non-recyclable materials
Economic	The product must remain financially viable within your team, cost to consumers, or broader economic effects on communities, markets, nations, and other groups.	Product needs to remain affordable for target users, product creates or diminishes opportunities for economic advancement, high development cost creates risk for organization

4.1.2 Prior Work/Solutions

When exploring prior solutions, the technology used for driving modern drones with onboard cameras meets the solutions our apps are being designed for. From the camera on the drone, video feed is displayed on the controller or phone being used and controls are sent to the drone continuously.

RC cars exist in a similar capacity to what we need for the soccer-inspired game. They can be controlled in a similar capacity to our use case and can, in some cases, stream video feed.

Advantages (Drone Controller)

- Supports the functionalities we desire for direct interaction with our robots
- Is not limited by wifi connection

Shortcomings (Drone Controller)

- Most are not open source
- Does not support the streaming video over wifi
- User interface doesn’t support 3rd party viewing

- Many don't support the UI customizations we would require

Advantages (RC Cars)

- Can drive and interface with RC controllers
- Are light and quick
- Can stream video in some case

Shortcomings (RC Cars)

- Cannot interface with wifi
- Do not have the interfaces necessary to apply machine learning
- Often does not have the mass to be as effective in a competition as we require
- Most often do not have the capacity to turn in a small enough radius

Advantages (our design)

- Our product is a unique combination of the products available
- Our product is an interpretation of an existing game (rocket league) but in a real world context
- Our product could enable a unique learning environment in the fields of robotics
- Our product is sturdy enough to not break for repeated use
- Our product involved machine learning for a real life CPU opponent
- Our product will not require the user to be in the same room as the arena

Shortcomings (our design)

- RC cars, and controllers with video capabilities currently exist
- Our product is more expensive than existing RC cars in the market

4.1.3 Technical Complexity

Our product is a combination of robotics and full stack application development that requires the following components:

- Multi-platform application
  - Capacity to display low latency video display
  - UI/UX design for ease of use
  - Robot driving game control interface
- Socket server
  - Support multithreaded inter-device communication(websockets)
  - Hosted on an ISU virtual machine
  - Support HTTP request handling
  - Support SQL database connections
- Database
  - MySQL
  - Simple schema
- Arena machine learning
  - Stream video of entire game
  - Camera with computer vision

    - Analyzed object detection data to optimize the path for the robot to drive, using an A* pathfinding algorithm
- Robots
    - Stream video live via UDP
    - Execute a stream of controls from socket connection
    - Perform video streaming and motor execution concurrently
    - Agile enough to keep game speed competitive

Complex components:

- Machine Learning- Since our project proposal had an option of a single user playing the game with the CPU, as well as the 2 robots playing autonomously using the CPU, we needed to train a machine learning model so that the CPU can maneuver in such a manner as to give a decent competition to the user. We are utilizing visual object detection using the arena cameras, which will implement the ML model that we have created to detect the robots and the game ball.
- Autonomous Play- The ML model provides the location of the robots and game ball, which are used as inputs to the pathfinding function that will ultimately provide the direction for the robots to drive autonomously.
  Note: This was not achieved by our team this semester. While the tools were there including a pathfinding algorithm based on the object detection data from the arena, a motor control mapping based on desired direction, and remote control communication, we did not end up having enough time to complete autonomous driving via machine learning. While most of the pieces are there, the act of putting them all together is still a significant overhead to successfully implementing autonomous gameplay.
- Design- Designing the robot and arena turned out to be a lot more difficult and time consuming than we originally thought. The robot had to be strong so that it doesn't take any damage in case of collision with the other robot or the structure of the arena. We wanted our project to be highly mobile due to its structure size, therefore the robot had to be small and lightweight so that it's easy to carry. The robot also has to be aesthetically pleasing and present a professional appearance, all of which becomes difficult when we try to add more strength to it with a restrained budget. Similar complexity was with the arena as well. The arena was sized to be 4' x 4' x 5', its structure takes a lot of space and is difficult to carry, therefore we needed to come up with something that's strong enough to bear the collision force of robots and has good mobility.
- Socket server - The server needs to support HTTP endpoints for CRUD operations (create, read, update, delete) for any data manipulation. This is an industry standard. When it becomes more complex is the upgrading of HTTP requests to a websocket connection. The controls need to concurrently transmit from controllers to robots being controlled in parallel without causing notable delay for the game.

4.2 Design Exploration

4.2.1 Design Decisions

1. Multi-platform framework: **Flutter/Dart**
    - Single platform products have only a limited audience
    - Our team will more effectively be able to test our products by being able to reach multiple devices such as webpages and phone devices.
    - Targeting a single device would confine our team in ways that could limit product down the line.
    - Flutter apps can be compiled to a wealth of the common device platforms with a single code base.
    - Dart is an object oriented programming language which suits the strengths of our team.
2. What computer(microcontroller) to use for the bots: **Raspberry Pi**
    - Ability to interface with wifi.
    - Capacity to stream low latency video.
    - Compatible and affordable video cameras available.
    - Raspberry Pi's have a variety of models available.
    - Capacity for embedded machine learning.
3. Decide whether we are using embedded machine learning or doing the computation on a server.
    - The calculations need to be fast enough to be useful
    - Calculations on the server add latency
    - Calculations on the microprocessor may be limited for space or compute power.
4. Database: **SQL (MySQL or SQL Server)**
    - Our data is going to be very structured which lends to SQL in strengths
    - NoSQL is good for highly connected data which our data will not be
    - Our data should be easily organized into a database schema which is best suited for SQL
    - Additionally it can be hosted on the server we have from ISU which is more suitable than cloud solutions for our project's duration.

4.2.2 Ideation

For what computer to use we used the lotus blossom technique.

Microprocessor options considered:

1. Arduino Uno
2. Arduino Nano 33 BLE Sense
3. Raspberry Pi Zero WH
4. Raspberry Pi 3B+
5. Raspberry Pi 4

Frontend options:

1. React Native
2. Angular
3. Flutter
4. Native android app via Android Studio
5. Electron

4.2.3 Decision-Making and Trade-Off

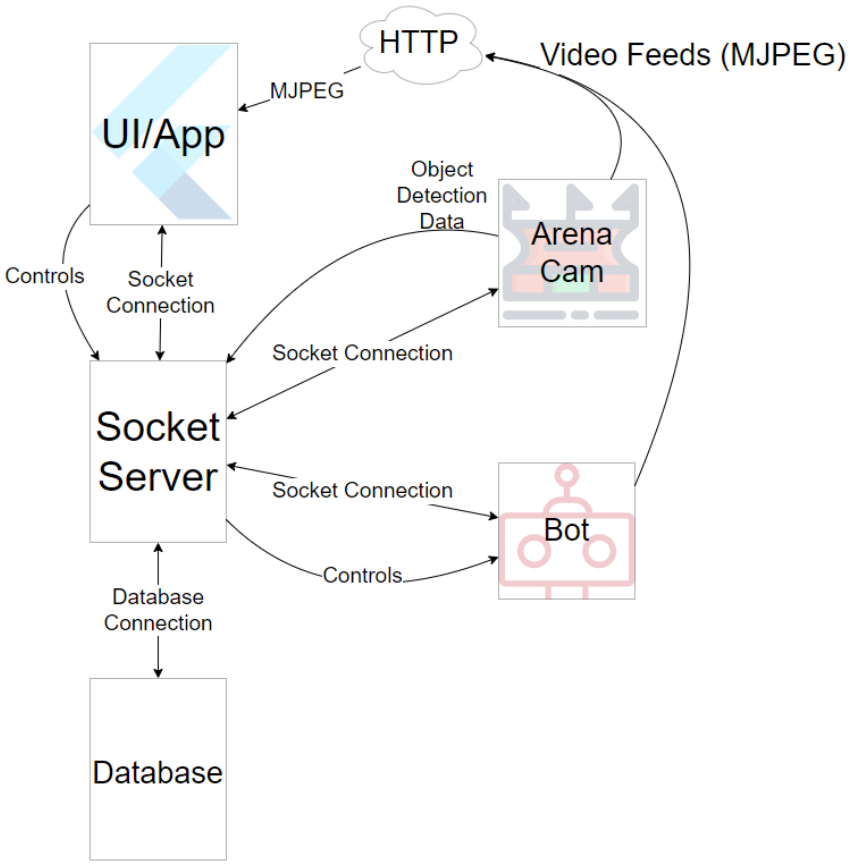
We looked at the official Raspberry Pi and Arduino sites in order to see hardware specification and whether they aligned without goals. We didn't decide on Arduino because they lack the processing power to be able to handle the object detection and server connections with low latency. We then discussed machine learning capabilities and decided that the Raspberry Pi 4 has enough RAM for the relatively intensive computation that comes with ML.

	React Native	**Angular**	**Flutter**	**Android Native App**	**Electron**
Supports android	X		X	X	X
Supports iOS	X		X		
Supports desktop	X		X		X
Supports web		X	X		
Supports devices we have access	X	X	X		X
Pure OOP			X	X	
Familiar to team members		X		X	
Results	**4**	**3**	**6**	**3**	**3**

4.2.4 Design Visual and Description

Arena for the game

We designed our arena with dimensions 4' x 4' x 5’ so it was large enough to allow gameplay and the camera to see the entire arena, but small enough so it can be taken apart and transported to another location if needed. We also made sure that the size of the goal post is big enough for the ball to go in even if there is a little bounce, but it is not too big so that the robot can enter. We used metal to make a ramp on the side of the arena floor as there was a chance of the ball rolling to the corners and getting stuck there. We also added a display screen on top that will show the score of both the players to make it more interesting.



High level architecture diagram

Pictured above is a high level drawing of the components involved in our project and how they interact. The socket server will orchestrate the various interactions between the bots, app, database, and arena camera. The cameras on the raspberry pi's will broadcast the video feed over HTTP in MJPEG format. The bots will send the host and port numbers to the server so that the user's applications can connect to the video feeds. The object detection data is also being planned to be sent to the server so that it can be overlaid over the video feed from the arena cam in a helpful fashion for users. Controls will be sent from the appropriate users to the correct bot for a given game.

4.2.5 Functionality

With the decision to translate to a three wheel system, the bot is able to change directions faster than with the prior four-wheel system. This was a useful change since the arena size was also shortened in this iteration of the project.

4.2.6 Areas of Concern and Development

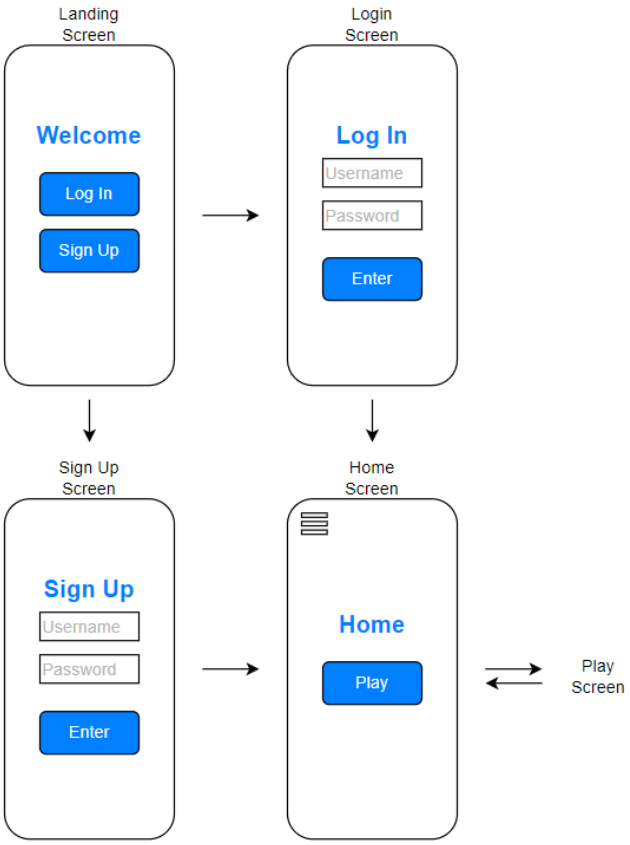
The issues from the last semester concerning the hardware side of the project were corrected by changing the top piece of the bot so that the images that allow for the machine learning model to track the bots and build the telemetry maps can be more easily read since the platform allowed for them to be larger.

The issues with streaming have been solved by using MJPEG and the clarity of video has been solved. If we had more time we would have implemented switching between the top-view camera and the camera on the robots. Currently, the feed is clear and allows for playing of the game without being in-person.

Our current design also does not implement the autonomous driving that we planned on implementing. All of the machine learning and detection of the ball and bots is all there, but we have not implemented autonomous controls for the bot. Also, we have an A* path finding algorithm implemented. If we had more time we would have implemented these features.

4.2.7 Problems faced in the design and implementation

Many problems came up as we went ahead with the project. While designing the arena for the game, we initially decided to have dimensions of 8’ x 4’ x 5’. However, we could not find a camera that would have a wide enough field vision and fit in our budget, or we would have to place the camera at the height of about 10 feet which was twice what we initially thought. Therefore we thought of using more than one camera to cover the playing field, but when we tried to implement the object detection model and other machine learning algorithms, we were unable to club the video analysis data coming from two different cameras and have a machine learning algorithm that will act accordingly, therefore we were forced to cut the arena size in half (4’ x 4’ x 5’). Although these dimensions solved many of our problems as we covered the entire playing field and performed all the required functions with high accuracy, things did not end there. Since the arena size has decreased, we also had to make the robots smaller to move freely on the playing field. This means a lot of changes in the robot design. We had to think multiple times about each and every component we used in the prototype and make sure none of them are taking aren't taking space for no reason. We cut down the size of the circuit that had voltage regulators and moved all the components around to make the size of the robot right with respect to the arena.



Application Flow Diagrams

Another issue we faced was the power supply for the robots. While deciding on the components for the final prototype, we made a circuit using passive components and voltage regulators to ensure the right power supply for the Raspberry Pi and motors used in the robot; however, somewhere in the middle, there was a loss of power which kept turning the raspberry pi off every once in a while. After a few debugging and troubleshooting methods, we concluded that there was some problem with the batteries, so we switched the AA batteries with a rechargeable battery pack specially designed for the Raspberry Pi.

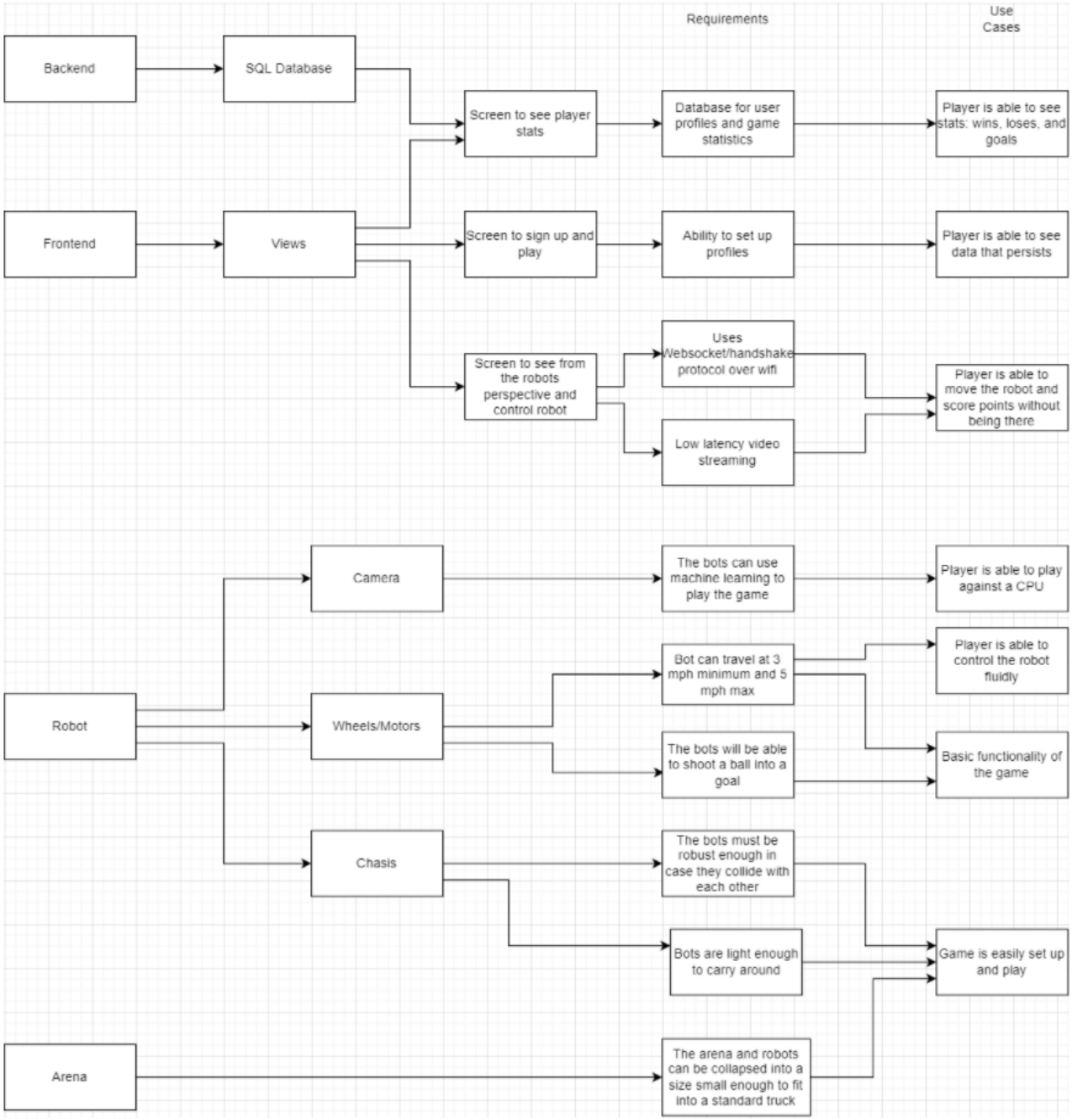
On the software side of our team, one problem we encountered early on was that to remotely access the Raspberry Pi's, we needed our server to be located within ISU's VPN. This we accomplished by requesting a dedicated virtual machine from the ISU ETG, which constrained us to go through the ETG to accomplish any networking tasks; this included opening and configuring ports for the server to be accessed. The ISU VPN's constraint also restricted where we could access video feed remotely from our raspberry pi cameras. To have a video feed available, it is required that the device accessing the feed be on the ISU WiFi or VPN. Many features become much more difficult to implement for remote access when going through the VPN.

4.3 Technology Considerations

The Raspberry Pi 4B was chosen to be the microprocessor to be utilized for the Arena camera which encompasses the frame collection for the machine learning module, we also had to change the microprocessor that we used for the bot from the Raspberry Pi Zero to the Raspberry Pi 3B +. this decision was made after several attempts of using the R.P. Zero, with all three attempts resulting in the Zero being burned up. Because the team was still under budget the increase in cost did not set the team back at all.

The SQL database was chosen as the ETG had multiple SQL databases available for students to use. We also wanted this project to be readily available after we graduate. This is the main reason why we didn't go with firebase as there would be a cost of time that would accrue. We stuck with a SQL based database as this is also what we were familiar with from previous classes. This decision was ultimately made by the whole team and we implemented our profiles using a SQL database.

4.4 Design Plan



5 Testing

Our testing plan was split between hardware and software.

On the software side, we tested using postman to ensure API endpoints on our server were being hit correctly. Initially, we set up a CI/CD pipeline in GitLab triggered when code was pushed to the master branch. The continuous integration portion of the pipeline targeted the golang code to run the formatter and compiler to ensure the code was passing. The reason we considered a pipeline to be useful was specifically to address the deployment of code to our VM. This did not happen due to the difficulty figuring out how to deploy the artifacts of the pipeline via SSH. This pipeline was set up to automatically run unit tests, but we ran out of time to write any such tests to apply. Since our app is meant to be cross-platform, we used emulators to test it in different environments. We also tested it using unbiased users to test if our design was intuitive for users to navigate and easy to play the game.

The hardware side measured the voltages throughout our bot to ensure that our power supplies were enough to sustain the bot through a game. We also tested the amount of heat produced by the bot to ensure no damage to the inner components and guaranteed that the bot would not damage any internals while playing the game for a sustained period of time. We also implemented a heatsink to ensure this would not happen and the temperature readings comply with what we set out for.

5.1 Initial Testing

In the application, widgets were manually tested in each platform (android and windows). During implementation we added in features such as buttons that display a sentence to the screen, such as “Testing” to make sure that the buttons were working correctly before moving to the next phase of the design. Another way we tested the software portions was to make sure our sockets were connected properly in order to send data for the motors over to the bots. We opened up our server connection and manually checked to see if the message was received correctly. When it comes to testing our hardware components, we feed the motors hardcoded movement commands to see how the bot reacts to a given input value. We also ran many battery-life tests to ensure that the bot is able to meet the play-time requirements.

5.2 Integration Testing

Critical parts to integrate was the connection between the camera feed and the mobile application. This was critical as the driver should be able to see what the robot is seeing and move the robot accordingly. We tested this by checking that our socket connections, one from server to bot and the other from application to bo, were transmitting network packages correctly and whether there were any packet drops.We used tools like ping to check if the Raspberry Pi has a stable connection for the same. Since if the Raspberry Pi is unreachable then there would be no way for packets containing the controls for the motors to be received correctly on the bots’ socket connection. While for the application to server connection we would test that packets were being received by sending simple strings or values over the connection.

Another critical part was the camera being able to identify the important objects in the arena: shapes in the robot, ball and the goal. was critical as our game requires these to be identified. We tested certain parts of our machine learning model to see if it was able to identify the objects with high accuracy. We also tested the precision for the model while the objects are moving at their maximum speed.

5.3 System Testing

Since our robots will be operating under predefined circumstances and rules, testing the system as a whole was done simply by setting up the game in the scenario that we want to test and observing if everything

proceeds in an acceptable manner. For example we just gave the robot set of input commands that equates to something as simple as “proceed forward at maximum speed for 2 seconds” we tested that the connection between the robot and the controller is working, the robot was able to decode the “move forward” instruction properly, the motors responded to that instruction by actually moving forward, the maximum speed was also an appropriate speed(as defined under project requirements) for our gameplay, and the robot stopped when expected to, all at once. This was done by both, our physical prototype robot and on an online simulator depending on what’s more convenient for what was being tested. While performing these tests it was important that we cover every possible scenario that can occur throughout playing the game and that we were thorough enough to ensure that the desired outcome was reliably achieved and not just coincidentally working on the few times it was tested.

5.4 Regression Testing

Most of our regression testing consisted of reevaluating that our subcomponents behaved as expected after deploying code to them. If we updated the server’s code, even if it had worked on localhost, we manually tested that devices could utilize the changed code externally as expected. When the raspberry pi code was updated, we ensured that the change to the code did not break the other connected systems and that it performed as expected when running in the designated environment. This was repeated in all areas of our project and is how we ensured that each component worked as expected after an update.

5.5 Acceptance Testing

With our end product being designed to be played as a game, it was best for many of our functional and nonfunctional requirements to be measured by third party input. In order to test our ease of usability and UI/UX requirements, we were required to see how well end users could learn the game controls without assistance since our team will not be around every time to teach how to play the game.

This could have been measured either in a guided way by timing and observing how long it took an end user to learn the game mechanics, and/or by asking the user about their experience with the controls and interfaces via survey. Additionally, nonfunctional requirements such as the aesthetic appeal of the bots and application were discussed with our client to see how our product stood up to meeting aesthetic appeal to end users.

Our client was involved in the acceptance testing by playing the game, driving the bots, interacting with our application, and providing feedback through each testing phase to help us iron out details before displaying it to all end users. Our acceptance testing took place over several demos with our client so that we could improve our end product with each iteration of feedback. We did not get very much in terms of final acceptance testing and feedback, as the time constraints did not allow for these tests.

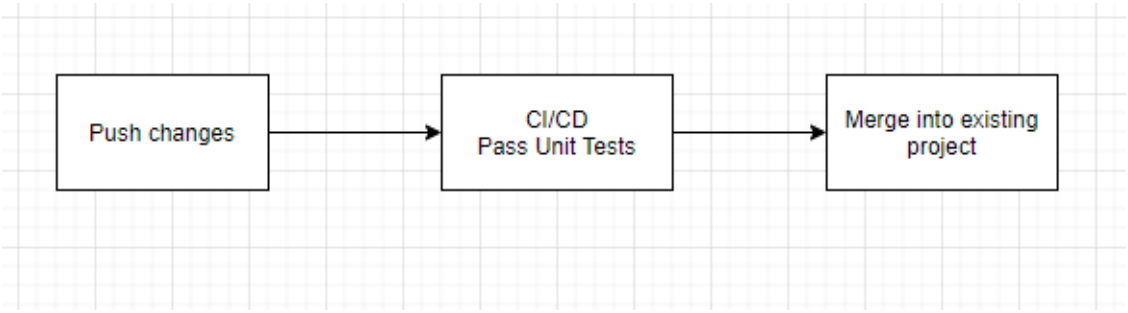
5.6 Results

Our integration testing was successful and we ensured that our socket connection was running smoothly. We did this by measuring the delay between the robots movement and how fast the camera updated that on our application. The delay was negligible. We also tested our endpoints using Postman, which ensured our endpoints were being hit and that we could hook up the backend with the frontend through API calls.

System testing results were also successful; there was no delay when it came to the server communicating with the bots. The movement between the bot and the controllers was fluid and we also got approval from

our advisor on the response time of the robot. We also tested the server without the application and the pure commands of moving the bots were fluid.

Regression testing was done using CI/CD pipelines and ensuring that new changes didn't break current tests that we've implemented. Currently, our application passes all the tests we've implemented in the pipeline and this has ensured no merges happen that could break our application.



Testing process on code level

The figure above shows how our code testing process will be completed by using Git as described in section 5.6. We will be pushing code from separate branches onto the master branch. Before it can be applied to the master branch, it must pass the CI/CD pipeline tests that show that there are no errors or conflicts between commits before being merged into the existing code on the master branch. In order to test for UI appeal and ease of use, we will be using alpha users to describe their experience while learning the game and how well the UI was laid out.

6 Implementation

6.1 App-Dev Team

- Database
  - ISU ETG allocated us a MySQL database on their senior design database server
  - Our schema is simple for user data and game record keeping
  - The DB is accessible on the ISU VPN which is in reach of our web server
- Server
  - Implemented in Go for its support of concurrent programming
  - Supports multiple parallel streams of controls from application instances to raspberry pis
  - Use of goroutines(light multithreading) ensures that inter-device communication is nonblocking
  - Controls are communicated via TCP websocket connection
  - Server is run on the ISU VM provided to us giving it access to the ISU VPN
  - Has a persistent connection to the MySQL database
  - Has HTTP endpoints for any CRUD operations (Create, Read, Update, Delete)
  - Is available outside of the ISU VPN for any external interactions
- Flutter App

- Implements simple user authentication for IAM(Identity and Access Management)
- UI is simple and relatively easy to navigate (best experience on android devices)
- Supports dark and light themes based on system preferences
- Views include landing screen, login screen, signup screen, home screen, profile screen, and control interface
- Supports android and windows devices
- Supports but is not well tested on web platforms
- Control interface
    - Designed for minimal, intuitive appearance
    - Sends commands via websocket client connection
    - Interface is responsive based on gestures of user to control bot
    - Displays live video stream from bot's onboard POV cameras or arena overhead camera

6.2 Hardware Team

- Bots
    - Registered with ISU WiFi
    - Receive control signals from socket server via websocket client connection
    - Broadcast POV video stream from onboard cameras via UDP
    - Double check prototype plan with P.H.I.L.L.I.P components sizes
    - Order components needed (motors, pi, L298N, camera, batteries) for second prototype
    - 3D print the components for the bots
- Arena
    - Build arena frame
    - Adding the particle board walls
    - Fix the arena camera and scoreboard to the arena
    - design, create, and attach the project sign

6.3 Evolution of Our Project

User Interface

Initially, we had overcomplicated the plans for the mobile app which led us to have little direction when beginning implementation. There were a lot of plans for views where users could configure their account heavily and view data such as game statistics based on the current game and the user's profile/preferences. As the project progressed, it became apparent that the boilerplate of the app was not focused enough on the core functionality of our project. Functionality the app is a remote controller for our robots. The focus needed to be on the control interface and video display. As it turned out, this shift in focus resulted in a minimalistic user interface.

The video feed took some time to figure out which streaming mediums were compatible with both the raspberry pi's and the flutter framework. After some research, .mjpeg streams seemed to be the best supported and the lack of audio was not an issue. With some manual testing, we determined that the latency of the video feed, while noticeable, is not a breaking issue for gameplay.

Our control interface took an iterative approach and was our best usage of the agile methodology. When designing the MVP (minimum viable product), the interface was 9 buttons laid out in a 3x3 grid with an

arrow for each direction. The next iteration, speed controls were added for high, medium, and low. After that, we began to look into gesture based controls for mobile instances of our application. The initial implementation mapped each gesture onto the original 9 button model and kept 3 buttons for speed adjustment. Our final implementation dug into the usage of the gesture detection widget we used to implement this all. Using some trigonometry, we mathematically modeled the drag motion and mapped it onto the bot motor controls. Rather than the position being classified as its position relative to the center of the screen, we classified the position of the user's drag relative to the drag's origin. This allows users to have a more intuitive experience. Next we mapped the direction of the drag into 4 quadrants where forward and backward were just that and left and right reached zero turn radius. When it was all done, we adjusted the function we used to classify the user's drag to calibrate the sensitivity to be more usable based on the screen size. This final iteration also allowed us to remove the speed buttons to be removed as the speed was calculated by how far your drag had reached from its point of origin relative to the size of the device's screen.

Final overview design of controller

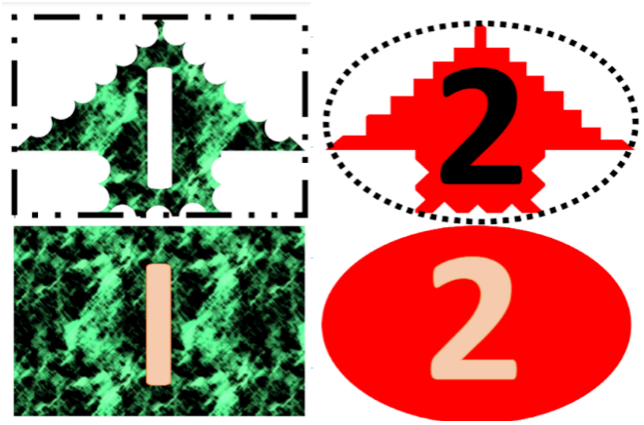
Server

Go(golang) is the programming language chosen to implement our server in. This choice was made due to Go's simple but powerful implementation of concurrent programming and its large capabilities with minimal usage of 3rd party libraries. Our web server was a simple, replicable implementation using Go's native http package. Our websocket model, while possible with the standard library, was implemented with the gorilla/websocket implementation and loosely based on a websocket chat model. This was adapted to support monodirectional communication of commands from the controller(app instances) to the Raspberry

Pi's on the bots as a stream of bytes acting as a proxy to direct the flows of information. Each socket connection is created in a first-come-first-served fashion where a controller is paired with a bot; if there is not a match, the connection simply waits for a corresponding device to connect. If multiple connections of the same device type, the latter connections wait in a queue behind the first.

This model supports "infinite" connections to be made. "Made" is the keyword here as the handling of socket disconnection is still rather messy. The large array of interdependent components in our project prevented some improvements such as handling device disconnection from being taken care of cleanly. This is simply due to the large amount of overhead presented by taking on 5+ major subcomponents that work in tandem when our team was not an expert in any of the technologies involved. As it stands, the server must be restarted each time that devices disconnect for the connections to be properly made again. While not ideal, this works for a game by game approach.

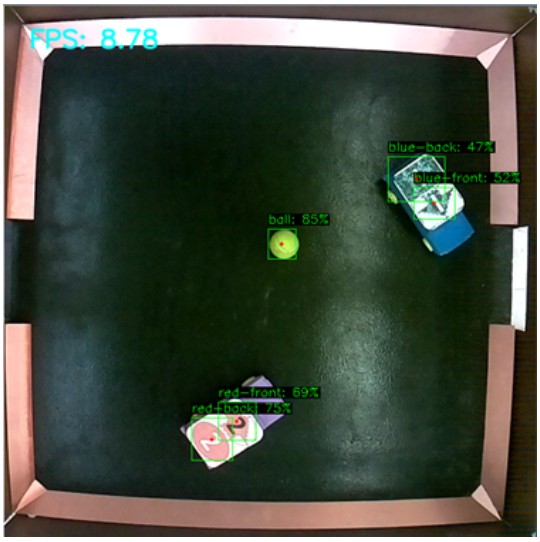
Object Detection



Object Detection Images

Instead of teaching the machine learning model to recognize the physical robots, we decided to instead teach the model to recognize the shapes shown below. Each robot has two shapes on top of them, which allows for the object detection model to differentiate between the two robots. The reason that there are two shapes on top of each robot, instead of one, is because this allows for the model to determine which direction the robot is facing. In order to use the object detection data to provide input to maneuver the robots, we must have the ability to determine which way each robot is facing, in addition to their locations. Because the object detection model is only able to provide the location of each object, a single object would not be sufficient to determine which direction each robot is facing. Basic trigonometry is used to find the direction that the robots are facing.

The shapes are noted to have uneven edges and varying texture for the color, and these decisions were made in order to increase the effectiveness of the object detection model. The model was more efficient at being able to locate shapes if they had uneven boundaries and varying textures, and we increased the mAP@0.5 from approximately 75% to approximately 100%.



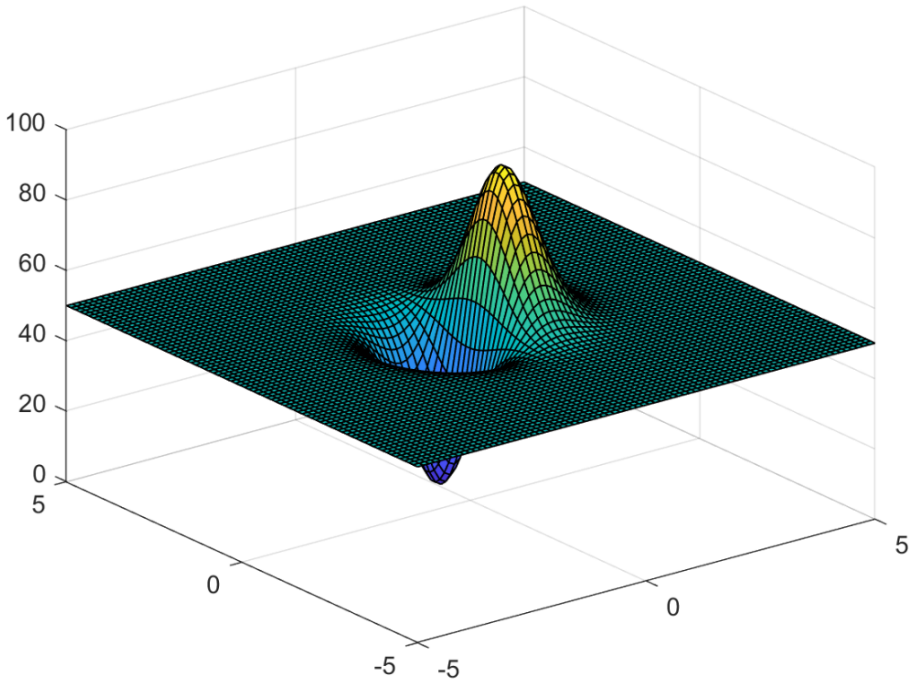
Object detection visualization from the arena camera

An illustration of our object detection model is shown in the image above. In addition to training the model to recognize the shapes shown above, we used a tennis ball to train the model for the game ball. Tennis balls are inexpensive and easily replaced if they are lost, so that is why we chose them. For each object detected, the model determines the bounding box around them (shown in green). The model also calculates the center of each object (shown by the red dot), which is necessary in order to determine the location of each robot and the ball during the gameplay. We painted the bottom of the floor black, which further increased the effectiveness of the object detection model as it increased contrast between the robots, ball, and the floor.

Modifications to the Original Bot

After measuring the components on the original prototype and accounting for the size of the future battery pack and motors, our team 3D modeled a more appealing car body to enclose the components and have a

more aesthetic appearance. During the second prototype, we discovered that the camera placement on the first 3D printed body was too low for an effective field of vision; to fix this we added the camera module to the roof component. The upper half of the bot is able to separate, allowing easy access to the electrical components. A large flat roof is needed for symbols used by machine learning. The final robot design utilizes a ball bearing wheel instead of moveable front wheels. This change was to improve mobility in the smaller arena, as the robot was now able to turn with a zero radius turn diameter.



3D plot representing the ball weights used for pathfinding algorithm

In order to add the ability to have autonomous driving, we had to implement a pathfinding function so the robot would be able to effectively defend its goal and potentially score a goal on the other player. We used the A* pathfinding algorithm, which uses a weighted 2D matrix to calculate the optimal path. The visual shown above shows an illustration of what the weighted matrix would look like if the ball was centered at (0,0). The lower the weight value in each index, the more likely the robot is to take that as part of its path. Intuitively, the path will prefer to go "down" in the matrix shown above, instead of "climbing up" the large peak. The destination is in the center, so the robot would plan a route that minimizes both the distance traveled to get to the ball, as well as the sum of the weights as it passes through them. By having the "valley" of the weighted matrix placed on the side of the ball that faces the defending goal, this pathfinding method provides a path for the robot to hit the ball that will not end up hitting the ball back into its own goal, which would score a point for the other team. The high peak on the opposite side serves the same purpose, as it would be unlikely for the optimized path to go through the peak.

In addition to the weights being added to reflect the location of the ball, large-value weights are added to the locations where the opposing team's robot is. This is to prevent the autonomous robot from trying to drive through the other team's robot to get to its destination.

7 Closing Material

7.1 Discussion

This semester, we have been working on implementing the design of our Robot League game's hardware and software portions from last semester. During the fall semester, we had a working prototype that we continued to build on as the semester progressed allowing for multiple iterations that helped improve the design as the project continued. The largest amount of progress that took place during this semester was on making an application that would house the controller that would allow users to drive the bots wireless either in person or through a camera feed for more remote play. As well as, integrating the robot to the controller and all our other parts that were a requirement to complete our project. We were able to create a basic controller set up that would allow for the proof of concept that we could drive the bot over a wireless connection and be able to meet our constraints that required it to be responsive from the video feed to movement. Once we confirmed our controller worked in that sense we could take the next steps to get to the final product we did which is a simple but more intuitive controller.

During this process of changing designs and layouts allowed for us to be able to take our project idea, a robotic game of soccer, and go "how can we make this enjoyable to play" by asking that it allowed us to come up with various ideas of how to solve this question while staying within our set requirements and addressed the issues that we came by along the way. Whenever we came to a new issue, for example how to change our controller to be more intuitive, we spent hours working through what the best approach would be and how to implement it. There were a few times throughout the process that we would start working on an idea and when we worked to implement it. We would find out that our design would not work and we had to sit down and rethink our approach to the problem. In the end, having these types of problems to solve along the way helped us grow on what we learned through our time in the classroom and learn new skills on how to handle problems we had never encountered before.

We were hoping to implement more when it comes to our application, but our current iteration is a fun accessible game that we're proud of. The game is fluid and the features we have are more than enough to enjoy and appreciate our game.

7.2 Conclusion

In conclusion, something that was just an idea from a team member's mind has become a reality. Many of the core features are implemented and we built upon a simple design. We've toyed with many new technologies: Machine Learning, Flutter, Golang, and API's. We've spent countless hours designing and refining the arena and bots. This project has taught us many things outside the scope of our classes and has provided an insight on what it means to be an engineer. Learning and manifesting our ideas into physicality has been a great experience. This project has provided us with confidence in our future endeavors and hope that this project was a good stepping stone before graduating and becoming real engineers.

7.3 References

[1] “Automl Vision Object Detection Documentation | google cloud,” *Google*. [Online]. 9510Available: https://cloud.google.com/vision/automl/object-detection/docs., [Accessed: 06-Dec-2021].

[2] B. Peabody, “Server-side I/O performance: Node vs. PHP vs. Java vs. go,” *Toptal Engineering Blog*, 11-May-2017. [Online]. Available: https://www.toptal.com/back-end/server-side-io-performance-node-php-java-go, [Accessed: 06-Dec-2021].

[3] “Flutter documentation,” *Flutter*. [Online]. Available: https://docs.flutter.dev/, [Accessed: 06-Dec-2021].

[4] “Flutter_mjpeg: Flutter Package,” *Dart packages*, 11-Jun-2021. [Online]. Available: https://pub.dev/packages/flutter_mjpeg, [Accessed: 06-Dec-2021].

[5] N20 Electric Mini Micro DC geared motors 3v 6v 12v low speed 15 600rpm in DC motor with mounting bracket wheel tire for DIY Toys: DC motor: - aliexpress,” *aliexpress.com*. [Online]. Available: https://www.aliexpress.com/item/4000125484788.html?spm=a2g0o.detail.1000023.9.532c2e09BGMML6, [Accessed: 06-Dec-2021].

[6] “Object detection : Tensorflow Lite,” *TensorFlow*. [Online]. Available: https://www.tensorflow.org/lite/examples/object_detection/overview, [Accessed: 06-Dec-2021].

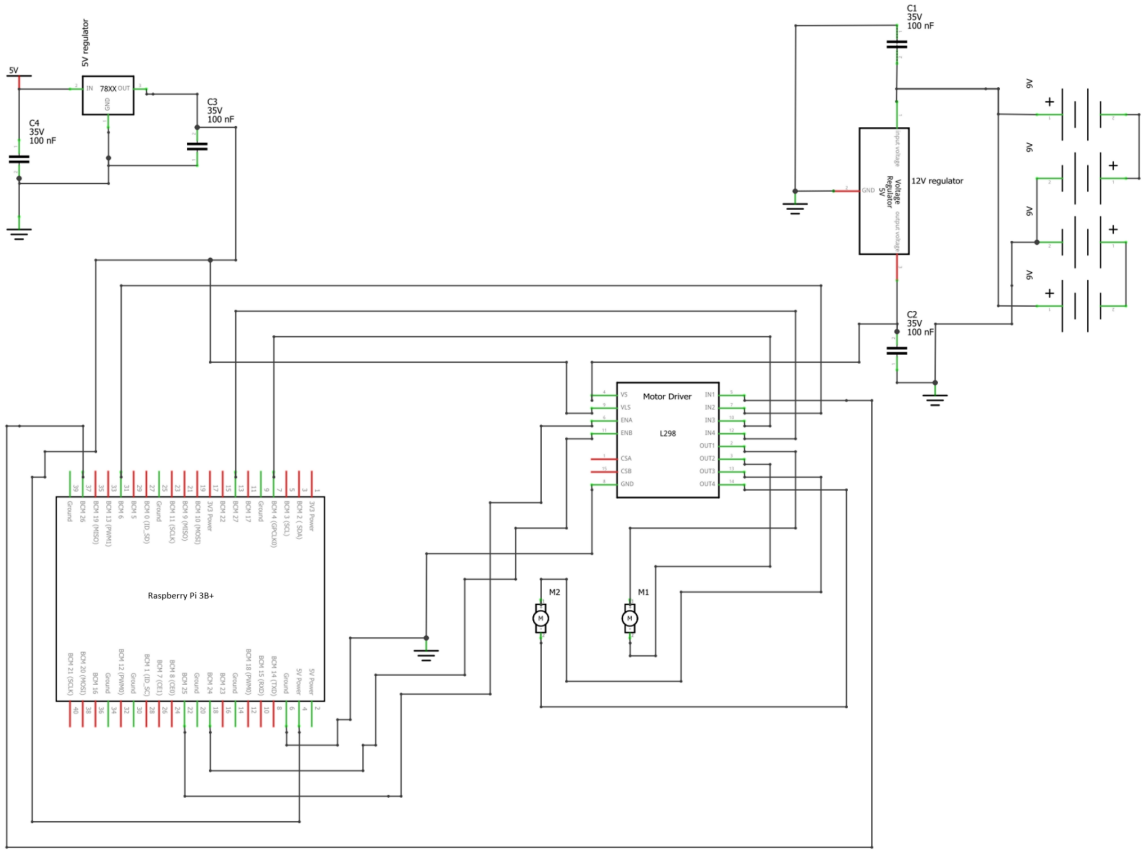
[7] “Raspberry pi documentation,” Raspberry Pi. [Online]. Available: https://www.raspberrypi.com/documentation/, [Accessed: 06-Dec-2021].

[8] V. Q. T. (vuquangtrong@gmail.com), “Camera live streaming using Mjpeg format,” *Code Inside Out*. [Online]. Available: https://www.codeinsideout.com/blog/pi/stream-picamera-mjpeg/, [Accessed: 06-Dec-2021].

[9] “IEEE Code of Ethics.” *IEEE*. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html/, [Accessed: 06-Dec-2021].

7.4 APPENDICES

Appendix I



Robot Schematic

Appendix II

Rules and Gameplay

Rules

To win the game you must be the first player to score 11 points.

- One point is awarded when the ball passes through the goal of the opposing player.

General gameplay:

The game is played on an arena of 4’x 4’

Upon a goal, the players reset the bots and a player must place the ball into the center.

Important Notes:

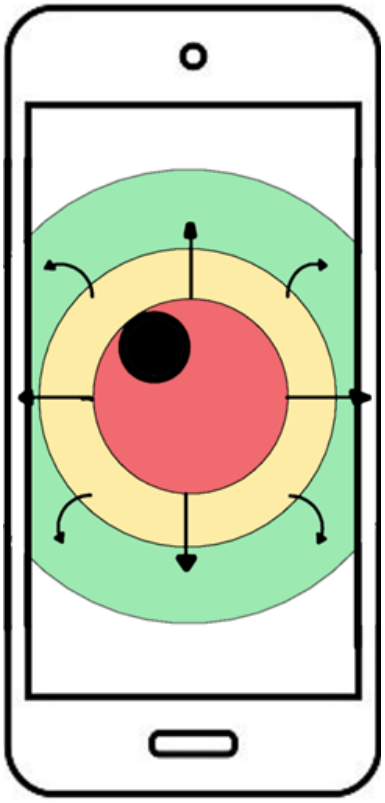
- The server must be running for the controls to connect successfully.
- To view the video feed, it must be running and with the default configuration, it must be connected to the ISU VPN or on ISU WiFi.
- If the video feed is not running or unable to connect, you will see an ugly red message at the current implementation.
- Restart the server when starting a game or after device disconnection.

Controls via desktop application:

Pictured below are the two methods of control of the bot. The desktop controller relies on the number pad having assigned a driving function. Each arrow is given set values that drive each motor separately, this means that outside of the: forward, backward, left, and right; there is also a curve forward-left, forward-right, etc.

Controls via Android application:

With the haptic control on the mobile app, the center of the controls is relative to where the user first touches the screen. The direction to where the user drags the finger controls the direction that the bot will travel, while the magnitude of the drag controls the speed the bot will travel in that direction.



If the screen has an X drawn along the y=x and y=-x around the origin of the starting point of the drag motion, you get 4 quadrants, forward, backward, left and right. Forward and backwards are just that. If you are in these quadrants, a movement towards right or left will shift the direction in that way. In the left and right quadrants, these are the zero turn radius regions. Use these for sharp turns.

Run the App (Windows)

1. In our team website (https://sdmay22-26.sd.ece.iastate.edu/), navigate to the Installation section.
2. Navigate to the Windows subsection.
3. Download either the Windows Setup executable or the .zip folder.
   Note: the Windows Setup executable will likely be flagged by both your web browser and your computer as dangerous and you will be advised to not run it. I promise it isn't a virus, but we do not know how to properly license or distribute our app in a trusted way at the moment.
4. If running the Windows Setup executable, follow the instructions of the installer. If using the .zip, extract the files to your chosen location.
5. Finally either run the Robot League Desktop app or run the app.exe file located where the app is installed.

Run the App (From Source)

1. In our team website (https://sdmay22-26.sd.ece.iastate.edu/), navigate to the Installation section.
2. Navigate to the From Source subsection.
3. Download the .zip file from the link provided and extract it to the location of your choosing.
4. To run this code, you must install the Flutter runtime from the following link: https://docs.flutter.dev/get-started/install.
5. Follow all instructions to get Flutter set up properly and make sure the "flutter doctor" command passes.
6. Once you have flutter installed and running, you can open a terminal and navigate to the root of the code you downloaded from our website.
7. With the "flutter run" command, you can choose the platform you want to run the application on between desktop or web. If you have any android devices, you can plug them into your computer via USB and the flutter run command will detect this. Otherwise if you have an emulator for a device, you can run it on there. Lastly, if you happen to have a MacOS device, you can run our app as a Mac or iPhone application, however this is not something we designed our app for and is not guaranteed to work. The recommended route is to run the app on an android device, otherwise defer to running as a Windows app.

Server Setup/Configuration

1. In our team website (https://sdmay22-26.sd.ece.iastate.edu/), navigate to the Installation section.
2. Navigate to the Server subsection.
3. Download the .zip file from the link provided and extract it to the location of your choosing.
4. To run this code, you must install the Go compiler from the following link: https://go.dev/doc/install.
5. Once Go is installed and added to your PATH environment variable, navigate to the code downloaded and run either "go run ." to run in an interpreted fashion, or compile with "go build". You may need to run the "go get" command to install dependencies for either method, you should be prompted. If you compiled successfully with go build, then you need to run the server.exe file output.

Below are some configuration steps you can take:

6. The script may be flagged as malicious, if you're worried, go ahead and read the code, it isn't long. Also Go is fairly easy to read, even if you don't know Go.
7. If the desire is to access this server remotely, you must make sure to set up a static IP address or DNS and/or port forwarding. You must restart the server when connection issues happen.
8. You can configure ports in main.go in the call to listenAndServe(), alter the port number. The default port is 50200.
9. The database connection can be configured in db_access.go.
10. Endpoints can be added in http_server.go.

Charging the batteries

CAUTION: Ensure the game is not running and the Raspberry Pi battery pack is turned off prior to removing the batteries.

For the motor batteries in black cases:

1. Lift the roof up slightly from the front side.
   WARNING: The camera cable connects the roof piece and the raspberry pi on the robot. Proper care must be taken to prevent the camera cable from becoming detached while opening the roof piece.
2. Carefully remove the black battery cases.
   WARNING: Exercise caution when removing the battery cases, as pulling on the wires might break off the connections.
3. Open the lid on each case and remove the rechargeable batteries..
4. Plug in the batteries with the supplied four-way Micro-USB to USB cable and allow them to charge until the indicator lights on the batteries turn blue.
5. Once the batteries are charged, unplug them and insert them into the case and connect them. Close the lid for each of the battery cases.
6. Place the battery cases back on its place and lower the roof carefully back into its original position.

For the 5V battery pack connected to Raspberry Pi:

1. Ensure that the battery pack is turned off by placing the small power switch into the “off” position
2. Lift the roof up slightly from the front side.
   WARNING: The camera cable connects the roof piece and the raspberry pi on the robot. Proper care must be taken to prevent the camera cable from becoming detached while opening the roof piece.
3. Attach a Micro-USB cable to the Raspberry Pi and let the battery pack charge.
   WARNING: Exercise caution when adjusting the battery pack, as pulling on the wires might severe the connections.
4. Once the battery is fully charged, unplug the Raspberry Pi.
5. Place the Raspberry Pi and battery pack in its original location and lower the roof carefully back into its original position.

Bot Set Up

1. Lift the roof up slightly from the front side.
   WARNING: The camera cable connects the roof piece and the raspberry pi on the robot. Proper care must be taken to prevent the camera cable from becoming detached while opening the roof piece.
2. Toggle the small switch between the Raspberry Pi and the battery pack attached to it. A blue light will turn on signaling that the power supply is turned on for the Raspberry Pi.
3. Lower the roof carefully back into its original position
4. Place the robot on the starting point on the playing field.