

CS 30700

LogBotics | Design Document

By Team #24:

Team Members: James Gilliam | Roger Braun | HyunShu Kim | Jenna Rigdon

Index

Purpose	3
Functional Requirements	3
Data Logging	3
Data Visualization	4
Data Export	5
System Configuration	5
Non-Functional Requirements	6
Performance	6
Scalability	6
Usability	6
Security	7
Portability	7
Development:	7
Design Outline	8
High-Level Overview	8
Components	8
Detailed High-Level Overview	9
Design Issues	10
Design Details	13
UI Mockups	15

Purpose

The **LogBotics** project aims to address a critical gap faced by **First Robotics Competition (FRC)** teams when diagnosing robot performance issues during competitions. Existing data logging and visualization solutions are either too simplistic or lack essential features, such as motor activity tracking, sensor feedback analysis, and control system operation monitoring. Teams often need help troubleshooting problems during matches, and current tools are insufficient for comprehensive data analysis. By providing a more robust and user-friendly application, **LogBotics** will help teams better understand their robot's performance through real-time data collection and visualization.

Additionally, **LogBotics** aims to make data analysis intuitive and accessible. The system will offer real-time graphical visualizations, enabling teams to observe motor and sensor data, control system statuses, and robot movement during both autonomous and teleoperated match phases. With the ability to export data in various formats, such as CSV and PDF, teams will also have the flexibility to analyze historical performance and share data with stakeholders. This comprehensive solution will save teams time and provide actionable insights, helping them improve robot performance and decision-making during high-pressure competition scenarios.

Functional Requirements

Data Logging

As a user,

1. I want the system to automatically log motor activity during matches so that I can diagnose issues.
2. I want the system to automatically log sensor feedback during matches so that I can diagnose issues.
3. I want the system to automatically log control system data during matches so that I can diagnose issues.
4. As a user, I want to be able to view each motor's activity separately so that I can see issues with specific motors.
5. I want the data to be automatically backed up such that I can restore the data in case I delete them by mistake.

As a developer,

1. I want a framework that can receive, store, and send logged data to the app so that data logging and display can be implemented in the app.
2. I want there to be a motor object that tracks data, errors, and states of different motors so that visualizing motor activity can be implemented.
3. I want there to be a robust library of errors with the robot that can be detected with the logged data so that error messages can be implemented.
4. I want to have a stored database of information on potential errors that could be associated with the robot's performance so that I can warn the user of potential problems the robot is facing.
5. I want to be able to view error messages from the CAN bus on the control system so that they are clear and easy to see.
6. I want to implement an API for retrieving real-time data from the robot's control system so that it can be streamed directly into the logging system without delay.
7. I want to create unit tests for the data logging module to ensure that it accurately captures and logs motor activity, sensor feedback, and control system data.

Data Visualization

As a user,

1. I would like to be able to see graphs of the motor, sensor, and control system activity so that I can better understand the current state and trajectory of the robot. And, if time allows, as a user, I would like to see tables of the motor, sensor, and control system raw data so that I can catch specific errors or anomalies with the robot system.
2. I would like to be able to view the robot's path during the autonomous part of a match as a map so that I can record/visualize the robot's resulting motion in a match.
3. I would like to be able to view the robot's path during the teleoperated part of a match as a map so that I can record/visualize how the robot is responding to commands overall.

As a developer,

1. I would like there to be a framework to back-calculate the expected path of the robot based on motor output data so that the motion visualization can be implemented.
2. I would like there to be a GUI to prompt the user for wheel dimensions and placement on the robot as well as the number of wheels (allowing between 3 and 6) so that the path can be calculated.

Data Export

As a user,

1. I want to be able to export individual plots as visual files (jpeg, pdf, etc) so that they are easy to open and use elsewhere.
2. I want to be able to export individual plots as .csv files so that they are easy to open and use universally.
3. I want to be able to export all data at one time or select which data plots to export so that such an export process is simple and/or storage/organization/time isn't wasted on unused data.

As a developer,

1. I want to implement robust error handling for the export feature to ensure that if the export process fails due to file permission issues, clear error messages are shown to the user, so that they know how to troubleshoot and correct the issue.
2. I want to implement data compression for exported files, so that large datasets can be exported efficiently without overwhelming storage space or upload/download time.

System Configuration

As a user,

1. I want to be able to set my team number and team name so that data won't be confused with other teams.
2. I want to be able to create an account on the app so that set up and save my personal settings/customizations.
3. I want to be able to access my account on any device that has the app and log in and out of it so that the software is easier to access and multiple accounts on the same machine are possible.
4. I want to be able to customize the color settings of the app to increase/decrease contrast so that the app is more ergonomic and tuned to preferences.
5. I want to be able to select different visualizations of data logging (various plots, various colors, etc.) so that the data is easier to see and customizable to preferences.
6. I want to be able to add or delete data visualization plots from the screen so that the most important data to see is always displayed/prioritized.
7. I want to be able to open multiple tabs for viewing data so that large numbers of data visualization plots are easier to work with.
8. I want to be able to label these tabs so that data is more organized
9. I want to be able to move plots from tab to tab so that data plots can be grouped according to needs, priority, type, ect.

10. I want to be able to freely label plots so that plots are easier to keep track of according to my specific needs.
11. I want to be able to view the source code of my robot.
12. (If time allows) I want to be able to develop my robot code in an integrated development environment within the application.

As a developer

1. I want there to be a server-client framework (fat-client) so that accounts can be changed and accessed on other devices.
2. I want there to be a plot object so that different visualizations, data types, windows, and tiles can be implemented.
3. I want there to be (or to find) a Windows tab framework and tile objects so that multiple tabs with different data can be implemented.

Non-Functional Requirements

Performance

The data collecting tool will be designed to allow for transmission of data, in real time, to the user's computer during the match, where the user can then instantly access and display the data within the desktop application. There should be little to no time where the user has to wait for transmission of data from the robot to the user's host computer.

Scalability

We intend for the data logging tool to be scalable in that it should be able to be downloaded and used as a package/application on any number of Windows computers. The tool should readily be accessible to all the participants of FRC should they desire to use it. In addition to this, since this tool will eventually be open source, we intend to make it maintainable and developer-friendly, allowing other developers to add their own features if they would like.

Usability

One of the most significant components of the desktop app is the ease-of-use. Users should be able to open their desktop application and easily be able to interact with their data in a variety of different, intuitive ways via graphs, tables, and other data formats. In addition to this, users should be able to perform file exports so that they can compare data from previous sessions with their robots. These aspects of the tool are paramount to the user's experience.

Security

Since the application is a locally hosted tool that will run on the user's computer with a connection simply to the user's robot, there is a lesser need for a security system to protect data. However, we will ensure that the database that hosts the user's data is both secure and robust in that their data will not become corrupted in any way while they are using the application.

Portability

The current intention for the tool is to make it deployable on all computers hosting a Windows operating system.

Development:

For ease of development, all scripts, objects, and functions will be limited to 200 lines. This ensures the modularization of code which helps greatly in debugging, understanding each other's code, and updates. Ideally, functions won't exceed 100 lines. For the readability of code, which is essential for debugging and working with other people, the code standard:

<https://www.cs.purdue.edu/homes/cs240/code.html> will be followed as applicable to ensure comments are formatted the same way in a coherent manner so they can always be understood.

Diagram ideas - 7 diagrams: 1 in Design Outline, 6 in Design Details

(a) Design Outline

1. High-level class diagram

(b) Design details

1. Class Diagram (mandatory, as detailed as possible)
2. Other optional UML diagrams:
 - (i) Sequence diagram (important)
 - (ii) Deployment diagram
 - (iii) Activity/State diagram
 - (iv) Entity-Relationship diagram
 - (v) Interaction diagram

Design Outline

High-Level Overview

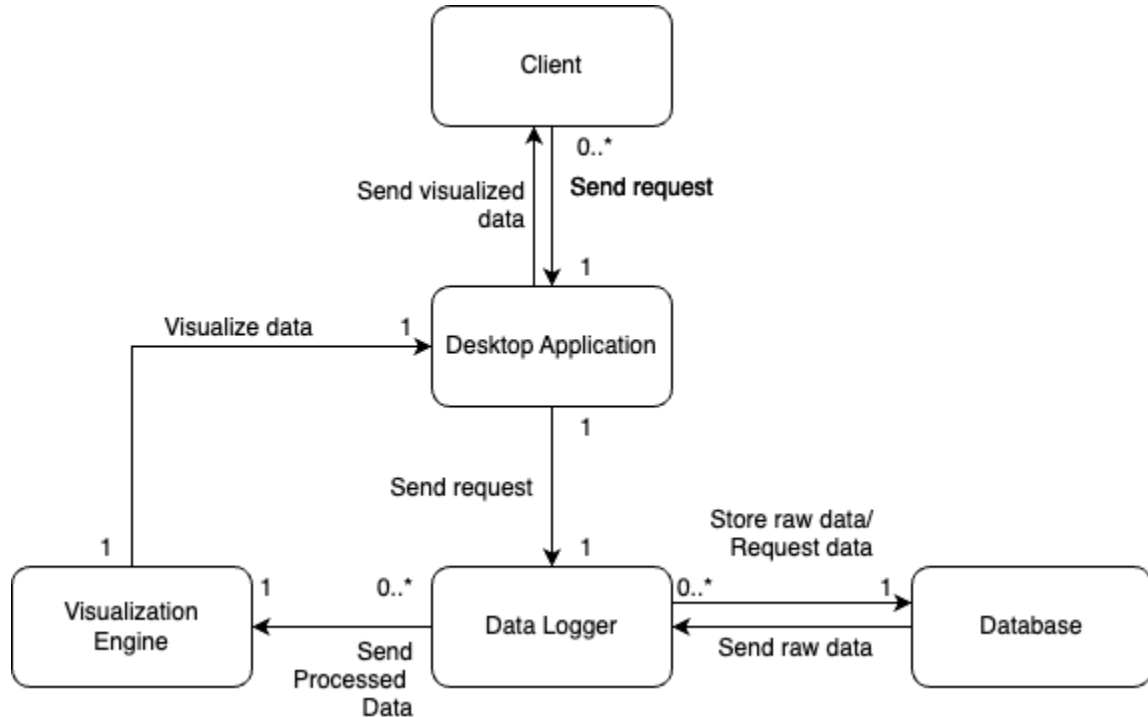
The system consists of three key components: the Client, the Desktop Application, and the Backend Engine. These components work together to provide a seamless data-driven environment, where the Client interacts with the Desktop Application to retrieve information, and the Desktop Application interfaces with the Backend Engine to process, store, and manage data.



Components

1. Client:
 - The client is the end-user interface, where FRC Robotics teams, mentors, and coaches interact with the system.
 - It sends requests to the Desktop Application for visualized data or specific logs.
 - Receives processed and visualized data from the Desktop Application in a readable format, such as graphs or reports.
2. Desktop Application:
 - The central controller that processes requests from the Client.
 - Interfaces with the Backend Engine to store and retrieve data.
 - Sends visualized data back to the Client for analysis or troubleshooting.
 - Acts as the mediator between the Client and Backend Engine, ensuring smooth data flow and processing.
3. Backend Engine:
 - Stores all the raw data collected during robot operations.
 - Provides persistent storage for system logs, robot performance metrics, and other critical data to the Desktop Application.
 - Supports queries from the Desktop Application to retrieve historical or real-time data for processing and visualization.

Detailed High-Level Overview



In the detailed view, we delve into the interactions between the various components and how they operate in conjunction to provide a seamless user experience. Also, we take a deeper look at the Backend Engine. Each component in the system works both independently and cooperatively, enabling efficient data logging, processing, and visualization.

Components

1. Client Interactions:
 - Multiple clients can interact with the Desktop Application. These clients send requests for visualized data and receive updated information based on real-time or historical data.
 - A client can request data in different formats, such as graphs showing motor activity or logs of sensor performance during matches.
2. Desktop Application:
 - The Desktop Application sits at the core of the system, handling requests from clients and coordinating data retrieval from the Data Logger and Database.
 - It can forward requests for data visualization to the Visualization Engine and manage communication between components.
3. Data Logger:

- This module is responsible for collecting raw data from the robot's control system during matches. It stores this raw data in the Database and sends processed data to the Visualization Engine.
 - The Data Logger ensures that all data from motors, sensors, and the control system is properly recorded for later analysis.
4. Visualization Engine:
- When the Desktop Application requests visualized data, the Visualization Engine processes the raw or processed data from the Data Logger.
 - The visualized data is then sent back to the Desktop Application, which forwards it to the client. This could include motor activity charts, sensor feedback graphs, or path visualizations.
5. Database:
- The Database serves as the long-term storage for all raw data collected during robot operations.
 - It allows the Data Logger to store data and permits the Desktop Application to retrieve stored data for future analysis and processing.

Design Issues

1. Determine data format when imported from robot (Functional)

Potential Solution: Decode the data from imported files that are received from the robot (post-match)

Justification for the Choice: According to the online documentation, we can download the files containing the robot data from the system, but that might not be visible through the app.

2. Transmission of data from robot to application (Functional)

Potential Solution:

1. Real-time data transmission as soon as the robot is connected to the computer.
2. Save the log data in the robot and extract the data to the computer later.

Justification for the Choice: Having instant data transmission or at least a very quick transmission of data post-match will make it convenient for the user to fix problems asap

3. Account security due to multiple users accessing the same account (Functional & Non-Functional)

Potential Solution:

1. Authentication
2. Encrypt log data

Justification for the Choice: As there will potentially be multiple users utilizing the same machine, it will be effective to protect data so other teams cannot access it

4. Storing and visualizing past sessions in the application (Functional)

Potential Solution:

1. Store it in the database in a standard format

Justification for the Choice: Storing the data in a standard format will make it easier to parse and visualize. Also, the number of past sessions should be limited to a certain number to make the data manageable.

5. Retrieve source code from robot and display in application (Functional)

Potential Solution:

1. Reverse engineer binary into source code
2. Restrict the languages to the ones that has the tool to reverse engineer binary into source code

Justification for the Choice: There might be multiple languages that may be used to run the robot

6. Data Compression and Archival (Non-Functional)

Potential Solution:

1. Implement data compression techniques (e.g., Gzip, Brotli) to reduce storage space for long-term logs. Compressed files can be stored on regular storage and decompressed when needed.
2. Use a tiered storage system that moves older, less frequently accessed data to lower-cost, slower storage mediums such as cloud-based cold storage (e.g., AWS Glacier, Google Cloud Archive).

Justification for the Choice: Tiered storage system is more scalable and cost-effective, efficiently managing large volumes of archived data by using cheaper storage for older, less accessed data.

7. Error Logging and Monitoring (Non-Functional)

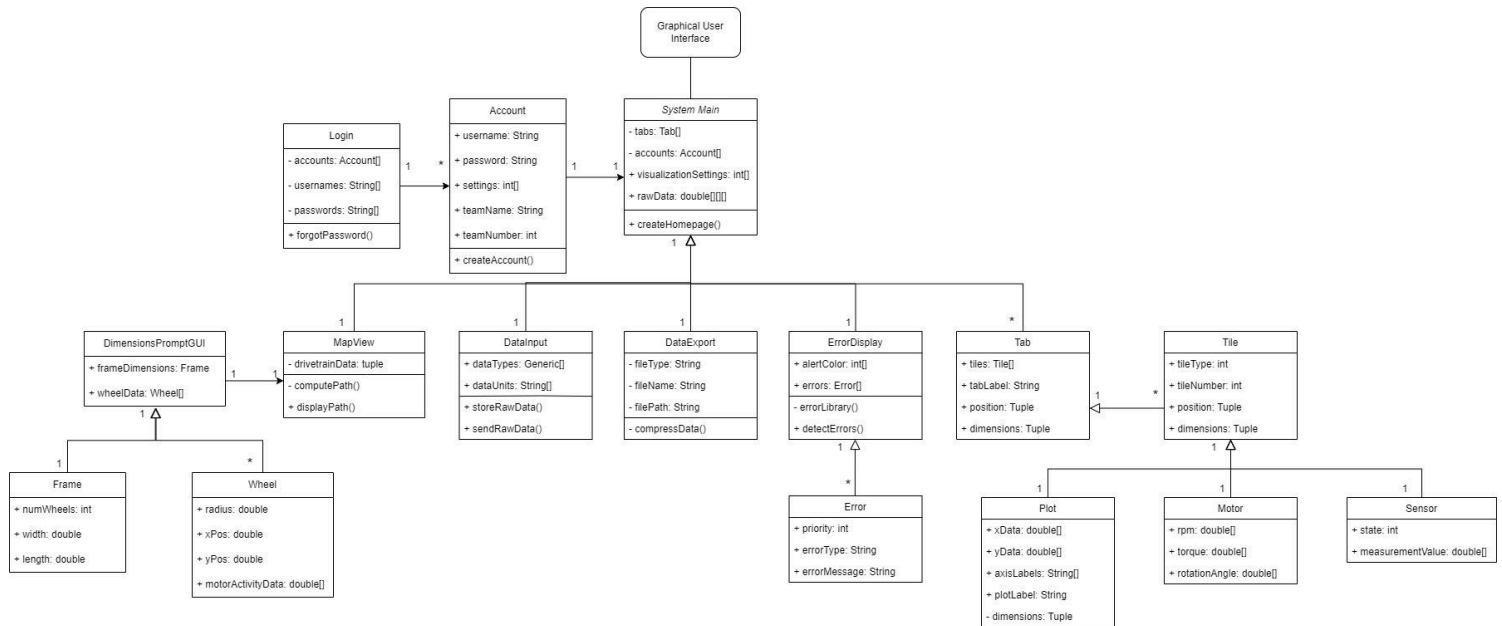
Potential Solution:

1. Implement centralized error logging and monitoring tools (e.g., ELK stack, Prometheus) to track issues across the system and provide real-time system health insights.
2. Integrate automatic alerting systems using services like PagerDuty or Slack to notify the development team of critical system failures in real-time.

Justification for the Choice: Centralized logging and monitoring tools provide comprehensive error tracking and system health insights, enabling faster issue identification and resolution.

Design Details

Class Level Design Diagram



Descriptions of Classes and Interaction between Classes

These classes are designed based on the objects necessary to run our system. Each class has a list of attributes which are the characteristics that drive its functionality.

1. Login

- The first thing to prompt the user when opening the app
- The user will fill out their username (which is the same as their email) and password
- This will access their corresponding account object
- If they do not have an account they can select to create an account which will prompt the `createAccount()` function in the `Account` class
- If they have an account but forgot their password, they can select to ‘Forgot Password’ and a random 6-digit code will be sent to their email and the app will prompt the user for that code to reset their password
- After login, the GUI will switch to the homepage

2. Account

- a. An Account object will store all the relevant user-specific information
- b. This will include their username and password and the corresponding FRC team name and number
- c. It will also store their visualization settings for the homepage
- d. The account information will also be used by a System Main class to access stored raw data on an external server

3. System Main

- a. This is the main runner class of the application (note the space in the title distinguishes it from the other classes since it is not necessarily an object like the others)
- b. Windows tabs objects are stored and organized here
- c. All account objects created are accessed from the server
- d. The current logged-in Account sets/stores the visualization settings as well as any stored raw data
- e. All stored raw data will be sent to the necessary classes as needed from here (arrows avoided for this so as not to clutter the diagram, as most classes use this)
- f. System Main will serve as the primary interface between the backend implementation of the app and the GUI that the user interacts with
- g. System Main will also implement the app homepage

4. DataInput

- a. The DataImport class handles raw data sent from either a continuous connection to the robot or many uploaded match data handled through the GUI
- b. It stores the data type classes of the corresponding data
- c. It also stores the data labels/units as strings
- d. Inputted data is stored both locally and automatically uploaded to an external server (so it can be accessed on other devices)

5. DataExport

- a. Takes in the filetype, file name, and desired file path, as prompted from a File Explorer GUI
- b. Uses this to save the desired data directly onto the user's computer, supporting the manual data storage ability

6. ErrorDisplay

- a. Detects any common errors or anomalies with the robot hardware based on the inputted data
- b. Displays the error(s) to the home screen with a corresponding priority level (shown with the error color)
- c. Stores/accesses a library of documented errors with corresponding error messages describing the possible problem

7. Error

- a. The Error class is used to form the library of robot hardware errors
- b. It stores an error priority, type, and an error message to be used by the ErrorDisplay

8. Tab

- a. Implements/stores the data elements displayed in a given Windows Tab in the GUI
- b. Contains a list of Tile objects to display the various plots and other data
- c. Labels for the Tab window itself will be allowed

9. Tile

- a. A Tile is a component of a given Windows Tab that displays a certain piece of data (i.e. plot, motor, etc.)
- b. There are 3 types a Tile can be depending on the data display being implemented: plot, motor, and sensor data
- c. The Tile's number is stored/accessed from here and denotes which space/order in the Windows Tab the Tile object is in

10. Plot

- a. Implements the raw x and y axis data into a graphical plot in a Tile
- b. Stores the axis labels/units and a label for the plot itself
- c. Has its own dimensions that act as the scope for the plot, scaled to highlight the data

11. Motor

- a. A motor object implements a motor's current data for the GUI to display what the motor is doing (in a Tile)

- b. Stores/displays the motor's RPM (Rotations Per Minute), resultant torque, and the current rotation angle

12. Sensor

- a. A sensor object implements a sensor's current data for the GUI to display what the sensor is doing or detecting (in a Tile)
- b. Stores/displays the sensor's current measurement values(s) over time with measurements depending on the type of sensor

13. MapView

- a. Implements a map of the robot's position over time based on the drivetrain dimensions and motor states which is then displayed on the homepage
- b. Based solely on the drivetrain dimensions and the rotation data from each of the wheels an internal function computes the course of the robot from basic geometry

14. DimensionsPromptGUI

- a. Immediately before the homepage is first shown after logging in, the dimensions of the robot are prompted by the user
- b. This includes:
 - i. The frame length and width
 - ii. The number of powered wheels (between 3 and 6 allowed)
 - iii. The radius of each powered wheel
 - iv. The x and y position of each powered wheel (assumed level in z-axis) with a coordinate system based in the lower-left corner of the robot frame

15. Frame

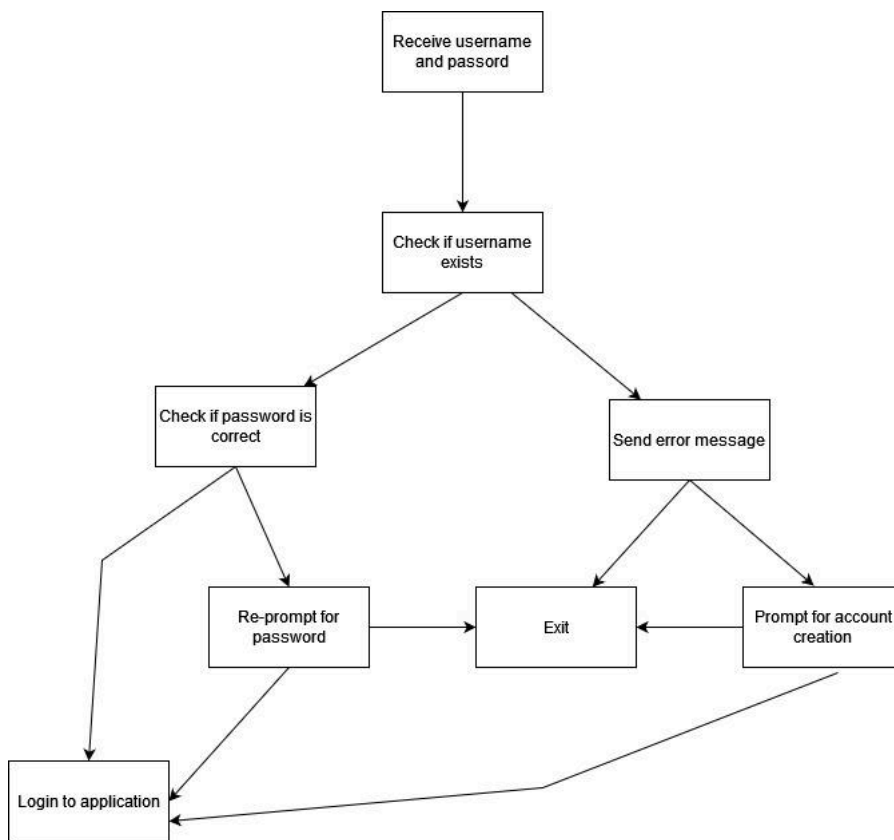
- a. Simply stores the number of powered wheels and the frame dimensions of the robot
- b. Meant for easy handling/organization of the frame data for more complicated processes like mapping the position

16. Wheel

- a. Simply stores the radius of a powered wheel and its corresponding x and y position relative to the robot frame

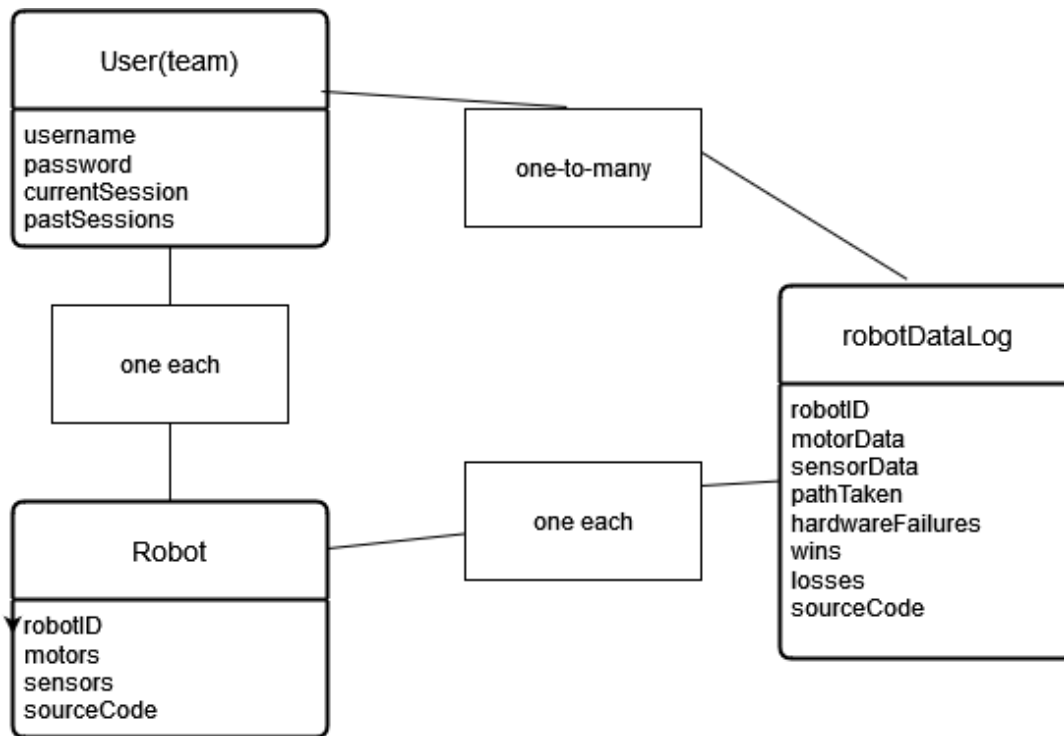
- b. Also stores the corresponding motor activity data for the given powered wheel for easy association
- c. Again, this meant for easy handling/organization of each powered wheel's data for the ease of more complicated processes like mapping the position

State Diagram for Logging in and Account Verification

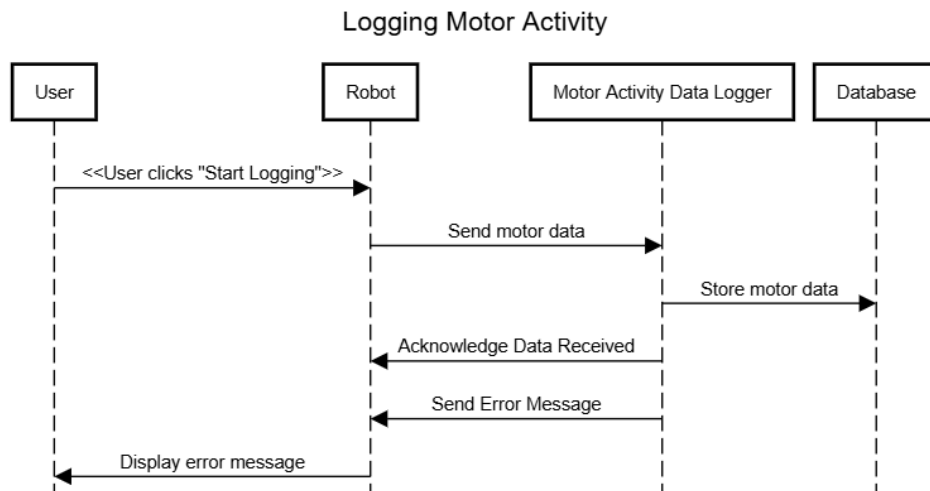


This state diagram illustrates the logical flow of the login and authentication process when a user attempts to log into the application. Initially, the program receives a username and password and verifies if the user exists in the database. Then, if the account does not exist, the user attempting login gets prompted to either create an account or exit the application. If they create an account, they can enter the application. On the other hand, if the username does exist, the user can now enter their password. If the password is incorrect, they can re-enter and try again (may implement a feature where they have limited re-tries). Here, they can either enter the correct password or decide to exit the application.

Entity-Relationship Diagram for Users, Robots and Data

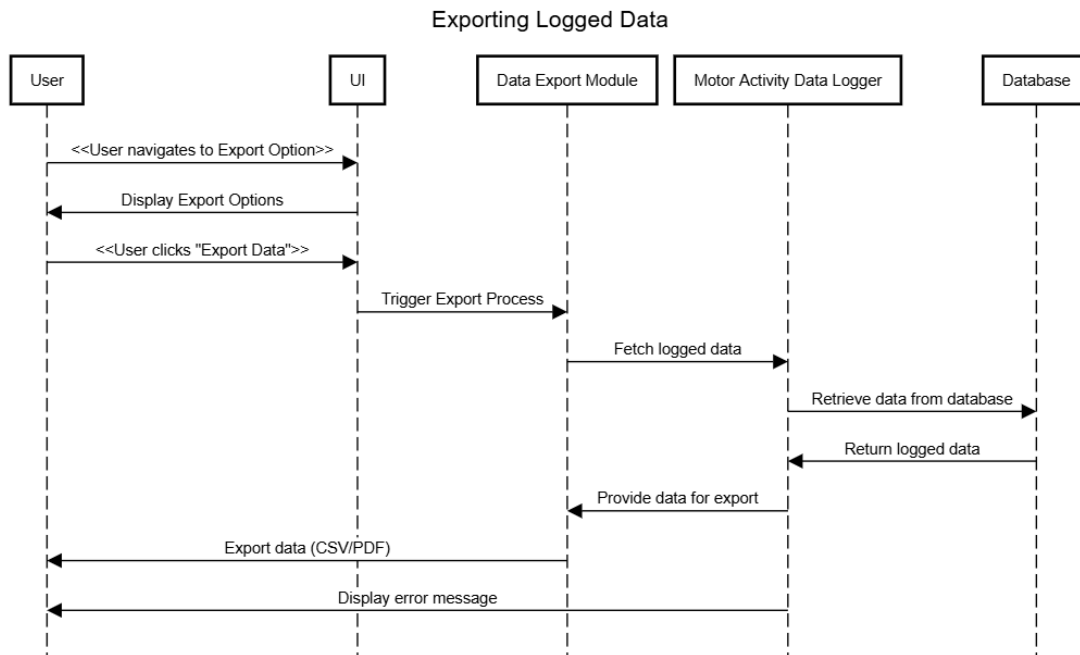


Sequence Diagrams



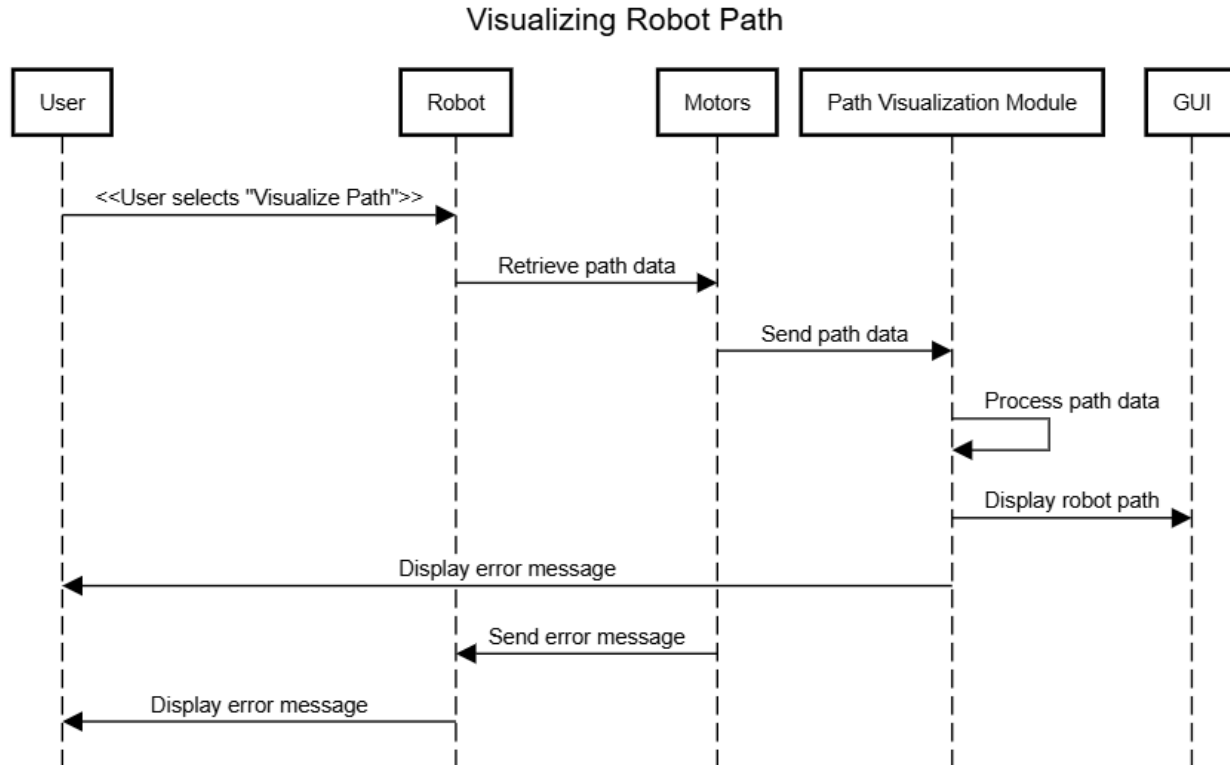
1. Logging Motor Activity

This diagram demonstrates the actions that happen when logging the robot's motor activity. It shows that the user (through the application) starts logging motor activity when prompted or when the match begins. The robot sends data to the data logger, and the data logger acknowledges to the robot that data was received. Then, the data logger saves the logged data to a database, where it can be retrieved from the user later.



2. Exporting Logged Data


This diagram demonstrates the steps for logged data to be exported. When a user requests a data export, the data exporting module sends the exported data over (in form of csv or pdf). The export module retrieves the data from the motor data activity logger, which retrieves its data from the database.




3. Visualizing Robot Path

This diagram illustrates how the system creates the robot path digital visualization. The user requests the visualization from the robot, which retrieves motor data from the motors. The data from the motors goes to the path visualization module, which then sends the generated robot path to the GUI.

UI Mockups



Username

Password  Hide

[Forgot password?](#)

Log in

Do not have an account?

! Create Account !

Create an Account

Already have an account? [Log in](#)

Team Name


Team Number

Username

Password

Confirm password

Log in





Real-Time

Export Data

Import Data

Raw Data

Data History

Log Out



Name		Date Modified	Size	Kind
> 2kuhf2bjf23nfkdsf23		Today at 8:34 PM	--	Folder
> ewqreqejn24g42k23		Today at 8:34 PM	--	Folder
> fkjnwef23uh2f3jfwef2	Rename	Today at 8:34 PM	--	Folder
> ij32fjwnwjf23fkbhwmwef	Import Data	Today at 8:35 PM	--	Folder
> jknwfkW923324bjf2f	Export Data	Today at 8:34 PM	--	Folder
> sdkjf23hubfe8fkb2f3		Today at 8:34 PM	--	Folder
> skdjf23fkfewjf23ff3r3	Edit Data	Today at 8:34 PM	--	Folder
> wif2h33kfmjdsf23ifhf3	Delete	Today at 8:35 PM	--	Folder
right click				

Data History

Return