**Internet of Things Fundamentals**

*Subject Project*

BS AI 6th Smester SP-25 (AIE-3079)

Date: June 26, 2025

**Project Title:**

Remote Control Rover

**Group Name/no.:**

AICodeX

**Team Members:**

|  |  |  |  |
| --- | --- | --- | --- |
| Members | Registration no | Name | Signature |
| **Member-1 (Leader)** | **22-NTU-CS-1363** | **M. Jahanzaib Arshad** |  |
| **Member-2** | **22-NTU-CS-1366** | **M. Talha Javed** |  |
| **Member-3** | **22-NTU-CS-1371** | **Safder Rehman** |  |

**Contributions in % of each Team Members for each component**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | Member-1 | Member-2 | Member-3 |
| Distribution Components | | Name | Name | Name |
| Coding | Raspberry Pi | 20 | 60 | 20 |
| Python Coding | 20 | 60 | 20 |
| UI Design | | 10 | 10 | 80 |
| Database | | 15 | 15 | 70 |
| Cloud Integration  (Firebase authentication) | | 10 | 10 | 80 |
| IoT Gateway | | - | - | - |
| Edge Processing | | 10 | 80 | 10 |
| Documentation | | 60 | 10 | 30 |
| Presentation  Design | | 60 | 15 | 25 |
| Unity Simulation | | 50 | 25 | 25 |
| Unity Coding | | 80 | 10 | 10 |
| RL Model | | 85 | 5 | 10 |
| Model Integration | | 40 | 40 | 20 |

*To be filled by the evaluator*

Team-Based Evaluation (60 Marks)

|  |  |  |
| --- | --- | --- |
| Criteria | Obtained Marks | Out of |
| System Design & Architecture |  | 10 |
| Hardware Integration & Circuit Setup |  | 10 |
| IoT Gateway and Cloud Communication |  | 10 |
| Working Prototype Demonstration |  | 10 |
| Performance & Reliability Testing |  | 10 |
| Presentation |  | 10 |
| Total (Team-Based) |  | 60 |

Individual-Based Evaluation (40 Marks per Member)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Member 1 | Member 2 | Member 3 | Member 4 |
| Criteria |  |  |  |  |
| Understanding of the Project & Role | /10 | /10 | /10 | /10 |
| Code Contribution and Explanation | /10 | /10 | /10 | /10 |
| Q/A VIVA | /10 | /10 | /10 | /10 |
| Documentation/Reporting & Communication | /10 | /10 | /10 | /10 |
| Total (Individual-Based) | /40 | /40 | /40 | /40 |
| Total Overall (60+40) | /100 | /100 | /100 | /100 |
| Weightage Lab Grade (50) |  |  |  |  |

**TABLE OF CONTENTS**

**Description Page**

Title………………..……………………………………………………………………….i

Table of Contents ………...……………………………………………………………… iv

List of Notations…………………………………………………………………………. vi

List of Abbreviations……………………………………………………………………...vii

List of Tables……………………………………………………………………………...viii

List of Figures……………….………………………………………………………….....ix

Abstract………….…………...………………………………………………………… …x

[Chapter 1 Introduction 1](#_Toc201871063)

[1.1 Background 1](#_Toc201871064)

[1.2 Problem Statement 1](#_Toc201871065)

[1.3 Objectives 1](#_Toc201871066)

[1.4 Scope 1](#_Toc201871067)

[Chapter 2 Literature Review 2](#_Toc201871068)

[2.1 Traditional Approaches to Line Following 2](#_Toc201871069)

[2.2 Equipments 2](#_Toc201871070)

[2.3 Deep Learning Approaches 3](#_Toc201871071)

[2.4 Reinforcement Learning (RL) Approaches 3](#_Toc201871072)

[2.5 Unity and ML-Agents Toolkit 3](#_Toc201871073)

[2.6 Web Interface Development using Next.js and Firebase 3](#_Toc201871074)

[2.7 Hardware Integration and Embedded Control 3](#_Toc201871075)

[Chapter 3 5](#_Toc201871076)

[System Design and Architecture 5](#_Toc201871077)

[3.1 Hardware Architecture 5](#_Toc201871078)

[3.2 Software Architecture 5](#_Toc201871079)

[3.2.1 Unity Environment 5](#_Toc201871080)

[3.2.2 RL Training 6](#_Toc201871081)

[3.2.3 Model Deployment 6](#_Toc201871082)

[3.2.4 Next.js Web Interface with Firebase 6](#_Toc201871083)

[3.3 Website Architecture and Flow 7](#_Toc201871084)

[3.3.1 Flow Diagram of Website 7](#_Toc201871085)

[3.3.2 Sequence Diagram (User-System Interaction) 8](#_Toc201871086)

[3.3.3 Class Diagram of Firebase Database 9](#_Toc201871087)

[Chapter 4 Implementation 10](#_Toc201871088)

[4.1 Unity Environment 10](#_Toc201871089)

[4.2 RL Agent Training 10](#_Toc201871090)

[4.3 Raspberry Pi Setup 10](#_Toc201871091)

[4.4 Next.js and Firebase Website 10](#_Toc201871092)

[4.4.1 Frontend (Next.js) 10](#_Toc201871093)

[4.4.2 Backend (Firebase) 10](#_Toc201871094)

[Chapter 5 12](#_Toc201871095)

[User Interface 12](#_Toc201871096)

[5.1 Overview 12](#_Toc201871097)

[5.2 Key Features 12](#_Toc201871098)

[5.3 Technology Stack 12](#_Toc201871099)

[5.4 Pages and Components 12](#_Toc201871100)

[5.5 UI Screenshots To support visual understanding 13](#_Toc201871101)

[5.6 Usability and Design Decisions 16](#_Toc201871102)

[Chapter 6 17](#_Toc201871103)

[Results and Evaluation 17](#_Toc201871104)

[6.1 Simulation Results 17](#_Toc201871105)

[6.2 Real-World Performance 17](#_Toc201871106)

[6.3 Web Interface Testing 17](#_Toc201871107)

[Chapter 7 Conclusion and Future Work 18](#_Toc201871108)

[7.1 Conclusion 18](#_Toc201871109)

[7.2 Future Work 18](#_Toc201871110)

[References 19](#_Toc201871111)

[Links 20](#_Toc201871112)

**LIST OF NOTATIONS**

|  |  |  |  |
| --- | --- | --- | --- |
| ML-Agents | | Machine Learning Agents (Unity Toolkit) | |
| Next.js | | A React-based Web Development Framework | |
| Realtime DB | | Firebase Realtime Database | |
| Grayscale Image | Image input format used by the RL model | |

|  |  |
| --- | --- |
| Discrete Actions | The action space used by the agent: forward, left, right, stop |

**LIST OF ABBREVIATIONS**

|  |  |
| --- | --- |
| AI | Artificial Intelligence |
| RL | Reinforcement Learning |
| PPO | Proximal Policy Optimization |
| CNN | Convolutional Neural Network |
| IR | Infrared |
| GPU | Graphics Processing Unit |
| API | Application Programming Interface |
| SDK | Software Development Kit |
| ML | Machine Learning |
| UI | User Interface |
| LTS | Long-Term Support |
| GPIO | General Purpose Input Output |
| DC | Direct Current |
| USB | Universal Serial Bus |
| ONNX | Open Neural Network Exchange |
| SLAM | Simultaneous Localization and Mapping |
| LIDAR | Light Detection and Ranging |
| Pi | Raspberry Pi (Single-board Computer) |
| HTML | HyperText Markup Language |
| CSS | Cascading Style Sheets |
| URL | Uniform Resource Locator |
| MIT | Massachusetts Institute of Technology |

**LIST OF TABLES**

Table 2.1: Comparison of Line Following Techniques……………………………………2

Table 3.1: Hardware Components…………………………………………………………5

Table 3.2: Unity Simulation Parameters…………………………………………………..5

Table 3.3: RL Agent Training Settings…………………………………………………....6

Table 4.1: Implementation………………………………………………………………10

**LIST OF FIGURES**

Figure 3.1**:** Website Flow Diagram 7

Figure 3.2: Website Sequence Diagram 8

Figure 3.3: Firebase Class Diagram 9

Figure 5.1: Website Home Page 13

Figure 5.2: Functionalities on Home Page 14

Figure 5.3: Website Products Page 14

Figure 5.4: User Control Page 15

Figure 5.5: Website Login Page 16

**ABSTRACT**

This project presents the design and implementation of an intelligent autonomous rover that follows a visual path using a reinforcement learning (RL) approach. The core objective was to develop a vision-based navigation system without relying on traditional line-following sensors, making the rover more adaptable to real-world environments. A custom simulation environment was built using Unity6 LTS, where a Proximal Policy Optimization (PPO) agent was trained using grayscale camera images as observations. The trained model was deployed on a Raspberry Pi, which controlled the rover's motors in response to real-time camera input. Additionally, a full-featured web interface was developed using Next.js and Firebase to handle user authentication, model uploads, and live camera feed visualization. All components—Unity simulation, RL training, hardware integration, and web development—were implemented from scratch, demonstrating a practical application of AI, embedded systems, and full-stack development. The final system successfully tracked paths under varied conditions, validating the feasibility of vision-based autonomous navigation on low-cost hardware.

# Chapter 1 Introduction

## Background

In recent years, autonomous vehicles have gained immense popularity due to their potential applications in transportation, agriculture, surveillance, and logistics. This project focuses on designing and implementing a low-cost intelligent rover capable of following a predefined path using only visual input from a front-mounted camera. Reinforcement Learning (RL) is used to train an agent in a simulated environment before deploying the model on a real rover.

## Problem Statement

Traditional line-following robots rely on IR sensors or manual thresholding. These methods are inflexible and fail under variable lighting or occlusions. Our objective is to build a rover that uses camera input to learn optimal motor control strategies through RL, making it more adaptable and robust.

## Objectives

* Develop a Unity simulation of a rover with a camera-based vision system.
* Train an RL agent using visual input to follow a path.
* Deploy the trained model to a real-world rover controlled by Raspberry Pi.
* Build a web interface using Next.js and Firebase for user login, model upload, and live image viewing.

## Scope

This project is limited to visual path following using a camera without GPS, LIDAR, or advanced SLAM. Real-world deployment is limited to indoor/outdoor tape-marked paths.

# Literature Review

We conducted an extensive literature review and practical study of technologies and methods across software and hardware domains relevant to our autonomous rover. Since we began with no prior experience in Unity, Next.js, or electronics, each element was learned from scratch.

## Traditional Approaches to Line Following

Conventional line-following robots use infrared (IR) sensors and threshold-based logic to detect line contrast and make directional decisions. While effective in controlled environments, these methods lack robustness to dynamic changes in lighting and surface textures. PID controllers are typically used to fine-tune movements, but are manually tuned and limited in adaptability.

Table 2.1: Comparison of Line Following Techniques

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method** | **Sensors Used** | **Adaptability** | **Cost** | **Limitations** |
| Traditional  (IR-based) | IR sensors | Low | Low | Affected by lighting, surface changes |
| OpenCV-based Vision | Camera | Medium | Medium | Needs manual tuning |
| Deep Learning (CNN) | Camera | High | High | Require large dataset |
| Reinforcement Learning | Camera | Very High | Medium | Needs training time & simulation |

## Equipments

Edge detection and color thresholding using OpenCV have been widely employed in robot navigation. Algorithms like Canny edge detection, Hough transforms, and morphological operations can extract lines or edges from the camera input. However, these techniques require manual parameter tuning and fail when background noise, poor lighting, or path obstructions are present.

## Deep Learning Approaches

CNN-based approaches have been applied to train end-to-end models where the input is a camera image and the output is a steering angle or direction. Models like PilotNet by NVIDIA showed that visual input alone could predict car control commands. However, such models require large labeled datasets and extensive compute resources for training.

## Reinforcement Learning (RL) Approaches

Reinforcement learning allows an agent to learn from interactions with the environment by maximizing a reward function. In particular, visual-based RL (using image frames as observation) is becoming popular for tasks where environment dynamics are complex or not explicitly modeled. The Unity ML-Agents toolkit provides a convenient simulation environment for training such agents using algorithms like PPO (Proximal Policy Optimization).

## Unity and ML-Agents Toolkit

Unity ML-Agents provides an interface to integrate RL agents with 3D simulation environments. We learned Unity engine fundamentals from scratch, including scene design, lighting, 3D object modeling, physics-based interactions, and camera setups. Learning how to write custom C# scripts and integrate Python APIs was essential to train agents effectively.

## Web Interface Development using Next.js and Firebase

We also built a full-featured web interface for interacting with the rover system. This involved learning modern web development practices including React-based frameworks (Next.js), Firebase Authentication, Firebase Storage, and Firebase Realtime Database. Starting from zero knowledge, we explored routing in Next.js, frontend form validation, backend integration with Firebase SDKs, and asynchronous data fetching. Security practices such as protecting routes, verifying uploads, and managing user sessions were studied and applied.

## Hardware Integration and Embedded Control

On the hardware side, we self-taught Raspberry Pi GPIO programming, motor driver configurations, USB camera setup, and deployment of lightweight ML models using TensorFlow Lite. The biggest challenge was ensuring real-time motor response based on model predictions under hardware constraints.

# 

# System Design and Architecture

## Hardware Architecture

* **Raspberry Pi** (central control unit)
* **Front-facing Camera** (visual input)
* **Motor Driver (L298N)**
* **4 DC Motor**
* **Power Supply**

Table 3.1: Hardware Components

|  |  |
| --- | --- |
| **Component** | **Description** |
| Raspberry Pi | Controls the system and runs inference |
| Camera | USB camera (front-facing) |
| Motor Driver | L298N |
| Motors | 4 DC motors |
| Power Supply | Battery pack |

## Software Architecture

### Unity Environment

* Custom 3D track
* Rover model with mounted camera
* Reward design based on path alignment

Table 3.2: Unity Simulation Parameters

|  |  |
| --- | --- |
| **Parameter** | **Value / Description** |
| Simulation Tool | Unity6 LTS |
| Track Type | Closed-loop with curves |
| Environment Randomization | Lighting and surface variation |
| Camera Resolution | 84 × 84 (Grayscale) |
| Observation Space | Visual (image input) |
| Action Space | Discrete: Left, Right, Forward, Stop |

### RL Training

* PPO (Proximal Policy Optimization) algorithm
* Visual input as observation space
* Discrete action space (forward, left, right, stop)

Table 3.3: RL Agent Training Settings

|  |  |
| --- | --- |
| **Setting** | **Value** |
| Algorithm | PPO (Proximal Policy Optimization) |
| Input Frame | Grayscale 84×84 |
| Reward Scheme | +1 (centered), -1 (deviated) |
| Number of Training Steps | ~500,000 |
| Success Rate in Sim | 90% |

### Model Deployment

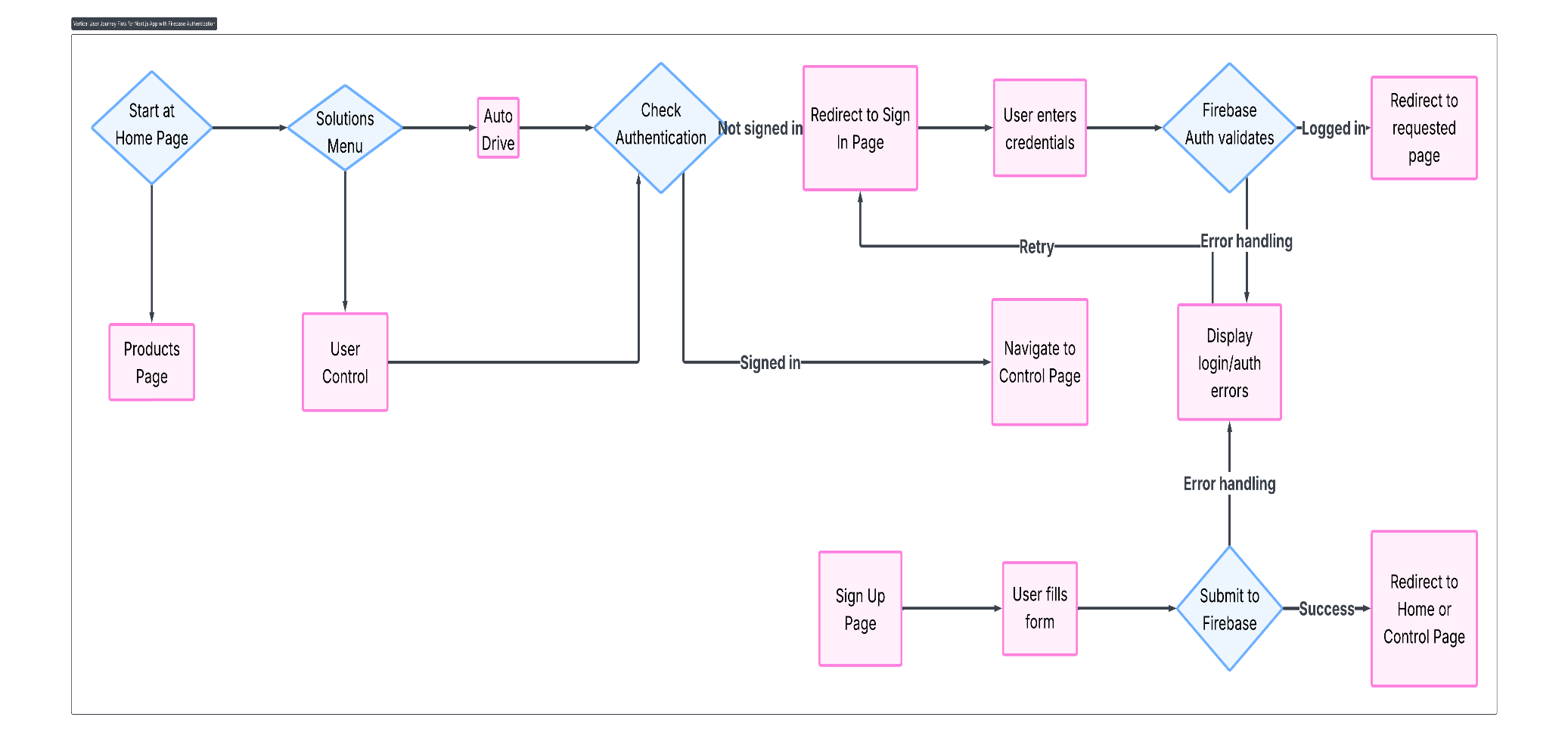
* Export ONNX or TensorFlow Lite model
* Raspberry Pi interprets camera frames and feeds to model
* Output actions are mapped to motor controls

### Next.js Web Interface with Firebase

* User sign-up/login
* Upload trained models
* Display live rover camera feed via Firebase Storage

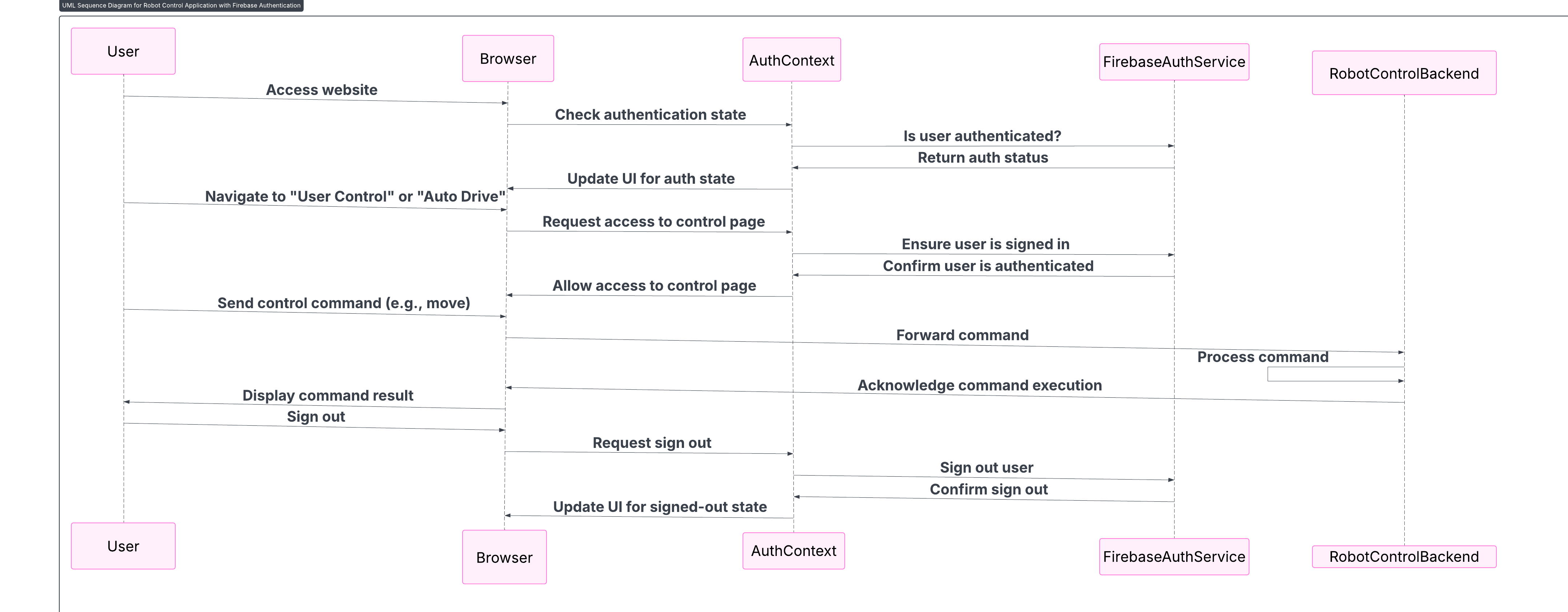
## Website Architecture and Flow

### Flow Diagram of Website



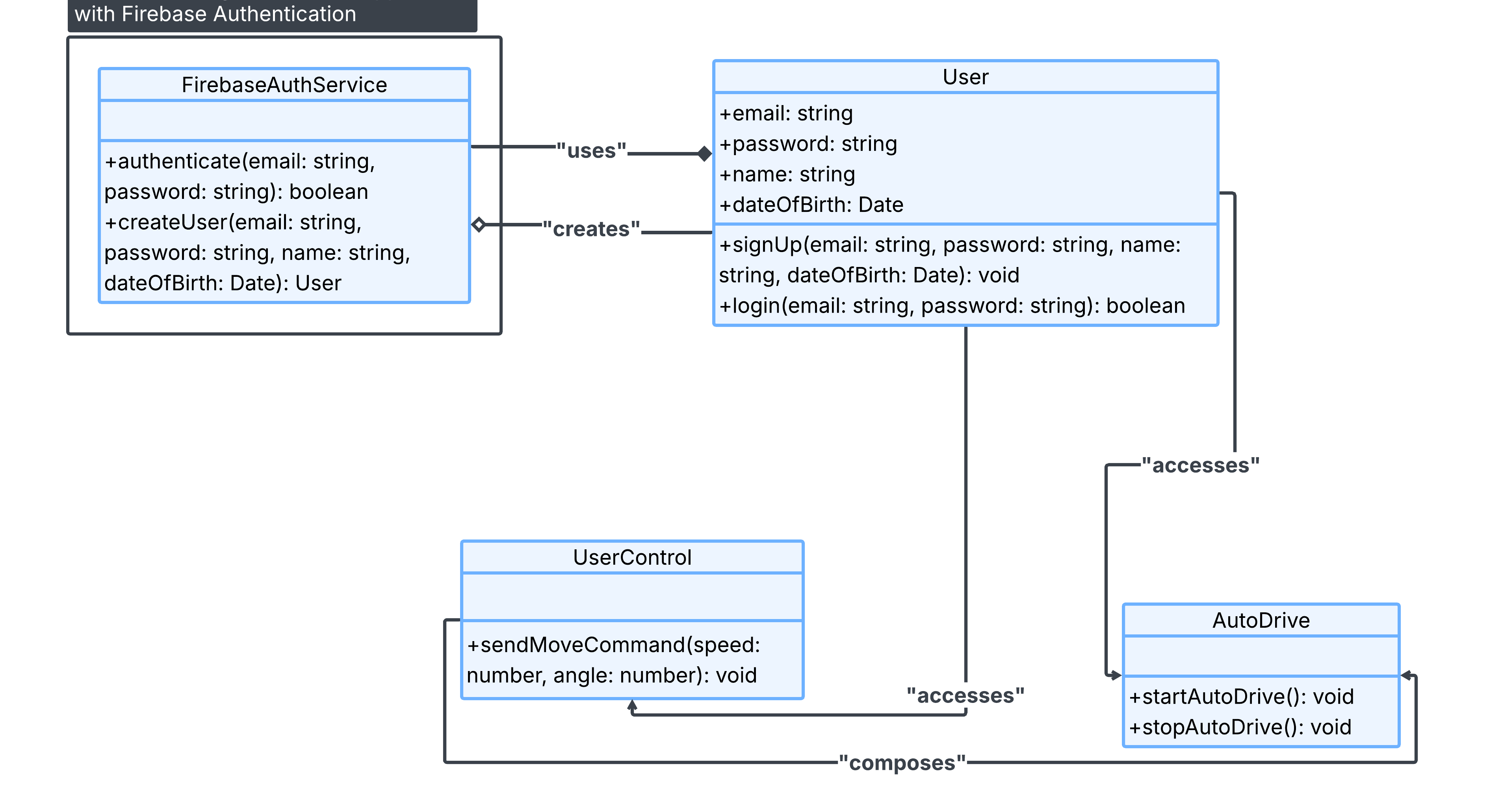
**Figure 3.1:** This is the flow diagram showing the transition of pages available in the website.

### Sequence Diagram (User-System Interaction)



**Figure 3.2:** This is the Sequence diagram showing the transition of actions performed by the website based on user inputs.

### Class Diagram of Firebase Database



**Figure 3.3:** This is Class diagram of Tables on Firebase and where there stored values are being used and called for.

# Implementation

## Unity Environment

* Used Unity6 LTS with ML-Agents 20+
* Designed a closed-loop track with variable lighting
* Implemented domain randomization for better generalization

## RL Agent Training

* Input: Grayscale 84x84 camera image
* Output: Discrete actions (left, right, forward)
* Reward: +1 for staying centered, -1 for deviation
* Training duration: ~500k steps

## Raspberry Pi Setup

* onnx runtime installed
* USB camera captures real-time frames
* Python script processes frame, runs model inference, controls motors

## Next.js and Firebase Website

### Frontend (Next.js)

* Sign-up/login pages using Firebase Authentication
* Live image view panel (images uploaded from Pi)

### Backend (Firebase)

* Realtime Database: Metadata and logs
* Firebase Storage: Model files and camera snapshots

Table 4.1: Implementation

|  |  |  |
| --- | --- | --- |
| **Feature** | **Technology Used** | **Purpose** |
| User Authentication | Firebase Auth | Secure login/signup |
| Model Upload | Raspberry Pi Storage | Upload trained onnx model |
| Image Streaming | Realtime | View rover's live images |
| Dashboard | Next.js (React) | UI for product management and streaming |

# 

# User Interface

## Overview

The user interface (UI) for the intelligent rover system was designed to be intuitive, secure, and responsive, enabling users to interact with the system without needing technical expertise. Built using Next.js, a modern React-based web framework, the interface integrates Firebase for authentication and model management, providing a seamless experience for researchers, developers, or field operators.

## Key Features

The following functionalities are integrated into the web interface:

* **User Authentication:** Secure sign-up and login pages implemented using Firebase Authentication with email and password credentials.
* **Model Upload:** After training the reinforcement learning model in Unity, users can upload their ONNX or TensorFlow Lite models through the interface.
* **Live Camera Feed:** The live video feed from the rover's front-facing camera is uploaded to Firebase Storage and rendered in near real-time on the dashboard.
* **Navigation and UI Layout:** A structured navigation menu (Navbar), a visually engaging landing page (Hero), solution explanations (SolutionsSlider), system overview (HowItWorks), and footer are included to make the site user-friendly and informative.

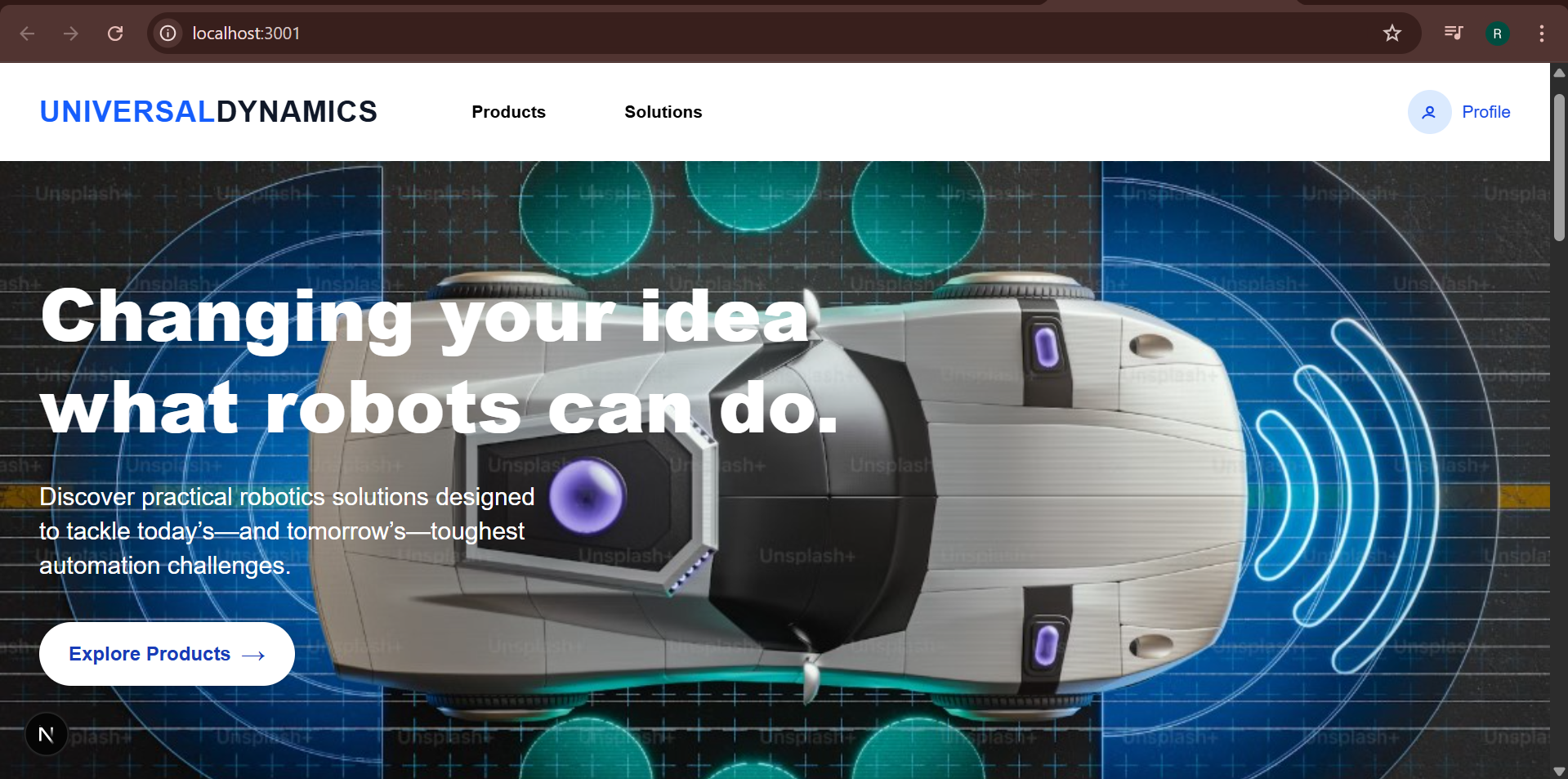
## Technology Stack

* **Frontend:** Next.js, Tailwind CSS, React Icons, Framer Motion
* **Backend Integration:** Firebase (Auth, Storage, Realtime DB)
* **Styling:** Tailwind CSS for responsive and utility-first styling

## Pages and Components

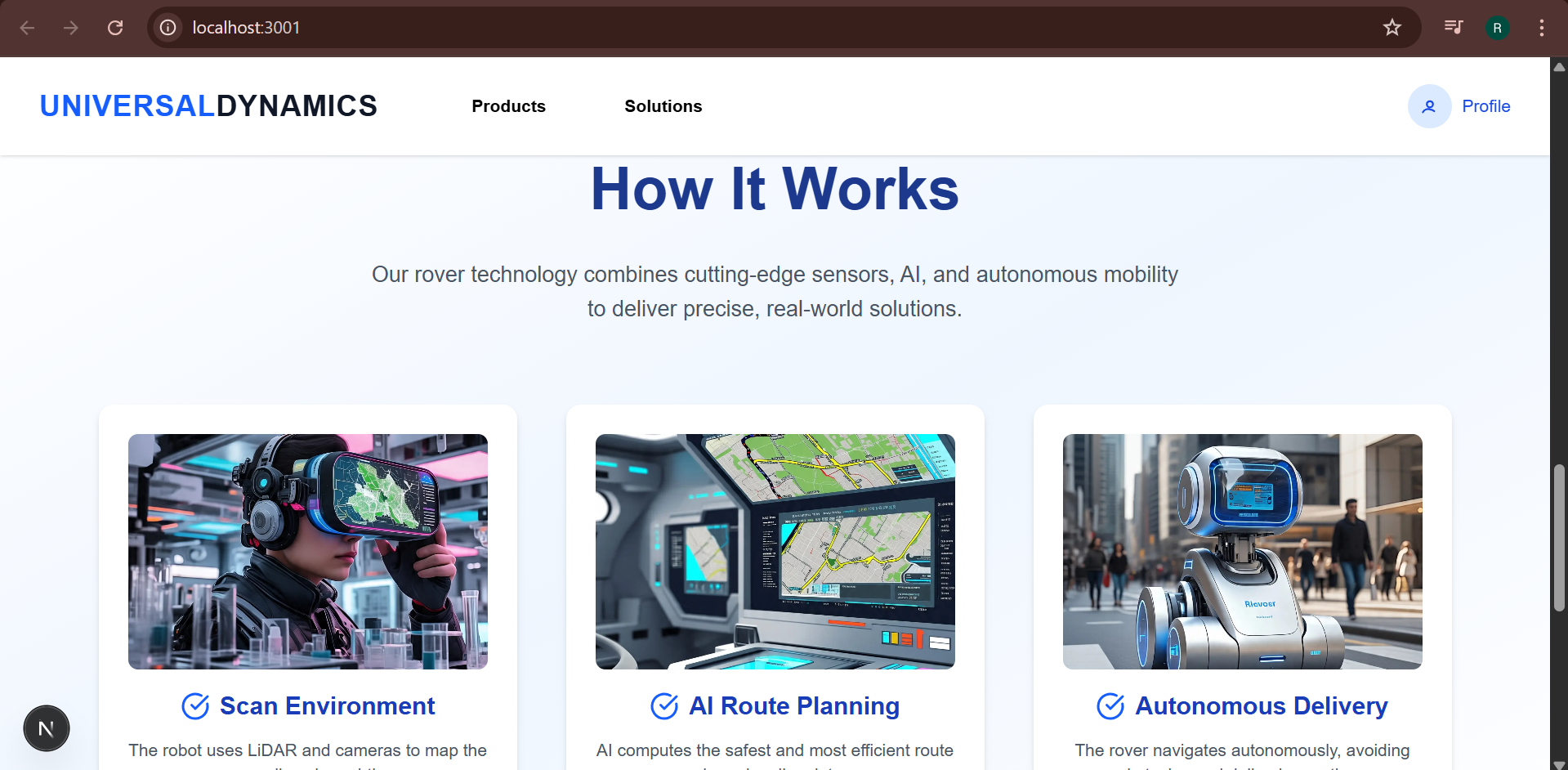
* **Home Page (/components/page.tsx)**
  + Displays the Navbar, Hero, SolutionsSlider, HowItWorks, and Footer.
  + Provides quick navigation to different parts of the system overview.
* **Sign Up Page (/sign-up/page.tsx)**
  + A two-step registration form (email/password -> personal details).
  + Validations, password confirmation, and visibility toggles.
* **Sign In Page (/sign-in/page.tsx)**
  + Email/password login functionality.
  + Displays error alerts if authentication fails.
* **Upload Page** (optional future component)
  + Enables users to upload trained model files to Firebase Storage.

## UI Screenshots To support visual understanding



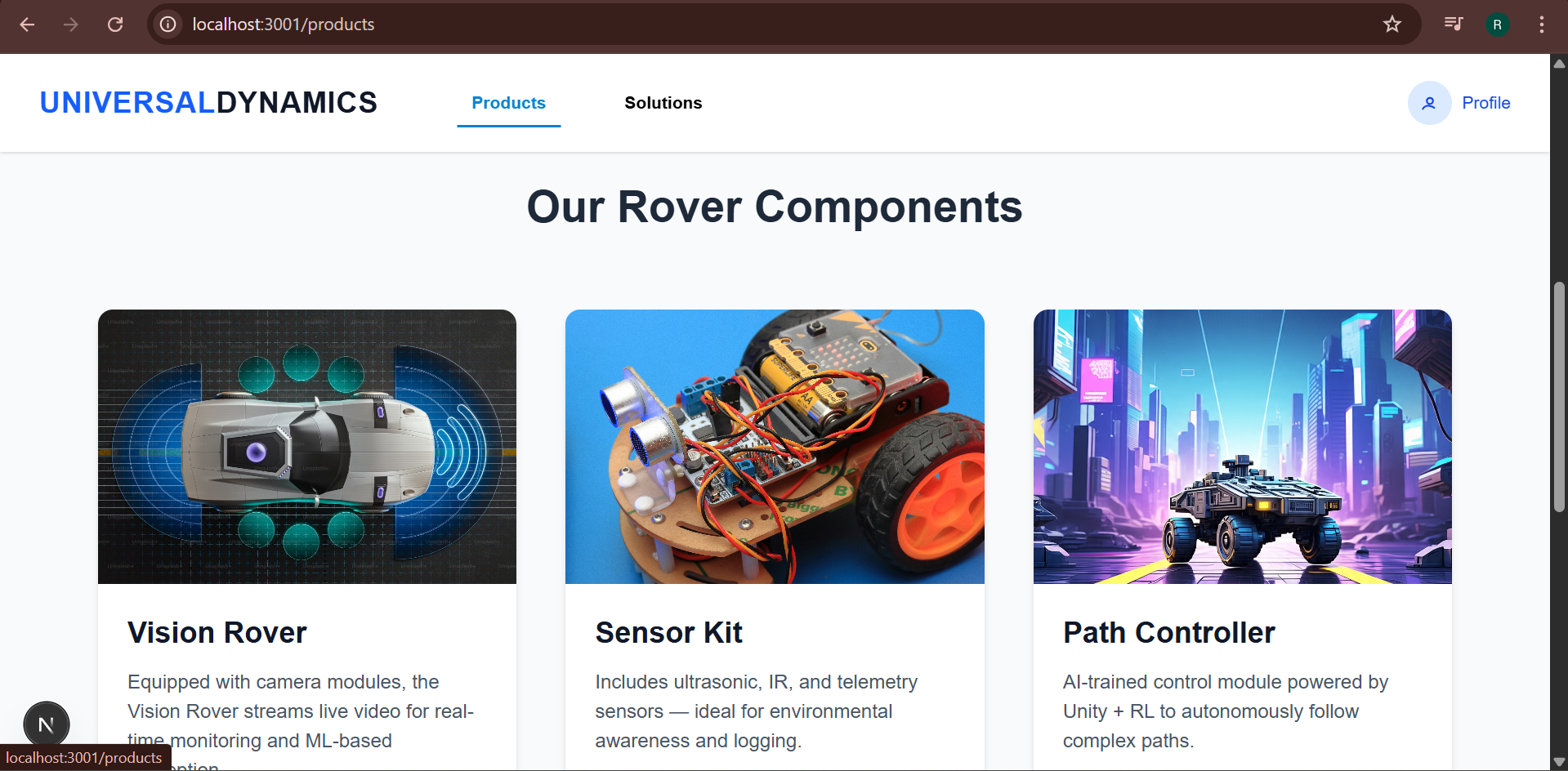
**Figure 5.1:** This is the home page on the website.

This is the main landing section of the website, designed to immediately capture user attention with bold typography and a futuristic image of an autonomous vehicle. The headline emphasizes the project's innovative impact: "Changing your idea what robots can do." A supporting tagline invites users to explore real-world automation solutions. The layout includes a prominent call-to-action button directing users to the product section. The navbar above provides intuitive navigation with links to “Products,” “Solutions,” and user profile access.



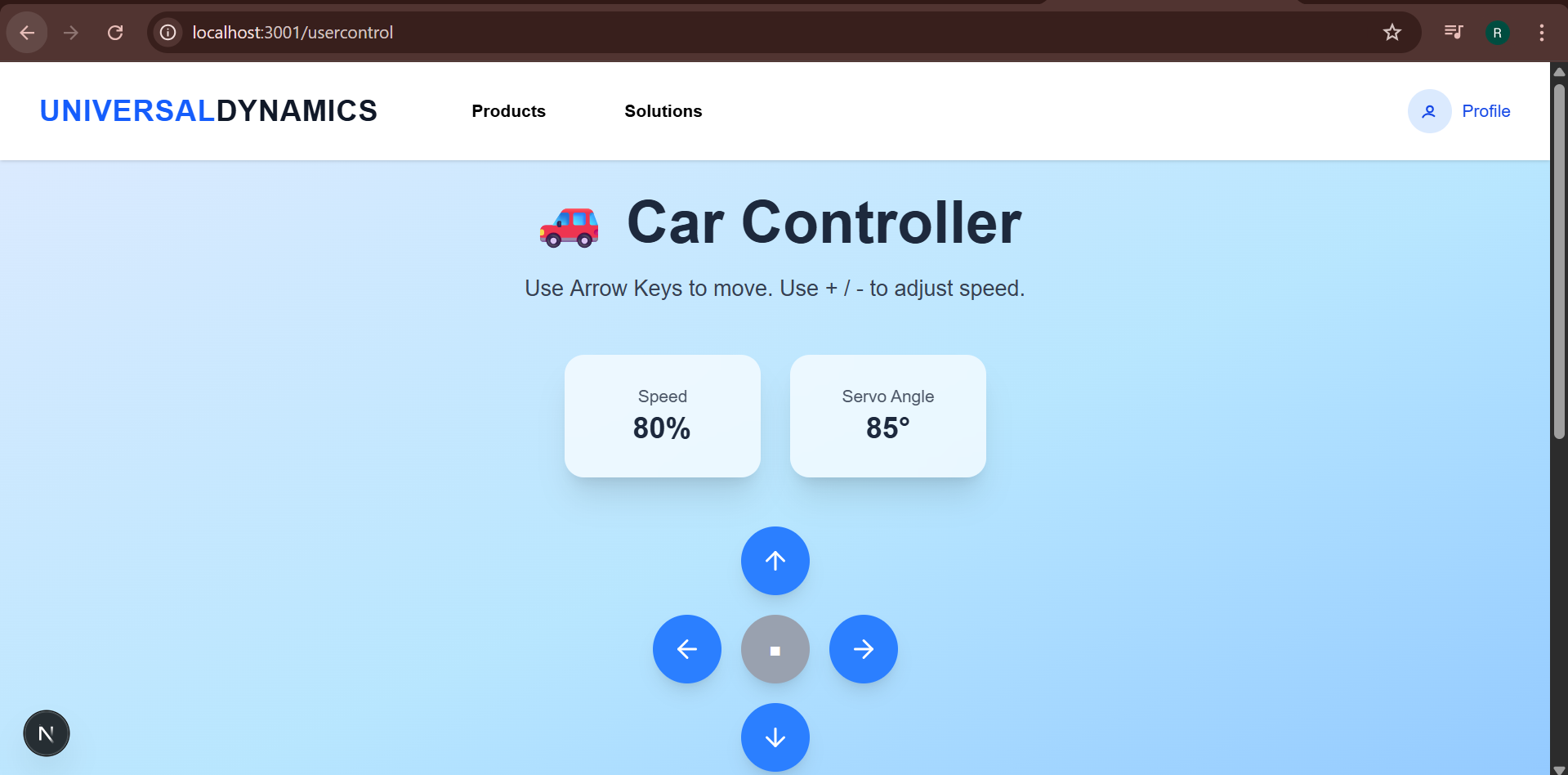
**Figure 5.2:** This is the functionalities offered by our product shown on the home page.

This section visually explains the rover's core functionality through three intuitive cards: **Scan Environment**, **AI Route Planning**, and **Autonomous Delivery**. Each card features a high-quality image and concise caption, highlighting how the rover maps its surroundings, computes optimal paths, and autonomously delivers. The layout uses a balanced white and blue theme to reflect modernity and trust. Positioned below the hero section, it educates users on the rover’s end-to-end intelligent workflow in a visually engaging format.



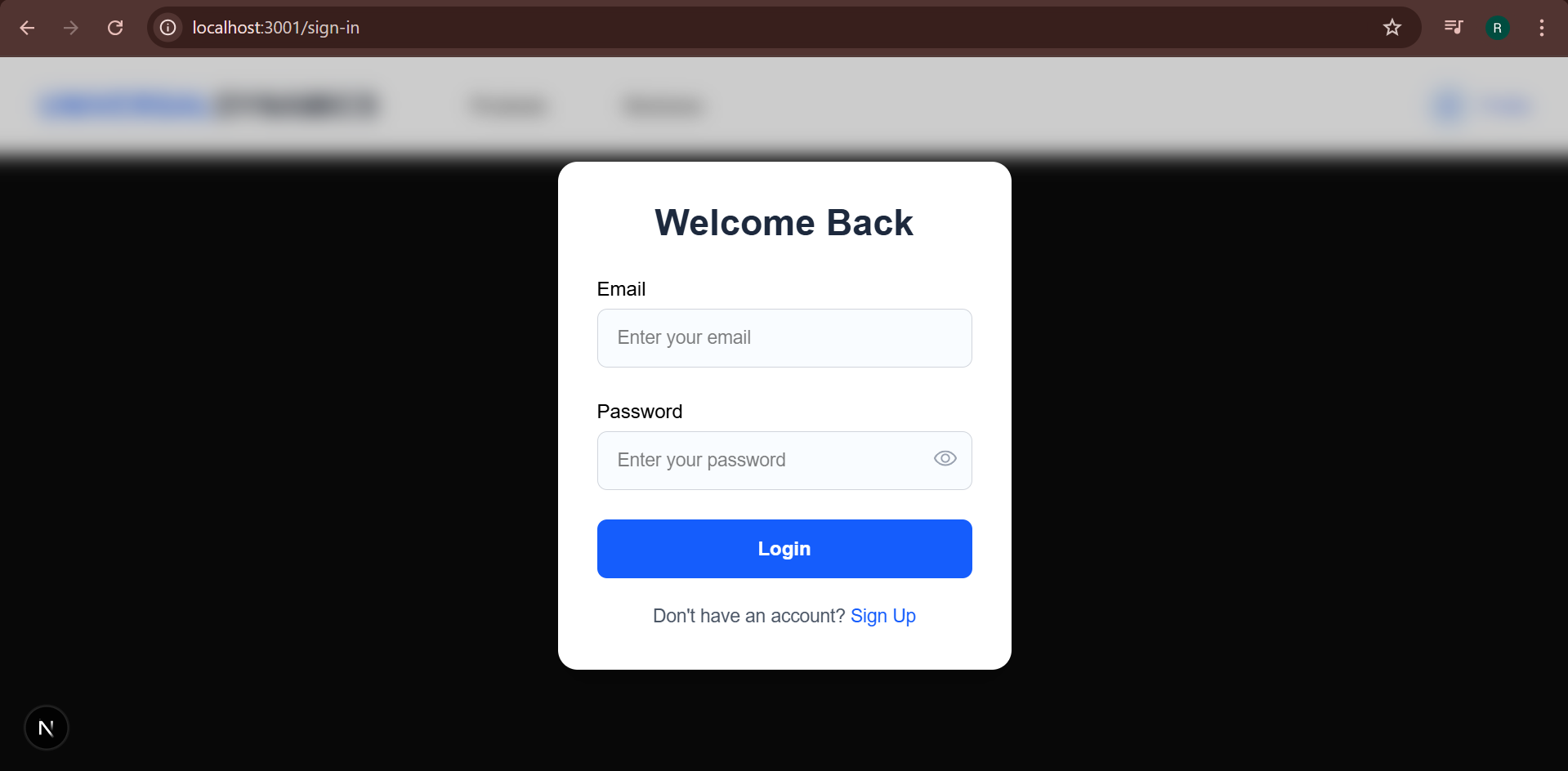
**Figure 5.3:** Products page on website, explaining the different components used in the product.

This section presents the three main hardware modules of the rover system in a clean card layout: **Vision Rover**, **Sensor Kit**, and **Path Controller**. Each card combines high-quality imagery with concise descriptions, explaining the technical function and relevance of each module. The Vision Rover focuses on live camera feeds, the Sensor Kit handles environmental detection, and the Path Controller uses RL for navigation. The design ensures clarity for users to understand each product at a glance. Highlighted headers and neat spacing make this page highly readable and professional.



**Figure 5.4:** User control page allow user of website to remotely control the rover.

The "Car Controller" interface by Universal Dynamics offers an intuitive way to control a vehicle using simple keyboard inputs. Users can navigate the car using arrow keys and adjust the speed with "+" or "-" keys. The dashboard displays real-time data including speed percentage and servo angle, enhancing user control and feedback. A clean, minimalistic design with soft gradients ensures a user-friendly experience. Navigation is also supported through a top menu bar with quick access to products, solutions, and user profile options.



**Figure 5.5:** Login page for authorization.

The login interface welcomes returning users with a clean and modern design. It features input fields for email and password, along with a toggle to show or hide the entered password. A prominent blue “Login” button provides easy access to user accounts. At the bottom, there’s a prompt for new users with a link to the sign-up page. The centered white card contrasts well against the dark blurred background, ensuring readability and focus.

## Usability and Design Decisions

* **Responsive Layout:** The site is mobile-friendly using Tailwind CSS responsive utilities.
* **Font Choices and Accessibility:** Clean and readable fonts (Inter, Sans-Serif). Proper color contrast for accessibility.
* **Security:** Firebase ensures secure storage, and authentication routes are protected from unauthorized access.

# 

# Results and Evaluation

## Simulation Results

* Training reward gradually increased from -10 to +50
* Agent successfully followed track in 90% of test simulations

## Real-World Performance

* Rover correctly followed path in controlled indoor settings
* Slight drop in performance in outdoor uneven lighting
* Latency under 0.5 seconds per inference

## Web Interface Testing

* Signup/login functions tested with multiple users
* Model upload verified (size < 5MB)
* Live images updated every 10 seconds

# Conclusion and Future Work

## Conclusion

We successfully developed a vision-based autonomous rover using reinforcement learning. The simulation-trained model was deployed to real hardware, and the system was complemented with a functional web interface.

## Future Work

* Add GPS for navigation
* Use stereo cameras for depth estimation
* Deploy the RL model on a microcontroller (e.g., ESP32 with Edge Impulse)
* Improve website for real-time control and feedback

# References

* Sutton, R. S. (2018). *Reinforcement Learning: An Introduction.* Cambridge: MIT Press.
* Unity ML-Agents Toolkit Documentation
* Firebase Web SDK Documentation
* NVIDIA PilotNet Research
* OpenCV Python Tutorials

# Links

* Github:
* Video: