Federal University of Technology - Paraná - Brazil – UTFPR
Electronics Academic Department – DAELN
Informatics Academic Department – DAINF
Computer Engineering
Integration Workshop 3 (EEX23) – S71 – 2023/1

Technical Report
Self Walking Luggage

Heitor D. Trevisol – htrevisol@alunos.utfpr.edu.br
João L. M. Ferreira – joaolucasferreira@alunos.utfpr.edu.br
Thiago R. Bernardo – thiagobernardo@alunos.utfpr.edu.br

June 2023

Abstract

Wheelchair users face daily challenges, from inaccessible environments to limited mobility, societal stigmas, and emotional strain. With this in mind the aim of this project is to develop a solution that facilitates carry-on luggage transportation for wheelchair users in airport environments. By addressing the specific needs of wheelchair users, the project aims to enhance accessibility and improve the overall travel experience. By addressing the challenges of a wheelchair user traveling trough an airport, this project has implemented a prototype of a system capable of carrying the user's carry-on luggage without the need for manual control freeing the user so he or she can safely wander from one terminal to another without having to struggle or calling for assistance.

1 Introduction

People with motor disabilities are subject to many difficulties and problems when they want/need to be independent, that is specially true to those that are wheelchair bound. One of these struggles is traveling with luggage, having to manage not only the wheelchair movement (even more so in the case of a non-automatic one), holding your documents and tickets but also having to push/pull a luggage can be very challenging and stressful in a situation that shouldn't be so.

With these considerations in mind, the idea of this project is to help alleviate some of the burden of a wheelchair user when travelling trough an airport by building a system capable of carrying the user carry-on luggage without the need to be manually controlled, while keeping the user freedom of movement in the airport hall.

However, due to the difficulties associated with building a robot robust enough in the allotted time and budget for the project, the result presented is a smaller proof of concept robot capable of carrying small loads up to 5 Kg, but it should work in a scaled up version.

1.1 Overview

The project consists of 3 major parts, the smartphone application (app) which is responsible for the interaction between user and the whole system, the Central System that controls the renting, state of each robot (as said before this project has only one robot as it is a proof of concept) and manages the communication between all parts, and finally the robot, or how we call it the Carrier, which centralizes the sensors and the processing of its signals, the actuators so it can move as expected and the mechanical parts that enable the Carrier to move and carry a carry-on luggage. For the system to be able to follow the person, a visual marker must be carried by the user, positioned in the back of the wheelchair so it is completely visible from the back. The technology used is called "Augmented Reality Tags", or ARTags. Figure 1 presents an overview of all parts of the developed solution.

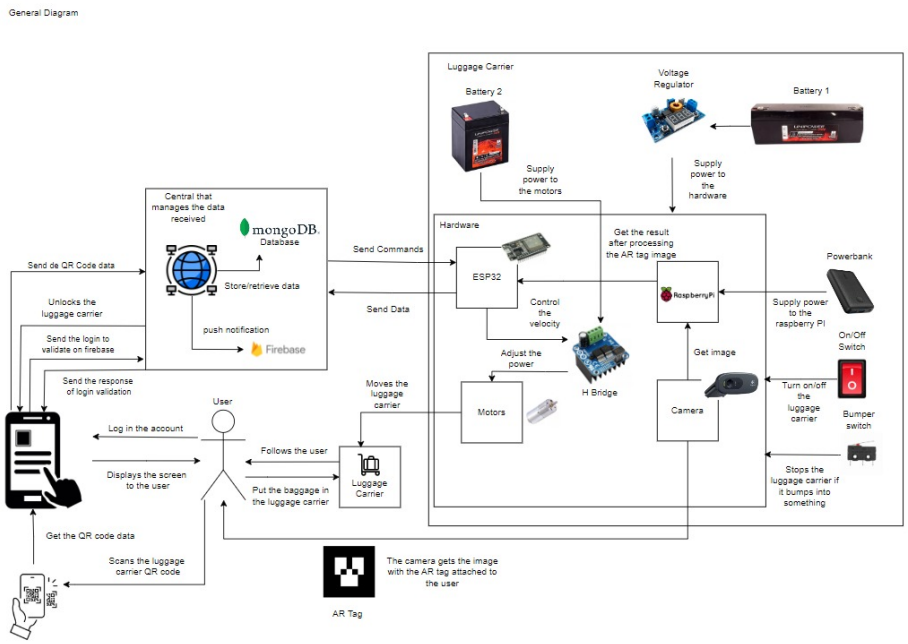


Figure 1: Full Diagram of the system

The mechanical aspect of the project serves as the base of the Carrier, it has 3 wheels in total, a dummy wheel on the back and 2 bigger wheels on the front, each connected to a motor with support rods for a second base where the basket for the carry-on luggage, and a support for the camera.

The Carrier part is also where a embedded system is located and it consists of a ESP32[1] connected to the 2 motors via H bridges[2] and to bumper switches

[1] ESP32 is a microcontroller, more information can be found on Espressif Systems website [1]

[2] nd thus control the movement of the robot. More details can be found in this Modular Circuits

that are part of the safety system. The ESP32 is connected to a Raspberry Pi 4B+[3] which in turn is connected to a camera, the pair Raspberry and camera is responsible to receive and process images and pass to the ESP32 signals for following the user.

The app was developed for Android smartphones, and serves as the Human-Computer Interaction enabling the user to connect to the system, rent a Carrier and control it so it can start or stop following the user.

The Central System consists of a server with a MongoDB database [4] and it is responsible for the integration between the app and the Carrier, it keeps the communication between the parts working as well as keeping records of the users, the rents and the state of the Carrier.

1.2 Requirements

1.2.1 Functional Requirements

- RF001 - The structure must be able to accommodate a standing carry-on luggage (limit size 0.55 x 0.40 x 0.23 m), held inside a basket of 0.45 x 0.25 m);
- RF002 - The structure will be powered by 2 plastic and rubber wheels of 14.5 cm of diameter in the front, and one caster wheel - for support only- in the back;
- RF003 - The structure will move based on parameters of the embedded system, impulsioned by 2 DC motors of 85RPM, 122.36 kgf.cm, and 12 Volts, being able to move at up to 27 cm/s (1 km/h);
- RF004 - The structure must have a bumper switch to disable the system in case of obstacle bumps;
- RF005 - The image processing of the embedded system must user Augmented Reality (AR) Markers to follow / calculate distance to the user;
- RF006 - The embedded system must have a on/off switch for the user;
- RF007 - The battery must have a eletric tension of 12V and a current of 2.3Ah;
- RF008 - The embedded system must communicate with the central using a ESP32 for WiFi communication;
- RF009 - The embedded system must monitor its communication state;
  - RF009.1 - The embedded system must present the communication states through LEDs (on, off);

article [2]

[3] Raspberry PI is an embedded computer, more information can be found in their website [3]

- RF010 - The embedded system must emit a noise in case of certain events;
  - RF010.1 - The events for it are:
    - WiFi communication is lost (1.5s timer from each beep);
    - System is not able to find the user to follow after 5 seconds (continuous);
- RF011 - The embedded system must receive state changes from the central;
  - RF011.1 - Stop the carrier if not received a ping back from the central for more than 10 seconds;
  - RF011.2 - Stop renting if not received pings for 5 minutes;
- RF012 - The central must receive a message from the luggage carrier every 5 seconds to check if it's connected;
  - RF012.1 - The central must sent a push notification to the user if it's not connected;
- RF013 - The central must provide a token for the user to perform actions that requires log in;
  - RF013.1 - Update/Delete personal information;
  - RF013.2 - Update mock credit card;
  - RF013.3 - Control a carrier (rent, stop, end);
  - RF013.4 - View past rentals;
- RF014 - The central must store each luggage carrier information;
  - RF014.1 - The QR Code of a luggage carrier must be stored;
  - RF014.2 - The central must store the current state of each luggage;
- RF015 - The central must map what luggage carrier is rented to which user;
- RF016: The app must listen to push notifications;
  - RF016.1: Notification for losing communication;
- RF017: The app must have:
  - RF017.1: Login screen with email, password and name text fields;
  - RF017.2: Information screen to warn dangers of the beta test and to be aware of the surroundings;
  - RF017.3: Settings screen to update name, sign out and delete account;

  - RF017.4: Home screen with history of past rentals and current rental, button to scan the carrier's QR Code, button to stop and end rental;
  - RF017.5: Card screen to update mocked credit card (card numbers are redacted);
  - RF017.6: Error screen for network connection lost.

1.2.2 Anti-Requirements

- AR001 - The carrier won't carry excess weight on the basket only the luggage;
- AR002 - The carrier won't deal with obstacles;
- AR003 - The carrier won't be able to climb stairs and/or escalators;
- AR004 - The carrier won't be able to work in dark places;
- AR005 - The central won't encrypt user information;
- AR006 - The central won't charge real money of the user;
- AR007 - The app won't receive push notifications without internet.

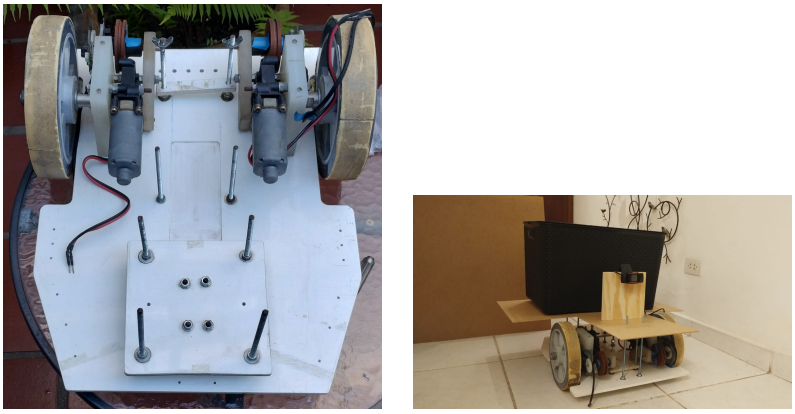
2 Development

In this section, all the processes done and difficulties faced during the project's development will be described, together with the technical details. More information can be found on our blog [5].

2.1 Mechanical Design

The Mechanical Design started with the free online CAD software *OnShape* [6], creating the initial design of the Carrier. This first design was not accepted due to its structural problems originated from the lack of experience in the mechanical field from the team members. As a solution for this big challenge our Professors Heitor Silvério Lopes and João Alberto Fabro offered to lend the robot *Bellator* a robot that was used in many other projects in UTFPR and now it is the structural base of the Self Walking Luggage Carrier.

Bellator consists of a robust base that holds a caster wheel on the back and 2 bigger front wheels, these 2 bigger wheels are attached to 12V DC motors as can be seen on Figure 2(a).



(a) (b)

Figure 2: (a): Picture of Bellator. (b): Picture of the Carrier

Using *Bellator* as the base an extra structural level was manufactured so that the plastic basket and the support for the camera could be affixed (Figure 2(b)).

2.2 Hardware Design

The start of the Hardware Design was the drawing of the electrical schematics in the tool *KiCad* [7], this diagram can be seen in Figure 3

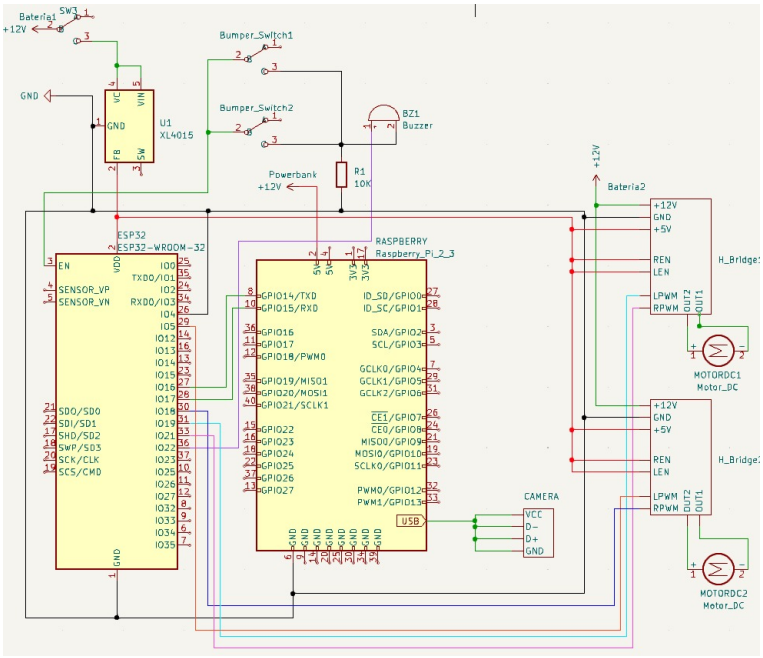


Figure 3: Electrical Diagram

From the diagram, we can see that the ESP32 is the heart of the design and connects all the components together. The Raspberry Pi 4B is the eyes of the project, it is responsible for processing the image captured by the camera and returning the results to the ESP32 so that it can control the motors correctly.

For the control of the DC Motors, after doing some tests on the motor to find it's peak current consumption, it was chosen the BTS7960 H-Bridge module because of its high current tolerance of 43A which is higher than the motor's peak current consumption (10.7A).

For the camera, after some research, it was decided to go with the Webcam Logitech C270, due to its features and reliability. The Logitech C270 stood out for its high-definition 720p resolution, ensuring clear visuals for the image processing and its compatibility with the Raspberry Pi 4B. Overall, the Webcam Logitech C270 met our needs for quality, simplicity, and affordability.

All of the components were tested individually before integration into the final hardware project.

2.3 Firmware

The firmware of the Raspberry Pi is designed to capture the image from the camera and calculate the distance to the ArUco marker as well as the horizontal distance. With this information, the firmware sends it to the ESP32 via UART.

The firmware of the ESP32, written in C++ using the *Arduino IDE* [8], was designed to receive commands from the central system via the internet after the carrier is rented. It includes functionality to compare the QR code received from the central system with the QR code of the carrier to authenticate the carrier. If the QR codes do not match, the firmware won't control the carrier. However, if the QR codes match, the firmware will proceed to control the carrier with the following logic.

When the firmware receives the "STOPPED" command, it will halt the carrier's movement, regardless of the distance between the ArUco marker and the carrier. The carrier will remain stationary until further commands are received.

On the other hand, when the firmware receives the "MOVING" command, it will control the carrier based on the following rules:

- If the carrier is 10% to the right and it is more than 1 meter away from the ArUco marker then the carrier will move diagonally to the left.
- If the carrier is 10% to the left and it is more than 1 meter away from the ArUco marker then the carrier will move diagonally to the right.
- If the carrier is centralized and it is more than 1 meter away from the ArUco marker then the carrier will move forward.
- If the carrier is between 1 and 0.9 meters away from the ArUco marker then the carrier will stop.
- If the carrier is less than 0.9 meter away from the ArUco marker then the carrier will move backward.

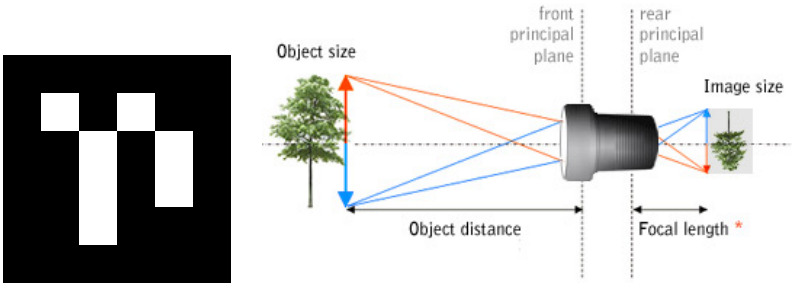
The firmware of the ESP32 will stop the carrier if any of the bumper switches are triggered or if it loses connection to the internet (after 10 seconds). Additionally, it will activate the buzzer at certain intervals if the camera fails to detect the ArUco marker or if there is no internet connection.

2.4 Computer Vision

The luggage carrier project incorporates computer vision software developed in Python using OpenCV [9]. This software plays a crucial role in detecting and measuring the distance between the camera and an ArUco marker. Additionally, it determines the horizontal position of the marker within the image, enabling an estimation of the camera's distance from the user.

The software utilizes the following key components and techniques:

1. ArUco Marker Detection: The computer vision software leverages OpenCV's capabilities to detect ArUco markers (Figure 4(a)). ArUco markers are square markers with unique patterns that can be easily identified and located within an image. By analyzing the image captured by the camera, the software identifies the ArUco marker's presence and extracts its position.
2. Distance Estimation: Once the ArUco marker is detected, the computer vision software employs known marker height and the focal length of the camera to measure the distance between the camera and the marker accurately (Figure 4(b)). This technique utilizes the principle of similar triangles, where the ratio of the marker's known height to its height in the image allows for the calculation of the distance.
3. Horizontal Position Measurement: In addition to distance measurement, the software determines the horizontal position of the ArUco marker within the image. By identifying the marker's center the software is capable of measuring the distance from the center of the image to the center of the marker.



(a) (b)

Figure 4: (a): ArUco marker. (b): Focal length and distance measure. Source: Lensation[10]

2.5 Backend

The backend API for the luggage carrier project is developed using NestJS [11], a robust framework for building scalable and efficient server-side applications. The API consists of three modules: carriers, auth, and users. Each module serves a specific purpose and contains routes to handle various functionalities.

1. Carrier Module:
   The carriers module is responsible for managing the state of the carriers, including their current status, connection checks, and rental handling. All routes within this module require user authentication. The following routes are implemented:
   - Start Rent: The route (**POST** /carriers/:id/start-rent) initiates the rental process for a specific carrier. It verifies whether the carrier is already rented and if the user has a registered credit card. This prevents multiple users from renting the same carrier simultaneously and ensures that users with valid payment information can proceed with the rental (Figure 5).
   - End Rent: Users can end a rental using this route (**POST** /carriers/:id/end-rent). This route includes additional checks to ensure the legitimacy of the request. It verifies if the carrier is currently rented and if the user making the request is the one who rented the carrier. This validation prevents unauthorized users from prematurely ending someone else's rental and ensures that only the rightful user can return the carrier.
   - Change Movement State: The route (**PATCH** /carriers/:id/state) enables users to update the movement state of a carrier. However, it includes an important safeguard to prevent unauthorized control. The system verifies if the carrier is rented to the user making the request. This ensures that only the user who rented the carrier can modify its movement state, preventing interference from other users.
   - Find Rents: This route (**GET** /carriers/rents) provides users with access to information about their past and current rentals. By accessing this route, users can view their rental history and monitor their ongoing rentals. This feature allows users to keep track of their interactions with the luggage carrier service, facilitating better management and accountability.

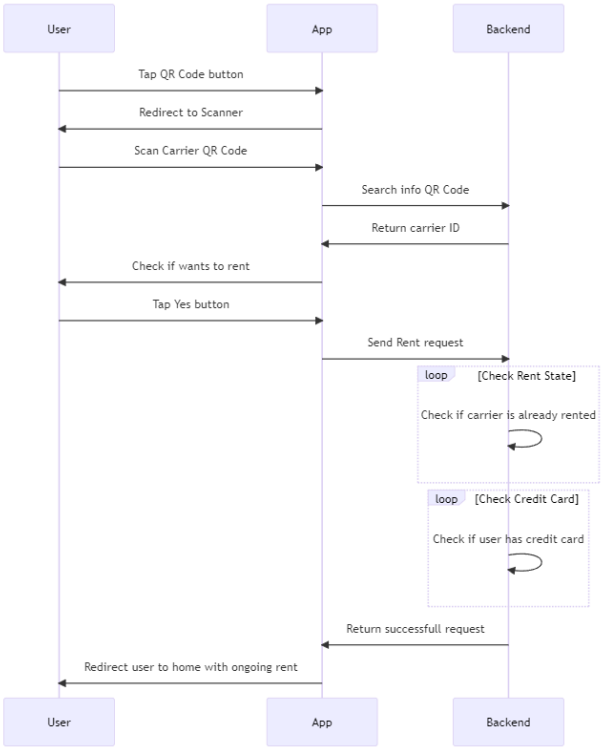


Figure 5: Start Rent Diagram

The Carrier module includes a cron job that runs every 5 seconds to monitor the connection status of rented carriers. This job performs regular checks to ensure that carriers remain connected and operational.

First, it retrieves all rented carriers from the database, identifying the carriers currently in use by users. For each rented carrier, the job checks if the database has received any state changes from the carrier within the last 10 seconds or the last 60 seconds. If the carrier has not sent any state changes within the last 10 seconds, it indicates a temporary loss of connection. In this case, the backend sends a notification to Firebase Cloud Messaging, notifying the user that the carrier has stopped. Simultaneously, the backend updates the carrier's status in the database, marking the rental as completed. This notification helps users stay informed about the carrier's status. If the carrier has not sent any state changes within the last 60 seconds, it suggests a prolonged loss of connection. The backend sends a notification to Firebase Cloud Messaging, indicating that the rental has ended. Simultaneously, the backend updates the rents's status in the database, marking the rental as completed.

The Carrier module is designed to handle state changes that occur on the physical structure of the Carrier. These state changes, including move-

ments and stops triggered by user interactions or external events like bumps, are received by the backend via MQTT [12](Message Queuing Telemetry Transport).

Specifically, the state changes are published to the topic "luggage-carrier/carrier/state" along with the carrier's unique identifier (ID) and the current state. The payload structure contains information about the state of the carrier, such as whether it is currently moving, stopped, or forcefully stopped.

On the other hand, when users initiate changes through the mobile application, such as starting or ending a rental or modifying the movement state, these user-triggered actions are sent to the carrier via MQTT. The payload structure for these messages follows the same structure and is published to the topic "luggage-carrier/carrier".

By utilizing MQTT, the Carrier module establishes a reliable and efficient communication channel between the backend and the carrier. This bidirectional communication allows for seamless coordination and synchronization of state changes between the user interface and the physical carrier.

1. Auth Module:
   The auth module handles user authentication, sign-in, sign-up, and authorization token management. It ensures secure access to the application's features. The following routes are implemented:
   - Sign In: Users can log in using this route (**POST** /auth/signIn). It validates the user's credentials and provides an authorization token for subsequent authenticated requests (Figure 6).
   - Sign Up: The route (**POST** /auth/signUp) enables users to create a new account. It securely stores user information and credentials, allowing them to access the application's features (Figure 7).

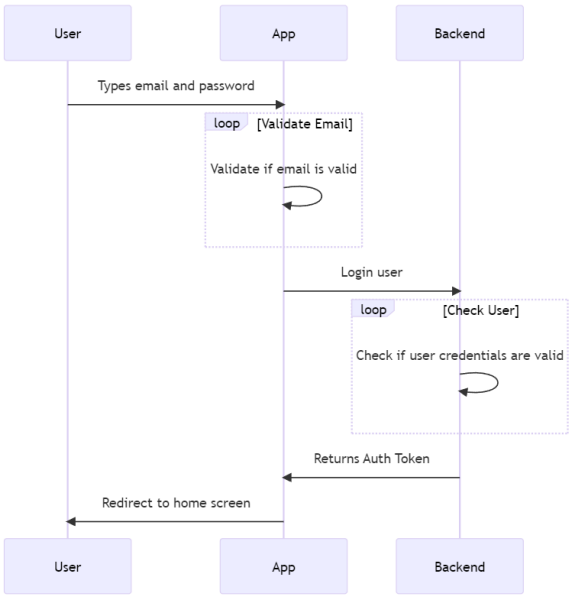


Figure 6: Sign In Diagram

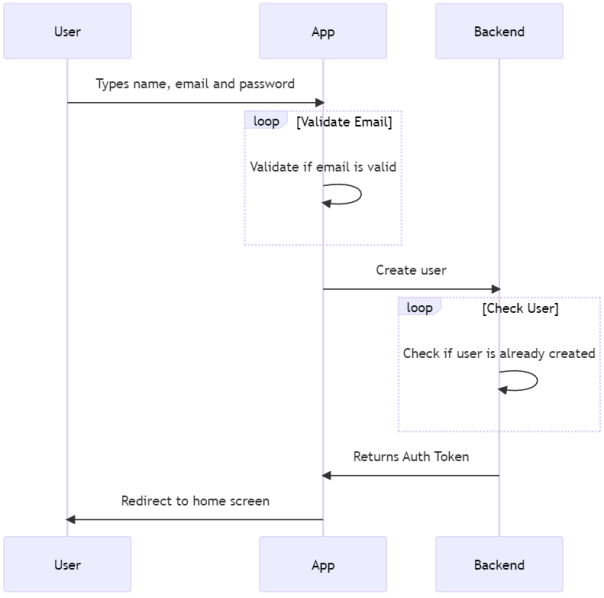


Figure 7: Sign Up Diagram

1. Users Module:
   The users module is responsible for managing user-related information such as name and credit card details. All routes within this module require user authentication. The following route is implemented:

- Update User Information: Users can modify their personal information through this route (**PATCH** /users/:id). It allows for updating user details, including name and credit card information, which were provided on the app.

2.6 Mobile Application

The mobile application for the luggage carrier project was developed using Flutter [13] with the Dart programming language. It serves as the user interface for interacting with the luggage carrier service. The application connects to the backend API using REST for data retrieval and updates, and utilizes Firebase [14] for authentication purposes. The key capabilities of the app are as follows:

- Onboarding Tutorial: The application provides an onboarding tutorial (Figure 8) that explains the potential dangers and safety precautions at the airport. This tutorial aims to educate users and make them aware of safety measures while using the luggage carrier service. It is important to note that since the project is in the beta phase, user feedback and suggestions regarding safety are highly encouraged.
- Authentication: Users can login or create an account using their email and password (Figure 9). The authentication process is handled by Firebase [14], which securely stores user credentials and manages the login functionality. Once authenticated, user information and messaging tokens (used for sending notifications) are sent to the backend to register the user.
- Home Screen: The home screen of the application displays both past and current rents. Users can view their rental history and ongoing rentals, providing them with an overview of their luggage carrier usage (Figure 10). The information is fetched from the backend API through REST requests.
- Start/Stop Carrier Movement and Rent Control: The application offers controls to start and stop the movement of the carrier (Figure 10). Users can remotely initiate the carrier's movement and bring it to a halt when necessary. Additionally, there is an option to stop the rental. These actions trigger the appropriate API calls to update the carrier's status in the backend.
- QR Code Scanning: To initiate a rental, users can scan a QR code (Figure 11) associated with a specific luggage carrier. Scanning the code triggers the necessary backend API calls to verify if the carrier is available and start the rental process for the user (Figure 12). By using QR code scanning and backend verification, the application ensures that only authorized and available carriers can be rented. This process enhances security and provides a streamlined experience for users to start their luggage carrier rental efficiently.
- Personal Information: The application allows users to add or change their

personal information, such as their name and credit card details (Figure 13). This feature enables users to update their profile and payment information conveniently within the app.

- Carrier State Notifications: Users receive notifications regarding changes in the state of their rented carrier (Figure 14). These notifications keep users informed about the status of their rental, such as when the carrier lost connection for more than 10 seconds and stopped, when lost connection for more than 60 seconds and the rent is finished, or when the carrier bumps into something or someone, prompting users to check on it. Firebase Cloud Messaging (FCM) is used to send push notifications to the user's device, providing real-time updates.

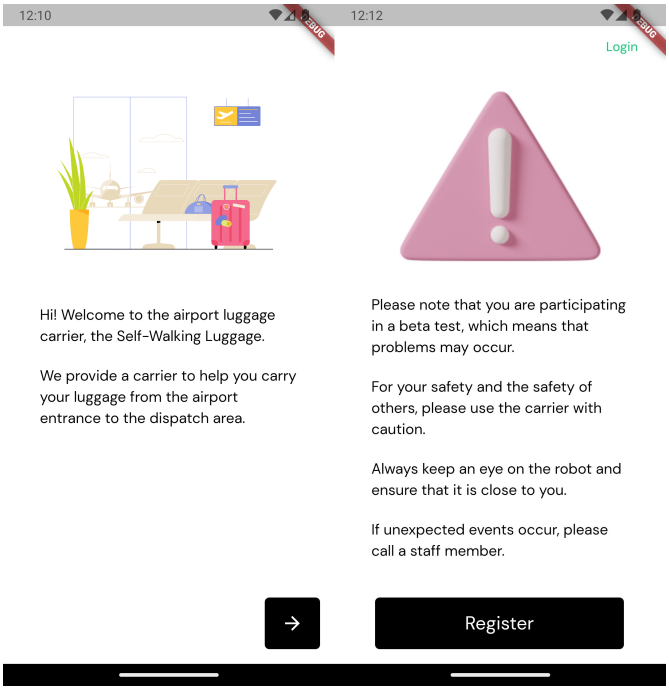


Figure 8: Onboarding screens

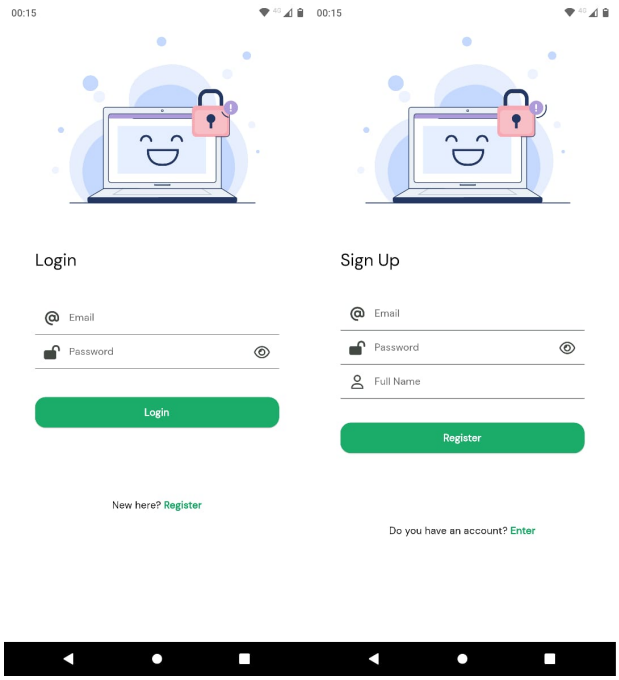


Figure 9: Authentication screens

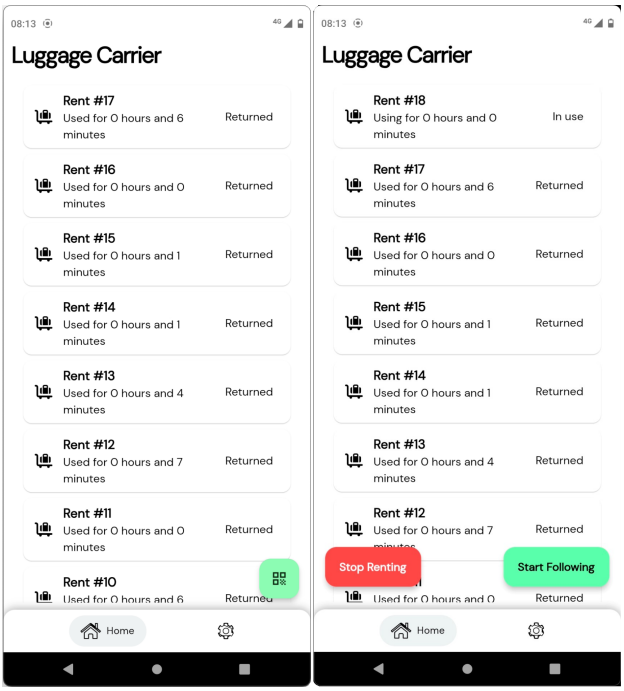


Figure 10: Home screen with QR code and start/stop buttons

Figure 11: Scan QR Code screen

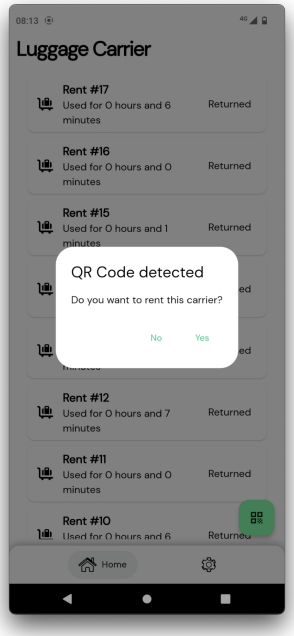


Figure 12: Scanned QR Code screen

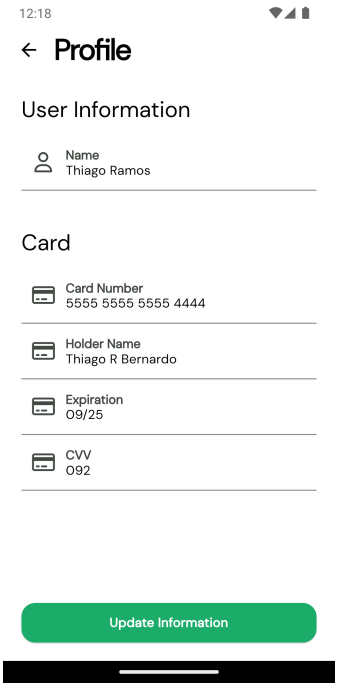


Figure 13: Settings screen

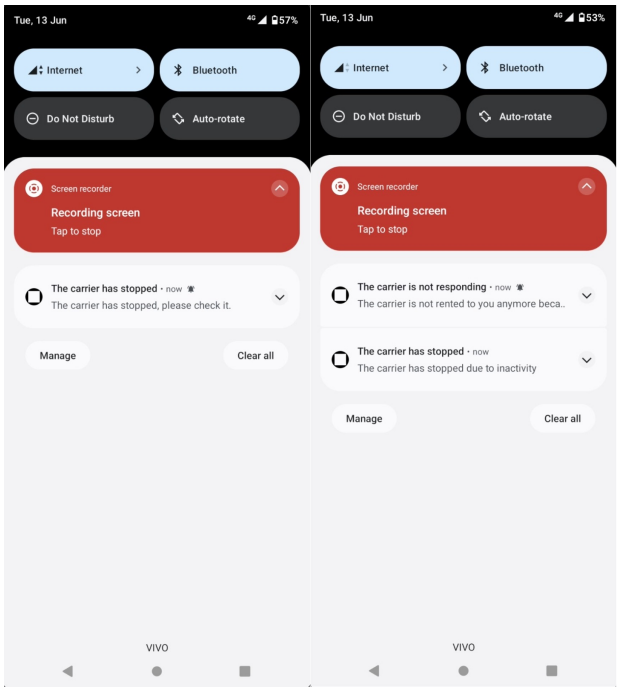


Figure 14: Notifications

By combining Flutter [13], Dart, Firebase [14] authentication, REST API integration, and Firebase Cloud Messaging, the mobile application provides a seamless and user-friendly experience for renting and managing luggage carriers.

3 Results

Our team successfully fulfilled all the initial requirements and completed the project development, resulting in the creation of a tool that is able to enhance accessibility for wheelchair users.

In its final iteration (Figure 15), the system is able to be rented and carry a carry-on luggage while following the user trough an open area, as normally seen in airports.

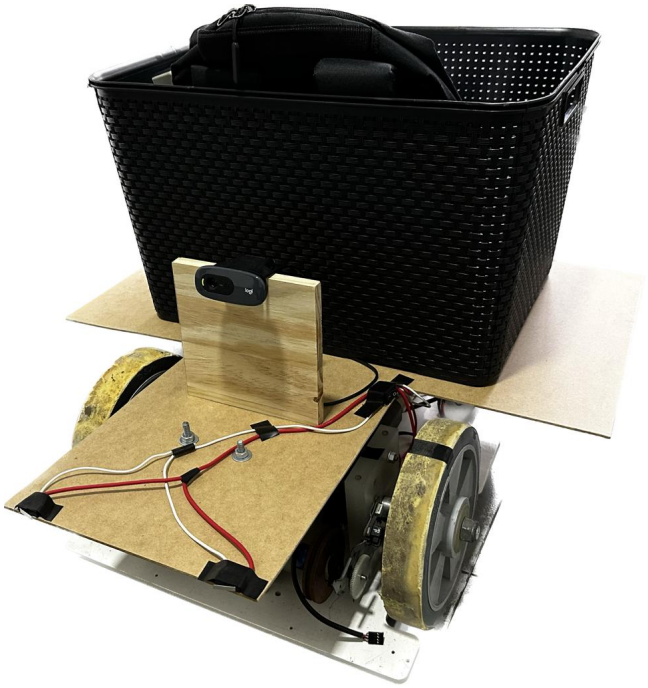


Figure 15: Final Iteration of the Carrier

3.1 Budget

Our project had a planned budget of R$333,33 per member of the team, or circa R$1000,00 in total. In our planning we had expected to spend up to R$1071,17 or R$1767,53 if counting the risk response plans. Our total cost was R$1769,06 considering the price of already owned components and the Bellator that was lent to us by the Professors, which is higher than the assigned budget.

Considering only the components that had to be bought during the development of the project our total cost was R$680,56.

Table 1: Budget

Component	Quantity	Unit Price (R$)
ESP32-WROOM-32D	1	58,50
Voltage Regulator Xl4015	1	37,00
Battery 12V 2,3Ah	1	120,00
Battery 12V 4,5Ah	1	160,00
Buzzer	1	5,00
BTS7960 - H Bridge	2(+1)	53,99
Wood structure	1	56,60
Plastic Basket	1	42,99
Webcam Logitech C270	1	100,00
Raspberry Pi 4B+	1	450,00
Bellator	1	400,00(estimated)
Powerbank 20000mAH	1	80,00
Miscellaneous Expenses		97,00
Total		1769,06

Table 2: Bought components

Component	Quantity	Unit Price (R$)
Voltage Regulator Xl4015	1	37,00
Battery 12V 2,3Ah	1	120,00
Battery 12V 4,5Ah	1	160,00
Buzzer	1	5,00
BTS7960 - H Bridge	2(+1)	53,99
Wood structure	1	56,60
Plastic Basket	1	42,99
Miscellaneous Expenses		97,00
Total		680,56

3.2 Schedule

Table 3 shows an overview of the schedule for this project separated in time planned and worked for each section of the project, divided by the deliverables, during the 8 weeks of development. The column **Expected + 30%** takes into account 30% extra time for the tasks in order to consider problems in the execution of the task or bad estimations during the planning phase.

Table 3: Schedule

Deliverable	Expected Hours	Expected + 30%	Worked Hours
Blog	21:30	28:27	22:30
Mechanical Project	37:00	48:06	34:10
Hardware Project	38:30	50:03	31:05
Software Project	41:00	53:18	40:30
Initial Integration	53:00	68:54	92:40
Overall Integration + Tests	32:00	41:36	38:30
Documentation	29:00	37:42	45:30
Total	252:00	328:06	304:55

Table 3 shows us that the total worked hours on the project exceeds the expected during the planning by 21%, which keeps the total in line with the extra 30% considered for problems in the execution or planning of tasks.

4 Conclusion

The development of this project was riddled with challenges, many of them originating from the mechanical aspect and from the user following system. Since none of our team members possessed prior experience in working with mechanical solutions we relied heavily on guidance from the professors of the course. The loaned equipment *Bellator* significantly minimized our errors and struggles with building a new mechanical structure from scratch.

Regarding the user following system, we underwent many iterations during planning. Ultimately, we decided on adopting the ArUco markers as our primary approach, while the other iterations were relegated to contingency plans in case Plan A showed potential shortcomings.

Looking ahead to future developments, the integration of PDI control (Proportional, Derivative, and Integral) could greatly enhance the robot's movements, offering more precise and seamless tracking capabilities. Overall, PDI control has the potential to elevate the robot's following tasks, enabling it to perform them with a more smooth and clean movement.

The luggage carrier proved to be a successful project throughout all of our tests. The implemented user following system, based on ArUco markers, effectively allowed the robot to track and follow the wheelchair user. The velocity tests demonstrated that the robot operated at safe speeds, ensuring a secure experience for the user. The tests using extra weights proved that the robot can carry up to 5 Kg of extra weight without loss of functionality and speed, with the drawback being more drawn of current from the DC motors battery.

Despite the challenges we encountered and successfully managed, the project

was successfully developed. This endeavor not only equipped each team member with technical expertise but also imparted valuable insights into the dynamics of collaborative teamwork required to accomplish larger-scale projects.

For a comprehensive demonstration of our robot's functionality, we have prepared a video showcasing its performance. You can watch the video by clicking on the following link: Video Link [15]. We encourage you to watch the video to gain a deeper understanding of our work.

Acknowledgements

We would like to express our deepest gratitudes for Professors Heitor Silvério Lopes and João Alberto Fabro help, ideas and criticism during the making of the project, but mostly for lending *Bellator* which became a integral part of our Carrier.

We'd also like to recognize our colleagues in particular Ricky Lemes Habegger, for the help in discussing technical solutions for the project.

References

[1] Espressif Systems. Esp32 https://www.espressif.com/en/products/socs/esp32.

[2] Modular Circuits. H-bridges – the basics https://www.modularcircuits.com/blog/articles/h-bridge-secrets/h-bridges-the-basics/.

[3] Raspberry Pi. https://www.raspberrypi.com/.

[4] MongoDB website. https://www.mongodb.com/.

[5] Self Walking Luggage blog. https://www.notion.so/Self-Walking-Luggage-49daafcffb6f4bda93ee760643b241b2?pvs=4.

[6] OnShape website. https://cad.onshape.com/.

[7] KiCad website. https://www.kicad.org.

[8] Arduino website. https://www.arduino.cc.

[9] OpenCV. Detection of ArUco Markers https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html.

[10] Lensation. https://www.lensation.de/calculator.html.

[11] NestJS website. https://nestjs.com/.

[12] MQTT website. https://mqtt.org/.

[13] Flutter website. https://flutter.dev/.

[14] Firebase website. https://firebase.google.com/.

[15] Self walking luggage - presentation video https://www.youtube.com/watch?v=bh-CVXfKhIg.