Brazil -
Federal University of Technology - Paraná – UTFPR
Academic Department of Electronics – DAELN
Academic Department of Informatics – DAINF
Computer Engineering
Integration Workshop 3 (ELEX22) – S71 – 2023/2

Technical Report

Patrole: Mall Security Bot

Anderson N. Silva– andersonnogueira@alunos.utfpr.edu.br
Andrea A. S. Oquendo– oquendo@alunos.utfpr.edu.br
Erick T. Andrade– henriquecastro@alunos.utfpr.edu.br
Fernanda R. C. Neto – fernandaneto@alunos.utfpr.edu.br
Gabriel H. K. Godinho – gabrielgodinho@alunos.utfpr.edu.br

November 2023

Abstract

In the ever-evolving urban environment, night security in spaces such as shopping malls has become a vital concern. Night security in shopping malls plays a crucial role in preserving the integrity of these urban spaces after business hours. During the night, the absence of commercial activity creates an environment prone to unwanted activities such as break-ins, theft and vandalism. This context highlights the need to guarantee the integrity of these spaces after business hours, which requires innovative solutions. In addition, the complexity of night security work is accentuated by the vastness of the areas to be monitored and the limited visibility provided by reduced lighting. The potential presence of intruders further adds to the complexity, requiring innovative approaches to maintain security and tranquility in these essential community facilities.The solution for night security in shopping centers has taken shape through the Patrole project, an autonomous robot that travels along a predefined route. Equipped with people and movement detection algorithms, the robot is capable of emitting a high-volume audible alarm when it identifies suspicious activity. This project is complemented by integration with a smartphone app, giving the night security guard the ability to monitor the robot's status in real time, including its location and any relevant detection. This comprehensive approach aims to improve the effectiveness of night security by offering a proactive response to potential threats.

1 Introduction

This project was developed for the Integration Workshop 3 subject, of the Computer Engineering course at UTFPR, Curitiba Campus.

As motivating factors for the development of this prototype, we can cite the following problem: Ensuring night security in places like shopping malls is crucial for preserving urban integrity after business hours. The absence of commercial activity at night creates a vulnerability to incidents such as break-ins

and vandalism. Protecting these areas requires inventive solutions because of the large areas to monitor and limited visibility from dim lighting. The potential presence of intruders adds complexity, necessitating creative approaches to maintain security and tranquility in these essential community hubs.

Considering these factors, we have established our mission and objectives: to enhance the efficiency of night security guards by facilitating their work through an automated method for detecting intruders in a specific region.

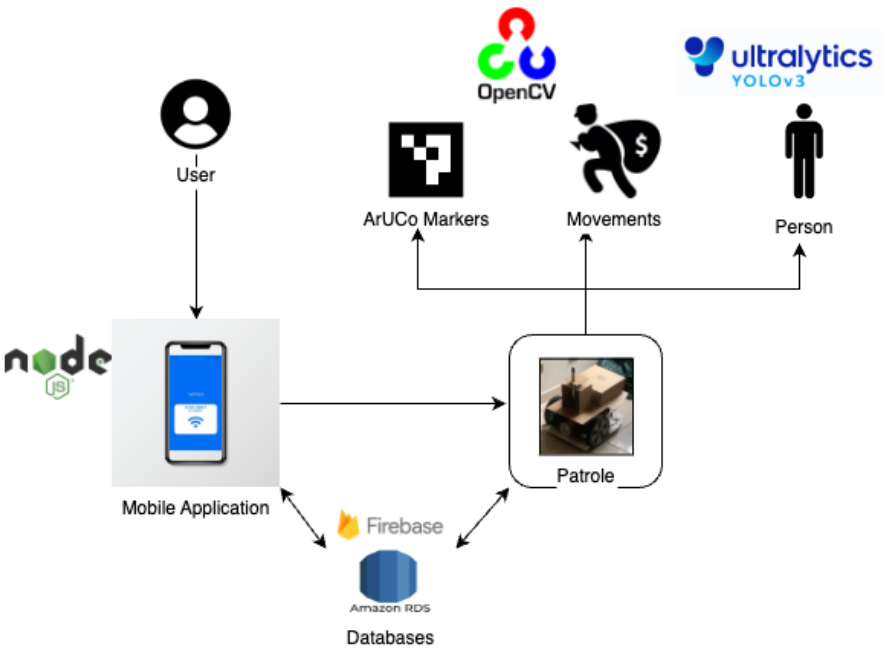


Figure 1: High-level diagram of how the project works

With these problems and motivations in mind, it was possible to develop the project described in this document. By analyzing the diagram in Figure 1, it is possible to understand how the user interacts with the robot, how it works and the technologies used to make this possible.

All user interaction is done via the mobile application, where the user can record routes, execute routes, view the current status of the patrol and receive detection alerts.

The application communicates with the robot via HTTP requests, through which all commands are sent. The data collected by the robot, such as saved routes, executed routes, steps, times and more, is all stored in Amazon's RDS. People and motion detection images are stored in Firebase.

The robot recognizes ArUCo markers on the wall to locate itself and repeats the recorded route, taking short breaks to detect people and movements.

To illustrate a complete use case, the user first connects to the robot through the application, then selects the option to record a route and guides the robot

along the route, starting and ending by reading the same hub. To save the route, the user can move the robot forward, rotate it to the left or to the right and, finally, the user names the route, selects the interval between executions in minutes and selects the number of repetitions. After saving the route, the user chooses the option to execute a route and selects the one that has been saved and positions the robot next to the first aruco. From that moment on, the robot will start repeating the recorded route, sending updates of the arucos read to the application and alerts of detections.

ArUcO marker

markers

(You should explain what are and where are positioned these ArUcO markers....say it is a marker that allow for identification of the "relative position" of the robot relative to the marker, and that the markers should be placed in the walls along the route, at most spaced how many meters from each other....and that each maker has an unique ID...)

1.1 Overview

2 Project Specification

The project specification was separated into four main categories: Mechanical, Embedded, Mobile Application and Web Server requirements. The functional requirements for each of these categories can be found in Sections 2.1, 2.2, 2.3 and 2.4. Additionally, Section 2.5 outlines the anti-requirements.

2.1 Mechanical Functional Requirements

- FRM01 - The structure will consist of a support base with wheels and an additional structure on top of that where will be positioned electronics and the camera. components are positioned.
- FRM02 - The structure must be able to allow the robot to move around the mall, monitoring the area.
- FRM03 - The structure will execute specific movements or actions based on data received from the embedded system.
- FRM04 - The structure will execute specific movements or actions based on data received from the embedded system. repetido?
- FRM06 - The structure must be able to move at a speed of at least 20 cm/s.
- FRM07 - The structure should stop moving and trigger an audible and app-based alarm if it detects a safety problem as unexpected: movement, presence or sensor data. presence?sensor data? what do you mean?
- FRM08 - The camera support must be able to maintain the camera's height throughout the entire journey without any obstruction to cameras vision.
- FRM09 - The structure will move by means of a system of a pair of motors.

2.2 Embedded Functional Requirements

- FRES01 - The embedded system must have a central unit for image processing, and a second unit for sensors and motors.
- FRES02 - The embedded system secondary unit must send all collected data to the central unit.
- FRES03 - The embedded system central unit must send all collected data to the cloud server.
- FRES04 - The embedded system central unit must send motor movement actions to the secondary unit which need to be performed.

5?

- FRES06 - The embedded system must be able to collect data about the orientation of the movement during a surveillance route.

7?8?

- FRES09 - The embedded system must be able to collect images in real time from the environment during a surveillance route.
- FRES10 - The embedded system must be able to trigger an alarm to make sound, alerting the operating team.
- FRES11 - The embedded system must have a 5V power supply to connect the microcontrollers, servo motor and sensors.x
- FRES12 - The embedded system must have a 12V power supply to power the robot's motors.
- FRES14 - The system must have an integrated circuit to centralize the connection of all sensors to its unit of processing.
- FRES15 - The embedded system must control the movement and speed of the motors.
- FRES16 - The embedded system must process the camera images looking for ARUCO markers.
- FRES17 - The embedded system must be able to reproduce a route previously recorded by the operational team.
- FRES18 - The embedded system must be able, for each identified ArUco marker, to reproduce the movements recorded up to the reading of the following ArUco marker.
- FRES19 - The embedded system must trigger the alarm if it detects a suspicious occurrence.
- FRES20 - The embedded system must process the camera images looking for the presence of people and movement.

21?22?23?24? Maybe remove the numbers?Re-number?

- FRES25 - The embedded system must collect and send the Camera data to the cloud server in real time.
- FRES26 - The embedded system requests the route's information to the cloud server and keeps it locally when starts executing a previously recorded route.
- FRES27 - The embedded system must be able to establish a communication with operating team through the mobile app.
- FRES28 - The embedded system must send security and control notifications to the operating team.
- FRES29 - The embedded system must be able to execute commands received from the operating team.
- FRES30 - The embedded system must stop working if an ArUco marker is not found when it was expected to be and wait for the operating team to intervene.
- FRES31 - The embedded system must stop right after the alarm system is triggered, depending on interaction with the security team to get back on track.
- FRES32 - The embedded system must be able to communicate between the central processing unit and the secondary one, and must use the I2C protocol to do so.
- FRES33 - The embedded system must have a voltage regulator that will be connected between the motors' power supply and the second processing unit.

2.3 Mobile Application Functional Requirements

1?2?3?4?

- FRMA05 - The mobile app must allow the operating team to select a previously recorded surveillance route to execute.
- FRMA06 - The mobile app must allow the operating team to visualize the status of a current running robot.

7?

- FRMA08 - The mobile application must allow the operation team to stop the robot at any time and the robot will remain stationary until someone physically intervenes.
- FRMA09 - The mobile application must notify the operating team that the surveillance route has been successfully completed.
- FRMA10 - The mobile application must notify the operations team of any anomalies in the data received from the sensors.

- FRMA11 - The mobile application must notify the operations team of any unwanted movements according to the images obtained.
- FRMA12 - The mobile application must allow the operational team, at the end of the route recording, to name it and define the time between rounds..
- FRMA13 - The mobile application must communicate with the cloud server to present information about routes and real-time events.
- FRMA18 - The mobile app must allow the operating team record one or more surveillances route.
- FRMA19 - The mobile application must indicate the start of recording of a new surveillance route.
- FRMA20 - The mobile application must have a mechanism to indicate the end of the recording of a surveillance route.
- FRMA21 - The mobile application must enable the operational team to navigate the robot's movements between one ArUco marker and the next one during the configuration phase.
- FRMA22 - The mobile application must have a mechanism to indicate when to search for an ArUco marker to read and store route information from the last marker to the current marker.

always spell the same way!

2.4 Web Server Functional Requirements

- FRWS01 - The web server must store data collected from the sensors.

(which sensors?)

- FRWS02 - The web server must store, for each user, routes that were recorded for each robot.

2.5 Anti-Requirements

- AR01 - The robot won't be able to map its surveillance route automatically.
- AR02 - The robot won't be able to go around obstacles and returns to its route.
- AR03 - The robot won't be able to climb stairs and/or escalators.
- AR04 - The robot won't be able to continue its patrol if there is no internet connection.
- AR05 - The robot will not work properly and may lose its way if the ground is wet.

- AR06 - The robot won't be able to detect its route if the ARUCOs are not positioned in the same places as when the route was recorded, including the same height from the ground.
- AR07 - The robot won't continue on its route if the ARUCO marker display is somehow obstructed.
- AR08 - The robot won't be able to recharge automatically, it will depend on the security team to connect it to the charger.
- AR09 - The robot won't be able to handle rough terrain or small objects on the ground by its own.
- AR10 - The robot will not be able to carry out the surveillance process if the user doesn't record a route manually.
- AR11 - The robot will not be able guarantee properly functioning in the event of deliberate or intentional actions attempting to alter its running route.
- AR12 - The robot will not be able to allow route changes during the execution of a surveillance route.
- AR13 - The robot will not be able to return to its route if it doesn't find the next ARUCO on the route.
- AR15 - The robot will not be able to send the history of data captured from the sensors while it was disconnected.
- AR16 - The system will not be able to allow the user to alter previously recorded and stored routes.
- AR17 - The system will not be able to access camera images if it does not have Internet access.
- AR18 - The height of the camera in the structure is not changeable.

3 Development

In this section, we will provide an in-depth overview of the processes and challenges encountered during the development of the project, accompanied by detailed technical information.

3.1 Mechanical Structure

To start the project, we borrowed the Bellator from our professors. It's a wooden structure with two windscreen wiper motors connected to two large front wheels, and at the back there's a ~~silly~~ caster (rear) wheel. The structure also has threaded bars attached to it. ~~As~~ As shown in Figure 2.

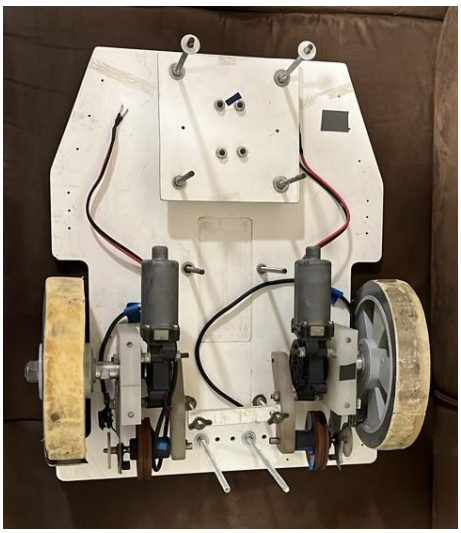


Figure 2: Bellator

However, the material of its wheels was very worn, so we removed it and put a non-slip material in place, because precision in the movement of the wheels was crucial for us, since any slightest slip could change the direction of the robot and it could lose its way. As shown in Figure 3.

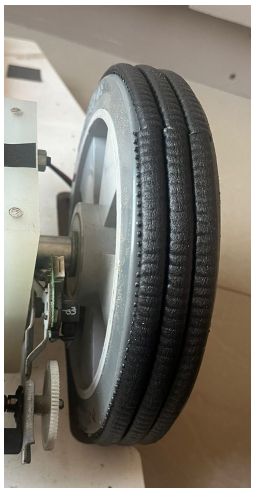
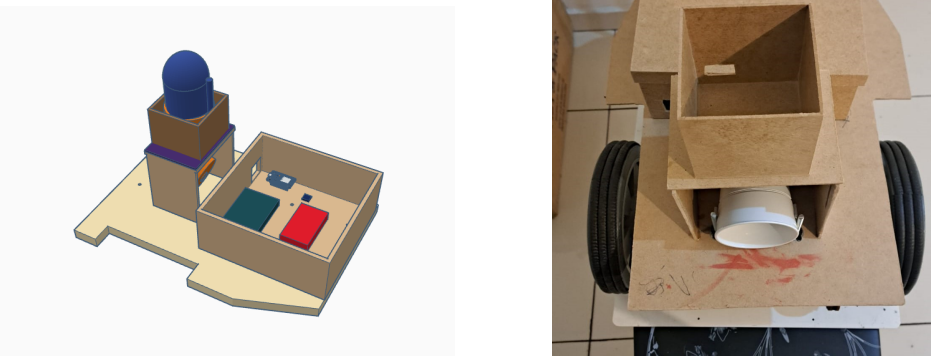


Figure 3: Bellator Wheels With the Non-Slip Material

Now we needed to define the structure of our robot, so we made a 3D model using TinkerCad[1]. The structure consists of a base and 3 boxes, one for the hardware components, another for the horn and the smallest for the camera, which was later removed. Figure 4a shows the result of the modeling and figure 4b the assembled structure.



(a) Structure Design (b) Structure Assembled

Figure 4: Robot Structure

Electronic

3.2 Hardware

The hardware part of the project isn't that complex, we can see in figure 5 below is presented in Figure 5. The ~~that the~~ circuit consists of the 2 wheel motors with an encoder in each of them so we can control the speed and number of revolutions of the wheels, a compass sensor so we know the direction the robot is pointing, an ESP32 that will control the robot and the communication, and finally a Raspberry Pi 3B with a camera and a servo motor, ~~the rasp will process~~ that acquires and processes the images from the camera. We used Kicad[2] to develop the hardware design.

The servomotor allows pointing the camera to the left(-90 degrees), front(0 degrees) and right(+90 degrees).

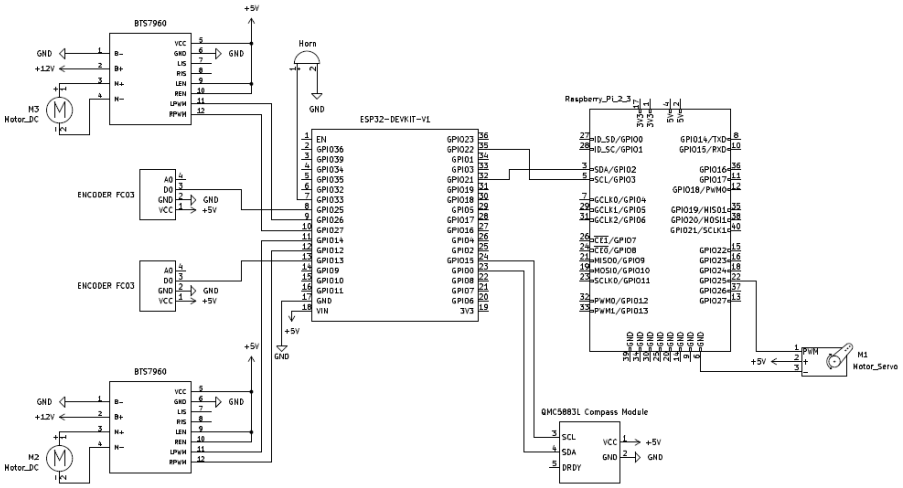


Figure 5: Hardware Design

To power the hardware we used a 12V, 7000mAh battery for the motors and the ESP32, and for the Raspberry we used a 10000mAh powerbank. As for the camera, our initial plan was to use one with night vision and a bluetooth connection, but after a series of tests we had to use our risk response plan and used a camera without night vision and put a flashlight next to it, as the robot will be guarding a shopping center at night, i.e. in a dark environment. Figure 6 shows the Raspberry Pi v2 8MP camera that was used to develop the project.

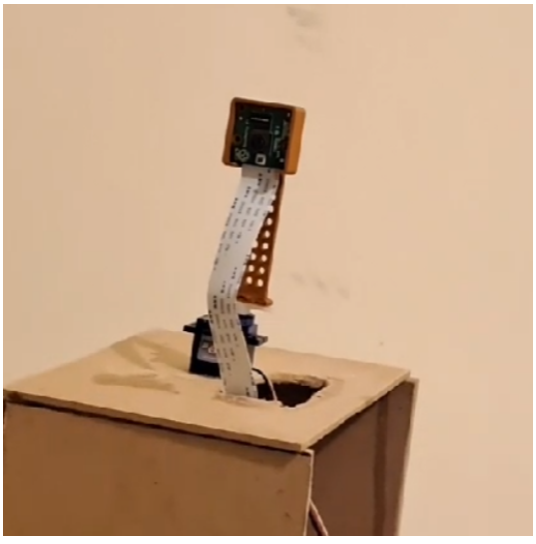


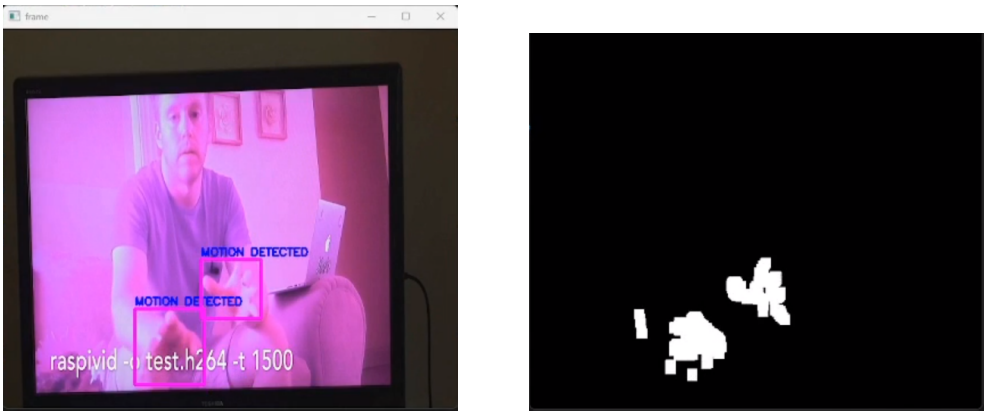
Figure 6: Camera

3.3 Software

3.3.1 Movement Detection

Our initial plan for motion detection was to detect while the robot was in motion, but after a lot of research and conversations with researchers in the field we discovered that it was very complex and we wouldn't be able to implement it in time. So we used our risk response and implemented small pauses in the robot and an algorithm to detect movement with a static camera.

To detect movement, we used Python's[3] OpenCV[4] module and the background subtraction technique [5]. It works as follows: the camera is initialized and the first frame captured is saved as a reference. From then on, all subsequent frames are compared with the first one, and if there is a pixel difference greater than the established threshold, movement is detected. Here we can see 2 images, in Figure 7a we see a frame from a video where a person is moving their hands, as we can see, the motion detection algorithm has already been applied and we have 2 rectangles indicating where, and in Figure 7b we see the result of the background subtraction, in the white squares is where the subtraction showed a difference from the background.



(a) Frame from Camera (b) Background Subtraction

Figure 7: Movement Detection

When using the robot, we set a time of 5 seconds for motion detection, i.e. the camera records and processes the frames for 5 seconds to try to detect something.

3.3.2 Person Detection

For people detection, our first attempt was also with OpenCV, but the number of false positives was enormous, which would have greatly hindered the operation of our project, because every time a person is detected, the alarm is triggered and the robot stops.

Then we moved on to the use of YOLO (You Only Look Once) [6], which is based on a neural network model. It is a deep learning algorithm that uses a convolutional neural network (CNN) for object detection. The results with this new algorithm were very satisfactory, managing to detect people when only part of them was in the image, such as an arm or a leg. However, the computational cost is much higher, so ~~rasp~~ the Raspberry Pi takes a few seconds to process everything.

In figure 8a we can see an example image that was submitted to the algorithm [7], the green rectangles indicate a detected person, and in figure 8b we see an example where the algorithm was able to detect a person with just their face.

(a) Test with whole body (b) Test with just face

Figure 8: Person Detection

3.3.3 Aruco Detection

ArUco?ArUcO?ArUCo?

We use OpenCV for ArUCo detection, as it has its own functions for this purpose. We take 1 frame and look for an aruco, if it is found, its id is returned along with an estimate of the distance from the camera to the aruco, we use this value to adjust the robot when executing the route. Figure 9 shows an ArUCo with id 2 being detected.

a marker

marker

maker

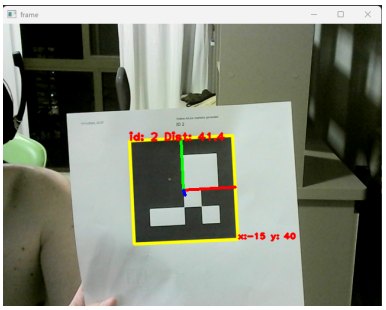


Figure 9: Aruco Detection

ArUCo? What does ArUcO means?Do you know?

"ArUco stands for
Augmented Reality University of Cordoba"

*ArUco markers were originally developed in 2014 by S.Garrido-Jurado et al., in their work “Automatic generation and detection of highly reliable fiducial markers under occlusion“.
(https://www.researchgate.net/publication/260251570_Automatic_generation_and_detection_of_highly_reliable_fiducial_markers_under_occlusion)

3.3.4 Mobile Application

For the application, we used React Native. The app is a very important part of our project, because it's where it that all the actions are done. The first screen that appears when you open the app is to connect to the robot. Once connected, you can record a new route or execute saved routes. To save a route you can rotate the robot in both directions or go forward, you also have the option of reading an aruco, to improve accuracy when executing the route. When the robot is executing the route, the screen shows information about the route, such as the number of markers, the last marker read, etc. If a person is detected or a movement is noticed, a notification is sent to the application and a photo taken by the camera is displayed.

Using Python, it's possible to communicate directly with the AWS database, but a front-end library like React Native doesn't give us that option. Therefore, a NodeJS back-end was developed using JavaScript and hosted on AWS EC2 [8]. The main calls to the back-end from the application are to retrieve the route list and to receive and send notifications during the monitoring phase.

Figures 10, 11 and 12 present the designed interface of the App.

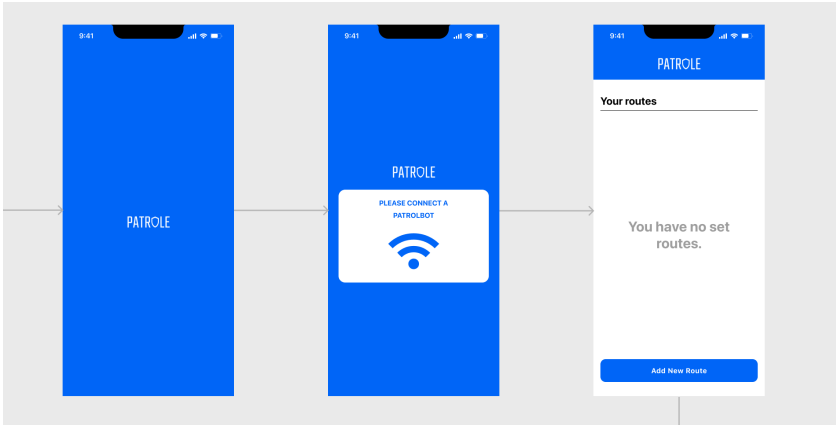


Figure 10: App Screens 1 to 3

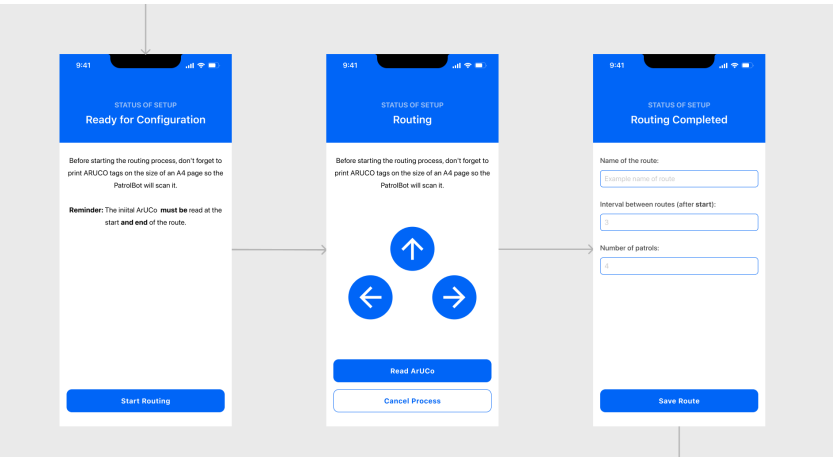


Figure 11: App Screens 4 to 6

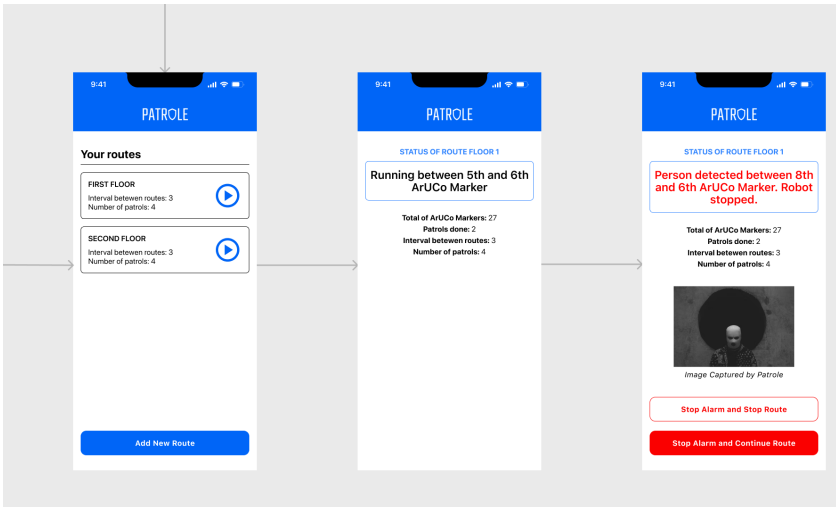


Figure 12: App Screens 7 to 9

3.3.5 Firmware

The firmware part of this project was implemented on two different processing units. The central, and primary one, was a Raspberry Pi 3 Model B, intended to work with high level operations, able to handle: communication with the mobile application; communication with the secondary unit, sending data to the cloud databases and detections of Aruco Markers, persons and movements. The secondary unit was used to centralize the peripherals of the robot as the compass module, the camera servo motor and the motor modules.

Specifically, the coding part of the central unit was made with Python version 3.9.2. This part of the firmware handled various types of tasks such as:

- **Mobile Application Communication**: implemented through a local server using the back-end framework Flask [9]. The communication between the central unit and the mobile app was made through HTTP requests to specific routes that corresponds to commands for Patrole.
- **ESP32 Communication**: established with an I2C interface. The communication was implemented using interruptions on the specific SDA port of the ESP, sending data that corresponds to a command which the secondary unit must execute and, after that, return a successful message indicating that the command was complete.
- **Cloud Database and Storage**: the cloud storage for the images was made using the Firebase Storage [10] feature, that can store files and allow access to them remotely. Finally the cloud database used was the Relational Database Service (RDS) [11] platform from Amazon Web Services (AWS).
- **Detections**: ArUco, person and movement detections was explained in sections 3.3.3, 3.3.2 and 3.3.1, respectively.

Furthermore, The coding of the secondary unit was made using C programming language, based on the ESP IDF API Reference [12]. The main feature of this part of the firmware were task management, in which a task to be performed was received from a command sent by the central unit, such as moving the camera's servo motor to the 90 degree position, activation of motors, etc. The received command triggered an interrupt that allowed immediate processing, or placement in a task queue, of that command understood through the bytes transmitted from one unit to another. At the end of executing this command, a response was sent indicating the success of the operation.

For more detailed information, the source code can be found at folder *firmware* on the remote repository located at GitHub in the following link: `https://github.com/Andxyz8/IntWork3_ELEX23`. Also the classes diagrams of the complete firmware, and the ERD of the cloud database can be found in the *documentation/diagrams* folder.

4 Results

4.1 Budget

Table 1 displays a list of the materials utilized in the construction of this project. During project planning, the expected budget was 1170 ~~reais~~, and if we included the risk plan, the expected budget would increase to 1770 ~~reais~~. After the project was completed, our total expenditure was 1242 reais, taking into account that we used one risk plan that would affect our budget, when compared to what was expected, there was a reduction of approximately 30% in our budget, when compared to the estimated with risk responses.

Table 1: Budget

Name	Qty.	Un. Cost	Cost
Powerbank 10000mAh	1	R$ 80,00	R$ 80,00
Baterry 12V 7000mAh	1	R$ 95,00	R$ 95,00
Raspberry Pi 3B+	1	R$ 300,00	R$ 300,00
Camera	1	R$ 80,00	R$ 80,00
Servo-Motor 360	1	R$ 19,00	R$ 19,00
Encoders	2	R$ 10,00	R$ 20,00
Bellator Base	1	R$ 200,00	R$ 200,00
Motor module BTS7960	2	R$ 60,00	R$ 120,00
Alarm siren	1	R$ 38,00	R$ 38,00
Compass module	1	R$ 25,00	R$ 25,00
ESP32	1	R$ 35,00	R$ 35,00
Non-slip tape	1	R$ 22,00	R$ 22,00
MDF Structure	1	R$ 96,00	R$ 96,00
Flashlight	1	R$ 57,00	R$ 57,00
Extra components	1	R$ 55,00	R$ 55,00
Total			R$ 1242,00

4.2 Schedule

This project was developed over a period of 10 weeks, involving the completion of 7 distinct deliverables, including the week allocated for elaborating this technical report and the final presentations. Table 2 provides an overview of the project schedule and the hours dedicated to each deliverable. In the planning, we expected to work a total of 461 hours on the project; in the end, we ended up working 475, i.e. a 3% increase in the hours worked.

Table 2: Schedule

Phase	Estimated hours	Worked hours
Mechanical Design	23	18
Mechanical Project + Electronic Design	55	39
Electronic Project + Software Design	114	96
Software Project	132	139
Mechanics, Hardware and Software Integration	32	30
Overall Integration and Functional Tests	36	116
Full Technical Report + Video + Presentation	69	37
Total	**461**	**475**

5 Conclusions

During the development of the project we faced many difficulties and challenges. But with challenges comes learning. The Integration Workshop 3 course proved to be a space for us to challenge ourselves and learn to work as a team and in a professional manner like never before on the course.

A lot of things went wrong, especially in the final part of the integration, the reflection of which can be seen in the schedule, where we worked much harder than expected. The team concludes that they underestimated many tasks in the planning phase, tasks that seemed quick were not. We had to use a lot of risk responses because we thought that things like motion detection during the robot's movement would be possible to do on time.

Despite all the difficulties, the team is satisfied with the result, as it met almost all the planned mandatory requirements, with exception of the 7 hours of battery autonomy. The team is also proud of the effort made to complete this project, as we didn't give up even when it looked like everything was going to go wrong.

References

[1] Kai Backman. TinkerCad, 2011. https://www.tinkercad.com/.

[2] KiCad EDA - Schematic Capture & PCB Design Software, 2023. https://www.kicad.org/.

[3] Guido van Rossum. Python 3.12.0, 2023. https://www.python.org/.

[4] Willow Garage. OpenCV 4.5.2, 2021. https://opencv.org/.

[5] Muhammad Sabih. Background subtraction in computer vision, 2022. https://medium.com/@muhammadsabih56/background-subtraction-in-computer-vision-402ddc79cb1b.

[6] Glenn Jocher. YOLO, 2021. https://github.com/ultralytics/ultralytics.

[7] Luana Gonçalves. Detecting people with YOLO and OpenCV, 2019. https://medium.com/@luanaebio/detecting-people-with-yolo-and-opencv-5c1f9bc6a810.

[8] Amazon Web Services. Amazon EC2 - Elastic Compute Cloud. Accessed: 2023-11-29.

[9] Armin Ronacher. Flask Documentation, 2023. `https://flask.palletsprojects.com/`.

[10] Google. Firebase Storage. `https://firebase.google.com/docs/storage`.

[11] Amazon Web Services. Amazon RDS - Relational Database Service. Accessed: 2023-11-29.

[12] ESPRESSIF. Espressif IoT Development Framework (ESP-IDF) - API Reference, 2023. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/index.html.