Federal University of Technology - Paraná - Brasil – UTFPR
Academic Departament of Electronics – DAELN
Academic Departament of Informatics – DAINF
Computer Engineering
Integration Workshop 3 (EEX23) – S71 – 2020/2

Technical Report
FollowYolo

Cesar Batista – cesar_batt@hotmail.com
Felipe Avelino P Rêgo – avelino.felipe@gmail.com
Lucas Rech – coradin.rech@gmail.com
Matheus B. D. da Costa – bigarellimatheus@gmail.com
Victor H. B. da Silva – belinello333@hotmail.com

April, 2021

Abstract

This document reports the development of FollowYolo, the project of a simulated semi-autonomous shopping cart that communicates with a user through an Android app, to make the shopping experience better. To do this, the user can rent the robot in the market, using fictional money in the app. The user has some options of what he or she can do with the robot, like sending it to a place of the market. To start a rent the user will scan a QR Code and use a vest with a printed Augmented Reality Tag (AR Tag) that the robot will use to follow the user. The communication between the app and the robot is made by a server. The app will communicate with the server, giving all the needed information to the server identify which robot the user wants to rent and the information the robot needs to identify the user. Due to the pandemic, all of the physical components were simulated on a robotic simulation environment, adjusting the project and scope to accommodate to those limitations.

1 Introduction

Everyone needs to go to the market some day, it's inevitable. The task of carrying a lot of products with you in a hand basket throughout the market can be very energy consuming and there isn't any technology applied to this problem yet.

FollowYolo is a robot capable of carrying the goods while following a person indirectly with the help of a tag attached to a person's clothes while avoiding the obstacles that it faces. This project's motivation came primarily from the difficulties that elder or disabled people have in carrying market baskets. There are also the ones that just do not like to carry them around. The ultimate goal is to enhance the shopping experience.

The main task at hand is to simulate a robot that can follow a person within a market, communicating with a costumer using an Android app, through a server. The user will be wearing a vest with a printed AR Tag[1]. The robot will have a camera that it will use to capture the user's AR Tag and get information about the user's distance from the robot.

The pandemic influenced directly on some of the project development choices. This was mainly the motivation to do this project using simulations. It was the best way to keep our team productive and minimizing the impacts and risks, including the risk of someone getting infected with the COVID-19 virus.

1.1 Requirements

To accomplish such goals, a list of functional and non-functional requirements was compiled. Some of the main functional requirements are presented in Tables 1,2, and 3. However, some observations are necessary: the app doesn't make any real monetary transactions and the robot will not follow the user if he isn't using the vest with the ARTag.

1.2 Project overview

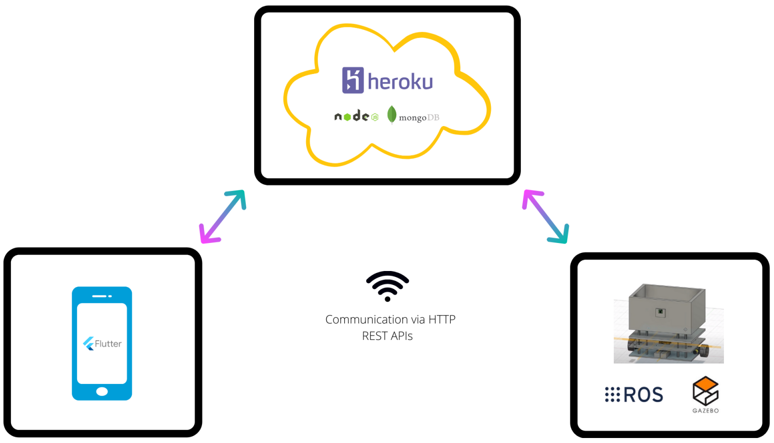


Figure 1: System Overview

Now that all the requirements are listed, a high-level abstract solution can be drawn. This is the intended purpose of Figures 1 and 2.

The initial solution design consists of three main components: the robot itself, a cloud server and an android app. Each of these have a specific purpose.

[1]AR Tag stands for Augmented Reality Tag. It is a fiducial marker system used in augmented reality applications. It was chosen for this project due to the fact that we could extract distance and orientation information about the tag relative to the robot relatively easy.

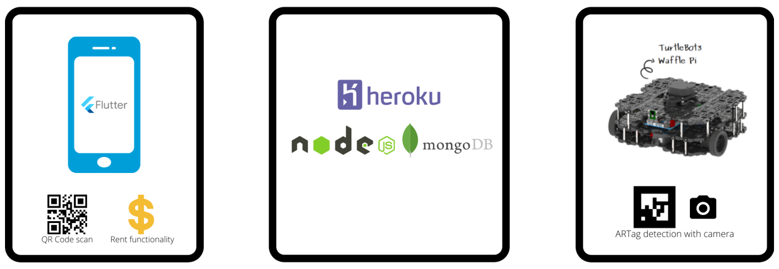


Figure 2: Sub components of the system

First, the robot will be responsible to carry the user's goods and follow him/her around the market. Then, the app will be an the interface between the user and the robot. Finally, the cloud server will manage all communications, data storage and back-end operations.

The robot has a set of functionalities. Those include AR Tag following (presented in more detail in section 2.1, obstacle avoidance and automatic return. An AR Tag is a fiducial marker system to support augmented reality. They can be used to facilitate the appearance of virtual objects, games, and animations within the real world. In this case, it was used not to make an object appear in a screen, but rather measure it's distance and orientation relative to the camera. The obstacle avoidance is the feature of deviating from objects while pursing the goal, which is the AR Tag. The automatic return is a functionality, in which the robot is commanded to return to it's initial position after the rent is finished.

As stated in the last phrase, the robot can be rented. This will happen in the app. One of the main functionalities of the app is to provide means to the user to rent the robot, finalize it (the rent), sign up and login, deposit some cash to rent the robot and notify the market administrator in case of any problems.

The cloud server is an intermediary for the two components mentioned before. It will take incoming requests from the app, such as credentials for the sign up and login functionalities, and store them in the database. If the robot needs any information, it will make a request on the server to get such information.

2 Technologies

The reader can see in Figures 1 and 2 that many technologies were used to design the solution. Here, all of them are presented.

Robot requirements	
Number	**Description**
01	The robot must move in the market environment
02	The robot must be able to follow the user.
03	The robot must be able to avoid obstacles.
04	The robot must go to predetermined locations.
05	The robot must go back to its bay when the rent is over.
06	The robot must be able to scan a environment.

Table 1: Robot (simulation) requirements.

Server requirements	
Number	**Description**
01	The server must communicate with the **app**.
01	The server must communicate with the **robot**.
03	The server must persist **user** data on a data base.
04	The server must persist **robot** data on a data base.
05	The server must handle rents.
06	The server must persist map data.

Table 2: Server requirements.

App requirements	
Number	**Description**
01	The app must register a user.
02	The app must authenticate a user.
03	The app must allow the user to add credits.
04	The app must be able to rent robots.

Table 3: App requirements.

2.1 Robot

To make a 3D model of the designed robot, the Fusion 360 [2] software from AutoDesk was used.

To make a hardware design, there are plenty of options to choose from, but the one used was Eagle[3], also from AutoDesk, a software that assists the design of schematics and PCB board, routing and much more.

First of all, the Diagrams.net[4] online diagram software was used to draw the initial designs and solution's architecture.

[2]`https://www.autodesk.com/products/fusion-360/overview`

[3]`https://www.autodesk.com/products/eagle/overview`

[4]Formerly Draw.io

ROS (Robot Operating System)[5] is a robotics software development framework. It is world-wide used for complex robotics applications. That is the main reason why this framework was used in this project.

A package in ROS in a collection of code with a specific purpose. Some that were used throughout the entire project were the Turtlebot packages. Turtlebot[6] is a well known robot system in the robotics community, which comes with a variety of components, such as sensors, motors, etc.

Another ROS package that plays a big role in this project is the ar-track-alvar[7]. This is a package which allows the user (programmer, roboticist, etc) to easily get the position of a tag in the environment. The key thing is that the tag is an AR Tag (Augmented Reality Tag). The package constantly looks for these markers in a camera feed and provides the system with specific position and orientation of this tag.

Finally, there is a development stack, the ROS navigation stack, that helped the team to develop the navigation algorithms. The main package is the move_base, it receives odometry data, LIDAR data, a goal and a map of the environment.

2.2 Simulation

Gazebo [8] is a simulation software widely used for robotics systems simulation. In fact, a standard Gazebo installation already comes with the installation of the ROS framework. The Gazebo software works side by side with ROS and outputs an accurate simulation of a real-world robotics application, including errors in sensor measures, physics, lighting conditions and much more.

Blender [9] is a free and open source 3D creation suite. This tool allowed the making of the 3D models required by the simulation.

2.3 Server

The server is the key communications and data storage handler.

For managing all communications, the NodeJS[10] runtime environment was used. It is a JavaScript based environment and framework that allows JavaScript to be run in the back-end of an application, allowing the developer to program server-side codes using JavaScript.

[5]https://www.ros.org/
[6]https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/
[7]http://wiki.ros.org/ar_track_alvar
[8]http://gazebosim.org/
[9]https://www.blender.org/
[10]https://nodejs.org

Express is a framework for NodeJS which allows responding to HTTP Requests very easily. JSON[11] stands for JavaScript Object Notation and is a format widely used in data transfer at the application level using HTTP.

MongoDB[12] is a document oriented database software. For data storage, MongoDB utilizes a JSON-based format with schemas. MongoDB was chosen for the database due to its flexibility, code-native access and ease to scale horizontally. This means that it can adapt to the number of incoming connections fairly easily.

MongoDB Atlas[13] is a fully-managed cloud database developed by the same people that build MongoDB. Atlas handles all the complexity of deploying, managing, and healing the deployments on the cloud.

Heroku[14] is a cloud platform used do deploy back-end applications like website hosting, production testing or application scalability.

2.4 App

Dart[15] is a programming language developed by Google, object-oriented, garbage-collected language with C-style syntax, with optional typing, among other things. Dart is the language used by the Flutter framework.

Flutter[16] is Google's UI toolkit for building natively compiled applications for mobile, web, and desktop from a single code base. It was chosen to develop the app for it's simplicity to work with, since a starting programmer can get an application up and running in a matter of a few minutes, excluding the time of installation.

3 Development

3.1 Robot

The main goal of the robot was to be able to follow an AR Tag attached to a person's clothes while avoiding obstacles. The robot itself was designed after Turtlebot 3 Waffle Pi.

[11] `https://www.json.org/json-en.html`
[12] `https://www.mongodb.com/`
[13] `https://docs.atlas.mongodb.com/`
[14] `https://www.heroku.com/`
[15] `https://dart.dev/`
[16] `https://flutter.dev/`

3.1.1 Turtlebot 3 Waffle Pi

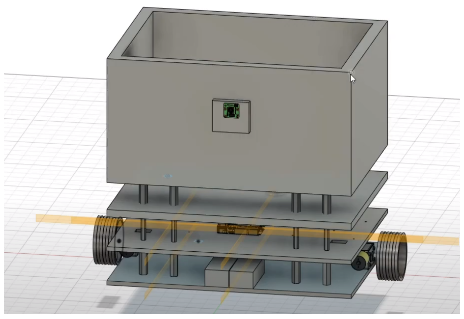
The turtlebot suite was chosen for this project, because it is one of the most famous robotics platform in Research and Development. Besides, it's firmware, hardware and software are open-source, which stimulates it's modification. It was developed having the goal to be cost-accessible, to be small and to have precise sensors. This was decided to be a minimum requirement of the robot platform for this project's development.

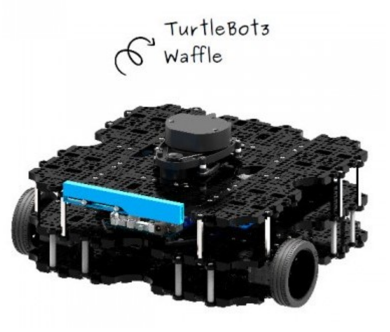
About the sensors, it is worth mentioning the LIDAR (Light Detection And Ranging). It is a technology that consists in detecting physical barriers and it's distance from the sensor in an environment using light.

3.1.2 Mechanical design

The main goal of the mechanical design of the robot was to be as similar as possible to the Turtlebot 3 Waffle Pi robot, so that the simulation can be an accurate representation of how the real robot would behave.

It can be seen in Figures 3a that the designed robot is fairly similar to the Turtlebot 3 Waffle Pi robot, displayed in Figure 3b.


(a) View of the 3D model



(b) Turtlebot 3 Waffle Pi

Figure 3: Comparison 3D model and Real Waffle Pi

The design was made to be 30 cm deep and 40 cm wide. That's because the robot should be able to carry some goods in it's basket. The basket can be seen in Figure 3a. The wheel composition is the same as the Turtlebot's, being two moving wheels, with motors attached, and a dummy one, without any motor attached, to stabilize the robot. Remembering that the mechanical design was not built due to the pandemics restrictions.

3.1.3 Hardware project

There were many things about the hardware specification that limited the design in a certain way. First of all, the current needed to supply the motors is far greater than the one needed to supply the logic circuitry. So the first thing to consider was to include two batteries with distinct purposes. One of them was included in the project to supply the motors, which require a greater energy by themselves. The other battery was included to supply the microcontrollers, the logic components of the modules and the sensors. Figure 4 presents the overview of the design.

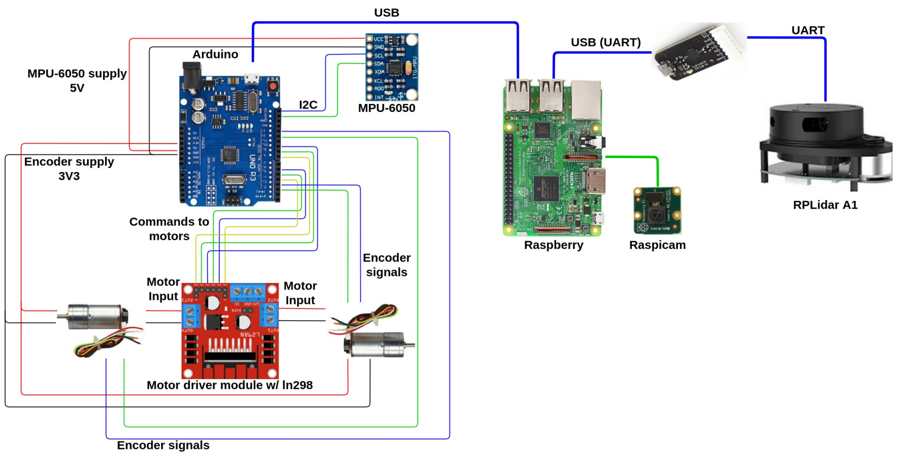
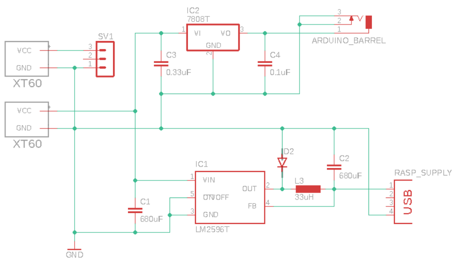
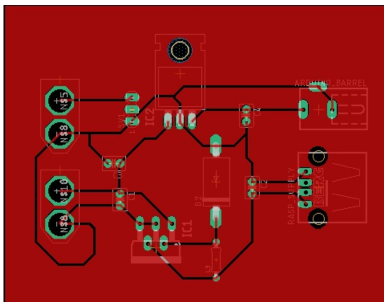


Figure 4: Overview hardware design

The design didn't include any complex electronics design. The first part was the power supply system with the respective connectors for the "power section", meaning the motors, and the "logic section", meaning the microcontrollers and sensors. The second part was the headers to connect all of the components in a single PCB. The reader can see this explanation in Figure 5a.



(a) Schematics of the designed circuit



(b) PCB of the designed circuit

Figure 5: Design circuit

A quick explanation of the designed circuit's components. The 7808 IC is an 8V DC voltage regulator, as is the LM2596T IC. XT60's are battery connectors. The arduino barrel is the input voltage for the arduino board.

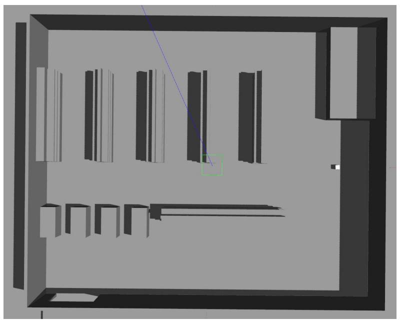
Finally, the corresponding PCB design is as displayed in Figure 5b. Remembering that the hardware design was not built due to the pandemics restrictions.

3.1.4 Simulation

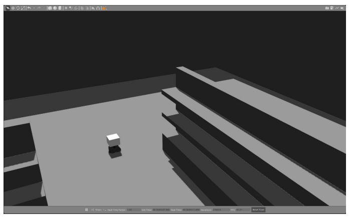
. The simulation contains 4 main components:

- The market
- Robots
- People
- Obstacles

The market was made using Blender. Figure 6a shows the top view of the market model inside the Gazebo simulation.


(a) Top view of the market model

(b) Top view of complete simulation environment


(c) FollowYolo model

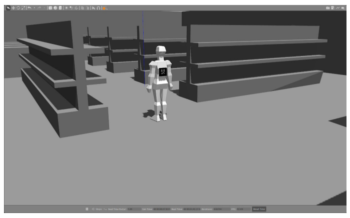

(d) Person model

Figure 6: Simulations Models

The robot models were based on the Turtlebot 3 Waffle Pi, which are available for free. A basket was added to the model to simulate it's real use. For a better visualization, 3 robots were simulated, though only one is actually used. This means that only one is actually being rented. Figure 6c shows the robot model on a simulated scene.

The people were also modeled using Blender and the AR Tag on their back was modeled separately and combined afterwards.

One person represents the user of the system, another is a static obstacle, simulating a person standing still looking at a shelf, and the third one is a dynamic obstacle, simulating another person walking around the environment. These 3 people can be better seen in Figure 6b. Figure 6d is an example of a person model as shown in the simulation.

The fixed obstacles were modeled as 2 boxes. One of them is 100x100x100 centimeters and the other is 15x15x15 centimeters. The complete simulation environment is depicted in Figure 6b.

3.1.5 Communication with the cloud server

To receive and send information to the cloud server a client was made to run on the robot. This client main responsibilities are:

- Receive messages from the server to start the robot when rented via app
- Receive messages from the server to finish a rent
- Send messages to the server when lost
- Receive messages from the server with coordinates to go to when lost
- Send messages to the server with the market map
- Receive messages from the server with the market map

This client was made using the programming language *Javascript* and *NodeJS* as the JS runtime that allows *Javascript* to run on the Back-end. Figure 7 represents the general flow of execution of the robot client.

3.2 Server

NodeJS was chosen as the framework to develop the cloud server using the library Express to build a REST (Representational State Transfer) API, so the communication between the app and the robot and the database could be established.

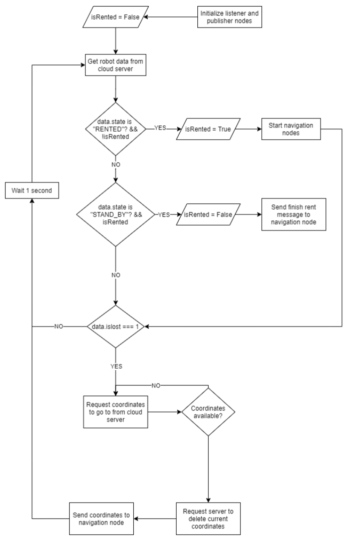

isRented = False
Initialize listener and publisher nodes
Get robot data from cloud server
data.state is "RENTED"? && !isRented
YES
isRented = True
Start navigation nodes
NO
Wait 1 second
data.state is "STAND_BY"? && isRented
YES
isRented = False
Send finish rent message to navigation node
NO
NO
data.islost === 1
YES
NO
Request coordinates to go to from cloud server
Coordinates available?
Send coordinates to navigation node
Request server to delete current coordinates

Figure 7: Robot client flowchart

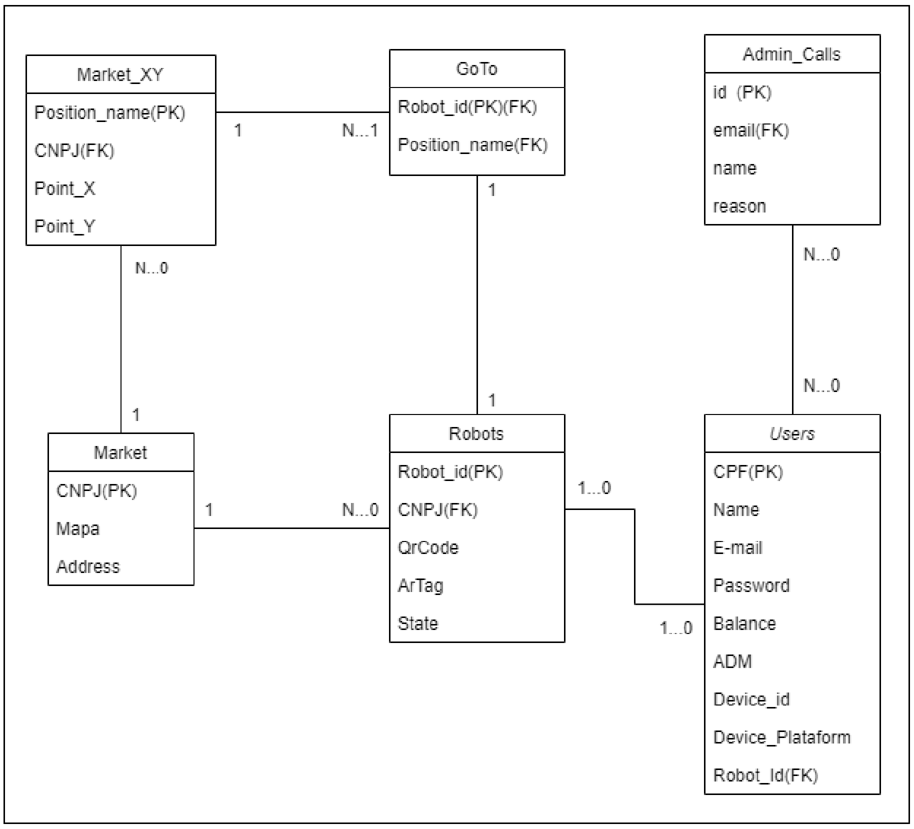


Figure 8: Entity Relationship Diagram.

3.2.1 Database

For the Database the NoSQL MongoDB is used. Figure 8 shows the Entity Relationship Diagram of the database.

The "Users" Table, holds all Users and Admin User information, which relates with the "Robots" table, where a user can rent a robot.

The "Admin_Calls" Table, holds the information of the user that has requested help to an Admin via the App.

The "Robot" Table, holds all the robot information, like his State, if it is rented or is in stand by.

The "Market" Table, holds all the market information, like the map pre-saved that the Robot will use.

The "Market_XY" Table holds all the position of the map that user can send the robot when it is lost.

The "GoTo" Table, holds the position name that the robot will go when the user via App request the robot to go when it is lost.

3.2.2 Host database on the Cloud

The database needed to be hosted on the cloud so the server could access it. Atlas MongoDB allowed the team to host the database on the Cloud, since it's the same creator of the database it self.

3.2.3 Host server on the Cloud

Similarly the Server needed to be hosted on the cloud so the app and the robot could have access more easily to the server. For that Heroku was chosen to host the Server on the cloud.

3.2.4 Integration with the server

The functionalities developed for the robot were mentioned in item 3.1.5. As for the app, they are:

- Receive sign up from the app
- Authenticate login from the app
- Receive message from the app to add balance to the user
- List active user to the Admin on app

- List states of the robots to the Admin on app
- Receive message from the app to start a rent
- Receive message from the app to end a rent
- Receive message from the User to call an Admin via app
- Send message to the admin when a user request via app
- List predetermined location of the map
- Receive message of the predetermined location via app
- Indicate if the robot is lost

Each functionality has an endpoint associated, where the app can communicate with.

3.3 App

The app objective is to provide a user interface in his/hers smartphone capable of helping the user to do all the required operations to rent a robot, like login, sign up, scan QR code, see balance, start rent, etc. It also gives the server all the needed information to start and end a rent, as which robot the user wants to use, etc.

Another set functionality is the administrative toolkit to an admin to operate in case of some failure. The app possible operations can be seen on the Figure 9.

For the app, Flutter and Dart were chosen as the development platform. Because of its simplicity and huge amount of tools to use, and despite of being a newer platform, there is a lot of help that you can find on the internet.

After some study, the software was made. Figure 9 shows the Use Case Diagram. This also includes some API routes documentation, things like all of the needed API Endpoints, the request and responses and the content of those. Because the app communicates uses the server REST API to send all the needed requests to the database, in JSON format, for example, sign up a new user or check the balance. In Figure 10 it is possible to see the normal renting process, with the app final design.

3.4 Tests

The following tests were made to check for the requirements defined in Section 1.1:

- New user sign-up and login using the app

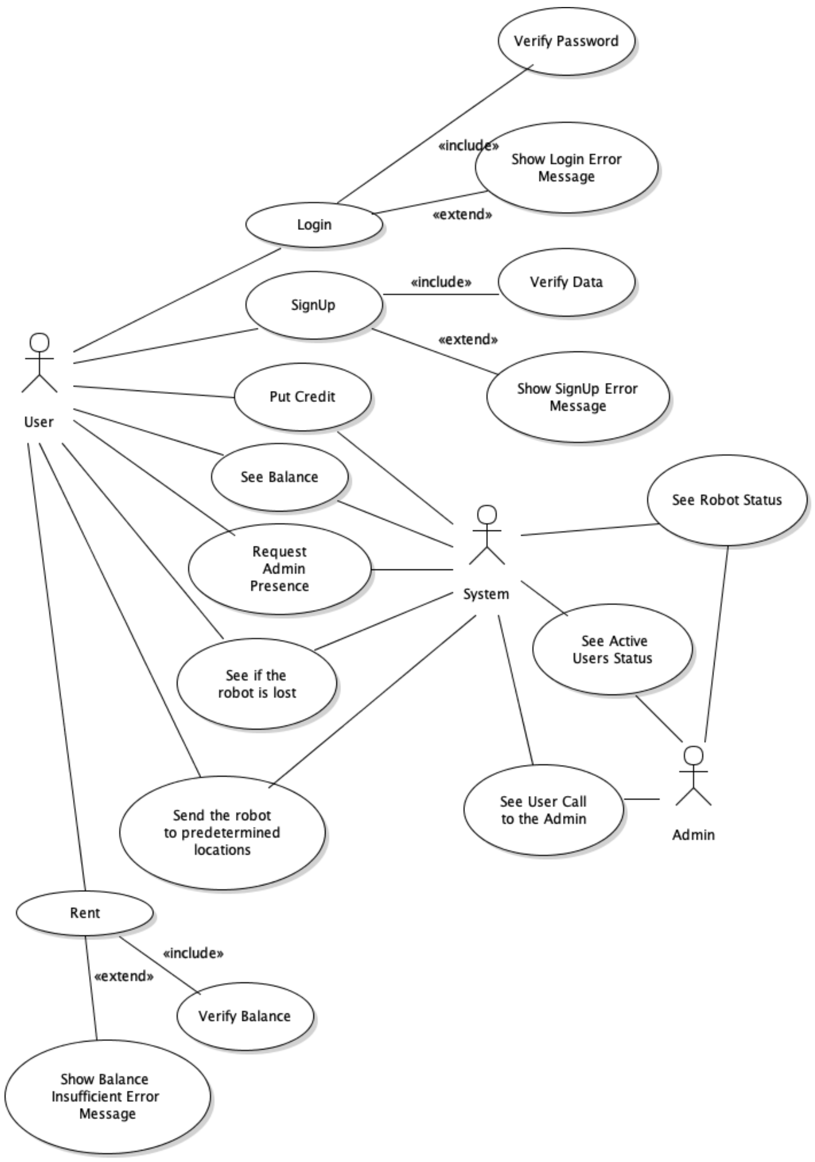
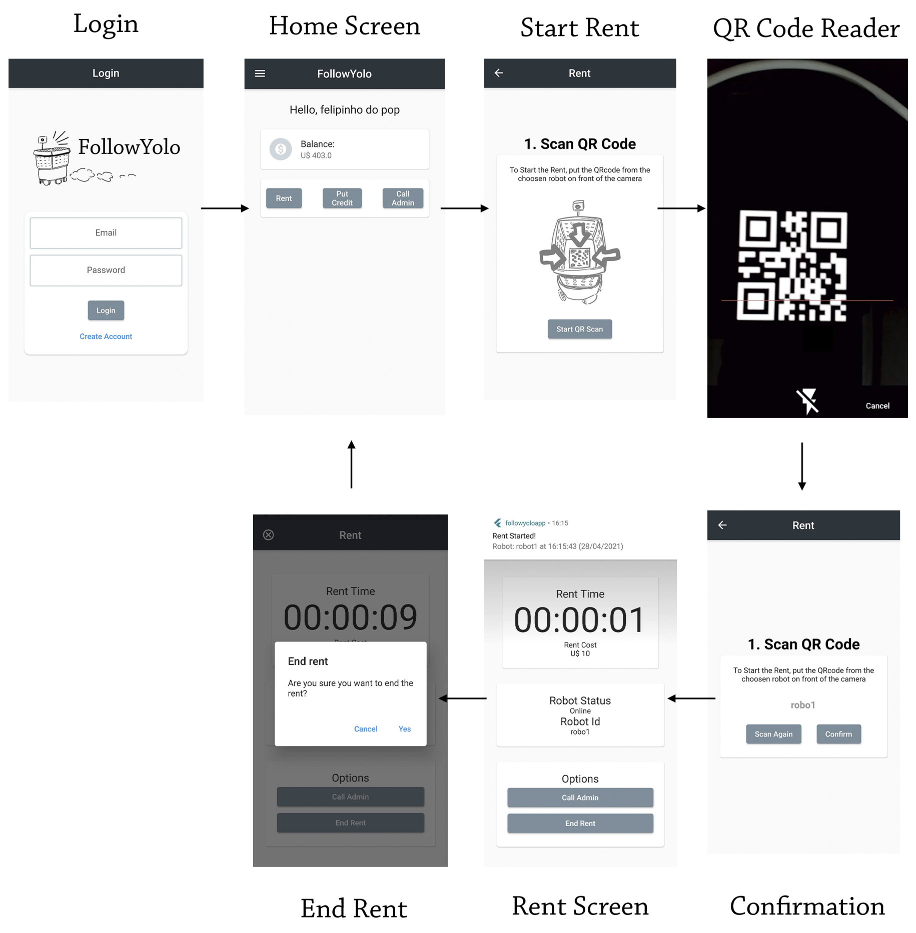


Figure 9: General use case diagram.

- User adds money to its account using the app
- User rents an available robot scanning a QR code using the app
- Person in simulation, representing the user, moves around the market (controlled by the team using a keyboard interface)
- The robot follows the person avoiding obstacles
- User ends the rent using the app



To Start the Rent, put the QRcode from the choosen robot on front of the camera

Are you sure you want to end the rent?

Figure 10: Renting process in the app.

The QR reading with the smartphone app triggers the start of the simulation. The simulation has four obstacles. Two of these obstacles are represented by a big box and a small box to test the capability of the LIDAR sensor. The other obstacles are a representation of people that usually are walking in the market. One of these is static and the other moves randomly throughout the environment. The user was controlled using keyboard commands to simulate the movement of a person. And FollowYolo follows the user throughout the market.

The navigation to specific spots in the market was also tested. In this navigation, without an ARTag as the objective, FollowYolo was able to avoid the obstacles.

The produced video shows some of the tests made as well as a quick explanation of the entire project. The code is available in the project's GitHub repository.

4 Difficulties

The main challenges related to the robot itself were:

- How to make the simulation itself. There wasn't a lot of updated documentation on the subject.
- The AR Tag tracking package (ar-track-alvar) is not well documented what made difficult to debug problems.
- The incompatibility of packages, ROS version and Ubuntu version. Mainly the ar-track-alvar package wasn't yet supported on the latest version of ROS what made the debug even more complicated. And the need for a specific version of a specific distribution of Linux also made the project more complicated when needed to change packages versions.
- Problems when sharing code between members. Almost always the code that worked on one member computer did not work for the rest of the team, despite the same environment.
- The complexity of autonomous navigation combined with target following was far greater than the expected.

While developing the server the main problems faced were:

- There were some problems when migrating from local host to the cloud, there were some differences that needed more attention, but after the transition was done, it was not an issue anymore.
- When the integration started with the app and later with the robot client, some issues with this integration were changed to adapt to the format that the client needed.

When developing the mobile app the main problems faced were:

- The major difficulty on the app development was the inexperience with the Flutter platform. Doing the project in a limited time and simultaneously learning and searching how to do the needed tasks was particularly challenging. Fortunately, there is plenty of material and documentation on the internet.
- Some flutter components change too fast, so the code that worked 5 months ago could not work now.

5 Results

Due the strict schedule, the app could not have all the planned features implemented, meaning, the secure communication with cloud server.

Since the project was simulated, the team had no costs making the hardware. The tables bellow show the estimated project's budget. Table 4 shows the total estimated cost for the construction of the FollowYolo hardware project.

Component	**Price**	**Comment**
Chassis	R$200.00	Value to print the model
Motor with encode	R$228.00	2 motors
Motor drive module with L298N	R$18.00	
MPU-6050	R$16.00	
Arduino Uno	R$34.00	
LiDar 360 sensor	R$664.00	Imported from China
Raspberry Pi 3	R$ 236.00	Imported from China
Raspberry Pi Camera v1	R$50.00	
Battery LiPo	R$281.00	Dolar cotation: R$5.64
Total:	**R$1,727.00**	

Table 4: Project budget.

About the robot, the team could not deliver a good target following, meaning, the robot wasn't able to follow the user very well mainly when doing curves. There were many complications with the AR Tag following, one of the main problems were the detection consistently failing, even when the person is static, sometimes the AR Tag was detected only when seen with an angle also the axis coordinate was big problem, for example the distance in the Z axis of the robot wasn't in the Z axis of the returned position of the ar-track-alvar package.

6 Conclusions

This project was very challenging for the whole team, mainly on the beginning when defining the project and in the end when integrating all the different parts, but it gave the team knowledge about new technologies, like ROS, Gazebo, Flutter and Heroku and also, just as important, knowledge about how to manage big projects and team work. These skills are very useful in a working environment since engineers always work in teams and needed to be able to communicate concisely and effectively, both requested during the project.

Another point that is worth mentioning is the need for documentation and planning, in particular the team faced problems with the schedule, that need a full rework 4 weeks after the start of the project, fortunately this was foreseen in the Risk Analysis, which was another document that showed its usefulness, and some requirements were dropped to keep the development of the project.

For the 6 weeks of the project development the team estimated a total of 360 hours. At the end, a grand total 423 hours of work were spent to develop the project. An excess of 63 hours were used, which composed about 15% of the total time spent. The actual executed time of the project exceeded the estimations by 17.5%.

Everything in this report and more can be found in the project's blog and in the GitHub repository. Also, a video was produced to summary all that was explained in this document and give an overview of the project.