Federal University of Technology – Paraná – UTFPR
Academic Department of Electronics – DAELN
Academic Department of Informatics – DAINF
Computer Engineering
Integration Workshop 3 (EEX23) – S71 – 2019/2

Technical Report

DLVR – A miniature driverless delivery system

Nicolas Abril – nicolasabril@alunos.utfpr.edu.br
Matheus G. Dias – matheusdias@alunos.utfpr.edu.br
Natan A. Junges – natanjunges@alunos.utfpr.edu.br
Álefe F. G. P. Dias – alefe@alunos.utfpr.edu.br

November 2019

Abstract

This document reports the creating of DLVR, a miniature driverless delivery system. It is a small and constrained version of an autonomous delivery system, that could potencially replace services like Uber Eats. The system is composed of a vehicle, that moves across a small model city, delivering objects as requested by the user using the system's mobile app. All the information about deliveries, users, routes and location is managed by a cloud server. This report details the development process as well as show the end result, comparing it to the initially proposed solution.

1 Introduction

Due to the increase of traffic in cities, and to the fact that people have less and less time, indisposed to leave their homes, delivery systems are becoming more popular by the day. Made by drivers that work hours without stop, this system usually creates problems, like a lack of employment contracts and poor regulation. Looking to solve these problems, investment in autonomous vehicles has increased and, although it has its own problems, it tends to be a safer and more efficient solution.

Inspired by some attempts of replacing human-made deliveries like Starship Technologies[1] and Amazon Scout[2] and by the structure of Duckietown[3], this project aims to create a solution to a small scale version of this problem. The solution proposed here consists of: A vehicle that carries the delivery and is able to navigate through the city given a route; A server, that manages the delivery from begining to end, acting as the system's central intelligence; A mobile app that provides the system's functionalities to the user; A model city, where the vehicle will drive on, with streets and delivery locations, constraining the problem to a simplified and controled environment. Figure 1 shows a diagram with the overview of the system's main components and the environment in which they interact.

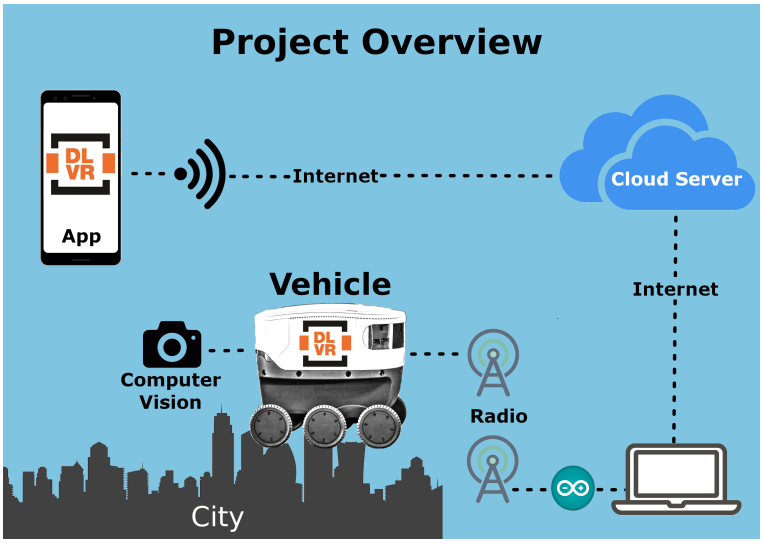


Figure 1: Overview of the system's architecture

To make a delivery, using the app, the user first logs into their account, selects the other user they want to deliver to, and provides their location. Then, the server confirms the delivery and location with the second user, who must also have the app, and selects a vehicle for the delivery. The server then calculates and sends the route the vehicle must follow to reach the first location (the origin). By navigating through the city, the vehicle moves to the first location, where the user will load the vehicle with the item to be delivered. The server confirms the arrival with the user, allows the item to be loaded, and sends the vehicle towards the second location (the destination). When the vehicle arrives, the server goes through the same procedure, confirming the arrival with the user, allowing the item to be removed, and finishing the delivery afterward. Finally, the vehicle is sent back to its garage, where it waits for the server to send it a new route.

The server acts as a central intelligence, deciding, authorizing, checking and logging each step of the system. The vehicle and the app are designed to be as simple as possible, leaving as many actions as they can to the server. This includes all knowledge about the city's layout and the route that the vehicle must take. To store all this information, a complementary database server was used. The server is hosted on the cloud and is accessible through the internet. For the vehicle to be able to access the internet, it must first communicate using a different protocol with an intermediate device which in turn sends the message to the server. This is due to the large amount of Wi-Fi signals around the university, producing interference that make it impractical to establish a stable Wi-Fi connection.

Lastly, the city has all the elements that the vehicle must be able to identify, placed in such a way to showcase all functionalities of the system. It has a

reduced size that is small enough to fit inside the classroom.

1.1 Requirements

The requirements for this project were subdivided into four categories: the Vehicle, the Server, the App and the City, each category then divided in functional and non-functional requirements. Table 1 contains the main requirements for each category.

2 Components and technologies used

This section details all the different technologies and components that were used in the system.

2.1 Mechanics

2.1.1 Chassis

For the vehicle chassis the 2WD chassis was used. It comes with an acryllic mount, with different holes to attach other components. It also comes with two 5V DC motors, two speed encoder disks, two wheels that can be attached to the motors, and one loose wheel that goes on the rear of the chassis, increasing stability.

2.1.2 Support structures

Support structures were made for the different components that needed to be placed in the chassis. These supports were handcrafted with MDF wood or 3D printed with ABS plastic.

2.1.3 Container

The container is made with a repurposed wooden box. A servomotor that latches to a small piece of wood was assembled on the side of the box acting as the lock. The item detection sensors are also attached to the sides of the box.

Figure 2 shows the modeling of the vehicle's mechanics.

2.1.4 City

The city was made with puzzle piece-shaped EVA mats arranged in a 2m x 2m square. Colored tape was added to represent the roads and cardboard makes the walls of the locations. Figure 3 shows the modeling of the city.

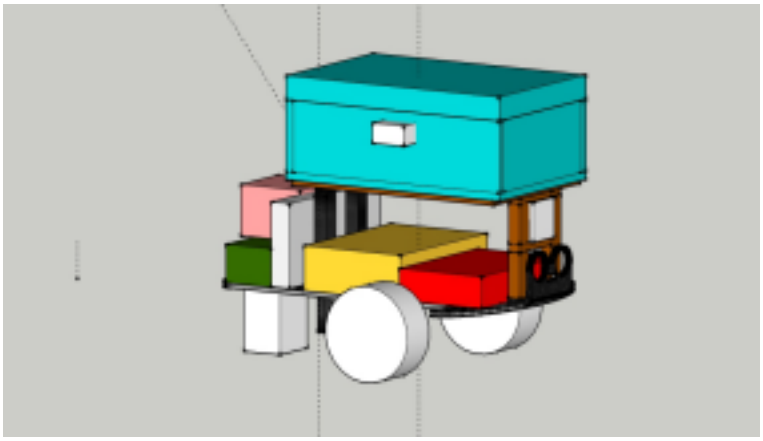


Figure 2: Vehicle Mechanics Model

2.2 Hardware

2.2.1 Raspberry Pi 3B

The vehicle's microcontroller is a Raspberry Pi 3B[4], a low cost single-board computer. It comes with a quad-core 1.2GHz processor, 1 GB of RAM, 40 GPIO pins, WiFi, a camera interface, and hardware SPI and PWM.

Two cameras were used, one looks downwards and the other looks straight ahead. The first one is the Raspberry Pi Camera Module v2, which interfaces Raspberry Pi using flat cable, allowing the usb ports to be free. The second camera is the Koup GZE877 PC Camera, a small webcam that one of the team members already had.

2.2.2 Main board

The vehicle has a main circuit board that connects all the electronic components and the Raspberry Pi. When possible, the components were integrated into this board, otherwise they connect to it with wires.

2.2.3 Optocouplers and H Bridge

Connecting the Raspberry Pi to the motors is a board with 4N25M optocouplers to isolate the motors from the digital systems. The signal from the optocouplers then goes to an LM298 H Bridge that controls the direction and speed of the motor.

2.2.4 Infrared sensor

Infrared sensors were used to detect if there is an item inside the container. They are made with one infrared LED and one infrared phototransistor. When an ob-

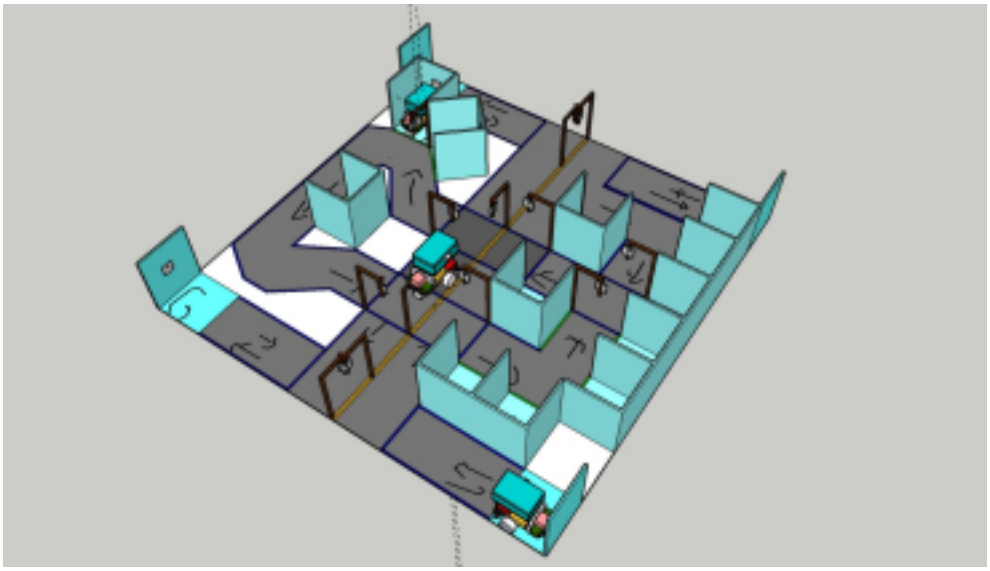


Figure 3: City modelling

ject obstructs the light, the voltage on the phototransistor drops.

2.2.5 HC-SR04

The HC-SR04 is an ultrasound sensor that detects objects from 3 to 400 centimeters of distance. It is used to detect obstacles and avoid colisions.

2.2.6 LM393 IR Module

This module has an LM393 comparator with a infrared LED and phototransistor pair. When coupled to a motor with a speed encoder disk, this module can be used as a speed encoder.

2.2.7 nRF24l01+

The nRF24l01+ is a wireless communication module that works in the 2.4GHz range. It is used as part of the communication between the vehicle and the server. It comunicates with the Raspberry Pi using an SPI interface.

2.2.8 Arduino Nano

Arduino is a single-board microcontroler with low cost and low processing power. An Arduino Nano is used as part of the communication between the vehicle and the server. It's purpose is to relay the data coming from the nRF24l01+ to its serial port. Another Arduino is used to control the traffic lights.

2.2.9 RGB LEDs

RGB LEDs are used as traffic lights.

2.2.10 Logic Level Shifter

The logic level shifter is a integrated circuit that allow communication between different logic voltage levels. It's used to interface the ultrasound (5V) with the Raspberry Pi (3V3).

2.3 Software

2.3.1 OpenCV

OpenCV[5] is a free and open-source library for computer vision. It is used to detect the visual elements and navigate the city.

2.3.2 ArUco

"ArUco is an OpenSource library for camera pose estimation using squared markers." [6] We use it's fiducial AR tags as unique markers for the city locations.[7][8] AR tags are used in imaging systems for augmented reality applications. They serve as points of reference or measure, allowing the imaging system to calculate the camera's position and orientation relative to physical markers.

2.3.3 AWS

The servers are hosted on the cloud using Amazon's AWS[9]. The system uses two servers, one for the database and one for the processing. AWS was chosen because it has a free trial and is simple to use.

2.3.4 WebSocket

WebSocket is a full-duplex layer 7 communication protocol that operates over a TCP connection. Using the Python library pywebsocket, it provides a simple and efficient way of exchanging asynchronous messages with the server.

2.3.5 Godot

Godot[10] is a free and open-source game engine. It was used to build the mobile app. Godot was chosen because it provides easy methods for building graphical interfaces and for showing graphics, and also because the team already has some experience with it.

3 Development

This section details the development of the project. Section 3.1 presents the mechanical design of the robot.

3.1 Mechanics

The development of the project started with the mechanical parts of the vehicle, as all the other physical parts depended on it. Overall, these parts were simple to model and create, with the main problem being fitting everything in the smallest space possible. Then the city was modeled and assembled taking into account the size of the vehicle.

3.1.1 Vehicle's Chassi and Support Structures

The vehicle's chassi was assembled to allow the placing of the support structures. Some of the support pieces were 3D printed, like the main container support and ultrasound's support, while the rest were created with MDF slabs, either 3mm or 5mm thick depending on the use. For the pieces that needed to be glued together, silicon glue and hot glue were used. Then, the ultrasonic sensor and the camera suport were attached with screws, while the others parts were attached with double sided tape.

One major change to the original model had to be made. Because a second camera had to be used and the first camera had to be raised to look from a more downwards angle, another supporting piece had to be made to accomodate for these two components.

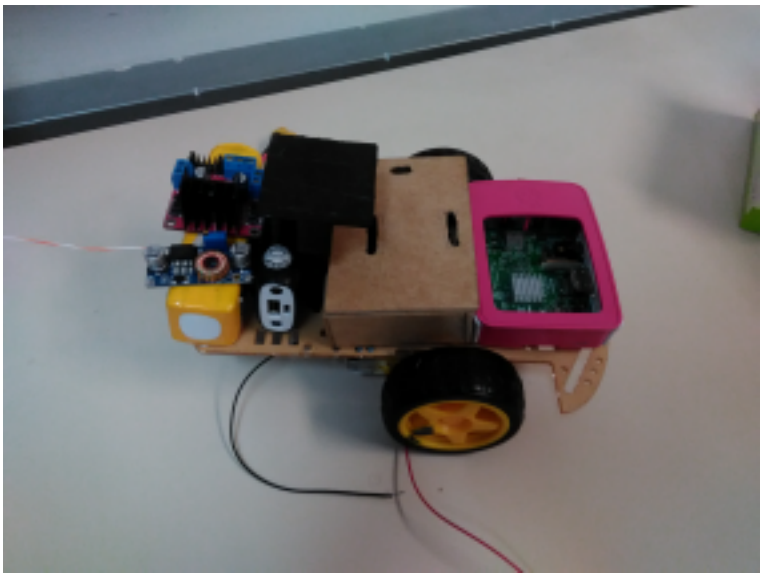


Figure 4: Assembled vehicle's chassi

3.1.2 Container

Some holes were made in the wooden box to put the detection sensors and the servo motor. A hook was attached to the servo's arm so that when the motor rotates it hooks to a small piece of wood that was put on the box's cover, locking it.

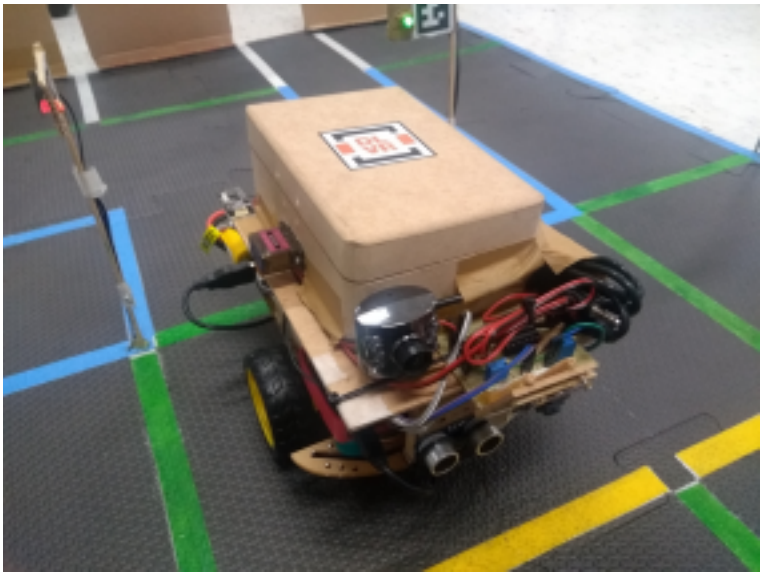


Figure 5: The finished vehicle

3.1.3 City

Using 16 pieces of 50x50cm EVA mats, the base of the city was layed down. Unfortunately, because the EVA mats overlap when they are connected, the city ended smaller than planned, with 1,85x1,85m, and because of this some locations had to be removed. Different colors were used to mean different things, blue means the edge of the road and so the vehicle is not allowed to go over it, green tape marks where it is possible to make a turn, and yellow represents the separation of two lanes, where the vehicle is only able to cross if it is coming perpendicularly.

To act as traffic lights and to simulate GPS, some poles were built containing an RGB LED and an AR marker. The LEDs are wired underneath the EVA mats and connect to an Arduino Nano, which controls their color, red or green.

The AR tags were also put in all the possible destinations of the vehicle, as a way to signal that it has reached its destination. These tags are held by a cardboard wall. Each location has in its back a piece of carboard and they also surround the city, isolating it from the background. Originally, each location would have carboard not only on its back but on the sides as well. However, it was too

difficult for the vehicle to turn inside them and they only served an aesthetic purpose and so were removed.

Figure 6: A picture of the finished city, with all the AR tags placed

3.2 Hardware

After the vehicle's mechanics were built, the hardware was constructed. Three boards were custom-made while the other components simply needed to be connected.

A board was made to binarize the signal from the infrared sensors. The main, infrared and optocoupler boards were made using perforated universal board. However, the final result is adequate. Because these boards concentrate the connections, they make it possible to change components without much effort, if needed.

To power up the hardware, two power sources were used. One, a 14.8V LiPo battery, goes through a voltage regulator reducing it to around 7V, and powers the DC motors and the H bridge. This battery is commonly used in robots for its reliability and high discharge rate. To power all the other hardware components, a 10000mAh 5V 2.4A power bank was used.

The last step to make the vehicle work was to develop its software.

3.2.1 Hardware-Interfacing Software

To test the hardware as it was being created, the software that drives the basic functionalities of the Hardware was developed in parallel. The first library chosen to control the GPIO functions was WiringPi, but it lacked hardware-based

timers which are necessary to control the motors. The library was thus changed to the aforementioned pigpio.

3.2.2 Movement Control

An essential part to this project is making sure the vehicle is able to move within the road. So, it was necessary to make a good control system, enabling the vehicle to move straight and do precise turns. To accomplish this, the encoder's response is filtered to remove the noise and then passed on to a proportional controller. The proportional controller was projected using the Tustin method, converting a continuous-time controller in the Laplace domain to a discrete-time controller in the Z domain through the bilinear transform. Because the motor's response is non-linear it was hard to properly adjust the controller and make the vehicle both move straight and do turns, but after some tinkering it was done. The proportional constant of the controller is Kp = 0.001.

3.2.3 Computer Vision

To be able to move within the roads, the vehicle first needs to know where the roads and the crossings are. It also needs to respect the traffic rules and not hit anything. To handle all these requirements a computer vision system capable of detecting all those requirements was developed. It has three main components, the traffic light detector, the AR marker detector, and the streets and crossings finder.

Detecting traffic lights is very simple, being accomplished by a simple color mask and then analyzing the found blobs' position and size. The AR markers are detected using ArUco in just one line of code. At first, QR codes were going to be used, but after some testing they were found to be very hard to detect at a distance. Both these parts use the camera that looks ahead. This camera was not originally going to be used, but it was very hard to look both at the street signs and at nearby streets at the same time, so a USB WebCam was added.

Finding where the streets and the crossings are was a lot harder. The approach used was to first find where the tape is and then convert that to a geometry problem by calculating the real world position of the tape using projection mathematics. Then, knowing these positions, the vehicle calculates where the center of the streets should be. By intersecting and joining these streets, the vehicle then creates a graph which is passed onwards to the robot's central intelligence.

Before deciding on this solution, many other approaches were tested, like detecting corners and analyzing image slices. All the alternatives were not adequate for this problem, which differed from others on constraints like scale and processing power. This solution had inspiration from some other solutions but is overall custom-made for this problem.

3.2.4 Vehicle's Intelligence

All the other software parts work separately, but they need to be tied together and respond properly to the server's orders. This was done in a module named the 'intelligence' of the vehicle. This module coordinates all the rest of the software and defines the main loop of the vehicle. The program works in a cycle of receiving a message from the server, acting on it when necessary, and then doing one vision and movement cycle, finishing with an update of the vehicle's status to the server.

The intelligence is also responsible for deciding how to move, using the information from the vision, the movement control and the sensors to try to fulfill the server's orders. It decides if the vehicle should move and where it is going, and then acts on the decision by sending the new movement to the control module.

3.3 Server

The server is the backbone of the entire system, serving as a communication bridge between the app and the vehicles, and coordinating all the delivery process.

The server was developed using the Python programming language, due to its ease to handle asynchronous processes, the amount of documentation available, the knowledge base of the team and its popularity. The server API is very similar to the RESTful API standard, using URIs to represent each entity of the system. But it is different as it also represents in the URIs the actions performed on the entities (something that resembles the object-oriented paradigm).

The database used was MySQL, a traditional relational database management system, that uses SQL for the queries. It was chosen due to its ease of use, reliability and popularity. The relational database model was chosen because, as the system and the information that it manages are not too complex, and the volume of data is not big, it is the simplest model viable.

3.4 App

The application is the main interaction tool between the system (server + robot) and the user. Through the application the user can signup in the system, request and accept deliveries and check the status of the delivery robot and the cargo.

The application was developed using Godot Game Engine version 3.1, using its native GDScript programming language, a language with syntax very similar to the Python language. Godot has a very intuitive interface for graphic development which allowed the application to develop much faster than Android Studio. Godot's ability to export applications to various platforms (such as HTML5 and PC) allowed for easy application testing and debugging.

3.5 Communication

The system has three parts that need to communicate with each other: the app, the server and the vehicle.

The server is only accessible through the internet, so our first idea was to have the vehicle connect with WiFi directly with the server. However, because the university WiFi is hard to work with, specially on embedded devices, an intermediate device that relays the data between the vehicle and the server was used. This computer communicates with the vehicle using an Arduino with an nRF24l01+ module and with the server through the internet. The app connects to the internet using whichever means the user chooses, be it WiFi, 4G or something else.

To communicate with the server, after some failed attempts at using plain HTTP, the WebSocket protocol was chosen for its simplicity when establishing bidirectional asynchronous communications. It is used both in the intermediate device and in the mobile app.

4 Problems

Along the development of the project, some problems were faced.

On the city, the tapes that were first used did not stick well to the EVA mats and reflected the ambient light directly in to the camera. The blue one was changed to blue masking tape and the green to hand-painted masking tape. The yellow tape worked better so it was still used.

On the hardware, the boards ended very dense with connectors and adding that to the methods chosen to build them and some initial design mistakes, a lot of time was spent building and fixing them.

5 Results

The final result of the project was not, unfortunately, able to fulfill all the planned requirements. The obstacle recognition and avoidance had to be dropped. The main difficulties found were a gross misjudging of how long it would take to do what we didn't know how to do, a lack of efficient communication, and a stubborness to change the original plan. Another problem that negatively affected the work done was poor collective time management. It often happened that a task would be delayed and the person that was going to do the next one would get lost because they depended on work that was not yet completed.

Nonetheless, all the functionalities that the team decided were the vital parts of the project were successfully implemented. Table 2 shows the initially time it would take to complete the project compared to the actual time taken. The activities were divided in 3 blocks that corresponded with the time they were expected to be done, which of course differed from when they were actually ex-

ecuted. In this table, it is visible that the second checkpoint took a lot longer than expected, and that caused all the activities from the third checkpoint to be delayed, and some abandoned altogether. The total time spent was very close to the the error margin because the project was dimensioned to, at worst, take almost all of the team's free time. Each checkpoint corresponded to around 3 weeks of work, for a team of four, during the 3 months of project development.

For almost every part developed on a task, a test was built to ensure that it was working reasonably well. Having a known point where a component was working helped a lot to find any errors, as there was something to compare. Figure 11 shows four pictures of parts of the vehicle's vision, taken as part of the vision's tests. Finally, Figure 12 shows some screens of the mobile app.

Regarding the money spent building the project, the final cost was very similar to the initially estimated cost. While about 15 items were not thought of in the starting budget, the cost of other things was lower than expected, and some things were replaced for a cheaper alternative. And considering that the team had already had or could borrow most items, the actual cost of the project was reasonably low, at only R$307.63 .

6 Conclusions

This project was able to complete its main objective of creating a driverless delivery system. Although some of the more specific objectives weren't achieved, like avoiding obstacles, the essential parts were.

This was a great opportunity for the team to learn about working on bigger projects, with problems that are not usually encountered in other courses, like managing a team, working on a strict time limit, and having to plan and schedule activities for a longer period of time. In this project, most of the knowledge taught in the Computer Engineering course was used, with subjects ranging from electronics, to computer vision and databases.

Aside from finishing all the proposed functionalities, some future projects are possible after this one. The first idea would be to work on a bigger scale, with a larger city, larger robots and more varied environment. Another possible branching would be to bring the vehicle to the real world, focusing on movement through real environments, both indoors and outdoors. In this project, the server is responsible for most of the processing; Another rendition of this project could be focused on having a more intelligent vehicle. The ultimate goal would be to build a system like Amazon Scout and Starship Deliveries.

Overall, making this project helped the team to mature a lot and develop many different abilities, in a way that would be very hard without working on something like this project.

The details of the development of the project, as well as a video of its final result, can be found at <deliveryrobot.wordpress.com> and <youtu.be/wU_OaUOvt88>.

Acknowledgement

We would like to thank our professors, João Alberto Fabro and Heitor Silvério Lopes, for all the help and guidance provided, and professor Bogdan T. Nassu for his help with the computer vision, and everyone at LASER (Advanced Lab on Embedded Systems and Robotics - laser.dainf.ct.utfpr.edu.br) for their help and support.

References

[1] Starship Technologies. `https://www.starship.xyz/`.

[2] Amazon. Meet scout. https://blog.aboutamazon.com/transportation/meet-scout.

[3] Duckietown. `https://www.duckietown.org/`.

[4] Raspberry Pi Foundation. Raspberry pi 3 model b. https://www.raspberrypi.org/products/raspberry-pi-3-model-b/.

[5] OpenCV. `https://opencv.org/`.

[6] A.V.A Universidad de Córdoba. Aruco. `https://www.uco.es/investiga/grupos/ava/node/26`.

[7] Sergio Garrido-Jurado, Rafael Muñoz-Salinas, Francisco Madrid-Cuevas, and Rafael Medina-Carnicer. Generation of fiducial marker dictionaries using mixed integer linear programming. *Pattern Recognition*, 51, 10 2015.

[8] Francisco Romero-Ramirez, Rafael Muñoz-Salinas, and Rafael Medina-Carnicer. Speeded up detection of squared fiducial markers. *Image and Vision Computing*, 76, 06 2018.

[9] Amazon. Amazon web services (aws). https://aws.amazon.com/.

[10] Godot Engine. `https://godotengine.org/`.

Table 1: The main requirements of the system

Server – Functional Requirements	
FR01	Allow users, vehicles and deliveries to be registered
FR02	Manage all steps of the delivery, updating the app of its status
FR03	Choose a vehicle to perform the delivery, sending it the routes
FR04	Send the vehicle to the garage if it has no deliveries to make
Vehicle – Functional Requirements	
FR05	Be able to move on the streets, but only along the route given by server
FR06	Be able to identify traffic lights, obstacles and location markers
FR07	Inform the server about it's status and the delivery's status
Vehicle – Non-Functional Requirements	
NFR01	Detection of the city's elements must be done with computer vision
NFR02	Be able to carry a load of 200g, no larger than a common cellphone
NFR03	Due to environmental limitations, the vehicle-server communication must not use WiFi
App – Functional Requirements	
FR08	User sign up
FR09	User's email confirmation
FR010	Password reset
FR011	Sign in
FR012	Delivery request, specifying sender's location and receiver's email
FR013	Delivery response, specifying receiver's location
FR014	Vehicle position and status tracking
FR015	Close and send vehicle to destination
FR016	Open vehicle's container to retrieve the delivery
FR017	Close the vehicle's container and finish the delivery
FR018	Sign out
FR019	Profile update
FR020	Account deletion
App – Non-Functional Requirements	
NFR04	The app must run on Android
NFR05	Arrival confirmation must be done by showing the screen to the vehicle
City – Non-Functional Requirements	
NFR06	Have Streets, blocks, traffic lights, garages and delivery locations
NFR07	Have markers that identify locations within the city
NFR08	Have obstacles that partially or completely obstruct some roads

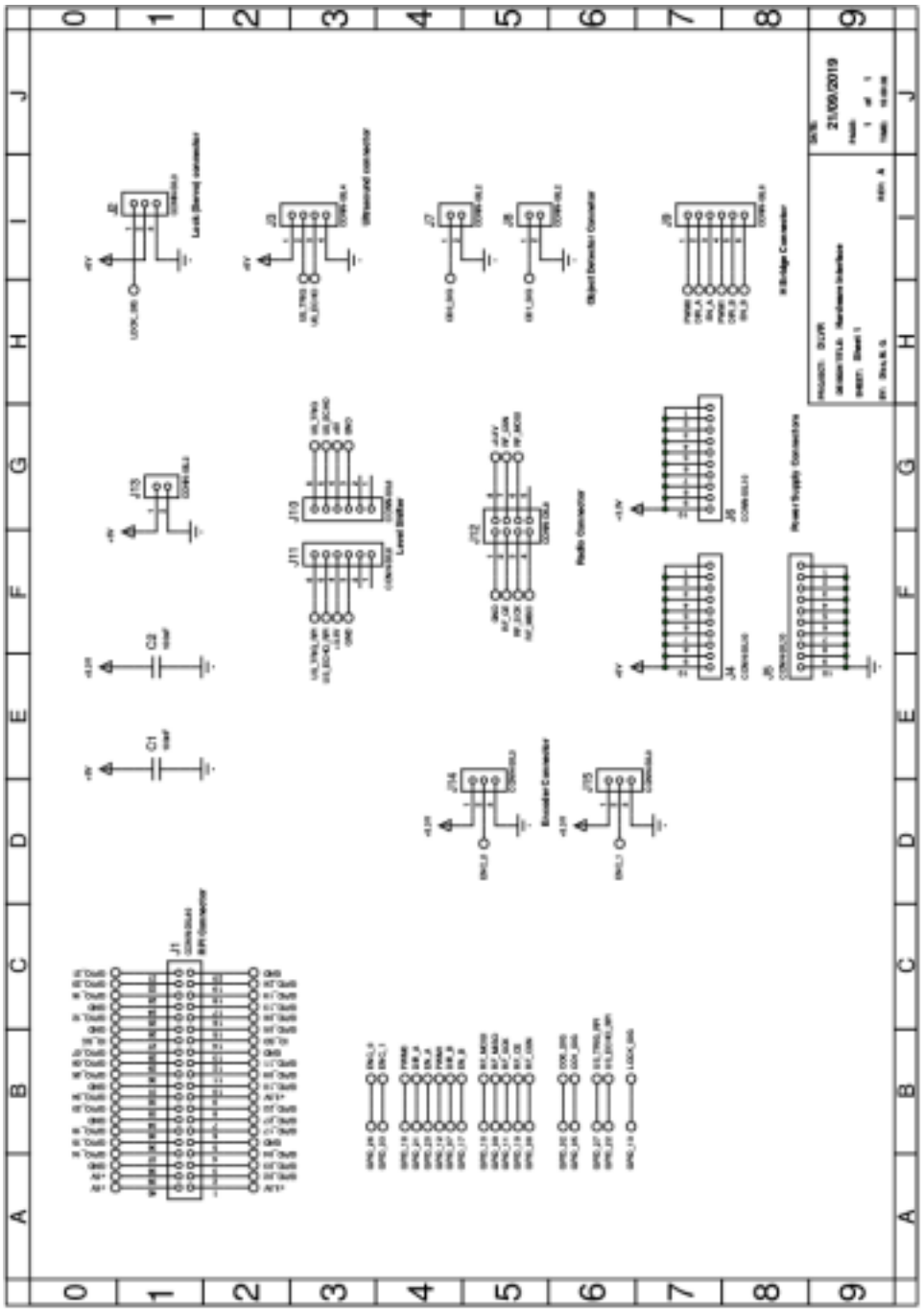


Figure 7: Main board schematics

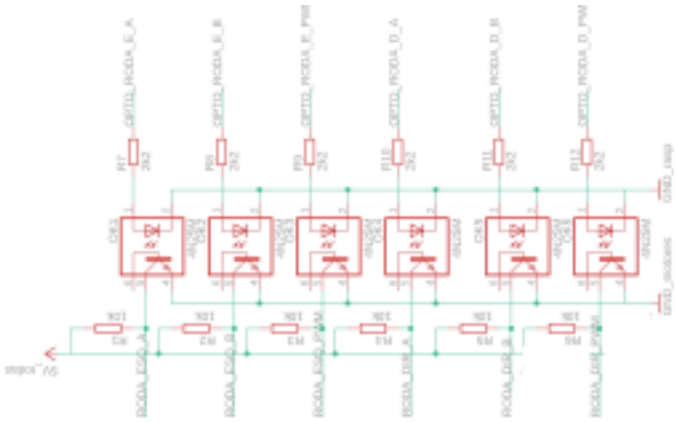


Figure 8: Schematics of (a) optocoupler board, and (b) IR sensor

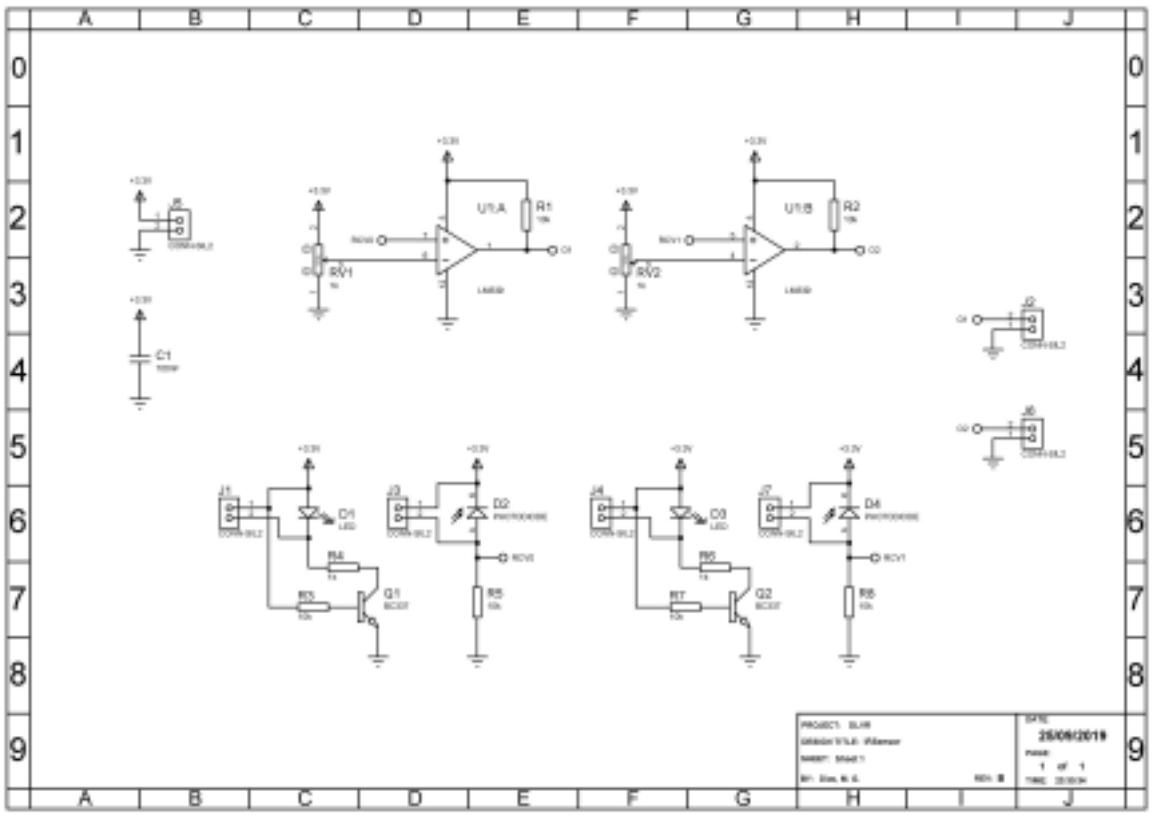


Figure 9: Schematics of (a) optocoupler board, and (b) IR sensor

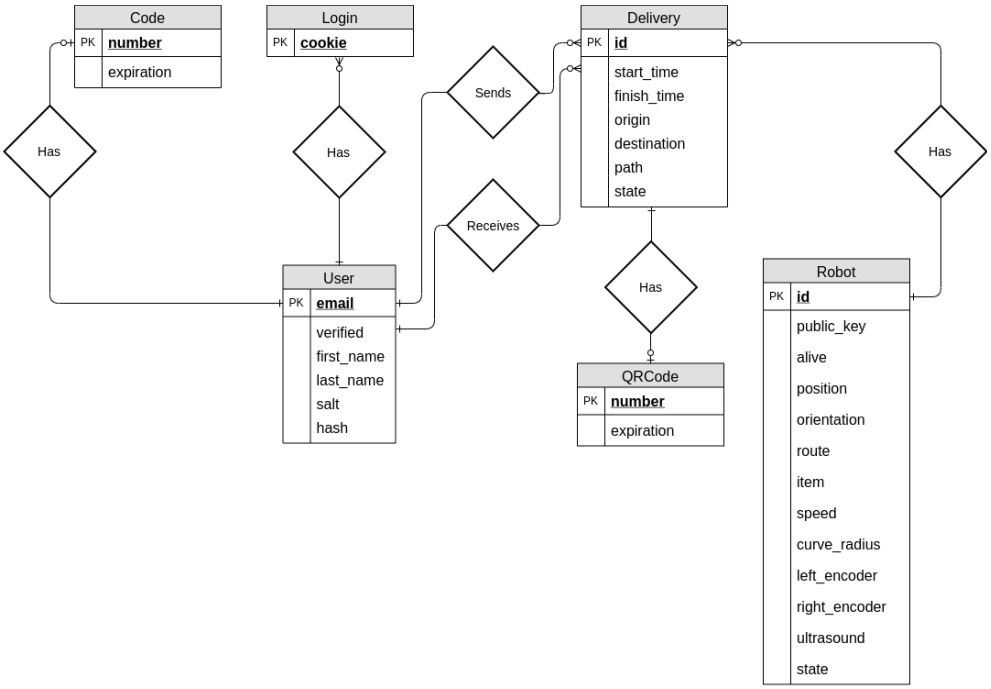
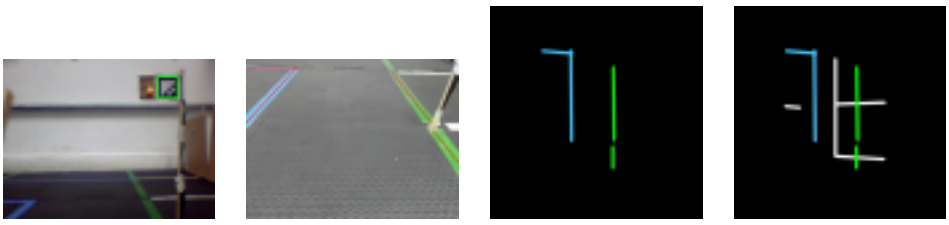


Figure 10: Entity-relationship diagram of the database

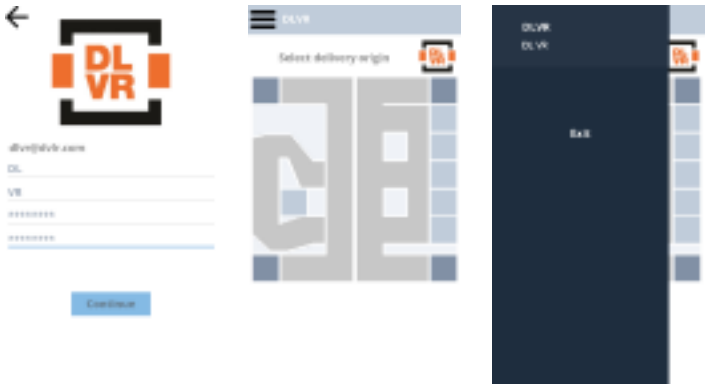
Table 2: The estimated time to complete the project and the actual time taken

Activities	Estimated	Estimated + 30% Error Margin	Time taken
First Checkpoint	104 hours	135.2 hours	134.28 hours
Second Checkpoint	161 hours	209.3 hours	302.68 hours
Third Checkpoint	180 hours	234 hours	87.75 hours
Total	465 hours	604.5 hours	602.01 hours



(a) (b) (c) (d)

Figure 11: Pictures of the vehicle's vision (a) What the frontal camera sees with Ar marker and traffic lights identification, (b) What the downwards camera sees, with the found tapes, (c) The found tapes with undistortion and in the correct place in real world, and (d) The final graph representing the streets.



(a) (b) (c)

Figure 12: Some of the app's screen (a) Account creation screen (b) Main screen, (c) Side bar

Table 3: The estimated and the actual cost of the project

	Vehicle	City	Total
Estimated cost buying all parts	R$715.19	R$177.00	R$892.19
Estimated cost with pre-owned parts	R$106.50	R$147.00	R$253.50
Actual cost buying all parts	R$764.04	R$202.28	R$966.32
Actual cost with pre-owned parts	R$140.35	R$167.28	R$307.63