Mimic Alphabots

Adriano Soares
up201904873@edu.fe.up.pt
Contribution: 33.3%

Eunice Amorim
up201904920@edu.fe.up.pt
Contribution: 33.3%

Joel Fernandes
up201904977@edu.fe.up.pt
Contribution: 33.3%

***Abstract*—This report documents the approach taken to develop two car robots where one of which is able to mimic the behaviours of the other, such as following and copying its orientation, in real time.**

***Index Terms*—Alphabot, follow, real time, monotonic rate**

I. Introduction

Over the years, humans controlled machines and devices. However, it's greatly common to see machines controlling other machines. One of those cases could be a robotic car that autonomously follows another one operated by an human.

This report describes the procedures taken to build such system. It is organized according to the steps taken to build them, namely: Specification, Hardware Architecture, Software Architecture and Results.

II. Specification

The main objective was develop two real time car systems able to interact with each other. For that reason, the following sections will define requisites that are common to both cars, and some that are specific to each car.

A. Common Requisites

One feature that was applied in both cars was the ability of display some information on the car screen. The data present is comprised by the relative power in each motor, the current angle (in degrees) of the car relatively to an initial position set during the calibration process, and the distance between the car and the nearest object right in front of it (in centimeters). Despite it not having a direct influence on the movement of the cars, that information was crucial during the development and debugging processes.

Another important aspect is the stability on the movement of the car. As the car is not perfectly balanced and the floor is not perfectly plane, if the engines did not correct their speed accordingly, the car wouldn't be able to follow a straight line. Measures were taken to make this corrections (see next sections).

B. Manually Operated Bot Requisites

The car that is manually operated needs to deal with the commands given by an infrared controller to perform the movement. On the other hand, it must assure the communication to the other car. As it leads the movement, it must send its current turning angle so that the follower may act accordingly.

C. Autonomous Bot Requisites

The autonomous car needs to obey to the behaviour presented by the manually controlled car. For that reason, it is crucial that it is able to communicate with it, receiving the information about the angle the controlled car has. It needs to be able to adjust according to it during each of its two modes: while following the other car, or just mimicking the angle of the other car, which serves as a steering wheel. If it's mimicking the other car, it needs to estimate the speed taken by checking the distance between it and the other car. If the front car stops, it shall do the same conserving a predefined safe distance of 20 centimeters, although with some tolerance.

D. Setup

Both cars have a period at the beginning of the operation to set their base relative angles. This process is done by placing them one after the other separated by some distance pointing to the same direction.

III. Hardware architecture

In order to develop the two systems, the hardware necessary is described in the next section.

A. Moving platform

It was necessary to find a proper car/moving platform to recreate the specified behaviour. The choice fell on an Alphabot2-ar from Waveshare [1]. It is a robotic car that already has integrated multiple sensors that aided the development, namely the ultrasonic sensor used to calculate the distance between the two alphabots and the infra red receiver used for the communication.

B. Comunication

Initially, the communication was thought to be implemented with the module kit emitter receiver RF 433MHz TEL0140-1 from DFRobot [2]. However, this approach showed to not be effective, as it lead to many errors and a slow transmission speed making impossible to the cars to be kept synchronized (for more details, please check section V-F).

For that reason, it was used an infra red emitter TSAL6200 from Seeed [3] instead. It allowed for a faster and more reliable communication, allowing us to achieve our objectives. Furthermore, we choose an infra red emitter as we already had an infra red receiver coupled on the other alphabot.

C. Processing Unit

Considering the processing unit for the code, it was chosen the Arduino Micro [4]. It proved to be the better trade off between performance and cost. It was also considered to use other types of Arduino recommended for the alphabot, however, this would occupy all the pins leaving no place for any additional hardware not included in the alphabot (the IR emitter, for instance). For that reason, the choice was not altered and the whole system was mounted in a breadboard attached on top of the alphabots.

D. Movement measures

To achieve the intended result, some angle measures need to be made, namely the yaw of the car. As a simple accelerometer wouldn't be enough to perform such measure, it was chosen the module LSM9DS1 from Adafruit that comprises an accelerometer, a gyroscope, a gravitometer and a temperature sensor [5]. This module was particularly important not only for the yaw but also for the stability of the alphabot, by applying small adjustments during its movement, making it able to follow a straight line even if the floor is not perfectly plane or the Alphabot engines are not balanced.

IV. Software architecture

Having all of the hardware available, it was time to structure the software to perform as needed.

A. Division between three programs

Since there wasn't enough memory to load the code of the three specifications, them being the mimicking car with RF communications, the mimicking car with IR communications and the steering wheel version, it was decided to split them into three different source files. For more details about memory management, please check section V-E. The RF version was decided to be kept for documentation purposes. They can be seen in the folders *comms-ir*, *comms-radio* and *steering-wheel*, respectively.

B. Tasks

1) Tasks Description: In order to meet the requisites, many tasks were considered. Their presence in each system is documented in the section IV-B3.

- *decodeIR* - Responsible to decode a message received by IR.
- *translateCommands* - Responsible to apply the command received (either from the remote control or communication) into the motors' state.
- *move* - Writes the state of the motors to the motors themselves.
- *displayData* - Responsible to display the information on the alphabot screen.
- *commsIR (IR comms)* - Responsible to send the angle of the manually controlled car using IR.
- *updateAngle* - Corrects the angle of the alphabot according to the gyroscope information
- *getDistance* - Gets the distance of the autonomous alphabot to the closest obstacle.
- *processAngleCarFront* - Calculations according to the angle and the distance to the front car to turn accordingly.
- *sendByte (RF comms)* - Responsible to send the angle of the manually controlled car using RF.

2) Tasks Deadlines and Periods: The periods and deadlines of each task were defined accordingly to the capabilities and maximum throughput of each hardware device used. The deadline and period have the same value as it's not critical for a task to be executed right after it was triggered. Their values can be found on Tab. I.

3) Tasks Priorities and Presence in each system: The priority of each task is defined accordingly to its importance for the systems and were tested to check if they would work. The best solutions found were (greater importance first):

- Follower specification with IR communications (similar to RF communications):
  1) Manually controlled car: *decodeIR, translateCommands, move, displayData, updateAngle, commsIR/sendByte*
  2) Autonomous car: *getDistance, decodeIR/None, translateCommands, move, displayData, updateAngle, processAngleCarFront*
- Steering specification:
  1) Manually controlled car: *displayData, commsIR, updateAngle*
  2) Autonomous car: *getDistance, decodeIR, translateCommands, move, displayData, updateAngle, processAngleCarFront*

C. Scheduler

To schedule the previously mentioned tasks, we developed a custom online scheduler with fixed priorities according to the importance of each task based on the example given in class. Ideally, we would have implemented a *Earliest Deadline First* scheduler. However, as the main focus of the project was the alphabots and their behaviour and considering that this one solved the problem, it was maintained.

V. Results

A. Worst case execution time

The Tab. II and Tab. III illustrate the worst case execution times measured for each task independently.

The values seem to be according to what was expected. We should only note that the disparity between the values from the Manual Operated Bot and Autonomous Bot *displayData* task is justifiable, as the Autonomous Bot has to display more data, such as the distance measured.

B. Schedulability analysis

According to the worst case execution time, it is possible to check if the set of tasks is schedulable or not, by assessing that

for each task its worst case response time is lower or equal than its deadline:

$$\forall_i R_{wc_i} = I_i + C_i \leq D_i$$

The calculations were performed for each car, but only in the IR communications setting, as it was the best solution we've achieved. Since the tasks from the steering specification are a subset of the follower specification tasks, if the ones from the latter are schedulable the first shall also be.

C. Schedulability analysis for the manually operated car

After performing the needed calculations, the worst case response times for each task of the Manually Operated Bot can be seen in Tab. IV.

Since every task has its Rwc lower than the corresponding deadline we can conclude that the set of tasks for the Manually Operated Bot are schedulable.

D. Schedulability analysis for the autonomous car

After performing the needed calculations, the worst case response times for each task of the Autonomous Bot can be seen in Tab. V.

As every task has its Rwc lower than the corresponding deadline we can conclude that the set of tasks for the Autonomous Bot are schedulable.

E. Memory usage analysis

Some measures about memory usage were taken. This was particularly important as the two versions code could not be load simultaneously to the Arduino.

1) The follower specification: When the program that makes the autonomous alphabot follow the other runs, the memory usage is different in the two cars. In the autonomous car, the Arduino program storage space is used at 87% capacity (25186 out of 28672 bytes) while the global variables use 84% of the dynamic memory (2160 out of 2560 bytes). In the manually controlled car, the Arduino program storage space is used at 85% capacity (24542 out of 28672 bytes) while the global variables use 84% of the dynamic memory (2168 out of 2560 bytes).

2) The steering specification: When the program that makes the autonomous alphabot steer in the same angle as the manually controlled one, the memory usage is also different in the two cars. In the autonomous car, the Arduino program storage space is used at 87% capacity (24406 out of 28672 bytes) while the global variables use 84% of the dynamic memory (2160 out of 2560 bytes). In the manually controlled car, the Arduino program storage space is used at 85% capacity (22038 out of 28672 bytes) while the global variables use 84% of the dynamic memory (2138 out of 2560 bytes).

3) Consideration about memory management: In both cases, the controlled car uses less storage space. This is expected as the autonomous car needs to perform more calculations to find its path and, as such, more code needs to be implemented. The dynamic memory is similar between the two cars as they are used to save the state of the car which is similar in both cars.

Between the follower and the steering specifications, the memory usage is lower in the steering version. This is also expected as the steering is, in essence, a subset of the capabilities of the follower version.

F. Obstacles found

The emitter/receiver RF 433MHz TEL0140-1 combo from DFRobot was proven not to be effective for the communication between the cars, as it, by design, was meant to be a trigger button used, for instance, as a car garage door opener. However, that didn't stop us from trying to code a protocol capable of sending at least a byte of information using binary signals. We've implemented a state machine with the following logic: during no communication the emitter sends a HIGH signal; when the communication starts it sends a LOW signal for a period; the receiver, with interrupts enabled for when a FALLING action happens, is waiting for the signal to go LOW; when it happens it waits 1.5 periods to start reading 8 bits, one bit each subsequent period; each packet is coupled with a checksum bit at the end (XOR of all bits sent) followed by some bits sent as HIGH (we set it up as 4 bits) so that the receiver is be able to distinguish and better separate packets. The protocol works, but it showed some flaws that rendered it useless for our project: the transmitter would sometimes take a lot of time synchronizing with the emitter and, during that time, no packets would be received correctly. For instance, if the manual operated car angle deviated a lot during that time period the autonomous car would simply follow a straight line. Therefore, we choose to use an infra red emitter instead as it proved to be more reliable. The protocol code can be further analysed, as it is present in the *comms-radio* folder in the repository.

VI. Conclusion

With this project is was possible to develop a system comprised of two cars where one can mimic/follow the other in real time, by measuring the distance between the front and back car and communicating the front car's angle to the back car through IR communication. Due to the meticulous planning and thorough execution that characterized our approach, the project's execution ran smoothly, despite the RF communications issues. In conclusion, the main objectives were accomplished, although with some limitations, and the team was able to learn not only about embedded systems, such as the Arduino, but also about handling RF/IR communications, handling data from accelerometers and gyroscopes and how to perform task scheduling through the simple micro-kernel developed. All of the work reported can be further analysed through the GitHub repository[1] and the video[2] publicly available.

[1] https://github.com/eunicejamorim/FEUP-SETR-2223
[2] https://youtu.be/FEbdW4j6IvM

References

[1] Waveshare, "Alphabot2-ar," 2023, accessed on 05/06/2023. [Online]. Available: https://www.waveshare.com/wiki/AlphaBot2-Ar

[2] DFRobot, "Gravity: Digital wireless switch kit - transmit and receive (433mhz)," 2023, accessed on 05/06/2023. [Online]. Available: https://www.dfrobot.com/product-2434.html

[3] S. Studio, "Grove - infrared emitter," 2023, accessed on 05/06/2023. [Online]. Available: https://wiki.seeedstudio.com/Grove-Infrared_Emitter/

[4] Arduino, "Arduino micro," 2023, accessed on 05/06/2023. [Online]. Available: https://store.arduino.cc/products/arduino-micro

[5] Adafruit, "Adafruit lsm9ds1 accelerometer + gyro + magnetometer 9-dof breakout," 2023, accessed on 05/06/2023. [Online]. Available: https://learn.adafruit.com/adafruit-lsm9ds1-accelerometer-plus-gyro-plus-magnetometer-9-dof-breakout/overview

VII. Annexes

TABLE I
Task Specifications

task	deadline (ms)	period (ms)
decodeIR	10	10
translateCommands	50	50
move	50	50
displayData	500	500
commsIR (IR comms)	200	200
updateAngle	10	10
getDistance	100	100
processAngleCarFront	50	50
sendByte (RF comms)	10	10

TABLE II
Manually Operated Bot Tasks

task	WCET (ms)
decodeIR	0.23
translateCommands	0.06
move	0.07
displayData	55.10
updateAngle	1.20
sendByte (RF emitter)	0.06
commsIR (IR emitter)	60.08

TABLE III
Autonomous Bot Tasks

task	WCET (ms)
getDistance	58.78
translateCommands	0.09
move	0.06
displayData	61.76
updateAngle	1.21
processAngleCarFront	0.60
decodeIR (IR receiver)	0.30

TABLE IV
Manually Operated Bot Response Time Analysis

task	Rwc (ms)	deadline (ms)
decodeIR	0.4927	10
translateCommands	0.1348	50
move	0.1572	50
displayData	99.4585	500
updateAngle	2.1284	10
commsIR	80.7310	200

TABLE V
Autonomous Bot Response Time Analysis

task	Rwc (ms)	deadline (ms)
getDistance	82.7328	100
translateCommands	0.7230	50
move	0.4843	50
displayData	250.8530	500
updateAngle	4.9655	10
processAngleCarFront	4.4550	50
decodeIR	1.9649	10