TUHH
Hamburg University of Technology

ComNets
Institute of Communication Networks

Master's Project Work

Design, Implementation, and Evaluation of an Automotive Experimental Platform for Autonomous Cooperative Adaptive Cruise Control (CACC)

Ahmed Shaheen

July 21, 2021

Design, Implementation, and Evaluation of an Automotive Experimental Platform for Autonomous Cooperative Adaptive Cruise Control (CACC)

Ahmed Shaheen
Matriculation number 54926
Mechatronics M.Sc.

Hamburg University of Technology
Institute of Communication Networks
First examiner: Prof. Dr.-Ing. Timm-Giel

Supervisor: Daniel Plöger

Hamburg, July 21, 2021

Declaration of Originality

I hereby declare that the work in this thesis was composed and originated by myself and has not been submitted for another degree or diploma at any university or other institute of tertiary education.

I certify that all information sources and literature used are indicated in the text and a list of references is given in the bibliography.

Hamburg, July 21, 2021

Ahmed Shaheen

Abstract

The aim of this project work is to develop an optimized research platform specially designed for testing autonomous vehicles platooning techniques such as Adaptive Cruise Control (ACC) or Cooperative Adaptive Cruise Control (CACC) with different control methods and communication schemes. It is known that this application is a safety-critical application in which the hardware and software parts play a major role, therefore, experimental effort and trials are required to prove which method suits than the other, which topology and communication scheme are more efficient and deterministic, which software algorithms are real-time, which control parameters should be considered and which method to finally chose and verify. To answer those questions, the work starts by defining requirements to achieve stable platooning, then selecting the hardware components as well as software functionalities in order to align with those requirements.

After testing the developed model, it was successful in matching the pre-defined model requirements in terms of sensor measurements, communication delay and packet loss, real timeliness of the written software code, remote control, and data logging. With this model, one can easily try different control methods and communication topologies/protocols without worrying about the capability of the system, yet there is still a big room for improvement in the hardware and motion estimation algorithms to finally reach a more optimized model. Building such a model requires knowledge about Sensors, Microcontrollers, Soldering, Embedded software programming, Software testing, and Internet of Things (IoT).

Contents

List of ACRONYMS	
ACC	Adaptive Cruise Control
CACC	Cooperative Adaptive Cruise Control
MAC	Media Access Control
L2CAP	Logical link control & adaptation control
SMP	Security manager
ATT	Attribute protocol
GATT	Generic attribute protocol
SoC	System on Chip
CPI	Count Per Inch
IPS	Inch Per Second
PWM	Pulse Width Modulation
OTA	Over The Air software update
SDK	Software Development Kit
CAM	Cooperative Awareness Message
MQTT	Message Queuing Telemetry Transport

Table 0.1.: List of Acronyms.

1. Introduction

1.1. Motivation

Every research topic serves in a specific area whether to solve a problem or provide an optimized solution, the provided solution then passes through a set of verification and validation steps, as we can not just rely only on assumptions and theoretical methods.

As for autonomous platooning[2.1], which is a safety-critical application the situation is rather complex, any solution needs to be validated via several validation steps, starting from building an experimental model until real-world experiments. This project work is focused on building the experimental model.

To understand the challenge well and to identify the requirements, one needs to understand the interdisciplinary nature of this model, starting from understanding mechanical concepts about vehicle movement, and vehicle dynamics, then exploring the sensors field, moving to wireless connectivity, and IoT, and finally the Embedded software part which serves as the point of integration to all of these fields.

1.2. Research Questions

Before starting into the design stage, and to counteract any possible issues that might appear in the future; brainstorming and logical analysis are vital to ensure problem comprehension, which has lead to the following research questions:

- Is it possible to build a small scale testbed giving the provided funds?
- Will this model produces a useful –close to real world- results?
- Is it possible to control and monitor the platform and extract data from it?
- Does that model have the capability to test different platooning techniques in terms of control and communication strategies?

1.3. Goals

To define requirements for such a system, one should start by defining goals and specifications in a way that these goals should answer the previously listed research questions, and define the final status that the model should have at the end. These goals can be broken down to the following list:

1. A well-constructed and designed platform that can be used as a testbed (i.e., vehicles and road emulation).
    1.1. Track should have a sufficient length and reliable shape.
    1.2. Vehicles should be able to carry electronics and can be easily controlled on the track with the least effort.
2. Equipped with accurate sensors with proper sampling frequency.
3. Stable communication.
    3.1. Able to provide different communication and message passing schemes as depicted in 2.3.
    3.2. Preserve the Cooperative Awareness Message (CAM) refresh rate.
    3.3. Works with minimum packet loss.
    3.4. Deterministic communication behavior.
4. Real-time and efficient controller.
    4.1. High controller frequency, to preserve the required control update rate.
    4.2. Real-time software using Real-Time Operating System.
    4.3. Atomic execution for critical sections (e.g., measuring relative distance)
5. Additional features:
    5.1. Remote control and monitoring.
    5.2. Multiple communication protocols such as ZigBee, & Wi-Fi, and Low Energy Bluetooth (BLE) to enable further research.
    5.3. Data logging for post-processing analysis.

1.4. Report Overview

After giving some insights about the project's objective, goals, and challenges, the upcoming chapters will describe the project phases with more technical and detailed explanations.

2. Background

In the following section, a brief introduction about each scientific term that is going to be used or has been previously used. Besides, a specific description of the implementation is provided in chapter[3].

2.1. Autonomous Platooning

Platooning is a description of the movement of a group of birds, fishes, or fighter jets; to form a specific shape that leads to more efficient overall movements. Likewise with vehicles. When several vehicles trail each other, it is platooning. But, when vehicles perform this motion automatically, this is called autonomous platooning of the vehicles. Therefore, Autonomous Platooning is an application of Autonomous and self-driving cars field[4].
In the following some of the advantages of vehicles platooning are provided[5]:

1. Increasing road capacity according to the minimum gap distance between vehicles.
2. Help reducing traffic and improve traffic flow..
3. Improving vehicle's aerodynamics and reduce drag forces, therefore, reducing fuel consumption.
4. Allow drivers to perform different tasks and benefit from transportation time.
5. Increases high-ways safety since the movement is determined by the leader vehicle, and if the leader vehicle is moving autonomously, it can be pre-programmed to prevent exceeding speed limit, unsafe lane change, and unexpected rapid braking due to human reaction.

After describing the advantages, the following are some of the challenges that should be taken into consideration when developing an autonomous platooning system[5]:

1. Platoon should have an optimum number of members, long platoon could result in blocking other vehicles movements, unless an Automated High Way system was implemented.
2. Embedded controller should be secured and protected from hacking and Cyber-attacks.
3. Smart handling for junctions and avoid blocking it.

4. Vehicles should be able to, unobstructedly, join and leave the platoon.
5. Overcome these challenges with a simple solution to avoid increasing the cost of vehicles.

2.2. Adaptive Cruise Control

Adaptive Cruise Control is a specific method used to force the vehicle to preserve a certain distance from the preceding vehicle. This exact spacing distance is measured or estimated using only sensors attached to the vehicle itself. The control strategy maintains steady vehicle speed and a preset gap distance. Meanwhile, it aims to reach a set point for a given parameter depending on the type of ACC, which the controller uses, such as; Speed limit-aware cruise control, Eco cruise control for fuel EV capacity savings, Cruise control in curves, etc. Each control method manipulates the vehicle's movement according to the specified setpoint by the driver[[6]. However, one of the main issues for ACC system is that, it is relying only on vehicle's sensors. These sensors are highly affected by environmental conditions. Another issue is that the system is not fully autonomous yet the driver needs to manipulate and maintain vehicle movements and steering[7]. As for control, it is difficult to achieve string stability as it requires vehicle to vehicle (V2V) communication which was one of the reasons to introduce the Cooperative Adaptive Criuse Control (CACC) control strategy[8].

2.3. Cooperative Adaptive Cruise Control

In addition to sensor measurements in ACC systems, the CACC system utilises a new feature which is wireless communication with other vehicles, whether with preceding, or leader vehicle (V2V) see Fig.2.1 It allows the i^{th} vehicle to have more information about the preceding vehicle, the leader vehicle, or any other vehicle -It depends on the running algorithm-. In addition to that, it also enables Vehicle to Infrastructure (V2I) communication ability and vice versa, which allows the controller to have more information about the environment such as weather, street status, best route, etc.

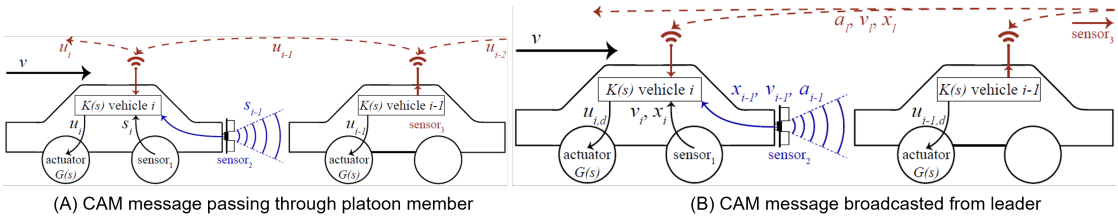


Figure 2.1.: CAM messages passing in CACC could be either A, B, or Combined [10]

Now returning to vehicles platooning, as mentioned in section$_{[2.1]}$ having number of vehicles "String of vehicles" requires the so-called string stability which should maintain the spacing error along them. As described in section$_{[2.2]}$ string stability can be achieved by enabling further information sharing between platoon members (i.e., using CACC strategies).

2.4. Control Methods

After introducing the CACC method, and its relevance to Autonomous Platooning. Achieving string stability is a key factor for stable platoon, one should understand the affecting parameters in this control strategy, and which data should be send and received between platoon members to have the necessary information.

Sting stability is a longitudinal control strategy. It introduces the amplification of the spacing error among a string of vehicles (platoon), and tries to attenuate this error. From control point of view, string stability is guaranteed if the transfer function of the spacing error between the i^{th} vehicle and preceding vehicle is less than one$_{[9]}$

Theorem 2.1. Definition$_{[11]}$: String stability of interconnected vehicles is given when the spacing errors

$$\varepsilon_i(t)$$

of vehicles $2 \leq$ i$\leq$ m attenuate along the vehicle platoon:

$$\| \varepsilon_2(t) \|_\infty > \| \varepsilon_3(t) \|_\infty > ... > \| \varepsilon_m(t) \|_\infty$$

Parameters affecting string stability:

1. Chosen control strategy and controller gain.
2. Vehicles movement and accuracy of measurements.
3. Communication network characteristics such as packet loss and communication delay.

2.5. Wireless Communication

To achieve a stable cooperation between platoon vehicles, wireless communication protocol must be carefully chosen in order to have a deterministic communication behavior, minimal packet loss, suitable data rate, and minimal communication delay.

Nowadays, there are several options such as Wi-Fi, Bluetooth, BLE, and ZigBee. Each protocol of them has different parameters and characteristics. To get some insights about

the differences between them, a simple comparison is presented in Table. 2.1, further information about each protocol stack and frame structure are presented in Appendix.A The platform has been designed to acquire the 4 types (currently only Bluetooth, BLE and Wi-fI are attached to the vehicle. Due to timing issues ZigBee integration was left as future work). Accordingly, it will enable further research and investigations not only for CACC application but also for communication.

2.5.1. Simple Comparison

In this section a simple comparison is made between different communication protocols to give some insights about each one of them as well as the differences.

Parameter	WiFi	ZigBee	BLE
Frequency band(GHz)	2.4 & 5	2.4	2.4
IEEE standard	802.11 a b g	802.15.4	None
Max data rate	High	Low	Moderate
Energy consumption	High	Low	Moderate
Communication layers	Physical Data Link Network Transport Session Presentation Application	physical (PHY) MAC "IEEE 802.15.4 radio standard" network (NWK) application support (APS) application framework (AF) . ZigBee Device Object (ZDO)	physical (PHY) Link layer L2CAP SMP GAP GATT ATT Application Layer

Table 2.1.: Simple comparison between each communication protocol[1]

2.6. Platooning Background Summary

In this chapter a brief introduction for each scientific term was introduced, starting with platooning definition, advantages and disadvantages, and the key difference between ACC and CACC which is string stability, how CACC is related to autonomous platooning, and what parameters should be considered from testbed point of view such as; sensor measurements and communication specifications for stable platooning.
In conclusion, platooning minimum requirements can be listed as follows:

1. Accurate measurements for preceding vehicle's relative distance, velocity and acceleration.
2. Accurate measurements for the i^{th} vehicle displacement, velocity and acceleration.
3. Preserve CAM message interval of 100 ms[12].
4. Preserve controller frequency at 50 Hz[12].

3. Building The Platform

In the upcoming sections, the journey of building this platform is going to be described.

3.1. Defining Model Requirements

To achieve the required goals of this model as presented in section$_{[1.3]}$, which has lead to a set of questions mentioned in section$_{[1.2]}$, the next step was defining the requirements of the model to be able to reach these goals at the end.

As a summary, the full model requirements can be listed as follows:

1. Car model and track should have a suitable length, and size.
2. A suitable Micro-controller should be selected to provide the real-timeliness task, and the size and the weight of the controller shall not exceed the car model limits as well.
3. The model should be equipped with wireless connectivity specifically: WiFi, Bluetooth, BLE, and ZigBee.
4. A set of sensors need to be carefully selected, and they should have a suitable refresh rate, precision, and accuracy.
5. Software design in cooperation with the selected MCU and sensors should allow a 20 ms interval for control update and 100 ms for CAM messages update.
6. Motor driver should be used in order to have control over the motors.
7. Suitable Power supply.
8. Different voltage levels should be available in each car, with sufficient current supply.
9. Create remote control procedure for both ends (Model and user end).
10. Embedded software code should be designed to allow easy test and debugging.
11. Code execution protection techniques shall be used to prevent sensors errors.
12. Code development environment should be flexible and modularized to allow future development and re-usage.

3.2. Components Selection

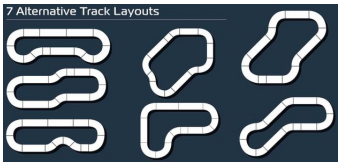
Car Model and Track

Both; Car model and track are the core hardware for such a model. The track should have a sufficient length and reliable shape to ensure that the data resulting from the model would be close to the real world case data. The car should have a compact size yet suitable for containing all components; moreover, the track should allow easy car movement and minimize the overhead resulting from controlling the car to follow certain trajectory.
Based on the knowledge provided in [2],the previous model track was just a simple printed circle followed by the vehicles. Although, the implementation for such a track is easy, However, huge software overhead is required to design a stable line following strategy, so the decision was to use the slotted track and slotted cars approach.

Valid options are presented below in Fig.3.1 and Fig.3.2.

1. SCALEXTRIC

(a) Scalextric (1:32) Car

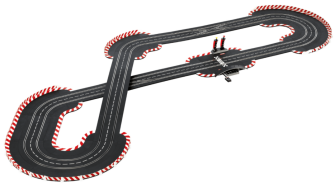


(b) Scalextric standard tracks

Figure 3.1.: Scalextric set[19]

2. CARRERA

(a) Carrera (1:24) Car



(b) Carrera 1:24 standard tracks

Figure 3.2.: Carrera set[20]

Carrera had better size options and less delivery time so the decision was to use it, and the following Car model and track in Fig.3.3 were ordered at the end. Customized track design was made using Carrera's track designer software "Autorennbahnplaner".

Figure 3.3.: Car model, and track with Maximum length of 6.3 m, curvature length of 54 cm. According to Carrera's track planner software, all gaps and clearances are within the safe limits

Microcontroller

To select a suitable microcontroller it should match the following set of requirements:

1. High clock frequency up to 240 MHz.
2. Have several wireless connectivity peripherals as System on Chip (SoC).
3. Has a suitable memory capacity to allow easy usage for external libraries.
4. API supports Real Time Operating System (RTOS) such as FreeRTOS and multi-thread execution.
5. Has several serial communication peripherals such as (SPI, I2C, and UART).
6. Available on Arduino framework as well as having a standalone framework.
7. Cost and energy efficient.
8. Small in size.

The availability of microcontrollers in the market is quite wide; however, the most common ones are; STM, NXP, Espressif, and a Jetson nano kit was proposed.The decision came to choose ESP32 specifically ESP32-dev-kit, as it was matching all the requirements with better size, weight, and price options. Additionally, a small comparison was made between ESP32 and Jetson nano kit in Table.[3.1].

Controllers	ESP32	Jetson Nano Kit
CPU	low-power Xtensa® 32-bit LX6 microprocessor Dual core	Quad-core ARM A57
CPU Clock	240 MHz	1.43 GHz
Flash memory size	External:Up to 16 MB of external flash can be mapped into CPU instruction Internal:448 KB of ROM, 520 KB of on-chip SRAM for data and instructions	4 GB 64-bit LPDDR4 25.6 GB/s
Peripherals	SD card, UART, SPI SDIO, I2C, LED PWM Motor PWM, I2S, IR pulse counter, GPIO capacitive touch sensor, ADC, DAC	GPIO, I2C, I2S SPI, UART USB: 4x USB 3.0, USB 2.0 Micro-B Display: HDMI 2.0 and eDP 1.4
GPIO	32	40
Price (€)	10	94
Wireless connectivity	BLE & Wi-Fi	None

Table 3.1.: $ESP32_{[21]}$ vs Jetson Nano $Kit_{[22]}$.

Selecting ZigBee Module

To complete the full wireless communication set, given that the ESP32 has Wi-Fi, Bluetooth and BLE, the only remaining issue was to select a ZigBee module.

The selected module was the Core2530 in Fig.3.4 provided from WAVESHARE and here are some of the module's features which were the reason for selecting it.

1. Utilises SoC, which makes it very efficient and compact in size.
2. XBee usb extension compatible which allows easy configuration.
3. Uses UART serial communication for data sharing.
4. Running firmware can be easily uploaded using the XBee USB adapter.
5. Can be used as Coordinator, Router, and End Device.
6. Uses CC2530F256RHAR chip provided from Texas Instruments, works on 2.4 GHz, and supports up to 250kbps.
7. The module has 21 GPIOs which makes it able to work as a minor MCU in the system in case of GPIOs shortage in the main MCU, or in case of direct control action need to be taken.
8. Wide supply-voltage operating range (2 V–3.6 V).

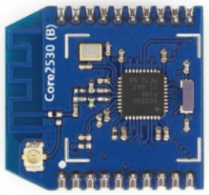


Figure 3.4.: Core2530 SoC ZigBee module[23]

Sensors selection: Speed measurement

1. Measurements within -the same- vehicle
   There are a lot of techniques used to measure vehicle's velocity, and the proposed approaches are:
   - Method(1): Shaft encoders
     There are 2 types of shaft encoders; Optical, and Magnetic. Both have almost the same characteristics, however, they perform differently depending on the environment.
   - Method(2): Linear distance measurement
     This technique was proposed as an alternative to shaft encoders, measuring the linear displacement, and travel time then the velocity is calculated accordingly. Proposed sensors were: Laser sensors and IR reflective sensor.

   At this point, one disadvantage of using slotted car appeared; they are very compact, and shaft encoders require specially designed tires so they can be placed on the motor shaft, so optical shaft encoder was considered a low priority alternative in case the second method was not performing well.

   Second method implementation and testing:

   Due to some difficulties in shipping and purchasing the ADNS-9800 laser sensor, the plan changed to use IR sensors and design distinctive slots, which then will be placed on the track.

   IR-slots adaptation plan:

   It was proposed to test and use the QTR-8A Reluctance Sensor Array and check its performance since it was used in the previous model, and was immediately provided. However, the module has failed directly due to the following reasons:

– The sensor was analog, so it requires using an ADC with the following procedure: Sample, then Wait for some time, and take new sample. Sampling time was huge up to 10 ms per sample, in addition to sampling time and software overhead. Compared to the IR sensor which has sampling time of 10 µs with lower software overhead due to the digital nature of the sensor.

– IR sensor was more compact, and weights less.

In order to verify the precision and accuracy of such a method, a test paper has been designed with different test cases as presented in Fig.3.5.

Calculations:
1. Displacement Calculation: whenever a slot is sensed by the IR sensor, a counter is incremented (Using the pulse counter peripheral in the ESP32).

$$Displacement = No.of slots * Slotwidth$$

2. Velocity estimation: when a pre-determined time has passed, total count is used to estimate the velocity afterwards.

$$Velocity = \frac{Displacement}{timeperiod} \tag{3.1}$$

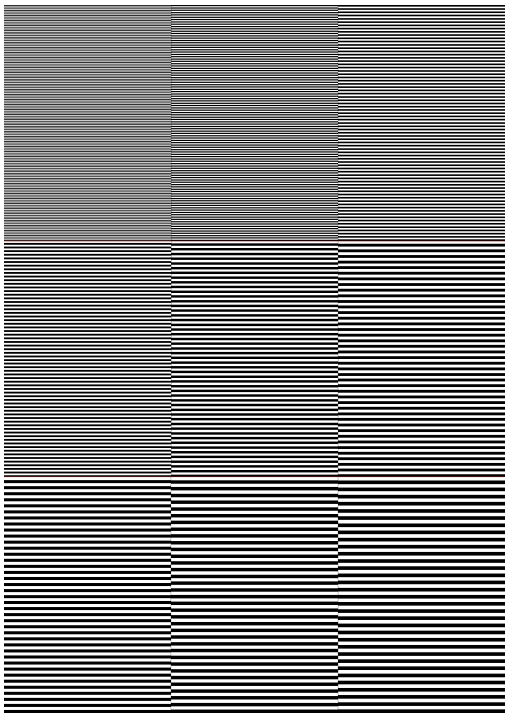


Figure 3.5.: The paper contains 9 sections, each section represents different resolution starting from 0.3 mm until 1.5 mm, so that the optimum resolution can be selected at the end

Results:
Unfortunately, the sensor was not performing as expected, error in counts was up to 50%, with very bad precision.
New workaround:
Another approach was roughly tested, and it was surprisingly performing well. Instead of depending on color difference (White/Black slots), a plastic rail has been designed, and the sensor was used as an artificial touch sensor. Test samples were designed using Solid-Works software and manufactured using CNC Laser cutting machine (see Fig.[3.6]).

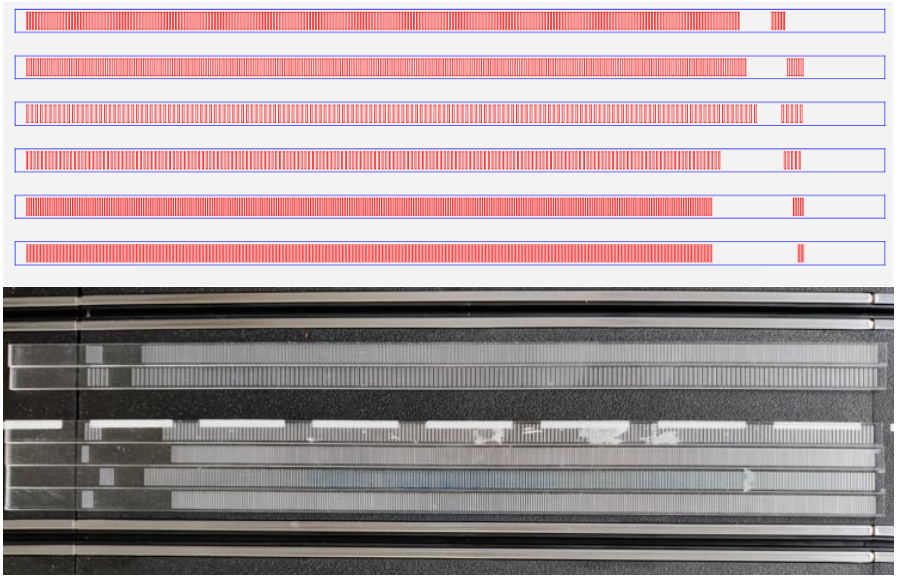


Figure 3.6.: 5 samples was manufactured with the following tuples (slot width "mm", spacing "mm");(0.3,0.3) (0.5,0.5) (0.8,0.8) (0.8,1) (1,1) (1,0.8). It can be seen that there are 5 extra slots in each sample to allow visual inspection when passing the sensor above them.

Results:
Test was done on the samples, and the output signal was captured using the logic analyzer; this approach was performing well, and the sensor was able to capture with a very low resolution (Signal analysis was done on the output signal and the latency was minimal in terms of 10 th of micro-seconds), however, one main issue appeared. The plastic samples was supposed to be placed underneath the vehicle, in order to do that, the thickness of the samples should not exceed 1.7 mm.

Final decision:
After discussing the issue, it was decided to use the PMW3360 Gaming mouse laser sensor module (see Fig.[3.7]) and here are some of its notable features:

- The sensor is using SPI serial communication for data sharing, which makes it has faster and suitable response.
- Frame rate up to 12000 Frame Per Second (fps).
- Due to high fps, delay was also minimal, approximately (1 - 2) ms.
- Resolution was also very high, up to 12000 Count Per Inch (CPI), however, the sensor can be easily configured to work with the desired resolution.
- Acceptable speed up to 250 Inch Per Second (IPS).
- Acceptable acceleration up to 50g.
- 3.3V or 5V compatible.

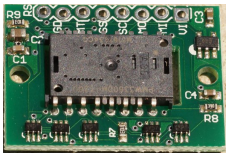


Figure 3.7.: PMW3360 Laser sensor module[24]

- Measuring preceding vehicle's speed:

  Two methods were suggested to be used to calculate preceding vehicle motion:

  - Method(1): Using Doppler(Radar) speed sensor

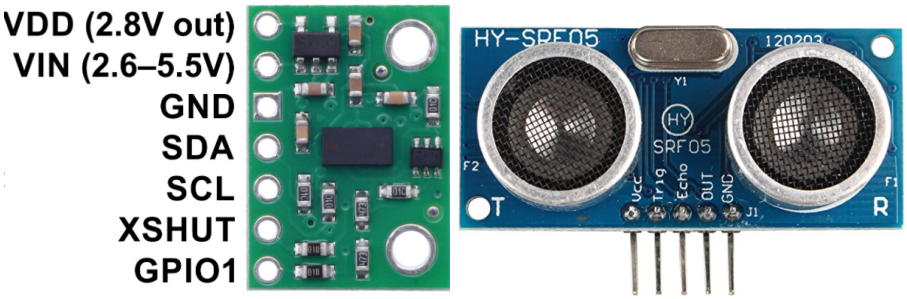
    After investing sometime searching about suitable Doppler effect sensor to measure the velocity of an object, two sensors were found which are; HB100 Doppler radar, and K-LD 2-EVAL sensor; however, both were not suitable for use due to the following reasons:
    1. They are meant to measure high speed, and high rate of change, therefore, they will not be accurate.
    2. Their output readings are noisy and require usage of Hardware and Software filters.
    3. Their size is large, and will not fit in our model.
  - Method(2): Using distance sensor and elapsed time

    The Second method is using the distance sensor, measure the rate of change of the spacing distance between the vehicles and divide it by the elapsed time.

  Due to the presented issues with the first method, second method was implemented.

Sensors Selection: distance measurement

Distance measurements is a popular topic, and there are plenty of sensors and techniques have been developed for it, each method has specific specs and suits different applications. In this model, two sensors have been proposed, presented in Fig.[3.8] also a simple comparison between them was made in Table.[3.2].



(a) ToF sensor:(Vl53l0X)[25] (b) Ultrsonic sensor:(HY-SRF05)[26]

Figure 3.8.: Proposed distance measurement sensors

Comparison between both sensors:

	Ultrasonic	ToF
Output type	Digital	Serial communication (I2C)
Resolution	2 mm	1 mm
Size	4.5 × 2.1	4 × 3 × 1
Wide angle	15	25

Table 3.2.: Ultrasonic[26] vs VL53L0X[25] specifications

Results:

Tests were performed on a test bench, output readings were recorded and the following results were plotted in Fig.[3.9], and Fig.[3.10], and analyzed using the Logic analyzer in Fig.[3.11]. Data recording and analysis was done using MATLAB.

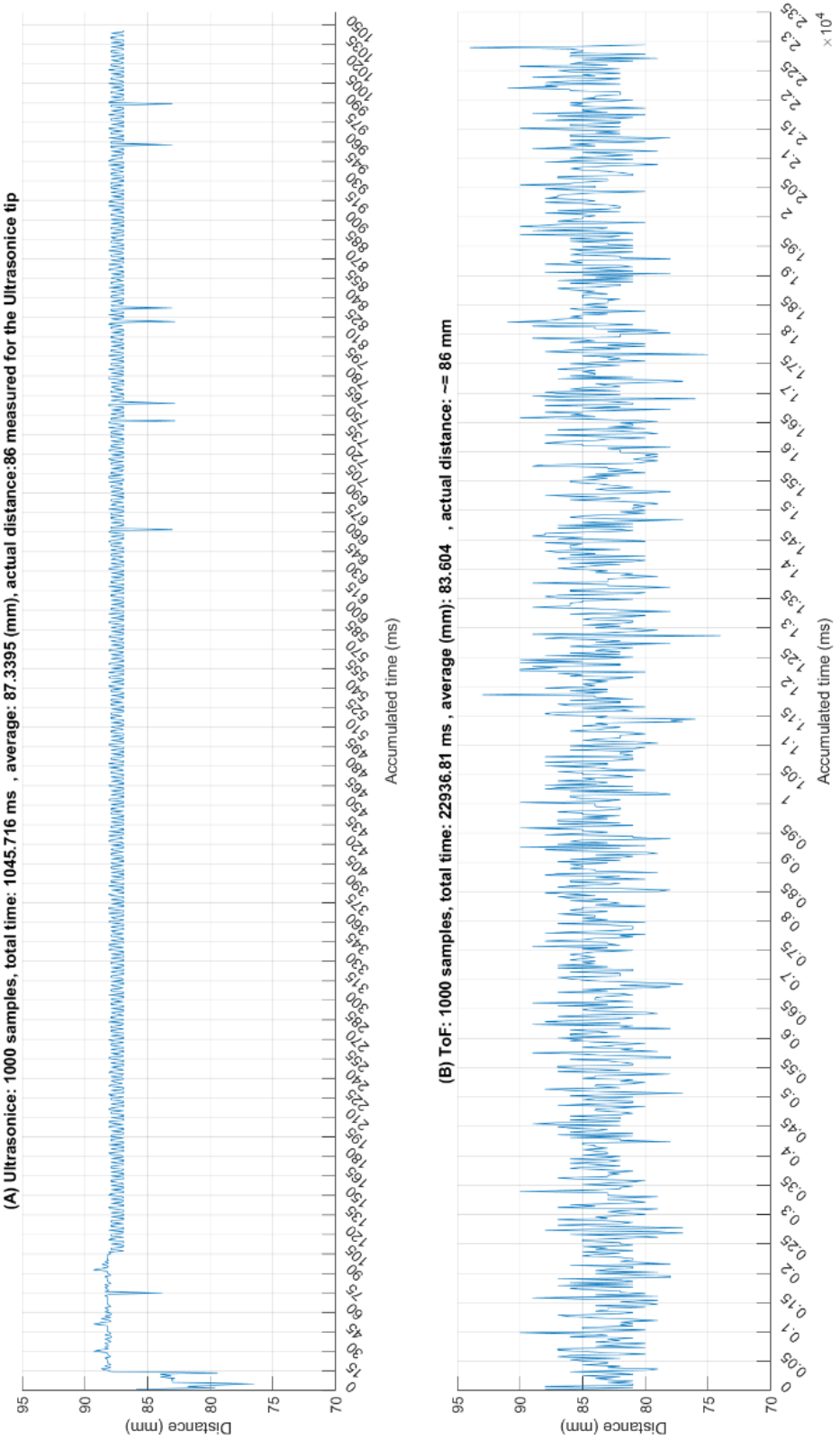


Figure 3.9.: Distance measurement comparison between ToF and Ultrasonic sensors

Figure 3.10.: Fig(A) shows the differences between the sampling speed of each sensor (i.e., sampling latency, Fig(B): Compares between the stability of measurement for the Ultrasonic and ToF sensors

To verify the sampling interval of the Ultrasonic sensor, the output signal was analyzed using the Logic Analyzer and recorded as follows:

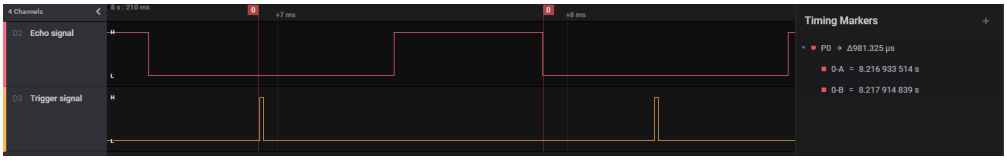


Figure 3.11.: Ultrasonic physical signal snapshot. It can be seen that the exact sampling time is 981.325 microseconds.

Conclusions:

Based on the previous observations, it was decided to use Ultrasonic at the end instead of the VL53L0X ToF sensor. Since the HY-SRF05 Ultrasonic was used, here are some of its specifications[26]:

1. Detection distance: 2cm-450cm.
2. High precision: Up to 0.2cm.
3. Input trigger signal: 10us TTL impulse
4. Input voltage: 5V.
5. Sensor angle: Not more than 15 degrees, it is narrower than the ToF, which is an advantage.

Power Supply

The power supply should provide the correct voltage levels for all components, and also must provide the platform with the required current. LEDMO adaptor -as shown in Fig.3.12- was used to provide this functionality, which has 12 V, and 5 A. To verify the conditions, a simple calculations have been made and presented in Table[3.3]:

Component	Voltage level (V)	Current rating (mA)	Quantity
ESP32	3.3	1200 MAX	3
Ultrasonic	5	2	3
PMW3360 Laser sensor	3.3	23	3
DEV8838 Motor driver	11	200	3
Core2530	3.3	29 MAX	3
Summary	Total current: Voltage levels:		4360 5,3,3,11

Table 3.3.: Power requirements summary for each component

Figure 3.12.: LEDMO power adaptor adaptor[28]

Motor Driver

To achieve stable and controllable motion, the motor driver should be used to provide control using PWM signals, additionally, protection components should be added as well to protect the MCU from reverse current.
After some investigation about the used motors in Carrera digital 124 cars, it turned out that it has brushed DC motor which requires 18 V, and consumes about 100 mA according to VIKING SCALEXTRIC SLOT CAR CLUB[1], to limit the speed the selected motor driver[Fig.3.13] supplies up to 11 V, and has 1.7 amps maximum current for safety.

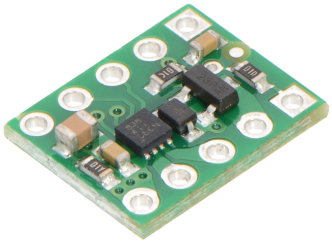


Figure 3.13.: DRV8838 Brushed DC motor driver, output voltage(0 - 11), current rating up to 1.7[29]

Logic Analyzer

Logic analyzers are vital in order to debug any serial communication issue that might appear; specially when dealing with serial communication sensors, additionally, visual inspection of output waves and the ability to inspect them in Digital and Analog forms. Therefore, Saleae Logic 8[Fig.3.14] has been used for this task.

[1] https://vikingslotcarclub.co.uk/motor-specifications/

Figure 3.14.: Saleae Logic 8[30]

3.3. Development Environment

The ESP32 is a popular MCU in the IoT community, therefore, it has more than one framework used for development, available options are micro-Python, PlatfrormIO, Arduino, and ESP-IDF. Each framework has advantages and disadvantages, and requires different skills and knowledge, additionally, it can be programmed using Python, C, C++, etc. PlatformIO, Arduino, and ESP-IDF have been tested and finally, a flexible and well-documented PlatformIO environment has been prepared for future development. A description of how to install, set-up, and use it easily has been provided in Git-lab Wiki[31].

3.4. Software Implementation

In this section a brief illustration for each software module is described, and execution flow chart is provided.

3.4.1. Communication Module

In the beginning, it was planned to compare between different communication protocols and select the most suitable one; however, due to the limitation in time only Wi-Fi protocol has been implemented and used.

Several options for Wi-Fi existed, either using the regular Wi-Fi protocol and treat each platoon member as a station, and log their data directly. However this scheme will also require to use each platoon member as an Access Point (SoftAP), to implement that each ESP32 will work as a Client and Server. Accordingly, another method was selected.

The better alternative is to use the MESH concept. When using MESH libraries; one needs only to perform some configurations regarding the mesh itself. There were 2

libraries available, either to use ESP-MDF (It is an API provided from Espressif, contains additional application layers running on top of the ESP-IDF), Or to use the so-called Painlessmesh library.

The decision was to try and test the Painlessmesh library because it was performing time synchronization mechanism between mesh members, however the performance was not suitable, and packet were lost with a very high percentage up to 80%, and the behavior was not deterministic, as for communication delay, it was changing from 10 ms up to 200 ms with no receive acknowledgment.

The other alternative which has been implemented at the end was the ESP-NOW protocol[32]; the behavior was deterministic, latency was in terms of micro-seconds (approximately 1 $\approx$ 1.5 ms), automatic receive acknowledgment, broadcasting ability, but MESH formation is done manually not like the MDF, or Painlessmesh.

Mesh communication flow is illustrated in the Finite State Machine (FSM) presented in Fig.[3.15].

ESP-NOW specifications[32]:

1. Application data is encapsulated in a vendor-specific action frame and then transmitted from one Wi-Fi device to another without connection.
2. CTR with CBC-MAC Protocol(CCMP) is used.
3. ESP-NOW is connection-less.
4. ESP-NOW is a low-power 2.4GHz wireless connectivity protocol.
5. Able to send up to 250 bytes at a time.
6. Frame format:

MAC Header	Category Code	Organization Identifier	Random Values	Vendor Specific Content	FCS
24 bytes	1 byte	3 bytes	4 bytes	7 ~ 255 bytes	4 bytes
Vendor Specific Content:					
Element ID	Length	Organization Identifier	Type	Version	Body
1 byte	1 byte	3 bytes	1 byte	1 byte	0 - 250 bytes
Frame section	Definition:				
Category Code	The Category Code field is set to the value(127) indicating the vendor-specific category				
Organization Identifier	The Organization Identifier contains a unique identifier (0x18fe34), which is the first three bytes of MAC address applied by Espressif				
Random Value	The Random Value filed is used to prevents relay attacks				
Element ID	The Element ID field is set to the value (221), indicating the vendor-specific element				
Length	The length is the total length of Organization Identifier, Type, Version and Body				
Type	The Type field is set to the value (4) indicating ESP-NOW				
Version	The Version field is set to the version of ESP-NOW				
Body	The Body contains the ESP-NOW data				

Table 3.4.: ESP-NOW protocol Frame format[32]

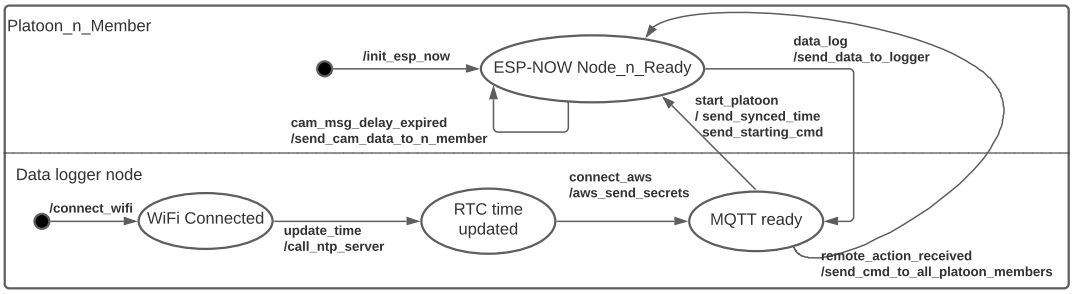


Figure 3.15.: Communication FSM for ESPs Mesh

3.4.2. Relative Motion Handler Module

To calculate the spacing error between the i^{th} vehicle and preceding vehicle, an Ultrasonic sensor was used, the sensor transmits a sound wave from the transmitter and receives it upon reflection from the preceding vehicle, given the velocity of sound, distance is calculated as follows:

Theorem 3.1.

$$D \ = \ (\frac{1}{2}) \times T \times C$$

where D is the distance, T is the trip time, and C is the speed of sound $\approx 343 \ \frac{m}{s}$.

After measuring the distance, relative velocity error and relative acceleration is estimated. The estimation algorithm (Finite State Machine) FSM is presented in Fig.3.16.

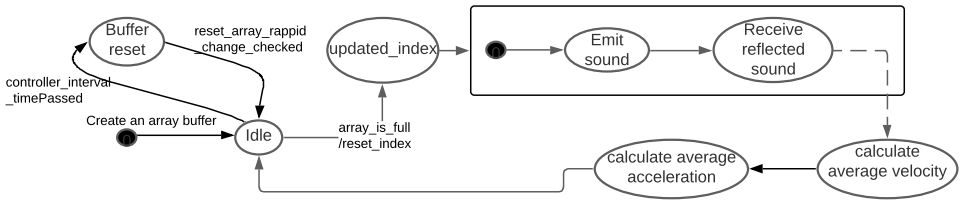


Figure 3.16.: Ultrasonic distance measurement FSM

3.4.3. Motion Handler Module

Laser sensors are using CMOS technology to work as a small camera that captures a number of pixels with a certain speed defined by Frame Per Second (FPS). Additional hardware peripherals shall be used such as Digital Signals Processors (DSPs), and filters for more stable capturing. Accordingly, a running firmware should exist, and a small MCU as well to manage these data, and send it to the main MCU via Serial Peripheral Interface (SPI) protocol. All of this is embedded in the PMW3360 module.

After capturing the image the firmware uses an algorithm to analyze the differences between each frame; therefore, calculates the change in pixels, and the change in distance consequently.

Displacement is measured in a pre-defined time; thus velocity and acceleration can be calculated as illustrated in the FSM presented in Fig.3.17

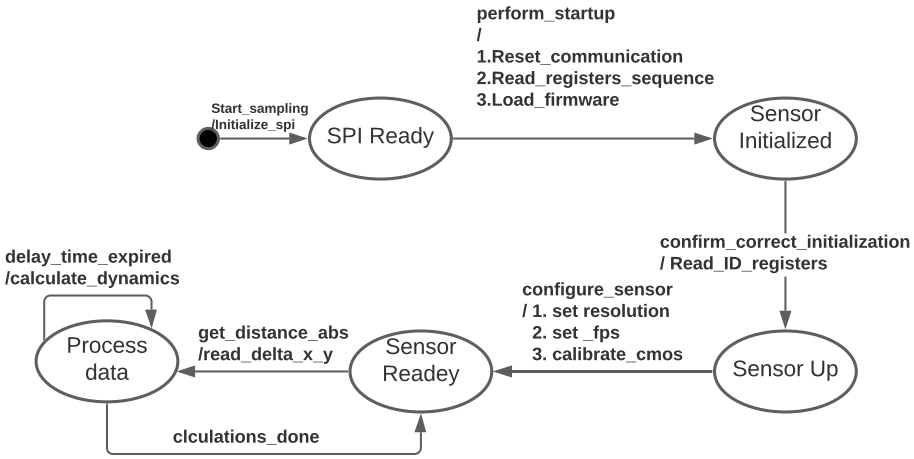


Figure 3.17.: Laser sensor own vehicle dynamics measurements

3.4.4. Pulse Width Modulation Module

Any motor driver is based on a Diode and additional protection electronics. It requires a modulated signal to vary the output voltage on motor terminals, signal modulation is done using a technique called Pulse Width Modulation (PWM); it continuously switches the output voltage signal on motor terminal between 0 and 1 with a predefined rate(i.e., frequency) and interval called Duty.

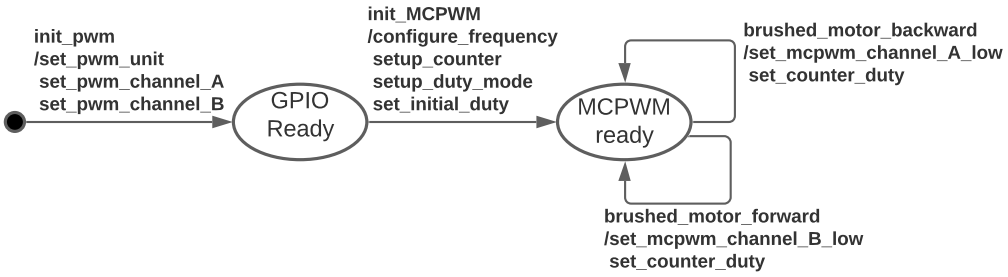


Figure 3.18.: Motor driver velocity control FSM

3.4.5. FreeRTOS Module

To ensure real-timeliness between different tasks, given that the ESP32 has 2 cores, FreeRTOS[2] was selected to handle communication between tasks, and define tasks priorities.

Fig.3.19, and Fig.3.20 illustrate how tasks are assigned to each CPU with priority indication. The platoon contains 4 main members, 3 vehicles(Leader + 2 followers), and a fourth node responsible for remote operation, control, and data logging.

3.4.6. Tasks Layers

1. Platoon members:

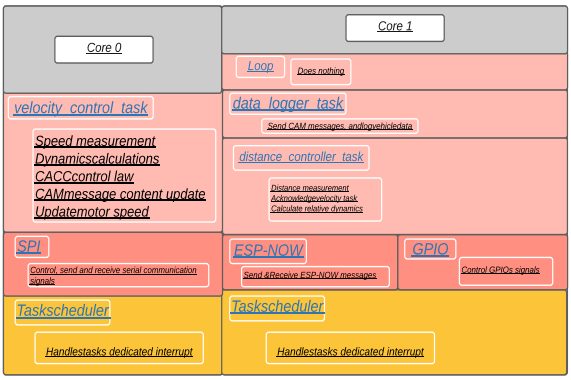


Figure 3.19.: Software Architecture for Platoon members, priority increases from top to bottom level tasks, each task uses different peripherals, and has different interrupts.

[2]FreeRTOS is a Real-Time Operating System that exists as a middleware layer in the software architecture

2. Control node:

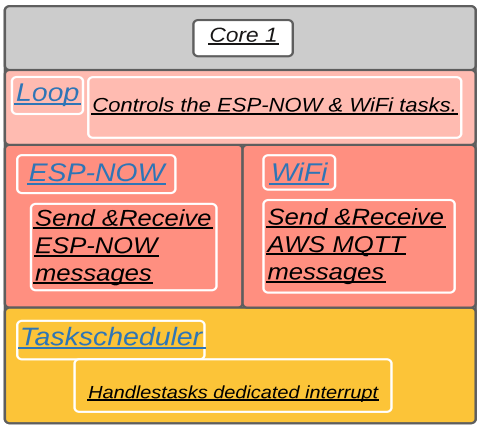


Figure 3.20.: Software Architecture for root node(Control node), priority increases from top to bottom level tasks.

3.4.7. Task Scheduling

The FreeRTOS uses the Round-Robin scheduling method, the scheduler assigns a core, priority, stack size, and name for each task, whenever a high priority task is delayed or suspended on a single CPU, the preceding task starts to execute in a cyclic way[18]. Round Robin scheduler is a preemptive scheduler, meaning that low priority tasks can be interrupted whenever a high-priority task is ready. Moreover, the system uses shared-memory scheme; thus, using Critical sections, Semaphores, Mutex, and Event Queues is a must in order to have a correct execution of the program.

Program Execution Flow

In this section, a detailed description is provided to understand how tasks are created, and how the FreeRTOS handles them accordingly. This flow chart applies for platoon vehicles only.

1. Core 1 execution and program start:
   The "main" function is implicitly pinned to Core_0 (PRO_CPU), in case of the Arduino framework, this function creates an additional task which is the "Arduino" task and assigns it to Core_1 (APP_CPU), afterwards, the main function execution stops.

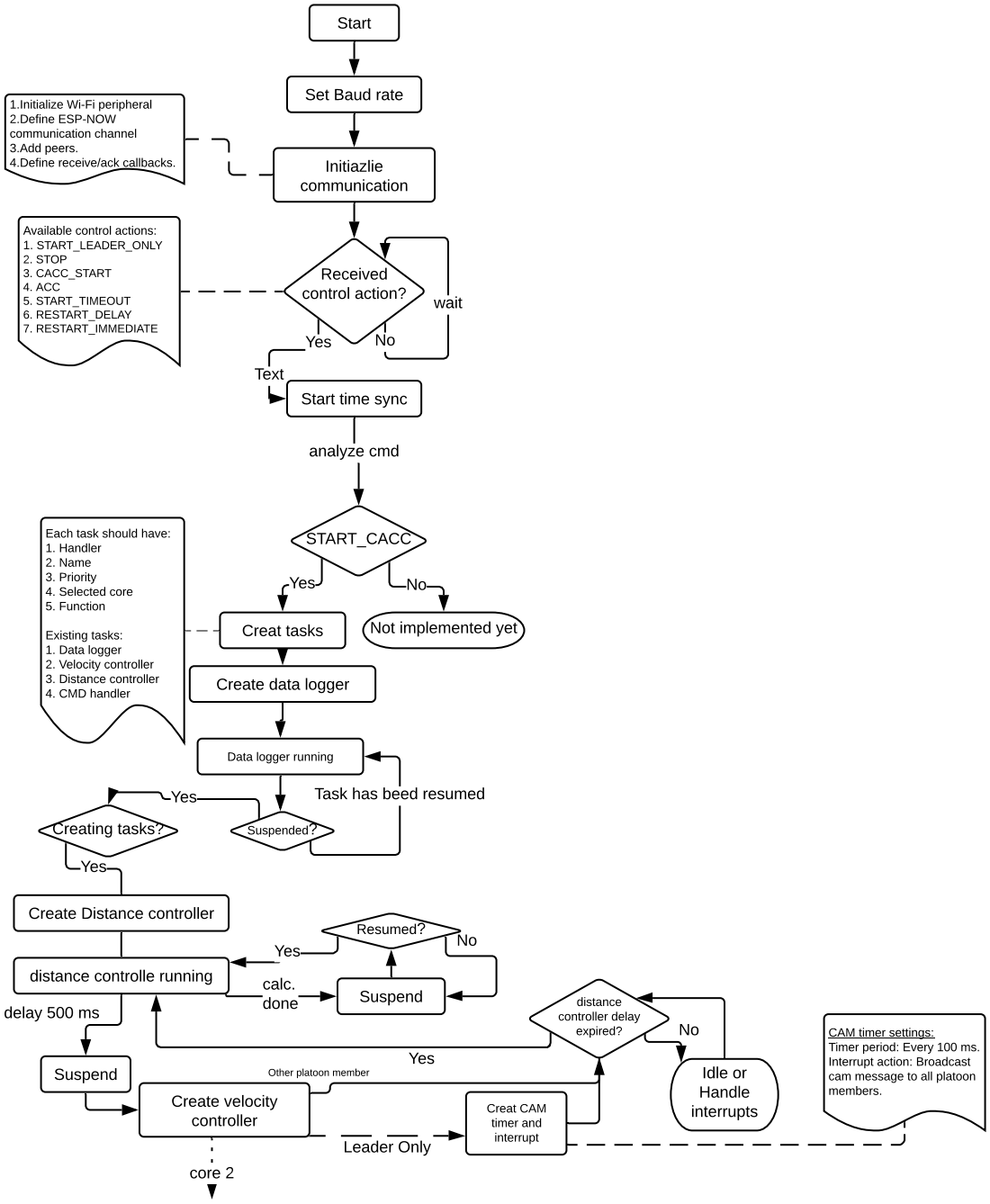


Figure 3.21.: Program execution flow for APP_CPU

2. Core 0 execution flow chart:
   After task creation was finished, the PRO_CPU (Core 0) will execute the tasks as illustrated.

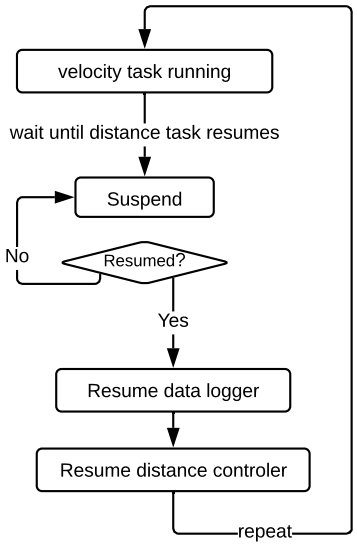
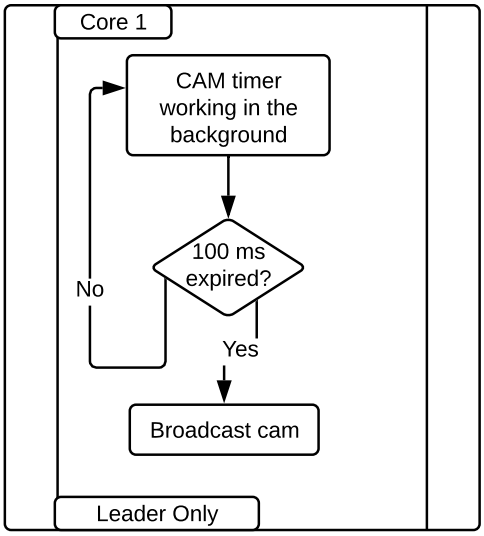
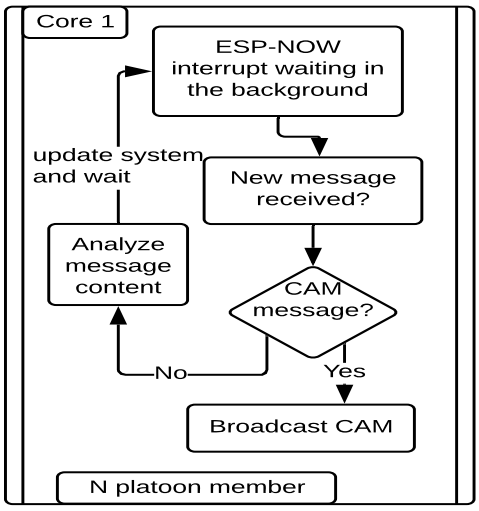


Figure 3.22.: Program execution flow for PRO_CPU

3. Finally, there are another tasks running in the background (On the Hardware peripherals); in Fig.3.23 the main 2 peripherals' interrupts are illustrated.



(a) CAM message timer interrupt; ensures that CAM message is broadcasted every 100 ms.



(b) ESP-NOW receive handler; send CAM message whenever a new message received from Leader and/or the preceding vehicle

Figure 3.23.: Main background tasks and interrupts

3.5. Hardware Schematic

In this section the wiring schematic is presented, including all the currently connected components, voltage levels is illustrated as well as other sensor and communication signals.

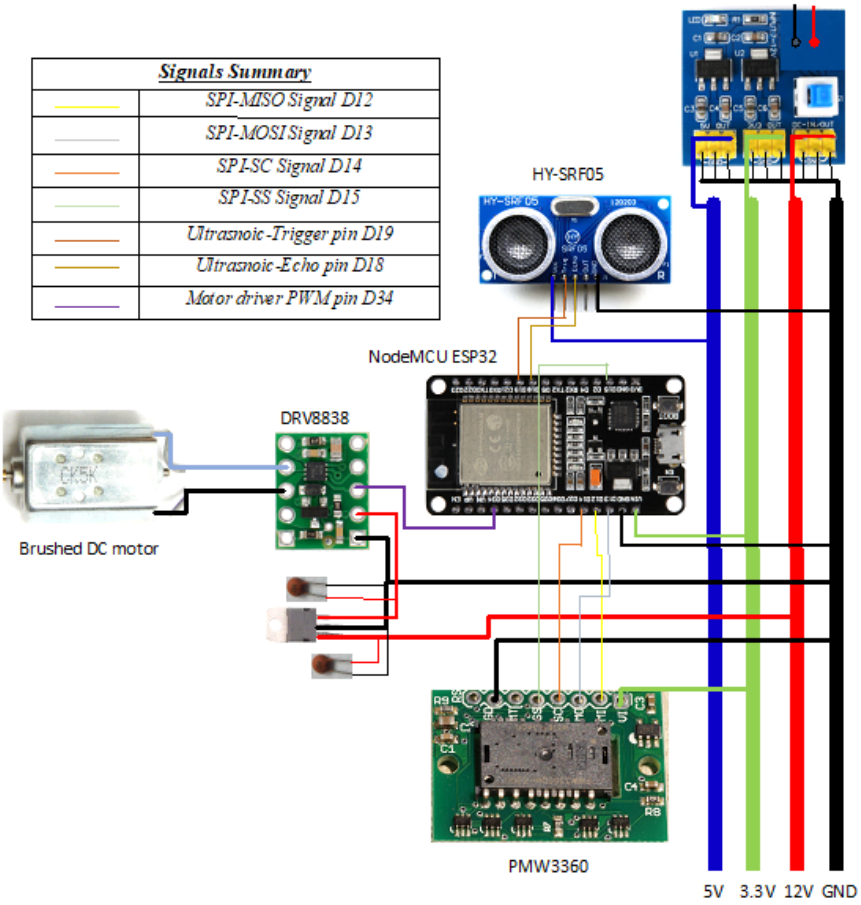


Figure 3.24.: Wiring schematic and signals summary

3.6. Hardware Implementation

In this section a set of pictures are presented in a sequential way to show the setup and construction steps of the model until it was finally completed as clarified in Fig3.25, Fig.3.26, and Fig.3.27.

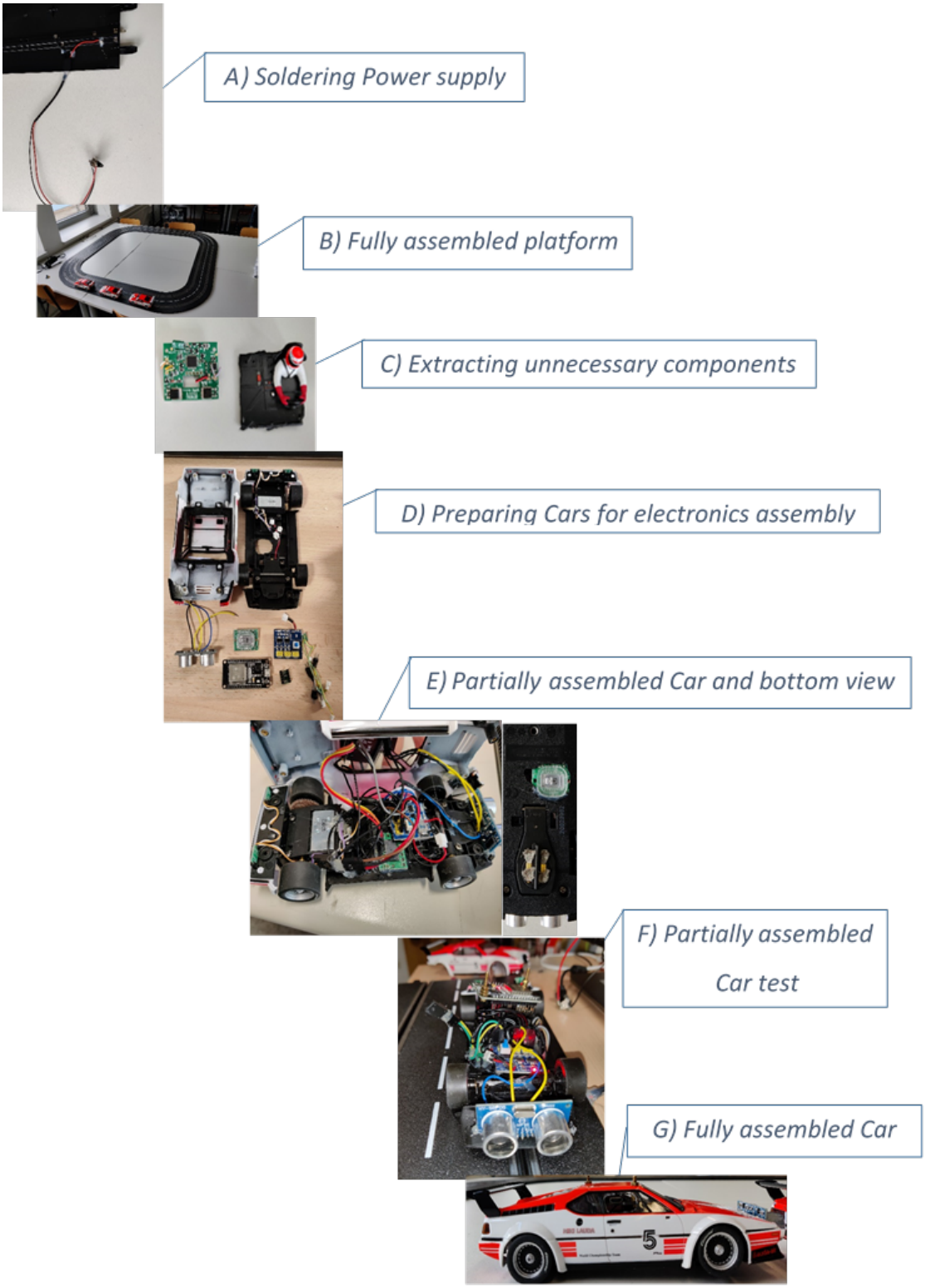


Figure 3.25.: Model hardware construction stages

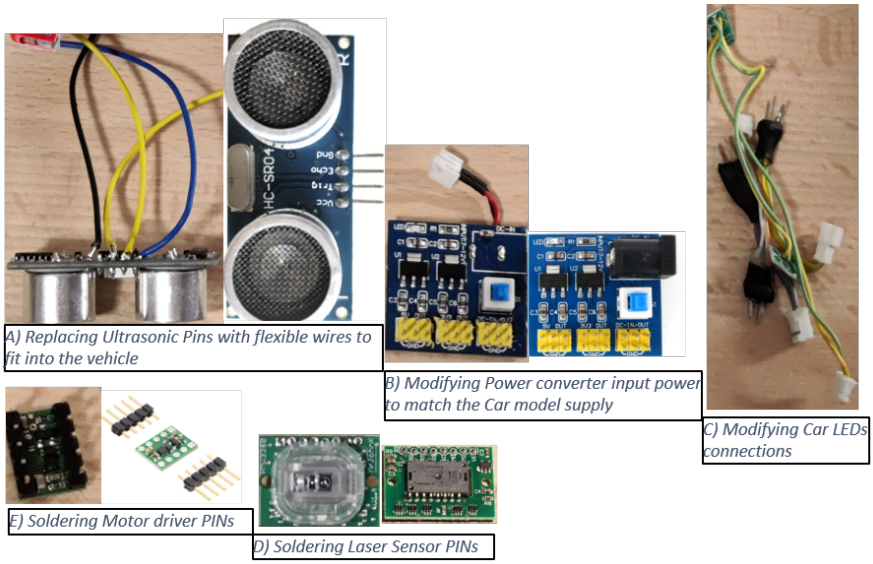


Figure 3.26.: Electronics before and after modifications

Figure 3.27.: Final Model

3.7. Additional Features

In addition to the main features, an additional features have been added to increase the flexibility, and efficiency of the system.

3.7.1. Message Queuing Telemetry Transport (MQTT)

MQTT is a publish/subscribe protocol, where two devices work as client, and a server like device called broker should run in the middle and broadcast messages to all clients. Since each device only subscribe for a set of topics, it will only listen to these topics, and publish on different topic, in other words, two devices will have two different channels one for sending and one for receiving[27].

In the current implementation Amazon Web Service (AWS) is used to provide the broker service. It listens to all published message and direct them to the subscribed devices. Additionally, it handles another tasks such as sessions establishment, subscription, authentication, and connections management.

In the current implementation the platoon starts in an idle state waiting for a command, this command should be sent from a remote PC; when the LogNode is connected to a network, it will listen to the published commands and forward it to all platoon members, then they shall react according to the received command. This flow is illustrated in Fig.3.28.

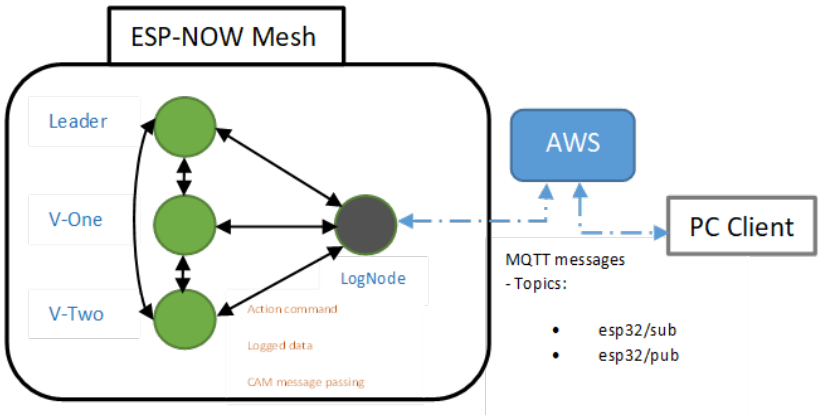


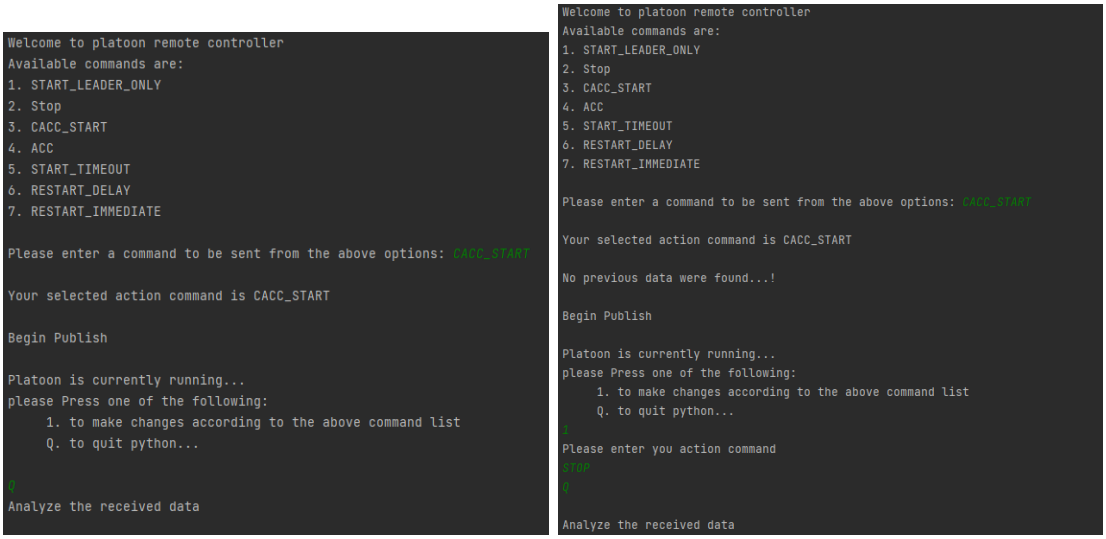
Figure 3.28.: MQTT messages using AWS and its interaction with platoon members

A python script was written to provide easy of use of this functionality, one should just run the script and a set of options will appear after entering a command, the script will first clean up and prepare the working directory to avoid mixing new data with previously saved data, then it will continuously run and listens to the publish data from the platoon

(the logged data) in order to be analyzed, this data is saved in a CSV file during run-time, when the session is terminated by the user, the script starts automatically to analyze the received data and separate them to different CSV files for each vehicle, this scenario is presented in Fig.3.29.

With this feature one can easily control the platoon, describe different motion trajectories, or even change communication schemes, and control laws on the fly. Currently, a simple set of control commands are implemented, it is not completed yet, but this left as future work since the basic blocks have been tested and verified.

A test case was also performed for data logging, a set of dummy CAM messages was sent from leader vehicle, received by the script and finally saved in a separate file as shown in Fig.3.30.



(a) Python UI Test case 1 (b) Python UI Test case 2

Figure 3.29.: MQTT messages using python script and AWS

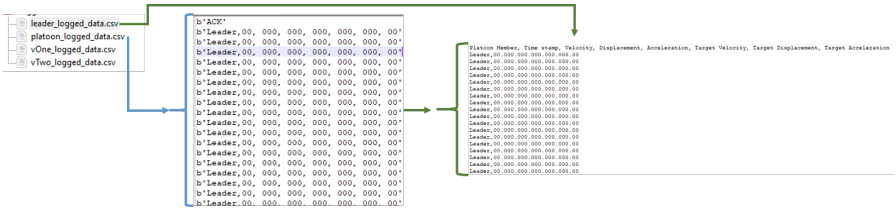


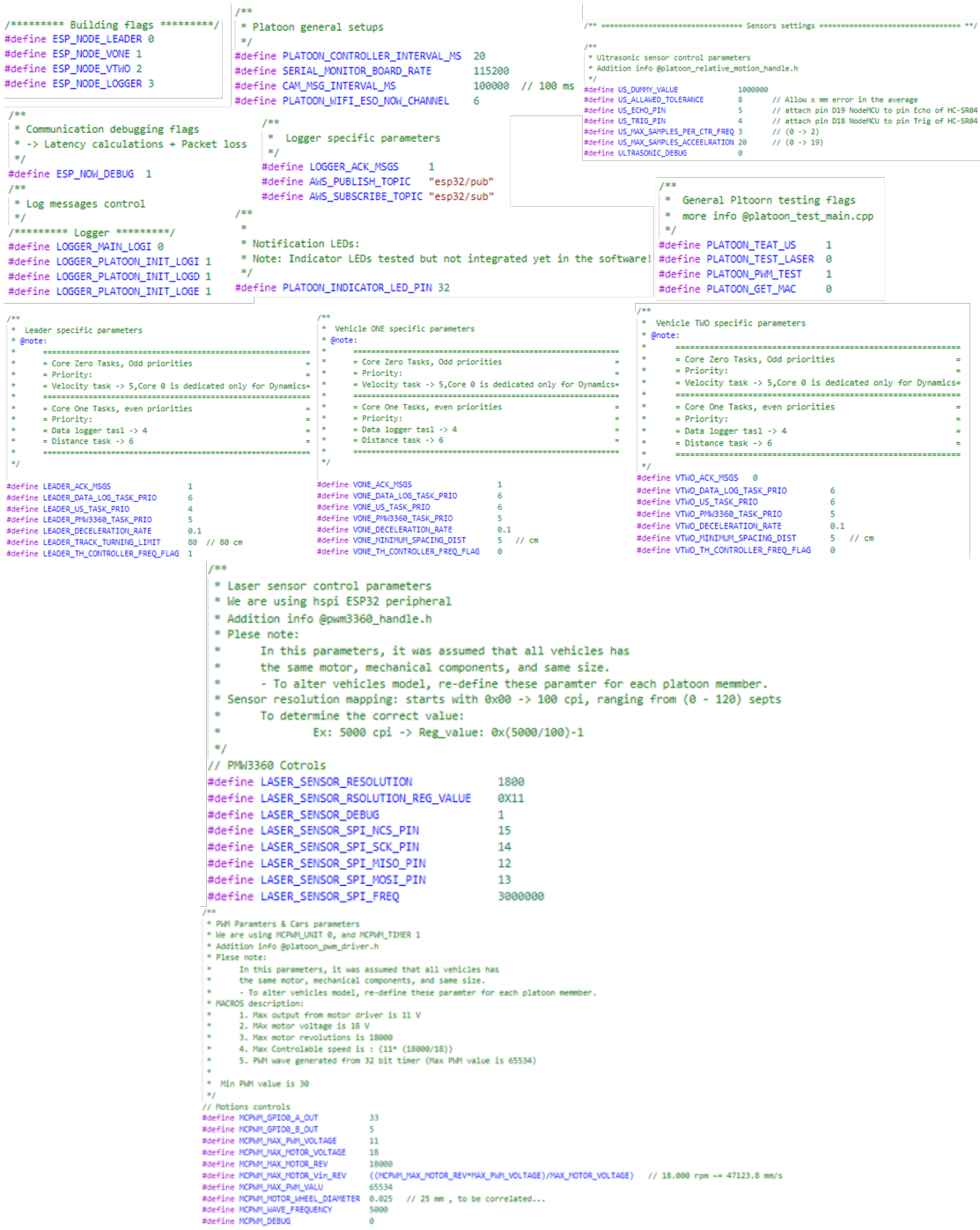
Figure 3.30.: Data logging and separation

3.7.2. System Configuration File

An SDK-like header file has been implemented, this file has all the hardware parameters as well as CACC parameters, debug control flags, sensors calibration parameters, etc. This is a common file used by the building system, all platoon members, and the logging node in the ESP32 mesh.

The essential role of this file is to provide easy and quick changes in all modules without having to dig inside each module to modify a specific parameter. For instance, if the wheel diameter has changed, one needs to search for wheel diameter macro and modify it, this can be done without even knowing where this variable is used (see Fig.3.31)
Advantages of this functionality are:

- Easy software configuration.
- Easy software adaptation in case of hardware changes.
- Easy CACC modifications, and test cases.
- Save researcher time instead of looking for a variable in a complete software for each node and change it, one just needs to look in one file, apply the changes, and compile.



```
/********* Building flags *********/
#define ESP_NODE_LEADER 0
#define ESP_NODE_VONE 1
#define ESP_NODE_VTWO 2
#define ESP_NODE_LOGGER 3
```

```
/**
 * Platoon general setups
 */
#define PLATOON_CONTROLLER_INTERVAL_MS  20
#define SERIAL_MONITOR_BOARD_RATE       115200
#define CAM_MSG_INTERVAL_MS             100000  // 100 ms
#define PLATOON_WIFI_ESO_NOW_CHANNEL    6
```

```
/** ============================== Sensors settings ============================== **/

/**
 * Ultrasonic sensor control parameters
 * Addition info @platoon_relative_motion_handle.h
 */
#define US_DUMMY_VALUE              1000000
#define US_ALLAWED_TOLERANCE        8       // Allow x mm error in the average
#define US_ECHO_PIN                 5       // attach pin D19 NodeMCU to pin Echo of HC-SR04
#define US_TRIG_PIN                 4       // attach pin D18 NodeMCU to pin Trig of HC-SR04
#define US_MAX_SAMPLES_PER_CTR_FREQ 3       // (0 -> 2)
#define US_MAX_SAMPLES_ACCEELRATION 20      // (0 -> 19)
#define ULTRASONIC_DEBUG            0
```

```
/**
 * Communication debugging flags
 * -> Latency calculations + Packet loss
 */
#define ESP_NOW_DEBUG  1
/**
 * Log messages control
 */
/********* Logger *********/
#define LOGGER_MAIN_LOGI 0
#define LOGGER_PLATOON_INIT_LOGI 1
#define LOGGER_PLATOON_INIT_LOGD 1
#define LOGGER_PLATOON_INIT_LOGE 1
```

```
/**
 *  Logger specific parameters
 */
#define LOGGER_ACK_MSGS     1
#define AWS_PUBLISH_TOPIC   "esp32/pub"
#define AWS_SUBSCRIBE_TOPIC "esp32/sub"
```

```
/**
 *
 * Notification LEDs:
 * Note: Indicator LEDs tested but not integrated yet in the software!
 */
#define PLATOON_INDICATOR_LED_PIN 32
```

```
/**
 *  General Pltoorn testing flags
 *  more info @platoon_test_main.cpp
 */
#define PLATOON_TEAT_US     1
#define PLATOON_TEST_LASER  0
#define PLATOON_PWM_TEST    1
#define PLATOON_GET_MAC     0
```

```
/**
 *  Leader specific parameters
 * @note:
 *      ============================================================
 *      = Core Zero Tasks, Odd priorities                          =
 *      = Priority:                                                =
 *      = Velocity task -> 5,Core 0 is dedicated only for Dynamics=
 *      ============================================================
 *      = Core One Tasks, even priorities                          =
 *      = Priority:                                                =
 *      = Data logger tasl -> 4                                    =
 *      = Distance task -> 6                                       =
 *      ============================================================
 */
#define LEADER_ACK_MSGS               1
#define LEADER_DATA_LOG_TASK_PRIO     6
#define LEADER_US_TASK_PRIO           4
#define LEADER_PMW3360_TASK_PRIO      5
#define LEADER_DECELERATION_RATE      0.1
#define LEADER_TRACK_TURNING_LIMIT    80  // 80 cm
#define LEADER_TH_CONTROLLER_FREQ_FLAG 1
```

```
/**
 *  Vehicle ONE specific parameters
 * @note:
 *      ============================================================
 *      = Core Zero Tasks, Odd priorities                          =
 *      = Priority:                                                =
 *      = Velocity task -> 5,Core 0 is dedicated only for Dynamics=
 *      ============================================================
 *      = Core One Tasks, even priorities                          =
 *      = Priority:                                                =
 *      = Data logger tasl -> 4                                    =
 *      = Distance task -> 6                                       =
 *      ============================================================
 */
#define VONE_ACK_MSGS               1
#define VONE_DATA_LOG_TASK_PRIO     6
#define VONE_US_TASK_PRIO           6
#define VONE_PMW3360_TASK_PRIO      5
#define VONE_DECELERATION_RATE      0.1
#define VONE_MINIMUM_SPACING_DIST   5   // cm
#define VONE_TH_CONTROLLER_FREQ_FLAG 0
```

```
/**
 *  Vehicle TWO specific parameters
 * @note:
 *      ============================================================
 *      = Core Zero Tasks, Odd priorities                          =
 *      = Priority:                                                =
 *      = Velocity task -> 5,Core 0 is dedicated only for Dynamics=
 *      ============================================================
 *      = Core One Tasks, even priorities                          =
 *      = Priority:                                                =
 *      = Data logger tasl -> 4                                    =
 *      = Distance task -> 6                                       =
 *      ============================================================
 */
#define VTWO_ACK_MSGS   0
#define VTWO_DATA_LOG_TASK_PRIO     6
#define VTWO_US_TASK_PRIO           6
#define VTWO_PMW3360_TASK_PRIO      5
#define VTWO_DECELERATION_RATE      0.1
#define VTWO_MINIMUM_SPACING_DIST   5   // cm
#define VTWO_TH_CONTROLLER_FREQ_FLAG    0
```

```
/**
 * Laser sensor control parameters
 * We are using hspi ESP32 peripheral
 * Addition info @pwm3360_handle.h
 * Plese note:
 *      In this parameters, it was assumed that all vehicles has
 *      the same motor, mechanical components, and same size.
 *      - To alter vehicles model, re-define these paramter for each platoon memmber.
 * Sensor resolution mapping: starts with 0x00 -> 100 cpi, ranging from (0 - 120) septs
 *      To determine the correct value:
 *              Ex: 5000 cpi -> Reg_value: 0x(5000/100)-1
 */
// PMW3360 Cotrols
#define LASER_SENSOR_RESOLUTION             1800
#define LASER_SENSOR_RSOLUTION_REG_VALUE    0X11
#define LASER_SENSOR_DEBUG                  1
#define LASER_SENSOR_SPI_NCS_PIN            15
#define LASER_SENSOR_SPI_SCK_PIN            14
#define LASER_SENSOR_SPI_MISO_PIN           12
#define LASER_SENSOR_SPI_MOSI_PIN           13
#define LASER_SENSOR_SPI_FREQ               3000000
```

```
/**
 * PWM Paramters & Cars parameters
 * We are using MCPWM_UNIT 0, and MCPWM_TIMER 1
 * Addition info @platoon_pwm_driver.h
 * Plese note:
 *      In this parameters, it was assumed that all vehicles has
 *      the same motor, mechanical components, and same size.
 *      - To alter vehicles model, re-define these paramter for each platoon memmber.
 * MACROS description:
 *      1. Max output from motor driver is 11 V
 *      2. MAx motor voltage is 18 V
 *      3. Max motor revolutions is 18000
 *      4. Max Controlable speed is : (11* (18000/18))
 *      5. PWM wave generated from 32 bit timer (Max PWM value is 65534)
 *
 *  Min PWM value is 30
 */
// Motions controls
#define MCPWM_GPIO0_A_OUT           33
#define MCPWM_GPIO0_B_OUT           5
#define MCPWM_MAX_PWM_VOLTAGE       11
#define MCPWM_MAX_MOTOR_VOLTAGE     18
#define MCPWM_MAX_MOTOR_REV         18000
#define MCPWM_MAX_MOTOR_Vin_REV     ((MCPWM_MAX_MOTOR_REV*MAX_PWM_VOLTAGE)/MAX_MOTOR_VOLTAGE)   // 18.000 rpm ~= 47123.8 mm/s
#define MCPWM_MAX_PWM_VALU          65534
#define MCPWM_MOTOR_WHEEL_DIAMETER  0.025   // 25 mm , to be correlated...
#define MCPWM_WAVE_FREQUENCY        5000
#define MCPWM_DEBUG                 0
```

Figure 3.31.: Software configuration file (platoonSDK.h)

3.7.3. Flexible Building System And Automated Scripts

A set of python and batch scripts were added to the development environment to make life easier for future usage.

Software Building System Flexibility

As described in section 3.3 PlatformIO framework is used as the building environment, to boost the development process for each build/upload/monitor action. Four environments were configured and attached with different flags, and COM ports, to allow parallel building and binary generation, parallel code flashing and multiple MCUs monitoring at the same time (see Fig.3.32), the fifth environment is for testing the hardware and validation purposes.

Separating the building environments allow for excluding the unused libraries, therefore prevents the system from building them which has a huge effect on reducing the building time.

```
; General configurations for each and every ESP32
[esp32dev]
platform = espressif32
board = esp32dev
framework = arduino

lib_deps =
  # RECOMMENDED
  # Accept new functionality in a backwards compatible manner and patches
  bblanchon/ArduinoJson @ ^6.18.0
  WiFi
  SPI
  Ticker
  platoon_relative_motion_handle
  platoon_vehicle_motion_handle

board_build.f_cpu = 240000000L
board_build.flash_mode = qio
upload_speed = 921600
monitor_speed = 115200
board_build.f_flash = 80000000L
board_upload.flash_size = 4MB

lib_ldf_mode = chain+
```

```
; Leader Vehicle Configurations
[env:leader]
extends = esp32dev
src_filter =
  +<platoon_n_mamber_main.cpp>
  -<platoon_logNode_main.cpp>
  -<platoon_test_main.cpp>
build_flags =
  -Iinclude
  -D ESP_PLATOON_NODE=0
  -D CORE_DEBUG_LEVEL=3
upload_port = com6
monitor_port = COM6
lib_ignore =
    platoon_logNode_init
    platoonVOne_communication
    platoonVOne_FreeRTOS_tasks
    platoonVTwo_communication
    platoonVTwo_FreeRTOS_tasks
```

```
; Vehicle ONE Configurations
[env:vOne]
extends = esp32dev
src_filter =
  +<platoon_n_mamber_main.cpp>
  -<platoon_logNode_main.cpp>
  -<platoon_test_main.cpp>
build_flags =
  -Iinclude
  -D ESP_PLATOON_NODE=1
  -D CORE_DEBUG_LEVEL=3
upload_port = com4
monitor_port = COM4
lib_ignore =
    platoon_logNode_init
    platoonLeader_communication
    platoonLeader_FreeRTOS_tasks
    platoonVTwo_communication
    platoonVTwo_FreeRTOS_tasks
```

```
; LogNode building environment
[env:logger]
extends = esp32dev
src_filter =
  -<platoon_n_mamber_main.cpp>
  +<platoon_logNode_main.cpp>
  -<platoon_test_main.cpp>
build_flags =
  -Iinclude
  -D CORE_DEBUG_LEVEL=3
upload_port = com5
monitor_port = COM5
lib_deps =
  # RECOMMENDED
  # Accept new functionality in a backwards compatible manner and patches
  bblanchon/ArduinoJson @ ^6.18.0
  WiFi
  bblanchon/StreamUtils @ ^1.6.1
  platoon_logNode_init

lib_ignore =
    platoonLeader_communication
    platoonLeader_FreeRTOS_tasks
    platoonVOne_communication
    platoonVOne_FreeRTOS_tasks
    platoonVTwo_communication
    platoonVTwo_FreeRTOS_tasks
    platoon_FreeRTOS_common
```

```
;test env configuration
[env:test]
extends = esp32dev
src_filter =
  -<platoon_n_mamber_main.cpp>
  -<platoon_logNode_main.cpp>
  +<platoon_test_main.cpp>
build_flags =
  -Iinclude
  -D CORE_DEBUG_LEVEL=3 ; info
upload_port = com4
monitor_port = COM4

# Discard unwanted libraries to boost building time.
lib_ignore =
    platoon_logNode_init
    platoonLeader_communication
    platoonLeader_FreeRTOS_tasks
    platoonVOne_communication
    platoonVOne_FreeRTOS_tasks
    platoonVTwo_communication
    platoonVTwo_FreeRTOS_tasks
    platoon_FreeRTOS_common
```

Figure 3.32.: Building environment configuration file

A secondary feature is provided using batch script (platoon_run.bat) to run and execute commands, one can use either Visual Studio Code (VSC) terminal, windows power shell,

or command prompt window to run this file. The file is equipped with different arguments to abstract from VSC PIO buttons. Also, it is able to save the data in a text file, for instance, if someone wants to build then upload and monitor for the Leader vehicle, but also wants to save the serial monitor data to be analyzed later can use the following arguments:
"platoon_run leader -b"
"platoon_run leader -u_m -f"

Automation

This feature is used for MATLAB scripts, during testing the algorithms, multiple figures and data is generated each time during parameters tuning. All MATLAB scripts save the generated figures in the "DOCS" directory with indexing criteria points to the tuning parameter as presented in Fig.3.33.

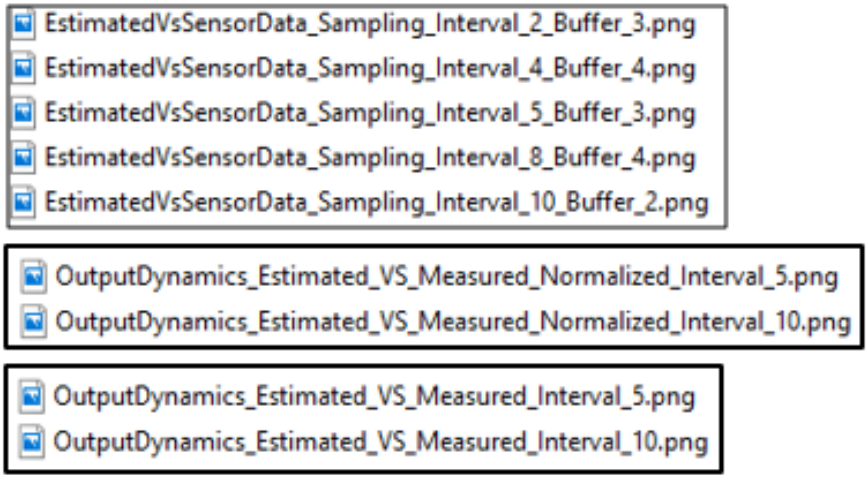


Figure 3.33.: Automatic figure saving with indexing according to the tuning parameter

Additionally, when tuning for the leader vehicle's velocity profile, and to avoid generating the trajectory in run time on the embedded processor, the script "generate_vel_profile.m" uses the provided velocity profile vector previously saved by the test script, and converts linear speed to PWM values, plots the trajectory, and calls the "profile_parser.py" which is a python script responsible for header file generation in the correct directory as shown below in Fig.3.34.

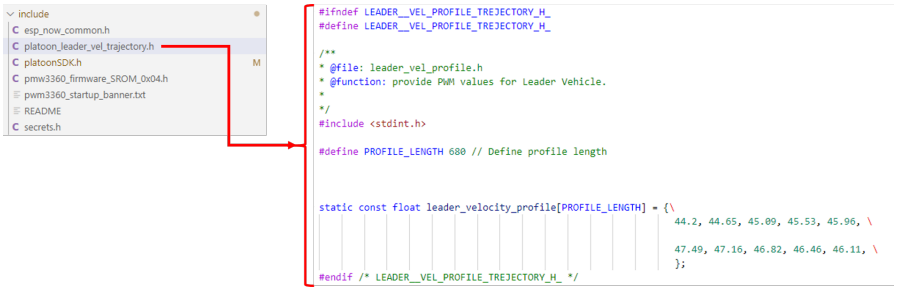


Figure 3.34.: Automatically generated velocity profile header file

3.7.4. Two Degrees Of Freedom Debugging

Debugging is the most powerfull tool that can have a huge effect on boosting the testing process, and catching bugs easily.

In this platform, there are 2 debugging criteria:

1. The ESP32 has a logging library API, which has 6 different levels of data logging as follows:

   - ESP_LOGE, error -> lowest.
   - ESP_LOGW, warning.
   - ESP_LOGI, info.
   - ESP_LOGD, debug.
   - ESP_LOGV, verbose -> highest.

   An SDK configuration file is also provided from Espressif to allow the developer to control and modify the running software and hardware parameters. Controlling the logging level is one of these parameters which can be chosen from this SDK file.
   How it works, one can easily control verbosity level, where if you enable ESP_LOGE level in this case, only messages assignee as error will be compiled and sent to the developer. If the highest level is enabled, all messages will be sent and appear on the serial console...etc.

2. In addition to the logging library, an additional debug flags have been added in the SDK like file. These flags allow the developer to select which module (File or software portion) should be compiled for debugging.
   For example to debug the Laser sensor, one can just enable this flag, define the logging level for the ESP logging library, based on the selection for debugging flags and levels, only the messages associated with this configuration will appear for the Laser sensor.

4. Platform Performance Analysis and Summary

In the previous chapter, the design and implementation parts of this experimental platform were carefully explained, in this chapter; the model will be evaluated and analyzed.

Taking into consideration that this work was mainly focusing on designing and implementing such a complex and highly interdisciplinary system; however, due to time limitation, performance evaluation effort was reduced in order to fit within the timeline, yet a general evaluation was done.

Main evaluation points:

- Distance measurement: Spacing error accuracy, relative velocity and acceleration estimation efficiency.
- Displacement measurement, velocity, and acceleration estimation.
- CAM message: Packet loss percentage, latency and CAM interval preservation.
- Controller frequency and task latency.
- Successful data logging and separation, which has been illustrated in section 3.7.1.
- Successful remote control action delivery.

4.1. Preceding Vehicle Motion Analysis

One of the main parts of the CACC control law; is spacing error control, the rate of change of this error, and the relative acceleration of the preceding vehicle.

As the Ultrasonic sensor performance explained in Chapter.3 Fig:3.9, and the measurement and estimation process was discussed in section [3.4.2]. In this chapter the complete algorithm will be evaluated.

Theoretical evaluation:

The first step of the evaluation process starts with simulation and theoretical analysis, for that; a MATLAB script was written (test_relative_motion_algo.m) it simulates the distance controller task during run-time with a set of assumptions and approximations as follows:

1. Leader vehicle's velocity assumed to be constant, and a spacing error profile was estimated. This profile represents an ideal sensor measurements in analog world.
2. According to the estimated profile, an estimated velocity and acceleration was also evaluated; thus the ideal profiles are prepared (see Fig.4.1).
3. To simulate real-world case, two factors should be taken into consideration:
   - Sensor error: which was assumed to vary from (1.04 - 1.1, these values are based on experimental test ,and additional safety margin was added) and injected in the ideal readings using the random number generator engine in MATLAB.
   - Discretization: as Embedded Systems are discrete, thus the actual reading will be a set of connected points separated by the sampling interval.
     Comparison between spacing error profiles is presented in Fig.4.2 with normalized plot.
4. Assuming that the distance controller task is repeated every 10 ms. The expected measured readings are shown in Fig.4.3 normalized plots can be seen in Fig.4.4.
5. Finally after injecting measurements error, and discretization effect, the evaluation process can be summarized as follows:
   - Control parameters are: Sampling interval, averaging buffer size, and sensor error.
   - Optimum buffer size for velocity is 3 - 5, as for acceleration up to 20 gives promising results, however, this also increases computation time on the Embedded target.
   - Increasing sampling frequency improves velocity profile, but acceleration profile becomes noisy.
   - Generally, the results are promising, yet need to be verified on the Embedded target.

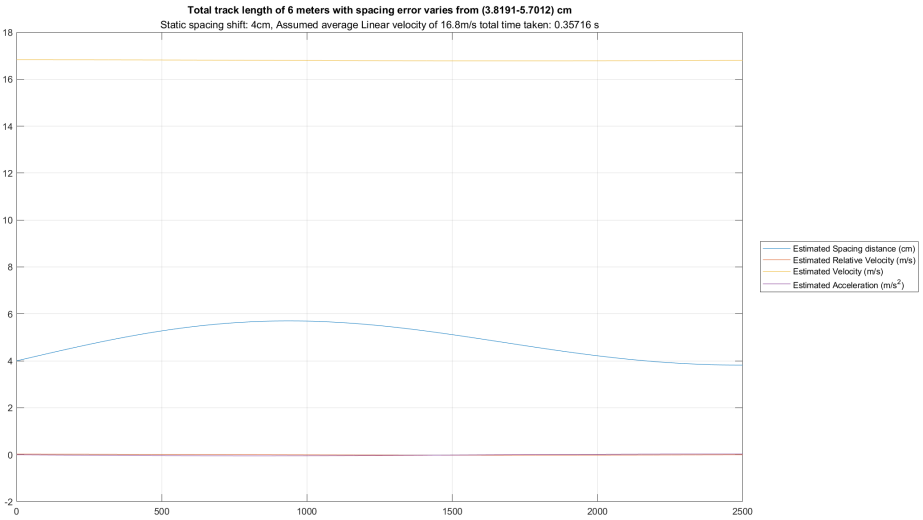


Figure 4.1.: Estimated relative motion profile

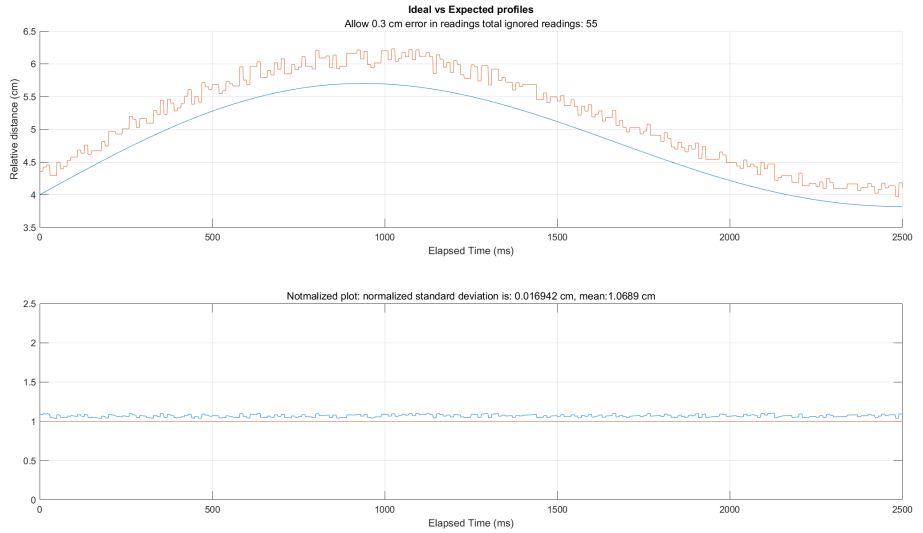


Figure 4.2.: Estimated Spacing distance vs Spacing distance with error injection and discretization effect.

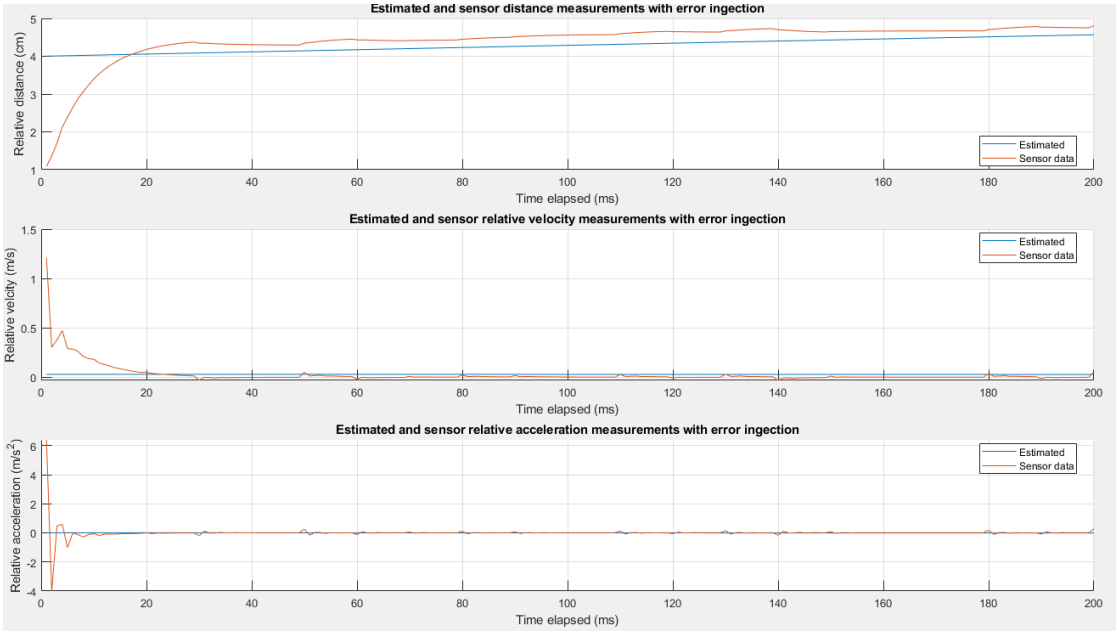


Figure 4.3.: Position, Velocity, and Acceleration profiles comparisons

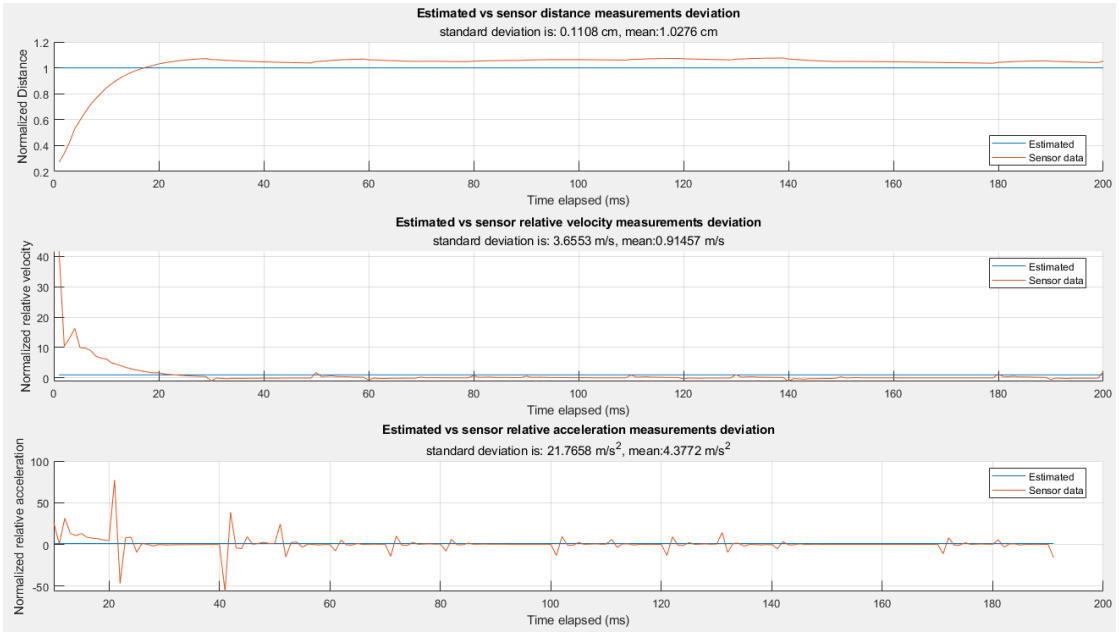


Figure 4.4.: Normalized Position, Velocity, and Acceleration profiles

4.2. Motion Analysis Within The Vehicle

In addition to measurements associated with the preceding vehicle, own vehicle measurements are also required for motion analysis, as the estimation algorithm is explained in section$_{[3.4.3]}$, a theoretical evaluation was performed: .

1. Following the same criteria as the previous test, starts by assuming vehicle's motion profile as shown in Fig.4.5.
2. Error injection: after testing the Laser sensor and calculating error percentage and adding a safety margin, the error is set to vary from (1.04 - 1.32).
3. For discretization, the sampling interval was set to be 10 ms.
4. Results and Summary:
   1. Increasing the sampling interval has a huge effect for velocity estimation accuracy different than the previous test.
   2. Acceleration requires a noise filter.
   3. Optimum average buffer size for velocity is 3 - 5, while acceleration starts from 20.
   4. Final plots are shown in Fig.4.6, and it can be seen that there is a constant shift in the velocity profile due to accumulated error in displacement measurements. As for the acceleration, it shows a promising profile with 20 points average buffer size, however, this comes with the price of increasing computation time.

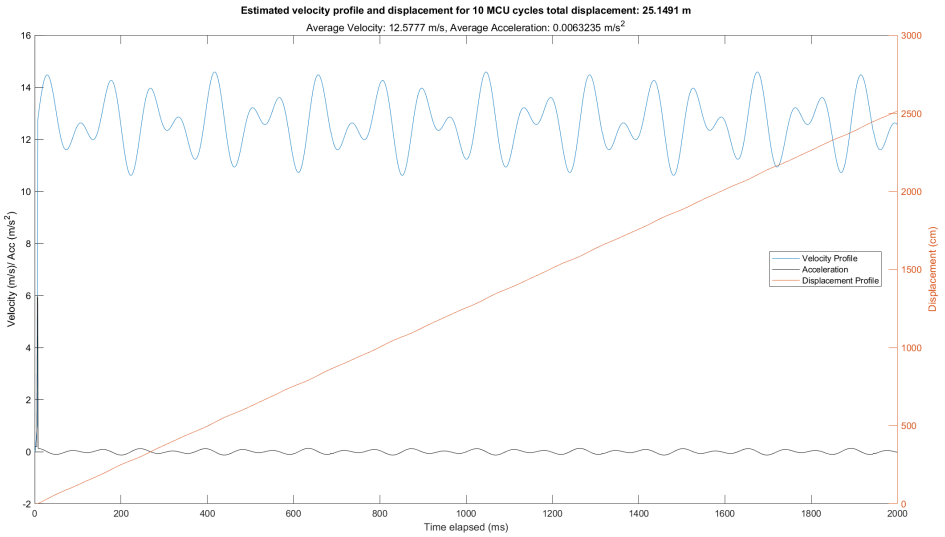


Figure 4.5.: Assumed Vehicle Motion Profile

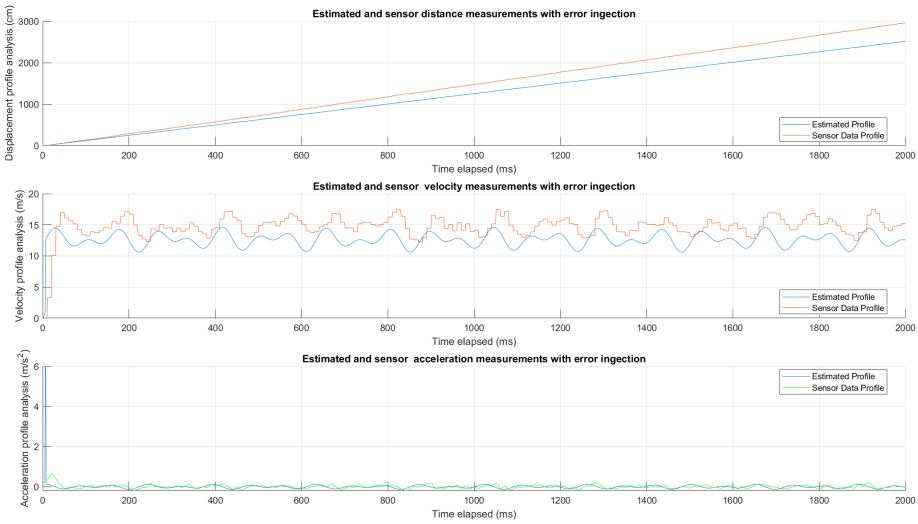


Figure 4.6.: Assumed velocity profiles vs Expected profiles.

4.3. CAM Messages

Cooperation is a critical part for CACC system. Therefore, to assure CAM message delivery, one needs to calculate packet loss percentage, as well as CAM message latency. Refer to section [3.4.1], to review the currently running communication protocol, and Fig.3.15 for algorithm explanation.

Test case:
Two MCUs were set to represent Leader and Vehicle One operation; One controller sends a CAM data each 100 ms, and the receiving controller sends another CAM message; once a new messages is received from the leader MCU, then the latency is calculated in terms of microseconds.
The results are shown in Table.4.1

No. of Sent messages	No. of acks	Packet loss %
6344	6330	0.22 %
Average CAM delivery:		
Varies from 100 - 104 ms which is a very satisfying result		

Table 4.1.: Communication test results summary

4.4. Controller Frequency Preservation

One of the conditions to achieve stable platooning is controller frequency. It should be $\leqslant$ 20 ms, giving that the FreeRTOS allows multi-thread and prioritized execution as well as tasks distribution among the two cores.

A test case was made to measure the latency of each task, considering their execution as described in section 3.4.7 as follows:

$$Total\ latency = Distance\ measurement\ task + Data\ logging\ task + Velocity\ control\ task$$
$$+ Communication\ send/receive\ time + \ software\ overheadtime$$

, where Total latency is $\leqslant 20ms$.

Results:

1. Distance (i.e., Ultrasonic task):
   1.1. Leader: Since the leader vehicle does not use the relative motion analysis algorithm, but only measures distance, task latency was in the range of (1 - 3) ms, note that; this time is also dependent on sound wave travel time associated with the target object.
   1.2. Other platoon members: Adding computation time for the motion analysis algorithm, the latency was in the range of (9 ms up to 16 ms)
2. Velocity (i.e., laser sensor task):
   Average task latency was in the range from (11 ms up to 20 ms).

Since the velocity (i.e., Laser sensor tasks); is running on a dedicated CPU without interruption, then the maximum latency in the cycle is 20 ms which is critical, yet matches the requirements. Note: These results are not completely in the safe margins, therefore, additional analysis is required with integration of all hardware components, also having the cars running on the track is required.

4.5. Remote Commands

As discussed in section [3.7.1], the importance and the advantages of having remote control actions over the platoon, which has been described in Fig:3.29, message sequence, and the corresponding status on the target side was tested and illustrated in Fig.4.7.

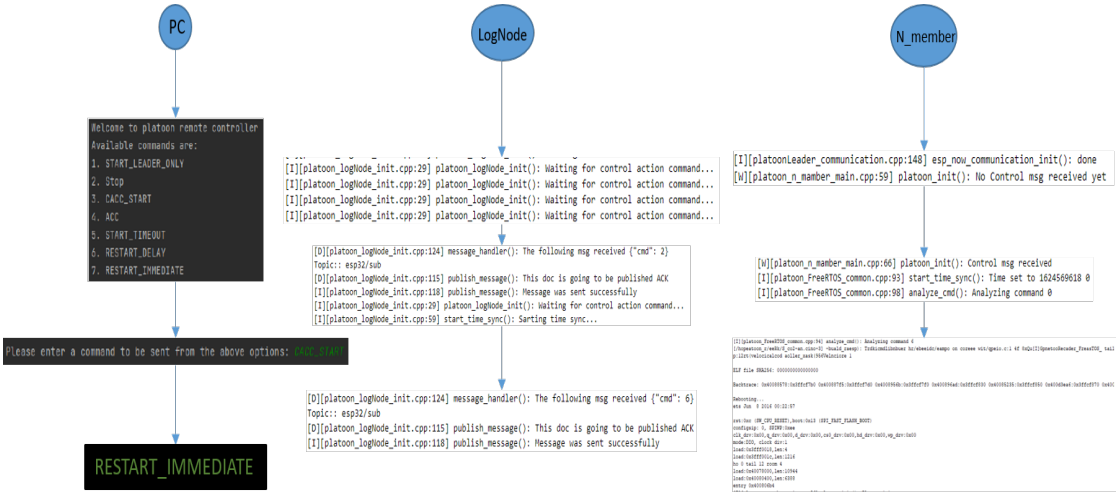


Figure 4.7.: Captured remote commands test.

5. Conclusion and Future Improvements

The aim of this project work is to build a stable testbed to experiment different Autonomous Vehicles platooning techniques and strategies either for ACC, or CACC systems.

The work was divided into three main phases which are: Design, Implementation, and evaluation. Firstly, the design phase. In section.1.3 a set of goals were defined to determine hardware, and software requirements. Accordingly and after a careful investigations a set of components were selected, which were matching the pre-defined goals and specifications.

Secondly, the implementation phase, which includes model hardware construction as well as software implementation as illustrated in chapter.3. On one hand the hardware construction includes; electronics soldering, wires routing, car model and track modifications and preparations. On the other hand, the software implementation part, which includes selecting a suitable development environment and building system which should be easy, flexible, and efficient, developing a set of software modules which should handles communication, motion analysis and estimation, sensors data handling, motor control, and task scheduling.

Thirdly, the evaluation phase. Each part of the software and the hardware was separately tested to ensure that all components are working as expected before integration. The hardware was tested after being assembled and it was working properly. Furthermore, wireless communication was evaluated in terms of packet loss, messages send/receive latency, remote action command, time synchronization, and data logging as presented in section.4.4, section.4.5 and Fig.3.30. Moreover, software performance-like analysis was done to validate whether the controller update frequency is preserved or not; the maximum latency was varying from 16 - 20 ms, although these values are in the safe-side, yet it still critical, and needs to be double-checked after integrating the CACC control law.

Finally, Motion estimation algorithms, which are responsible for own vehicle's motion analysis, and preceding vehicle's relative motion analysis was evaluated -on a software basis- using MATLAB; Through this test, the main affecting parameters on the estimation quality were identified, with the prescribed assumptions and approximations in section.4.1 and section.4.2. The estimated plots for the relative motion handler algorithm are promising and shows satisfactory results, however, a software filter such as Extended Kalman Filter can be added to the acceleration part for better estimation quality. As for the motion analysis within the same vehicle algorithm, the estimated velocity profile shows a delayed-like trajectory, which means that further effort should be done to reduce

the sampling interval(i.e., increase the sampling points). As for the acceleration, the values are quite low, meaning that any noisy values will have a huge impact on the actual values, therefore, a software filter can be integrated as well, or using a dedicated IMU sensor for acceleration measurements and avoid estimating it.
To sum-up, the model has reached most of the pre-defined goals. The research questions presented in section.1.2 can be answered as follows:

1. A small-scale experimental platform has been designed and implemented successfully with the giving funds.
2. It can be remotely monitored, and controlled via the implemented techniques.
3. It is capable to run different control strategies associated with ACC, and CACC systems.
4. With the implemented communication strategy, it is capable to perform different CAM message passing schemes.

As mentioned in the abstract, the model still has big room for improvement. In the below Fig.5.1 listed the possible improvement areas. Also, a detailed description is provided afterward.

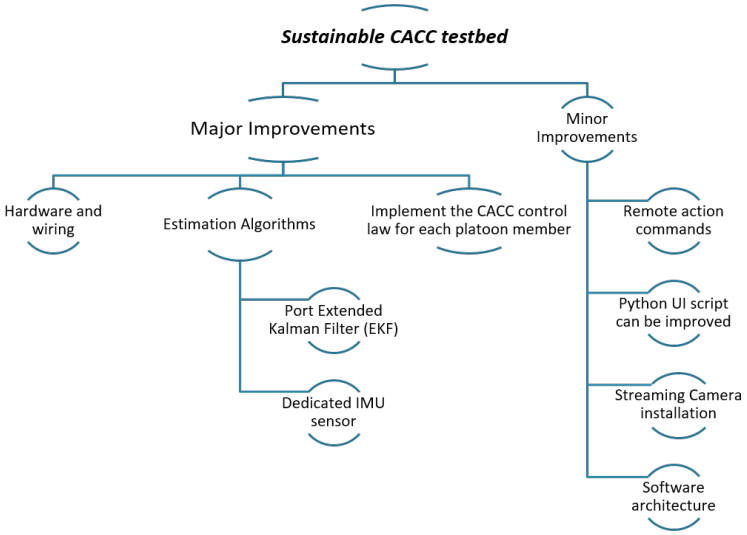


Figure 5.1.: Future improvement areas.

1. Hardware and wiring: The current hardware status is not as desired (see Appendix.B for further information), it could be improved by designing a simple PCB, and take off most of these wires, additionally, it is better to replace all these lengthy wires with more suitable ones, remove all workaround and replace them with new components. (8/10)

2. Software architecture and the coding part can also be improved by implementing C++ classes instead of just using functions and create self-contained modules. (4/10)
3. Port Extended Kalman Filter (EKF), integrate it in the software and compare between the currently implemented algorithms, EKF, and combined. (7/10)
4. A dedicated IMU sensor can be added for acceleration measurements and avoid estimating it using software, however, the sensor must have a suitable accuracy and refresh rate as mentioned before. (5/10)
5. Implement the CACC control law for each platoon member and add it to the platoonSDK.h file for easy modifications. (9/10)
6. The model is not fully tested due to time limitation, each part was separately tested, thus a complete test is still required. (7/10)
7. Remote action commands are not fully implemented. (5/10)
8. A streaming camera can be installed in the LAB to remotely monitor the model. (3/10)
9. Python UI script can be improved by implementing functionality for automatic data plotting upon receive for each vehicle based on values placed in each file (e.g., directly plot the values dedicated for leader vehicle which are found in "leader logged data" file. (5/10)

A. Appendix

Wi-Fi, ZigBee, and BLE specifications

– Layers illustration:

In this section, an elementary illustration is presented in Fig.A.1 and Fig.A.2 for each communication protocol layers including Hardware and Software layers in order to get the actual meaning of communication stack and how data is passed for each protocol.

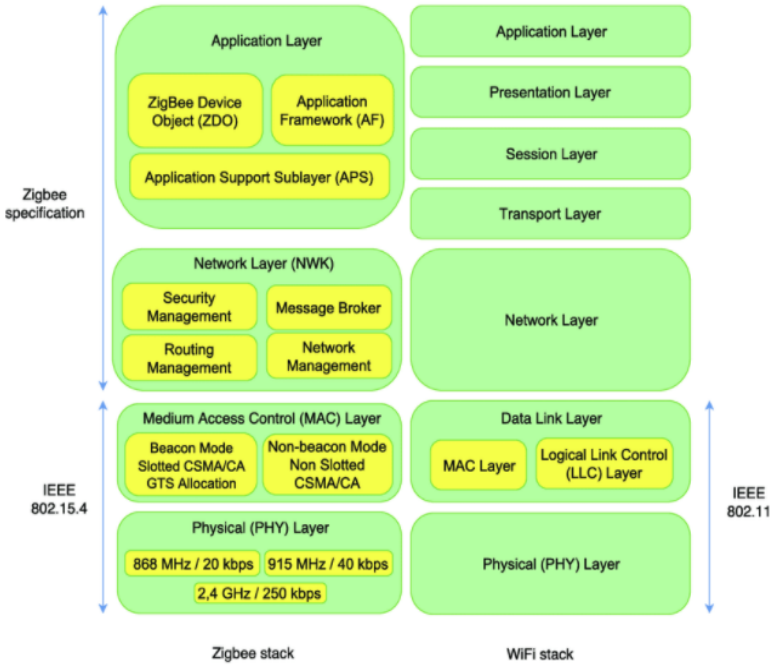


Figure A.1.: ZigBee and WiFi stack [13]

.

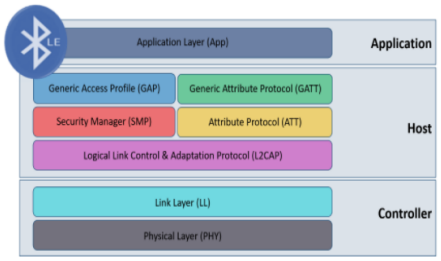


Figure A.2.: BLE stack[14]

– Frame structure:

To fully understand the differences between each protocol and to complete the layers part, frame structure should be mentioned, therefore, each protocol frame structure is presented in Fig.A.3, Fig.A.4, and Fig.A.5. When a frame is passing through the previously presented protocol layers an additional data is added until reaching the physical layer and transmitted afterwards.

1.Wi-Fi

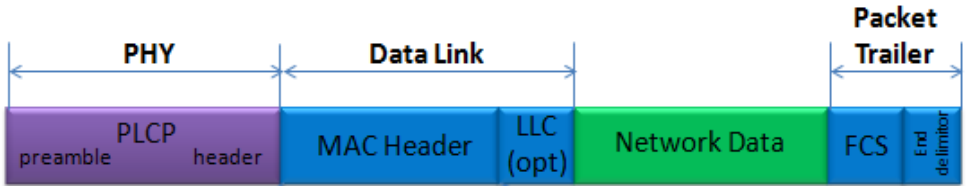


Figure A.3.: WiFi Frame structure[15]

2.ZigBee

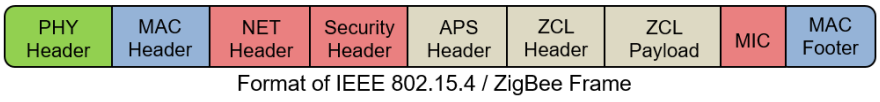


Figure A.4.: ZigBee frame structure[16]

Figure A.5.: BLE frame structure[17]

B. Appendix

Issues and problems in Hardware

During the assembly process several issues appeared in the hardware, mostly was due to usage of lengthy wires, a huge time wasted on trying to develop a customized wires by cutting and soldering these wires, however, this was time consuming approach, and due to time limitation was not implemented in all vehicle.

Another issue is that the motor driver requires maximum 11 voltage, while the available voltage levels are 12,5, and 3.3, so an additional voltage regulator is add to provide a 10 voltage level.
During the soldering, assembly and testing process 4 power converters were damaged 2 of them was due to human error, the other 2 was due to issue with soldering, however, a workaround is implemented in 2 vehicles, one vehicle has an extra 5V voltage regulator to compensate the damaged channel in the power converter module.
As for the other vehicle both the 3.3 and 5 voltage channels were damaged, so 2 extra voltage regulators were soldered in the power converter PCB ad depicted in Fig.B.1, accordingly the voltage level has changed from 5, and 3.3 voltage to 5 and 10 v voltages to avoid adding the extra voltage regulator for motor supply.

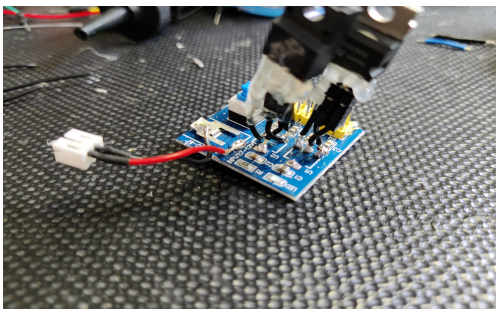


Figure B.1.: Power converter workaround

After fully assembling the last vehicle, the controller was damaged due to uncovered wire attached to the 10 voltage bus, this wire was a customized wire, and this was done due the problem mentioned previously, so the vehicle has been fully disassembled and the controller was replaced by a new one.

List of Figures

List of Tables

Bibliography

[1] Marie Christiano. *"ZigBee vs Bluetooth and Bluetooth Smart." AllAboutCircuit, September 09, 2015*

[2] Daniel Plöger. *"ROBUS T COMMUNICAT ION FOR CONTROL OF ROBOT IC VEHICL E P LATOONING." Institute of Communication Networks Institute for Technical Logistics, TUHH, July 17th, 2017*

[3] Maider Larburu, Javier Sanchez, and Domingo José Rodriguez. *"Safe road trains for environment: Human factors aspects in dual mode transport systems." In: ITS World Congress, Busan, Korea (2010), pp. 1–12.*

[4] Jennifer Chu. *"Driverless platoons: Analysis finds autonomous trucks that drive in packs could save time and fuel" MIT News Office Date, December 20, 2016*

[5] Darren Cottingham. *"What is vehicle platooning?" DT Driver Training Ltd.*

[6] Kris Kobylinski. *"What is adaptive cruise control and how does it work?" TomTom International BV, Nov 28, 2019*

[7] Keith Canete. *"What Is Adaptive Cruise Control?" Caranddriver*

[8] Gerrit Naus, Ren´e Vugts, Jeroen Ploeg, Ren´e van de Molengraft, Maarten Steinbuch *"Cooperative adaptive cruise control, design and experiments." In: Proceedings of the American Control Conference, August 2010, 10.1109/ACC.2010.5531596*

[9] Chi-Ying LIANG and Huei PENG *"String Stability Analysis of Adaptive Cruise Controlled Vehicles." Department of Mechanical Engineering and Applied Mechanics, University of Michigan.*

[10] Jeroen Ploeg, Bart T. M. Scheepers, Ellen van Nunen, Nathan van de Wouw, and Henk Nijmeijer. *"Design and experimental evaluation of cooperative adaptive cruise control." In: 14th International IEEE Conference on Intelligent Transportation Systems (ITSC) (2011), pp. 260–265.*

[11] D. Swaroop. *"String stability of interconnected systems: An application to platooning in automated highway systems." Tech. rep. 1. Berkeley: University of California, 1997, p. 65. doi: 10.1016/ S0965-8564(97)88297-3.*

[12] *"Intelligent Transport Systems (ITS);Vehicular Communications;Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service." In: The European Telecommunications Standards Institute 1.3.1.1 (2014).*

[13] Iván Froiz-Míguez, Tiago M. Fernández-Caramés * , Paula Fraga-Lamas * and Luis Castedo. *"Design, Implementation and Practical Evaluation ofan IoT Home Automation System for Fog ComputingApplications Based on MQTT and ZigBee-WiFiSensor Nodes"*

[14] Olivia. *BLE Protocol Stack — Host Controller Interface (HCI)*

[15] Share Technote. *WLAN - Frame Structure*

[16] Mathworks article. *IEEE 802.15.4 - MAC Frame Generation and Decoding*

[17] Jiun-Ren Lin, Timothy Talty, and Ozan K. Tonguz. *"On the Potential of Bluetooth Low Energy Technology for Vehicular Applications." IEEE Communications Magazine, January 2015*

[18] AshishBansal, SHUBHAMSINGH10, and nitin mittal. *Program for Round Robin scheduling*

[19] Scalextric. *Racing cars manufacturer*

[20] Carrera. *Racing cars manufacturer*

[21] Espressif Systems. *ESP32-DevKitC*

[22] NVIDIA Developer. *Jetson Nano Developer Kit*

[23] WAVESHARE. *ZigBee Module, Features The CC2530F256 Onboard, XBee Compatible Interface*

[24] JACK Enterprises. *PMW3360 Motion Sensor*

[25] STMicroelectronics. *Time-of-Flight ranging sensor*

[26] Arduino. *Ultrasonic Sensor HY-SRF05*

[27] u Blox. *MQTT beginner's guide - Tech, 16 June 2020*

[28] LEDMO store. *LEDMO AC adaptor*

[29] POLOLU Robotics and Electronics. *DRV8838 Single Brushed DC Motor Driver Carrier*

[30] Saleae. *Saleae Logic 8*

[31] Gitlab-TUHH. *cacc-testbed repository - WiKi*

[32] Espressif Systems. *ESP-NOW communication protocol*