Universidade do Minho
Mestrado Integrado Engenharia Eletrónica Industrial e Computadores
(MIEEIC)

Embedded Systems

REMOTE CONTROLLED SONAR CAMERA

-Project Analyse-

Authors:
João Sousa nº82273
Pedro Sousa nº82041

Professor:
Adriano Tavares
José Mendes

January, 2019-2020

Contents

List of Figures

List of Tables

1. Problem Statement

The impact of technology evolution on humanity has increased very fast. The use of technology is spreading through all the tasks of our professional and personal life. As so, assuring our security and safety it's one of our main concerns.

With our project we intend to improve the cameras of today. For example, most car's back cameras, used for parking, can't measure distance precisely and that could result into serious problems. We design to make a remote-controlled camera that can measure distances accurately of close objects. That's why we call our project "RCSC"- Remote Controlled Sonar Camera.

Our project would improve not only car's cameras, but in robots it could be used for a good environment view and object collision.

1.1 Problem Analysis

The problem above described needs to be deconstructed and analyzed in more detail. It is mandatory to define Entities, as well as their functions and correlations.

After the statement core of our problem, we encounter 5 main subsystems: the "Device System", the "Remote Station", the "Sensors & Actuator" the "User" and the "Interface".

- The "User" views the video transmitted and remote controls the motors through the "Interface",
- The "Interface" mainly shows the video, the radar and the controls to the motors. It receives and sends data to the "Remote Station".
- The "Remote Station", works as a bridge of Wi-Fi communication and data storage between the "Device System" and the "Interface", increasing the project flexibility.
- The "Device System" is the main processor and controller. Receives data from "Sensors" and sends it to the "Remote Station". Receives from the "Remote Station" the defined controls set by the "User" and based on them actuates on the "Motors"

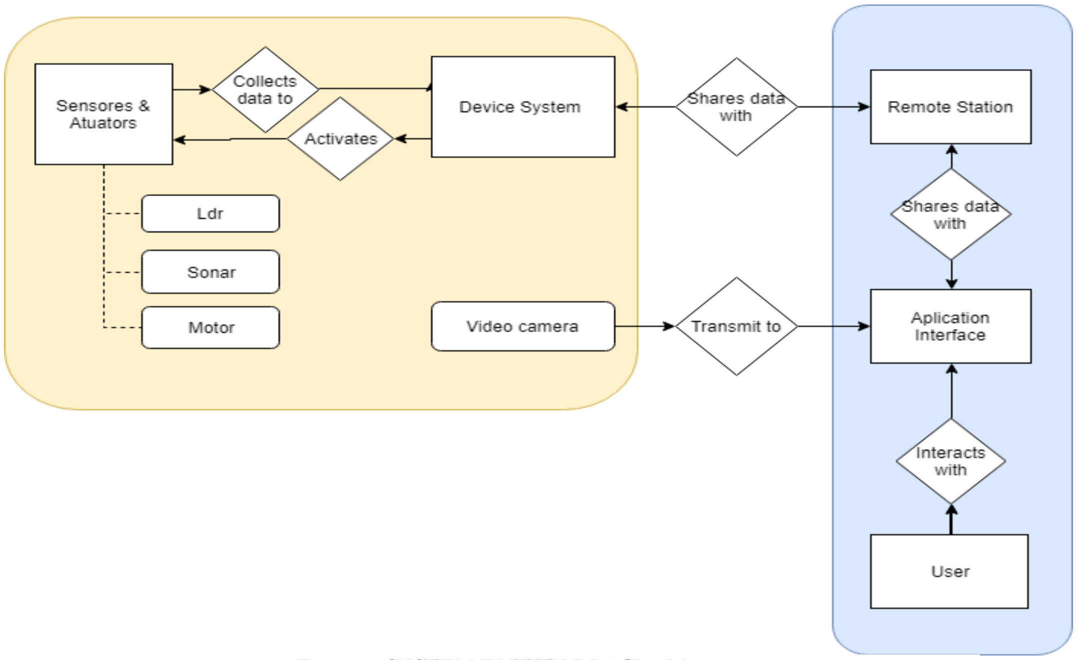


Figure 1 - SYSTEM ENTITY DIAGRAM

2. Analysis Phase

On this phase our project we identify the overall direction that the project will take through the creation of the project. Gathering the requirements and constraints is the main attraction of the Analysis Phase, such as defining the system overview and the system architecture.

2.1. Project

To define the project and present a well structure overview of the system, its necessary to define the requirements and constraints parameters.

2.1.1 System Overview

Based on the analysis above the project will be composed by 1 microcontroller, 3 sensors and 1 actuator:

- The STM32 – The main controller of the application. It gathers data from the sensors, activates the motors and communicates with the remote system. Reads from the Remote Station Server the data inputted by the User and based on them controls the motors.
- The Sensors – collecting useful data for the System
  - Sonar - to radar scan the environment;
  - LDR - detect day or night through light;
  - ESP8266 – establish Wi-Fi communication;
- The Motors – 2 servomotors that allows to move the camera/sonar (coupled to the motors) in a horizontal and vertical axis.

The Microcontroller after gathering the data from sensors, sends it to the remote station through the help of a Wi-Fi module. It also receives from the remote station the data to control the motors. As so, there is an external application running the Interface that allows the User to remote control the camera/sonar direction and shows a radar of the environment.

However, due the system limitation, the camera shall be external. Directly connected to the device of the interface for the video sharing.

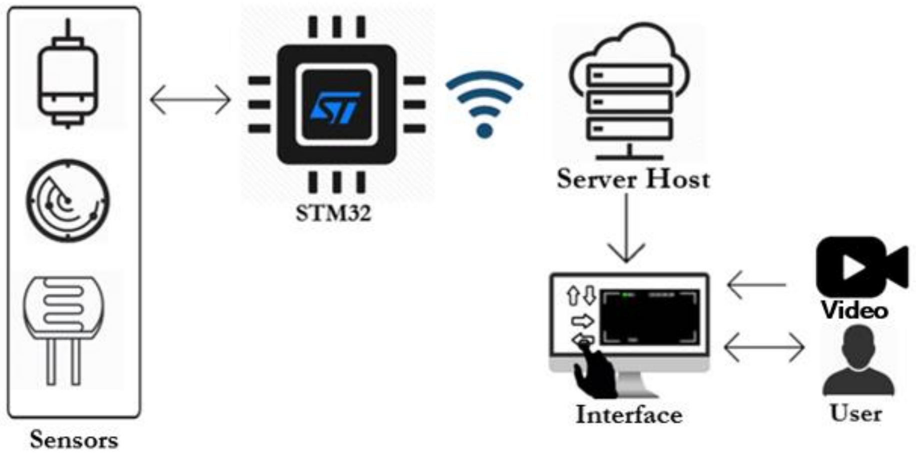


Figure 2- System Overview Diagram

2.2. Requirements and Constraints

In order to the better predict a good performance from the final product, its mandatory to establish all the requirements and constraints implied.

2.2.1 Non-Technical Constraints

Represents all the limitations that exist apart from the practical development of the project. In this case, the project group must:

- Not surpass the final deadline;
- Maximum of two students per group;
- Be budget friendly duo to money limitations;

2.2.2 Technical Constraints

Represents all the established limiting factors that affects the execution of a project. In this project, its implementation requires:

- STM32F767ZI;
- Real time system – FreeRTOS;
- Keil MKD-ARM (uVision) as Run-Time Environment;
- 3 non-identical sensors;
- Machine-Learning and/or adaptive control;

2.2.3 Functional Requirements

It defines all the specific functionality that defines what the system will accomplish. As so, this project must:

- Scan the environment;
- Move the camera and sonar direction;
- Communicate with remote station server;
- Allow the User control of the direction;
- Display radar scan and video;

2.2.4 Non-Functional Requirements

It defines all the quality attributes we pretend our system to have. As so, we design it to be:

- Friendly user and coherent interface;
- High flexibility;
- Low cost;
- Sturdiness;

2.3. System Architecture

As any other good project, it starts on a good fundamental well-constructed Architecture. Making the connection between the hardware and the software, is the essential key to achieve a certain state of quality.

2.3.1 Hardware Architecture

The System hardware consists on a station of sensors connected to the STM32, the main processor/controller. The STM32 connected indirectly to the interface, where they exchange data. The interface allows the control of the motor (changing the camera/sonar direction) and displays the sonar inputs and the video. The diagram bellow shows a simple representation of the Hardware.

2.3.2 Hardware Specification

- STM32F767ZIT6 board – The main controller of the application for data processing and data transmission:
- Light sensor – to get the light luminosity of the environment. Simple GPIO implementation;
- Servo-Motors – to aim the camera to the desired location. The system can read the motors to know its angular position;
- Camera – for capturing image/video through USB connection.

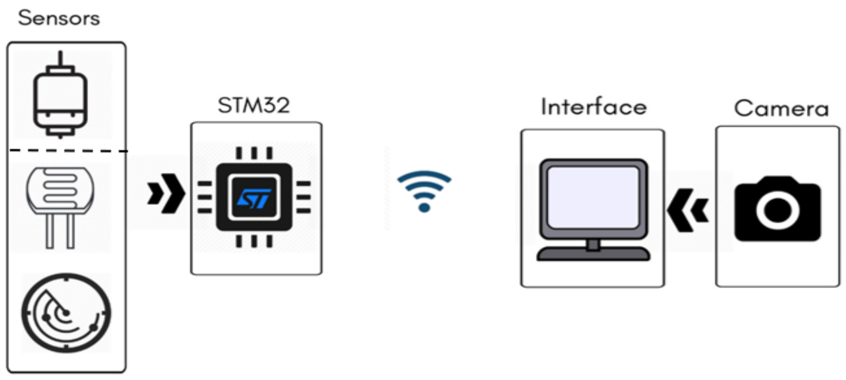


Figure 3 - Hardware Architecture Diagram

2.3.3 Software Architecture

In terms of the software, our system requires some specific drivers for the STM32 establish communication with the sensors and the remote station, and to actuate on the motors. The Middleware covers the acquisition and processing of the data and the communication through specific ethernet libraries. In Application there is the Graphic Interface showing all the data gathered, and the control of the Camera/Sonar Tower.The diagram to the side shows a simple representation of the Software.

2.3.4 Software Specification

- FreeRTOS – Real time Operation System;
- Keil µVision - programming IDE;
- C and C++ programming language
- Qt creator - for build the interface;

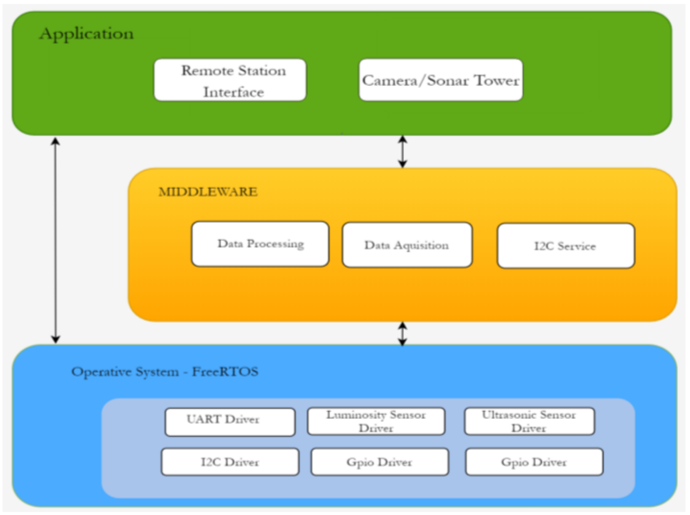


Figure 4 - Software Architecture Diagram

2.4 Local System

In our system, RCSC (Remote Controlled Sonar Camera), there are two primary actors (the User and Time) and three secondary actors (Motors, Camera and Sensors). The User must be capable to set the program on automatic or manual mode The System also has the possibility of an adaptive control, meaning two different modes:

2.4.1 Manual mode:

As we see bellow on diagram, when the user interacts with the interface, the system receives commands from webpage application to control the motors allowing the user for the manual analysis.

With the triggering of Time, the system samples camera image and reads the environment. After that, the system groups the collected information, processes it and sends it to the webpage application. The only effect of the User in the system is to move the camera angle through control of the Motors.

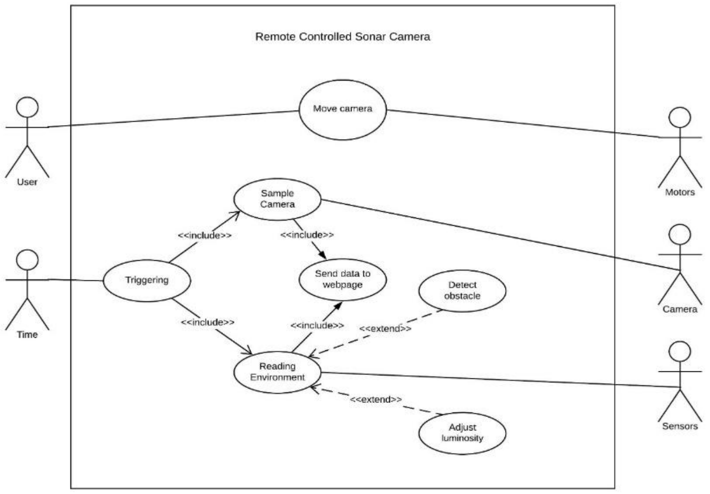


Figure 5 - System Use Case Diagram in Manual Mode

2.4.2 Automatic mode:

The automatic mode differs in the control of the movement of the camera. Here, the system is autonomous, where it scans the environment and adapts the position of the camera pointing to the closest object. In this diagram, the User no longer makes part as one of the actors, relying only on the rest of them the good performance of our system.

Here, Time is the principal actor, causing the trigger to read the sensors and read the sample video from camera. When the Sonar Sensor detects an obstacle, it performs an adaptive control with the information gathered, operation on the motors to make the camera follow the obstacle. Then, when all information is processed, the local system sends it to the webpage application.

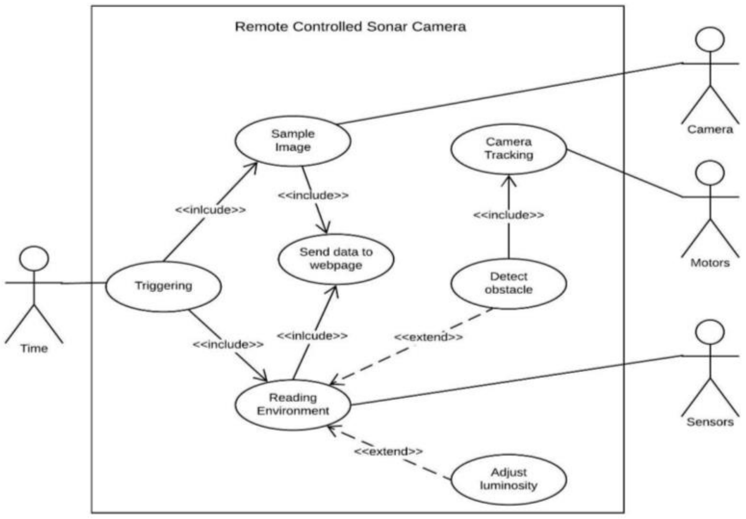


Figure 6 - System Use Case Diagram in Automatic Mode

2.4.3 System sequence

As shown in use cases diagram above, "Time" is the main actor on the device system. On sample period, that it's triggered by "Time" (synchronous event), it requests sample of the sensors (sonar, ldr and motor position) and streams the camera video. The Local system request samples to the sensors and if the response is accepted the data will be analyzed, but if was not, the "Local system" will request again until the answers be acceptable. After collecting and analyzing the data, the system request access to communicate with "Remote System" (webpage) and sends the information to it and. If the system is in Automatic mode, it will adjust the angle of the camera based on the analyzed data

.

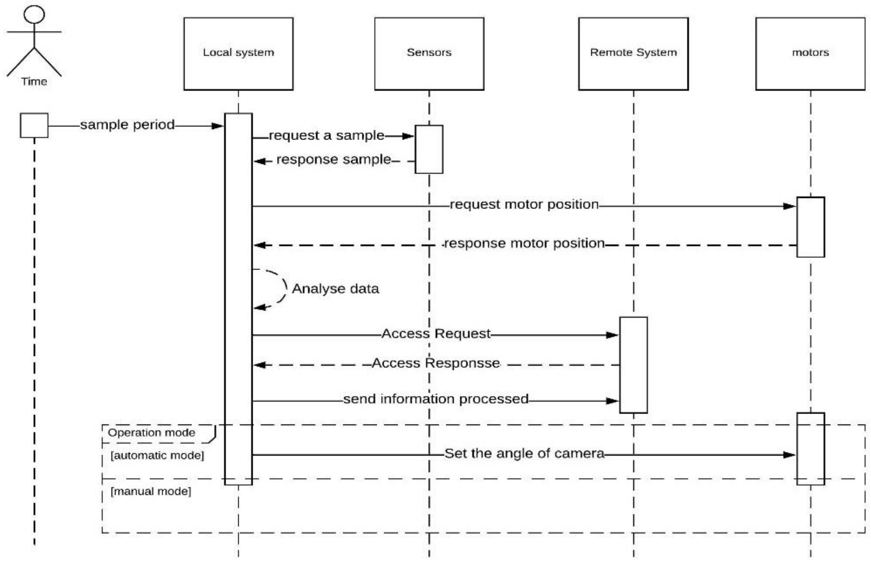


Figure 7 - System Sequence Diagram with Time as Actor

With this second diagram we pretend to show what happened when the User interacts with our system (asynchronous event). When the user press on one of the direction buttons (UP, DOWN, LEFT or RIGHT) of the interface, the "Remote system" request access to the "Local system" and sends the "user command". Then the local system translates the command to activate the servo-motors to move the camera in the chosen direction.

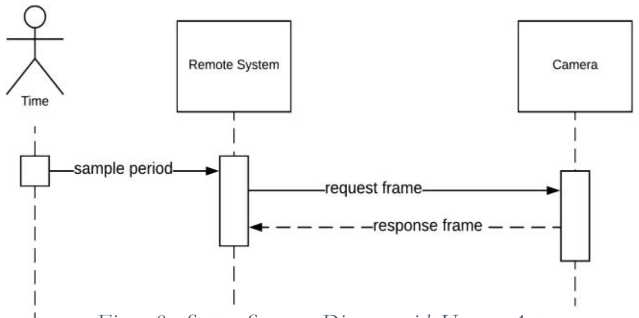


Figure 8 - System Sequence Diagram with User as Actor

Other process is the streaming of the video. There is refresh rate period in which the remote system request to the camera the video frame, to show on the interface.

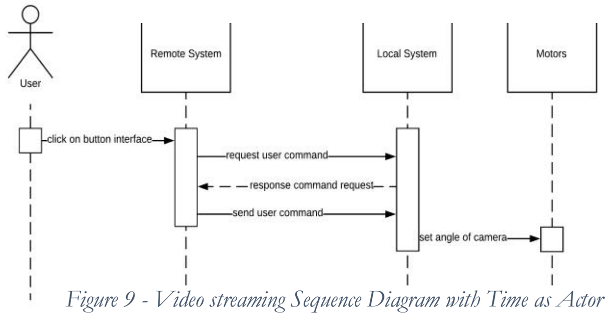


Figure 9 - Video streaming Sequence Diagram with Time as Actor

3. Design Phase

On this phase, we decrypt the fundamentals that was described on last phase. We intend describe the product components, module and parts in greater detail. At first, we going to introduce some bases on some theoretical concepts that are important to understand the contents covered on this project.

With the learned subject of class, we intend to meticulously cover all of software and hardware topics and respect its features.

3.1. Hardware Specification

3.1.1 STM32F767ZIT6 board

Our main development board is the STM32F767ZI and has an ARM®32-bit Cortex®-M7 CPU with an FPU (floating point unit), able to speed up the calculations, improving the processing time required to the operations.

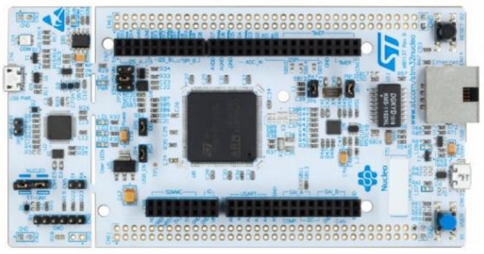


Figure 10 - STM32F767

3.1.2 Servo Motors

The 2 servo motors allow the angle adjustment of the structure in order to follow an object, rotating in the two axis (horizontal and vertical axis). The SONICMODELL Servo Motor is a tiny and lightweight motor with high output power and has metal gears for added strength and durability.
The servo has three wires. Two are for power and the third wire is for signals to position the servo. The signal wire expects input from a pulse width modulator.
The period should be 20 milliseconds long and the duty-cycle encodes the position of the motor. If the duty cycle is 1.5 millisecond, the servo is positioned at 0 degrees. If the duty-cycle is 2 milliseconds, is all the way to the right (90 degrees). If the duty cycle is 1 millisecond, the servo is all the way to the left (-90 degrees).

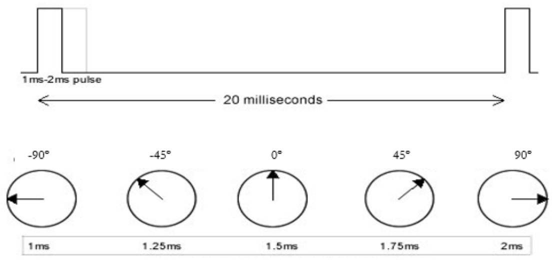


Figure 12 - PWM period cycle

Figure 11 - Servo Motors

3.1.3 DC POWER and Battery

For powering the STM32F7 a 5V DC Powerbank is used. The camera and the sensors are then powered by the board. The motors powered by another 5V DC converter to avoid current overload and damage the board. In order to provide enough current to the board and the sensors the board must at least have 2A:

- Output voltage: 5Vdc voltage and capacity up to 2500mAh;2 USB outputs;
- 2 USB outputs (only one is necessary)

Figure 13 - Powerbank

This is a simple AC-DC converter that has an alternate current (AC) input tension of 230 V from the power grid and a direct current (DC) output tension of 5V. The AC-DC that we will use has the following characteristics:
• Input voltage: 100-240Vac and between 50/60Hz;
• Output voltage: 5Vdc voltage and capacity up to 2500mAh;
• Protection against overload and over voltage.

Figure 14 - AC/DC converter

3.1.4 Luminosity Sensor – LDR

The system will collect data about the ambient luminosity, and to do so, it will use this Light sensor module based on photodetector GL5528 to detect the light intensity of the environment. As the resistance of the sensor varies depending on the amount of light it is exposed to, the output voltage changes with the light intensity. This module outputs both analog and digital signals, which can be used to trigger other modules. Also, the potentiometer can be used to adjust the sensitivity of digital output. The digital terminal output HIGH when the light intensity exceeds the value set by potentiometer and vice versa.
The outputs of analog terminal increase with the light intensity. Features:

- Working voltage: 5V
- Output signal: Analog and Digital;

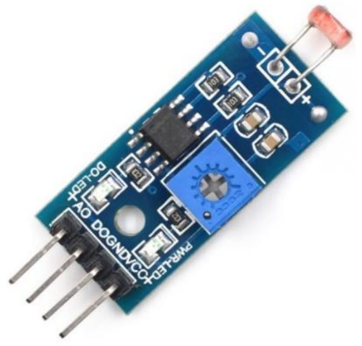


Figure 15 - LDR module

3.1.5 HC-SR04 Ultrasonic Sensor

The sonar uses sound waves to measure the distance of an obstacle. To implement object detection and to measure distance we will need a HC-SR04 Ultrasonic Sensor module.

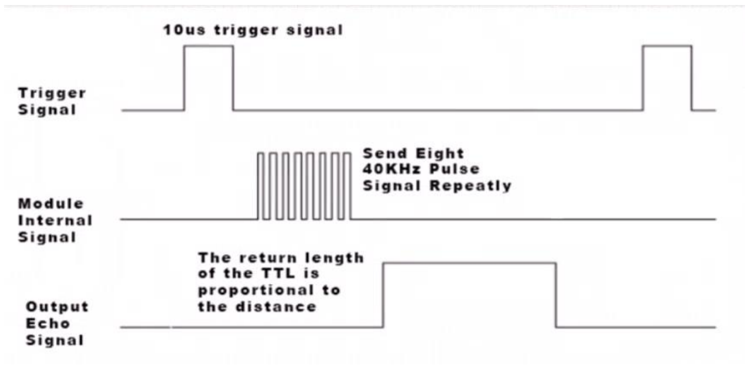
Specifications:

- Power supply: 5V DC
- Operation current: 2mA
- Effect angle: 15º
- Range: 2cm – 4m
- Precision: 3mm

Figure 16 - HC-SR04 sensor

It is composed by a transmitter and a receiver. The transmitter sends ultrasonic sensor with the specific sensor frequency, non-audible for humans, and part of this wave will be reflected in an object and retrieves it back to the receiver.

With this mechanism we can measure the time that the wave spends until achieve the obstacle and back. After that, we can finally, by software calculate the distance, we just need to multiply the time by the velocity of sound in air (340m/s) and split all by two, since wave go through the same distance two times.


10us trigger signal
Trigger Signal
Send Eight 40KHz Pulse Signal Repeatly
Module Internal Signal
The return length of the TTL is proportional to the distance
Output Echo Signal
Figure 17 - Sonar pins signal

To enable the measurement, it´s necessary to put in High state the pin trigger for ten seconds. After the module emits a signal, when an object is detected and the signal reflected, the pin ECHO will be in High state for a period proportional to the distance between the sensor and the obstacle.

3.1.6 Esp8266 module

In order to establish a remote communication with our system we need a module that be responsible for that. The ESP8266 module, it is a self-contained SOC with integrated TCP/IP protocol stack that can give any microcontroller access to our Wi-Fi network.

The ESP8266 is capable of either hosting an application or offloading all Wi-Fi networking functions from another application processor. Each ESP8266 module comes pre-programmed with an "AT commands set" firmware, meaning, you can simply hook this up to your Microcontroller device and get about as much Wi-Fi-ability as a Wi-Fi Shield offers.

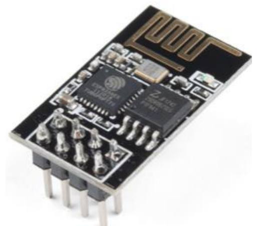


Figure 18 - WiFi Module

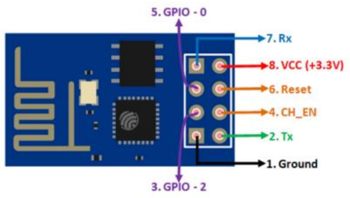


Figure 19 - WiFo module pinout

3.1.7 Pan & Tilt

The structure of the application we based on a Pan-Tilt model, where the two servo-motors will be coupled. It will enable the orientation of camera in 2 axis, necessary for the object tracking (aiming the camera in the object direction). Specifications:

- PWM driver: PCA9685
- PWM resolution: 12-bit
- Ambient light sensor: TSL2581FN
- Ambient light resolution: 16-bit
- Communication interface: I2C
- Operating voltage: 3.3V/5V
- Dimension: 56.6mm x 65mm

Figure 20 - Pan & Titlt

The communication can be performed by I2C. However, it's possible to use the individual pins to control the motors individually, and the light sensor will be implemented with a different module. So, the I2C interface pins will not be connected to the board.

3.1.8 Structure

In order to compact all the components in one structure with all the required software and hardware, it was designed the following 3D model. In the bellow area is compacted the STM board, the sensors and the Power supply. It shall have all the entries necessary for the cabling, cooling and have enough visibility for the sensors. In the above part there is the tilt and pan structure for the direction control of the camera and ultrasonic sensor.

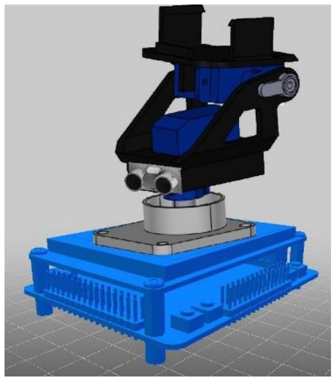


Figure 21 - Model Structure

3.2. Hardware Peripherals and Communication Protocols

These protocols are responsible for the interaction and the stability of the communication between the hardware and our board (STM32). For different hardware, in our case, we have different protocols, for instance, the local system has 1 UART, 1 DCMI, 2 PWM, 1 GPIO and 1 ADC. The UART is used to receive and send information via wifi with the WiFi module. The DCMI protocol is used to connect a parallel camera module to the STM32. In order to retrieve information from two analog devices a ADC is needed as it is possible to see in the LDR. The Sonar works with a digital function that is connected to a GPIO to warn the warn the local system the distance of the detected object.

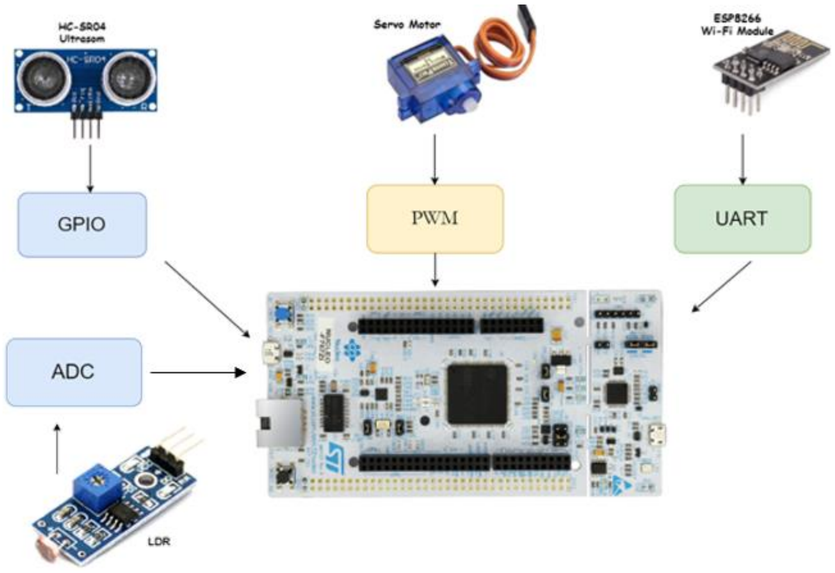


Figure 22 - - Hardware Peripherals and Communication

3.2.1 PWM Protocol

Pulse Width Modulation is a method of reducing the average power delivered by an electrical signal, by effectively chopping it up into discrete parts. The average value of voltage fed to the load is controlled by turning the switch between supply and load on and off at a fast rate. PWM is be used to control the servo motors, due to the need of a duty cycle.

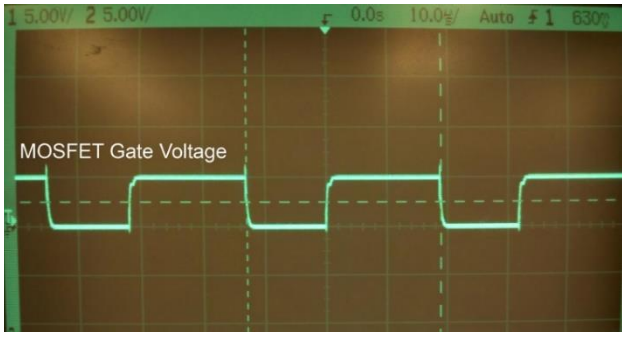


Figure 23 - PWM output wave

By controlling the time that signal spend in high state over a consistent interval, the percentage of time ON relative to the percentage of time to off, is called by duty-cycle. The signal can only in the range of 5V(high) or ground(low).

3.2.2 UART protocol

The UART (Universal Asynchronous Receiver /Transmitter) is a peripheral used for full duplex serial communication that is presented in our board. It is a single LSI (large scale integration) chip designed to perform asynchronous communication. This device sends and receives data from one system to another system.

In UART communication, two UARTs communicate directly with each other. Basically, the transmitting UART converts parallel data into a serial form, and the destination receiving UART converts the serial form to a parallel data. The data flows from the Tx to the Rx pin of the receiving UART.

UARTs transmit data asynchronously, which means there is no clock signal to synchronize the output of bits from the transmitting UART to the sampling of bits by the receiving UART. Instead of a clock signal, the transmitting UART adds "start" and "stop" bits to the data packet being transferred. These bits define the beginning and end of the data package so the receiving UART knows when to start reading the bits. In our system will be used to communication between the board and the wi-fi module.

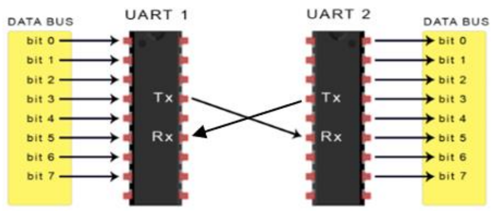


Figure 24 - UART connectioin logic

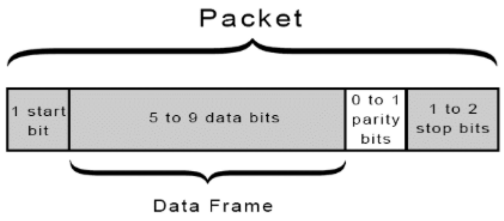


Figure 25 - UART data packet

3.3. Software Specification

In this chapter, be define and mention specific software modules, tools and applications, as it is necessary to have a description of the software that's going to be developed. As so, this chapter should provide an inside out of the work and structure that needs to be followed to have a good implementation of the project.

3.3.1 Tools

In order to develop the project, some tools are needed for training the neural network, an IDE to code our program (and its language and libraries), and other applications. For this project it will be used the following ones:

- Keil _Vision5 – IDE (Interface Development Environment) for programming for the STM board with capability for debug.
- ST-LINK - Debugger on chip tool, that allows to follow the state of the board while debugging
- the code.
- STM32CubeMX - Configuring tool for STM board. Generates code for the initialization and
- some configurations of the board's peripherals.

3.3.2 COTS

Commercial-Off-The-Shelf (COTS) are packaged solutions which are then adapted to satisfy the needs of the project. Since there are some APIs needed to develop the project, some COTS are essential. The ones used on this project are:

- FreeRTOS - real time operation system;
- HAL - abstraction layer for peripheral drivers;
- CMSIS-DSP - for signal processing

3.3.3 FreeRTOS

A Real Time Operating System is an operating system that is optimized to be used in embedded/real time applications. Their primary objective is to ensure a timely and deterministic response to events.

Using a real time operating system allows a software application to be written as a set of independent tasks. Each task is assigned a priority and it is the responsibility of the Real Time Operating System to ensure that the task with the highest priority is the one that is running.

To avoid race conditions, the OS also provides methods for thread synchronization/communication like mutexes, semaphores and software timers. More ahead, major tasks are presented and explained.

3.3.4 CMSIS-DSP

The idea behind CMSIS is to provide a consistent and simple software interface to the processor for interface peripherals, real-time operating systems, and middleware, simplifying software re -use, reducing the learning curve for new microcontroller developments and reducing the time to market for new devices (CMSIS library comes with ST firmware).

3.3.5 HAL

The HAL driver layer provides a generic multi instance simple set of APIs (application programming interfaces) to interact with the upper layer (application, libraries and stacks). It is composed of generic and extension APIs. It is directly built around a generic architecture and allows the built-upon layers, such as the middleware layer, to implement their functions without knowing in-depth how to use the MCU. This structure improves the library code reusability and guarantees easy portability to other devices.

The HAL drivers include a complete set of ready-to-use APIs which simplify the user application implementation. As an example, the communication peripherals contain APIs to initialize and configure the peripheral, to manage data transfers based on polling, to handle interrupts or DMA, and to manage communication errors.

Cube uses the STM32 hardware abstraction layer (HAL) library to create the initialization code, which makes it a lot easier to migrate between STM32 microcontrollers if needed.

3.4. Task Overview

To achieve a proper functioning of our system we need to establish the threads communication and how they reach synchronization. Therefore, this diagram includes the types of synchronization between the tasks, which are the Mutex, the Semaphore and the crossed information through the different tasks (Queue).

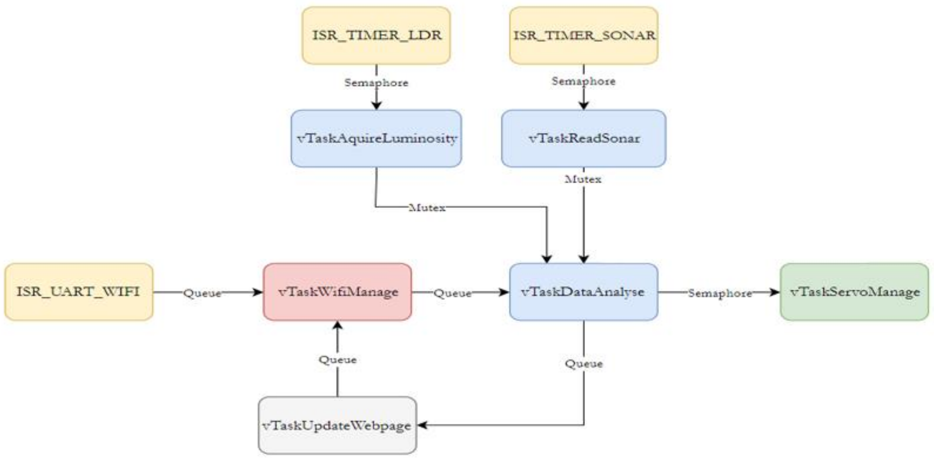


Figure 26 - Task Overview Diagram

3.4.1. Tasks

vTaskAquireLuminusity: This task is responsible to acquire the value of luminosity through ADC and saves it to a structure for later analysis.
vTaskReadSonar: This task is responsible to read sonar value and indicate the system if an object was detected. Whether an obstacle is detected the distance can be measured and saves the measurement into a structure for later analysis.
vTaskServoManage: This task will be responsible for actuate and set the angle of the Servo Motors.
vTaskDataAnalyse: After all of data is acquired, the system gathers and analyze all the data, signals to update the Webpage, and actuate on the servos motors based on the values read/chosen.
vTaskUpdateWebpage: This task will receive the analyzed information and update the file that contain the sensor's information on the webpage.
vTaskWifiManage: this task will manage all the communications via Wi-Fi, by sending requests and information to receive and analyze the incoming data.

3.4.2 ISR

ISR_TIMERLDR: This interrupt is responsible to alert TaskAquireLuminusity that can make the reading of the adc and acquire the luminosity values.

ISR_TIMERSonar: This ISR is responsible for to signal the gathering of samples of the sonar.

ISR_UART_WiFi: This ISR is responsible for alerting the system that a message was received from the webpage.

3.4.3 Task Synchronization

Sem_ActivateServo: This semaphore unlocks the *vTaskServoManage,* allowing the actuation of servo-motors.

Mutex_Luminosity: This mutex is responsible for the luminosity data protection, that is a shared resource between the *vTaskAquireLuminusity* and *vTaskDataAnalyse.*

Mutex_Sonar: This mutex is responsible for the sonar variables protection, that is a shared resource between the *vTaskReadSonar* and *vTaskDataAnalyse.*

3.4.4 Queue

Queue_DataToAnalyze: Send the inputs of user from *vTaskWifiManage* to *vTaskDataAnalyse* via queue.

Queue_UpdateWebpage: Send the analyzed data from *vTaskDataAnalyse* to *vTaskUpdateWebpage* via queue.

Queue_WifiManage: Send the file that build webpage from *vTaskUpdateWebpage* to *vTaskWifiManage* via queue.

Queue_RxWIFI: Send data received by the ISR_UART_WiFi interrupt to the *vTaskWifiManage* via queue.

3.4.5 Task priority:

In a system with a great number of tasks, each with its purpose, it's necessary to prioritize them in order to achieve our goal.

Not every task is equally important, some more than others. It is necessary to identify what is the most important task at any moment and give those tasks more attention, energy, and time. For our system we design these priority task pyramid diagram below. The reading sonar task was chosen as the most important tasks because in case a object appears instantly, or get closer more and more, the system must have to know about the sudden movement, and alert the User. The lower layer is presented by acquisition of environment luminosity just because is a task that don´t have most importance relatively to the purpose of our system.

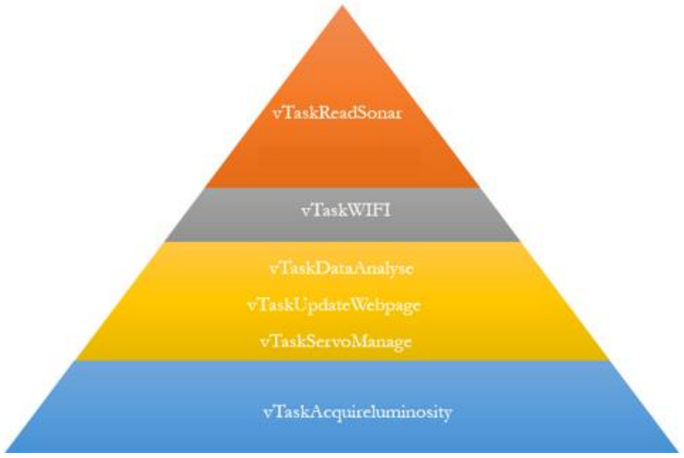


Figure 27 - Task priority

4.IMPLEMENTATION

4.1System Initialization

To start the project, the system first initializes all the process. In the diagram below is represented the initialization of all the modules. It goes by calling the respective init function of each module, so it can enable/init all the necessary timer, interrupts, adc, etc.

This is the first instructions accomplished in our system, before the system is prepared to work as pretended.

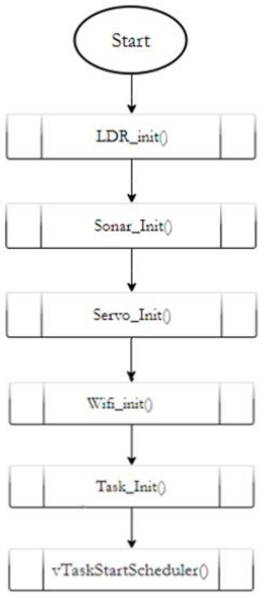
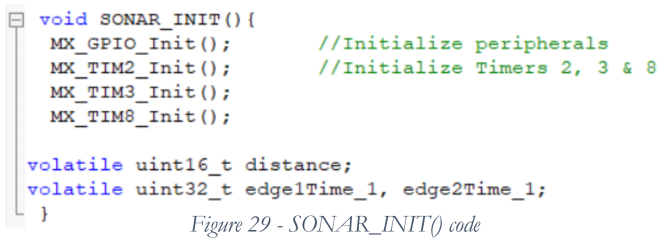


Figure 28 - Initialization Flowchart



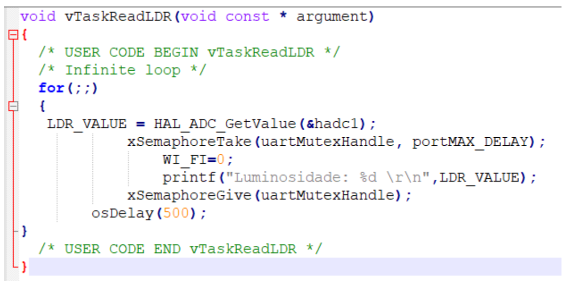
```
void SONAR_INIT(){
  MX_GPIO_Init();        //Initialize peripherals
  MX_TIM2_Init();        //Initialize Timers 2, 3 & 8
  MX_TIM3_Init();
  MX_TIM8_Init();

volatile uint16_t distance;
volatile uint32_t edge1Time_1, edge2Time_1;
 }
```

Figure 29 - SONAR_INIT() code

4.2Tasks Implementation

After the initialization, the system jumps to its normal operation, scheduling through task to task, running the main application. Here we describe a brief description on the implementation and the flow of actions on each task.

vTaskAcquireLuminusity: the task to acquire the luminosity, is set to the lowest priority. Every second it activates the HAL function to read the value of the LDR on the ADC(the reading its protected by a mutex);



```
void vTaskReadLDR(void const * argument)
{
  /* USER CODE BEGIN vTaskReadLDR */
  /* Infinite loop */
  for(;;)
  {
   LDR_VALUE = HAL_ADC_GetValue(&hadc1);
            xSemaphoreTake(uartMutexHandle, portMAX_DELAY);
                WI_FI=0;
                printf("Luminosidade: %d \r\n",LDR_VALUE);
            xSemaphoreGive(uartMutexHandle);
        osDelay(500);
}
  /* USER CODE END vTaskReadLDR */
}
```

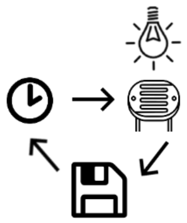
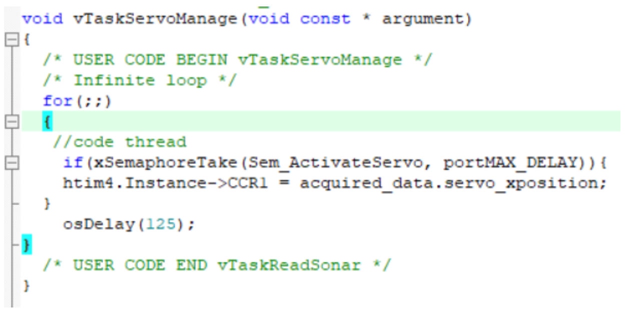


Figure 30 - vTaskAcquireLuminusity() code and schematic

vTaskServoManage: the task to control servos, is activated by a semaphore, so every 125 ms the task checks for the semaphore. Activated, the desire PWM for the desire angle is inputted for the motor



```
void vTaskServoManage(void const * argument)
{
  /* USER CODE BEGIN vTaskServoManage */
  /* Infinite loop */
  for(;;)
  {
   //code thread
    if(xSemaphoreTake(Sem_ActivateServo, portMAX_DELAY)){
    htim4.Instance->CCR1 = acquired_data.servo_xposition;
  }
    osDelay(125);
}
  /* USER CODE END vTaskReadSonar */
}
```

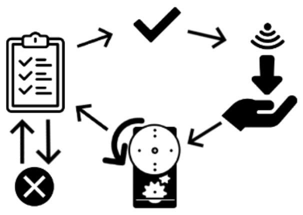


Figure 31 - vTaskServoManage() code and schematic

vTaskUpdateWebpage: the task to upload the webpage and the data, waits for a Queue telling it to update the webpage with new values or with a new webpage. When it receives the Queue signaling new data and webpage page information (which page to upload), it enables the function to print, through the UART4, the Webpage array with the specific commands to the Wi-Fi module, in order to update the webpage. The printing to the module must be protected (with mutex, for example) in order to avoid interruptions that lead to fails in the communication.

vTaskWifiManage: the task to manage the information received from the webpage and the information to send. It waits for a Queue from the analyze task (with information to send) and checks for any response from the parsing executed on the UART interrupt (the information received from the webpage, for example, a bottom selected). Based on the information it has, updates the Webpage array with the new information and signals the task UpdateWebpage through a Queue.

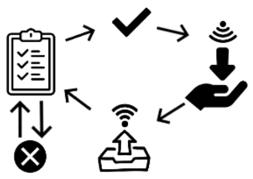
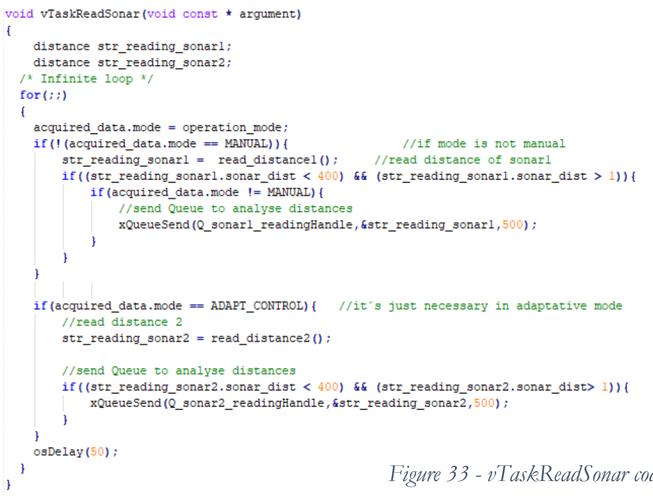


Figure 32 - vTaskUpdateWebpage schematic

vTaskReadSonar: the task to read the sonar, is set to operate each 50ms. First, it evaluates the operation mode (Manual or Adaptive Control). Based on the mode, it reads the specific sonar (Manual reads sonar 1, Adaptive reads sonar 2). With the distance value, if inside the establish range, it signals the value through a Queue.



```
void vTaskReadSonar(void const * argument)
{
    distance str_reading_sonar1;
    distance str_reading_sonar2;
  /* Infinite loop */
  for(;;)
  {
    acquired_data.mode = operation_mode;
    if(!(acquired_data.mode == MANUAL)){                //if mode is not manual
        str_reading_sonar1 =  read_distance1();     //read distance of sonar1
        if((str_reading_sonar1.sonar_dist < 400) && (str_reading_sonar1.sonar_dist > 1)){
            if(acquired_data.mode != MANUAL){
                //send Queue to analyse distances
                xQueueSend(Q_sonar1_readingHandle,&str_reading_sonar1,500);
            }
        }
    }

    if(acquired_data.mode == ADAPT_CONTROL){   //it´s just necessary in adaptative mode
        //read distance 2
        str_reading_sonar2 = read_distance2();

        //send Queue to analyse distances
        if((str_reading_sonar2.sonar_dist < 400) && (str_reading_sonar2.sonar_dist> 1)){
            xQueueSend(Q_sonar2_readingHandle,&str_reading_sonar2,500);
        }
    }
    osDelay(50);
  }
}
```

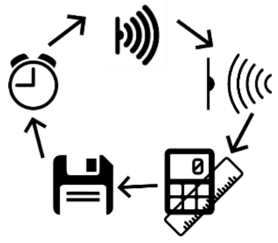
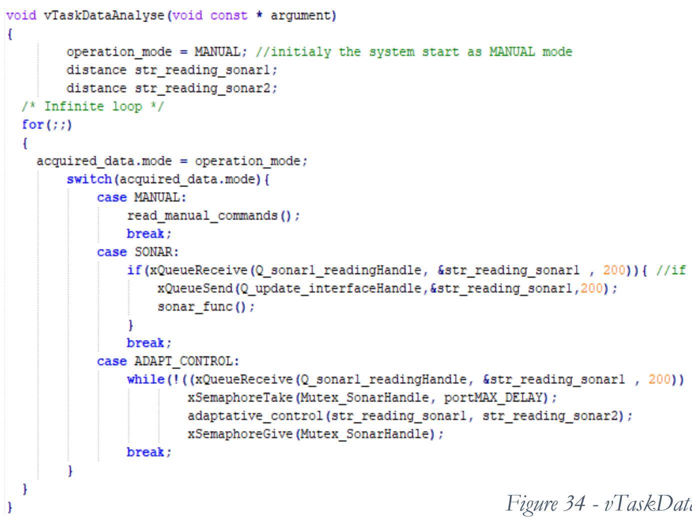


Figure 33 - vTaskReadSonar code and schematic

vTaskDataAnalyse: the task to analyses all of system data, is always running without any delay. It starts to verify which is the system operation mode, according to this will analyses the data and apply the respective functions:
-If "Manual", operates the *read_manual_commands(),*
-If "Sonar", waits for the Queue of the reading of sonar1, and then sends a Queue to update the interface with the value. After that operates the *sonar_func()*
-If "Adapt_Control", waits for the Queue of the reading of sonar1, and after that opens the mutex to execute the *adaptative_control()* with protection, without the risk of being interrupted.



```
void vTaskDataAnalyse(void const * argument)
{
        operation_mode = MANUAL; //initialy the system start as MANUAL mode
        distance str_reading_sonar1;
        distance str_reading_sonar2;
  /* Infinite loop */
  for(;;)
  {
    acquired_data.mode = operation_mode;
        switch(acquired_data.mode){
            case MANUAL:
                read_manual_commands();
                break;
            case SONAR:
                if(xQueueReceive(Q_sonar1_readingHandle, &str_reading_sonar1 , 200)){ //if
                    xQueueSend(Q_update_interfaceHandle,&str_reading_sonar1,200);
                    sonar_func();
                }
                break;
            case ADAPT_CONTROL:
                while(!((xQueueReceive(Q_sonar1_readingHandle, &str_reading_sonar1 , 200))
                        xSemaphoreTake(Mutex_SonarHandle, portMAX_DELAY);
                        adaptative_control(str_reading_sonar1, str_reading_sonar2);
                        xSemaphoreGive(Mutex_SonarHandle);
                break;
        }
  }
}
```

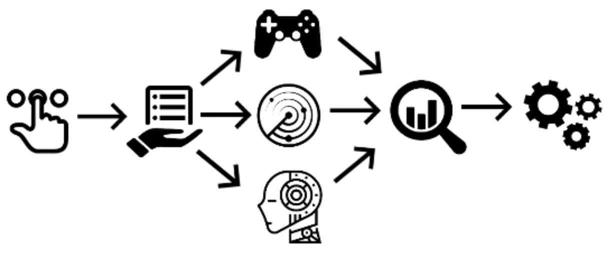


Figure 34 - vTaskDataAnalyse code and schematic

4.3 Adaptive Control

The implementation of the adaptative control was split in three parts. The **first part** consisting of detecting the object and knowing which side it is closest to. To implement that we use two ultrasonic sensors that are disposed in the structure like the image bellow shows it. With this setting method we can get two different distances in the same system and difference between them allow us to know if the structure shall move to the left or moving to right.

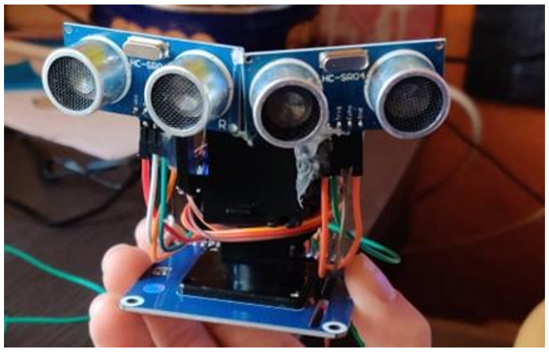


Figure 36 - Sonar Structure

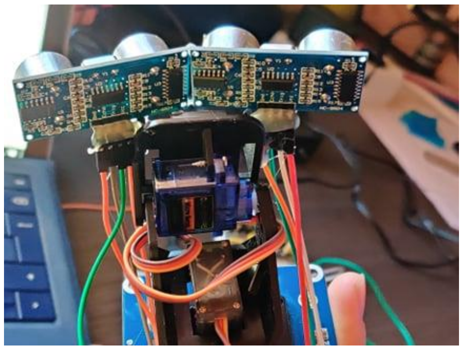
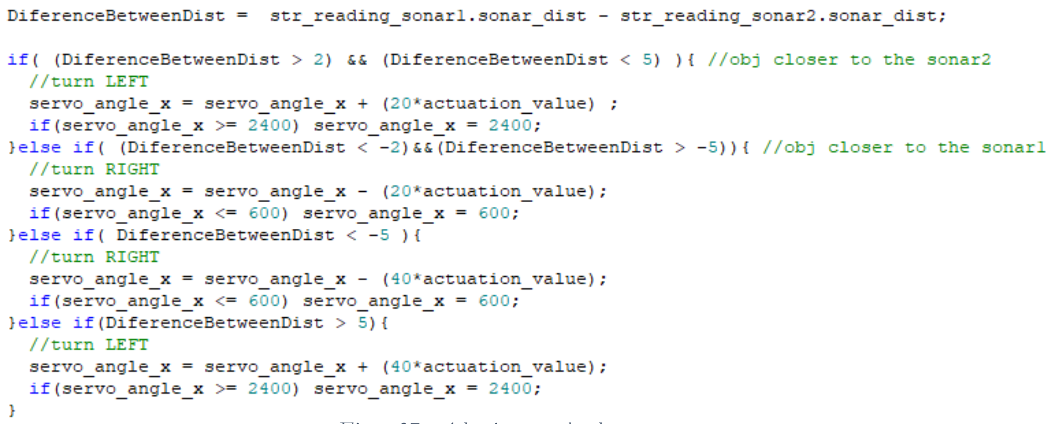


Figure 35 - Sonar Structure

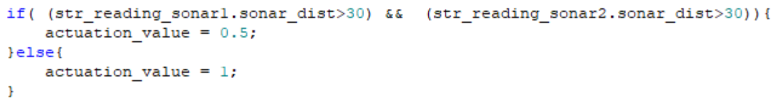
The **second part** of adaptative control consists on the control of the speed of movement. The system takes the difference between two readings and check if this difference is more than 5cm. Whether it is the speed of movement it´s faster.



```
DiferenceBetweenDist =  str_reading_sonar1.sonar_dist - str_reading_sonar2.sonar_dist;

if( (DiferenceBetweenDist > 2) && (DiferenceBetweenDist < 5) ){ //obj closer to the sonar2
  //turn LEFT
  servo_angle_x = servo_angle_x + (20*actuation_value) ;
  if(servo_angle_x >= 2400) servo_angle_x = 2400;
}else if( (DiferenceBetweenDist < -2)&&(DiferenceBetweenDist > -5)){ //obj closer to the sonar1
  //turn RIGHT
  servo_angle_x = servo_angle_x - (20*actuation_value);
  if(servo_angle_x <= 600) servo_angle_x = 600;
}else if( DiferenceBetweenDist < -5 ){
  //turn RIGHT
  servo_angle_x = servo_angle_x - (40*actuation_value);
  if(servo_angle_x <= 600) servo_angle_x = 600;
}else if(DiferenceBetweenDist > 5){
  //turn LEFT
  servo_angle_x = servo_angle_x + (40*actuation_value);
  if(servo_angle_x >= 2400) servo_angle_x = 2400;
}
```

Figure 37 - Adaptive control code

The **third part** of adaptative control consisting of giving the system the information how far is the object. The system must have the notion of this because if an object it´s closer, the movement must be faster and consequently, the angle of moving must be bigger. To implement that we verify the distances of the object and give to speed of movement an actuation value.



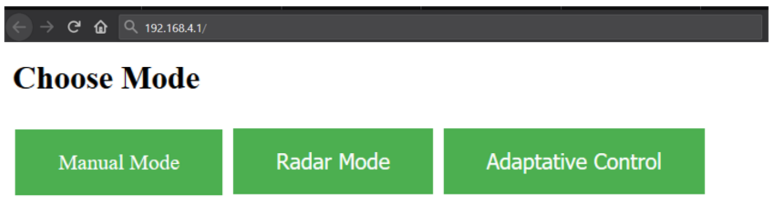
```
if( (str_reading_sonar1.sonar_dist>30) &&  (str_reading_sonar2.sonar_dist>30)){
    actuation_value = 0.5;
}else{
    actuation_value = 1;
}
```

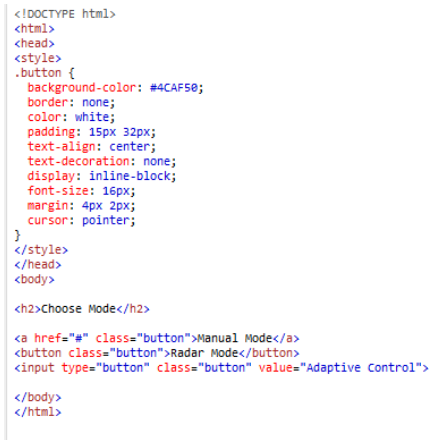
Figure 38 - Adaptive control code

4.4 Webpage

The webpage was developed through html text. After our station (in this case the pc) is connected via Wi-Fi to the Wi-Fi, by opening our browser and requesting the specific page IP (in this case 192.168.4.), the page is uploaded to the browser.

On the first connection, it is displayed the main page, to select the desired operation mode.

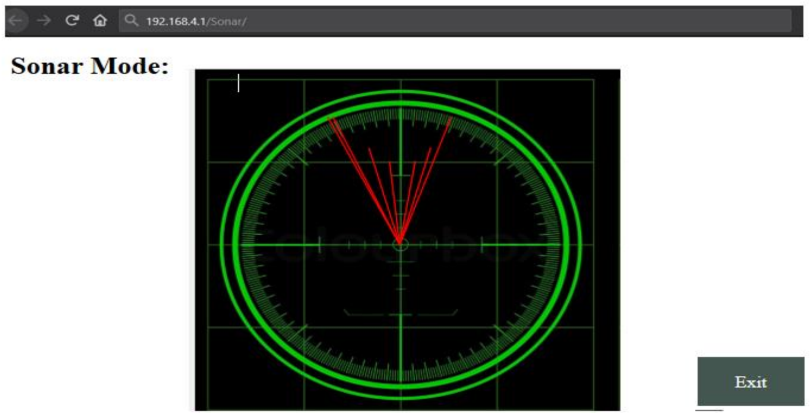


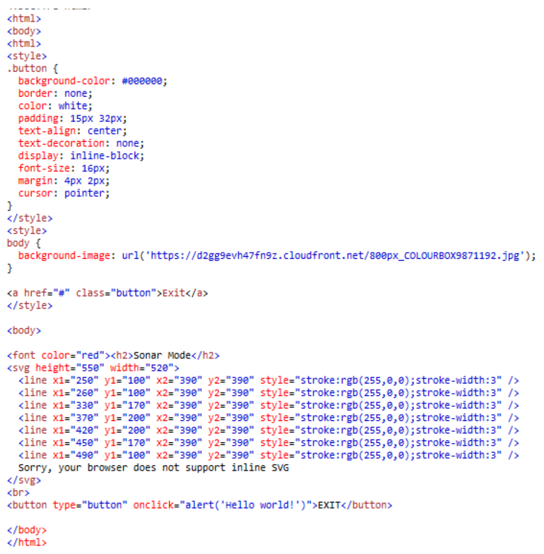


```
<!DOCTYPE html>
<html>
<head>
<style>
.button {
  background-color: #4CAF50;
  border: none;
  color: white;
  padding: 15px 32px;
  text-align: center;
  text-decoration: none;
  display: inline-block;
  font-size: 16px;
  margin: 4px 2px;
  cursor: pointer;
}
</style>
</head>
<body>

<h2>Choose Mode</h2>

<a href="#" class="button">Manual Mode</a>
<button class="button">Radar Mode</button>
<input type="button" class="button" value="Adaptive Control">

</body>
</html>
```

Figure 39 - Main page and code

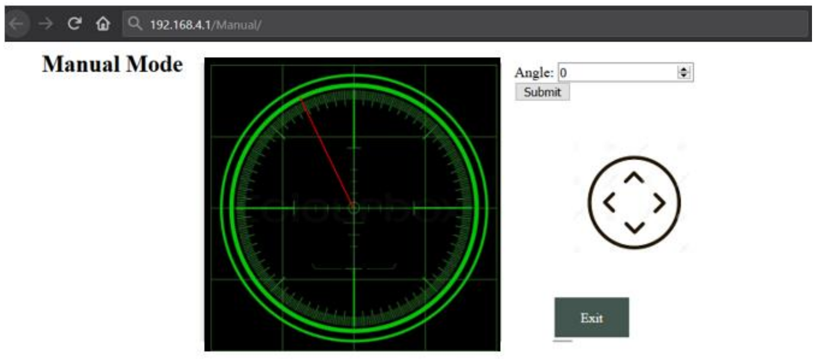
The User has the ability to choose between the three different modes. When pressed, the Browser makes a post, communicating with the Wi-Fi module about the mode selected. Based on the mode selected, it opens the respective page.

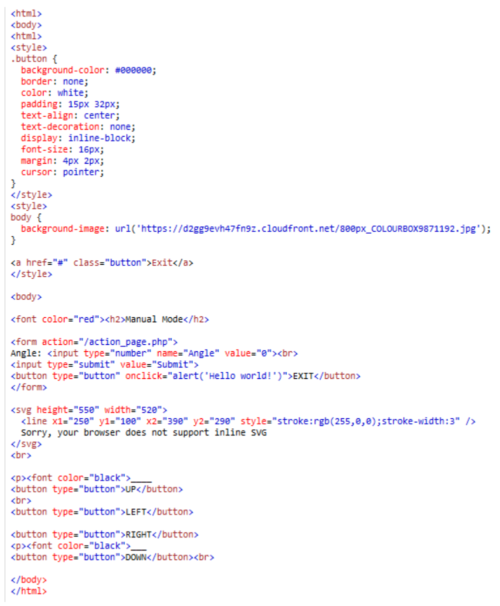




```
<html>
<body>
<html>
<style>
.button {
  background-color: #000000;
  border: none;
  color: white;
  padding: 15px 32px;
  text-align: center;
  text-decoration: none;
  display: inline-block;
  font-size: 16px;
  margin: 4px 2px;
  cursor: pointer;
}
</style>
<style>
body {
  background-image: url('https://d2gg9evh47fn9z.cloudfront.net/800px_COLOURBOX9871192.jpg');
}

<a href="#" class="button">Exit</a>
</style>

<body>

<font color="red"><h2>Sonar Mode</h2>
<svg height="550" width="520">
  <line x1="250" y1="100" x2="390" y2="390" style="stroke:rgb(255,0,0);stroke-width:3" />
  <line x1="260" y1="100" x2="390" y2="390" style="stroke:rgb(255,0,0);stroke-width:3" />
  <line x1="330" y1="170" x2="390" y2="390" style="stroke:rgb(255,0,0);stroke-width:3" />
  <line x1="370" y1="200" x2="390" y2="390" style="stroke:rgb(255,0,0);stroke-width:3" />
  <line x1="420" y1="200" x2="390" y2="390" style="stroke:rgb(255,0,0);stroke-width:3" />
  <line x1="450" y1="170" x2="390" y2="390" style="stroke:rgb(255,0,0);stroke-width:3" />
  <line x1="490" y1="100" x2="390" y2="390" style="stroke:rgb(255,0,0);stroke-width:3" />
  Sorry, your browser does not support inline SVG
</svg>
<br>
<button type="button" onclick="alert('Hello world!')">EXIT</button>

</body>
</html>
```

Figure 40 - Sonar mode page and code





```
<html>
<body>
<html>
<style>
.button {
  background-color: #000000;
  border: none;
  color: white;
  padding: 15px 32px;
  text-align: center;
  text-decoration: none;
  display: inline-block;
  font-size: 16px;
  margin: 4px 2px;
  cursor: pointer;
}
</style>
<style>
body {
  background-image: url('https://d2gg9evh47fn9z.cloudfront.net/800px_COLOURBOX9871192.jpg');
}

<a href="#" class="button">Exit</a>
</style>

<body>

<font color="red"><h2>Manual Mode</h2>

<form action="/action_page.php">
Angle: <input type="number" name="Angle" value="0"><br>
<input type="submit" value="Submit">
<button type="button" onclick="alert('Hello world!')">EXIT</button>
</form>

<svg height="550" width="520">
  <line x1="250" y1="100" x2="390" y2="290" style="stroke:rgb(255,0,0);stroke-width:3" />
  Sorry, your browser does not support inline SVG
</svg>
<br>

<p><font color="black">____
<button type="button">UP</button>
<br>
<button type="button">LEFT</button>

<button type="button">RIGHT</button>
<p><font color="black">___
<button type="button">DOWN</button><br>

</body>
</html>
```

Figure 41- Manual Mode Page and code

5. TESTING

Hardware Test Cases	Expected Output	Real Output
Initialization PWM for the servos	Servo moves	Servo moves successfully
Drive servos to several directions	Servo moves with our inputs	Servo moves with our inputs successfully
Test Ldr Operation	Gathering values through ADC	Gets the value from ADC
Measure distances with ultrasonic sensor	Retrieve the exact value of the object distance	Receives distance but sometimes there is noise
Detect Object with ultrasonic sensor	Detect an interference in the signal	Object detected
Test the connection with esp8266	Stablish a connection between two devices.	PC and ESP8266 connecred via Wi-Fi
Test data transmission with esp8266	Send AT commands	Configuration with succes of ESP8266 through AT commands
Test data transmission with esp8266	Send a byte.	Information sent, ready for full html text
Test the structure (the tilt and pan)	input PWM in tilt and pan servo-motor´s	Pan & Tilt moves in both axis about 180º
Test the structure (the mechanism)	Visualize the movements	Setting is robust, and moves without any problem

Table 1 - Hardware Test Cases

Software Test Cases	Expected Output	Real Output
Test the distance measurement (sonar)	See the values through the terminal	Measure the distance precisely
Test different values of PWM (servo)	Vizualize different values of PWM	PWM has the pretended frequency and duty-cycle
Teste send different AT commands (esp8266)	Vizualizes the respective responses	Response ok from ESP, however some commands don't work at the first tim
Test the post through the webpage (esp8266)	Vizualize the post, communication from the browser to the esp	--------------Not tested------------
Test the buttons on page (esp8266)	Vizualize the buttons	Buttons created
Test queues (freeRTOS)	Vizualize by the sending values	Queues successfully sent
Test semaphores (freeRTOS)	Vizualize by debug the binary semaphore operation	Semaphore successfully operated
Test mutexes (freeRTOS)	Vizualize through the UART good synchronization	Protection applied successfully
Test task scheduling	Visualize the scheduling	Scheduling successfully, however it sometimes has delay or crashes
Test the operation modes of the system	Good results of Adaptative control mode/Manual mode/Sonar mode	All modes work correctly

Table 2 - Software Test Cases

6. Estimated Budget

On the table below we present the estimated budget of our project. The components used present a certain degree of quality and at low-cost. With this table we can estimate the cost of producing one product.

Item	Price
Structure Tilt and Pan	26,58€
HC-SR04 x2	2 x 3,60€
LDR	1,40€
Stm32	29,90€
Power Source	4,67€
Esp8266	4,31€
Total:	74,06€

Table 3 - Estimated Budget

7. Gantt Diagram

Based on the Waterfall method we can schedule the project in five major entities: Documentation, Analysis, Design, Implementation and Testing. Estimating each task, test, and proceeding, it concludes on a 4 months long project.

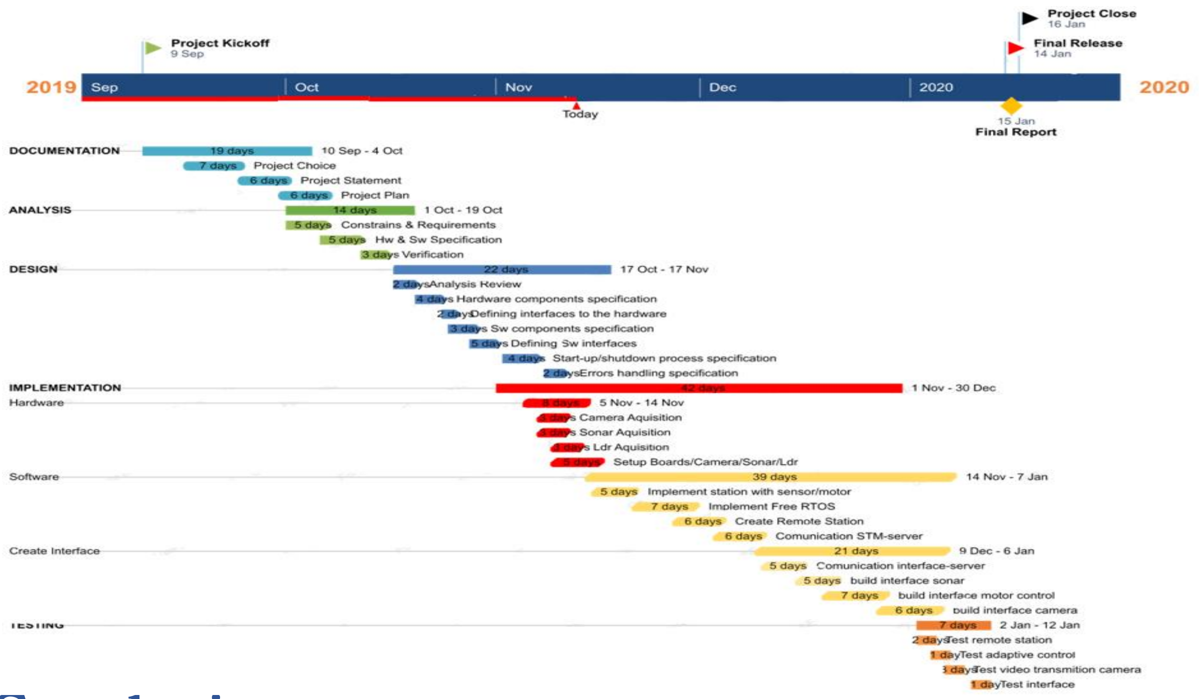


Figure 42 - Gantt Diagram

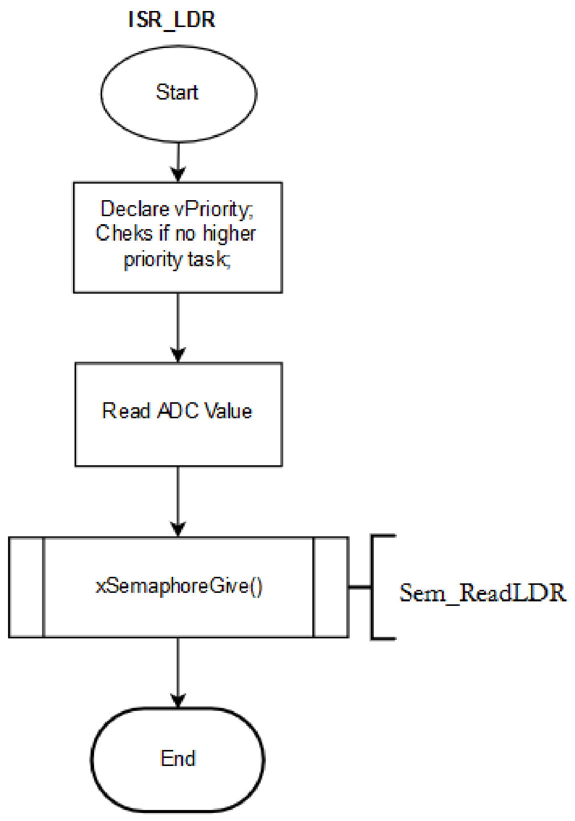
8. Conclusion

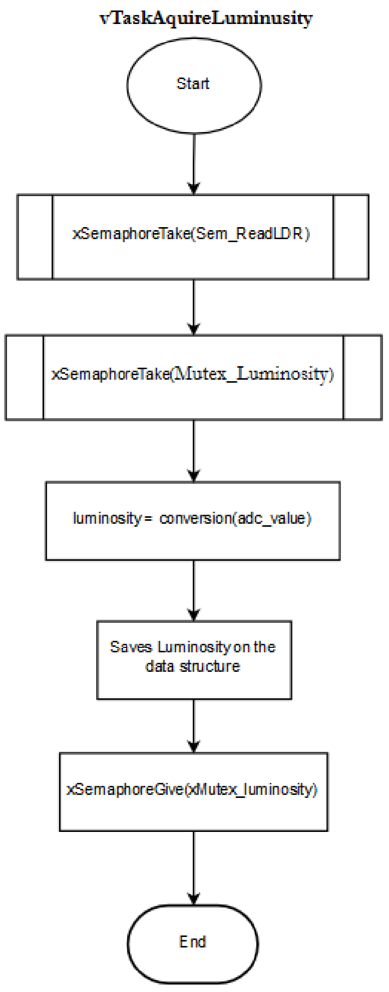
With this project we develop our understanding on the IoT applications, were we accomplish to create a website an interact it with the microcontroller. Also, we had the opportunity to put in practice the concepts learned on embebbed systems about FreeRTOS and concurrent programming.

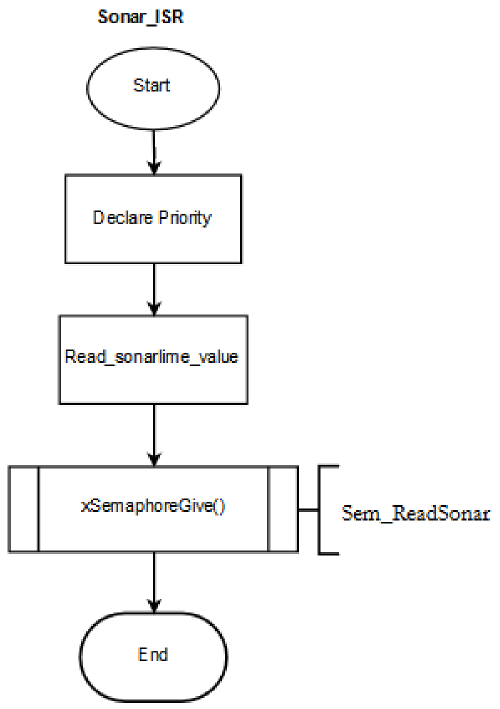
With that being said, our project as some unfinish work, giving it lot of space for improvements to fix some bugs. For future work we can pin point some features to improve:

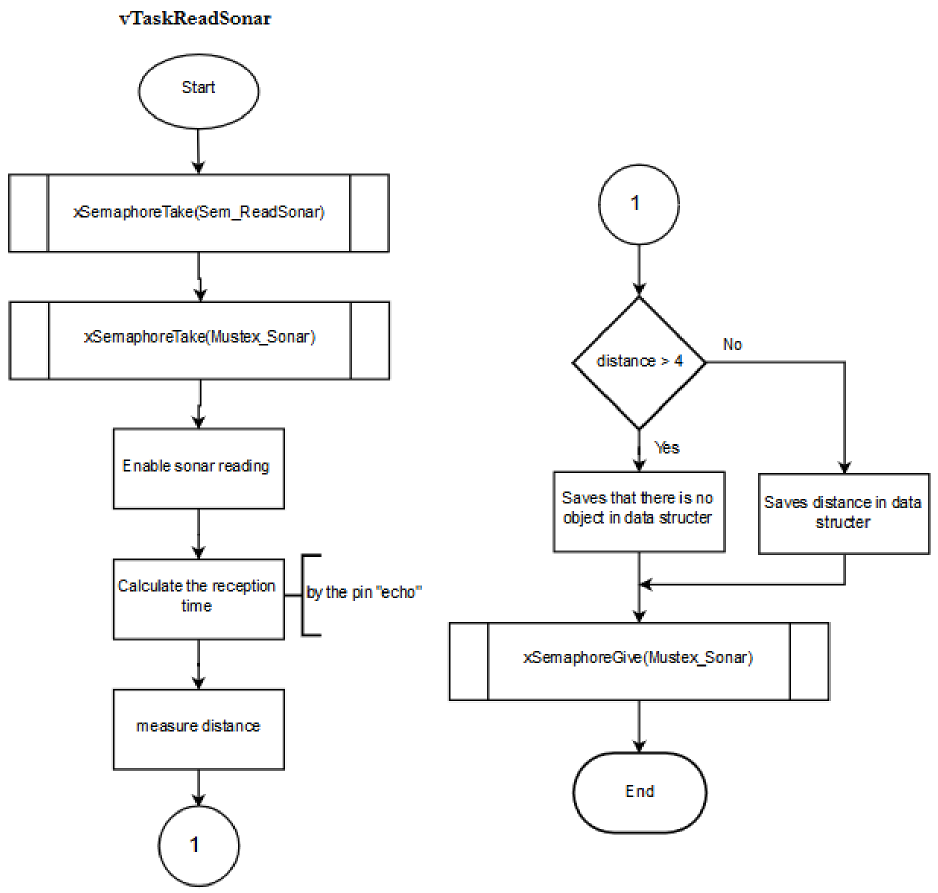
- Change the network typology, were the ESP8266 works as Station instead of Access Point
- Fix some of the project bugs in order to improve performance and avoid more failures
- Expand the application, by creating a mobile app to operate the device;

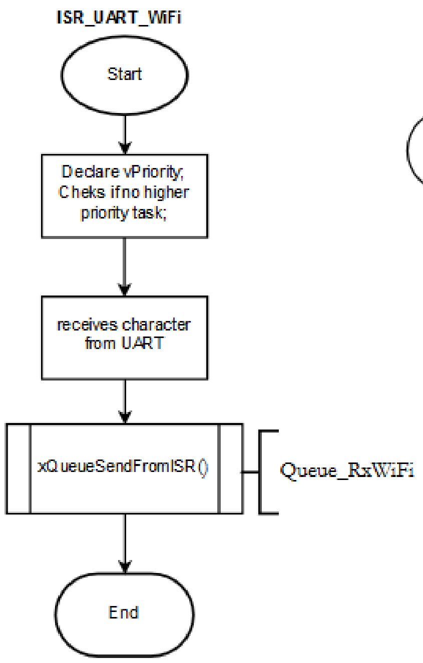
9. APPENDIX

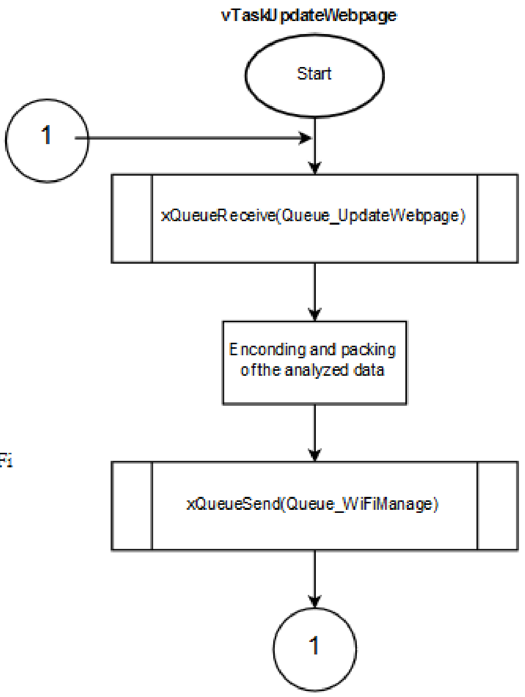

ISR_LDR
Start
Declare vPriority; Cheks if no higher priority task;
Read ADC Value
xSemaphoreGive()
Sem_ReadLDR
End

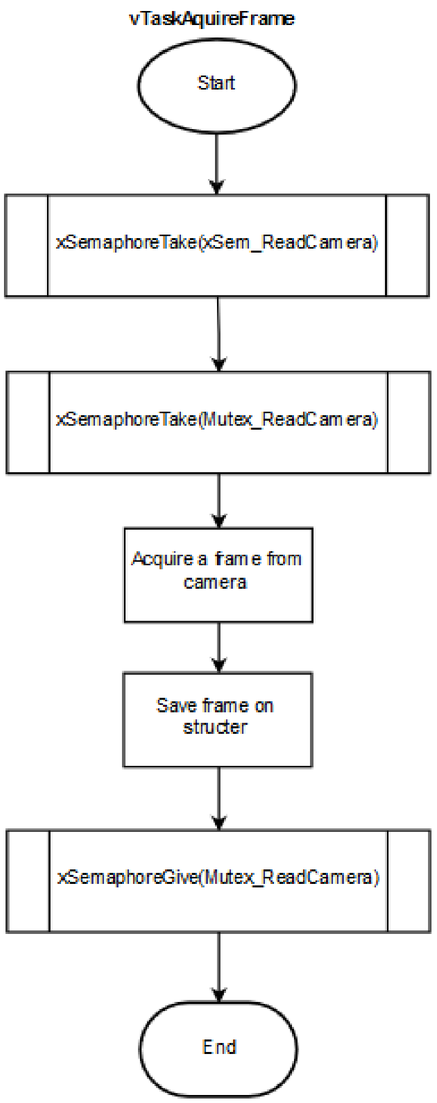

vTaskAquireLuminusity
Start
xSemaphoreTake(Sem_ReadLDR)
xSemaphoreTake(Mutex_Luminosity)
luminosity = conversion(adc_value)
Saves Luminosity on the data structure
xSemaphoreGive(xMutex_luminosity)
End

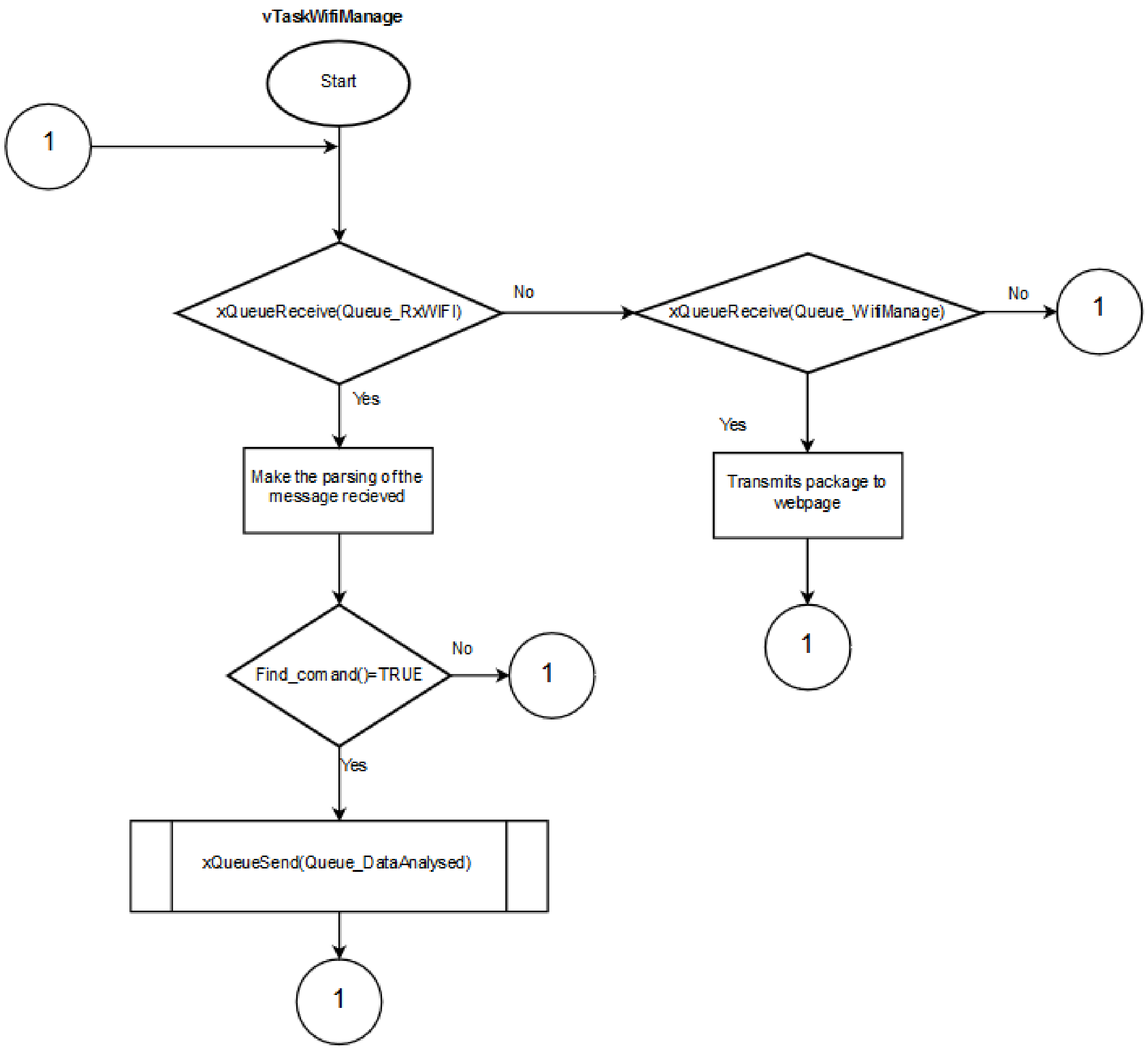

Sonar_ISR
Start
Declare Priority
Read_sonarlime_value
xSemaphoreGive()
Sem_ReadSonar
End


vTaskReadSonar
Start
xSemaphoreTake(Sem_ReadSonar)
xSemaphoreTake(Mustex_Sonar)
Enable sonar reading
Calculate the reception time
by the pin "echo"
measure distance
1
1
distance > 4
No
Yes
Saves that there is no object in data structer
Saves distance in data structer
xSemaphoreGive(Mustex_Sonar)
End


ISR_UART_WiFi
Start
Declare vPriority; Cheks if no higher priority task;
receives character from UART
xQueueSendFromISR()
Queue_RxWiFi
End


vTaskUpdateWebpage
Start
1
xQueueReceive(Queue_UpdateWebpage)
Enconding and packing of the analyzed data
xQueueSend(Queue_WiFiManage)
1


vTaskAquireFrame
Start
xSemaphoreTake(xSem_ReadCamera)
xSemaphoreTake(Mutex_ReadCamera)
Acquire a frame from camera
Save frame on structer
xSemaphoreGive(Mutex_ReadCamera)
End


vTaskWifiManage
Start
1
xQueueReceive(Queue_RxWIFI)
No
xQueueReceive(Queue_WifiManage)
No
1
Yes
Make the parsing of the message recieved
Yes
Transmits package to webpage
1
Find_comand()=TRUE
No
1
Yes
xQueueSend(Queue_DataAnalysed)
1

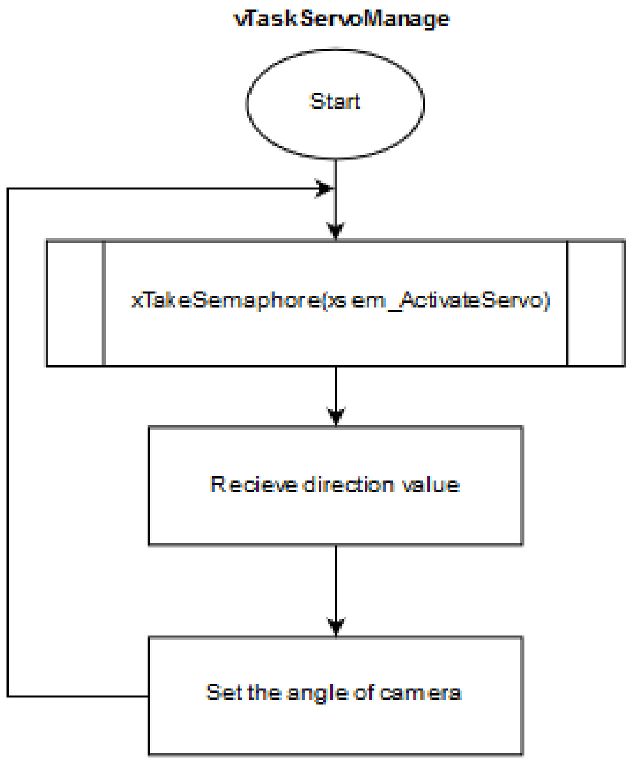
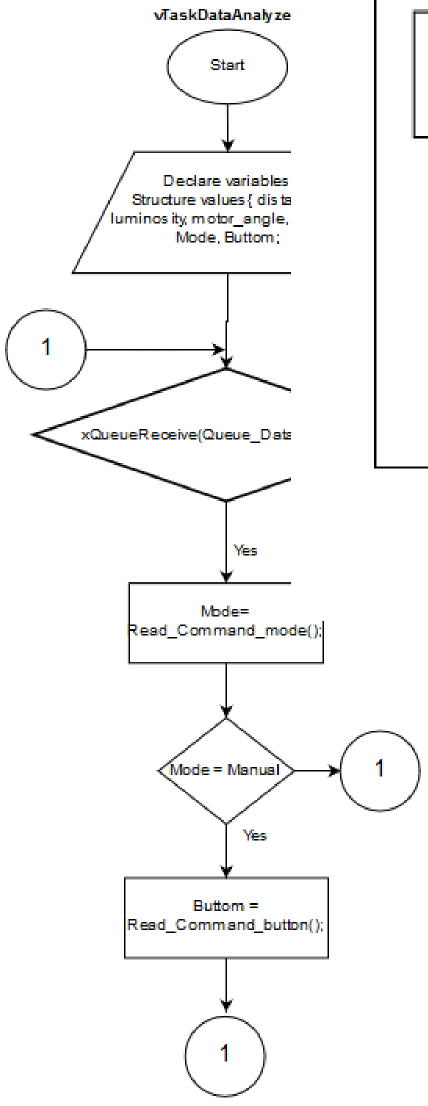
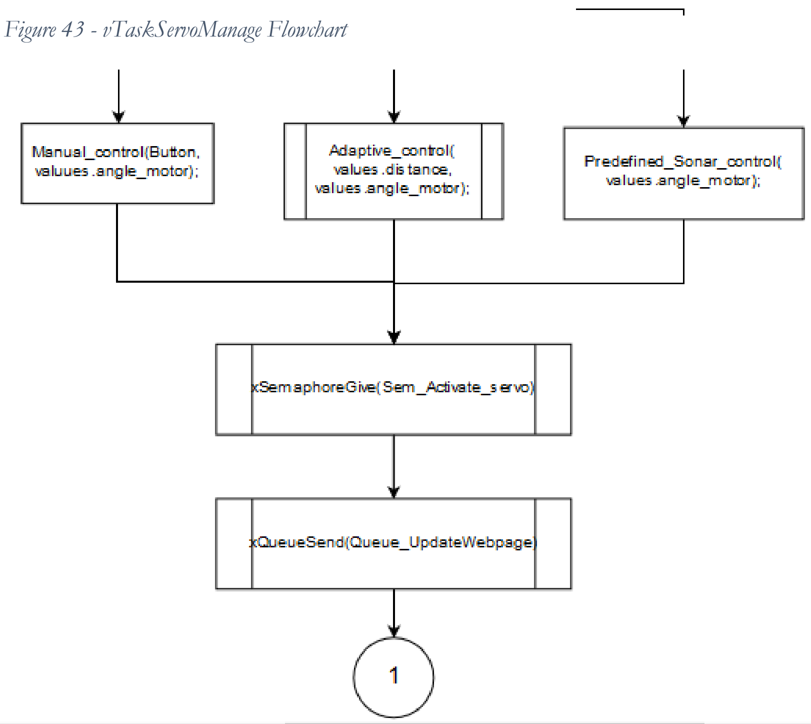
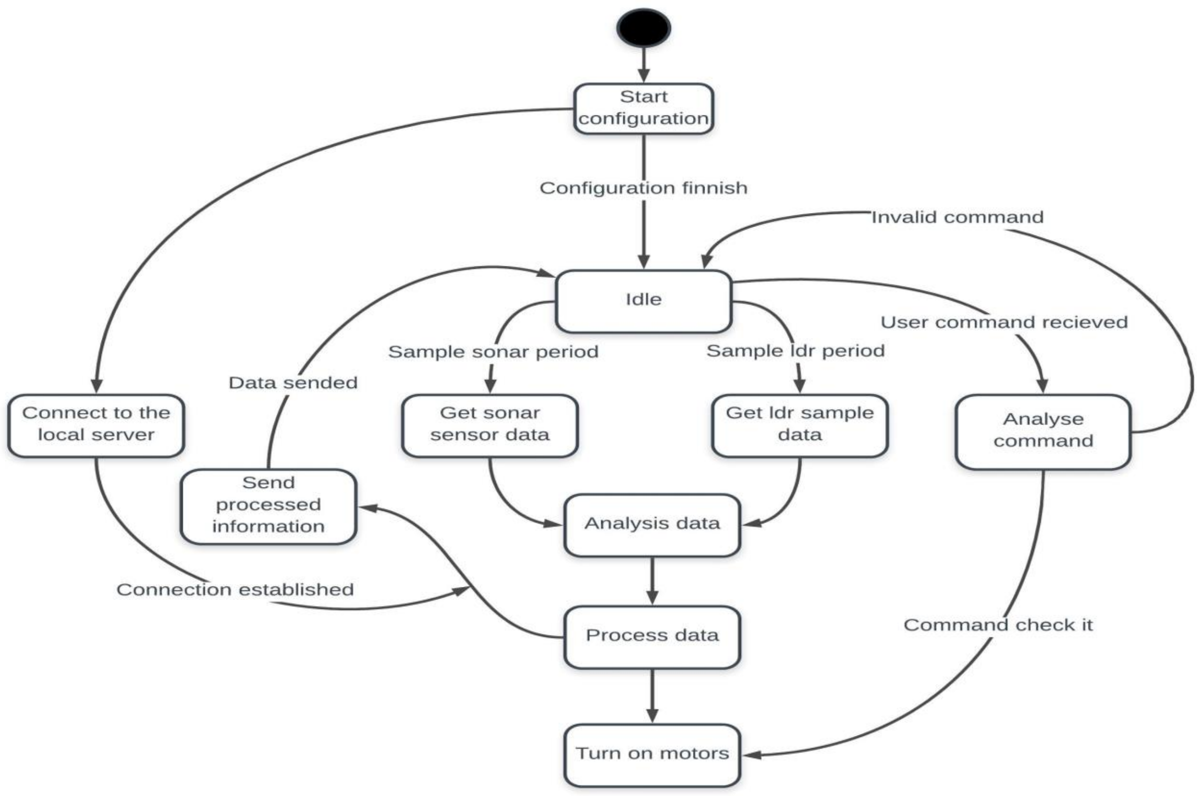


Figure 43 - vTaskServoManage Flowchart







Event Name	**System Response**	**Source**	**Type**
Sensor Data Gathering	Collects data from sensors.	Time	Synchronous
Access Request	Establish the connection between remote system and local system	Local System	Synchronous
Data Transfer	Establish communication between host-client. Transfer data between webpage and local system.	Time	Synchronous
Data Analysis	Process the data collected and make important decisions	Local System	Synchronous
Push button	Sends command to local System	User	Asynchronous
Choose Operation Mode	Defines if Automatic or Manual mode	User	Asynchronous
Streaming Camera	Put the video of the camera on interface	Camera	Synchronous