Self Driving Car

Batuhan Celik
Ilgaz Er

CMPE 443 Term Project

February 5, 2023

Table of Contents

List of Tables

List of Figures

Note that our sequence diagrams show different components in different colors.

1. Orange: Software Modules
2. Green: STM32 NUCLEO-l55 Peripherals
3. Blue: External Hardware Components

1 Introduction

Aim of this project is designing an embedded system for a car and providing electronic-hardware an software implementation of the design. The car shall be run in 2 different modes: manual and auto which are interchangeable using a button on the car.

In the manual mode, the user should control the direction and breaks of the car with the help of a joystick. Even though project requirements implied only 4 directions for the controls, forward, backward, clockwise rotation, counterclockwise rotation, our system supports mixture of these directions and tries to mimic car controls from computer games that use joystick controllers as much as possible.

During the auto mode, the car is expected to follow a path indicated by led strides surrounding the path as fast as possible. To do so, the car is equipped with light dependent resistors on its left and right side, using these sensors the autonomous driver decides when and how much to turn. The user shall start the execution of the autonomous scenario using the joystick by increasing the throttle.

There is also shared behaviour between these 2 modes. Firstly, the car is equipped with header LEDs which indicate the direction of the car and indicator LEDs which provide information about the state of the car, like the currently active mode. These LEDs are active regardless of the mode. In addition, auto break system which engages breaks automatically when an obstacle is detected in front of the car is always active, however, in manual mode the user can release the breaks using the joystick push button while the autonomous driver cannot release the breaks. Finally, motor controller and motors are active in all phases of the execution.

2 System Components

2.1 Hardware Components

2.1.1 STM32 NUCLEO-l552ze-q

System uses STM32 NUCLEO-l552ze-q as the MCU. This board controls the drive logic and runs the motor driver. In addition, it provides us the ADC and timers required.

This unit will be mentioned as MCU in the rest of the report.

2.1.2 L298N Dual H-Bridge Motor Driver

The motor driver connects to 4 motors, one for each tire, and controls these motors according to PWM and motor enable signals it receives from the MCU.

Also, this unit connects to the batteries and provides regulated 5 Volts for the MCU and other components

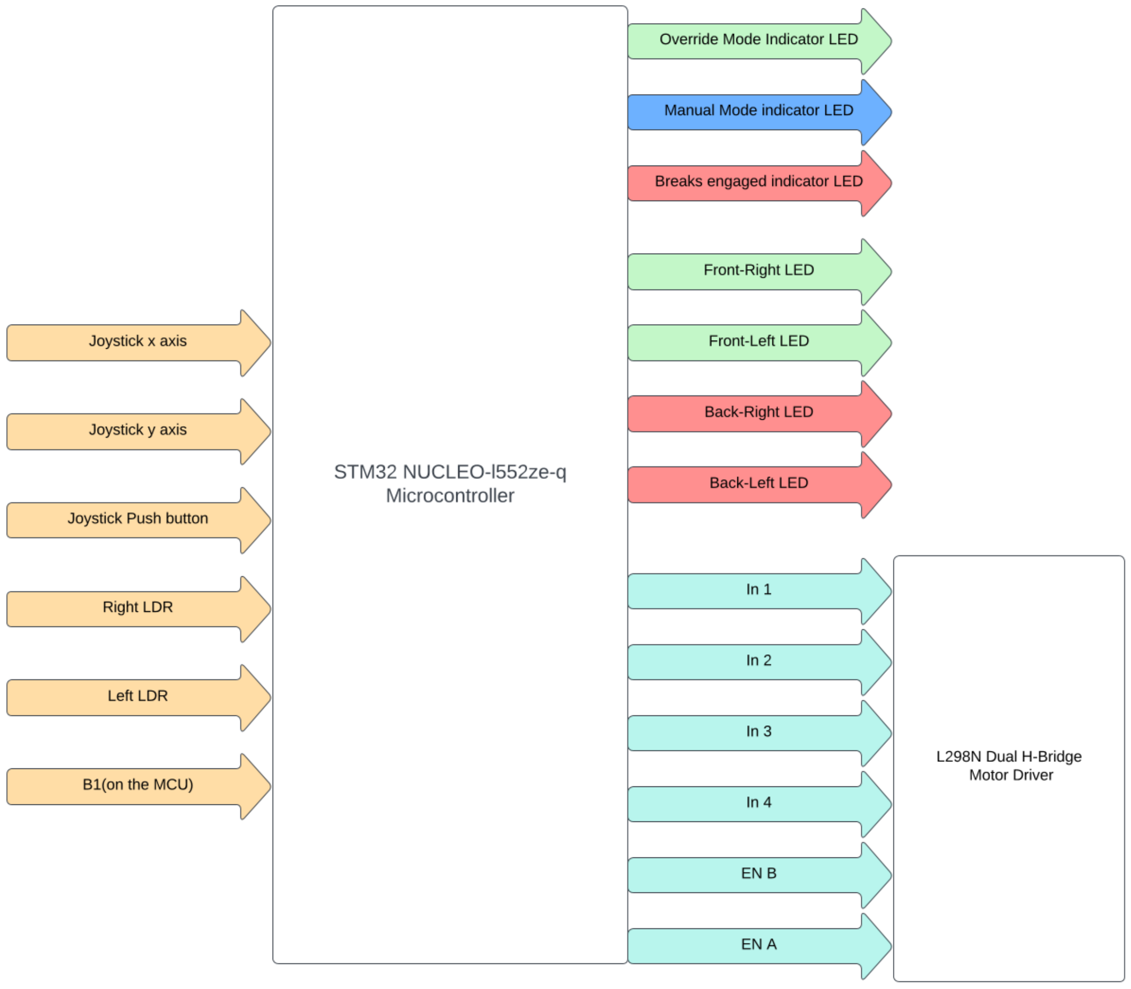

Figure 1. Block Diagram
You can view it more detailed by clicking here

2.1.3 Joystick

The car is equipped with a 2 dimensional joystick controller for the direction control. Joystick is connected to the MCU with 3 pin connections: 2 analog connections that convert the horizontal and vertical tilt of the controller to voltage values and the last one is for the push button that shorts to ground when pressed which activates breaks when pressed. In the manual mode, this button can be used for releasing breaks too.

2.1.4 Light Dependant Resistors

There are 2 LDRs on the front-left and the front-right side of the car. By measuring resistance values of these LDRs, the MCU understands which side of the car is brighter and tries to move towards darker side in the autonomous mode.

2.1.5 Headlight LEDs

The car is equipped with 4 headlight LEDs places on the each corner of the car, namely front-right, front-left, back-left, back-right. These LEDs are responsible for indicating the direction of the car and they light in 4 different modes:

- Forward: Front LEDs are lit and back LEDs are turned off.
- Left: Left LEDS are blinking with a frequency of 2Hz.
- Back: Back LEDs are lit and front LEDS are turned off.
- Right: Right LEDS are blinking with a frequency of 2Hz.

2.1.6 Indicator LEDs

Indicator LEDs are the LD1, LD2, LD3 leds on the STM32 NUCLEO-l552ze-q microcontroller, each indicator LED indicates gives information about the current driving state.

- LD1(Green) : This LED is lit when the car is in manual override mode, meaning the automatic breaking is disabled.
- LD2(Blue) : This LED being lit indicates the car is in manual mode and the driver can use the joystick to control the direction.
- LD3(Red) : This LED turns on when the car detects an obstacle and activated breaking autonomously.

2.1.7 HC-SR04 Ultrasonic Distance Sensor

The ultrasonic distance sensor measures distances between 2 and 400 cm by measuring the time between the departure of an ultrasonic wave and its return. The sensor is quite reliable with detecting large smooth surfaces lying perpendicular to the car. However, it is prone to random spikes in the measured distance and can also miss objects lying at an angle to the car.

Using the ultrasonic sensor at the front of the car, the MCU checks if there are any obstacles closer than 10 cm’s, and activates the brakes in that case.

2.1.8 Blue Button

This push button is the B1 button on the STM32 NUCLEO-l552ze-q. This button is used for switching between manual and autonomous modes.

2.1.9 Motor Enable Switch

This switch connects motor controller, which provides power to the whole system, to the batteries. The switch powers the whole system on when closed.

2.2 Software Components

2.2.1 Module Oriented Structure

Our software consists of packages called Modules that encapsulates a piece of hardware and its drive logic and provides an abstraction layer for other modules. For example, Motors module is responsible for generating PWM signals and setting motor enable pins for the motor controller. It utilizes TIM3 of the MCU and contains all the logic required to achieve these tasks and other modules, like the Drive module, can use geter and setter functions to interact with the motors.

Each module consists of following components:

- Hardware
  All modules except Drive and Main are associated with a piece of hardware.
- Logic
  The logic required to convert interface calls to the actions.
- Interface
  Modules provide an interface to communicate with other modules.
- Initializer
  Module initialzers set the initial state of the module and MCU peripherals used by the module.

Our implementation consists of 6 modules:

- Main module is contains the main function and is responsible for initializing other modules, putting the MCU in sleep mode(wfi) when required, and starting ADC conversions.
- Drive module implements the controller logic and upon sensor readings from Analog Sensors module and Module, it utilizes Motors module, Indicators module and LED module in response.

- Motors module inputs speed and direction parameters from the drive module and runs the motor driver accordingly.
- LED module runs a basic timer(TIM6) to drive headlight LEDs.
- Indicators module converts the Drive module state to led indications using GPIO output pins.
- Analog Sensors module drives the MCU ADC and requests for driver updates upon end of conversion.
- Ultrasonic module runs TIM3 to drive the ultrasonic distance sensor.
- Buttons module listens for external interrupts from joystick button and B1 button and requests state changes from Drive module accordingly.

There are 2 Modules which are not associated with a piece of hardware: Main module and Drive module. These are the modules associated with high level control logic of the car.

Contents of each module is discussed in detail in the next chapter.

2.3 Control Logic

Control logic both in manual mode and autonomous mode is implemented as a finite state machine where both mode have their respective machines. Drive module is responsible for implementing these state machines. At each state change Motors, LED and Indicators modules are updated.

2.3.1 Manual Controls

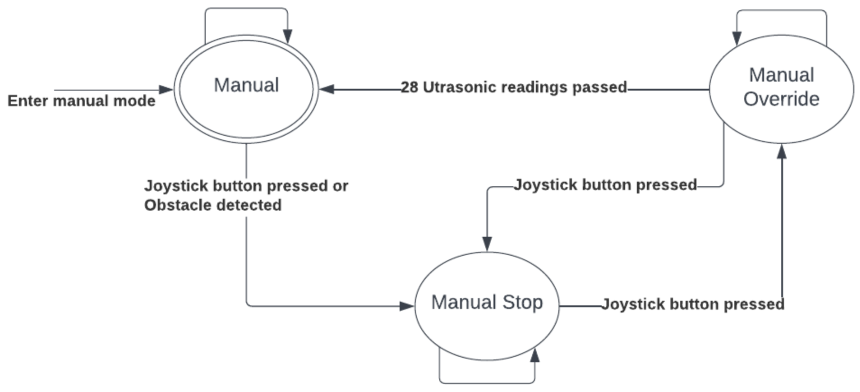

Figure 2. State diagram for Manual Mode

In manual mode, the Drive module follows the state machine above. A state change occurs periodically and each state change indicates updates to the other modules. States point to themselves if none of the state change conditions are met, providing updates happen at each iteration of the state machine.

State change to the Manual and Manual Override states map the joystick values retrieved from Analog Sensors Module to direction and speed values for the Motors module whereas state change to the Manual Stop state enables breaks using the Motors Module. In addition, all state changes asks for updates to LED and Indicator modules.

2.3.2 Autonomous Controls

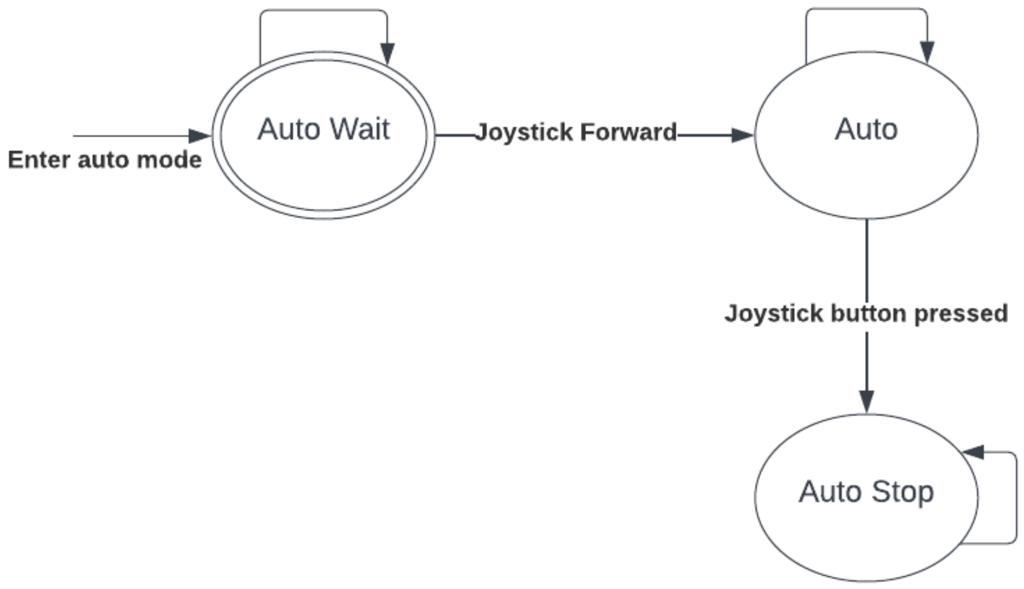

Figure 3. State diagram for Autonomous Mode

In the autonomous mode, the logic is a simpler version of the manual mode where it is not possible to escape from Auto Stop state. State changes to Auto Wait does not change the motor state whereas state changes to Auto state maps the difference between LDR resistance readings to the speed and direction. Finally state changes to the Auto Stop state engages breaks.

2.3.3 Update Loop

In normal operation, The MCU updates the system every 20 miliseconds and goes back to power saving mode when no interrupts are being handled. Each update is triggered by a TIM7 interrupt. Alternatively, there is also a debug mode where in each iteration, the Main module spin waits for 20 ms before directly starting the update. This was implemented as the sleep modes interfere with Monitoring, which will be discussed in the Main section of the modules chapter.

Each update starts with a reading of the four sensors connected to the ADC. Upon end of the conversion, the `drive()` function of the Drive module is called by the ADC interrupt handler. This function represents a state change in the finite state machines from the Control Logic section. To calculate the next state of the machine, the function uses the Ultrasonic module to get the ultrasonic distance and the Analog Sensor module to get either the joystick reading or the difference between LDR values, depending on the current state of the car. Then, these values are clipped, scaled and combined with the current car state to produce the speed and direction values for the Motors module and state values for the LED and indicator modules.

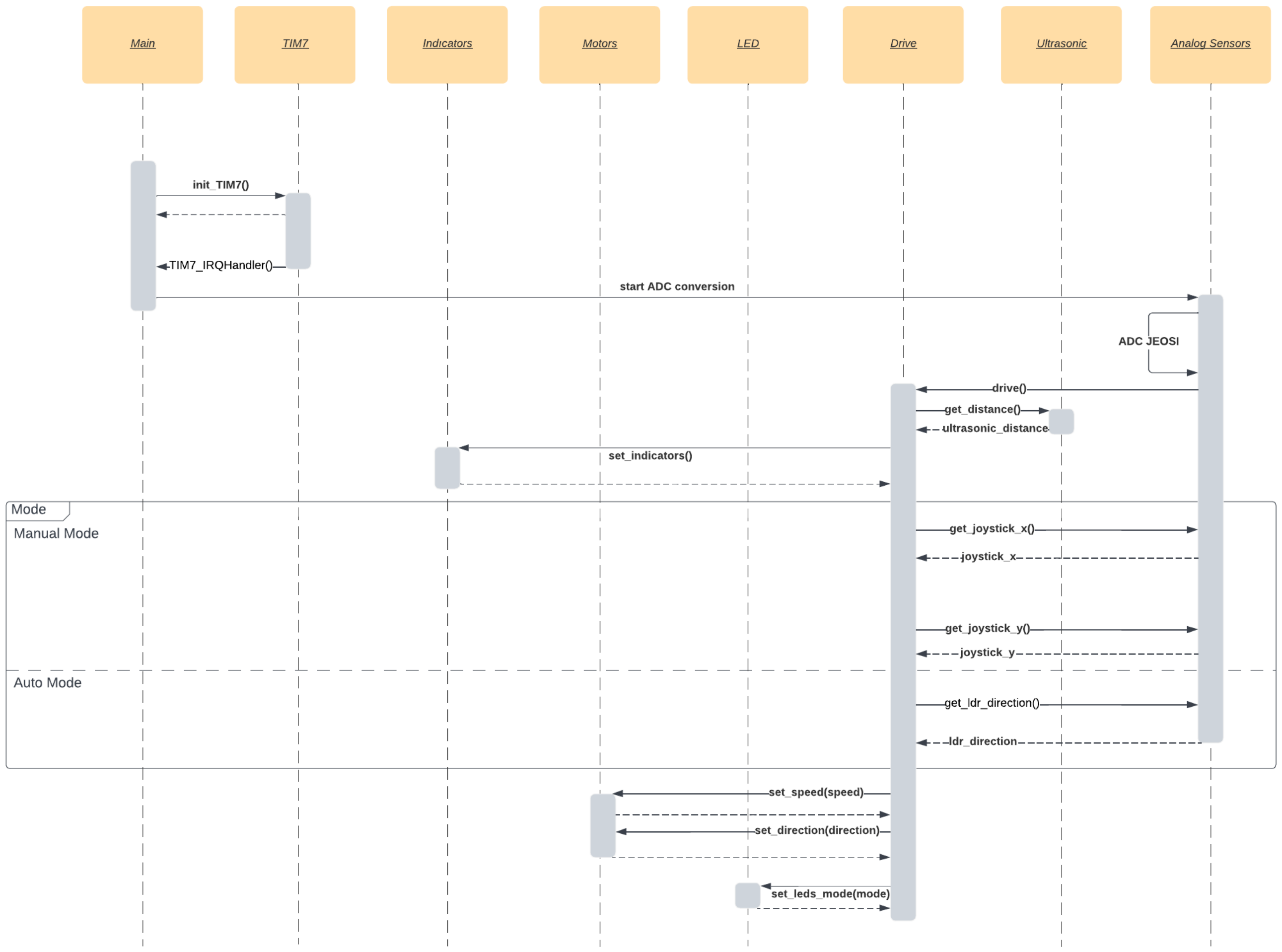

Figure 4. Sequence Diagram for the Update Loop
You can view it more detailed by clicking here

3 Modules

3.1 Main

Main is the first module to initialize and only one that runs on out of instruction handlers. Its responsibilities cover initializing other modules during startup and triggering update loops periodically with 50 Hz by starting ADC injected conversions. This module has also 2 states defined by macros: power consumption state and monitoring state.

In the power consumption state, 50Hz is generated by the TIM7 timer interrupts and the module puts the MCU to sleep between updates. On the other hand, monitoring state produces 50Hz update frequency using pooling.

These 2 states are implemented because we used STM32 Cube Monitor for monitoring sensor values and fine tuning the drive logic. However while the MCU is in the sleep mode, STM32 Cube Monitor cannot access the microcontroller, rendering monitoring not possible. Thus, we are setting the state of the Main module during compilation time as it satisfies our needs.

3.2 Driver

Driver module implements the controlling logic used during updates and is the only high level module of our implementation. States from the state diagrams are stored as enums defined by macros in the global static variable named `mode` and sensor readings combined with this variable runs the output modules(Motors, Leds, Indicators)

3.2.1 Interface

Drive()

`drive()` function implements the update loop of the car. Firstly it reads the ultrasonic distance and checks if it is smaller than 10cm. If so it engages breaks and sets the `mode` to `MANUAL_STOP` if the current state is from Manual state diagram or sets `mode` to `AUTO_STOP` otherwise.

If this condition is not met, it means there wont be a state change in this update, thus it calls `set_indicators(mode)` function from indicators module to set the indicator leds. Then, the current state is matched to a function and that function is called. State function calls are shown in the Table 1 and discussed in more details in the next section.

set_state(uint8_t mode) and init_drive(uint8_t mode)

`set\_state(uint8\_t mode)` changes the `mode` variable to the given mode and calls the drive function to force an update. It is accompanied by `init\_drive(uint8\_t mode)`

State	Implementing Function
MANUAL	drive_manual()
MANUAL_OVERRIDE	drive_manual()
MANUAL_STOP	drive_stop()
AUTO	drive_auto()
AUTO_WAIT	auto_wait()
AUTO_STOP	auto_stop()

Table 1. Matching of state variables to state handlers

to change the next state without forcing an update. They are used for driver initialization.

joystick_button_handler() and blue_button_handler()

These functions are called by external interrupt handlers and they are wired to the same logic. They engage breaks upon call and set the state to `MANUAL_STOP` or `AUTO_STOP` according to the current state.

get_mode()

Returns the current mode. Used by indicators to update the indicator leds.

3.2.2 State functions

drive_manual()

This functions gets the joystick x and joystick y values from the Analog Sensors module and clips them to 0 if they are between -200 and 200 to eliminate the noise. Firstly it scales joystick x and joystick y values up to be used as direction and speed parameters by the Motors module. Then it checks the joystick y to see if the car is rotating, if so sets the headlight LEDs using the `set_led_direction()` from the LED module. If the joystick y value is not sufficient for a turn, it checks if the joystick x value about the movement direction and updates leds likewise.

Notice both MANUAL and MANUAL_OVERRIDE states are wired to the this function. Only difference between these states is that, distance check at the beginning of drive function is discarded when the state is MANUAL_OVERRIDE.

drive_stop() and auto_stop()

Sets the header leds to stop state and engages breaks using Motors module.

auto_wait()

Recalibrates LDR values using `refresh_ldr_calib()` from the Analog sensors module.

Detailed sequence diagrams of interaction between Drive module functions and other Modules software and hardware are given in the sections regarding hardware driving modules.

3.3 Ultrasonic

The ultrasonic module is in charge of periodically initiating reads from the ultrasonic sensor, as well as processing the result to eliminate spurious data and noise. TIM3 is used to send the 10 microsecond pulse to the sensor's TRIG pin and to record the timestamps of ECHO rising and falling. As per the datasheet, the sensor is triggered every 60 milliseconds.

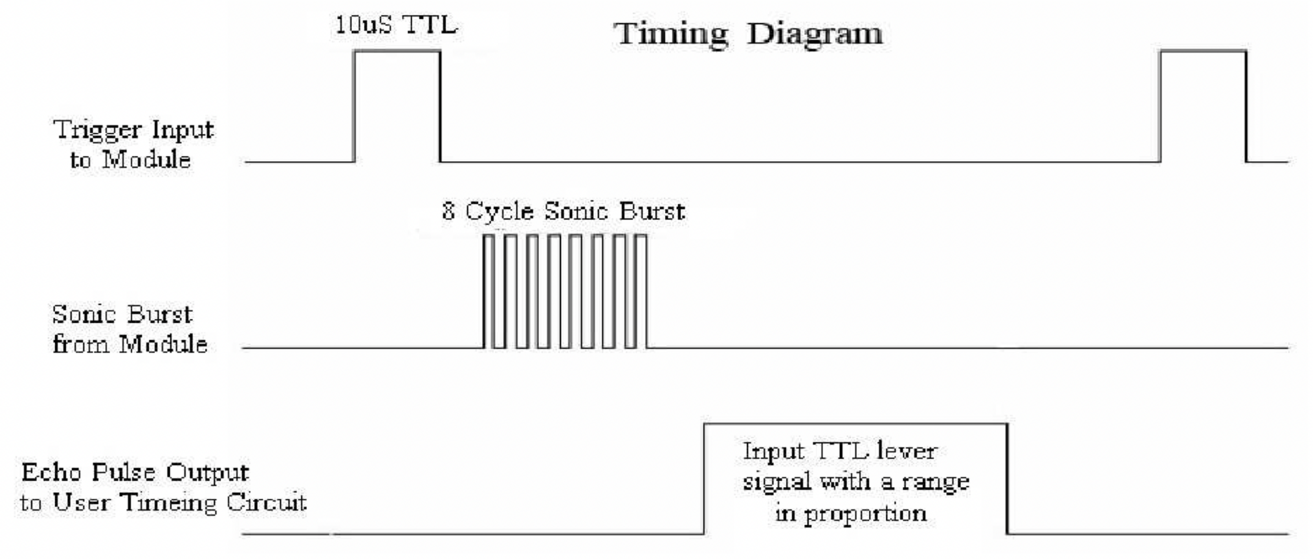
10uS TTL
Timing Diagram
Trigger Input to Module
8 Cycle Sonic Burst
Sonic Burst from Module
Echo Pulse Output to User Timeing Circuit
Input TTL lever signal with a range in proportion

Figure 5. Timing Diagram for the Ultrasonic Sensor Inputs

In total, three of the four capture compare modules in TIM3 are used. CC1 is used in PWM output mode to generate the 10 µsecond pulse. CC3 and CC4 are connected to the same pin in input capture mode. CC3 captures the rising edge, whereas CC4 captures the falling edge, and sends out an interrupt for the result to be calculated.

In contrast with a design where one CC module is used to capture both edges of the ECHO signal, this design is more resilient. In this design, the interrupt thrown by CC4 is not time sensitive, the 60 ms delay between measurements ensures that the CC3 and CC4 values will not be overwritten before the interrupt is handled. On the other hand, if the rising edge interrupt in the alternative design is not served before ECHO falls (which could happen if the object is very close), the measurement could fail or provide erroneous data.

In terms of data processing, each new measurement is first compared to the previous distance. If there is a change of more than 100 cms, the data point is discarded as none of our scenarios involve objects moving at that speed. The accepted data is then put through a decaying average filter, a low-pass filter that weighs the most recent sample the most heavily.

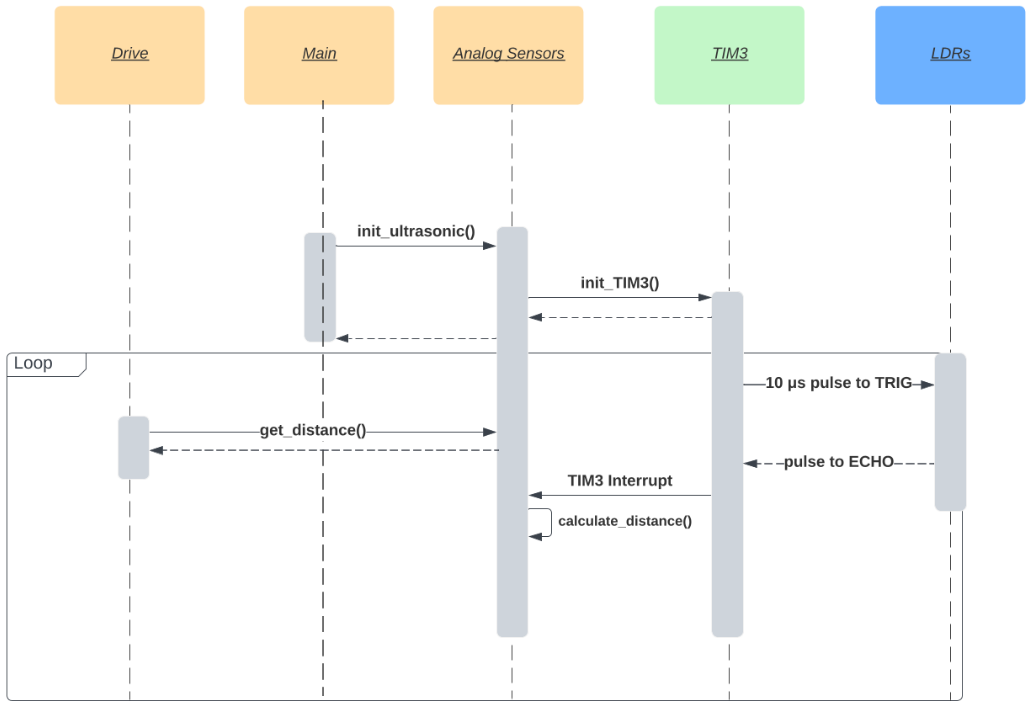

Figure 6. Sequence diagram for Ultrasonic Module

3.4 Analog Sensors

This module encapsulates ADC1 of the MCU which connects to 4 external channels in the following order: joystick y, joystick x, LDR right, LDR left. Their circuits and connection to the microcontroller is illustrated in Figure 6.

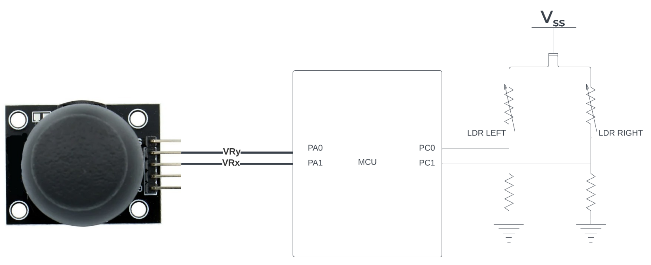

Figure 7. Analog Components Connection

Besides reading the analog values, this module is also responsible for calibration. Our drive logic does not take ADC values into account directly, yet, driver makes decision regarding how much the ADC readings derived from the calibration values. For example, if LDR left derives from calibration value a 1000 units while LDR right reading differs from calibration value only a 100, the Driver turns right.

Calibration value for joystick values are the ADC values read from the first conversion whereas calibration values for the LDR are the last ADC readings before state changes to AUTO in the Driver module.

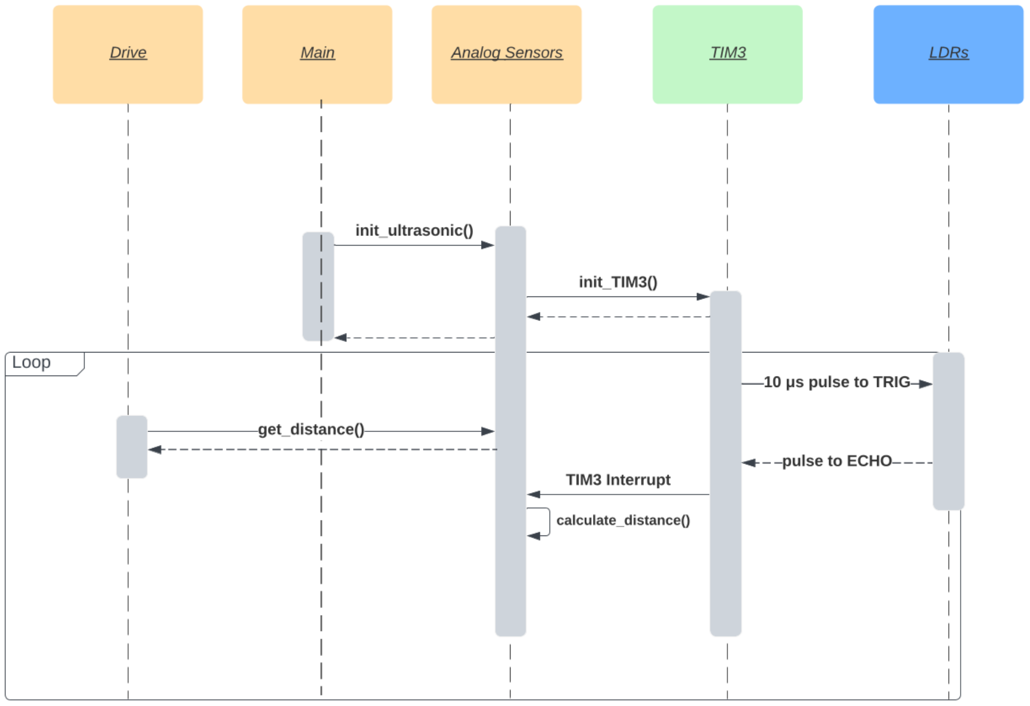

Figure 8. Analog Sensors Sequence Diagram

3.5 LEDs

LEDs module uses TIM6 to periodically update the headlight leds. This module also contains 5 states being LED_STOP, LED_FORWARD, LED_BACKWAR, LED_RIGHT and LED_LEFT. Driver module accesses this module using `set_leds_direction(LED_STATE)` function.

Headlight leds are connected to microprocessor using a common anode, led to pin mapping is given in the Table 2.

The LEDs are driven from the GPIOC port using the output mode. While implementing the blinking of the LEDs, we considered using one of the timer output compare peripherals of the MCU. However, as the blinking rate is 1 Hz, we came to the conclusion that replacing the interrupt and the short handler routine with a timer capture compare would not imply significant

LED	Pin
PC12	Front Right
PC13	Front Left
PC14	Back Right
PC15	Back Left

Table 2. Headlight Leds to Pins

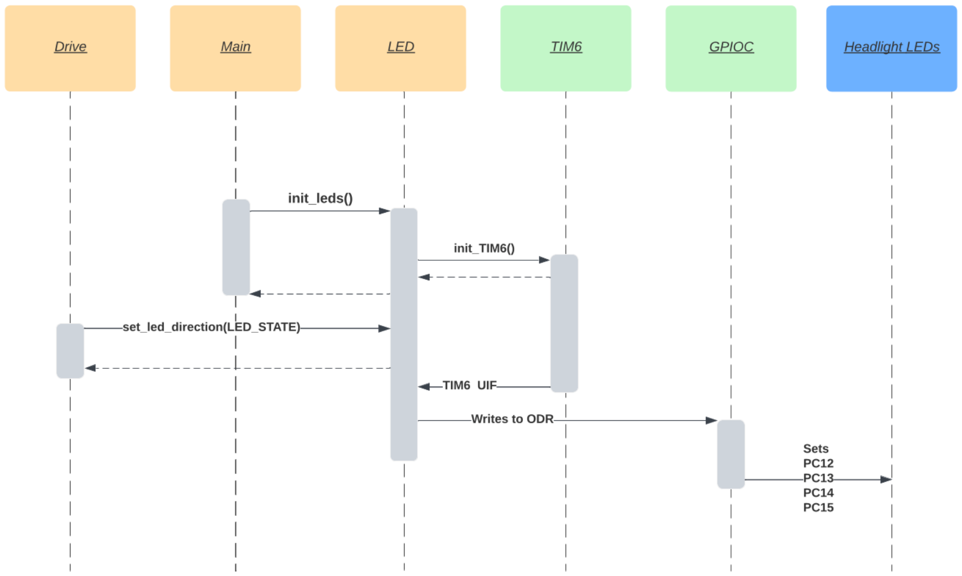

Figure 9. LED update sequence diagram

savings in energy. Thus, the simpler implementation that also left a valuable capture compare module open for future use was chosen.

TIM6 is used for generating interrupts at 2Hz where the interrupt handler triggers a LED update which writes to GPIOC according to the LED state and a variable that is inverted every interrupt. To generate 2Hz interrupts, the TIM6 prescaler is set to 15999 and ARR is set to 5 to minimize CNT updates.

Lifecycle of a single LED update is illustrated in figure 8.

3.6 Buttons

The Buttons module listens to clicks on the Blue Button and the Joystick button using the external interrupt feature of the MCU. This allows the MCU to quickly respond to stop requests and also avoid the waste of power polling the GPIO would entail.

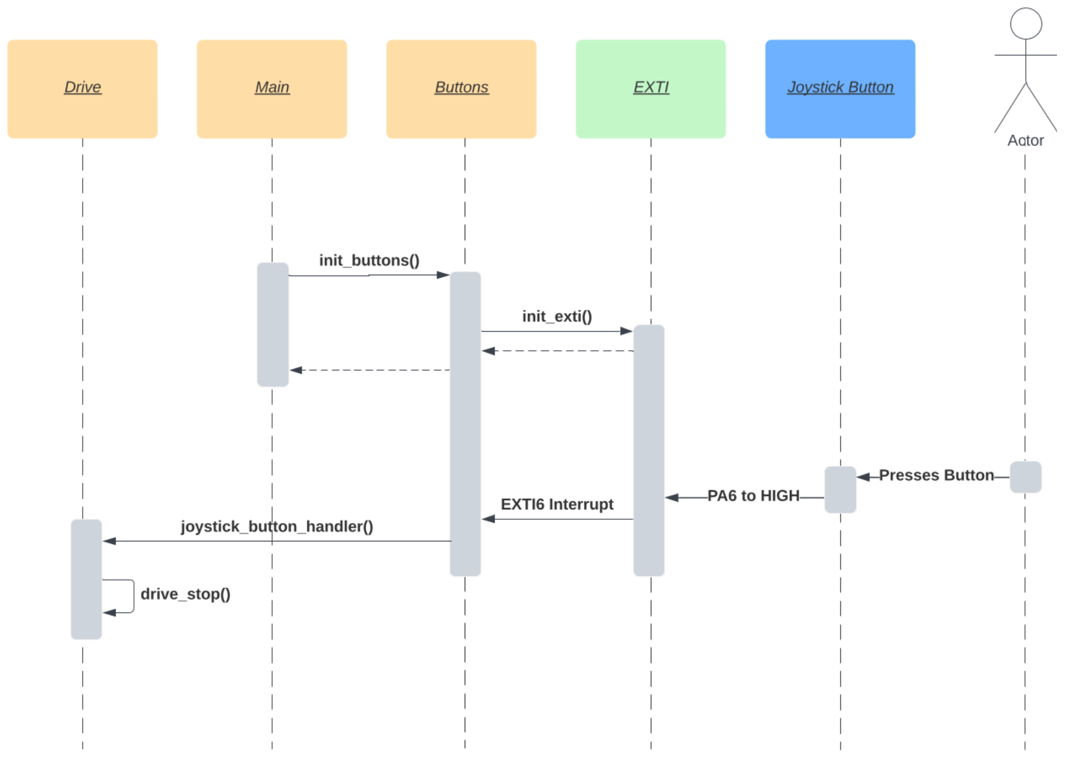

Figure 10. Joystick button press sequence diagram

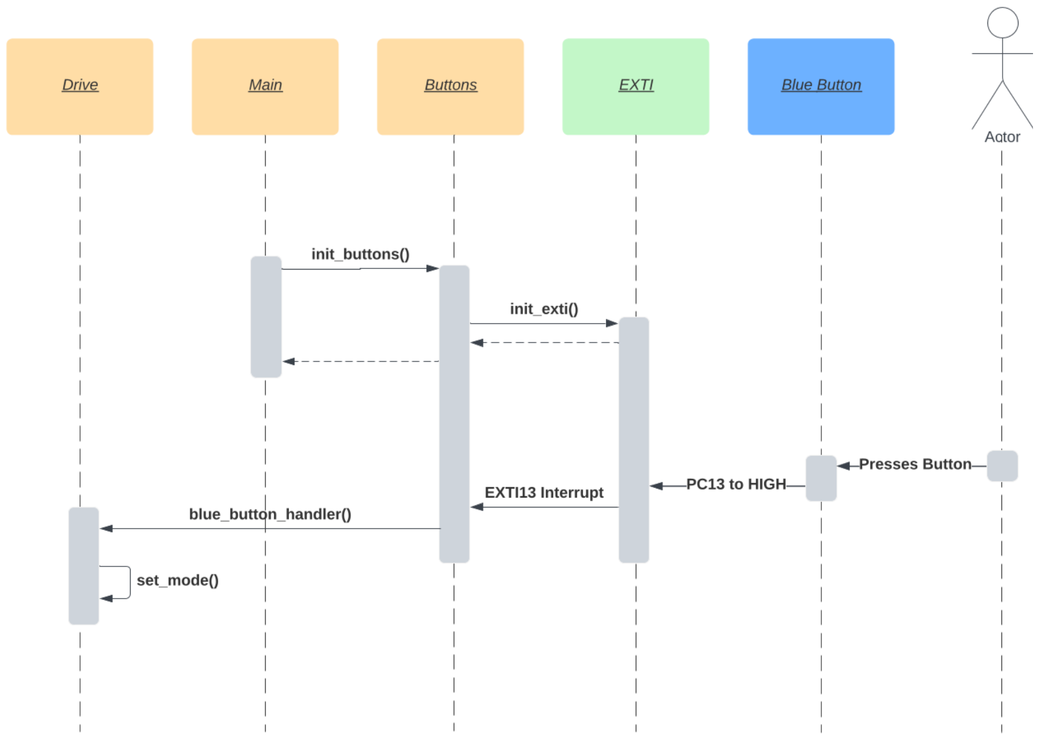

Figure 11. Blue button press sequence diagram

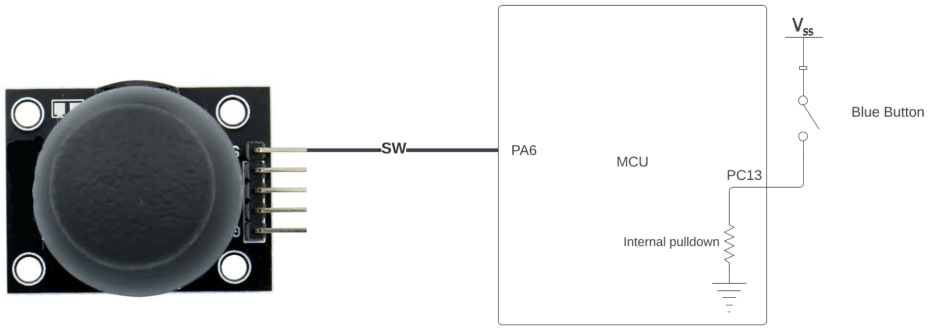

Figure 12. Circuit diagram for the buttons

3.7 Indicators

The indicators module inputs the Drive module state and sets the 3 LEDs on the MCU. Each led is matched to a set of states.

- LD1(Green LED) indicates that the car is in `MANUAL_OVERRIDE` mode.
- LD2(Blue LED) indicates the car is in manual mode, state can be either `MANUAL`, `MANUAL_OVERRIDE` or `MANUAL_STOP`
- LD3(Red LED) indicates that the breaks are engaged, the Driver is in either `MANUAL_STOP` or `AUTO_STOP` state.

By using these 3 indicators and the movement of the car, the user can exactly point the state of the car.

State	Indicators
MANUAL	LD1 is off, LD2 is on, LD3 is off
MANUAL_OVERRIDE	LD1 is on, LD2 is on, LD3 is off
MANUAL_STOP	LD1 is off, LD2 is on, LD3 is on
AUTO	LD1 is off, LD2 is off, LD3 is off, Car is moving
AUTO_WAIT	LD1 is off, LD2 is off, LD3 is off, Car is not moving
AUTO_STOP	LD1 is off, LD2 is off, LD3 is on

Table 3. Matching of states to state indicators

Indicator leds are updated at each Drive module update in the `drive()` method.

3.8 Motors

The motors module is in charge of receiving the speed and change in direction that the vehicle should go in, and writing the correct values to the motor driver based on these. The amount of power delivered to each motor is calculated with the formula `speed + direction` for the left motor and `speed - direction` for the right motor. The `speed` and `direction` values must be in the range of -4000 to 4000. The module also applies the emergency brakes in case the `stop()` function is called.

The motor driver hardware has 6 input pins. `ENA` controls the speed of the right motor and `IN1 IN2` together control whether the right motor turns clockwise, counterclockwise, or applies brakes. `ENB` controls the speed of the left motor and `IN3 IN4` together control whether the left motor turns clockwise, counterclockwise, or applies brakes. The state of the right motor corresponding to the inputs are as follows:

ENA	IN1	IN2	Motor State
0	x	x	Turning Freely
1	0	0	Brakes Applied
1	0	1	Turning Forwards
1	1	0	Turning Backwards
1	1	1	Brakes Applied

Table 4. Motor State w.r.t Driver Input

The module controls IN1-4 using GPIOC and outputs a PWM signal to ENA and ENB using the two capture compare modules of TIM15. The modules are configured in PWM mode and the frequency is 250 Hz. PWM mode, instead of output compare mode was used, as it can control the output signal without requiring any interrupts. This saves power and decreases the load on the CPU.

Implementing positive motor control using the encoder sensors of the car was considered. However, this was judged to be unnecessary, as neither the manual nor the autonomous mode requires precise control over the speed or direction of the car.