ANDREW AND ERNA VITERBI FACULTY OF ELECTRICAL ENGINEERING

ICST

CRML
Control Robotics and Machine Learning Laboratory

TECHNION
Israel Institute of Technology

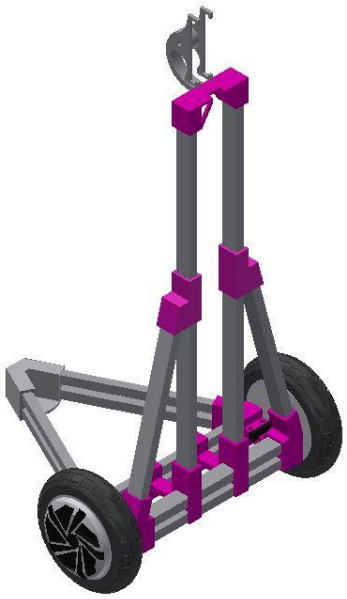
Project topic:

IoT, Control robotics, & Embedded systems

Project name:

Telemedicine Robot Assistant

(project-EVA)



Delivered by:

Christian Shakkour, chrisshakkour@gmail.com

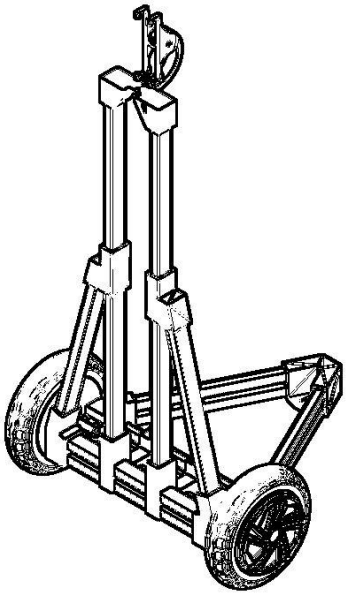
Instructors:

Itai Dbran, idabran.cs.technion@gmail.com

Kobi Kohai, kohai@ee.technion.ac.il

Semester, Year:

Winter, 2020/2021



1. With Thanks & Appreciations

This section is dedicated to thanking everyone who has contributed to this project, I would like to start with thanking companies and brands for their free services.

Thanks Autodesk inventor for a free student License, all mechanical parts were designed and visualized with Autodesk inventor. Thank you Ultimaker for a free & open-source software, Ultimaker Cura was used to slice and tune 3D-printing parameters to print the designed parts. Thank you Technion for a free MATLAB & OrCad membership, MATLAB was used to simulate, tune, & calibrate software controllers prior to deploying on the robot, OrCad was helpful in building the power distribution controller where schematic was superior compared to breadboard testing and blueprint sketching. Thank you STM Microelectronics & ARM Inc for providing a fully detailed documents, datasheets, & tutorials that were used extensively in this project, & thank you for your open forum quick support that helped debug software issues.

Firstly, many thanks to Itai our instructor who inspired and believed in making this project happen, Thanks to both ICST, & CRML Labs for lending us components, & resources to build our robot, Thanks Koby for your guidance along the way.

Table of Contents:

1. With Thanks & Appreciations 2
2. Abstract 4
3. Acronyms and definitions 5
4. project Overview 6
   4.1. Features 6
   4.2. Sensors 6
5. Robot system 7
   5.1. System Block diagram 7
   5.2. Component's list 8
   5.3. Required Tools 8
   5.4. Mechanical Design 9
   5.5. Power system 11
      5.5.1. PDU 12
   5.6. Microcontroller 13
      5.6.1. Features 13
      5.6.2. Architecture 14
      5.6.3. Clock distribution 15
      5.6.4. HW Timers 16
      5.6.5. PWM 18
      5.6.6. I2C 19
      5.6.7. UART 20
      5.6.8. SPI 21
      5.6.9. NVIC 22
   5.7. Software Architecture 23
      5.7.1. Main 23
      5.7.2. Scheduler 23
      5.7.3. Kinematics controller 23
      5.7.4. Bluetooth message handler 24
      5.7.5. Practical implementation 24
   5.8. Software Drivetrain controller 25
      5.8.1. Drivetrain controller 26

5.8.2. Drivetrain controller Equations 27
5.8.3. Gaussian transform 28
5.8.4. Motion Analysis 29
5.8.5. Compute power & resources 32
6. References 33
7. Index 33

2. Abstract

Telemedicine or Telehealth is the distribution of health-related services and information via electronic information and telecommunication technologies. It allows long-distance patient and clinician contact, care, advice, reminders, education, intervention, monitoring, and remote admissions. In this project we aim to build an easy to use, yet intelligent Robot assistant that will serve as a RC gateway between long distance patients and their families, doctors, loved ones providing the patient with care, monitoring, & Telemedicine. The robot itself is a 3-wheel differential drive robot built on a 3D printed chassis and a STM32F407VET microcontroller Featuring an Arm-Cortex-M4 168[MHz] Core. The robot is equipped with a mobile device (Tablet) to enable video conferencing. anyone with permissions can call the robot and remote control it from far away to monitor the patient/family member. The robot built in this project fully occupies these features, calling the robot from far away and driving in the patient's home to firstly check on him and secondly check on his health and satisfaction, our conclusions from this robot are, firstly this concept could be implemented into a real life product with many use cases, secondly it is a product customers would like to use but don't have the will to purchase because cheaper solutions can bring the same results at zero investment like using a smartphone to video conference, Etc.

3. Acronyms and definitions

Acronym	Definition
CMSIS	Cortex microcontroller software interface standard
FLASH	Flash memory
API	Application programming interface
GPIO	General purpose I/O's
HAL	Hardware abstraction layer
I2C	Inter-integrated circuit
NVIC	Nested vectored interrupt controller
SPI	Serial peripheral interface
SysTick	System tick timer
TIM	Advanced-control, general-purpose or basic timer
UART	Universal asynchronous receiver/transmitter
PWM	Pulse width modulation
PDU	Power distribution unit
HW	Hardware
SW	Software
RWS	Right wheel speed
LWS	Left wheel speed
RC	Remote control
SWD	Serial wire debug
JTAG	Joint Test Action Group
PLL	Phase locked loop
APB	Advanced peripheral bus
UV	U*nmanned vehicle*

4. project Overview

Project EVA is a combination of 2 systems, an embedded real-time Robot system & an IoT software agent. the Robot system is independently handled with a microcontroller equipped with sensors, power distribution & management components, a camera, and an intelligent Real-time software, whereas the IoT agent is built upon 2 applications running on different mobile devices connected via a firebase. The Robotic system receives commands from the IoT software agent via a Bluetooth wireless communication protocol as shown below.

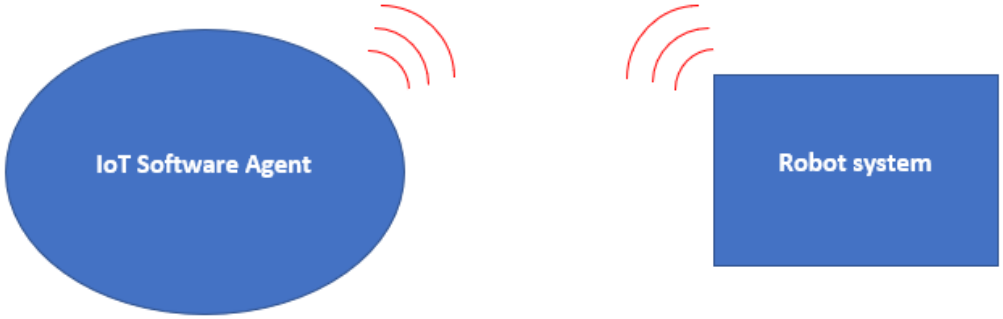


Figure 1: High level system overview

4.1. Features

1. Remote controlled UV.
2. Smooth differential drive robot with real time motion response.
3. Live video conferencing.
4. Failsafe mode.
5. Obstacle collision avoidance.

4.2. Sensors

1. Proximity (HCSR).
2. Accelerometer.
3. Pixy Camera.

5. Robot system

In this section you can find the different aspects for building EVA robot, the mechanical, Electrical, & software controlling the robot.

5.1. System Block diagram

Following system block diagram, on the left in green are sensor modules feeding the controller with desired data for monitoring & controlling the robot, the Bluetooth module communicates with a mobile device and receives control data to maneuver the robot. the pixy camera detects pre-trained color objects and barcodes. The proximity sensors are meant to give the robot the confidence of moving around without colliding. And the Accelerometer is used to monitor robot motion. On the right colored in yellow are the 3-Phase BLDC motor drivers receiving PWM signals to control the motor speed.

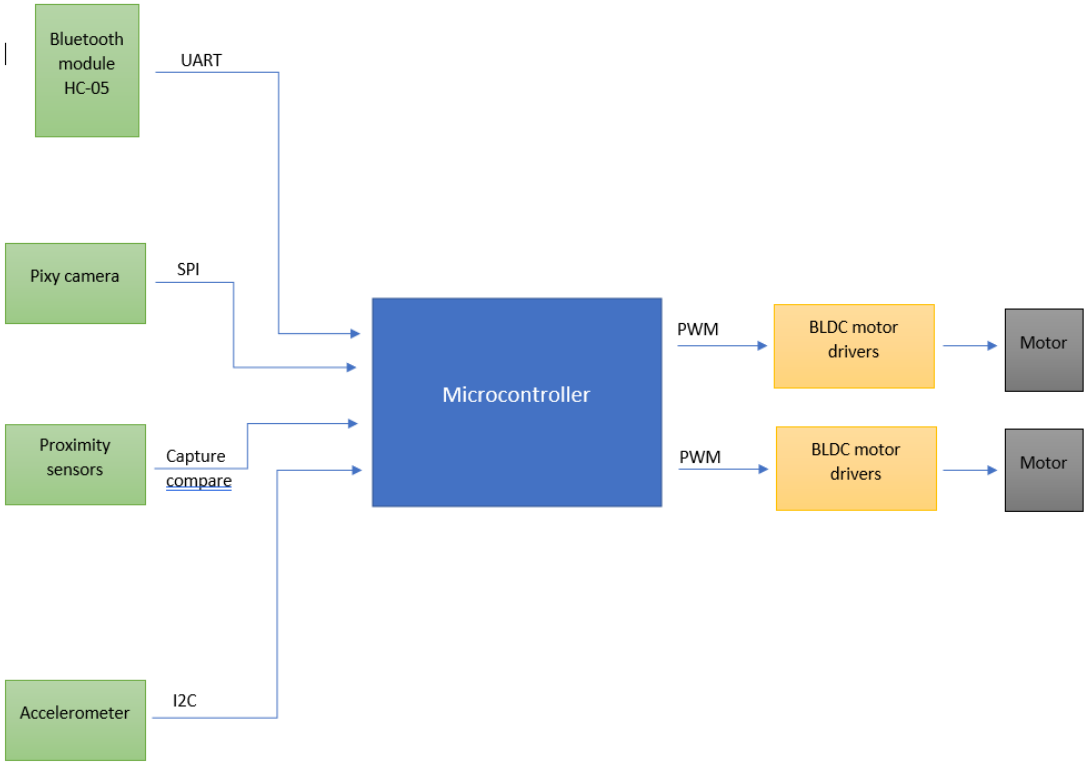


Figure 2: system block diagram

The following sections in this chapter will cover the communication protocols & methods with those modules, and how the controller handles them all continuously in parallel.

5.2. Component's list

in this project 2 tablet devices were used alongside custom built hardware system, following the custom system components list.

NO	Component	Description	link
1	STM32F407VET6	ARM-Cortex-M4 Microcontroller	STM32F4
2	STM32F407VET6 black board	Development board	black board
3	Pixy Cam	Color/barcode detection camera	Pixy2
4	BLDC motor drivers	3-Phase motor drivers	BLDC
5	Accelerometer	6 Axis Accelerometer	MPU 6050
6	Proximity	HC-sr04	HC-SR04
7	Bluetooth module	HC-05 Bluetooth module	HC-05
8	Hover board wheels	8-inch 3 phase brushless motor	Wheels
9	Passive components	Capacitors, resistors, Etc.	
10	3D printed parts	Mechanical joints holding the chassis	
11	Hookup wires	Fast plug hookup wires	
12	Linear voltage regulators	7805 + 7806 5v and 6v 1Amp regulators	LM7805

Table 1: components table

5.3. Required Tools

NO	Tool	Use case	link
1	Segger J-link base	ARM-Cortex-M debbuger	
2	Keysight oscilloscope	Debugging PWM, I2C, Timing, SPI, Etc.	
3	Bakkon soldering iron	Building custom PCB circuits (PDU)	
4	UNI-T multimeter	Measuring voltage, current, Etc.	
5	Technician tools	Cutter, long nose plyer, Etc.	
6	Hot glue gun	Gluing components & parts	
7	Screws & bolts	Tightening 3D printed parts together.	
8	3D printer	Printing joint parts	

Table 2: tools table

5.4. Mechanical Design

The robot is built from plastic tubes joint together in a press fit mechanism with custom 3D-printed parts. All parts are originally modeled via Autodesk inventor specifically for this project and printed with a custom 300x300mm private 3D-printer.

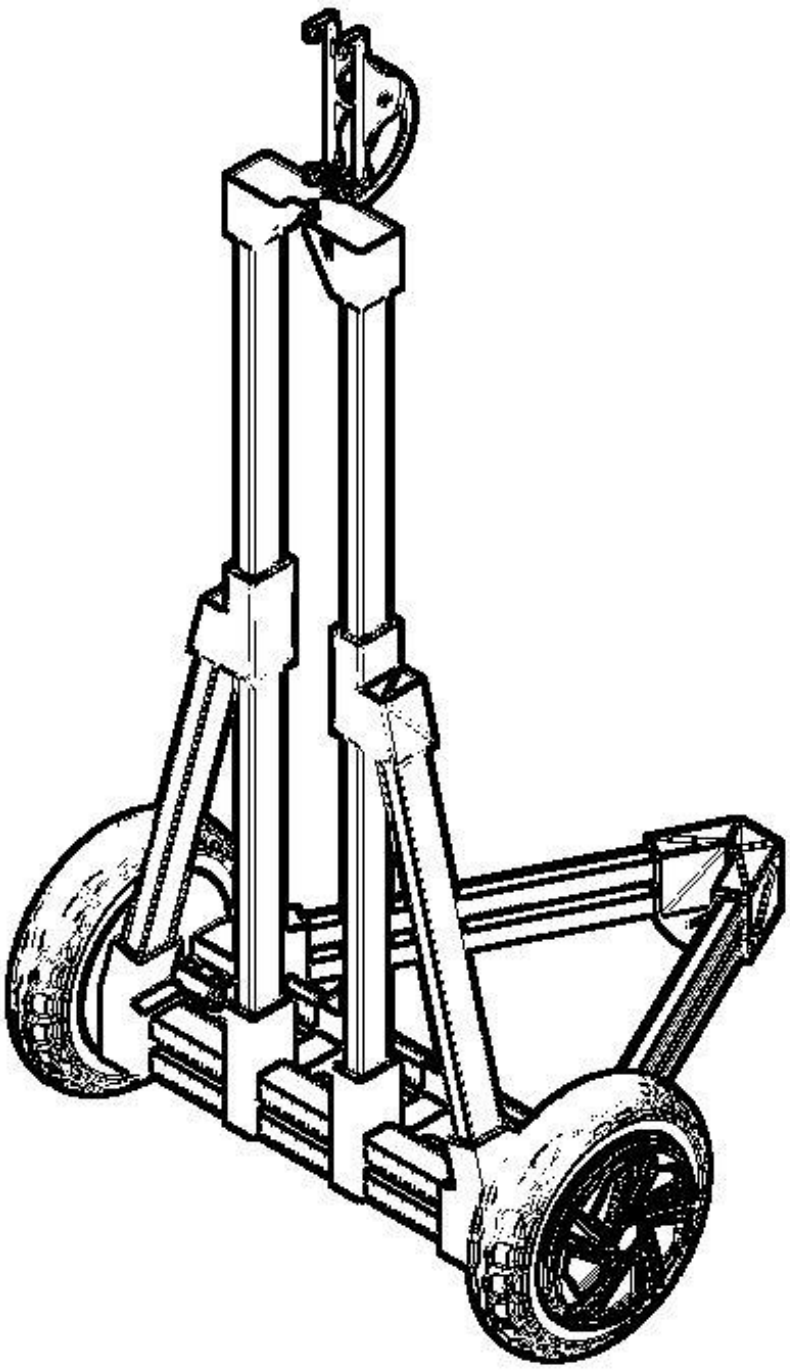


Figure 3: Shaded black&white robot chasis

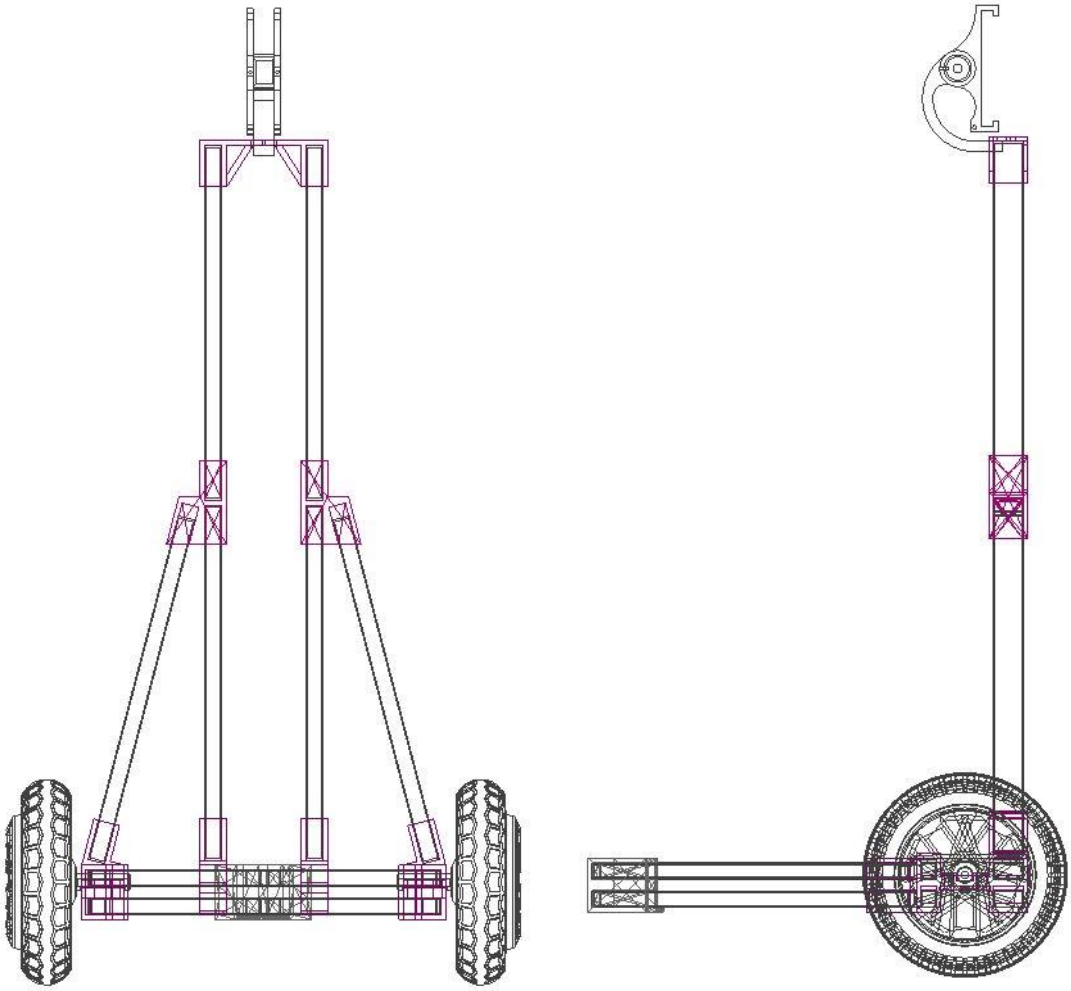


Figure 4: chassis Front & Left transparent view.

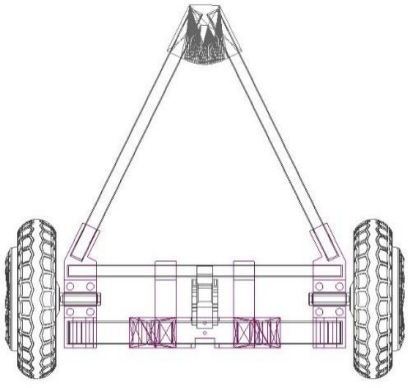


Figure 5: chassis bottom transparent view.

5.5. Power system

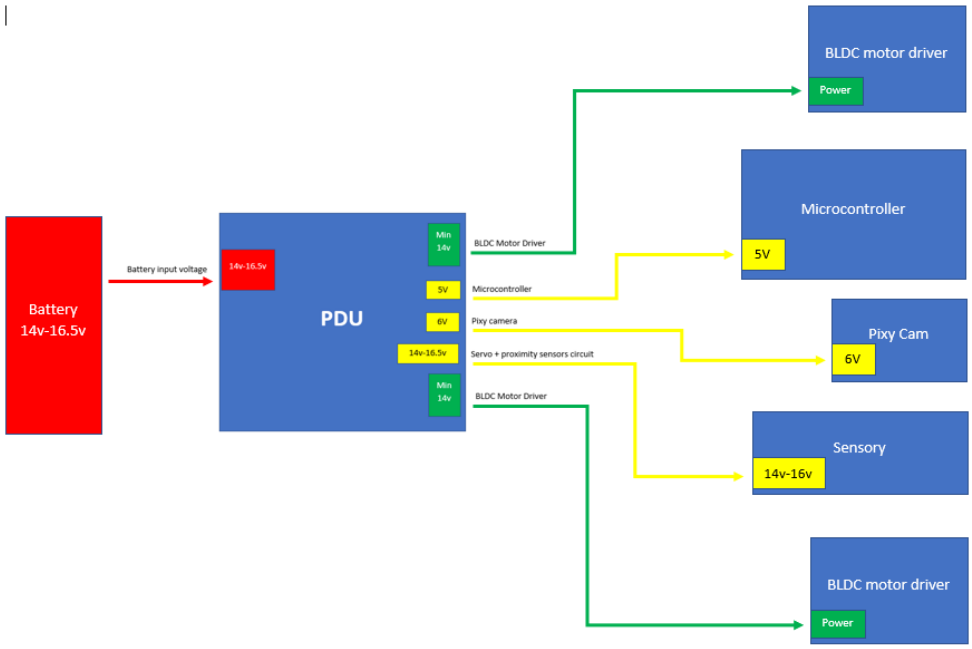


Figure 6: power system block diagram

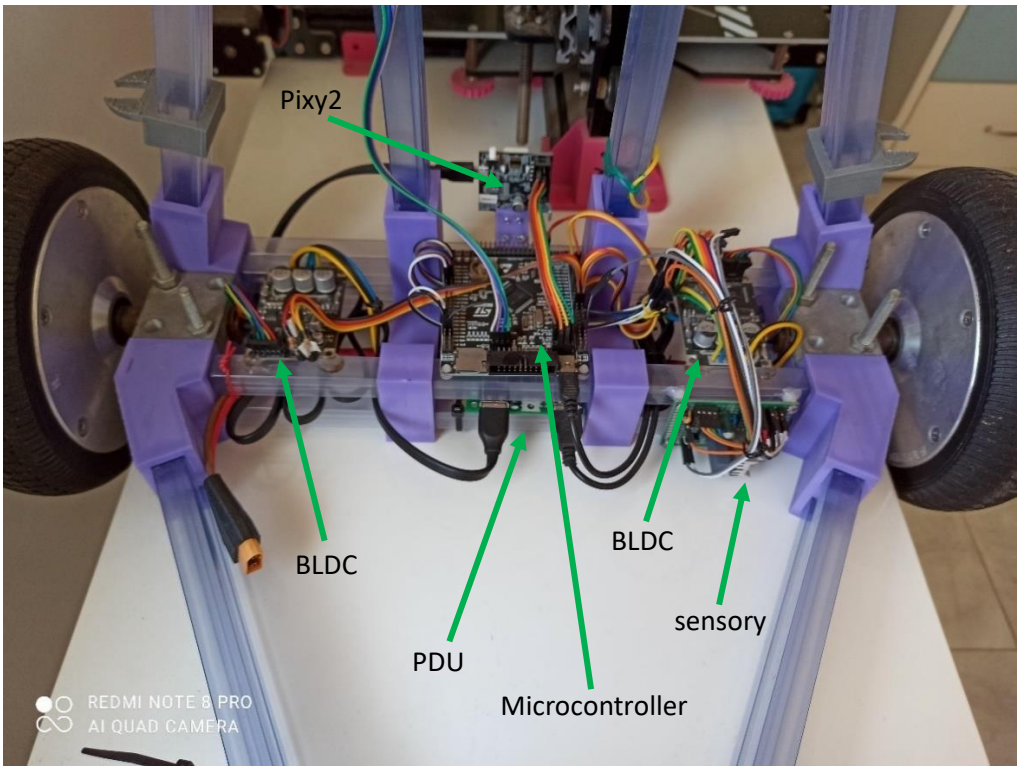


Figure 7: pysical implementation & connections

5.5.1. PDU

The power distribution unit, distributes the power that is being consumed from the battery to the different modules at different working voltages, the microcontroller board requires 5v up to 500mAmp of current to work effectively, the 5 Volts are being provided by a 7805 Linear voltage regulator supporting 1 Amp of current with efficiency of 70% meaning our application can drain only 700mAmps from the regulator prior to damage because the 300mAmps remaining are wasted on heat and power transfer. The same goes to the 6v port feeding the pixy cam with a 7806 Linear voltage regulator supporting 1Amp at 70% efficiency. The BLDC motor Power ports are directly connected to the battery source with a diode and a varistor (variable resistor) to protect reverse voltage and limit spike disturbance current or motor flyback current.

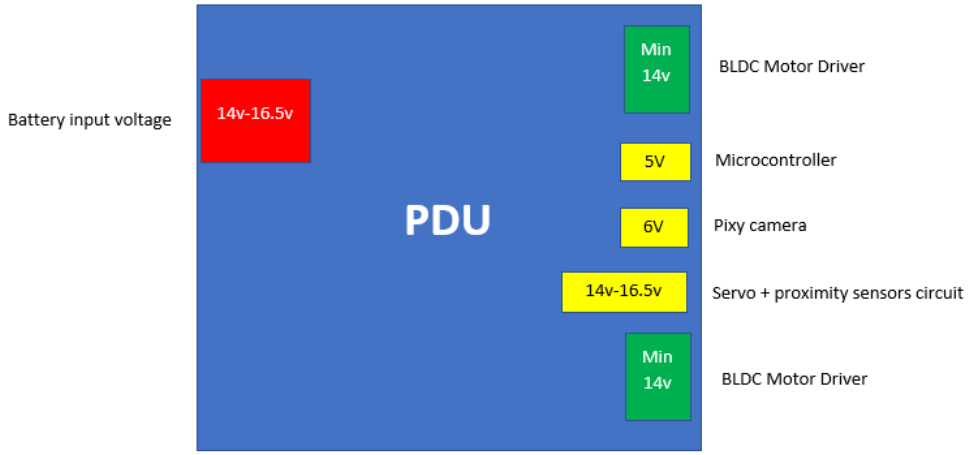


Figure 8: PDU block diagram

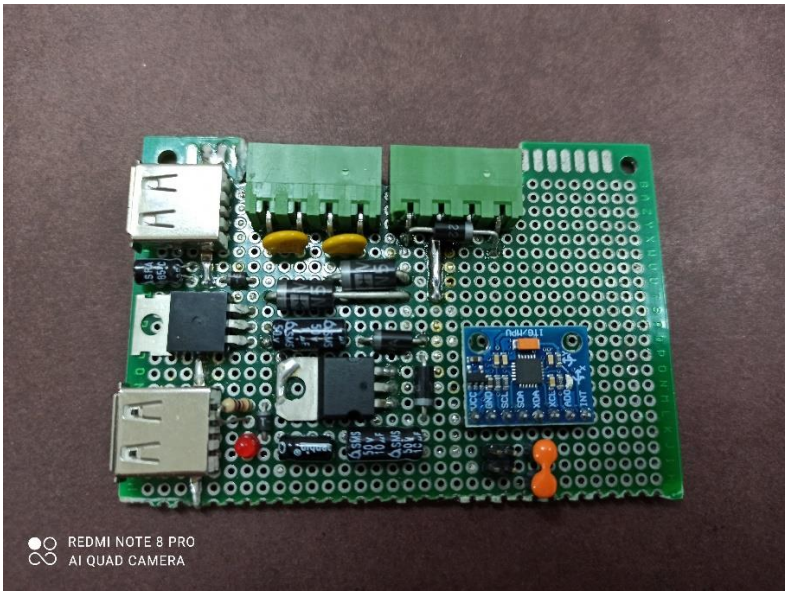


Figure 9: PDU custom circuit board

5.6. Microcontroller

5.6.1. Features

The microcontroller used in this project is STM32F407VET featuring an ARM-Cortex-M4 Core running at 168 MHz, with 512Kbyte Flash memory, supporting JTAG and SWD for debugging.

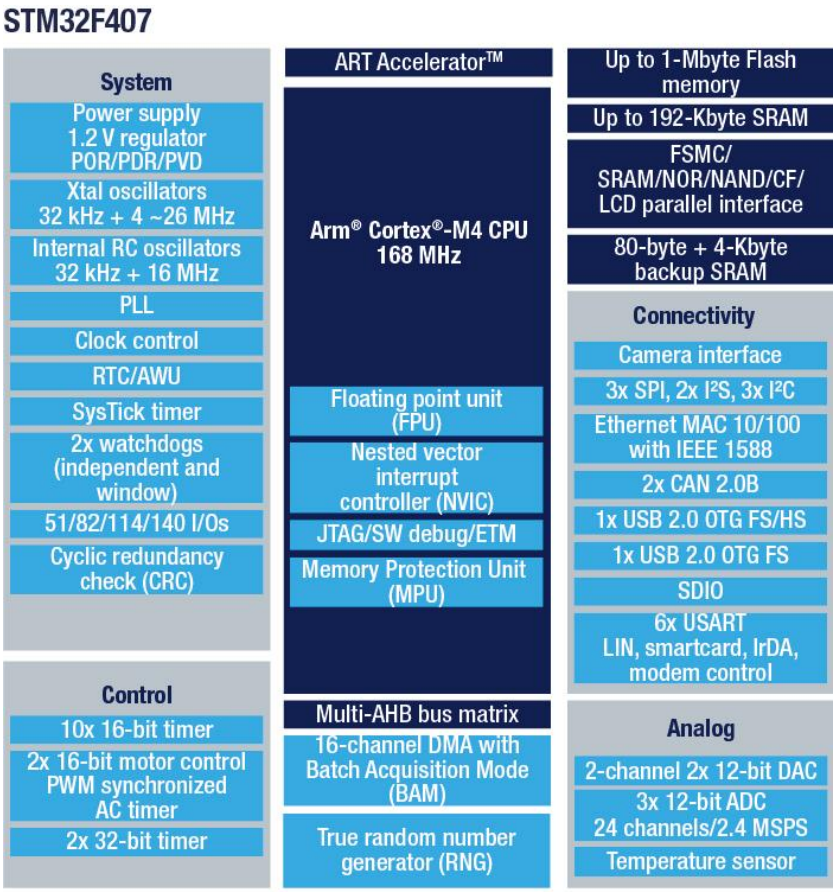


Figure 10: High level STM32F407 device features

Microcontroller main features:

- 168 MHz max CPU frequency
- 512 KB Flash
- GPIO (82) with external interrupt capability
- Timers (14)
- I2C (3)
- I2S (2)
- USART (4)
- SPI (3)

5.6.2. Architecture

Microcontroller's architectures are built of a processor covering less than 10% of the die fabric area and the rest are independent HW modules like timers, peripheral engines, PMU's, internal bus infrastructure, PLL's, clock distribution fabric, & Memory systems/controllers.

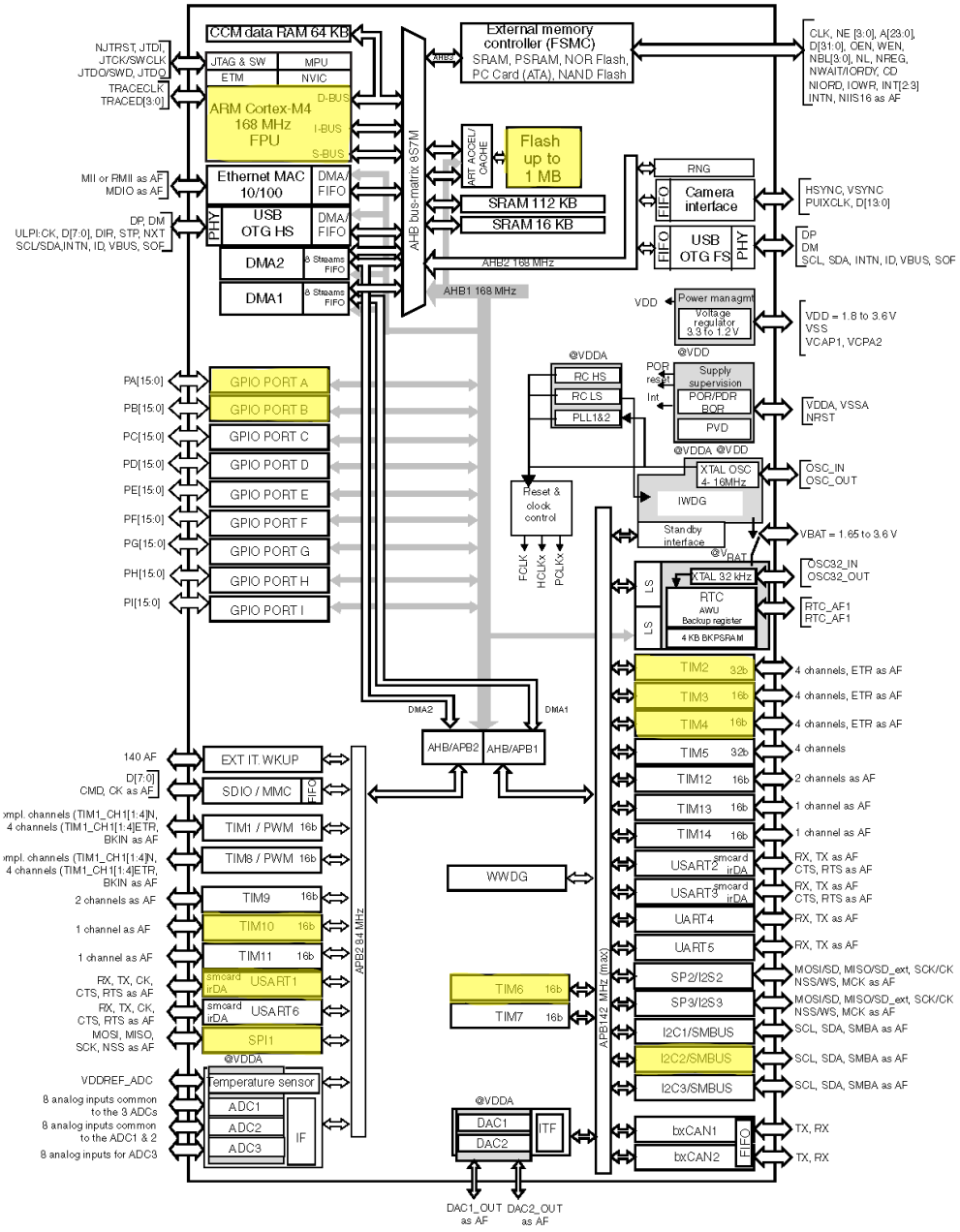


Figure 11: Device Architecture block diagram

Highlighted in yellow are the HW modules that have been used in this project. in the following sections we will shortly cover the functionality and implementation for those HW modules.

5.6.3. Clock distribution

Following Clock distribution tree for STM32F407VET6 microcontroller, the system is feed with internal Low-speed and High-speed Oscillators, LSI-RC-32Khz & HSI-RC-16Mhz

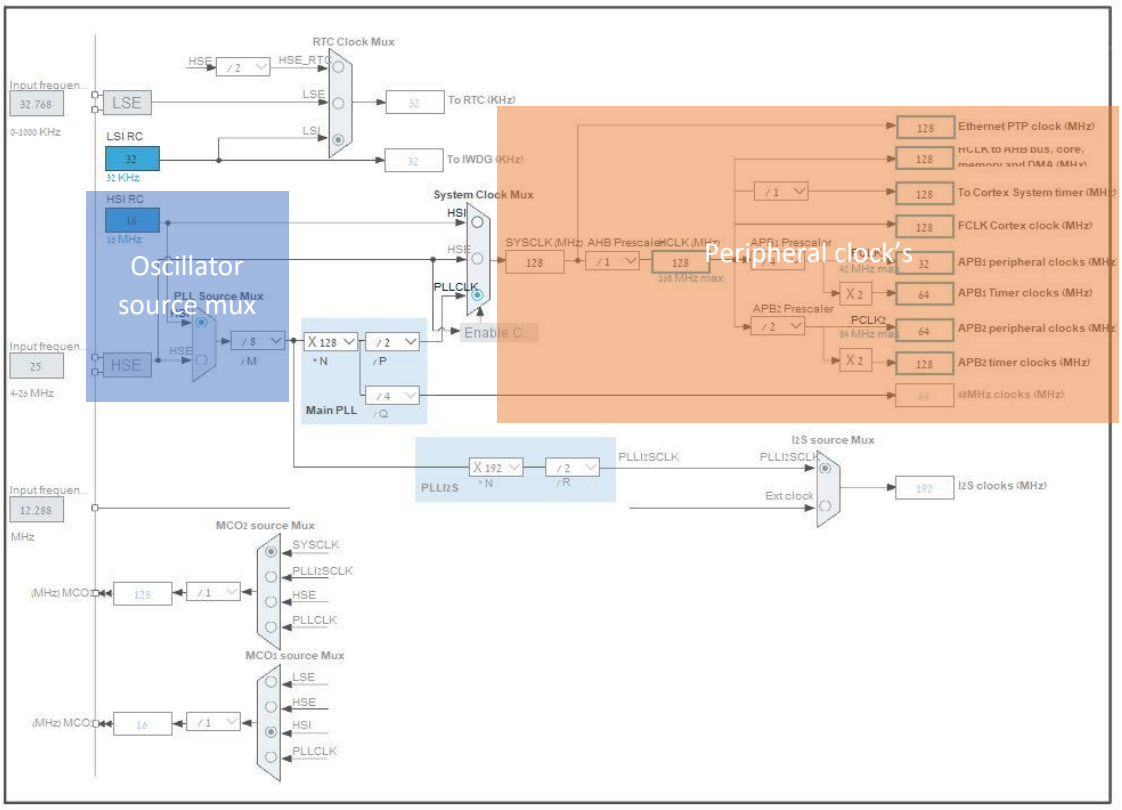


Figure 12: clock distribution tree GUI

Oscillator source mux:

This mux sets which High-speed oscillator feeds the system PLL's it selects between the internal Oscillator within the die fabric, and the external oscillator provided on the development board directly connected into the right controller pins. the external oscillator is usually used in applications where a super precise clock demands are required, for example driving a three-phase motor, capturing fast signals, or driving stepping motors and many other industrial uses, in this project the internal Oscillator was sufficient and precise enough to fulfill timing demands, communication protocols and time capturing events.

Peripheral clocks:

The device architecture block diagram has 2 advanced peripheral buses, each bus runs at different speed and any engine connected to those buses corresponds to the buss frequency, for example APB 1 timers are running at 64Mhz meaning Tim2+Tim3+Tim4 are being feed with a 64Mhz clock. However, APB2 is running at 128Mhz so Tim10 is being feed with 128Mhz. choosing the clock division is quite challenging when microcontroller resources are limited hence it is preferable to choose a larger MCU to be on the safe side.

5.6.4. HW Timers

The STM32F407 device includes two advanced-control timers, eight general-purpose timers, two basic timers and two watchdog timers. All timer counters can be frozen in debug mode.

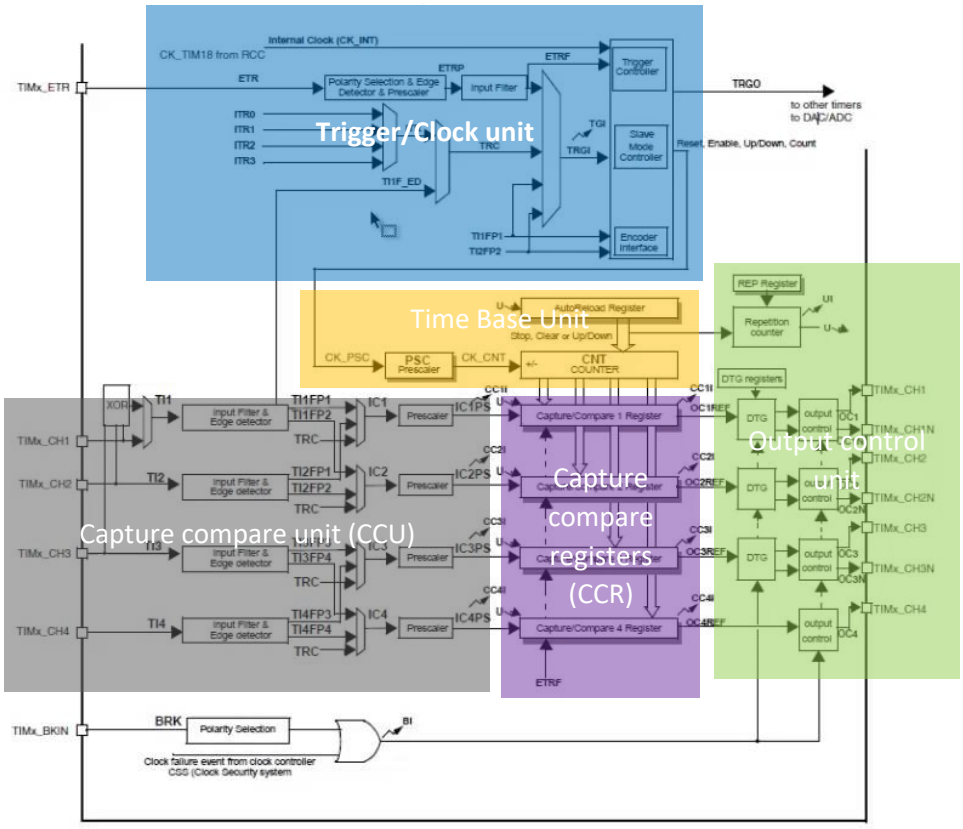


Figure 13: general purpose timer block diagram

In this project Tim2, Tim3, tim4, tim6, & Tim10 are utilized by the project demands and their capabilities, Tim6 is a basic 16-bit timer used for setting the scheduler's fixed cadence at 50Hz, meaning every 20ms Tim6 will initiate an interrupt to the NVIC directing the Program counter (PC) to a scheduler function executing user code. Tim2, tim3, & Tim4 are fully featured general purpose timers with auto-reload, up/down counter and a 16-bit prescaler. Each timer features 4 independent channels for input capture compare, output compare, PWM generation, or one-pulse mode output. Tim2 is a 32-bit timer used for generating the PWM signal driving both the software drivetrain module, & the HW motor controllers. Tim3 & Tim4 are 16-bit timers, Tim3 is used to generate a 10-microsecond pulse triggering the ultra-sonic proximity sensors (HC-SR04), Tim4 is set to capture compare mode capturing the time taken for a sonic wave to reach an object thus calculating the proximity. Tim10 is a 16-bit auto preload timer set to PWM output channel generating a PWN signal driving servo motors.

Trigger/Clock unit (blue area):

This unit handles Any internal system input triggers and output triggers that can be configured by the user, this unit also prescale's and sets the timer's working frequency also user configured.

Time base unit (orange area):

This unit has a basic counter, an auto-reload register, & a repetition counter, this unit counts up/down and resets once overflow/underflow is reached, the repetition counter is responsible for limiting the iterations the counter must go through before halting.

Capture compare unit (grey area):

This unit triggers an interrupt when one of the inputs correspond to an event like a rising/falling edge, this unit is the key behind measuring pulse width signals.

Capture compare registers (violet area):

This unit has 4 registers holding user defined values, when the counter (located in the time-based unit) reaches any of the stored values in the registers, an event is triggered to the output control unit to respond accordingly.

Output control unit (green area):

This unit controls the output behavior corresponding to a capture compare event, the output behavior is user configured.

Timers Configuration GUI:

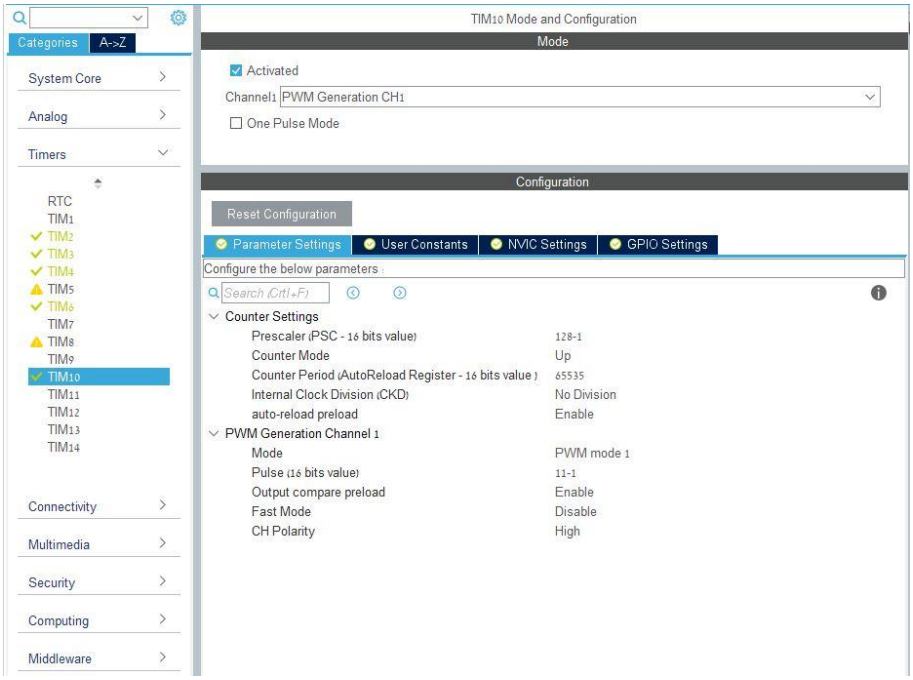


Figure 14: Tim_x config GUI

5.6.5. PWM

PWM stands for Pulse Width Modulation where the width of the signal in logic high varies in time, PWM is useful for many applications, generating analog waves, motor control, timing events and much more. Following a brief explanation of a PWM engine in STM32.

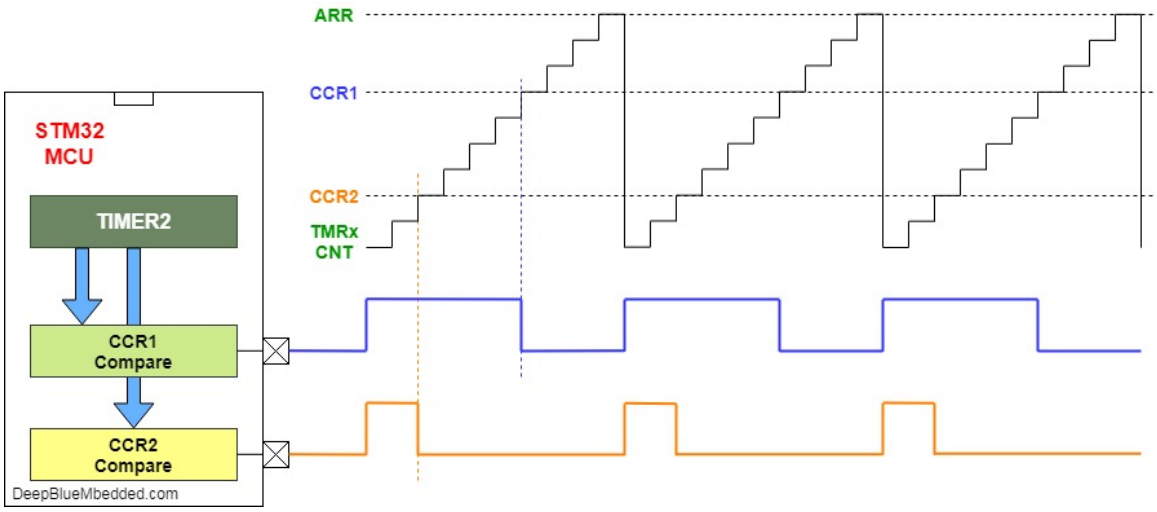


Figure 15: PWM up timer wavform

The PWM engine is integrated within a timer module discussed earlier in this documentation, a timer is set to count up/down, once overflow occurs the timer resets and starts counting back again infinitely, to generate the PWM signal the CCR must be loaded with a desired value, once the counter reaches this value the capture compare unit sets the out signal low. The counter sets back the signal to a logic high at every reset. In order to change the modulation, one must load the CCR with the desired modulation value. The counter sets the frequency of the PWM and the CCR value sets the duty cycle, to calculate the duty cycle divide the value in the CCR with the the timer expiration value for example a timer counter from Zero to 255 with CCR value of 127 will generate a 50% duty cycle PWM signal.

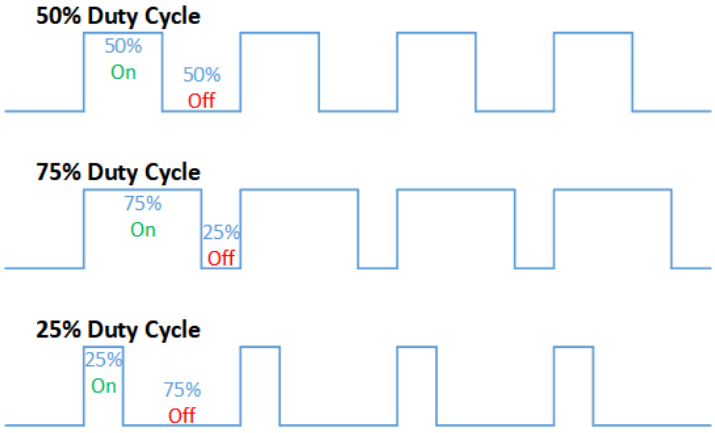


Figure 16: pulse width modulation examples

5.6.6. I2C

I2C or IIC stands for Inter-Integrated circuit is a synchronous multi master multi slave bus topology serial communication protocol, it is widely used for attaching lower-speed peripheral ICs to processors and microcontroller in short-distance, or intra-board communication. I2C is appropriate for peripherals where simplicity and low manufacturing cost are more important than speed.

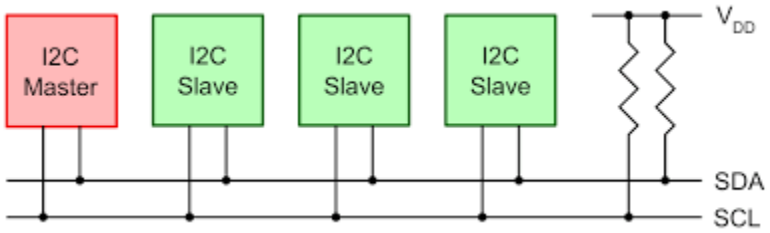


Figure 10: I2C bus topology

The I2C message consists of a starting condition an address frame, read/write bit, acknowledge bit, data frames, and a stop sequence. Following below an example of a message with two single byte data frames. The address frame limits the use of I2C to a finite number of master/slave devices, the number of bits to be transmitted as a function of data frames is:

$$num\ of\ bits = start + address + R/W + Dataframes * (1 + 8) + stop$$

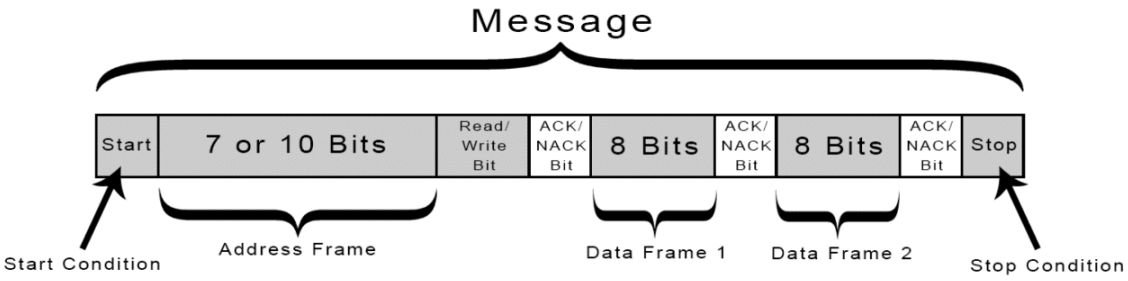


Figure 18: I2C message structure

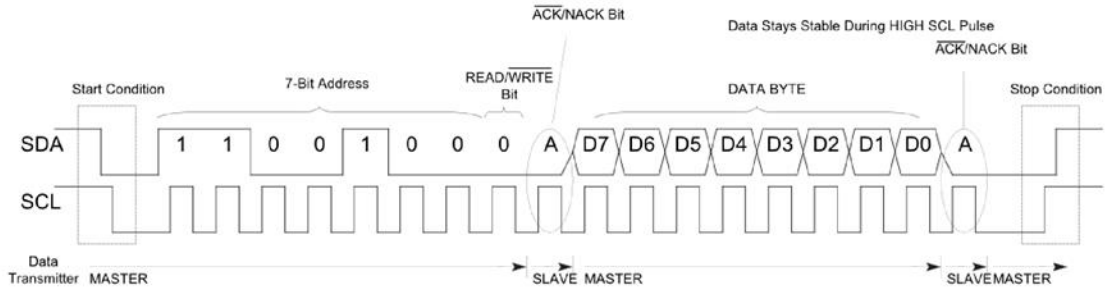


Figure 19: I2C single data frame message

5.6.7. UART

UART stands for Universal Aysyncrounouse Receive-transmit, The UART takes bytes of data and transmits the individual bits in a sequential fashion, it only enables serial data exchange between two devices. It can operate at simplex, half-duplex, or full-duplex modes.

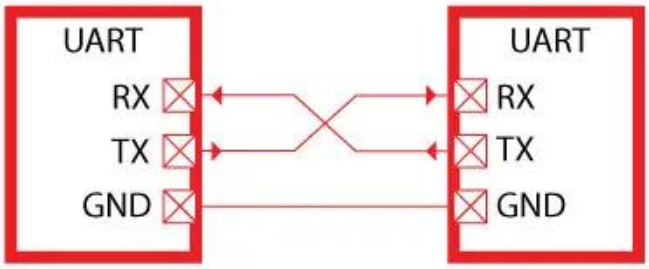


Figure 20: UART connection configuration

Transmitting and receiving UARTs must be set for the same bit speed, character length, parity, and stop bits for proper operation. The receiving UART may detect some mismatched settings and set a "framing error" flag bit for the host system; in exceptional cases, the receiving UART will produce an erratic stream of mutilated characters and transfer them to the host system.

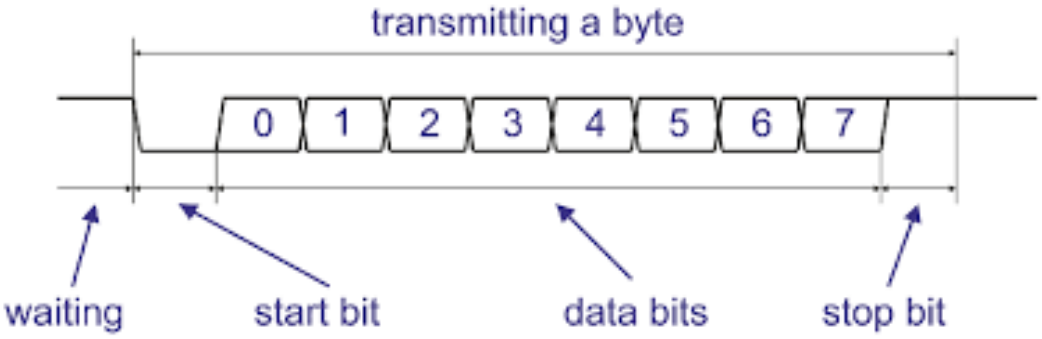


Figure 21: UART message frame

5.6.8. SPI

The Serial Peripheral interface (SPI) is a synchronous serial communication interface specification used for short-distance communication, primarily in embedded systems. SPI devices communicate in full duplex mode using a master-slave architecture with a single master. The master device originates the frame for reading and writing. Multiple slave-devices are supported through selection with individual slave select (SS), sometimes called chip select (CS), lines.

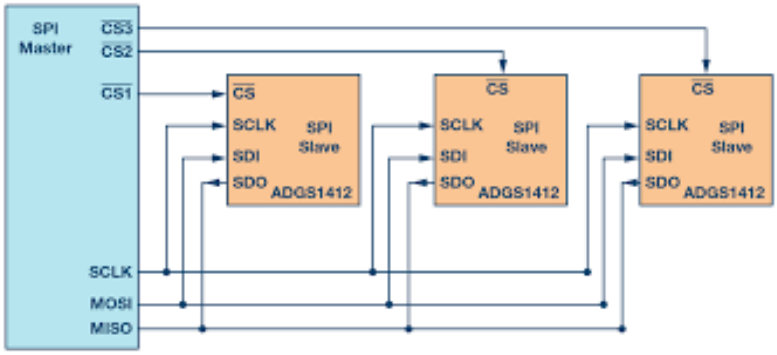


Figure 22: SPI master-slave bus

- SCLK: Serial Clock (output from master)
- MOSI: Master Out Slave In (data output from master)
- MISO: Master in Slave Out (data output from slave)
- SS: Slave Select (often active-low, output from master)

Following an SPI frame example, cs line is forced low by the master to notify the slave module of a coming data transmission, the MOSI signal is driving the byte sent from the master to the slave, however the MISO signal is driving the corresponding byte from the slave to the master, the clock signal is generated by the master to guarantee a synchronous

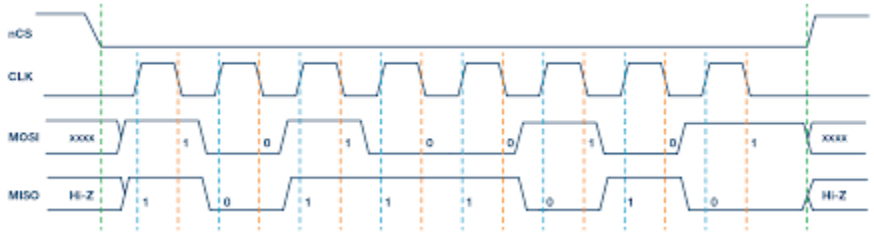


Figure 11: SPI waveform example

5.6.9. NVIC

NVIC stands for Nested Vectored Interrupt Controller, this controller has the capability to prioritize interrupts and execute the higher priority interrupt, and hence comes the name nested. Vectored however, means that the controller knows what type of exception has occurred according to a stored vector table in the memory. The interrupt stands for the functionality available, the user can enable/disable, clear/set event interrupts. Following a block diagram of the NVIC controller connected to an ARM Cortex-M Core handling CPU & system interrupts. The NVIC is also connected to Un-core modules handling external & user interrupts.

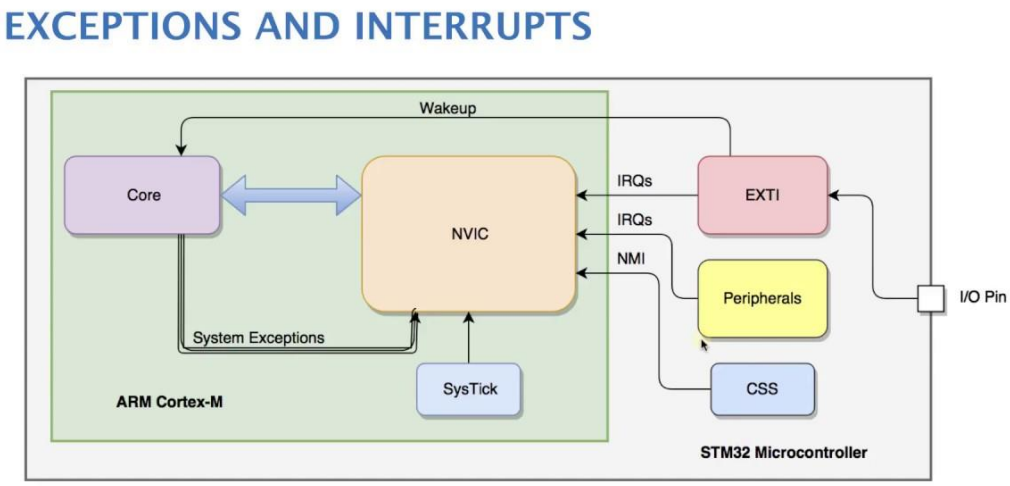


Figure 124: NVIC Environment

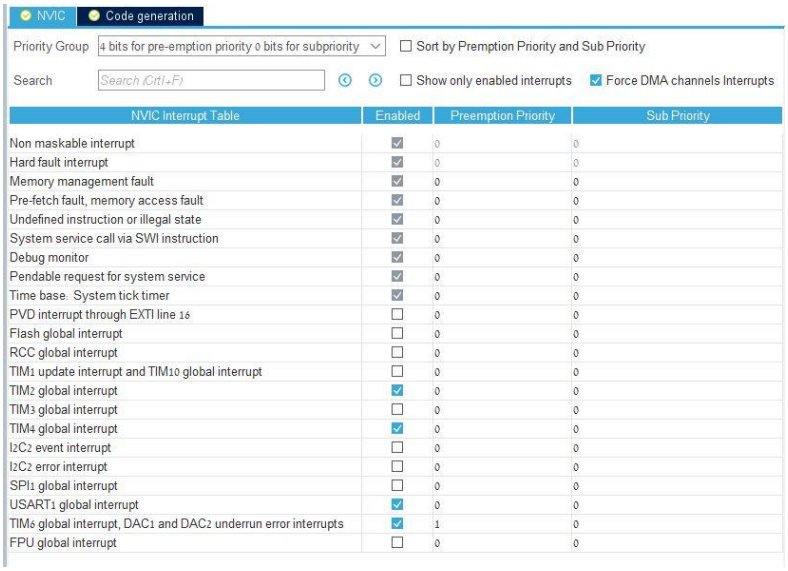


Figure 25: NVIC configuration GUI

5.7. Software Architecture

The software running on this robot is a bare metal real-time code executing on a CPU managing HW engines, no operating system is used to orchestrate the resources on this microcontroller. the programmer has full control over any resources available and any application like PWM, I2C, SPI needs to be handled by the software. Here you will find the Architecture behind the software handling of all the applications working flawlessly together.

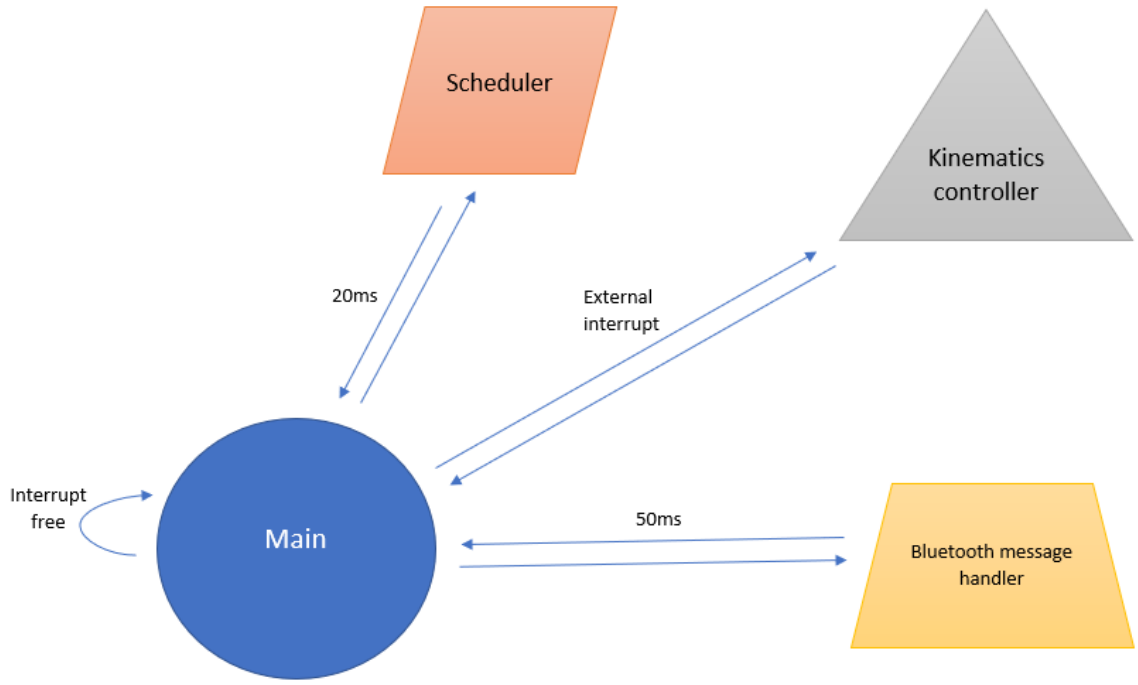


Figure 26: software timing diagram

5.7.1. Main

Believe it or not, main function is empty. this is due to timing demands. every handler is called upon an exact HW timer event usually when a timer has expired, or an external interrupt has occurred, once all system calls/interrupts are fulfilled the CPU is back to main waiting for interrupts calls.

5.7.2. Scheduler

The scheduler is called every 20ms. in the scheduler many of the sensor data is collected and managed accordingly, the proximity sensors get triggered to initiate a proximity measurement, the SPI is burst into collecting camera data & many other processes are initiated and handled.

5.7.3. Kinematics controller

The Kinematics controller is called every 0.5ms. and it handles the software drivetrain controller explained in the following section .6.8 this handler is called very frequently because of the high PWM frequency needed for the CLPD motor driver to understand the signal.

5.7.4. Bluetooth message handler

This handler is called upon external interrupt, anytime the user is pressing on the joystick in his application the Bluetooth module initiates a UART interrupt at every byte or data received, each packet of data is built from 8 bytes, the first three are angle as we need to present a number from 0 to 360 in characters the forth byte is “a” indicating angle data is done, the next 2 bytes are speed data again presenting a number from 0 to 99 in character, and the last byte is “s” indicating the end of the transmission, using an 8 byte buffer and easy string manipulations, the angle and speed that were transmitted from the mobile device are converted into an int numbers that will get processed by the kinematics controller moving the robot as desired.

5.7.5. Practical implementation

Following is an Oscilloscope measurement conducted to show the harmony of the three handlers presented above on the same time scale. Every toggle represents an interrupt call where the CPU jumped from main to the desired handler. The yellow presents scheduler calls, blue presents Bluetooth calls and Green is for the Kinematics controller, notice! In the lower part of the figure below we can see the width of every handler indicating the time between every call supporting the declarations provided above in the block diagram, for Scheduler it is around 20ms, kinematics controller around 0.5ms.

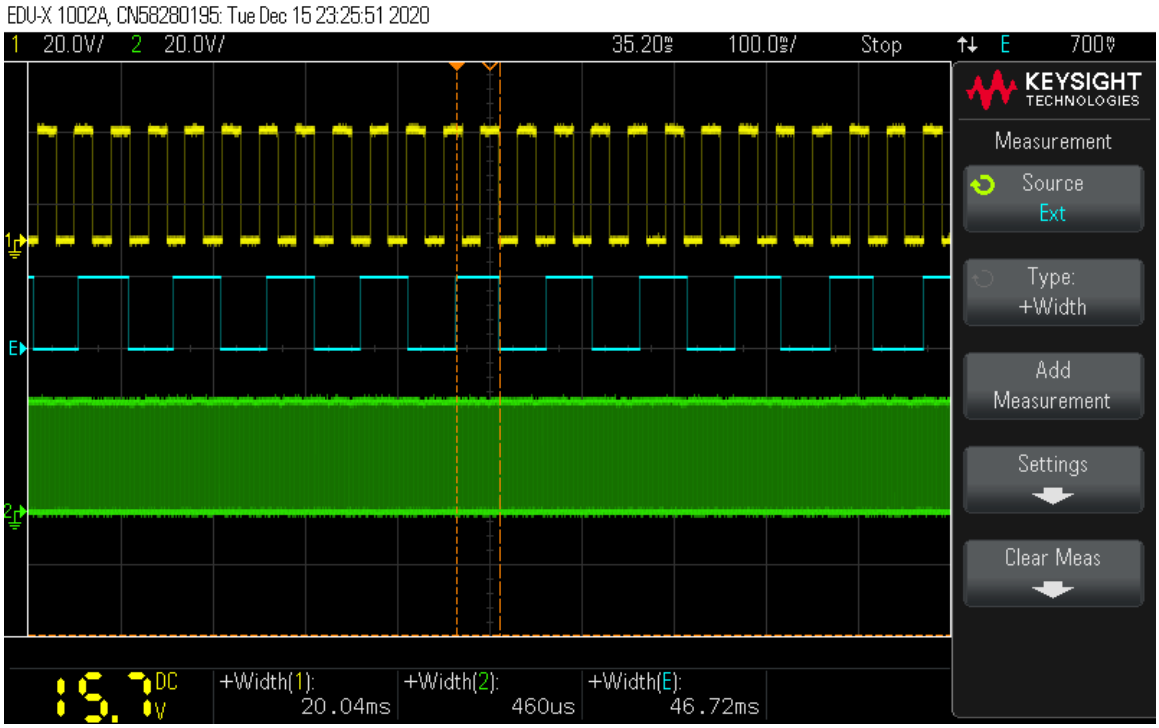


Figure 137: Oscilloscope interrupt measurement

5.8. Software Drivetrain controller

in this section you will find a brief explanation of the drivetrain controller, & how the robot moves.

The software drivetrain controller is feed with an angle and a magnitude out of our control application like shown below, internally this controller is built of 2 elements, a gaussian transform, & an equation-based drivetrain controller feeding the HW motor drivers moving the wheels as intended, in this section we will deep dive into the implementation of both elements.

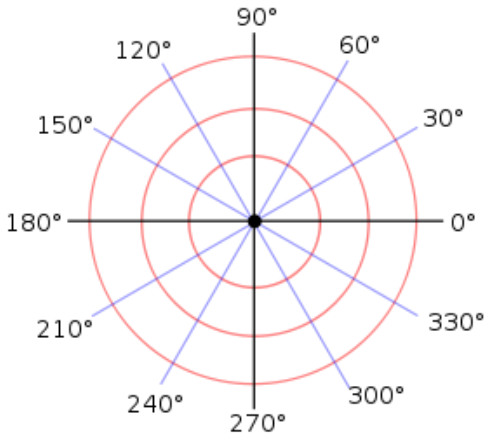


Figure 148: Polar axis, system input space, magnitude, θ

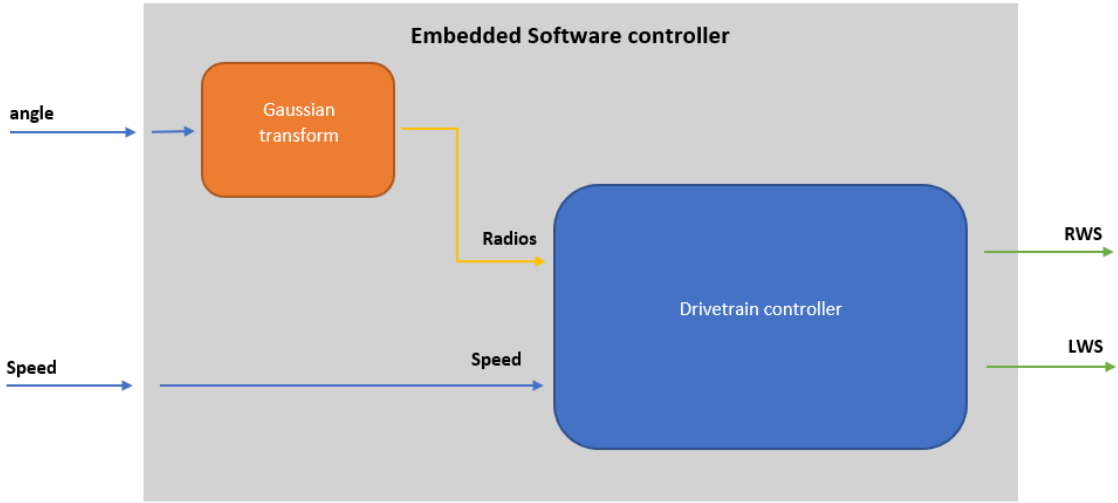


Figure 29: software controller block diagram

5.8.1. Drivetrain controller

Firstly, let us look at a differential drive system and define a few variables, constants, and rules. In a differential drive system, the robot has a huge freedom of movement, one of which the robot can move forward & backward in any horizontal direction by rotating along its center axis, the second however is an arc-based movement like shown below where the robot has a rotational speed Omega [Rad/sec] and R [m] (radios of arc). and so, to control such robot we need 2 input variables, in our case we will be working with robot speed which can be calculated from Omega and R, and the radios of arc (R).

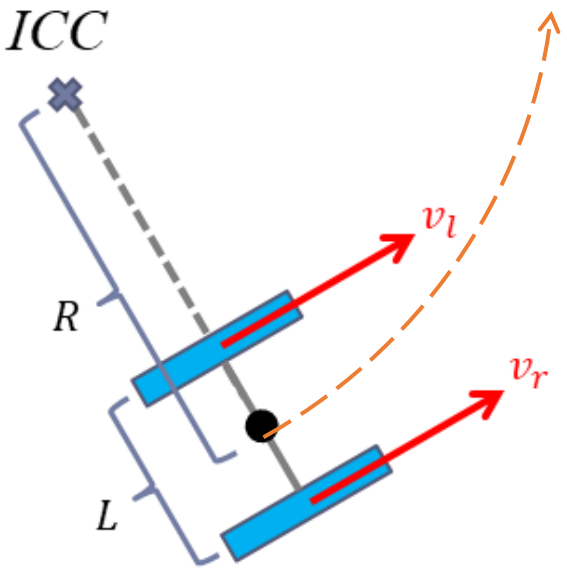


Figure 3015: Differential drive parameters & constants

The drivetrain controller is a MIMO (Multiple input multiple output) system controlling a differential drive mechanism built of 2 front motorized wheels, & a rear wheel. following a block diagram of this controller.

Figure 31: Drivetrain controller IO

5.8.2. Drivetrain controller Equations

Firstly, let us define our I/O, robot parameters, & dummy variables that will be used consistently from this point on.

Constants:

1. L: $The\ distance\ between\ the\ front\ wheels.$

I/O's:

1. Input, S: $robot\ velocity.$
2. Input, R: $radius\ of\ arc.$
3. Output, RWS: $Right\ wheel\ velocity.$
4. Output, LWS: $Left\ wheel\ velocity.$

Dummy Variables:

1. ω: $angular\ velocity\ in\ rad\ per\ second.$

Given a state where we know the inputs, S, R our objective is to calculate the velocity that needs to be feed into the right & left wheels to drive in the desired arc & in the specified velocity.

We can first calculate our angular velocity ω derived by dividing the velocity S with the radius R or arc like shown below.

$$\omega = \frac{S}{R}$$

Equation 1

By Knowing the Angular velocity ω we can calculate both RWS, LWS like follows:

$$RWS = \omega\left(R + \frac{L}{2}\right), \qquad LWS = \omega\left(R - \frac{L}{2}\right)$$

Equation 2

Now let's plug in the dummy variable to transform the equation back into our I/O variables:

$$RWS = S\left(1 + \frac{L}{2R}\right), \qquad LWS = S\left(1 - \frac{L}{2R}\right)$$

Equation 3

Our Outputs RWS & LWS are dependent on S & R, and by controlling them we control the behavior & movement of the robot. The input S does not have much effect on the movement in terms of control, however, the real challenge here is to control the radius of arc R to match and fulfill a desired and satisfying movement. In the following section we will discuss a unique mapping transformation (The Gaussian transform) that controls R in a way featuring smooth movement, accurate positioning, and a wide range of motion.

5.8.3. Gaussian transform

This transform maps the input angle values received from the control application to a radius of arc values feed into the drivetrain controller, see figure x above. The name gaussian transform is derived upon its functionality, it is an input output transformation implemented by summing 4 gaussian distribution functions. The motive behind this approach is inspired by the Fourier series, in such way that by summing an infinite number of Sine/Cosine functions any continuous curve, function, or drawings can be constructed. However, using a gaussian distribution function instead of sine/cosine function seemed better suiting this project's control needs & compute power, as the gaussian series must be finite and as little as possible. Following block diagram of this transform system constructed form 4 gaussian functions.

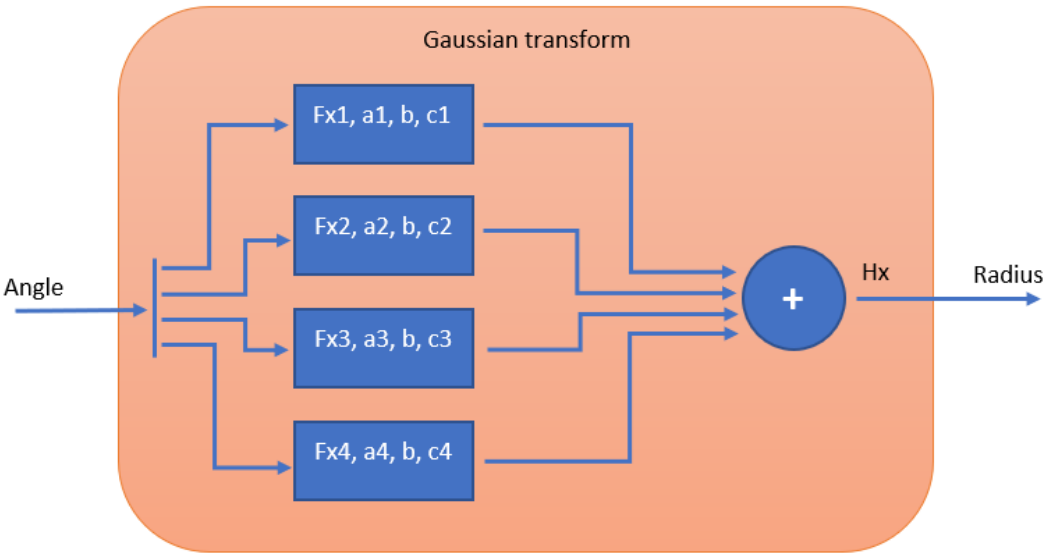


Figure 32: Gaussian transform Under the hood

Following the gaussian function with respect to angle, used in each function block (Fx1, 2, 3, 4).

$$Fx_i(angle) = a_i * e^{-\frac{(angle-b)^2}{2c_i^2}}$$

Equation 3: Gaussian distribution function.

The numbers presented in the table below were calibrated using a MATLAB script and multiple field tests with the robot emphasizing on smooth motion and precise all directional control.

Function	*a = amplitude*	*b = expectance*	*c =sqrt(variance)*
Fx1	a1 = 3000	b = 90	c1 = 4
Fx2	a2 = 100	b = 90	c2 = 20
Fx3	a3 = 30	b = 90	c3 = 40
Fx4	a4 = 32	b = 90	c4 = 1000

Table 3: transformation calibration constants.

5.8.4. Motion Analysis

Following you can see 5 functions overlapping on the same plot, 4 gaussian functions Fxi, and their sum Hx (the gaussian series).

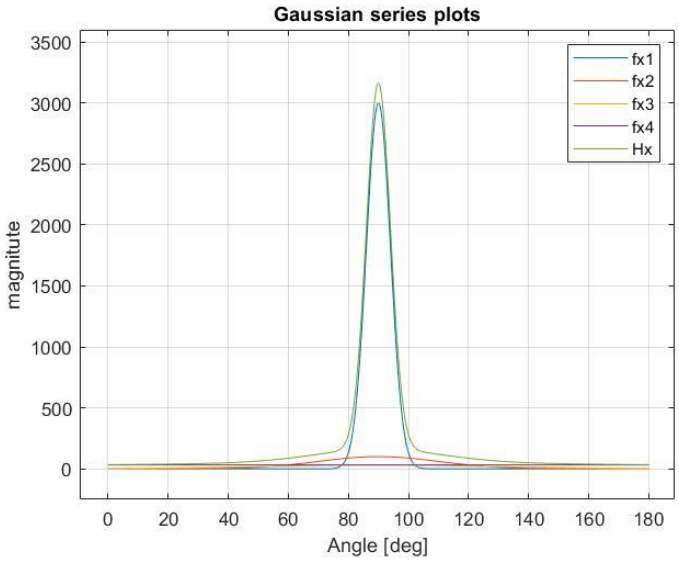


Figure 3316: Gaussian series plot.

notice the interesting behavior when the angle θ is in the region $80° < \theta < 100°$, the desired radius (Hx) is relatively infinite with respect to other regions on the angle θ axis, this behavior is very much intended as when the user needs to drive in a straight line we must eliminate the robot from aggressively reacting to small noise and fluctuations around $\theta = 90° \pm bias$ coming from the control station or his unstable finger on the joystick, and by setting a region around $\theta = 90°$ with a radius which is relatively large, we can insure that the curvature in motion is very much negligible and the robot will approximately drive completely straight forward when the user is pointing in this region $85° < \theta < 95°$.

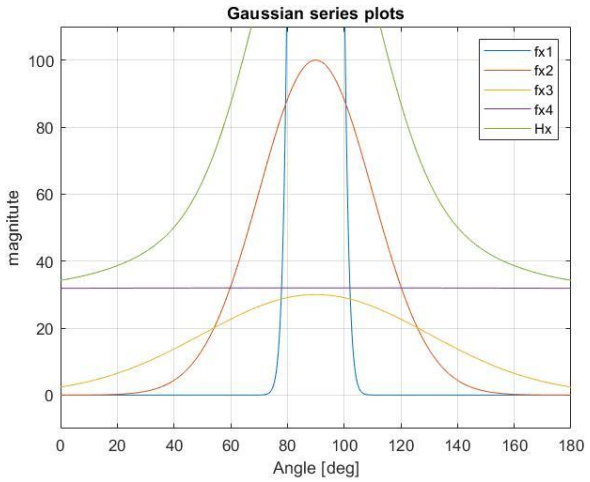


Figure 34: Gaussian series plot zoom in.

If we take a look at the differential velocity plot below (RWS-LWS), we can observe approximately 0 differental speed in the region $85° < \theta < 95°$ meening both wheels are moving in the same speed, thus driving forward.

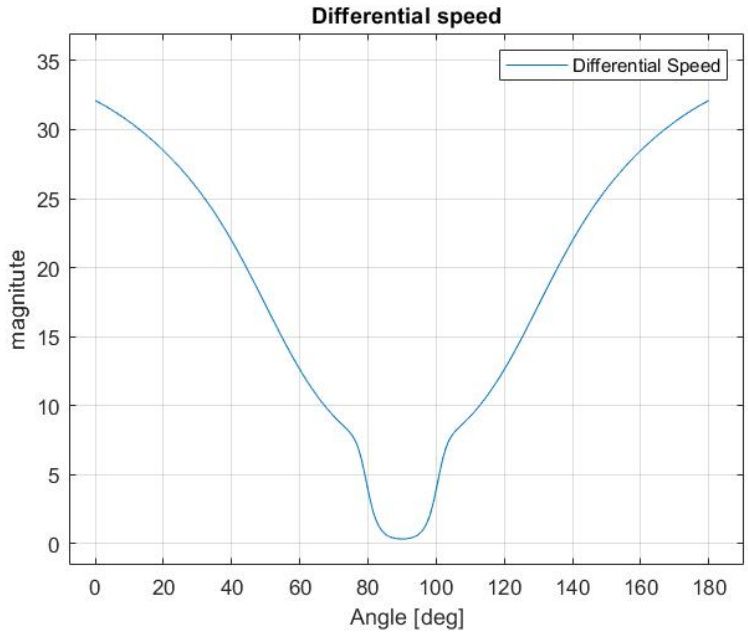


Figure 35: Differential speed plot

Following both wheels speed visualized, notice same speed in the region $85° < \theta < 95°$.

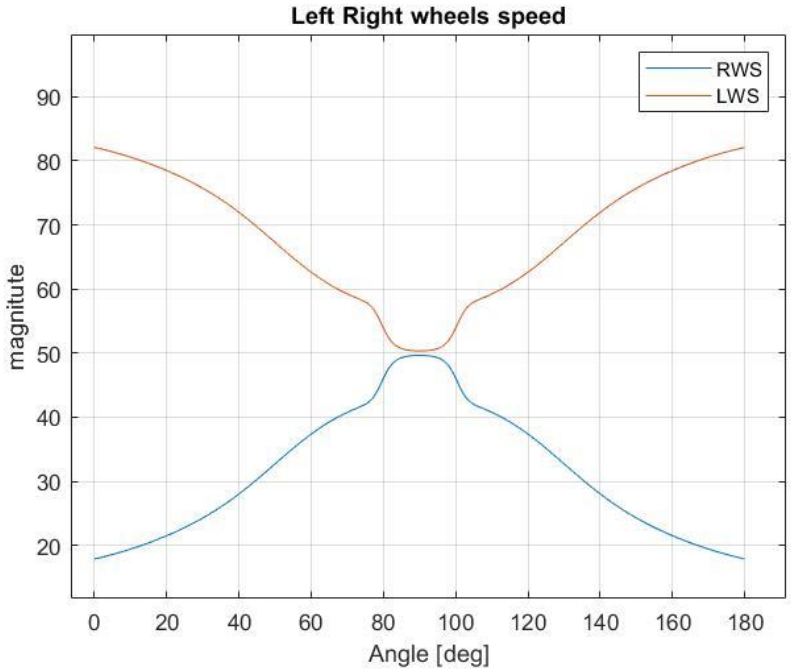


Figure 36: Wheels Speed plot

Observing the figures above will leave us wondering, what is the purpose of this continuous jump in the regions $80° \cong \theta \ \ or \ \ \theta \cong 100°$? to answer this, one must conduct a few different tests to observe the behavior of the robot, and after a few iterations the conclusion is that there must be a noticeable differential speed between the wheels to start feeling the arc movement of the robot. recalling the theory & motive behind the region $85° < \theta < 95°$ where the radius of arc was intended to be infinitely large, however, this jump in differential speed is caused by the rapid degradation in the radios of arc as outside this region $85° < \theta < 95°$ we need the robot to react and change direction accordingly so the radius of arc jumps back down to a realistic value which is finite with respect to the robot length (L) see Hx in green.

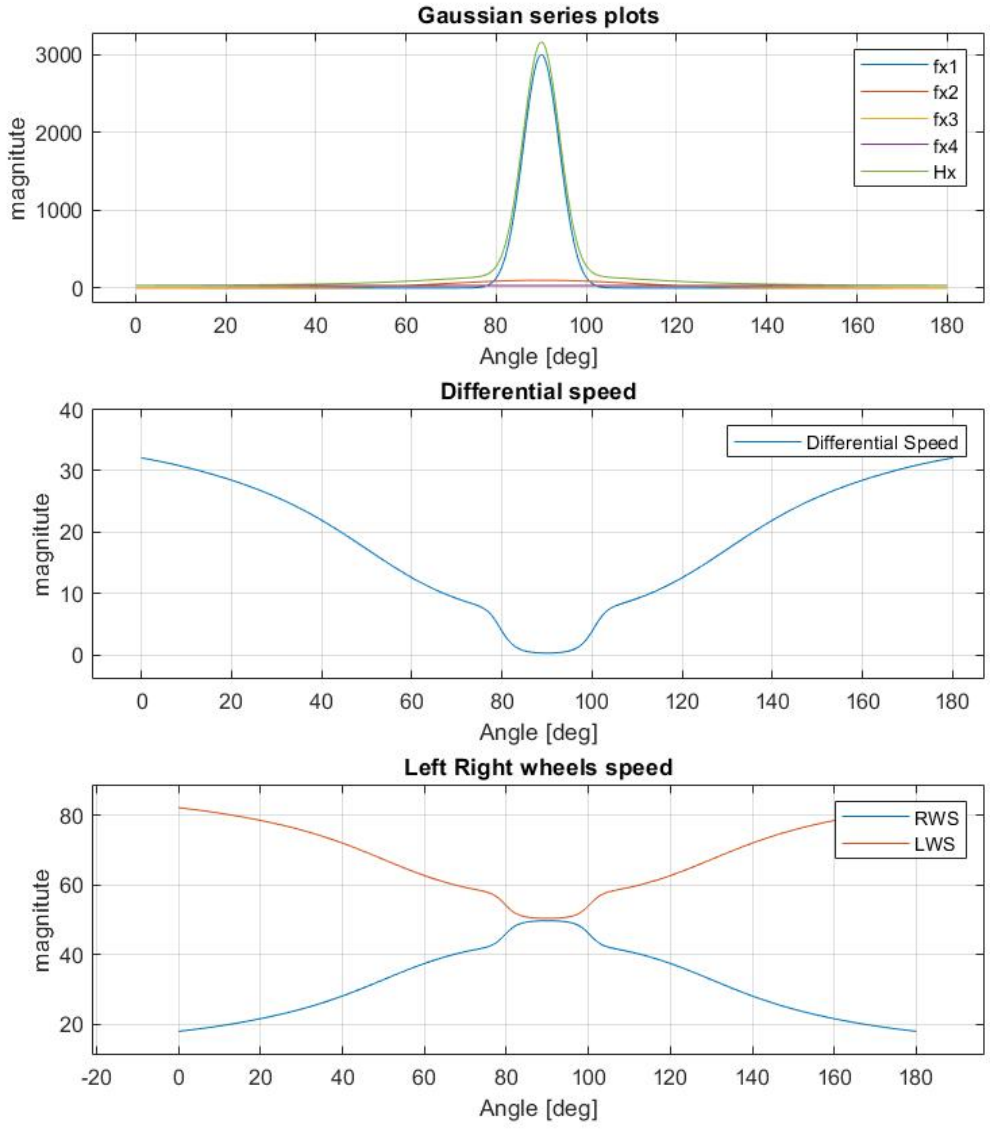


Figure 37: combined plots

In the regions $0° < \theta < 80°$ & $100° < \theta < 180°$ the objective was to make a linear differential change, and indeed the plot seems to relatively obey a linear change, however, to achieve a perfect linear behavior, more Gaussian functions must be summed into the series increasing compute demands, power, latency, & and memory usage. such addition is considered unnecessary as the robot moves quite smooth and responsively, anyhow, adding more function would not add much to the control and smoothness, but add complexity preserving any other behavior achieved so far as all constants need to be recalibrated.

5.8.5. Compute power & resources

Any software or code running on a CPU consumes compute power, here you will find a short resource and latency analysis of the software drivetrain controller.

As discussed, prior in this section the controller inputs a speed and an angle and outputs a right wheel speed and a left wheel speed, the angle is transformed into a radius via the Gaussian transform logic, this transformation is very compute heavy as is must compute 4 exponential functions in floating point and add them together. this operation in time consuming and adds a lot of latency to the controller if computed on the fly at every controller iteration and for every angle change. Taking this approach consumes very little memory as we hold only one floating point variable, but on the other hand it is time consuming, many complex calculations have to be done, meanwhile the CPU is loaded, and the system is halted and can't handle other applications and exceptions. after some-time this will cause a huge delay in the response of the robot and may overflow the NVIC controller causing the system to restart the NVIC and dump important system calls, exceptions, & interrupts.

$$Approach1 = \{\, Latency: very\ High \quad memmory\ usage: low \,\}$$

A different approach makes latency negligible but memory heavy, once the controller is out of reset and starts running, we calculate the output value of the transformation for every possible input once and save the output in the system memory for later use. When the kinematics software controller is called, no on the fly calculations are needed but one memory call is needed to bring the desired pre calculated value.

$$Approach2 = \{\, Latency: very\ low \quad memmory\ usage: high \,\}$$

6. References

To continue with this project download and follow the steps below. All custom hardware components are listed in the first few sections, the development board is a black-board featuring an STM32F407VET6 microcontroller with an Arm-Cortex-M4 processor. to program the robot download STM32CubeIDE and to configure the microcontroller download STM32CubeMx. you will also need a debugger to physically program and debug the microcontroller, the Segger-jlink-base is recommended.

To get started follow these steps:

1. Configure the controller IO using STM32CubeMx.
2. Export generated code into the STM32CubeIDE.
3. Connect the debugger in SWD mode (see STM32F407 datasheet).
4. Write some C code (a bunch of it actually).
5. Program the code.
6. Debug system architectural behavior.
7. Debug your Code.
8. Iterate Over and over from bullet 4.

7. Index

Most of the work was done using datasheets downloaded from the internet, following a list of datasheets used along with all the attached links in previous sections.

Sensors:

https://datasheetspdf.com/pdf/1380136/ETC/HC-SR04/1

https://docs.pixycam.com/wiki/doku.php?id=wiki:v2:porting_guide

http://ecee.colorado.edu/ecen3000/labs/lab6/MotionSensing.pdf

https://www.amazon.ca/KKmoon-12V-36V-Brushless-Controller-Balanced/dp/B01LQ80K9K

Differential Drive:

http://www.cs.columbia.edu/~allen/F17/NOTES/icckinematics.pdf

https://www.sciencedirect.com/science/article/pii/S2212017312002460

https://en.wikipedia.org/wiki/Gaussian_function