WAREHOUSE AUTOMATION THROUGH MOBILE ROBOTICS

LOW LEVEL CONTROL DOCUMENTATION

BY ROBO RAIDERS TEAM
FACULTY OF ENGINEERING
DAMIETTA UNIVERSITY
ELECTRICAL DEPARTMENT

Table of Contents

Table of Figures

1. Low level control system overview

The core of the proposed system is the STM32F401RCT6 microcontroller, which serves as the central processing unit responsible for coordinating the movement and navigation of the mobile robot. Through precise control of its embedded peripherals and interfaces, such as General-Purpose Input/Output (GPIO), Timers, Interrupts, Reset and Clock Control (RCC), and Universal Asynchronous Receiver-Transmitter (UART), the microcontroller orchestrates the functionalities required for seamless operation of the mobile robot. The mobile robot's mobility is achieved through the manipulation of four DC motors, each controlled independently to enable omnidirectional movement. High-level control commands, originating from a Raspberry Pi 4 unit, are transmitted to the STM32 microcontroller via UART communication protocol. These commands dictate the desired trajectory and velocity of the robot, guiding it through the warehouse environment.

• **MCAL Drivers:**

1. GPIO (General Purpose Input/Output):
   GPIO driver enables the user to configure the pins of the stm32 to any desired function as it can work as (Digital input/output pin – external interrupt pin – Timer pin - ….etc.).
2. RCC (Reset and Clock Control):
   RCC driver helps the user to configure the desired clock frequency of the microcontroller and enables/disables the clock of any peripheral.
3. Interrupt:
   Interrupt driver provides a handful management of ISRs.
4. Timer:
   Timer driver provides a lot of options for use such as: generating PWM signal with any desired frequency and duty cycle, generating interrupt after specific time (used for delays) and encoder mode.
5. USART (Universal Synchronous Asynchronous Receiver/Transmitter):
   USART driver enables the communication between Low level control and high-level control systems. We are depending on HAL_USART.

• **HAL Drivers:**

1. mec_move:
   mec_move is the driver responsible of controlling DC motors signals, there are ten different movements implemented in this driver.
   It is combined with the encoder and PID drivers so we can achieve the desired position.
2. Stepper:
   Stepper driver contains some functions that control the stepper motor direction and speed built in PWM signals generated by timers.

• **Helper Drivers:**

1. PID:
   PID driver contains a program that helps us to achieve the desired position with a reasonable accuracy, this is done through getting encoder pulses and compare it with the desired pulses (distance)

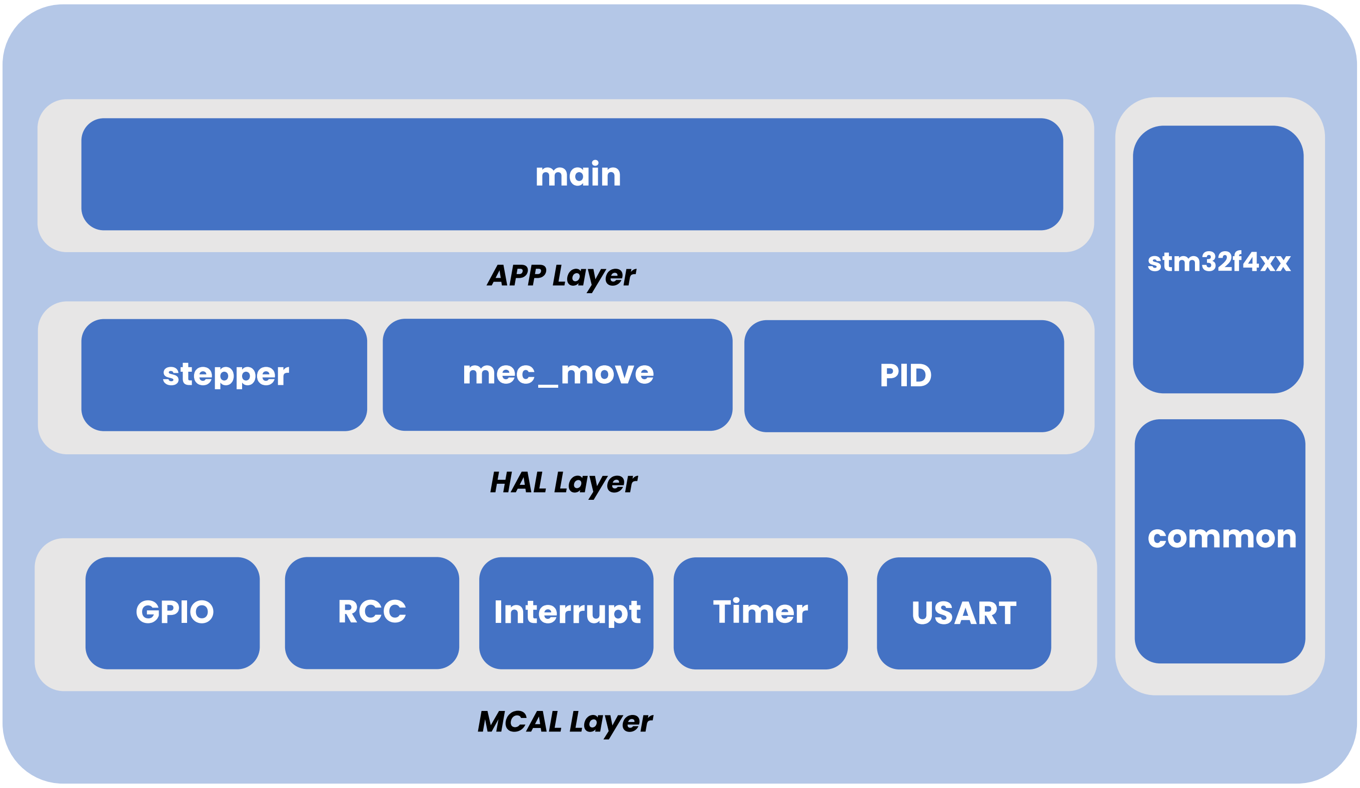

Figure 0: ECU layer architecture

2. System architecture design

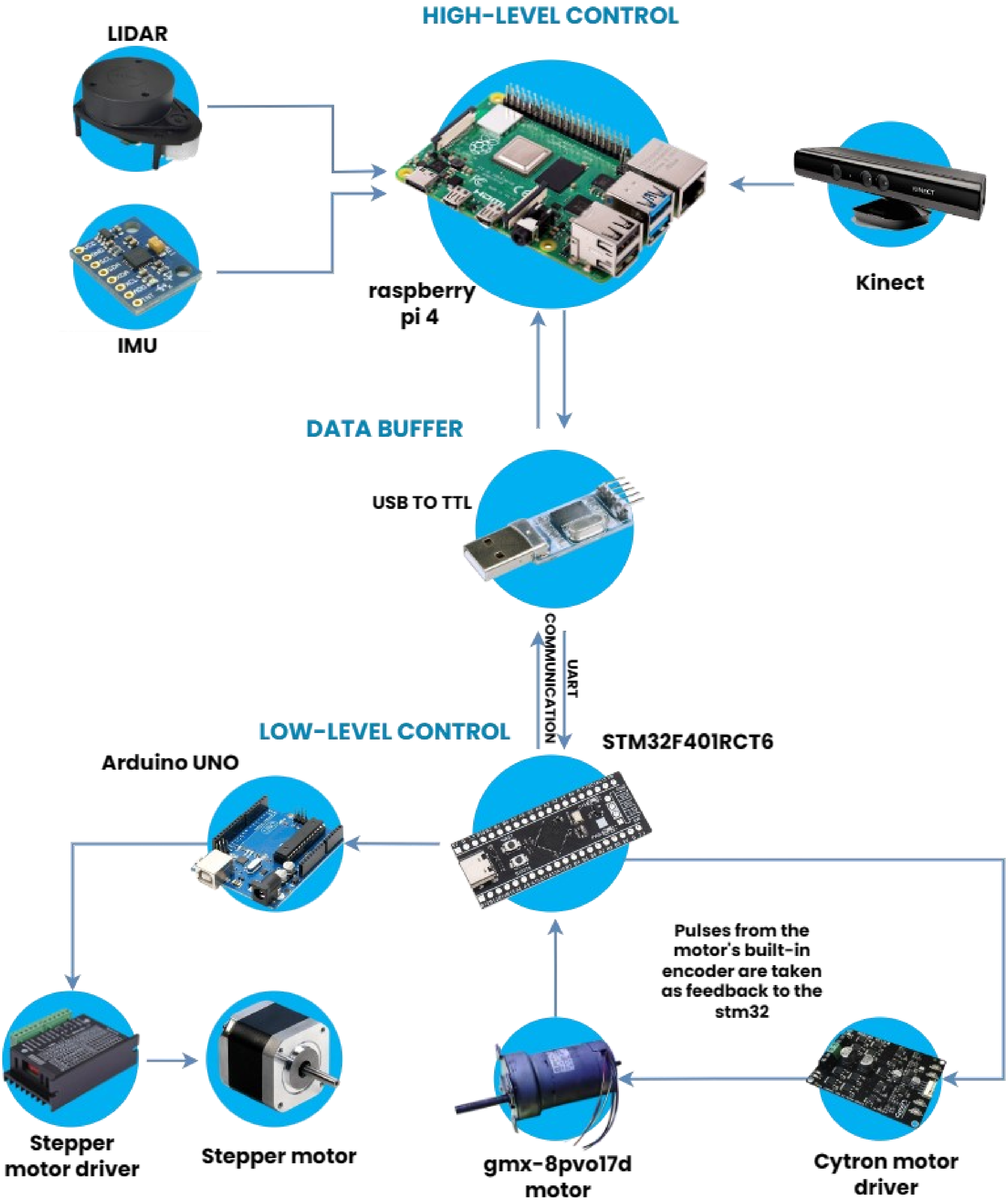

Figure 1: Overall control System architecture design.

- **Why did we choose stm32f401rct6 (black pill board)?**

  - ***Cost-Effectiveness:*** The STM32F401RCT6 offers a balance between performance and cost, making it an affordable option for embedded projects, especially for those on a budget.
  - ***High Performance:*** With a Cortex-M4 core running at up to 84 MHz and a rich set of peripherals, the STM32F401RCT6 provides sufficient processing power and capabilities for controlling multiple DC motors and handling real-time tasks.
  - ***Low Power Consumption:*** The microcontroller is designed for low power operation, which is crucial for battery-powered or energy-efficient applications like mobile robotics (as in our project).
  - ***Peripherals:*** It features a wide range of peripherals including GPIO, timers, UART, ADC, SPI, I2C, and more, providing flexibility for interfacing with various sensors, actuators, and communication modules required for warehouse automation.
  - ***Memory Capacity:*** With 512 KB Flash memory and 96 KB SRAM, the STM32F401RCT6 offers sufficient memory for storing program code, data, and configurations, allowing for the implementation of complex control algorithms and data processing tasks.
  - ***Availability of Development Tools:*** The STM32 microcontroller family is well-supported by a variety of development tools, including integrated development environments (IDEs), compilers, debuggers, and libraries, which streamline the development process and facilitate debugging and testing.
  - ***Community Support:*** The STM32F401RCT6 is popular among hobbyists, students, and professionals, resulting in a vibrant community that provides ample resources, tutorials, forums, and open-source projects, which can aid in project development and troubleshooting.
  - ***Scalability and Compatibility:*** The STM32F401RCT6 is part of the STM32 family, offering scalability to higher or lower-end devices

within the same family, allowing for easy migration or expansion of the project in the future. Additionally, it is compatible with a wide range of development boards, shields, and modules, enhancing its versatility and ease of integration.

- ***Reliability and Robustness***: STMicroelectronics, the manufacturer of STM32 microcontrollers, is known for producing high-quality and reliable components, ensuring the durability and longevity of the embedded system in harsh operating environments typically found in warehouse settings.

• STM32f401rct6 (black pill) specifications:

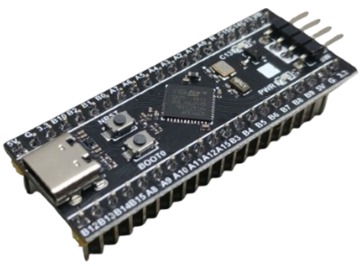

Figure 2: Black pill Perspective view.

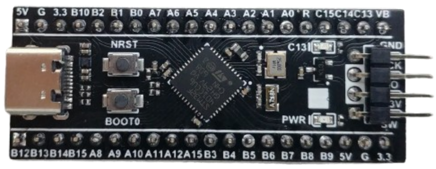

Figure 4: Black pill top view.

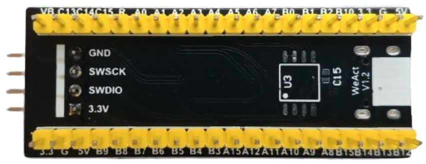

Figure 3: Black pill bottom view.

• Microcontroller:

Part	STM32F401RCT6
Manufacturer	ST-Microelectronics
Core	Arm Cortex-M4
Max. Clock Speed	84MHz
Package	UFQFPN 48 pins

• Internal memories

FLASH	256KiB
SRAM	64KiB

• Oscillators

HSI	16MHz
LSI	32kHz
HSE	25MHz
LSE	32.768kHz

3. MCAL Drivers

1-GPIO driver:

GPIO driver is responsible of configuring the pins, pins can be set as inputs, outputs, timer pins, UART pins, external interrupt pins, …etc., also it can configure the speed of the output pin, type of the output pin and it can add pull-up/pull-down resistor to the input pin.

This section will contain the user-defined data types in the driver and the following functions in details:

- ***GPIO_Init().***
- ***GPIO_Write_Pin().***
- ***GPIO_Toggle_Pin().***
- ***GPIO_Read_Pin().***
- ***GPIO_Set_EXTI().***

• Detailed description of the user-defined data types used:

Name	Pin_cfg	
Type	structure	
Elements	GPIO	A pointer to GPIO_TypeDef so that we can access the desired port.
	pin_no	A *uint32_t* variable that specifies the pin number in the port.
	mode	A *unit32_t* variable that specifies the pin mode (output-input-analog-alternate).
	outType	A *uint32_t* variable that Specifies the output pin type (Push Pull- Open Drain).
	outputSpeed	A *uint32_t* variable that Specifies the output pin speed (Low-Medium-High-Very High).
	pull	A *uint32_t* variable that Specifies the output pin speed (Low-Medium-High-Very High).
	altFunc	A *uint32_t* variable that Specifies the alternate function needed (0 -> 15).
Description	It is the backbone of the driver as it contains the desired configurations for the pin, and it is passed to almost all the functions.	

Name	Logic_t	
Type	enum	
Elements	GPIO_LOW	0
	GPIO_HIGH	1
Description	A typedef that contains the pin logic(state.)	

Name	GPIO_mode	
Type	enum	
Elements	Input	0
	Output	1
	AltFunction	2
	Analog	3
Description	A typedef that contains the pin modes.	

Name	OUT_type	
Type	Enum	
Elements	PushPull	0
	OpenDrain	1
Description	A typedef that contains the output pin type.	

Name	OUT_speed	
Type	Enum	
Elements	LowSpeed	0
	MedSpeed	1
	HighSpeed	2
	VeryHighSpeed	3
Description	A typedef that contains the output pin speed.	

Name	PULL_type	
Type	Enum	
Elements	None	0
	PullUp	1
	PullDown	2
Description	A typedef that contains the input pullup/pulldown resistor.	

Name	ALT_num	
Type	Enum	
Elements	AF0	0
	. . . .	1 . . 13
	AF15	14
Description	A typedef that contains the alternate function numbers.	

Name	edge_detect	
Type	Enum	
Elements	RISING	0
	FALLING	1
	RISING_FALLING	2
Description	A typedef that contains the type of edge detection for external interrupts.	

• ***Detailed description of the functions used:***

Name	GPIO_init	
Type	Function	
Arguments	Pin_cfg *pin	A pointer to Pin_cfg
Return	Std_ReturnStatus	E_NOK
		E_OK
Description	initialization of GPIO configurations according to elements of the pointer to structure Pin_cfg.	

Name	GPIO_Write_Pin	
Type	Function	
Arguments	GPIO_Typedef *Gpio	A pointer to GPIO_Typedef to specifiy the port
	uint16_t Pin	Specifies the pin number
	Logic_t logic	Specifies the pin output state.
Return	Std_ReturnStatus	E_NOK
		E_OK
Description	Sets (1) or Resets (0) the output pin.	

Name	GPIO_Toggle_Pin	
Type	Function	
Arguments	GPIO_Typedef *Gpio	A pointer to GPIO_Typedef to specifiy the port.
	unit16_t	pin number.
Return	Std_ReturnStatus	E_NOK
		E_OK
Description	Toggles the pin output (0 -> 1), (1 -> 0).	

Name	GPIO_Read_Pin	
Type	Function	
Arguments	GPIO_Typedef *Gpio	A pointer to GPIO_Typedef to specifiy the port.
	unit16_t pin	pin number.
Return	Logic_t	GPIO_HIGH
		GPIO_LOW
Description	Read the pin status (Input pin).	

Name	GPIO_Set_EXTI	
Type	Function	
Arguments	GPIO_Typedef *Gpio	A pointer to GPIO_Typedef to specifiy the port.
	unit16_t pin	pin number.
	edge_detect edge	Specifies the desired edge detection.
Return	Std_ReturnStatus	E_NOK
		E_OK
Description	Sets the required pin as an external interrupt source.	

2-RCC driver:

RCC driver is responsible of configuring the clock of the microcontroller, it gives us the flexibility to obtain any desired clock frequency in the limits of the microcontroller buses.

This section will contain the user-defined data types in the driver and the following functions in details:

- ***OSC_CFG_init().***
- ***CLK_CFG_init().***
- ***PLL_CFG_init().***
- ***HSE25Mhz_84MHzOUT().***
- ***RESET_DEFAULT_CLCK().***

• Detailed description of the user-defined data types used:

Name	osc_type	
Type	Enum	
Elements	RCC_OSC_HSI	0
	RCC_OSC_HSE	1
	RCC_OSC_LSE	2
	RCC_OSC_LSI	3
Description	A typedef that contains the types of oscillators included.	

Name	CLKSRC_type	
Type	Enum	
Elements	CLKSRC_HSI	0
	CLKSRC_HSE	1
	CLKSRC_PLL	2
Description	A typedef that contains the clock source of the microcontroller	

Name	OSC_CFG_type	
Type	structure	
Elements	OSC_type	Contains the oscillator type.
	HSI_state	Contains the HSI state, active or inactive.
	HSE_state	Contains the HSE state, active or inactive.
Description	A typedef that contains the clock source of the microcontroller	

Name	PLL_CFG_type	
Type	structure	
Elements	unint32_t PLLN	Specifies N parameter value.
	unint32_t PLLM	Specifies M parameter value.
	unint32_t PLLQ	Specifies Q parameter value.
	unint32_t PLLP	Specifies P parameter value.
	unint32_t PLL_state	Specifies whether is PLL ON or OFF.
	unint32_t PLL_SRC	Specifies the PLL clock source.
Description	A typedef that contains the PLL parameters and clock source.	

Name	CLK_CFG_type	
Type	structure	
Elements	AHB_prescaler	0
	APB1_prescaler	1
	APB2_prescaler	2
	SYSCLK_SRC	3
Description	A typedef that contains the types of oscillators included.	

• ***Detailed description of the functions used:***

Name	OSC_CFG_init	
Type	Function	
Arguments	OSC_CFG_type* OSC	A pointer to OSC_CFG_type
Return	Std_ReturnStatus	E_NOK
		E_OK
Description	initialization of oscillators configurations.	

Name	CLK_CFG_init	
Type	Function	
Arguments	CLK_CFG_type* CLK	A pointer to CLK_CFG_type
Return	Std_ReturnStatus	E_NOK
		E_OK
Description	initialization of System clock configurations.	

Name	PLL_CFG_init	
Type	Function	
Arguments	PLL_CFG_type* PLL	A pointer to PLL_CFG_type
Return	Std_ReturnStatus	E_NOK
		E_OK
Description	initialization of PLL configurations.	

Name	HSE25Mhz_84MHzOUT
Type	Function
Arguments	void
Return	void
Description	generate System clock = 84MHz using HSE = 25MHz and PLL.

Name	RESET_DEFAULT_CLCK
Type	Function
Arguments	void
Return	void
Description	generate default clock = 16MHz using HSI.

3-Timer driver:

Timer driver is responsible of generating controllable PWM signals with various frequencies and duty cycles, it is used to run the motor drive hence controlling the speed of the DC motors, it is also used to generate delay in the program or generating interrupt after a specific time.

This section will contain the user-defined data types in the driver and the following functions in details:

- ***PWM_GEN ().***
- ***INT_GEN_ms ().***
- ***tdelay_ms ().***
- ***timer_ext_clk ().***

• *Detailed description of the user-defined data types used:*

Name	channel_t	
Type	Enum	
Elements	CH1	0
	CH2	1
	CH3	2
	CH4	3
Description	A typedef that contains the channels' numbers.	

Name	PWM_config	
Type	structure	
Elements	uint32_t ch	Timer's channel number.
	uint32_t freq	PWM output frequency.
	uint32_t duty_cylce	PWM output duty cycle.
Description	A typedef that contains the clock source of the microcontroller	

• *Detailed description of the functions used:*

Name	PWM_GEN	
Type	Function	
Arguments	TIM_TypeDef *TIMER	A pointer to TIM_TypeDef.
	PWM_config* PWM_cfg	A pointer to PWM_config.
	uint32_t timer_clk	Timer's input clock frequency.
Return	Std_ReturnStatus	E_NOK
		E_OK
Description	Generate the required PWM according to the inputs.	

Name	INT_GEN_ms	
Type	Function	
Arguments	TIM_TypeDef *TIMER	A pointer to TIM_TypeDef.
	uint32_t t_ms	The required delay in millisecond.
	uint32_t timer_clk	Timer's input clock frequency.
Return	Std_ReturnStatus	E_NOK
		E_OK
Description	Generate interrupt after the required time.	

4-UART driver:

UART driver is responsible of the communication between the low level controller (stm32) and high level controller (raspberry pi 4), commands of movements and speed are sent through it from the high to low level.

This section will contain the user-defined data types in the driver and the following functions in details:

- ***USART_enable ().***
- ***USART_send_char ().***

• Detailed description of the functions used:

Name	USART_enable	
Type	Function	
Arguments	USART_TypeDef *USART	A pointer to USART_TypeDef.
	uint32_t baud_rate	Baud rate value.
	uint32_t sys_clk	System clock frequency.
Return	Std_ReturnStatus	E_NOK
		E_OK
Description	Enable the USART and configure the required pins according to the USART selected.	

Name	USART_send_char	
Type	Function	
Arguments	USART_TypeDef *USART	A pointer to USART_TypeDef.
	char data	The data needed to be sent as a character value.
Return	Std_ReturnStatus	E_NOK
		E_OK
Description	Send character value through USART protocol.	

4. HAL Drivers

1-mec_move driver:

mec_move driver is responsible of defining the movement of the robot, as it controls the dc motors speed and direction, with the right configuration we can achieve ten different movements and with the help of the pid driver we can reach the desired destination.

The functions are:

- *void forward(int target, int position).*
- *void backward(int target, int position).*
- *void rotate_left(int target, int position).*
- *void rotate_right(int target, int position).*
- *void side_right (int target, int position).*
- *void side_left (int target, int position).*
- *void diagonal_right_fw (int target, int position).*
- *void diagonal_right_bw (int target, int position).*
- *void diagonal_left_fw (int target, int position).*
- *void diagonal_left_bw (int target, int position).*

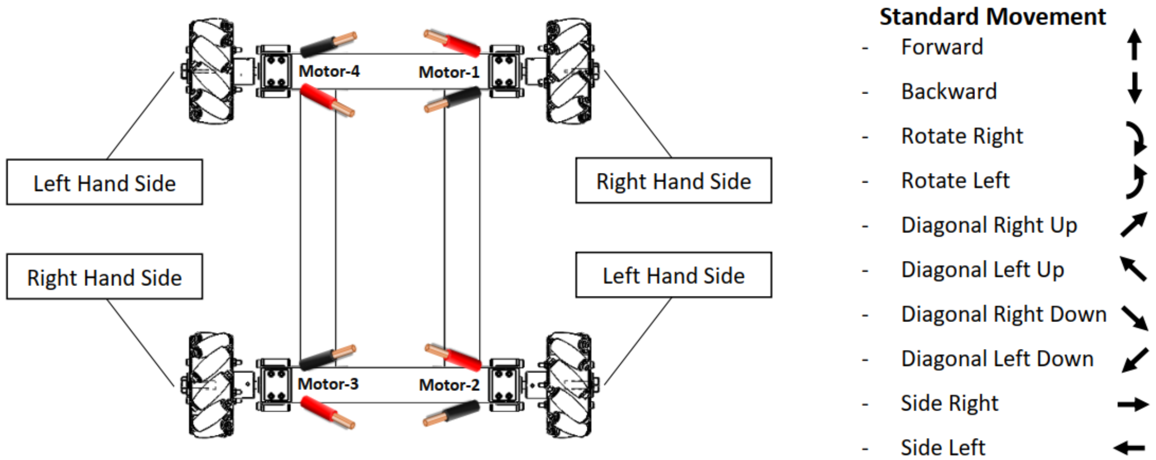

Figure 5: mecannum wheel installation.

	Motor-1		Motor-2		Motor-3		Motor-4	
	RED Wire	BLACK Wire	RED Wire	BLACK Wire	RED Wire	BLACK Wire	RED Wire	BLACK Wire
Forward	1	0	1	0	0	1	0	1
Backward	0	1	0	1	1	0	1	0
Rotate Right	0	1	0	1	0	1	0	1
Rotate Left	1	0	1	0	1	0	1	0
Diagonal Right Forward	0	0	1	0	0	0	0	1
Diagonal Left Forward	1	0	0	0	0	1	0	0
Diagonal Right Backward	0	1	0	0	1	0	0	0
Diagonal Left Backward	0	0	0	1	0	0	1	0
Side Right	0	1	1	0	1	0	0	1
Side Left	1	0	0	1	0	1	1	0

Figure 6: Control signals of each movement.

2-pid driver:

pid driver is responsible of cancelling overshooting and reaching the desired destination according to the input sent from the high-level control depending on PID formula and which contains the proportional, differential, and integral constants.

The functions are:

int pid_calc(int target, int actual).

```
int pid_calc(int target, int actual)
{
    err = target - actual;
    diff = err - prev_err;
    integral += err;
    result = kp*err + kd*diff + ki*integral ;
    if(result > 1000)
    {
        result = 1000;
    }
    else if(result < 0)
    {
        result = 0;
    }
    prev_err = err;
    return result;
}
```

Figure 7: Simple PID implementation