Student Project

Yin Chheanyun

August 2023

Contents

List of Figures

1 Introduction

The ABU Robocon 2024 contest hosted by Vietnam, has developed robot tasks that depict the stages of rice cultivation. These tasks include sowing, harvesting, and transporting the harvested grains to the warehouse.

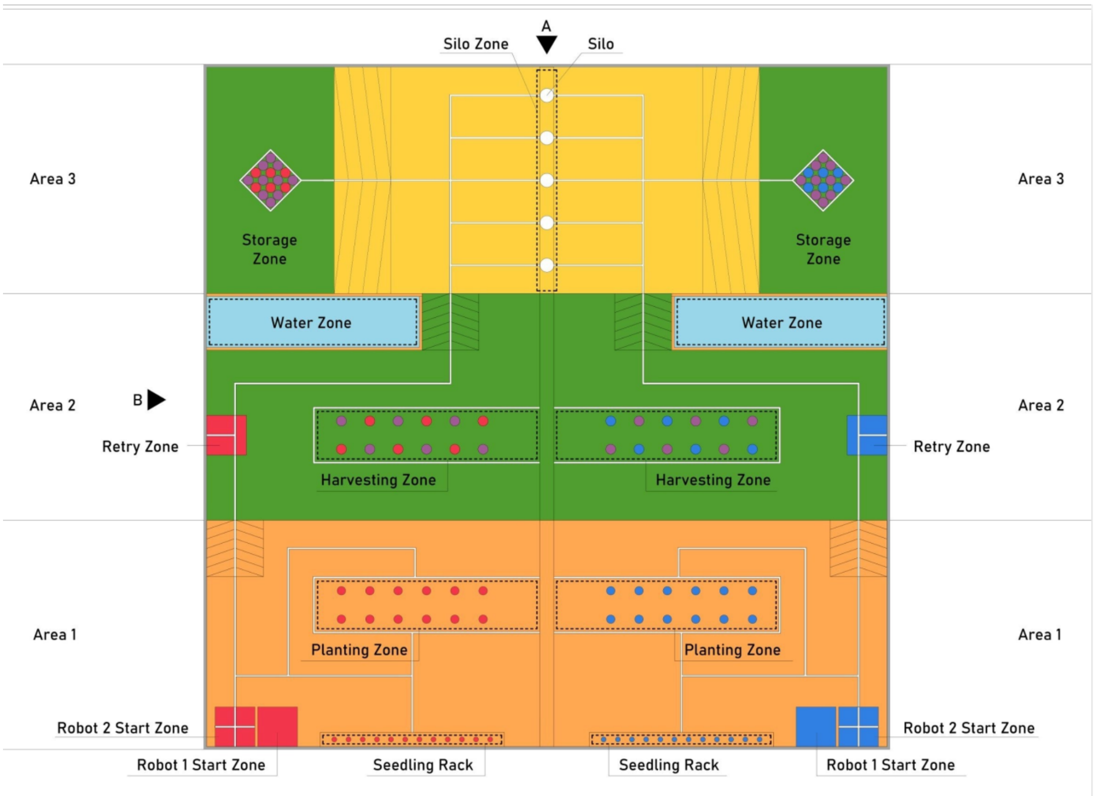

Figure 1: Game field of ROBOCON 2024

The team have to build 2 robots:

- Robot 1: Represented farmer.
- Robot 2: Represented buffalo (Autonomous Robot).

2 PI Control on speed DC_Motor

2.1 Find Modeling DC Motor

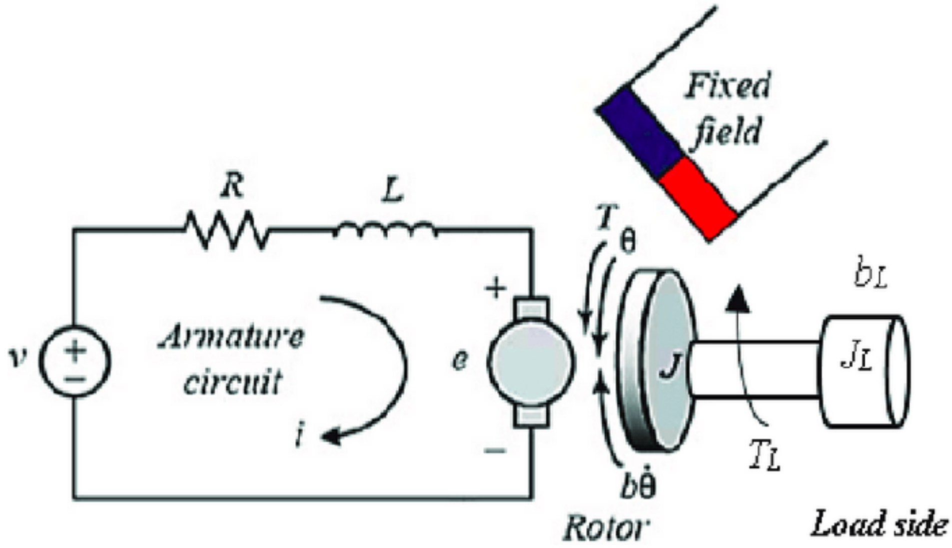

Figure 2: Modelig DC Motor

- **Electrical part:**

$$\boldsymbol{V = L_a \frac{di}{dt} + R_a i_a + K_b \omega}$$

- **Mechanical part:**

$$\boldsymbol{J \frac{d\dot{\theta}}{dt} + b\dot{\theta} + K_t i_a = T_L}$$

- **Transfer function:**

$$\boldsymbol{H(s) = \frac{\omega(s)}{V(s)} = \frac{K_t}{(Js+b)(Ls+R) - K_t K_b}}$$

Where:
_ **J:** moment of initial rotor ($Kg.m^2$)
_ **R:** resistance (Ω)
_ **L:** inductance (H)
_ **b:** motor viscous friction constant (N,m.s)
_ $\mathbf{T_L}$: Torque load (N.m)
_ $\boldsymbol{\omega = \dot{\theta}}$ **:** Speed (rd/s)
_ **Kb:** electromotive force constant (V/rad/sec)
_ **Kt:** motor torque (N.m/Amp)
Since, the time constant in electrical part $\tau = \frac{L}{R}$ too small compare mechanical time constant .
We can write transfer function of DC motor:

Speed: $\boldsymbol{H(s) = \frac{\omega(s)}{V(s)} = \frac{K}{\tau s+1}}$
Position: $\boldsymbol{H(s) = \frac{\omega(s)}{V(s)} = \frac{K}{(\tau s+1)s}}$

- **Method1 for find Modeling of DC Motor:**
  _ **Transfer function form:**

$$\boldsymbol{H(s) = \frac{\omega(s)}{V(s)} = \frac{K}{\tau s + 1}}$$

  _ **Step1:** you can Input Voltage (from 2V to Maximum voltage between 2 time) to find line equation **y=ax+b** graph (V , $\omega(rad/s)$)
  _ **Step2:** you take **a=K** .
  _ **Step3:** How to find time constant τ
  In first-order system, the settling time is equal to :

$$\boldsymbol{T = 4\tau \Rightarrow \tau = 4/T}$$

- **Method2:**
  _ **Step1:study only on Electrical part**

$$V_A = L_a\frac{di}{dt} + R_a i_a + K_B\omega$$

  Where L_a, R_a and K_B are constant
  At steady state : ω_{ss}=const,i_a=const, then $\frac{di_a}{dt} = 0$ Hence,(1) becomes

$$V_A = K_B\omega + R_a i_a$$

  _ **Step2:Measurable variable**V_A, ω **and** i_a
  You input Voltage($V_A = 2V : 24V : 2V$) and measure i_a and ω
  _ **Step3:Identifying Ra and** K_B **for a DC motor**
  We have $V_A = K_B\omega + R_a i_a = \begin{bmatrix}\omega & i_a\end{bmatrix}\begin{bmatrix}K_B \\ i_a\end{bmatrix}$

$$\begin{bmatrix}\omega_1 & i_1 \\ \omega_2 & i_2 \\ \omega_3 & i_3 \\ \omega_4 & i_4 \\ \omega_5 & i_5 \\ \omega_6 & i_6 \\ \omega_7 & i_7 \\ \omega_8 & i_8 \\ \omega_9 & i_9\end{bmatrix}\begin{bmatrix}K_B \\ R_a\end{bmatrix} = \begin{bmatrix}V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \\ V_6 \\ V_7 \\ V_8 \\ V_9\end{bmatrix}$$

Ax=B

  Then $\mathbf{x} = \begin{bmatrix}\mathbf{K_B} \\ \mathbf{R_a}\end{bmatrix} = \mathbf{(A^TA)^{-1}A^Tb}$

  Note: For method2 I have not tried it yet

2.2 PI Tuning

- **Method1:**

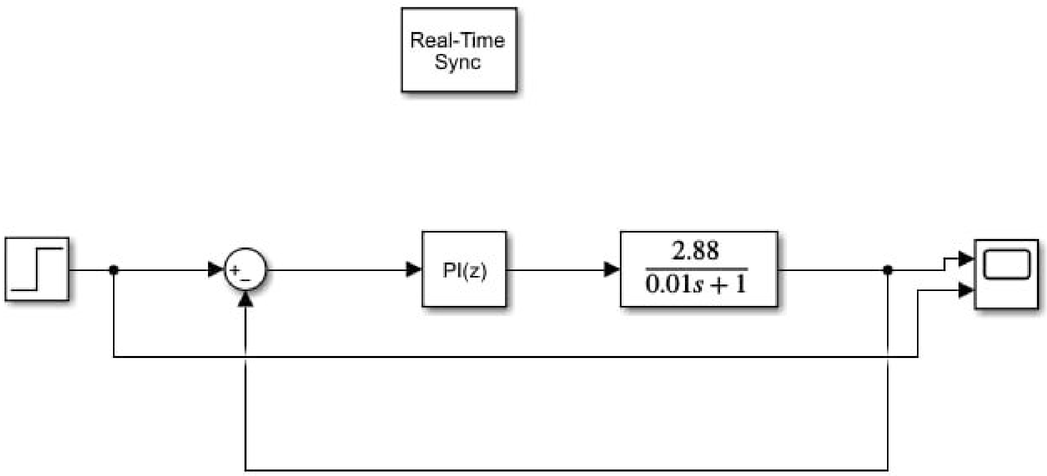

Figure 3: PI Tuning Method1 on Speed DC motor

- **Method2:**

```matlab

% Parameter modeling DC Motor
K = 3.27;
tau = 0.044;
ts = 0.2;  %settling time
zeta = 0.9    ;
w = 4/(zeta*ts);

% CACULATE Kp, Ki
Kp = (2*zeta*tau*w-1)/K;
Ki = (tau*w^2)/K;

s = tf('s')
P = K/(tau*s+1) % Plant
C = Kp+Ki/s % Control

H = C*P/(1+C*P) % Closed loop transfer function
minreal(H)

step(H)
stepinfo(H)
```

Figure 4: PI Tuning Method2 on Speed DC motor

3 Rotary_Encoder

3.1 How to work of Rotary_Encoder

Encoders use different types of technologies to create a signal, including: mechanical, magnetic, resistive and optical being the most common. In optical sensing, the encoder provides feedback based on the interruption of light.

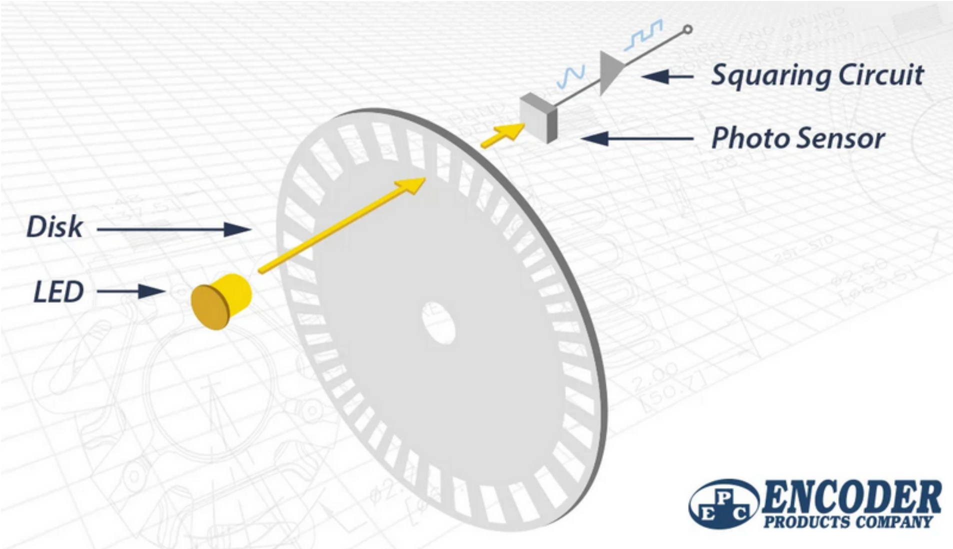

Figure 5: Process of Encoder

- A beam of light emitted from an LED passes through a code Disk, a transparent disk patterned with opaque lines (much like the spokes on a bike wheel)
- The light beam is picked up by a Photodetector Assembly,also called a photodiode array or a photosensor.
- The Photodetector Assembly responds to the light beam,producing a sinusoidal wave form,which is transformed into a square wave or pulse train.
- This pulse signal is simple:
  light=on
  no light =off
- The pulse signal is then sent to the counter or controller through the Electronics Board.
- The counter or controller (not pictured)then sends the signal to produce the proper function (stop,go,rotate,etc.).

3.2 Calculate Position from Rotary_encoder

1. **Step1:How to get count**
   There two method to got it:
   - External Interrupt
   - TIMER Encoder
2. **Step2:Calculate speed(rd/s)**

- $\mathbf{speed} = \frac{\mathbf{dcount}}{\mathbf{dt}}(rpm)$
  Where: $\mathbf{dcount} = \mathbf{count_k} - \mathbf{count_{k-1}}$, $\mathbf{dt} = \boldsymbol{\Delta}\mathbf{t} = \mathbf{sampling\_time}$
- $\boldsymbol{\omega} = \frac{\boldsymbol{2\pi speed}}{\boldsymbol{CPR \times dt}}$

  Where CPR: Count Per Revolution.

3. **Step3:Calculate Speed (one round wheel per second)**

$$\boldsymbol{V = \omega \times R}$$

4. **Step3: Integral Speed**

$$\boldsymbol{X_k = X_{k-1} + V * dt}$$

3.3 Code Read Rotary_Encoder

1. **Using External Interrupt**

```c
/* USER CODE BEGIN 4 */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    // ENCODER Rotary 1
    if (GPIO_Pin == E1_C1_Pin || E1_C2_Pin)
    {
        nowA[0] = HAL_GPIO_ReadPin(E1_C1_GPIO_Port, E1_C1_Pin);
        nowB[0] = HAL_GPIO_ReadPin(E1_C2_GPIO_Port, E1_C2_Pin);
        Enc_count[0] = encoder(0);

    }
    // ENCODER Rotary 2
    if (GPIO_Pin == E2_C1_Pin || E2_C2_Pin)
    {
        nowA[1] = HAL_GPIO_ReadPin(E2_C1_GPIO_Port, E2_C1_Pin);
        nowB[1] = HAL_GPIO_ReadPin(E2_C2_GPIO_Port, E2_C2_Pin);
        Enc_count[1] = encoder(1);
    }

}
/* USER CODE END 4 */
```

Figure 6: Read 1,0 from encoder

```
float encoder(int i)
{
	if (nowA[i] != lastA[i]){
		lastA[i] = nowA[i];
		if (lastA[i] == 0){
			if (nowB[i] == 0){
				dir[i] = 0;
				cnt[i]--;
			}
			else{
				dir[i] = 1;
				cnt[i]++;
			}
		}
		else{
			if (nowB[i] == 1){
				dir[i] = 0;
				cnt[i]--;
			}
			else{
				dir[i] = 1;
				cnt[i]++;
			}
		}
	}
	if (nowB[i] != lastB[i]){
		lastB[i] = nowB[i];
		if (lastB[i] == 0){
			if (nowA[i] == 1){
				dir[i] = 0;
				cnt[i]--;
			}
			else{
				dir[i] = 1;
				cnt[i]++;
			}
		}
		else{
			if (nowA[i] == 0){
				dir[i] = 0;
				cnt[i]--;
			}
			else{
				dir[i] = 1;
				cnt[i]++;
			}
		}
	}
	return cnt[i];
}
```

Figure 7: Convert Input read (1,0) to count

2. **Using TIMER Encoder**

```c

void read_encoder(Encoder *enc, TIM_HandleTypeDef* timer){
    enc->new_counter = __HAL_TIM_GET_COUNTER(timer);
    enc->counter_status = __HAL_TIM_IS_TIM_COUNTING_DOWN(timer);
    int16_t count_change = enc->new_counter - enc->counter;
    if(enc->counter_status && count_change <0){
        count_change += 65536;
    }else if (!enc->counter_status && count_change > 0){
        count_change -= 65536;
    }
    enc->counter = enc->new_counter;
    enc->counter_status = (count_change >=0);
    enc->speed = (float)count_change*1000.0f/(CPR_X * sampling_time);
    enc->rdps = (float)count_change*2*PI*1000.0f/(CPR_X * sampling_time);
}
/* USER CODE END PFP */
```

Figure 8: function read count and speed

```c
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  /* USER CODE BEGIN Callback 0 */
    if (htim->Instance == TIM3)
      {
        read_encoder(&encoder0, &htim1);
        read_encoder(&encoder1, &htim2);
        c2 = encoder0.counter;
        c1 = encoder1.counter;
        W1 = -1 * encoder1.rdps * r;
        W2 = encoder0.rdps * r;
        Vx_enR = W1 * cosf(theta) - W2 * sinf(theta);
        Vy_enR = W1 * sinf(theta) + W2 * cosf(theta);

        X_enR = X_old_enR + Vx_enR * dt;
        Y_enR = Y_old_enR + Vy_enR * dt;
        X_old_enR = X_enR;
        Y_old_enR = Y_enR;
```

Figure 9: calculate position

4 IMU BNO055

4.1 How to work

Bosch BNO055 is 9-axis orientation sensor with 3-axis gyro, 3-axis accelerometer and 3-axis magnetometer. It's smart sensor, that support sensor fusion and self-diagnostics. We made a library to use it with STM32 micro-controllers. We needed only roll, pitch and heading, so we wrote it in the first place, but adding other sensor values are easy. If you want to test the library, you will need STCubeMX and IAR (demo) to build simple project. Here is how to do it.

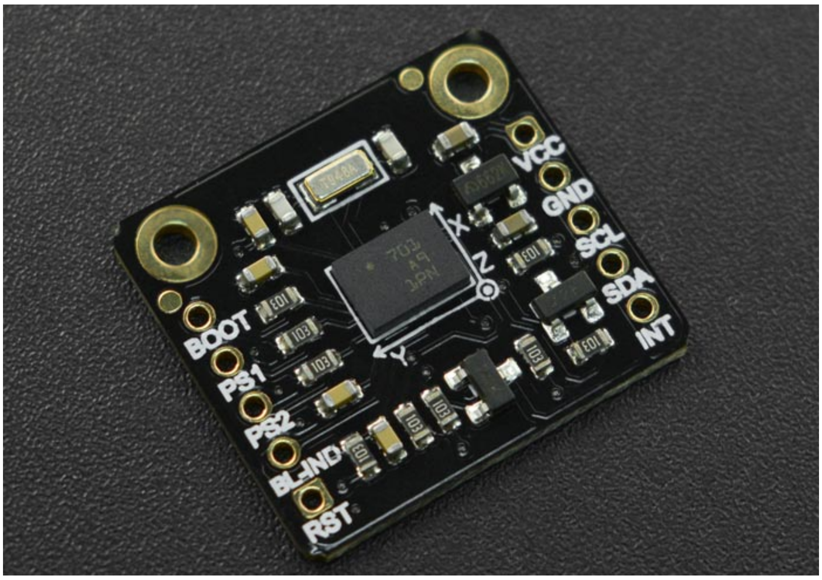

Figure 10: IMU BNO055 Black

4.2 How to Configuration

Open STCubeMX and create new project. Select board or chip. I used Nucleo(stm32F446RETx) in my avionics project and will show periphery configuration as example. Here is the minimal settings you need to configure:

- **Step1:** Enable debug: Sys − > Debug − > Serial Wire.
- **Step2:** Enable I2C: Connectivity − > I2C1 − > I2C1.
- **Step3:** Enable fast mode: I2C1 − > I2C speed mode − > Fast mode.

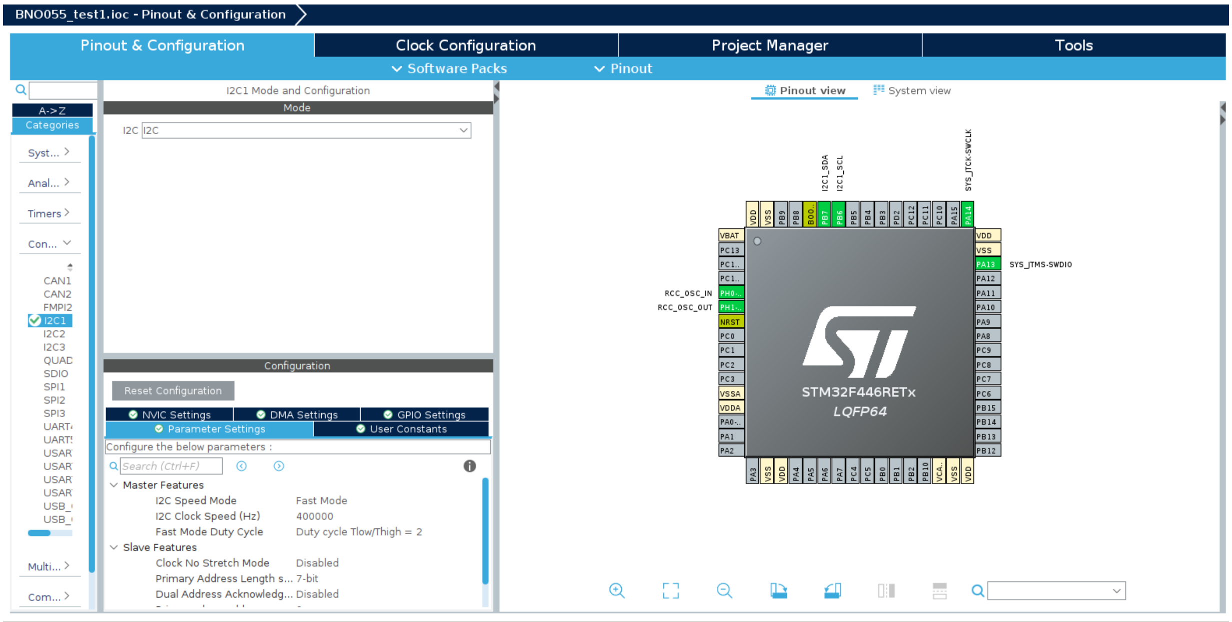

Figure 11: Configure I2C for IMU

- **Step4:** Go to Clock Configuration and set Frequency CPU (180 MHz).
- **Step5:** Go to Project Manager – > Code Generate and Click (Generate peripheral initialization as a pair of '.c/.h' file per peripheral)

4.3 How to Write Code

- **Step1:** Go to github (https://github.com/ivyknob/bno055_stm32) and download 3 files:
  - bno055.c - main library code
  - bno055.h - common header for every platform
  - bno055_stm32.h - stm32 specific I2C header files
- **Step2:** Copy all files above to your project. Using IAR as example, you need to copy header files to ./Core/Inc and bno055.c to ./Core/Src and then add file with right click on Application – > User – > Core in project tree and Add – > Add files ... (select bno055.c).
- **Step3:** Write Code in main.c
  - In USER CODE BEGIN Includes code block add:
    #include "bno055_stm32.h"
  - In USER CODE BEGIN PTD
    bno055_vector_t Q;

```c
/* USER CODE END Header_RotaryIMU_Init */
void RotaryIMU_Init(void const * argument)
{
  /* USER CODE BEGIN RotaryIMU_Init */
    bno055_assignI2C(&hi2c1);
    bno055_setup();
    bno055_setOperationModeNDOF();

  /* Infinite loop */
  for(;;)
  {
      // Qauternion_to_Euler(Angle);
        Q = bno055_getVectorQuaternion();
        // yaw (z-axis rotation)
        siny_cosp = 2 * (Q.w * Q.z + Q.x * Q.y);
        cosy_cosp = 1 - 2 * (Q.y * Q.y + Q.z * Q.z);
        Angle.Yaw = atan2(siny_cosp, cosy_cosp);

    osDelay(10);
  }
  /* USER CODE END RotaryIMU_Init */
}
```

Figure 12: calculate yaw from quaternion

5 CAN_BUS

The Controller Area Network (CAN bus) is a message-based protocol designed to allow the Electronic Control Units (ECUs) found in today's automobiles, as well as other devices, to communicate with each other in a reliable, priority-driven fashion. Messages or "frames" are received by all devices in the network, which does not require a host computer.

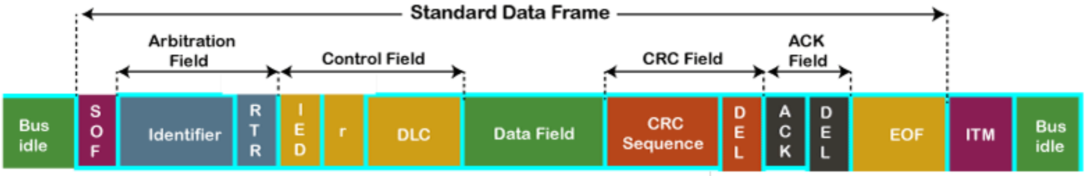

Figure 13: CAN Frame

5.1 CAN_BUS in STM32

Master STM32F401VGT6 with Slave STM32F103C8

1. **How to Configure STM32F401VGT6**

   - textbfStep1:Go to TIMER3 and Setup Internal Clock 10ms to crate timer Interrupt loop in 10ms.
   - **Step2:**Go to Connectivity $->$ CAN1 $->$ Click Activeted and I want to choose baud rate (1Mbit/s):Formular Buad Rate CAN

   $$\mathbf{Baud\_Rate} = \frac{\mathbf{APB(PeripheralClock)}}{\mathbf{Prescaler(tims1 + tims2 + 1)}}$$

   $\Rightarrow$ PSC=6, tims1=4, tims2=2;
   After go to click **CAN1 RX0 interrupt**

2. **Setup Code in STM32F401VGT6**

   - **Step1:** CAN Interrupt Receive Function

```
void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan) {
    HAL_CAN_GetRxMessage(hcan, CAN_RX_FIFO0, &RxHeader, RxData);
    cntt++;
    while (cntt - 100 > 0) {
        //HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_13);
        cntt = 0;
    }

    if (RxHeader.StdId == 0x215) {
            RxData1 = (RxData[0] << 8) | RxData[1];
            RxData2 = (RxData[2] << 8) | RxData[3];
            V1_back = map(RxData1, 0, 65535, -30.0, 30.0);
            V2_back = map(RxData2, 0, 65535, -30.0, 30.0);
            datacheck = 1;
    }
    else if (RxHeader.StdId == 0x211) {
            RxData3 = (RxData[0] << 8) | RxData[1];
            RxData4 = (RxData[2] << 8) | RxData[3];
            V3_back = map(RxData3, 0, 65535, -30.0, 30.0);
            V4_back = map(RxData4, 0, 65535, -30.0, 30.0);
            datacheck = 1;
    }
}
```

Figure 14: function CAN receive interrupt in STM32F407VGT6

- **Step2:** Create frame Transmit

```c
  /* USER CODE BEGIN 2 */
  // CAN  Transmition
    HAL_CAN_Start(&hcan1);
    HAL_CAN_ActivateNotification(&hcan1, CAN_IT_RX_FIFO0_MSG_PENDING);
    TxHeader.DLC = 8; // data length
    TxHeader.IDE = CAN_ID_STD;
    TxHeader.RTR = CAN_RTR_DATA;
    TxHeader.StdId = 0x407; //Id 0x7FF
```

Figure 15: Create Header Transmit

- **Step3:** Transmit CAN in TIMER Interrupt Function

```c
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  /* USER CODE BEGIN Callback 0 */
    if (htim->Instance == TIM3) {

        V1_out = map(Vx, -2.0, 2.0, 0, 65535);
        V2_out = map(Vy, -2.0, 2.0, 0, 65535);
        V3_out = map(Omega, -3.14, 3.14, 0, 65535);
        V4_out = map(Speed, 0.0, 2.0, 0, 65535);
        TxData[0] = ((V1_out & 0xFF00) >> 8);
        TxData[1] = (V1_out & 0x00FF);
        TxData[2] = ((V2_out & 0xFF00) >> 8);
        TxData[3] = (V2_out & 0x00FF);
        TxData[4] = ((V3_out & 0xFF00) >> 8);
        TxData[5] = (V3_out & 0x00FF);
        TxData[6] = ((V4_out & 0xFF00) >> 8);
        TxData[7] = (V4_out & 0x00FF);
        HAL_CAN_AddTxMessage(&hcan1, &TxHeader, TxData, &TxMailbox);
        }
```

Figure 16: Transmit CAN_BUS

- **Step4:** Go to Configure Filter CAN(Filter Bank and Mask ID Filter)

```c
/* CAN1 init function */
void MX_CAN1_Init(void)
{

  /* USER CODE BEGIN CAN1_Init 0 */

  /* USER CODE END CAN1_Init 0 */

  /* USER CODE BEGIN CAN1_Init 1 */

  /* USER CODE END CAN1_Init 1 */
  hcan1.Instance = CAN1;
  hcan1.Init.Prescaler = 6;
  hcan1.Init.Mode = CAN_MODE_NORMAL;
  hcan1.Init.SyncJumpWidth = CAN_SJW_1TQ;
  hcan1.Init.TimeSeg1 = CAN_BS1_4TQ;
  hcan1.Init.TimeSeg2 = CAN_BS2_2TQ;
  hcan1.Init.TimeTriggeredMode = DISABLE;
  hcan1.Init.AutoBusOff = DISABLE;
  hcan1.Init.AutoWakeUp = DISABLE;
  hcan1.Init.AutoRetransmission = DISABLE;
  hcan1.Init.ReceiveFifoLocked = DISABLE;
  hcan1.Init.TransmitFifoPriority = DISABLE;
  if (HAL_CAN_Init(&hcan1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN CAN1_Init 2 */
  CAN_FilterTypeDef canfilterconfig;

        canfilterconfig.FilterActivation = CAN_FILTER_ENABLE;
        canfilterconfig.FilterBank = 0; // anything between 0 to slaveStartFilterBank (10,14)
        canfilterconfig.FilterFIFOAssignment = CAN_FILTER_FIFO0;
        canfilterconfig.FilterIdHigh = 0x0000;
        canfilterconfig.FilterIdLow = 0x0000;
        canfilterconfig.FilterMaskIdHigh = 0x0000;
        canfilterconfig.FilterMaskIdLow = 0x0000;
        canfilterconfig.FilterMode = CAN_FILTERMODE_IDMASK;
        canfilterconfig.FilterScale = CAN_FILTERSCALE_32BIT;
        canfilterconfig.SlaveStartFilterBank = 14; // how many filter to assign to the CAN1 (master Can)(13 to 27 a

        HAL_CAN_ConfigFilter(&hcan1, &canfilterconfig);
  /* USER CODE END CAN1_Init 2 */

}
```

Figure 17: Configure filter bank

3. **How to Configure STM32F103C8T6**

- **Step1:**Go to Connectivity $->$ CAN1 $->$ Click Activeted and I want to choose baud rate (1Mbit/s):Formular Buad Rate CAN

$$\mathbf{Baud\_Rate} = \frac{\mathbf{APB(PeripheralClock)}}{\mathbf{Prescaler(tims1 + tims2 + 1)}}$$

$\Rightarrow$ PSC=9, tims1=2, tims2=1;
After go to click **CAN RX1 interrupt**

4. **Setup code in STM32F103C8T6**

- **Step1:** CAN Rx1 Interrupt function

```c
void HAL_CAN_RxFifo1MsgPendingCallback(CAN_HandleTypeDef *hcan) {
    HAL_CAN_GetRxMessage(hcan, CAN_RX_FIFO1, &RxHeader, RxData);
        cntt++;
        while (cntt - 100 > 0) {
            //HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_13);
            cntt = 0;
        }


    if (RxHeader.StdId == 0x407) {
        RxData1 = (RxData[0] << 8) | RxData[1];
        RxData2 = (RxData[2] << 8) | RxData[3];
        RxData3 = (RxData[4] << 8) | RxData[5];
        RxData4 = (RxData[6] << 8) | RxData[7];
        V1 = RxData1;
        V2 = RxData2;
        V3 = RxData3;
        V4 = RxData4;

        Vx = map(V1, 0, 65535, -1.5, 1.5);
        Vy = map(V2, 0, 65535, -1.5, 1.5);
        Vz = map(V3, 0, 65535, -3.14, 3.14);
        speed = map(V4, 0, 65535, 0.1, 0.5);
        flag = 1;

    }
}
```

Figure 18: function CAN Rx1 interrupt in STM32F103C8T6

- **Step2:** Create frame Transmit

```c

    HAL_CAN_Start(&hcan);
    HAL_CAN_ActivateNotification(&hcan, CAN_IT_RX_FIFO1_MSG_PENDING);
    TxHeader.DLC = 8;
    TxHeader.IDE = CAN_ID_STD;
    TxHeader.RTR = CAN_RTR_DATA;
    TxHeader.StdId = 0x215; //Slave1
```

Figure 19: Create Frame Transmit in STM32F103C8T6

- **Step3:** Transmit Data

```
        V1_out = map(Motor1_speed, -30, 30, 0, 65535);
        V2_out = map(Motor2_speed, -30, 30, 0, 65535);
        TxData[0] = ((V1_out & 0xFF00) >> 8);
        TxData[1] = (V1_out & 0x00FF);
        TxData[2] = ((V2_out & 0xFF00) >> 8);
        TxData[3] = (V2_out & 0x00FF);
        if (flag == 1){
            HAL_CAN_AddTxMessage(&hcan, &TxHeader, TxData, &TxMailbox);
            flag = 0;
        }
    }
```

Figure 20: Transmit Data in STM32F103C8T6

- **Step4:** Go to Configure Filter CAN(Filter Bank and Mask ID Filter)

```
/* USER CODE BEGIN CAN_Init 2 */
CAN_FilterTypeDef canfilterconfig;

  canfilterconfig.FilterActivation = CAN_FILTER_ENABLE;
  canfilterconfig.FilterBank = 10; // which filter bank to use from the assigned ones
  canfilterconfig.FilterFIFOAssignment = CAN_FILTER_FIFO1;
  canfilterconfig.FilterIdHigh = 0x407 << 5;  //407<<5
  canfilterconfig.FilterIdLow = 0;
  canfilterconfig.FilterMaskIdHigh = 0x407 << 5;  //407<<5
  canfilterconfig.FilterMaskIdLow = 0x0000;
  canfilterconfig.FilterMode = CAN_FILTERMODE_IDMASK;
  canfilterconfig.FilterScale = CAN_FILTERSCALE_32BIT;
  canfilterconfig.SlaveStartFilterBank = 0; // doesn't matter in single can controllers

  HAL_CAN_ConfigFilter(&hcan, &canfilterconfig);
/* USER CODE END CAN_Init 2 */
```

Figure 21: Configure filter bank in STM32F103C8T6

5.2 CAN_BUS in Python_ROS2

1. **How to configure USB CAN**: Use SocketCAN
   - sudo ip link set can0 type can bitrate 1000000
   - sudo ip link set up can0
2. **Install library can and command to check CAN**
   - pip install python-can
   - sudo pip3 install canprog
   - sudo apt install python3-can
   - candump can0 (to view frame of CAN Tx,Rx)
   - Then canprog seems to work:
     canprog -h
   - STM32 bootloader option:
     canprog stm32 [-h]
3. **Setup Code**
   - Create timer for transmit and receive data and create interface can

```
        #Publish CANBUS to STM32
        self.timer = 0.001
        self.bus =can.interface.Bus(channel='can0', interface ='socketcan', bitrate=1000000)
        self.can_timer_ = self.create_timer(self.timer, self.timerCanCB)
```

Figure 22: Interface of can

- Transmit and Receive data

```python
###*** Publisher CAN_BUS to STM32 ***###
def timerCanCB(self):
    # encoder = Float32MultiArray()
    # external = Float32MultiArray()
    msg = can.Message(arbitration_id=0x103, is_extended_id=False, data=self.TxData)
    self.bus.send(msg,0.001) #time out 10ms
    # self.get_logger().info('Velocity transmit to STM32:[%f, %f, %f]'%(self.Vx,self.Vy,self.Vyaw))
    for i in range(3):
        try:
            can_msg =self.bus.recv(0.01)
            if(can_msg != None):
                if can_msg.arbitration_id == 0x407:
                    self.rotary[0] = ((can_msg.data[0] << 8) | can_msg.data[1])
                    self.rotary[1] = ((can_msg.data[2] << 8) | can_msg.data[3])
                    Omega_Back = ((can_msg.data[4] << 8) | can_msg.data[5])
                    self.Omega_back = float(map(Omega_Back,0,65535,-6.28,6.28))
                elif can_msg.arbitration_id == 0x215:
                    self.V1 = ((can_msg.data[0] << 8) | can_msg.data[1])
                    self.V2 = ((can_msg.data[2] << 8) | can_msg.data[3])
                    self.V1_back=float(map(self.V1,0,65535,-30,30))
                    self.V2_back=float(map(self.V2,0,65535,-30,30))

                elif can_msg.arbitration_id == 0x211:
                    self.V3 = ((can_msg.data[0] << 8) | can_msg.data[1])
                    self.V4 = ((can_msg.data[2] << 8) | can_msg.data[3])
                    self.V3_back=float(map(self.V3,0,65535,-30,30))
                    self.V4_back=float(map(self.V4,0,65535,-30,30))
            else:
                self.get_logger().error('time out on msg recv!')

        except can.CanOperationError:
            pass
```

Figure 23: Tx and RX in python-can

6 Kinematic of Mecanum Wheel

6.1 What is Mecanum Wheel

The mecanum wheel is a form of tireless wheel, with a series of rubberized external rollers obliquely attached to the whole circumference of its rim. These rollers typically each have an axis of rotation at 45° to the wheel plane and at 45° to the axle line.

6.2 Kinematic of Mecanum wheel

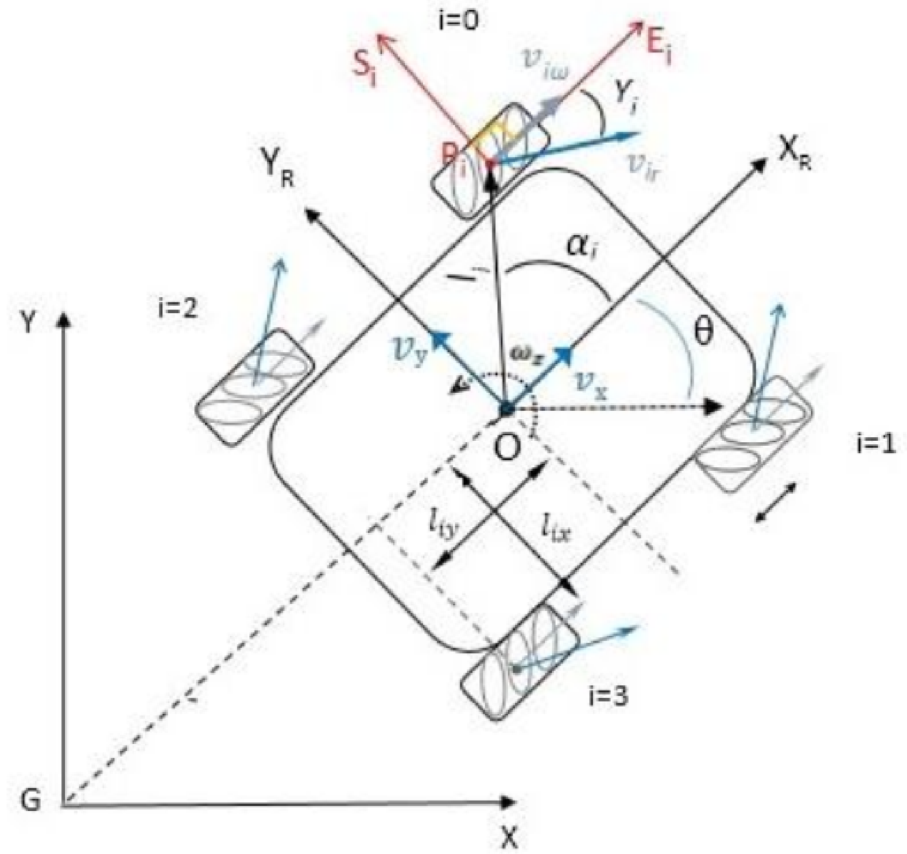

Figure 24: Wheels Configuration and Posture definition

The configuration parameter and system velocities are defind as follows:

- x, y, θ**:** Robot's position (x,y) and its orientation angle θ (The angle between X and X_R);
- **X,G,Y:** Inertial frame;x,y are the coordinates of the reference point O in the inertial basis;
- $X_R O Y_R$**:** Robot's base frame; Cartesian coordinate system associated with the movement of the body center;
- S_i, P_i, E_i**:** Coordinate system of ith wheel in the wheel's center point P_i;
- O, P_i**:** The inertial basis of the Robot in Robot's frame and $P_i = \{X_{P_i}, Y_{P_i}\}$ the center of the rotation axis of the wheel i;
- $\vec{OP_i}$**:** is a vector that indicates the distance between Robot's center and the center of the wheel ith;

- l_{ix}, l_{iy}: l_{ix} :half of the distance between front wheels and l_{iy} : half of the distance between front wheel and the base (center of the robot O);
- l_i: distance between wheels and the base (center of the robot O);
- r_i: denotes the radius of the wheel i (Distance of the wheel's center to the roller center)
- r_r: denotes the radius of the rollers on the wheel.
- α_i: the angel between $OP_i and X_R$;
- β_i: the angle between S_i and X_R;
- γ_i: the angle between v_{ir} and E_i;
- θ**[rad/s]:** wheels angular velocity;
- $v_{i\theta}$**[rad/s]:** wheels angular velocity;
- V_{ir}: the velocity of the passive roller in the wheel i;
- $[W_{si}\ \ W_{Ei}\ \ \theta_i]^T$: Genralizd velocity of point P_i in the frame $S_iP_iE_i$;
- $[V_{si}\ \ V_{Ei}\ \ \theta_i]^T$: Generalized velocity of point P_i in the frame X_ROY_R;
- $v_x v_y[m/s]$: Robot linear velocity;
- $\theta[rad/s]$:Robot angular velocity;

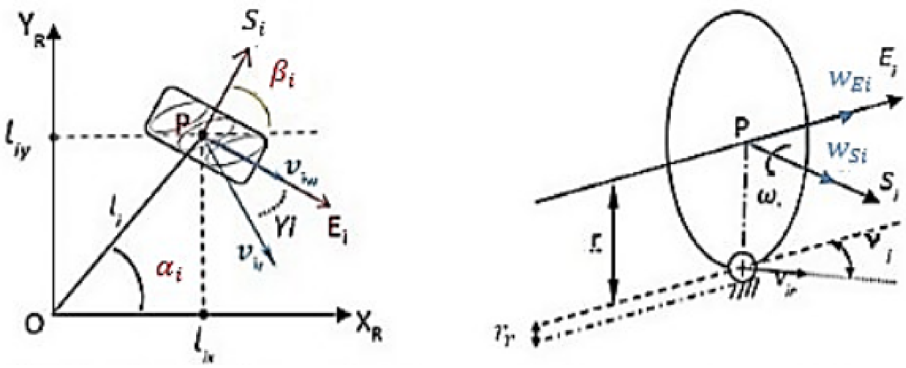

Figure 25: Wheels i in the robot coordinate and wheel i motion principle

- **Wheel i and the tangential velocity of the free roller attaced to the wheel touching the floor:**

$$\mathbf{v_{ir}} = \frac{\mathbf{1}}{\mathbf{cos(45)}}\mathbf{r_r}\omega_\mathbf{i}, \quad \mathbf{W_{Ei}} = \mathbf{r_i}\theta_\mathbf{i}, \quad \mathbf{i = 0, 1, 2, 3}. \quad \textbf{(eq.1)}$$

- **The velocity of the wheel i in the frame $S_iP_iE_i$, can be derived by:**

$$\boldsymbol{v_{si} = v_{ir} sin(\gamma_i)}$$

$$\boldsymbol{v_{Ei} = \omega_i r_i + v_{ir} cos(\gamma_i).}$$

$$\begin{bmatrix} \mathbf{v_{S_i}} \\ \mathbf{v_{E_i}} \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \mathbf{sin(\gamma_i)} \\ \mathbf{r_i} & \mathbf{cos(\gamma_i)} \end{bmatrix} \begin{bmatrix} \boldsymbol{\omega_i} \\ \mathbf{v_{ir}} \end{bmatrix} = \mathbf{T^{w_i}_{P_i}} \begin{bmatrix} \boldsymbol{\omega_i} \\ \mathbf{v_{ir}} \end{bmatrix}. \quad \textbf{(eq.2)}$$

- **The transformation matrix from velocities of the ith wheel to its center:**
  $T_{P_i}^{w_i} == \begin{bmatrix} \mathbf{0} & \mathbf{sin}(\gamma_\mathbf{i}) \\ \mathbf{r_i} & \mathbf{cos}(\gamma_\mathbf{i}) \end{bmatrix}$

- **The velocity of the wheel's center translated to the $X_R O Y_R$coordinate system can be achieved by equation 7:**

$$\begin{bmatrix} \mathbf{v_{iX_R}} \\ \mathbf{v_{iY_R}} \end{bmatrix} = \begin{bmatrix} \mathbf{cos}(\beta_\mathbf{i}) & -\mathbf{sin}(\beta_\mathbf{i}) \\ \mathbf{sin}(\beta_\mathbf{i}) & \mathbf{cos}(\beta_\mathbf{i}) \end{bmatrix} \begin{bmatrix} \mathbf{v_{S_i}} \\ \mathbf{v_{E_i}} \end{bmatrix} = T_{P_i}^{w_i} T_R^{P_i} \begin{bmatrix} \boldsymbol{\omega_i} \\ \mathbf{v_{ir}} \end{bmatrix} \qquad \textbf{(eq.3)}$$

- **The transformation matrix from the ith wheel's center to the robot coordinate's system can be obtained from equation**
  $T_R^{P_i} = \begin{bmatrix} \mathbf{cos}(\beta_\mathbf{i}) & -\mathbf{sin}(\beta_\mathbf{i}) \\ \mathbf{sin}(\beta_\mathbf{i}) & \mathbf{cos}(\beta_\mathbf{i}) \end{bmatrix}$

- **The robot's motion is planar,we also have:**

$$\begin{bmatrix} \mathbf{v_{iX_R}} \\ \mathbf{v_{iY_R}} \end{bmatrix} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & -\mathbf{l_{iy}} \\ \mathbf{0} & \mathbf{1} & \mathbf{l_{ix}} \end{bmatrix} \begin{bmatrix} \mathbf{v_{X_R}} \\ \mathbf{v_{y_R}} \\ \omega_\mathbf{R} \end{bmatrix} = T' \begin{bmatrix} \mathbf{v_{X_R}} \\ \mathbf{v_{y_R}} \\ \omega_\mathbf{R} \end{bmatrix} \qquad \textbf{(eq.4)}$$

  Where:$T' = \begin{bmatrix} \mathbf{1} & \mathbf{0} & -\mathbf{l_{iy}} \\ \mathbf{0} & \mathbf{1} & \mathbf{l_{ix}} \end{bmatrix}$

- **_The inverse kinematic model can be obtained:**

$$\mathbf{T_{P_i}^{w_i} T_R^{P_i}} \begin{bmatrix} \boldsymbol{\omega_i} \\ \mathbf{v_{ir}} \end{bmatrix} = \mathbf{T'} \begin{bmatrix} \mathbf{v_{X_R}} \\ \mathbf{v_{y_R}} \\ \theta_\mathbf{R} \end{bmatrix} \qquad \textbf{(eq.5)}$$

- **The robot's base velocity (at point O) related to the rotational velocity of the ith wheel can be obtained from eq.6.**

$$\begin{bmatrix} \boldsymbol{\omega_i} \\ \mathbf{v_{ir}} \end{bmatrix} = \mathbf{T_{P_i}^{w_i} . T_R^{P_i} . T'} \begin{bmatrix} \mathbf{v_{X_R}} \\ \mathbf{v_{y_R}} \\ \boldsymbol{\omega_R} \end{bmatrix} \qquad \textbf{(eq.6)}$$

  Where: $\mathrm{T}=(T_{P_i}^{w_i})^{-1}.(T_R^{P_i})^{-1}.T'$

$$\mathbf{T}=\begin{bmatrix} \mathbf{cos}(\beta_\mathbf{i}) & -\mathbf{sin}(\beta_\mathbf{i}) \\ \mathbf{sin}(\beta_\mathbf{i}) & \mathbf{cos}(\beta_\mathbf{i}) \end{bmatrix}^{-1} . \begin{bmatrix} \mathbf{0} & \mathbf{sin}(\gamma_\mathbf{i}) \\ \mathbf{r_i} & \mathbf{cos}(\gamma_\mathbf{i}) \end{bmatrix}^{-1} . \begin{bmatrix} \mathbf{1} & \mathbf{0} & -\mathbf{l_{iy}} \\ \mathbf{0} & \mathbf{1} & \mathbf{l_{ix}} \end{bmatrix}$$

  Where: $l_{ix} = l_i cos(a_i)$ **and** $l_{iy} = l_i sin(a_i)$

$$\mathbf{T} = \frac{\mathbf{1}}{-\mathbf{r}} \begin{bmatrix} \frac{\mathbf{cos}(\boldsymbol{\beta}_\mathbf{i}-\gamma_\mathbf{i})}{\mathbf{sin}(\gamma_\mathbf{i})} & \frac{\mathbf{sin}(\beta\mathbf{i}-\gamma\mathbf{i})}{\mathbf{sin}(\gamma_\mathbf{i})} & \frac{\mathbf{l_i sin}(-\alpha_\mathbf{i}+\beta_\mathbf{i}-\gamma_\mathbf{i})}{\mathbf{sin}(\gamma_\mathbf{i}} \\ -\frac{\mathbf{rcos}(\beta_\mathbf{i})}{\mathbf{sin}(\gamma_\mathbf{i}} & -\frac{\mathbf{rcos}(\beta\mathbf{i})}{\mathbf{sin}(\gamma_\mathbf{i})} & -\frac{\mathbf{l_i sin}(-\alpha_\mathbf{i}+\beta_\mathbf{i})\mathbf{r}}{\mathbf{sin}(\gamma_\mathbf{i})} \end{bmatrix} \qquad \textbf{(eq.7)}$$

  Since there is a relation between independent variables v_{ir} and ω_i in each joint and the systems angular and linear velocity, assuming that there is no wheel slipping on the ground, the system inverse kinematic can be obtained by eq.8.

- **Take Velocity of 4_wheel:**

$$\begin{bmatrix} \boldsymbol{\omega_1} \\ \boldsymbol{\omega_2} \\ \boldsymbol{\omega_3} \\ \boldsymbol{\omega_4} \end{bmatrix} = \frac{-\mathbf{1}}{\mathbf{r}} \begin{bmatrix} \frac{cos(\beta_1-\gamma_1)}{sin(\gamma_1)} & \frac{sin(\beta_1-\gamma_1)}{sin(\gamma_1)} & \frac{l_1 sin(-\alpha_1+\beta_1-\gamma_1)}{sin(\gamma_i} \\ \frac{cos(\beta_2-\gamma_2)}{sin(\gamma_2)} & \frac{sin(\beta 2-\gamma 2)}{sin(\gamma_2)} & \frac{l_2 sin(-\alpha_2+\beta_2-\gamma_2)}{sin(\gamma_2} \\ \frac{cos(\beta_3-\gamma_3)}{sin(\gamma_3)} & \frac{sin(\beta 3-\gamma_3)}{sin(\gamma_3)} & \frac{l_3 sin(-\alpha_3+\beta_3-\gamma_i)}{sin(\gamma_3} \\ \frac{cos(\beta_4-\gamma_4)}{sin(\gamma_4)} & \frac{sin(\beta 4-\gamma 4)}{sin(\gamma_4)} & \frac{l_4 sin(-\alpha_4+\beta_4-\gamma_4)}{sin(\gamma_4} \end{bmatrix} \begin{bmatrix} \mathbf{V_x} \\ \mathbf{V_y} \\ \boldsymbol{\omega_z} \end{bmatrix} \qquad \textbf{(eq.8)}$$

i	Wheels	α_i	β_i	γ_i	l_i	l_{ix}	l_{iy}
0	1sw	$\pi/4$	$\pi/2$	$-\pi/4$	l	l_x	l_y
1	2sw	$-\pi/4$	$-\pi/2$	$\pi/4$	l	l_x	l_y
2	3sw	$3\pi/4$	$\pi/2$	$\pi/4$	l	l_x	l_y
3	4sw	$-3\pi/4$	$-\pi/2$	$-\pi/4$	l	l_x	l_y

Figure 26: Robots Parameter

- **We get Inverse kinematic form:**

$$\begin{bmatrix}\omega_1\\\omega_2\\\omega_3\\\omega_4\end{bmatrix} = \frac{1}{r}\begin{bmatrix}1 & -1 & -(l_x+l_y)\\1 & 1 & (l_x+l_y)\\1 & 1 & -(l_x+l_y)\\1 & -1 & (l_x+l_y)\end{bmatrix}\begin{bmatrix}V_x\\V_y\\\omega_z\end{bmatrix} \quad \textbf{(eq.9)}$$

- **Forward kinematic form:**

$$\begin{bmatrix}V_x\\V_y\\\omega_z\end{bmatrix} = \frac{r}{4}\begin{bmatrix}1 & 1 & 1 & 1\\-1 & 1 & 1 & -1\\-\frac{1}{(l_x+l_y)} & \frac{1}{(l_x+l_y)} & -\frac{1}{(l_x+l_y)} & \frac{1}{(l_x+l_y)}\end{bmatrix}\begin{bmatrix}\omega_1\\\omega_2\\\omega_3\\\omega_4\end{bmatrix} \quad \textbf{(eq.10)}$$

- **he resultant velocity and its direction in the stationery coordinate axis (x, y, z) can be achieved by the following equations (eq.11,12):**

$$\rho = \tan^{-1}(\frac{V_y}{V_x}), \quad \textbf{(eq.11)}$$

$$V_R = \sqrt{V_x^2 + V_y^2} \quad \textbf{(eq.12)}$$

- **Kinematic of Wheel for write Program**
  - **Forward Kinematic**

$$V_x = \frac{R}{4} * (\omega_1 + \omega_2 + \omega_3 + \omega_4)$$

$$V_y = \frac{R}{4} * (-\omega_1 + \omega_2 + \omega_3 - \omega_4)$$

$$\omega_z = \frac{R}{4(l_x + l_y)} * (-\omega_1 + \omega_2 - \omega_3 + \omega_4)$$

  - **Inverse Kinematic**

$$\omega_1 = \frac{1}{R} * (V_x - V_y - \omega_z(l_x + l_y))$$

$$\omega_2 = \frac{1}{R} * (V_x + V_y + \omega_z(l_x + l_y))$$

$$\omega_3 = \frac{1}{R} * (V_x + V_y - \omega_z(l_x + l_y))$$

$$\omega_4 = \frac{1}{R} * (V_x - V_y + \omega_z(l_x + l_y))$$

7 PID_Controller

7.1 What is PID_Controller?

A PID-Controller is a widely used feedback control mechanism in engineering and industrial applications. It is employed to regulate processes and systems by continuously adjusting an output based on the difference between a desired setpoint and the measured process variable.

7.2 PID Algorithm

- **Step1:Calculate Error**
  Error = Reference - Measurement.

- **Step2:Calculate P,I,D**
  $Propotional = kp * Error[-1]$
  $Integral = Integral\_old + ki * (Error[-1] + Error[-2]) * dt$
  $Derivative = kd * (Error[-1] - Error[-2])/dt$
  where kp,ki,kd (you can tunning on Modelling thought MATLAB_Simulink or tunning by hand(observe)

- **Step3:Calculate Output**
  Output=Propotional + Integral + Derivative.

7.3 Modifications to the algorithm

7.3.1 Integral windup

Integrator windup or reset windup, refers to the situation in a PID controller where a large change in setpoint occurs (say a positive change) and the integral term accumulates a significant error during the rise (windup), thus overshooting and continuing to increase as this accumulated error is unwound (offset by errors in the other direction). The specific problem is the excess overshooting.
How to use Integral windup:

- Set Integral Min, Integral Max.

- The Standard Integral= Integal_pev + ki*Error*dt.

- When you use Integral winup:
  if (Integral >= Integral_Max)
  {
  . Integral=Integral_Max;
  }
  else if (Integral <= Integral_Min)
  {
  . Integral=Integral_Min;
  }

7.3.2 Low pass filter

A low-pass filter in a PID (Proportional-Integral-Derivative) controller is a component that filters out high-frequency noise from the derivative (D) term of the PID controller's output. The purpose of the low-pass filter is to smooth the derivative term and reduce its sensitivity to rapid, high-frequency changes in the error signal. By doing so, the controller can better distinguish between genuine changes in the process variable and short-term noise, resulting in more stable and accurate control. How to use Low pass filter in PID:

- E(t)=setpoint(t)- measurement(t)
- Derivative of Positional Error after low pass filter:

$$\boldsymbol{E(t) = (1-\alpha) * E(t-1) + \alpha * \frac{derror(t)}{dt}}$$

- α Smoothing Factor (smaller for greater smoothing) (0< α <1)

7.3.3 Output Saturation

This objective is achieved by including a saturation block that ensure the output is always in the range $[OUT_{min}, OUT_{max}]$

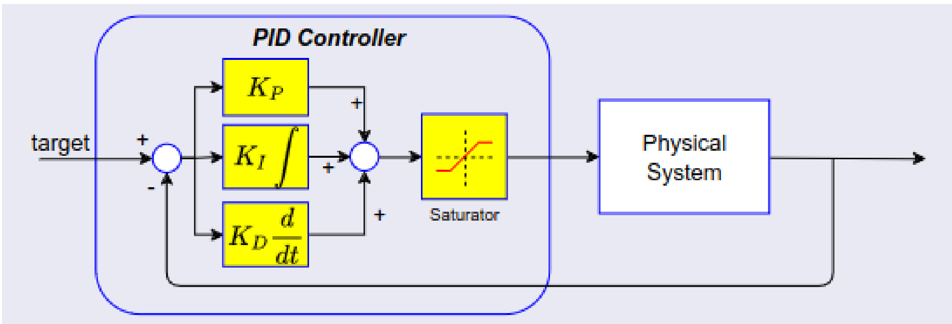

Figure 27: Output Saturation in PID Controller

How to use Output saturation in code:
Output = Propotional + Integral + Derivative.
if (Output >= Output_max)
{
. Output=Output_max;
}
else if (Output <= Output_min)
{
. Output=Output_min;
}
else
{
. Output=Output;
}

8 Extended Kalman Filter (EKF)

8.1 What is EKF?

The extended Kalman filter (EKF) is an approximate filter for nonlinear systems, based on first-order linearization of the process and measurement functions.It is a method used for estimating noise sensors.

8.2 EKF Algorithm

- **Step1:Initialization**
  - For the first iteration of EKF, we start at time k. In other words, for the first run of EKF, we assume the current time is k.
  - We initialize the state vector and control vector for the previous time step k-1.
- **Step2:Predicted State Estimate**
  - **Predicted state estimate:**$\mathbf{\hat{X}_{k|k-1} = f(\hat{X}_{k-1|k-1}, u_k)}$
  - We use the state space model, the state estimate for timestep k-1, and the control input vector at the previous time step (e.g. at time k-1) to predict what the state would be for time k (which is the current timestep).
  - That equation above is the same thing as our equation below. Remember that we used t in my earlier tutorials.
    $\mathbf{X_t = \hat{X}_{k|k-1} = A_{k-1}x_{k-1} + B_{k-1}U_{k-1} + V_{k-1}}$
- **Step3:Predicted Covariance of the State Estimate**
  - **Predicted covariance estimate:** $\mathbf{P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k}$
  - $\mathbf{P_{k-1|k-1}}$ is a square matrix. It has the same number of rows (and columns) as the number of states in the state vector x.
  - P (or, commonly, sigma $\sum$) is a 3×3 matrix. The P matrix has variances on the diagonal and covariances on the off-diagonal.
  - it is a matrix that represents an estimate of the accuracy of the state estimate we made in Step 2.
  - we initialize $\mathbf{P_{k-1|k-1}}$ to some guessed values (e.g. 0.1 along the diagonal part of the matrix and 0s elsewhere).

$$\mathbf{P_{k-1|k-1}} = \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{bmatrix}$$

  - $\mathbf{F_k and F_k^T}$:are equivalent to A_{t-1} and A_{t-1}^T, respectively, from my state space model tutorial.

$$\mathbf{F_k = F_k^T} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

  - $\mathbf{Q_k}$:is the state model noise covariance matrix. It is also a 3×3 matrix in our running robot car example because there are three states.

$$\mathbf{Q_k} = \begin{bmatrix} Cov(x,x) & Cov(x,y) & Cov(x,\gamma) \\ Cov(y,x) & Cov(y,y) & Cov(y,\gamma) \\ Cov(\gamma,x) & Cov(\gamma,y) & Cov(\gamma,\gamma) \end{bmatrix}$$

- The covariance between two variables that are the same is actually the variance. For example, $\mathbf{Cov}(\mathbf{x}, \mathbf{x}) = \mathbf{Var}(\mathbf{x})$.
- Variance measures the deviation from the mean for points in a single dimension (i.e. x values, y values, or yaw angle values).
- We can start by letting Q be the identity matrix and tweak the values through trial and error.

$$\mathbf{Q_k} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- when Q is large, it means we trust our actual sensor observations more than we trust our predicted sensor measurements from the observation model... more on this later in this tutorial.

- **Step4:Innovation or Measurement Residual**
  - **Innovation or measurement residual** $\mathbf{\tilde{y_k}} = \mathbf{z_k} - \mathbf{h}(\mathbf{\hat{x}_{k|k-1}})$
  - we calculate the difference between actual sensor observations and predicted sensor observations.
  - $\mathbf{z_k}$ is the observation vector. It is a vector of the actual readings from our sensors at time k.Matrix in z_k in mobile robot:

$$z_k = \begin{bmatrix} x_k \\ y_k \\ \gamma_k \end{bmatrix}$$

  - $\mathbf{h}(\mathbf{\hat{x}_{k|k-1}})$ is our observation model. It represents the predicted sensor measurements at time k given the predicted state estimate at time k from Step 2. That hat symbol above x means "predicted" or "estimated".

$$\boldsymbol{h(\hat{x}_{k|k-1}) = H_k \hat{x}_{k|k-1} + w_k}$$

  - We use the:
    measurement matrix $\mathbf{H_k}$ (which is used to convert the predicted state estimate at time k into predicted sensor measurements at time k),

$$\mathbf{H_k} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

    predicted state estimate $\mathbf{\hat{X}_{k|k-1}}$ that we calculated in Step 2,

$$\mathbf{\hat{X}_{k|k-1}} = AnswerFromStep2.$$

    and a sensor noise assumption $\mathbf{w_k}$ (which is a vector with the same number of elements as there are sensor measurements)

- **Step5:Innovation (or residual) Covariance**
  - Innovatiion (or residual) covariance : $\mathbf{S_k} = \mathbf{H_k P_{k|k-1} H_k^T} + \mathbf{R_k}$
  - use the predicted covariance of the state estimate $\mathbf{P_{k|k-1}}$ from Step 3.
  - The measurement matrix **Hk** and its transpose.

- **Rk** (sensor measurement noise covariance matrix... which is a covariance matrix that has the same number of rows as sensor measurements and same number of columns as sensor measurements)
- To start out by making $\mathbf{R_k}$ the identity matrix. You can then tweak it through trial and error

$$\mathbf{R_k} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- **Step6:Near-optimal Kalman Gain**
  - Near-optimal kalman Gain: $\mathbf{K_k = P_{k|k-1} H_k^T S_k^{-1}}$
  - If sensor measurement noise is large, then K approaches 0, and sensor measurements will be mostly ignored.
  - If prediction noise (using the dynamical model/physics of the system) is large, then K approaches 1, and sensor measurements will dominate the estimate of the state [x,y,yaw angle].
- **Step7:Updated State Estimate**
  - Update state estimate: $\mathbf{\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \tilde{y}_k}$
    In this step, we calculate an updated (corrected) state estimate based on the values from:
    _Step 2 (predicted state estimate for current time step k),
    _Step 6 (near-optimal Kalman gain from 6),
    _Step 4 (measurement residual).
- **Step8:Updated Covariance of the State Estimate**

$$\mathbf{P_{k|k} = (I - K_k H_k) P_{k|k-1}}$$

8.3 Example EKF on Mecanum Wheel

- **State Space Model:**
  - Form state_space: $\mathbf{X_k = AX_{k-1} + BU_k}$
  - State: X=$\begin{bmatrix} \mathbf{x} & \mathbf{y} & \boldsymbol{\theta} \end{bmatrix}^T$
  - Input: U=$\begin{bmatrix} \mathbf{u_1} & \mathbf{u_2} & \mathbf{u_3} & \mathbf{u_4} \end{bmatrix}$
  - Matrix A = $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
  - Matrix B = $\frac{\mathbf{r \times dt}}{\mathbf{4}} \begin{bmatrix} \boldsymbol{cos(\theta)} & \boldsymbol{-sin(\theta)} & 0 \\ \boldsymbol{sin(\theta)} & \boldsymbol{cos(\theta)} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{-1} & \mathbf{1} & \mathbf{1} & \mathbf{-1} \\ -\frac{\mathbf{1}}{\mathbf{(l_x+l_y)}} & \frac{\mathbf{1}}{\mathbf{(l_x+l_y)}} & -\frac{\mathbf{1}}{\mathbf{(l_x+l_y)}} & \frac{\mathbf{1}}{\mathbf{(l_x+l_y)}} \end{bmatrix}$
  - Adding Process Noise:

$$\boldsymbol{X_k = AX_{k-1} + BU_k + V_k}$$

Where $\mathbf{V_k} = \begin{bmatrix} noisex_k \\ noisey_k \\ noise\theta_k \end{bmatrix}$

- **Observation Model:**
  - Form : $\mathbf{Y_k = HX_k + W_k}$
  - Matrix H is measurement matrix (used to convert the state at time t into predicted sensor observations at time t) that has number_states rows and number_sensors columns. sensor: Rotary read (x,y) and IMU read (θ) so I have three sensor.
    $\Rightarrow H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
  - W_k is noise of sensor.
    $W_k = \begin{bmatrix} noise_k \\ noise_k \\ noise_k \end{bmatrix}$
- **Predicted:**
  - $\mathbf{X_k = \hat{X}_{k|k-1} = AX_{k-1} + B_{k-1}U_k + V_k}$
  - $\mathbf{P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k}$
  - Q is Matrix tunning on state
    $Q = \begin{bmatrix} \mathbf{gain\_Qx} & 0 & 0 \\ 0 & \mathbf{gain\_Qy} & 0 \\ 0 & 0 & \boldsymbol{gain\_Q\theta} \end{bmatrix}$
- **Update:**
  - Calculate difference between actual sensor and predicted sensor: $\mathbf{\tilde{Y}_k = Z_k - h(\hat{X}_{k|k-1})}$
  - Sensor read:$Z_k = \begin{bmatrix} RotaryX_k \\ RotaryY_k \\ IMU\theta_k \end{bmatrix}$
  - Predicted sensor : $\mathbf{h(\widehat{X_{k|k-1}}) = Y_k}$
  - Optimal Kalman Gain: $\mathbf{K_k = P_{k|k-1} H_k^T (H_k P_{k|k-1} H_k^T + R_k)^{-1}}$
  - $R_k = \begin{bmatrix} \mathbf{gain\_Rx} & 0 & 0 \\ 0 & \mathbf{gain\_Ry} & 0 \\ 0 & 0 & \boldsymbol{gain\_R\theta} \end{bmatrix}$
  - Update State estimate: $\mathbf{\hat{X}_{k|k} = \hat{X}_{k|k-1} + K_k \tilde{Y}_k}$
  - Update Covariance State estimate:$\mathbf{P_{k|k} = (I - K_k H_k) P_{k|k-1}}$

9 Linear Quadratic Regulator(LQR)

9.1 What is LQR?

LQR is a method that calculates the optimal feedback gain K. The feedback gain can determined by tuning of Q and R. The feedback gain is used to control the system in control signal form.

9.2 LQR Algorithm

- An LQR control system generates the control law using four matrices:
  - A Matrix:physical dynamics
  - B Matrix:control dynamics
  - Q Matrix:state cost
  - R Matrix:control cost
- Setting up the optimization problem
  - Positive Semidefinite: $\mathbf{X^TQX >= 0}$
  - Positive definite: $\mathbf{U^TRU > 0}$
  - If Q "is bigger" than R $\Rightarrow$ fast regulation of X$\Rightarrow$0,U is Large
  - If R "is bigger" than Q $\Rightarrow$ slow regulation of X$\Rightarrow$0,U is Small
- Solving the optimization problem
  We are using one of the optimal control method which is Linear Quadratic Regulator.
  Minimize the cost:
  $$\boldsymbol{J = \frac{1}{2}\sum_{k=0}^{N}(X_{k+1}^T Q X_{k+1} + U_k^T R U_k)}$$
  or
  $$\boldsymbol{J = \int_0^\infty (X^T Q X + U^T R U) dt}$$
  Where: X is State, U is input control.
- Design Controller
  - The Optimal feedback gain:
    $$\mathbf{K = R^{-1}B^TP}$$
  - Algebraic Riccati Equation:discrete time algebraic Riccati equation (DARE):
    $$\mathbf{P = A^TPA - A^TPB(R + B^TPB)^{-1}B^TPA + Q}$$
  - The optimal input:
    $$\mathbf{U_t = -KX_t}$$

10 Nonlinear Model Predicted Control (NMPC)

10.1 What is MPC and NMPC?

Model Predictive Control is one of the advanced technique control process that can obtained an optimal solution over a period with finite and infinite horizon. Model Predictive Control(MPC) or Receding Horizontal Control (RHC) is also an original framework of feedback control of the nonlinear dynamic system.
Nonlinear Model Predicted Control(NMPC) is a variant of model predictive control that is characterized by the use of nonlinear system models in the prediction.

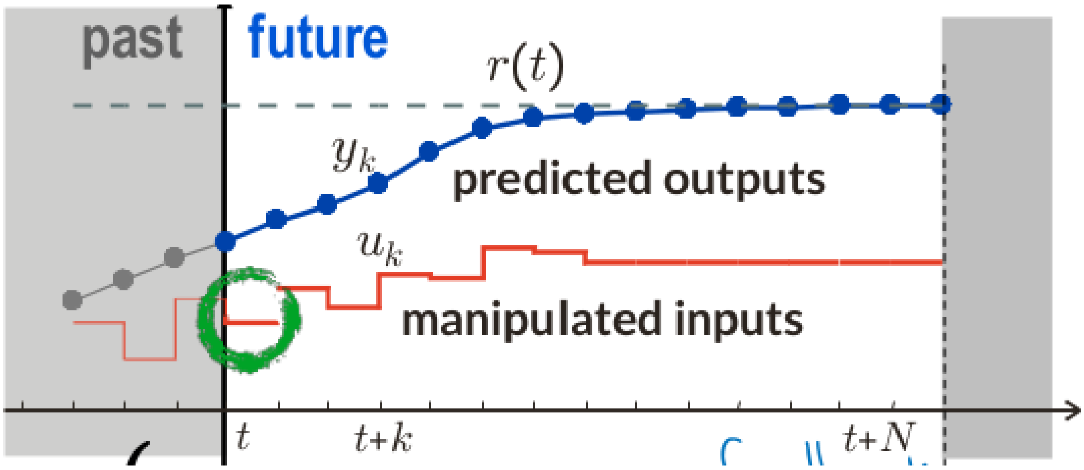

Figure 28: Predicted Output

10.2 NMPC Algorithm on Trajectory Tracking

- **Linear prediction model:**
  _State Space model: $\mathbf{X_{k+1}} = \mathbf{AX_k} + \mathbf{BU_k}$
  _Observation model: $\mathbf{Y_k} = \mathbf{CX_k}$

- **The optimal control problem (quadratic performance index):**
  _State error: $\mathbf{E_k} = \mathbf{X_k} - \mathbf{Xref_k}$
  _Input error: $\boldsymbol{\Delta U = U_k - U_{k-1}}$

$$\min_{z} \boldsymbol{E_N^T P E_N + \sum_{k=0}^{N-1} E_k^T Q E_k + \Delta U_K^T R \Delta U}$$

subject to

$$\mathbf{X_{min}} \leq \mathbf{X_k} \leq \mathbf{X_{max}}, K = [0, N]$$

$$\mathbf{U_{min}} \leq \mathbf{U_k} \leq \mathbf{U_{max}}, K = [0, N-1]$$

Where: $R = R^T > 0$; $Q = Q^T \geq 0$; $P = P^T \geq 0$; z=$\begin{bmatrix} u_0 & u_1 & \ldots & u_N - 1 \end{bmatrix}$
_Xref : the reference state of the system or desired input state.
_N : number prediction horizon.

- **For update state has two methode:**

$$\dot{\mathbf{X}} = \mathbf{f}(\mathbf{X}, \mathbf{U}(\mathbf{t}))$$

$$\begin{bmatrix} Vx \\ Vy \\ \omega z \end{bmatrix} = \frac{r \times dt}{4} \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & -1 \\ -\frac{1}{(l_x+l_y)} & \frac{1}{(l_x+l_y)} & -\frac{1}{(l_x+l_y)} & \frac{1}{(l_x+l_y)} \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix}$$

_ **Euler Discretization**

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} + \Delta T \begin{bmatrix} Vx_k \\ Vy_k \\ \omega z_k \end{bmatrix}$$

_ **Runge-Kutta 4th (RK4)**

$$k_1 = f(X_k, U_k)$$
$$k_2 = f(X_k + \frac{\Delta T}{2} k_1, U_k)$$
$$k_3 = f(X_k + \frac{\Delta T}{2} k_2, U_k)$$
$$k_4 = f(X_k + \Delta T k_3, U_k)$$
$$X_{k+1} = X_k + \frac{\Delta T}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

- **Nonlinear Program Problem(NLP)**:A standard problem formulation in numerical optimization.

  Problem decision variables :
  _ **Single shooting:** $\omega = [u_0, ..., u_{N-1}]$
  _ **Multiple shooting:** $\omega = [x_0, ..., x_N, u_0, ..., u_{N-1}]$

$$\min_{\omega} \Phi(F(\omega, X_N), \omega)$$

subject to

$$g_1(F(\omega, X_N), \omega) \leq 0 \qquad \text{Inequality constraints}$$

$$g_2(F(\omega, X_N), \omega) = \begin{bmatrix} \bar{X}_0 - X_0 \\ f(X_0, U_0) - X_1 \\ \vdots \\ f(X_{N-1}, U_{N-1}) - X_N \end{bmatrix} = 0 \quad \text{Equality constraints}$$

11 Testing

11.1 Testing Position Control in STM32 with PID

In the testing I have board such as:

- **Main Board(STM32F407VGT6):** Use for Control PID Position , Communication (NRF(SPI),CAN) and Read Sensors(rotary_encoder, IMU) for feedback.
- **Two Driver_Board(STM32F103C8T6):** Use for control 4 DC_Motor , Communication CAN with main board.
- **Buck_Converter:** Step Down from (12V to 5V) to supply main board.

11.1.1 Block Diagram of system

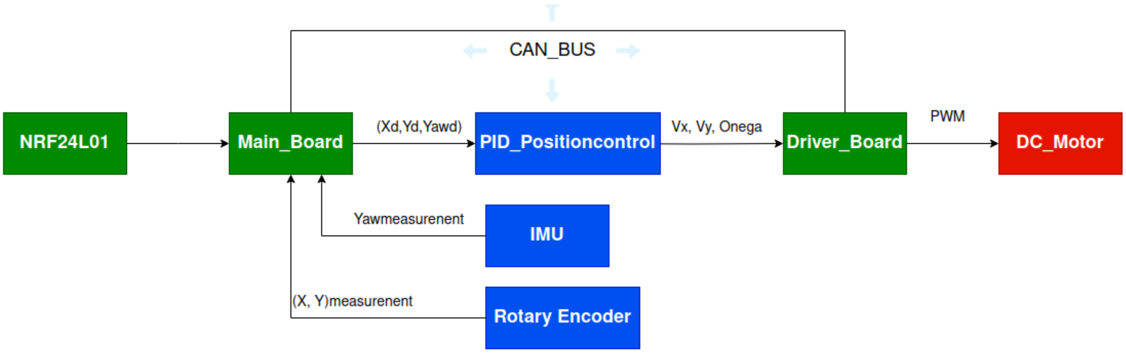

Figure 29: Block diagram of PID on STM32

11.1.2 Result

- **Code:**
  _Library PID controller: https://github.com/boyloy21/PID_Controller
  _Control on Board STM32: https://github.com/boyloy21/CAN_BUS/tree/main/can_bus_stm32
- **Picture:**

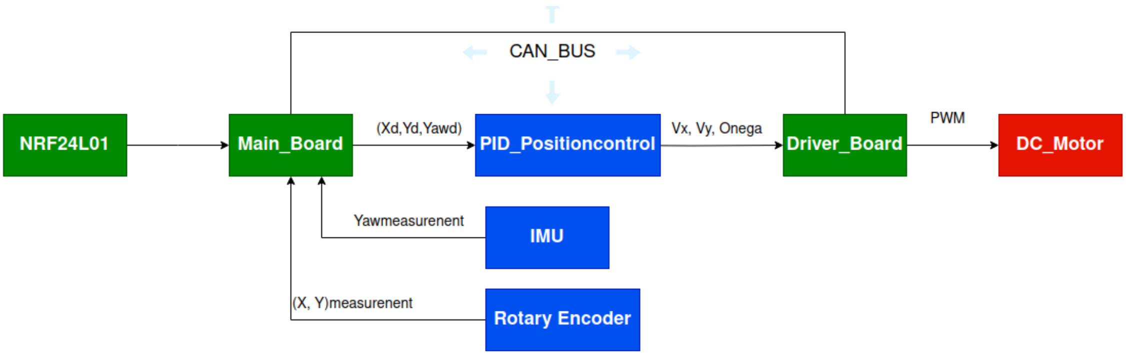

Figure 30: Block diagram of PID on STM32

- **Problem: DONE**

11.2 Testing Position Control in ROS2

In the testing has Component such as:

- **MINI PC:** Use for operating ROS2 .
- **Two Driver_Board(STM32F103C8T6):** Use for control 4 DC_Motor , and feedback speed from encoder motor to (Mini_pc or ROS) to calculate position.
- **PS4:** use for control (position and speed) and set(manual or auto).

11.2.1 Block Diagram of system

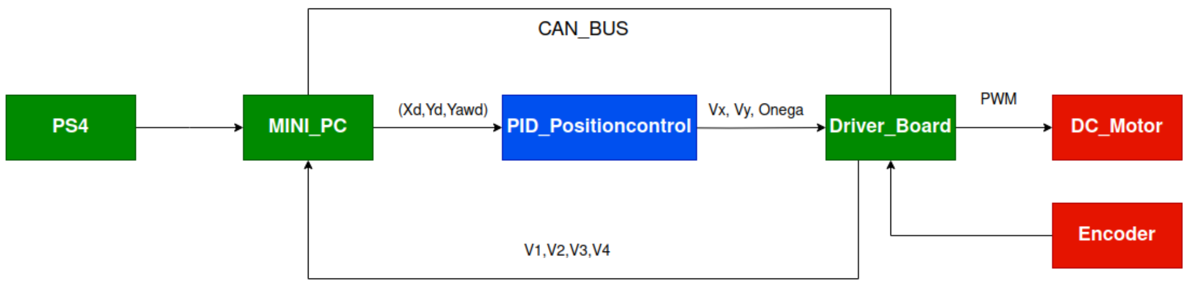

Figure 31: Block diagram of PID on ROS2

11.2.2 Result

- **Code**:
  _CAN ROS: https://github.com/boyloy21/CAN_BUS/blob/main/can_control_Motor.py
  _Control Position in Real for run: https://github.com/boyloy21/PID_Controller/blob/main/mecanum_pidV1.py
  _PID_Simulation: https://github.com/boyloy21/PID_Controller/blob/main/pid_simulation.py
- **Picture:**

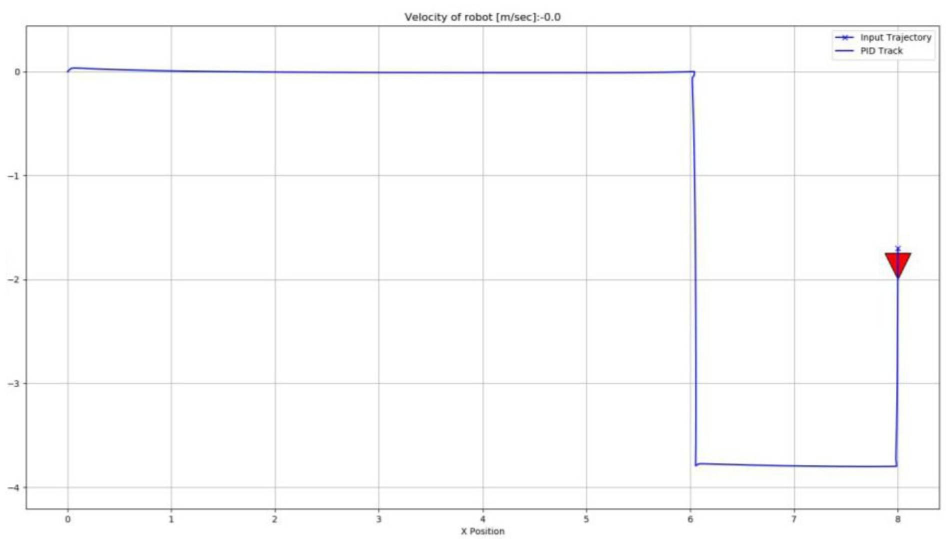

Figure 32: Result of PID in ROS2 on Position point to point

- **Problem: DONE**

11.3 Simulation PID with EKF in Python

11.3.1 Block Diagram of system

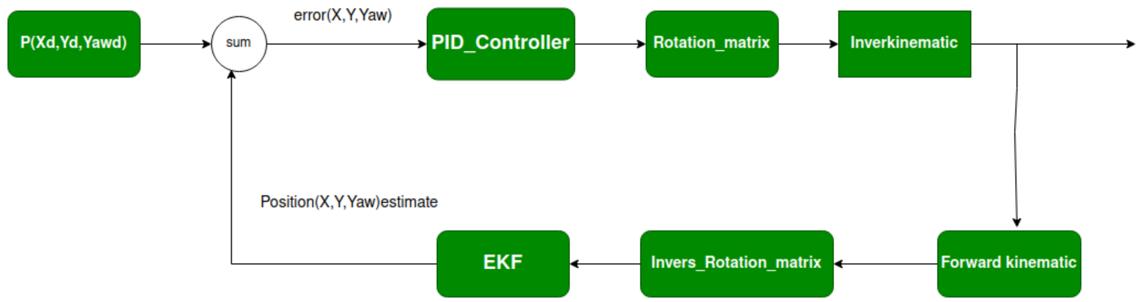

Figure 33: Block diagram of EKF with PID on Trajectory

11.3.2 Result

- **Code:** https://github.com/boyloy21/EKF
- **Picture:**

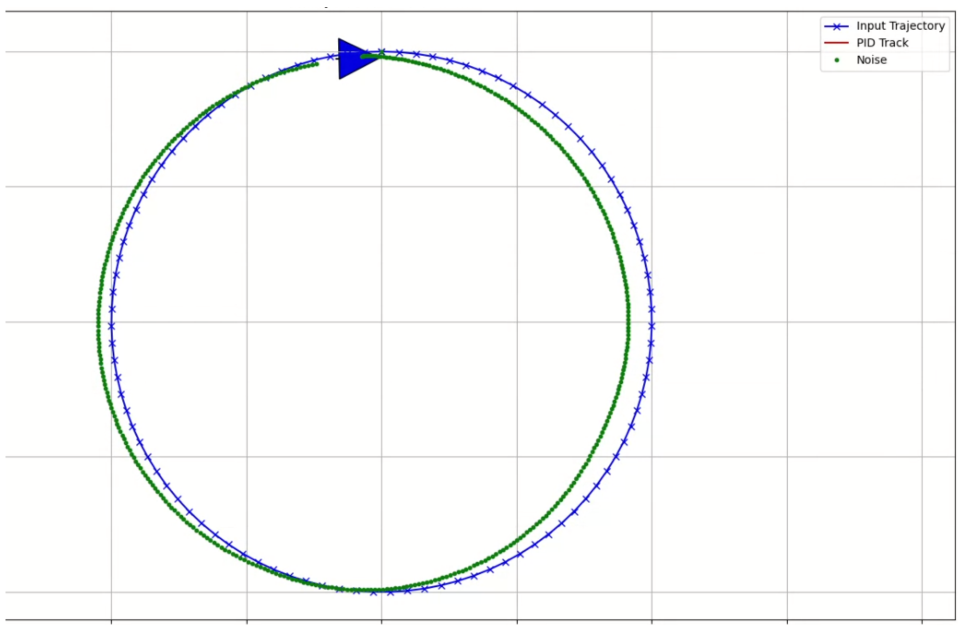

Figure 34: Result PID with EKF on trajectory circular

- **Problem:** I can not write C program for use in STM32 but I have tried it 2 or 3 times. But I can make it out at any time.

11.4 Simulation LQR in Python

11.4.1 Block Diagram

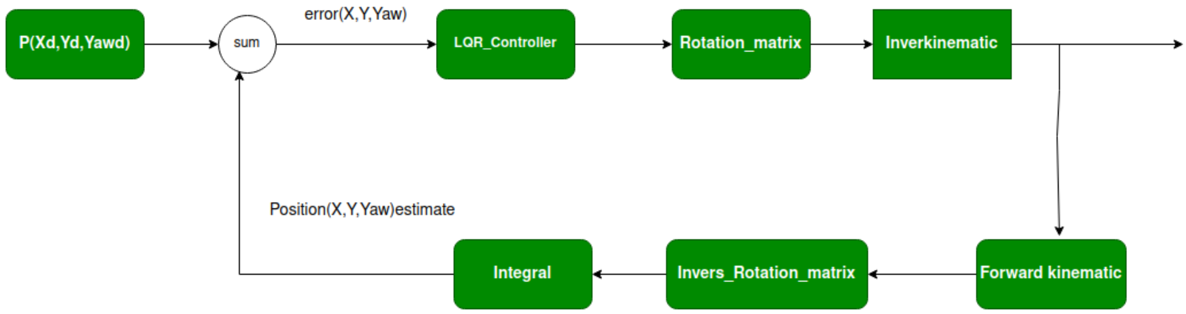

Figure 35: Block diagram of LQR on Trajectory

11.4.2 Result

- **Code:** https://github.com/boyloy21/LQR_Controller
- **Picture:**

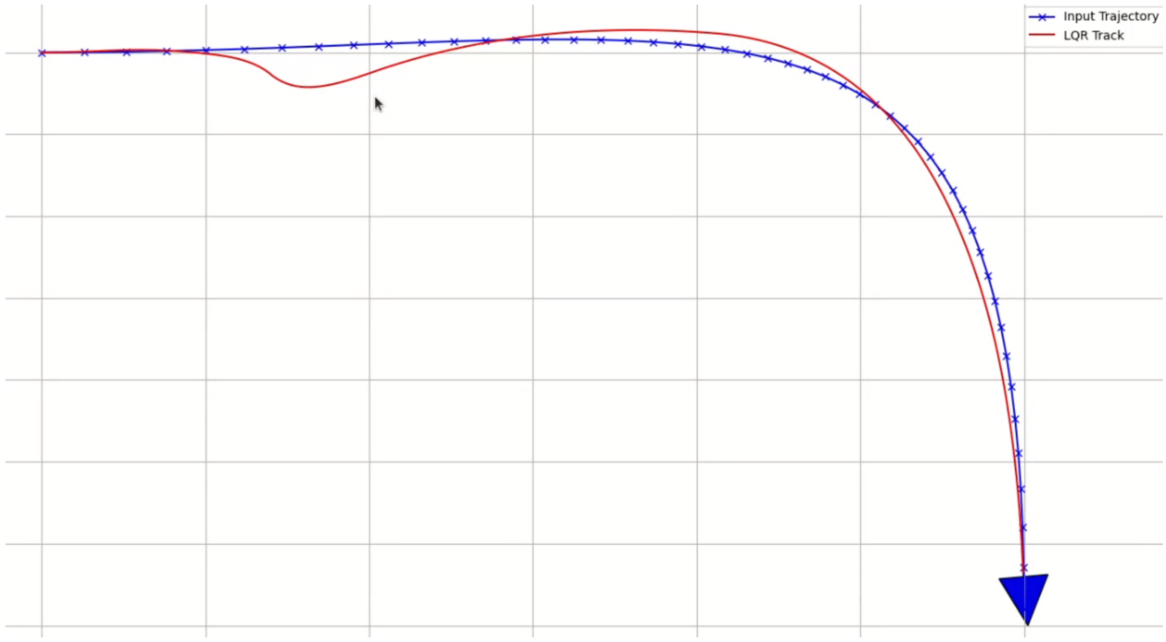

Figure 36: Result LQR on trajectory Bezier path

- **Problem:** difficult for tuning Q and R to robot follow on trajectory. But I have two reasons for my problem :
  - My code if wrong because when I run their code is good.
  - I do not yet understand how to use it.

11.5 Simulation NMPC in ROS2

In the testing I use PS4 to control and input goal to robot .

11.5.1 Block Diagram

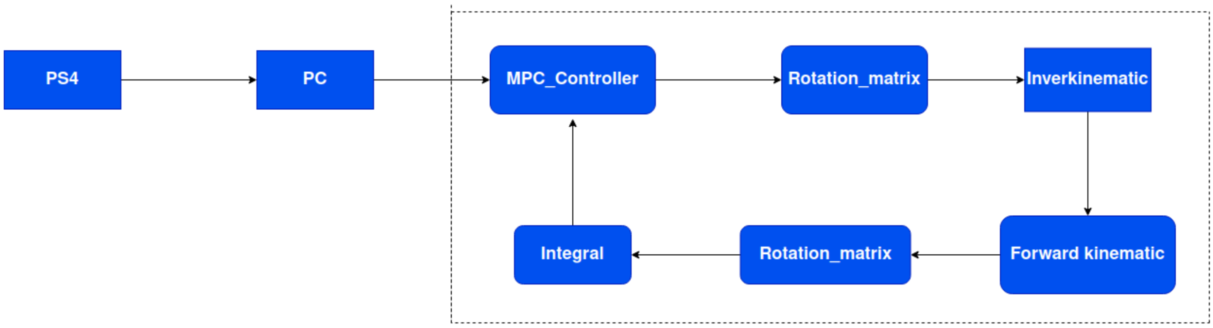

Figure 37: Block Diagram of MPC controller

11.5.2 Result

- **Code:** https://github.com/boyloy21/MPC_Controller
- **Picture:**

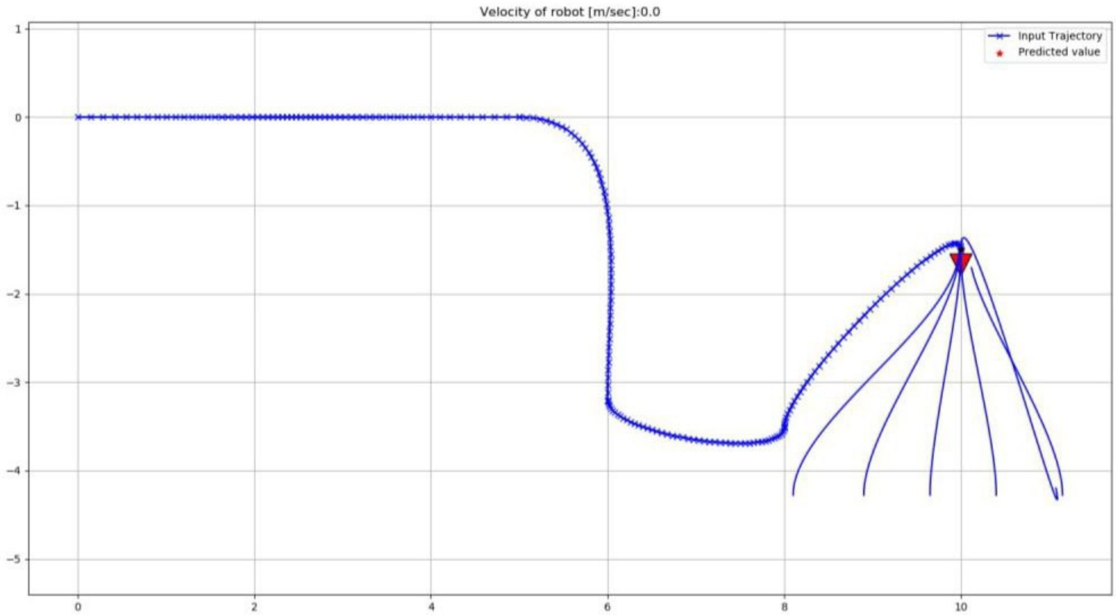

Figure 38: Result MPC on trajectory Game field ROBOCON 2024(from area1 to area3 and take ball to 5 silo)

- **Problem:** 70% of result, I will develop further on real robot to end.

12 Conclusion

In conclusion, this project has demonstrated the successful integration of mecanum wheels with a Model Predictive Control (MPC) strategy to enhance the agility and maneuverability of a mobile robot. By combining the unique capabilities of mecanum wheels for omnidirectional movement with the predictive control capabilities of MPC, we have achieved significant advancements in robot navigation performance.