Final Project:
Driving an RC Car with Automatic Parking System

21300399 Hyeongseok Song
21600372 Yoonkyoung Song

I. Introduction

In this project, we want to control RC cars trekking lines and auto-parking. Through the embedded controller class, we learned how to use STM32 and set the value by touching the register directly, and we used it to implement the movement using multiple registers in combination.

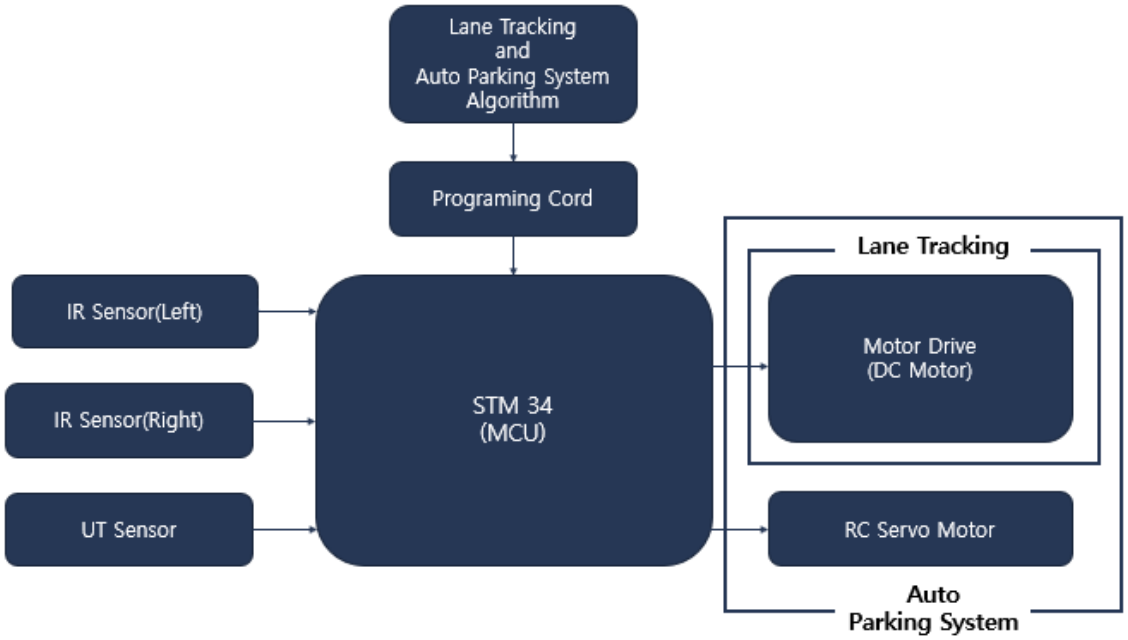


Figure 1. Overview of project

II. Device

A. Hardware: Preparation

Mode	Hardware	Quantity
RC Car	DC motor	2
	Motor Driver	1
	Bluetooth	1
Lane Tracking	IR sensor	2
Automatic Parking System	Ultrasonic sensor	1
	RC servo motor	1

Table 1. Hardware Prat List

B. Software: Pin Setting of STM32F411

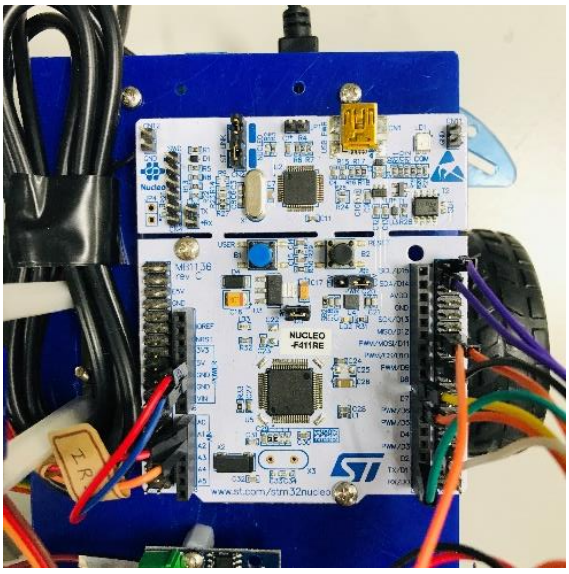
This table represents our using GPIO pin, timer, and channel in STM32F411.

Mode	Hardware	GPIO Pin		TIM	Channel
RC Car	Trigger	-	-	TIM9	1
	Motor Driver	PWM1	PA 8	TIM1	1
		PWM2	PA 9		2
		PWM3	PA 10		3
		PWM4	PA 11		4
	Bluetooth	Tx	PB 6	TIM4	1
		Rx	PB 3	TIM2	3
Lane Tracking	IR sensor	Left	PA 0	TIM2	1
		Right	PA 1	TIM2	2
		ADC trigger	-	TIM5	1
Automatic Parking System	Ultrasonic sensor	trigger	PC8	TIM4	3
		echo	PB8		4
	RC servo motor	-	PA 6	TIM3	1

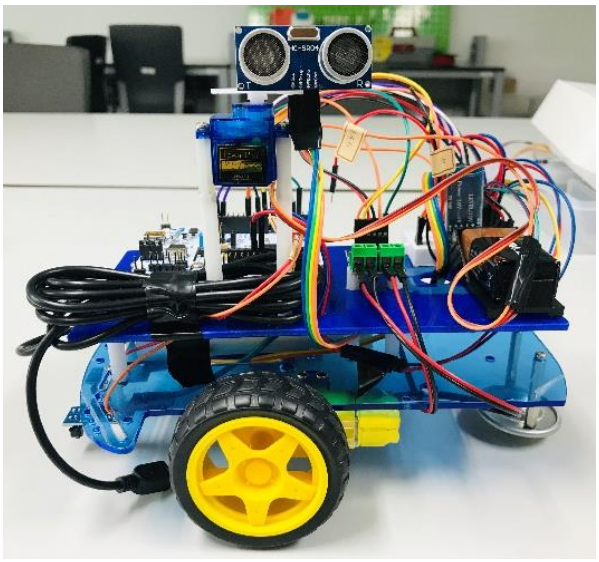
Table 2. Pin Setting of STM32F411

C. Actual Connection

Figure 2 (a) is representing connection pin and hardware. And (b) is RC car that our uses in experiment.



(a) STM32F411



(b) RC Car

Figure 2. Hardware Appearance

III. Method

A. Lane Tracking

- **Using two IR sensor**

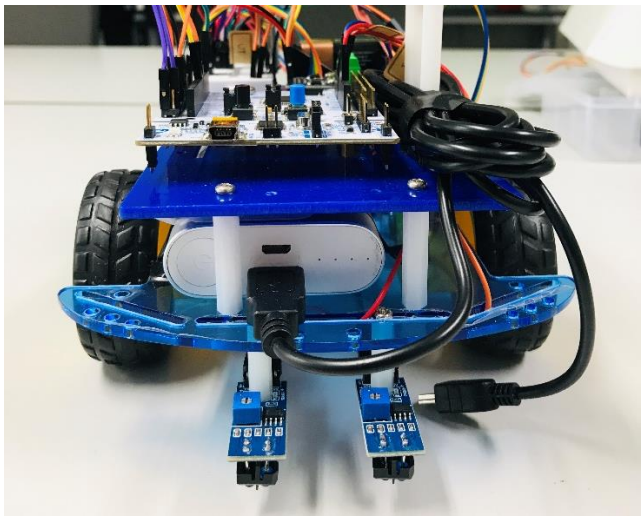


Figure 3. Position of IR Sensor

Two IR sensors are attached to the front of the RC car to read the values of the sensor and send to enter the motor driver. Depending on the sensor value, change the RPM that the motor outputs to perform the lane tracking

- **Lane tracking method**

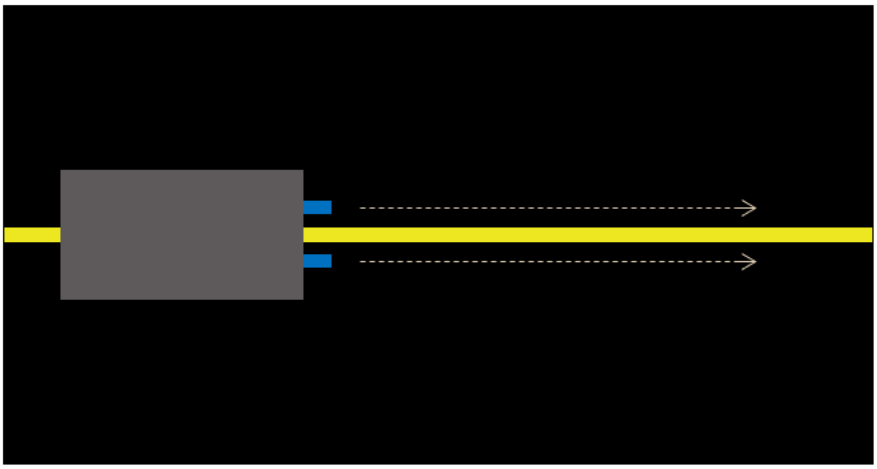
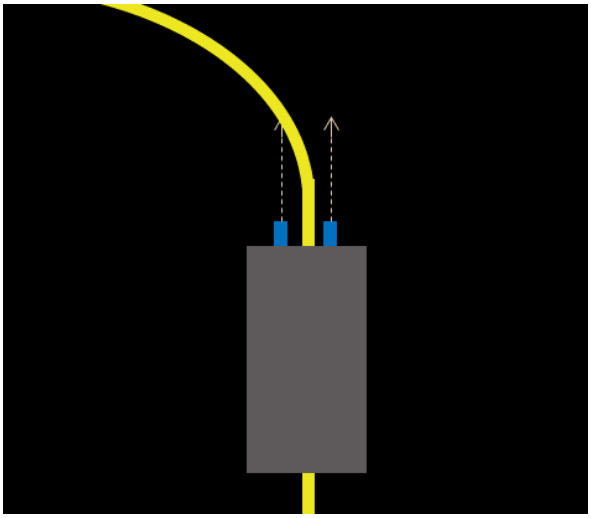
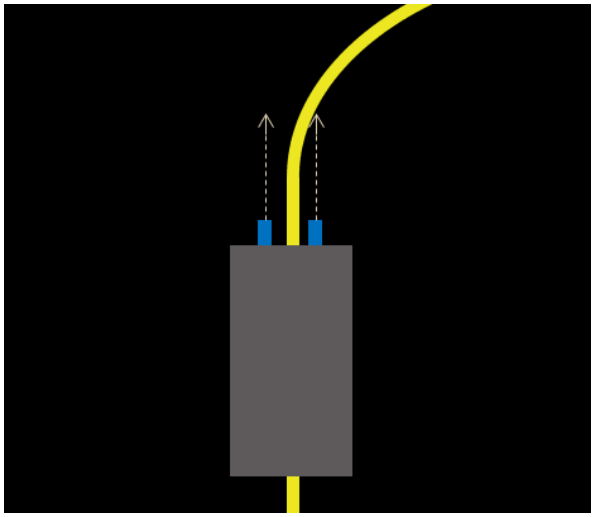


Figure 4. Basic Principle of Lane Tracking

The basic principle of lane tracking is to read brightness using IR sensors. The IR sensor has a smaller sensor value when it is yellow compared to black. Therefore, if the IR sensor becomes smaller than the threshold by determining the threshold from the sensor value, rotate in the opposite direction of the reading sensor and follow. An experimental measurement shows that the IR sensor reads 1800 to 2000 on average when viewing a black background, and 200 to 300 when viewing a yellow line. Therefore, the threshold was set to 1500 to recognize yellow.



(a) Turn Left



(b) Turn Right

Figure 5. Principle of Turning RC Car

If the RC car must be rotated to the right or to the left, like figure 5, the value of one of the IR sensors will be less than threshold. The direction of RC cars is controlled by different PWM applied to the motor driver. If it is necessary to rotate to the left, the IR sensor value on the left will be reduced by 1500 and the PWM on the right wheel is given high to rotate to the left. The PWM's duty on the left and right motors is 0.48 and 0.8 respectively. Conversely, if you need to rotate to the right, the IR sensor on the right will be smaller than the threshold, so give the left PWM greater than the right so that it can rotate to the right. The PWM given to the left and right motors is 0.65 and 0.55, respectively.

- **Results of implementation and directions for development**

  The biggest problem with lane tracking was two. The problem of DC motor torque and battery performance. This issue is dealt with in detail in the analysis later. These two problems result in the same PWM being applied to the motor and not producing the same output. To solve this problem, the PWM applied to the DC motor was experimentally tuned for calibration. Nevertheless, it is difficult to make a straight line accurately by empirical tuning. Therefore, it was decided that changing direction without going straight would affect speed, i.e. narrowing the gap between IR sensors on the body to make the trek more smooth and accurate.

B. Automatic Parking System

- Using sensor – UT sensor, RC servo motor

- Automatic Parking System method

IV. Results & Analysis

DC모터의 배터리 종류나 상태가 PWM 차이가 많이 난다. 이걸 해결하기 위해서 적절한 값을 찾아야 했다

오른쪽으로 도는 거랑 0.6/0.5 ////// 왼쪽으로 돌 때는 0.5/0.8 -> 이 때 좌우 회전이 비슷하게 출력

개인적으로는 주차할 때 모터의 출력을 저희가 측정해서 수식화 할 수 있었으면 초음파 센서의 거리에 따라서 회전하는 정도를 조절할 수 있지 않았을까 생각해 봤어요 그러면 저희 로직이 좀 더 안정적으로 작용할 것 같아요

V. Conclusion

Through this project, STM32F411 was used to design RC car with the functions of lane tracking and auto parking system. Starting with setting the most basic clock of the embedded system, STM32 functions were used: reading and writing input and output using GPIO, reading values of IR sensors using digital conversion of analog values using ADC (analog to digital converter), and creating a periodic trigger using the timer, reading the ultrasonic sensor with input capture, controlling the motor through the output of PWM and controlling of the duty of PWM, etc.

The algorithms of lane tracking and auto parking system were designed, understanding the basic operating principles of automobiles and analyzing the impact of hardware on the results. Even if the algorithms of the software were perfect, there were difficulties in producing the desired results by the limitations of the hardware, and rather, the limitations of the hardware were supplemented by modifying the code of the software. Thus, not only was the understanding of the algorithms of the underlying systems, but also the corresponding resolution of the interaction between hardware and software.

Appendix

1. Pin Configuration of NUCLE-F411RE

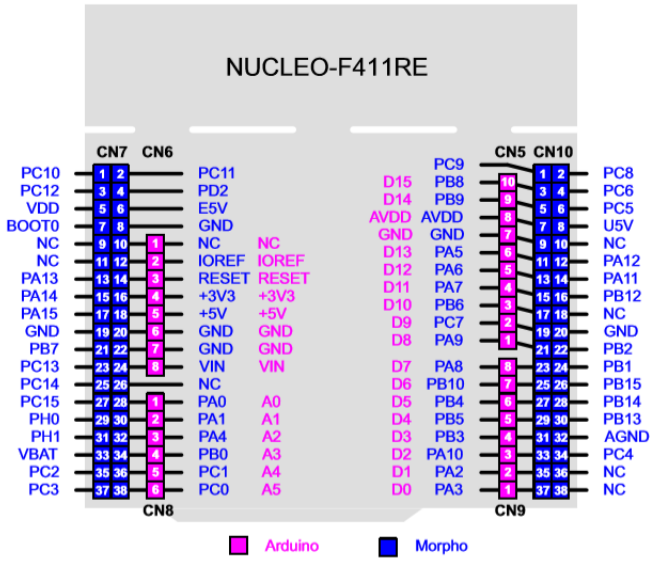


Table 29. ST morpho connector on NUCLEO-F401RE, NUCLEO-F411RE, NUCLEO-F446RE

CN7 odd pins		CN7 even pins		CN10 odd pins		CN10 even pins	
Pin	Name	Name	Pin	Pin	Name	Name	Pin
1	PC10	PC11	2	1	PC9	PC8	2
3	PC12	PD2	4	3	PB8	PC6	4
5	VDD	E5V	6	5	PB9	PC5	6
7	BOOT0(1)	GND	8	7	AVDD	U5V(2)	8
9	-	-	10	9	GND	-	10
11	-	IOREF	12	11	PA5	PA12	12
13	PA13(3)	RESET	14	13	PA6	PA11	14
15	PA14(3)	+3.3V	16	15	PA7	PB12	16
17	PA15	+5V	18	17	PB6	-	18
19	GND	GND	20	19	PC7	GND	20
21	PB7	GND	22	21	PA9	PB2	22
23	PC13	VIN	24	23	PA8	PB1	24
25	PC14	-	26	25	PB10	PB15	26
27	PC15	PA0	28	27	PB4	PB14	28
29	PH0	PA1	30	29	PB5	PB13	30
31	PH1	PA4	32	31	PB3	AGND	32
33	VBAT	PB0	34	33	PA10	PC4	34
35	PC2	PC1 or PB9(4)	36	35	PA2	-	36
37	PC3	PC0 or PB8(4)	38	37	PA3	-	38

1. Default state of BOOT0 is 0. It can be set to 1 when a jumper is on pin5-7 of CN7.Two unused jumpers are available on CN11 and CN12 (bottom side of the board).
2. U5V is 5 V power from ST-LINK/V2-1 USB connector and it rises before +5V.
3. PA13 and PA14 share with SWD signals connected to ST-LINK/V2-1, it is not recommend to use them as IO pins if ST-LINK part is not cut.
4. Refer to *Table 10: Solder bridges* for details.

2. Keil uVision Cord

- **main**

main.c

```
/*-----------------------------------------------------

            2019-2 Embeded Controller 1

                  Final Project:
    Driving an RC Car with Automatic Parking System

              21300399 Hyeongseok Song
              21600372 Yoonkyoung Song

-----------------------------------------------------------*/

#include "myInclude.h"
#include "myProject.h"

// IR Sensor parameter//
uint32_t IR1 = 0;
uint32_t IR2 = 0;
int flag_adc = 0;
uint32_t thresh = 1500;  // black 200~~500 whilte > 1000


// UT Sensor parameter//
int UT1 = 0;
float timeInterval = 0;
int overf = 0;
float time1 = 0;
float time2 = 0;

// DC Motor parameter//
extern PWM_t pwm2, pwm3, pwm4, pwm5;


// Servo Motor parameter//
extern PWM_t pwm1;

//BT parameter//
uint8_t BT_Data = 1;
uint8_t PC_Data = 0;

//Systme//
int CNT=0;
int CNT_1=0;
int flag_p =0;


int main(void){

   RCC_PLL_init();
   SysTick_init();

   UART2_init(9600, POL);
   BT_init(9600);
   printf("Hello Nucleo\r\n");

   Set_TIM9();
   Set_TIM9();
   Set_ADC();
   Set_Inputcapture();
   Set_Servomotor();
   Set_DCmotor();
   while(1){;}

   }

 void TIM1_BRK_TIM9_IRQHandler(void){  //for debug
  if(TIM9->SR & TIM_SR_UIF){

   if(timeInterval<=0) UT1=30;
   else UT1 = (float) timeInterval/58;

   if(CNT==1000){

     CNT =0;
   }
   if(BT_Data=='1'&&flag_p==0){
     PWM_duty(&pwm1, (0.06));
     if(IR2<thresh){
       Front(0.65,0.55);
     }
     else if(IR1<thresh){
       Front(0.48,0.8);
     }
     else{
       Front(0.85,0.9);
       if(UT1<17){      // wall detecting
         Stop();
         flag_p=1;
       }
     }
   }
   else{
     Stop();
   }

   if(flag_p==1){
     PWM_duty(&pwm1, (0.125));
     Back(0.8,0.8);
     if(UT1>32) flag_p=2;
   }

   if(flag_p==2){
     PWM_duty(&pwm1, (0.125));
     Back(0.8,0.8);
     if(UT1<25) flag_p=3;
   }

   if(flag_p==3){
     PWM_duty(&pwm1, (0.06));
     Stop();
     flag_p=4;
   }
   if(flag_p==4) CNT_1++;

   if(CNT_1==200) BT_Data='p';

   if(flag_p==4 & BT_Data=='p'){
     Front(0.45,0.85);
     if(UT1<28) flag_p=5;
   }
   if(flag_p==5){
     Stop();
     flag_p=6;
   }

   if(flag_p==6){
     PWM_duty(&pwm1, (0.125));
     Back(0.7,0.7);
     if(UT1>40) flag_p=7;
   }

   if(flag_p==7){
     Stop();
     CNT_1=0;
     flag_p=8;
   }

   if(flag_p==8){ CNT_1++; }
   if(flag_p==8 & CNT_1==200) flag_p=9;
   if(flag_p==9){
     PWM_duty(&pwm1, (0.06));
     Front(0.9,0.9);
     if(UT1<27) flag_p=10;
   }

   if(flag_p==10){
     PWM_duty(&pwm1, (0.06));
     Front(1,0.5);
     if(UT1<14) flag_p=11;
   }

   if(flag_p==11){
     PWM_duty(&pwm1, (0.06));
     Back(0.8,0.8);
     if(UT1>20) flag_p=12;
   }

   if(flag_p==12){
     PWM_duty(&pwm1, (0.125));
     Stop();
    }

   CNT++;

  TIM9->SR &= ~1;               // clear update interrupt flag
  }
}
```

```c
void USART1_IRQHandler(){
    if(USART1->SR & USART_SR_RXNE){

        // Bluetooth
        BT_Data = BT_read();
        BT_write((uint8_t*) "BT sent : ",10);
        BT_write(&BT_Data,1);
        BT_write((uint8_t*) "\r\n",2);

        // PC
        printf("NUCLEO got : %c (from BT)\r\n", BT_Data);

    }
}
void TIM4_IRQHandler(void){
  if(TIM4->SR & TIM_SR_UIF){
     overf = overf + 1;

     TIM4->SR &= ~1;
     }

  if((TIM4->SR & TIM_SR_CC3IF) != 0){
     time1 = TIM4->CCR3;

     TIM4->SR &= ~(1<<3);
     }
```

```c
  if((TIM4->SR & TIM_SR_CC4IF) != 0){
    time2 = TIM4->CCR4;
    //printf("Ut\r\n");
    overf = 0;
    TIM4->SR &= ~(1<<4);
    timeInterval = (overf * (TIM4->ARR+1) + time2 - time1);
  }
}
void ADC_IRQHandler(void){
  if((ADC1->SR & ADC_SR_OVR) == ADC_SR_OVR){
  ADC1->SR&=~ADC_SR_OVR;
  }

  if(ADC1->SR & ADC_SR_JEOC){       //after finishing sequence
    //printf("hi ADC\r\n");
    IR1 = ADC1->JDR1;   //read JDR1 value

    IR2 = ADC1->JDR2;   //read JDR2 value


  ADC1->SR &= ~(ADC_SR_JEOC);
 }
}
```

- **myADC**

myADC.c

```c
void JADC_init(GPIO_TypeDef *port,  int pin){
  uint32_t channel;

  channel =  ADC_pinmap(port,pin);

    printf("channel = %d\n",channel);
   RCC->APB2ENR &= ~(1<<8);
   RCC->APB2ENR |= 1<<8;          //ADC clock setting

   ADC ->CCR &=~(3<<16); //reset
   ADC ->CCR |=0<<16;    //PCLK2/2

   ADC1->CR2 &=~(1<<1);  //reset
   ADC1->CR2 |= 1<<1;    //continuouis conversion mode

   ADC1->CR2 &=~(1<<11); //reset
   ADC1->CR2 |= 0<<11;   //right allignment

   ADC1->CR1 &=~(3<<24);   //reset
   ADC1->CR1 |= 0<<24;     //12bit resolution

   ADC1->CR1 &= ~(1<<8); //reset
   ADC1->CR1 |= 1<<8;    //SCAN mode

   ADC1->JSQR &= ~(3<<20); //reset
   ADC1->JSQR |= 1<<20;      //1 conversion in the regular channel

   ADC1 -> JSQR &= ~(31<<10); // reset
   ADC1 -> JSQR |= 1<<10; // channel setting

   ADC1 -> JSQR &= ~(31<<15); // reset
   ADC1 -> JSQR |= channel<<15; // channel setting
```

```c
   if(channel<10){                    // 0~9 channel
     ADC1->SMPR2 &= ~(7<<channel*3);  // channel x samling time 84
     ADC1->SMPR2 |= 4<<channel*3;
   }

   else if(channel>9){                          // 10~18 channel
     ADC1->SMPR2 &= ~(7<<(channel%10)*3);  // channel x samling time 84
     ADC1->SMPR2 |= 4<<(channel%10)*3;
   }
   ADC1->SMPR2 &= ~(7<<1*3);  // channel 1 samling time 84
   ADC1->SMPR2 |= 4<<1*3;

   ADC1->CR2 &=~(0xF<<16);
   ADC1->CR2 |= 11<<16;

   ADC1->CR2 &=~(3<<20);
   ADC1->CR2 |= 1<<20;

   ADC1->CR1 &= ~(1<<7);    // JEOC interrupt enable
   ADC1->CR1 |= (1<<7);

   ADC1->CR2 &= ~(1<<0);
   ADC1->CR2 |= 1<<0;
   ADC1->CR2 &= ~(1<<22);
   ADC1->CR2 |= 1<<22;

   NVIC_SetPriority(ADC_IRQn,1); //NVIC interrupt setting
   NVIC_EnableIRQ(ADC_IRQn);     //Enable NVIC
  }
```

- **myTIM**

myADC.c

```c
void TIM9_Init(float msec){

    RCC->APB2ENR |= RCC_APB2ENR_TIM9EN;

    TIM9->CCMR1    &= ~(7<<4);
    TIM9->CCMR1    |= 6<<4;      //  output compare pwm_1 mode

    TIM9->CCER |= 1;          // CC1 Capture enabled
    TIM9->CCER &= ~(4<<1) ;      // CC1 Capture rising edge
     TIM9->CCR1 = (10*msec)-1;


    TIM9->PSC = 8299 ;   // Timer counter clock: 84Mhz / PSC + 1 = 10Khz
    TIM9-> ARR = (10*msec)-1 ;     // Set auto reload register
    TIM9->DIER |= 1 ;

    TIM9->CR1  |= 1;              // Enable counter


}
```

- **myProject**

myProject.h

```c
#include "stm32f4xx.h"
#include "stm32f411xe.h"

#ifndef PROJECT_HEADER
#define PROJECT_HEADER

  void Set_TIM9();
  void Set_ADC();
  void Set_Inputcapture();
  void Set_Servomotor();
  void Set_DCmotor();

  void Front(float leftSpeed, float rightSpeed);
  void Back(float leftSpeed, float rightSpeed);
  void Stop();
  void Front_speed(float Speed1,float Speed2,float Speed3, float Speed4);

#endif
```

myProject.c

```c
#include "myInclude.h"

// DC Motor parameter//
PWM_t pwm2, pwm3, pwm4, pwm5;

// Servo Motor parameter//
PWM_t pwm1;

void Set_TIM9(){

  /*---------- TIM9 ---------------*/
  TIM9_Init(0.5);                    // 100Hz ,10msec
  NVIC_SetPriority(TIM1_BRK_TIM9_IRQn, 5);  // Set the priority of TIM9 interrupt request
  NVIC_EnableIRQ(TIM1_BRK_TIM9_IRQn);

}


void Set_ADC(){

  /*---------- ADC IR sensor (PA0) ---------------*/
  TIM5_Init(1);
  GPIO_init (GPIOA, 0, ANALOG); //GPIO
  GPIO_init (GPIOA, 1, ANALOG); //GPIO
  JADC_init(GPIOA,0);           //JADC_init for  GPIOA 0,1 continous coversion mode

}


void Set_Inputcapture(){

  /*---------- Trigger Generation(PC8) ----------*/

   GPIO_init(GPIOC, 8, ALTE);

   PWM_t   trig;
   PWM_init(&trig, GPIOC, 8);

   PWM_period_ms(&trig, 25);
   PWM_pulsewidth_ms(&trig, 0.01);

  /*---------- Echo Reception(PB8) -------------*/
   GPIO_init(GPIOB, 8, ALTE);
   TIME_period_us(4, 1);           // TIM4, 1us(1MHz) counter

   TIM_t   echo;
   CAP_init(&echo, GPIOB, 8);

   NVIC_SetPriority(TIM4_IRQn, 2); // Set the priority of TIM4 interrupt request
   NVIC_EnableIRQ(TIM4_IRQn);      // TIM4 interrupt request enable
   printf("\r\nhi UT");

}


void Set_Servomotor(){

  /*---------- RC servo motor       ----------*/

  GPIO_init(GPIOA, 6, ALTE);       // GPIOA pin6 set as ALTERNATE
  GPIO_ospeed(GPIOA, 6, HIGH);     // give speed as high
  GPIO_pupd(GPIOA, 6, NOPUPD);     // No pullup pulldown

  PWM_init(&pwm1, GPIOA, 6);       // PWM init as GPIOA pin6
  PWM_period_ms(&pwm1, 25);        // set period as 25ms

  PWM_duty(&pwm1, (0.125));        // degree 0.025 ~ 0.125

  //printf("\r\nhi RC servo");

}
void Set_DCmotor(){

    /*---------- DC motor (TIM1)       ----------*/

  GPIO_init(GPIOA, 8, ALTE);     // GPIOA pin8 set as ALTERNATE
  GPIO_ospeed(GPIOA, 8, HIGH);     // give speed as high
  GPIO_pupd(GPIOA, 8, NOPUPD);     // No pullup pulldown

  GPIO_init(GPIOA, 9, ALTE);     // GPIOA pin8 set as ALTERNATE
  GPIO_ospeed(GPIOA, 9, HIGH);     // give speed as high
  GPIO_pupd(GPIOA, 9, NOPUPD);     // No pullup pulldown

  GPIO_init(GPIOA, 10, ALTE);     // GPIOA pin8 set as ALTERNATE
  GPIO_ospeed(GPIOA, 10, HIGH);    // give speed as high
  GPIO_pupd(GPIOA, 10, NOPUPD);     // No pullup pulldown

  GPIO_init(GPIOA, 11, ALTE);     // GPIOA pin8 set as ALTERNATE
  GPIO_ospeed(GPIOA, 11, HIGH);    // give speed as high
  GPIO_pupd(GPIOA, 11, NOPUPD);     // No pullup pulldown

  PWM_init(&pwm2, GPIOA, 8);        // PWM init as GPIOA pin8
  PWM_period_ms(&pwm2, 1);         // set period as 1ms - 1kHz

  PWM_init(&pwm3, GPIOA, 9);        // PWM init as GPIOA pin8
  PWM_period_ms(&pwm3, 1);         // set period as 1ms - 1kHz

  PWM_init(&pwm4, GPIOA, 10);        // PWM init as GPIOA pin8
  PWM_period_ms(&pwm4, 1);         // set period as 1ms - 1kHz

  PWM_init(&pwm5, GPIOA, 11);        // PWM init as GPIOA pin8
  PWM_period_ms(&pwm5, 1);         // set period as 1ms - 1kHz

}
void Front(float leftSpeed, float rightSpeed){

  PWM_duty(&pwm2, leftSpeed);   // left
  PWM_duty(&pwm3, (0));
  PWM_duty(&pwm4, rightSpeed);   // right
  PWM_duty(&pwm5, (0));

}
void Front(float leftSpeed, float rightSpeed){

   PWM_duty(&pwm2, leftSpeed);   // left
   PWM_duty(&pwm3, (0));
   PWM_duty(&pwm4, rightSpeed);   // right
   PWM_duty(&pwm5, (0));

}
void Back(float leftSpeed, float rightSpeed){

  PWM_duty(&pwm2, 0);   // left
  PWM_duty(&pwm3, leftSpeed);
  PWM_duty(&pwm4, 0);   // right
  PWM_duty(&pwm5, rightSpeed);

}
void Stop(){

  PWM_duty(&pwm2, 0);   // left
  PWM_duty(&pwm3, 0);
  PWM_duty(&pwm4, 0);   // right
  PWM_duty(&pwm5, 0);

}
void Front_speed(float Speed1,float Speed2,float Speed3, float Speed4){

  PWM_duty(&pwm2, Speed1);   // left
  PWM_duty(&pwm3, Speed2);
  PWM_duty(&pwm4, Speed3);   // right
  PWM_duty(&pwm5, Speed4);

}
```