Embedded Controller Final Project : MG/MC Excavator

LAB: EC Design Problem

Date: 2023-12-19

Author/Partner: Gyeonheal An / HyeonHo Moon

Github: https://github.com/AnGyeonheal/Embedded_Control_GH

Demo Video: https://www.youtube.com/watch?v=uYmLbaKMJFY

I. Introduction

Overview

Excavation work is essential when constructing a building for several reasons. Firstly, it is crucial to prepare and shape the ground at the construction site before laying the building's foundation. This ensures the stability and durability of the building while securing proper support from the ground. Secondly, excavation work is necessary for installing underground facilities such as pipes, electrical lines, and sewage systems. These infrastructures need to be placed underground, requiring excavation to perform the necessary tasks. Lastly, on a construction site, excavation work is undertaken to shape and adjust the terrain around the building, creating a safe working environment and minimizing the impact on the surrounding area. Therefore, excavation work is a pivotal phase in building construction, ensuring a stable foundation and efficient installation of facilities.

Issue

The primary reasons for the prolonged duration of traditional excavation work are mainly attributed to its manual nature and a lack of precise information. In manual operations, operators control the excavator manually, imposing limitations on accurate terrain modeling and depth control. Reliance on visual judgment and experience often makes precise operations challenging, leading to the possibility of rework. The absence of precise location information makes it difficult to define work boundaries and operate efficiently, requiring efforts to address issues that may arise during operations. Due to these limitations, traditional excavation work tends to be time-consuming and may face challenges meeting the high precision and efficiency requirements of modern construction practices.

Improvement

Machine Guidance and Machine Control excavators offer various innovative technologies and advantages compared to traditional excavators. Here's an explanation of these differences and strengths:

1. **Automation and Precision:**
    - *Machine Guidance:* Utilizes GPS and sensor technologies to monitor terrain in real-time and provide accurate location information, allowing operators to perform precise tasks.
    - *Machine Control:* Automated systems control the excavator, minimizing operator intervention and ensuring consistent accuracy.
2. **Productivity Enhancement:**
    - *Machine Guidance:* Defines job boundaries and efficiently operates within specified areas, improving productivity and preventing unnecessary work.
    - *Machine Control:* Accurate depth and angle control yield consistent results, increasing productivity and minimizing rework.
3. **Data Collection and Analysis:**
    - *Machine Guidance:* Collects various data during operations and analyzes it to enhance job efficiency. Real-time monitoring of soil conditions and job performance is possible.
    - *Machine Control:* Utilizes collected data to optimize productivity and predict maintenance tasks, enhancing machine reliability.
4. **Safety Reinforcement:**
    - *Machine Guidance:* Provides accurate location information of the work area, preventing collisions and enhancing safety.
    - *Machine Control:* Precise depth and angle control improve safety, reducing the risk of accidents during operations.
5. **Efficient Resource Utilization:**

- *Machine Guidance and Machine Control:* Precise operations promote efficient resource utilization, optimizing fuel consumption for environmentally friendly operation.

Application Objective

The design challenge selected for the final project in this rap is the unmanned crane design. As future construction sites are expected to witness the operation of heavy machinery without human intervention, we aim to design and build a crane that can be controlled autonomously. In this crane project, the crane moves to a designated location using line tracing. Moreover, as people may be present in the construction site during autonomous operation, the crane is programmed to stop and sound an alarm (utilizing ultrasonic sensors and a buzzer) when a person crosses its path. Once safely past the individuals, the crane resumes movement and reaches the specified location, where it excavates the ground to a given depth (utilizing ultrasonic sensors). The control of the crane arm, which includes four DC motors and a stepper motor for body rotation, is facilitated through Bluetooth communication.

Requirements

Hardware

- **Driving part**

Item	Model/Description	Qty
MCU	NUCLEO -F411RE	1
Analog Sensor	IR reflective optical sensor(TCRT5000)	2
Digital Sensor	HC-SR04	1
	DC motor	2
Actuator	DC motor driver(L9110s)	1
Communication	USART1 - Bluetooth Module(HC-06)	1
Others	buzzer	1
	breadboard	1

- **Crane upper body control**

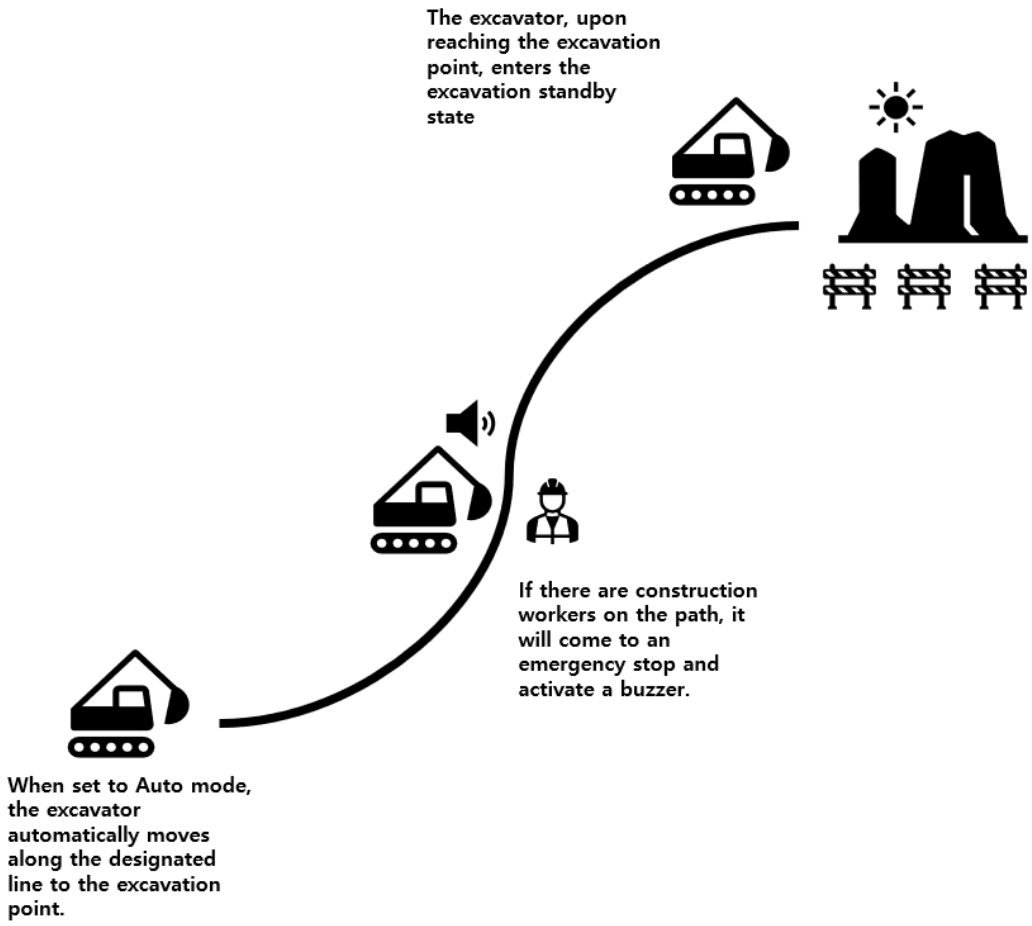
Item	Model/Description	Qty
MCU	NUCLEO -F411RE	2
Digital Sensor	DC motor	4
	3Stepper Motor 28BYJ-48	1
	HC-SR04	1
Actuator	DC motor driver(L9110s)	2
	Motor Driver ULN2003	1
Communication	USART1 - Bluetooth Module(HC-06)	1
Others	breadboard	1

Software

- Keil uVision IDE
- CMSIS
- EC_HAL
- Clion(with PlatformIO core plugin)
  Library: STM32Cube library package(Official), EC_HAL library
  Compiler: GNU Arm Embedded Toolchain
  Debugger: ST-Link
- SolidWorks(2022)

II. Problem

Problem Description

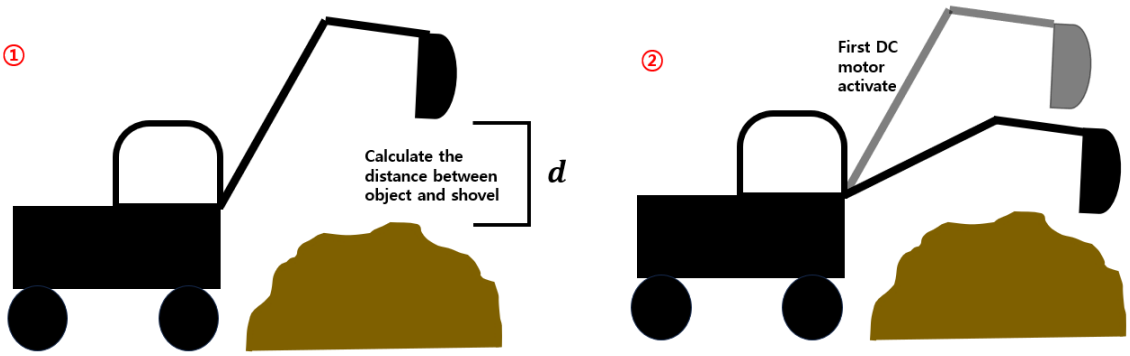


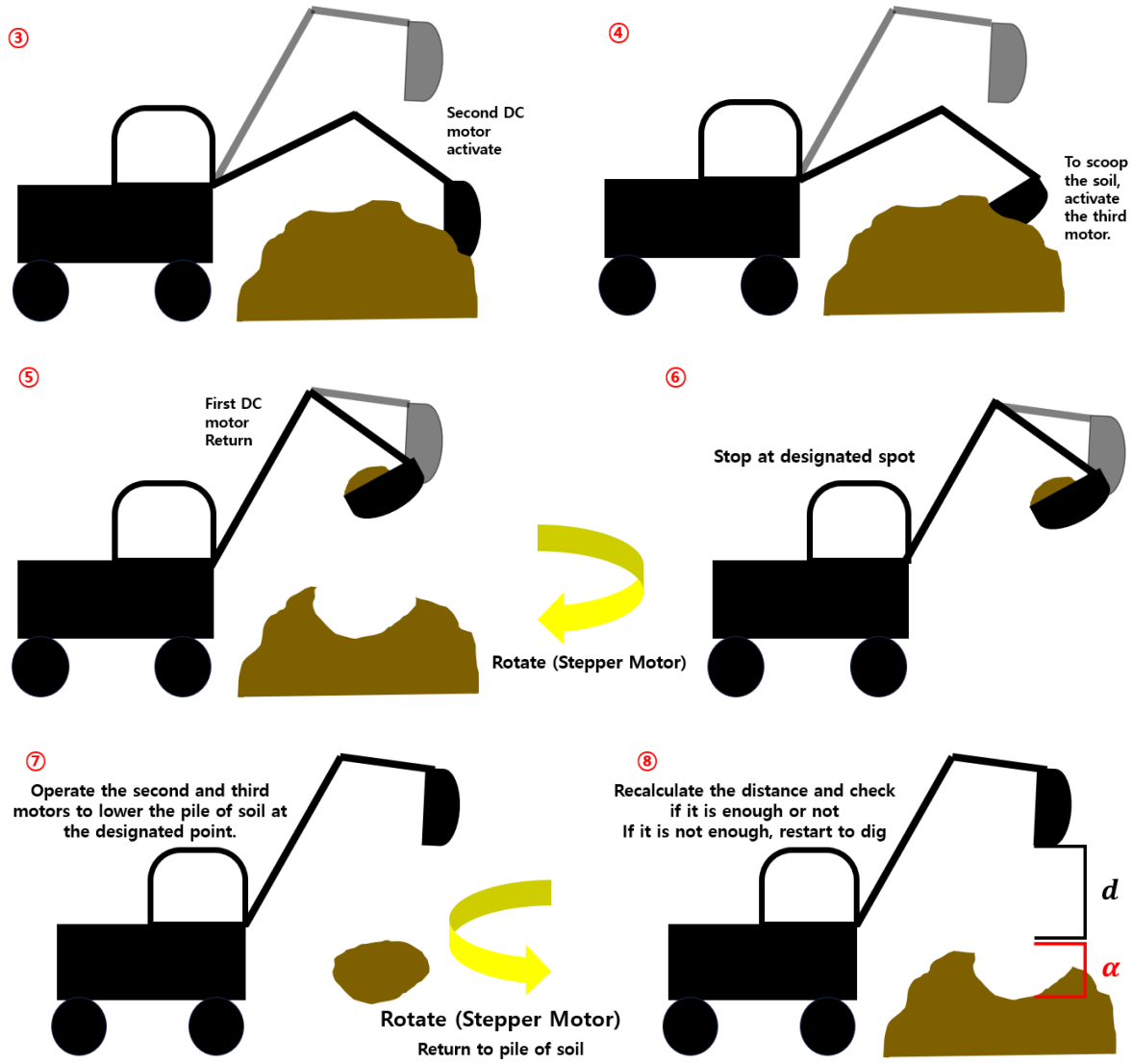
- **Driving Part - Mode**

MODE	Description	
Manual mode	Pressing the 'M' key transitions from the default mode to the manual mode. RESET mode ←→Manual mode	In manual mode, the STM board and computer are engaged in Bluetooth communication. Using the WSDA keys, you can move forward (W), backward (S), turn right (D), and turn left (A). Pressing 'E' brings the system to a stop.
Automatic mode	Pressing the 'L' key transitions from the default mode to the Automatic mode. RESET mode ←→Automatic mode	After creating a path using black electrical tape and placing the crane on the line, pressing the 'L' key activates the autonomous driving mode. The crane will follow the line and move to the specified location.

- **Driving Part - Functions**

Function	Sensor	Configuration	Comments
Worker detect (front)	Ultrasonic distance sensor, Buzzer	Sampling 10 usec, 50ms period Check object presence within 7cm Generates and Buzzer operation Person detect flag	Need to check for outlier measurements Person detect
Autonomous driving	IR sensor	IR values distinguish between black and white data.	Need to check for that the data values of the IR sensor do not overlap for distinguishing between white and black





- **Excavation Part - Mode**

  1-2. In the initial state, measure the distance between the soil pile and the shovel. Activate the first DC motor until the distance narrows by a certain amount (4cm). To record the descending distance, set a flag on TIM3, allowing the count value to increment by 1 and store the value.

  3-4. Operate the second motor to pull the soil with the shovel, and activate the third motor to scoop the soil with the shovel.

  5-6. Using the count value saved in the second step, raise the shovel by the previously descended distance. Apply additional weight to the count to account for the increased motor effort. Subsequently, rotate through the Stepper motor until reaching the position to lower the soil pile.

  7-8. Operate the second and third motors in reverse to lower the soil pile while simultaneously returning to the initial state, which is the first state. Afterward, measure the distance between the soil pile and the shovel, confirm if the desired amount has been excavated according to user requirements, and if not, repeat the process starting from the second step.

MODE	Description	Comments
Manual mode	Pressing the 'M' key to manual mode.	<u,j> First DC motor <i,k> Second DC motor <o,l> Third DC motor
Automatic mode	Pressing the 'A' key to the Automatic mode.	Automatically excavates the pile of soil and displays to the user how much it has excavated.

- **Excavation Part - Function**

Function	Sensor/TIM	Configuration	Comments
Distance measurement from the soil pile	Ultrasonic distance sensor	TIM4 _CH1, CH2	When the excavator is in the initial state, it begins measuring the distance, and when it reaches a certain distance, excavation operations commence based on the measured distance.
Use of a timer for restoring to the initial state.	TIM3	1msec	The excavator records the time taken to descend when narrowing the distance to the soil pile, and then uses this value during the return to the initial state for restoration.

III. MCU Configuration

Driving Part

Functions	Register	PORT_PIN	Configuration
System Clock	RCC		PLL 84MHz
Motor DIR	Digital Out	AF(PC_2, PC_3)	OUTPUT
DC Motor Speed	PWM2	AF(PA_0, PA_1)	Push-Pull, No pull-up, pull-down, Fast
Digital OUT: LD2	Digital Out	PA_5	Push-Pull, Fast
TIMER	TIM2	PA_0, PA_1	TIM2 Period: 1msec
	TIM3	PA6	Up-Counter, 50msec OC1M(Output Compare 1 Mode) : PWM mode 1 Master Mode Selection: (Trigger) OC1REF
	TIM4	PB_0, PB_1	50 msec PWM period 10 usec PWM pulse width
ADC	ADC	PB_0: ADC1_CH8 (1st channel) PB_1: ADC1_CH9 (2nd channel)	ADC Clock Prescaler /8 12-bit resolution, right alignment Continuous Conversion mode Scan mode: Two channels in regular group External Trigger (Timer3 Trigger) @ 50ms Trigger Detection on Rising Edge
PWM		PA6 (TIM4_CH2)	AF, Push-Pull, No Pull-up Pull-down, Fast, PWM period: 50msec pulse width: 10usec
Input Capture		PB6 (TIM4_CH1)	AF, No Pull-up Pull-down, Counter Clock : 0.1MHz (10us) TI4 -> IC1 (rising edge) TI4 -> IC2 (falling edge)
RS-232 USB cable(ST-LINK)	USART2		No Parity, 8-bit Data, 1-bit Stop bit 38400 baud-rate
Bluetooth	USART1	TXD: PA9 RXD: PA10	No Parity, 8-bit Data, 1-bit Stop bit 9600 baud-rate

Crane upper body control

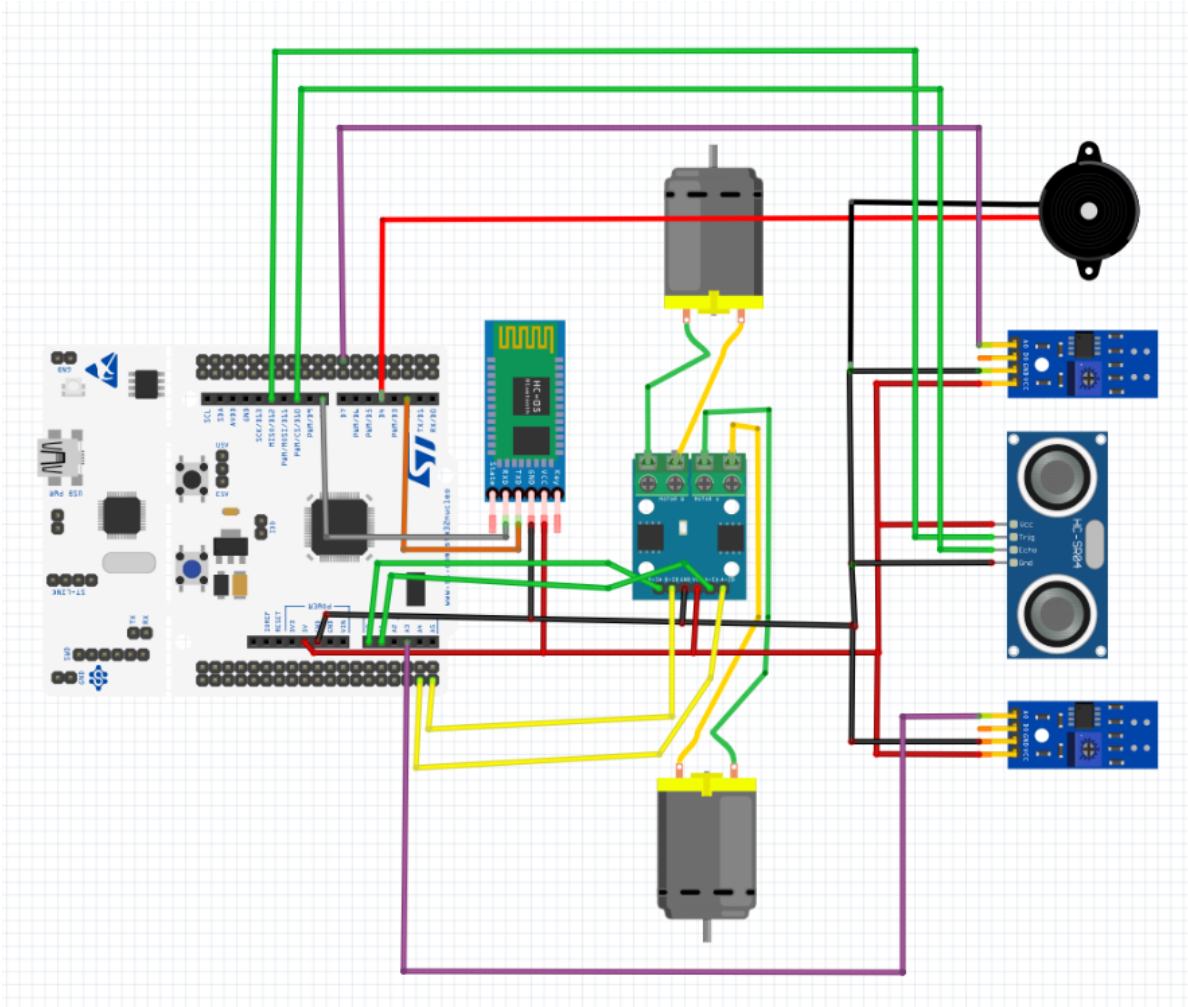
Functions	Register	PORT_PIN	Configuration
System Clock	RCC		PLL 84MHz
Motor DIR	Digital Out	AF(PC_3, PC_2, PC_4)	OUTPUT, Push-Pull
DC Motor Speed	PWM	AF(PA_0)	Push-Pull, pull-up, Fast
	PWM	AF(PA_1)	Push-Pull, pull-up, Fast
	PWM	AF(PB_10)	Push-Pull, pull-up, Fast
TIMER	TIM2	PA_0,PA_1,PB_10	TIM2 Period: 1msec
	TIM4	PB_0, PB_1	50 msec PWM period 10 usec PWM pulse width
Stepper	Digital Out	PA_8 PB_4 PA_5 PB_3	No pull-up Pull-down , Push-Pull, Fast
Input Capture	PWM	PA6 (TIM4_CH2)	AF, Push-Pull, No Pull-up Pull-down, Fast, PWM period: 50msec pulse width: 10usec
Input Capture	PWM	PB6 (TIM4_CH1)	AF, No Pull-up Pull-down, Counter Clock : 0.1MHz (10us) TI4 -> IC1 (rising edge) TI4 -> IC2 (falling edge)
RS-232 USB cable(ST-LINK)	USART2		No Parity, 8-bit Data, 1-bit Stop bit 38400 baud-rate
Bluetooth	USART1	TXD: PA9 RXD: PA10	No Parity, 8-bit Data, 1-bit Stop bit 9600 baud-rate

```
//State number
#define S0 0
#define S1 1
#define S2 2
#define S3 3
#define S4 4
#define S5 5
#define S6 6
#define S7 7
```

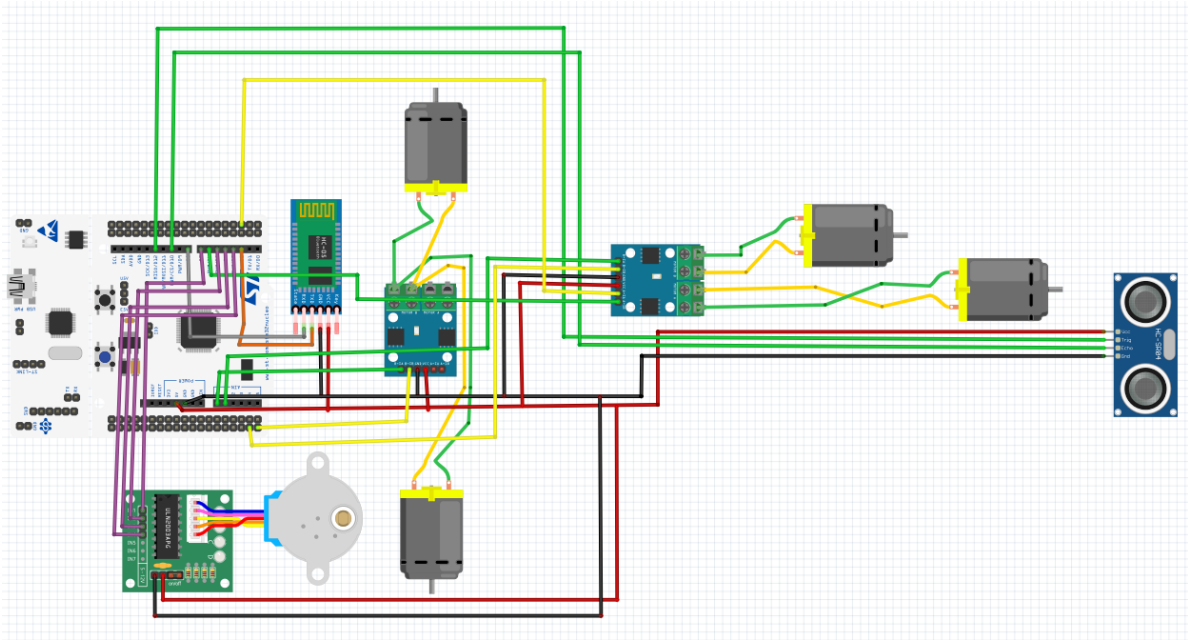
```
State_full_t FSM_full[4] = {
   {0b1100,{S1,S3}},
   {0b0110,{S2,S0}},
   {0b0011,{S3,S1}},
   {0b1001,{S0,S2}}
 };

 //HALF stepping sequence
typedef struct {
   uint8_t out;
     uint32_t next[2];
 } State_half_t;

State_half_t FSM_half[8] = {
   {0b1000,{S1,S7}},
   {0b0100,{S2,S0}},
   {0b0100,{S3,S1}},
   {0b0010,{S4,S2}},
   {0b0010,{S5,S3}},
   {0b0001,{S6,S4}},
   {0b0001,{S7,S5}},
   {0b1000,{S0,S6}}

 };
```

IV. Circuit Diagram

Driving Part

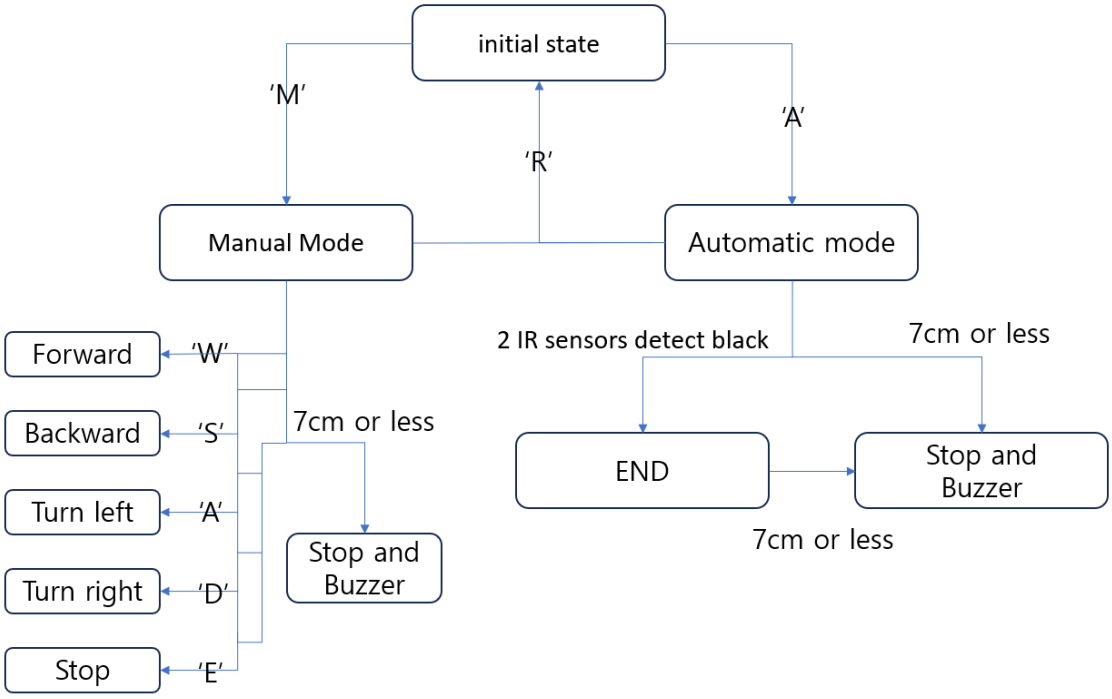


Crane upper body control

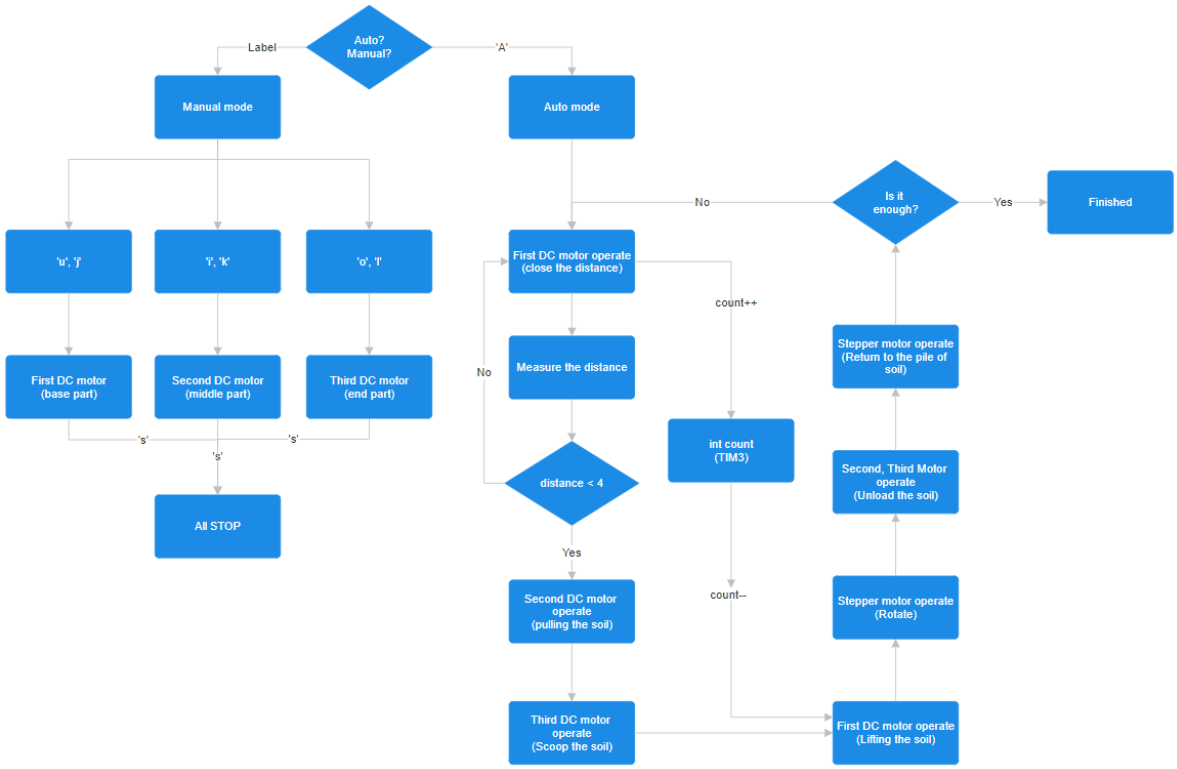


V. Algorithm

Logic Design



Depending on the received Bluetooth data, the system determines whether it is in manual mode or autonomous mode. The WSADE is used as a flag for control, and common flags used in both manual and autonomous modes include halting when the ultrasonic sensor distance measurement is 7cm or less, accompanied by the activation of a buzzer.



At the beginning, you choose whether to control the excavator automatically or manually. In manual mode, you control the excavator using the 'u,' 'j,' 'i,' 'k,' 'o,' 'l' keys, and press 's' to stop. In auto mode, as described earlier, the Ultrasonic sensor is used to measure the distance. Four DC motors are operated to narrow the distance to the soil, and the scoop action is performed to lift the soil. The Stepper motor is then used to rotate the excavator's body. The measurement position must always be constant, but controlling DC motors is difficult, and they lack awareness of the current position. Therefore, TIM3 is used to measure the time it takes for the excavator to descend and ascend, enabling a return to the initial state.

Finally, once the soil transfer task is complete, measure the distance between the soil pile and the excavator in the initial state. If the user has spread the soil as needed, stop the excavation. If more spreading is required, repeat the above task.

Code

Driving Part

```
#include "ecSTM32F411.h"

#define GO              0x57  //W
#define BACK            0x53  //S
#define RIGHT           0x44  //D
#define LEFT            0x41  //A
#define STOP            0x45  //E
#define ARM_R           0x43  //C
#define ARM_L           0x5A  //Z

#define REST           0x52  //R
#define Control        0x4D  //M
#define AUTO           0x4C  //L

#define TRIG PA_6
#define ECHO PB_6
#define auto_stop    0
#define auto_up      1
```

```
#define auto_right   2
#define auto_left    3

static volatile int MODE = 0;

static volatile int cnt = 0;
uint32_t ovf_cnt = 0;
uint32_t auto_cnt = 0;
static volatile float distance = 0;
float timeInterval = 0;
float time1 = 0;
float time2 = 0;
int display_auto = 0;

uint8_t btData = 0;

int flag = 0;
PinName_t seqCHn[2] = {PB_0, PB_1};

static volatile uint32_t right_IR, left_IR;
static volatile int BUZZ = 0;

void MCU_init(void);                        //Port and pin configuration in the
configuration
void Display_State(uint8_t direction);  //Outputting which key the received
value corresponds to
void Display_MState(void);          //Outputting manual mode state every 1
second
void Display_AState(void);          //Outputting line tracer state and LED
toggle every 1 second
void Control_mode(void);            //Output PWM as per the keys input in manual
mode
void Auto_mode(void);               //PWM output based on AEC sensor values
void USART1_IRQHandler(void);       //Receiving values whenever Bluetooth
communication comes in
void TIM4_IRQHandler(void);         //Ultrasound distance sensor input,
calculation, and counting
void ADC_IRQHandler(void);          //Receive ADC values every 50 msec

int main(void) {
   // Initialiization --------------------------------------------------------
   MCU_init();
   // Inifinite Loop ---------------------------------------------------------
   while (1){
       //Auto_Display_State(MODE, drirec)
         GPIO_write(GPIOB, 5, BUZZ);
       //Manual_Set_State(MODE, direc, drc, speed);
        if(btData == Control){
            MODE=1;
         }else if(btData == AUTO){
            MODE=2;
         }else if(btData == REST){
            MODE=0;
         }
         if(MODE ==1){
                     GPIO_write(GPIOC, 2, 1);
            GPIO_write(GPIOC, 3, 1);
            PWM_duty(PA_0, (float)(1));
```

```
            PWM_duty(PA_1, (float)(1));
            GPIO_write(GPIOA, 5, 1);
                    Display_MState();
                    Control_mode();

         }else if(MODE ==2){
                    GPIO_write(GPIOC, 2, 1);
           GPIO_write(GPIOC, 3, 1);
            Auto_mode();
            Display_AState();

         }else if(MODE ==0){
            GPIO_write(GPIOC, 2, 1);
            GPIO_write(GPIOC, 3, 1);
            PWM_duty(PA_0, (float)(1));
            PWM_duty(PA_1, (float)(1));
                    GPIO_write(GPIOA, 5, 0);
         }
   }
}
```

- **MCU_init(void) :** Function to configure pins and ports for UART1, ADC, Buzzer, Motor Output, * Motor Direction Pin, and Ultrasonic sensors.
- **Direction_display(uint8_t direction) :** When receiving a Windows key input via Bluetooth communication, it informs about the State taken based on the received key.
- **void Display_MState(void) :** When receiving a Windows key input via Bluetooth communication, it informs about the corresponding actions taken based on the received key.
- **void Display_AState(void) :** Through Bluetooth communication, it periodically indicates the current autonomous movement state every second.
- **void Control_mode(void) :** The motor output is determined by the received key values when in manual control mode, as specified in the operation manual.
- **void Auto_mode(void) :** The motor output in autonomous driving mode is determined based on the received ADC data.
- **void USART1_IRQHandler(void) :** When the Bluetooth data is received, the USART1_IRQHandler is triggered.
- **void TIM4_IRQHandler(void) :** In the TIM4_IRQHandler function, the cnt for each state output is incrementing, and distance-related data is being calculated.
- **void ADC_IRQHandler(void) :** In the TIM4_IRQHandler function, the cnt for each state output is incrementing, and distance-related data is being calculated.

Crane upper body control

```
#include "ecSTM32F411.h"
// Functions
void setup(void);
void TIM3_IRQHandler(void);
void motor_init(void);
void motor_stop(int motor);
void motor_operate(int motor, int dir);
void printState(void);

// ARM Part
PinName_t PWM_PIN1 = PA_0;
#define DIR_PIN1 3
PinName_t PWM_PIN2 = PA_1;
```

```
#define DIR_PIN2 2
PinName_t PWM_PIN3 = PB_10;
#define DIR_PIN3  4
float period = 1;
float duty1;
float duty2;
float duty3;
int dir1 = 0;
int dir2 = 0;
int dir3 = 0;

// stepper motor
#define A 8
#define B 4
#define NA 5
#define NB 3
int RPM = 2;

// UltraSonic parameter define
#define TRIG PA_6
#define ECHO PB_6
uint32_t ovf_cnt = 0;
float distance = 0;
float timeInterval = 0;
float time1 = 0;
float time2 = 0;

// Variables
char dis[4];
char start = 0;
int input;
char mode;
uint32_t count = 0;
int temp = 0;
int flag = 0;

// Bluetooth Data
static volatile uint8_t BT_Data = 0;
```

This code defines functions and variables. It specifies the pins used for DC motors and the ultrasonic sensor, as well as variables for storing values required for logic implementation and other purposes.

```
int main(void) {
    // Initialization --------------------------------------------------
    setup();
    motor_init();
    // Infinite Loop ---------------------------------------------------
    while (1) {
        motor_init();
        if(mode == 'A'){
            distance = (float) timeInterval * 340.0 / 2.0 / 10.0;    // [mm] ->
[cm]
            if(distance > 4.5 && distance <= 13 && distance >= 30){
                motor_operate(1, 1);
                motor_stop(2);
                motor_stop(3);
```

```
                temp = 1;       // count 시작
            }
            else if(distance >= 13){    // 굴삭 종료
                motor_stop(1);
                motor_stop(2);
                motor_stop(3);
                USART1_write("THE END\r\n", 7);
                mode = 'M';
            }
            else{
                temp = 0;       // count 종료
                // load
                // first step : 중간 모터 작동
                USART1_write("Digging...\r\n", 12);
                motor_stop(1);
                motor_operate(2, 0);
                delay_ms(10000);
                // second step : 삽 모터 작동
                USART1_write("Loading...\r\n", 12);
                motor_stop(2);
                motor_operate(3,0);
                delay_ms(20000);
                // third step 첫 번째 모터 복귀
                flag = 1;
                count += 3000;   // 올라가는 시간 가중치 (중력보상)
                while(count >= 2870){
                    motor_operate(1, 0);
                    motor_stop(3);
                }
                flag = 0;
                motor_stop(1);
                // rotate
                motor_init();
                Stepper_step(800, 1, FULL);
                USART1_write("Unloading...\r\n", 14);
                // unload
                // firth step0 : 중간 모터 작동
                motor_stop(1);
                motor_operate(2, 1);
                delay_ms(10000+2500);   // 중간 모터 시간 가중치
                // sixth step : 삽 모터 작동
                motor_stop(2);
                motor_operate(3,1);
                delay_ms(20000+4500);   // 삽 작동 시간 가중치 (중력보상)
                // rotate
                USART1_write("Returnning...\r\n", 15);
                motor_init();
                Stepper_step(380, 0, FULL);
                delay_ms(1000);
            }
        }
        else if(mode == 'M'){
            distance = (float) timeInterval * 340.0 / 2.0 / 10.0;    // [mm] ->
[cm]
            // First
            GPIO_write(GPIOC, DIR_PIN1, dir1);
            PWM_duty(PWM_PIN1, duty1);
            // Second
```

```
            GPIO_write(GPIOC, DIR_PIN2, dir2);
            PWM_duty(PWM_PIN2, duty2);
            // shovel
            GPIO_write(GPIOC, DIR_PIN3, dir3);
            PWM_duty(PWM_PIN3, duty3);
        }
        printState();
        delay_ms(1000);
    }
}
```

The main code distinguishes between Auto mode and Manual mode.

In the `USART1_IRQHandler()` function, when the data 'A' or 'M' is received through Bluetooth via Tera Term, the mode is switched accordingly. If 'A' is received, it transitions to Auto mode, and if 'M' is received, it transitions to Manual mode.

Auto mode

1. If the distance value obtained through the ultrasonic sensor is greater than 4.5cm (length of the shovel), the first motor on the Base side operates at the same speed, narrowing the distance. When the first motor operates, the temp flag activates, causing the count value in TIM3 to increase. The count value is used as a variable to return to the initial position of the DC motor using the timer, as the rotation count and position of the DC motor cannot be determined due to its characteristics.
2. When the distance value becomes less than 4.5, the top second motor operates, pulling the soil towards the excavator side. At this time, the delay_ms() function from the Systick header file is used to operate for a certain amount of time.
3. The pulled soil is excavated by the third motor, which operates as a shovel.
4. To perform the task of lifting the soil with the shovel, the first motor is operated to rise to the initial height. The flag operates, and during this time, the count variable stored in step 1 is counted down in TIM3. During this period, the first motor operates, allowing it to return to the initial position.
5. Using a stepper motor, rotate the body to position it where the soil is to be deposited.
6. Operate the second and third motors to deposit the soil pile.
7. Use the stepper motor to rotate the body back to the working position.
8. After returning to the initial position, the excavator measures the distance from the excavation point and compares it with the depth set by the user. If it is deeper, it stops, and if not, it repeats the process from step 1.

Manual mode

This mode is for direct user control. The control keys received through USART1's Bluetooth data are used to move the first, second, and third motors accordingly. Similar to the auto mode, the manual mode displays the depth from the excavation point to the user every second using the printstate() function.

```
void printState(void){
   sprintf(dis, "%f", distance);
   USART1_write("Distance is ", 12);
   USART1_write(&dis, 4);
   USART1_write("\r\n", 2);
}
```

This code is designed to display the distance between the excavator and the ground using Tera Term. Since the distance is output through USART1 and the variable "distance" is of integer type, it cannot be directly output. To address this, the sprintf() function is used to store the distance in a character array called "dis," and then USART1_write() function is employed to output the distance.

```
void printState(void){
   sprintf(dis, "%f", distance);
   USART1_write("Distance is ", 12);
   USART1_write(&dis, 4);
   USART1_write("\r\n", 2);
}

void USART1_IRQHandler(){
    if(is_USART1_RXNE()){
        BT_Data = USART1_read();
        // First
        if(BT_Data == 'z') input = 10;
        else if(BT_Data == 'x') input = 15;
        else if(BT_Data == 'c') input = 20;

        if (BT_Data == 'M'){
            mode = 'M';
        }
        else if (BT_Data == 'A'){
            mode = 'A';
        }

        if(mode == 'M'){
            if(BT_Data == 'u'){
                dir1 = 0;
                duty1 = 1;
            }
            else if(BT_Data == 'j'){
                dir1 = 1;
                duty1 = 0;
            }
            else if(BT_Data == 'i'){
                dir2 = 0;
                duty2 = 1;
            }
            else if(BT_Data == 'k'){
                dir2 = 1;
                duty2 = 0;
            }
            else if(BT_Data == 'o'){
                dir3 = 0;
                duty3 = 1;
            }
            else if(BT_Data == 'l'){
                dir3 = 1;
                duty3 = 0;
            }
            else if(BT_Data == 's'){
                dir1 = dir2 = dir3 = 0;
                duty1 = duty2 = duty3 = 0;
            }
        }
```

```
    }
}
```

This code is a function responsible for reading data received via Bluetooth in interrupt format, serving the purpose of mode configuration. Additionally, in Manual mode, it enables control of the actions and directions of each of the three motors using the 'u', 'j', 'i', 'k', 'o', 'l' keys. The 's' key is used to stop the operation of all motors.

```
void motor_init(void){
    GPIO_write(GPIOC, DIR_PIN1, 0);
    PWM_duty(PWM_PIN1, 0);
    GPIO_write(GPIOC, DIR_PIN2, 0);
    PWM_duty(PWM_PIN2, 0);
    GPIO_write(GPIOC, DIR_PIN3, 0);
    PWM_duty(PWM_PIN3, 0);
}
```

This is the motor initialization code designed to prevent the motors from rotating due to the existing PWM signals.

```
void motor_stop(int motor){
    int dir_pin;
    int motor_pin;
    if(motor == 1){
        dir_pin = DIR_PIN1;
        motor_pin = PWM_PIN1;
    }
    else if(motor == 2){
        dir_pin = DIR_PIN2;
        motor_pin = PWM_PIN2;
    }
    else if(motor == 3){
        dir_pin = DIR_PIN3;
        motor_pin = PWM_PIN3;
    }
    GPIO_write(GPIOC, dir_pin, 0);
    PWM_duty(motor_pin, 0);
}
```

This function takes the motor number as an argument and stops the motor with that specific number. Using this function helps simplify the code and makes it easier to interpret.

```
void motor_operate(int motor, int dir){
    int dt;
    int dir_pin;
    int motor_pin;
    if(motor == 1){
        dir_pin = DIR_PIN1;
        motor_pin = PWM_PIN1;
    }
    else if(motor == 2){
        dir_pin = DIR_PIN2;
        motor_pin = PWM_PIN2;
    }
    else if(motor == 3){
```

```
        dir_pin = DIR_PIN3;
        motor_pin = PWM_PIN3;
    }
    GPIO_write(GPIOC, dir_pin, dir);
    if(dir == 1) dt = 0;
    else if(dir == 0) dt = 1;
    PWM_duty(motor_pin, dt);
}
```

This function takes the motor number and direction as arguments, allowing the motor to operate in the specified direction. Using this function helps simplify the code and makes it easier to interpret.

```
void TIM4_IRQHandler(void){
    if(is_UIF(TIM4)){                     // Update interrupt
        ovf_cnt++;                                  // overflow count
    // count for 1sec
        clear_UIF(TIM4);                          // clear update interrupt
flag
    }
    if(is_CCIF(TIM4, 1)){                         // TIM4_Ch1 (IC1) Capture
Flag. Rising Edge Detect
        time1 = TIM4->CCR1;                           // Capture TimeStart
        clear_CCIF(TIM4, 1);                // clear capture/compare interrupt
flag
    }
    else if(is_CCIF(TIM4, 2)){                           // TIM4_Ch2 (IC2)
Capture Flag. Falling Edge Detect
        time2 = TIM4->CCR2;                           // Capture TimeEnd
        timeInterval = ((time2 - time1) + (TIM4->ARR+1) * ovf_cnt) * 0.01;    //
(10us * counter pulse -> [msec] unit) Total time of echo pulse
        ovf_cnt = 0;                          // overflow reset
        clear_CCIF(TIM4,2);                           // clear capture/compare
interrupt flag
    }
}
```

This function is a TIM4 interrupt that reads the values from the ultrasonic sensor.

```
void TIM3_IRQHandler(void) {
    if (is_UIF(TIM3)) {            // Check UIF(update interrupt flag)

         if(temp == 1){
            count++;
         }
         if(flag == 1){
            count--;
         }
    }
    clear_UIF(TIM3);            // Clear UI flag by writing 0
}
```

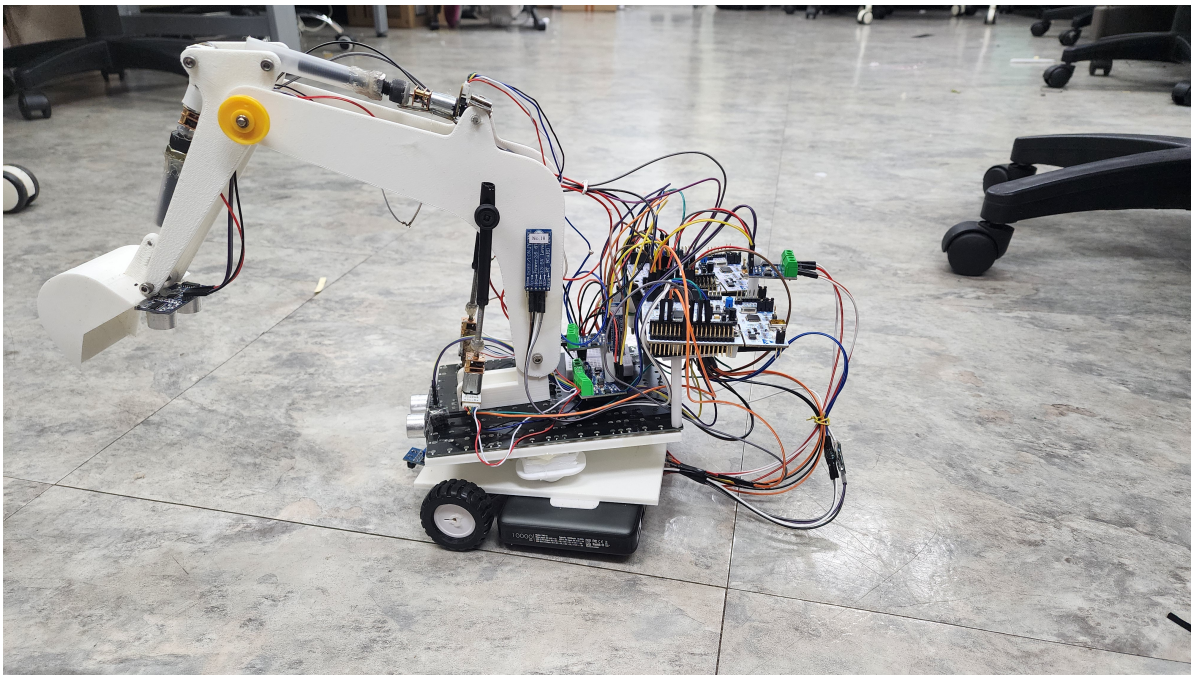
This function is a TIM3 interrupt that performs counting based on the mentioned flag in the main function.

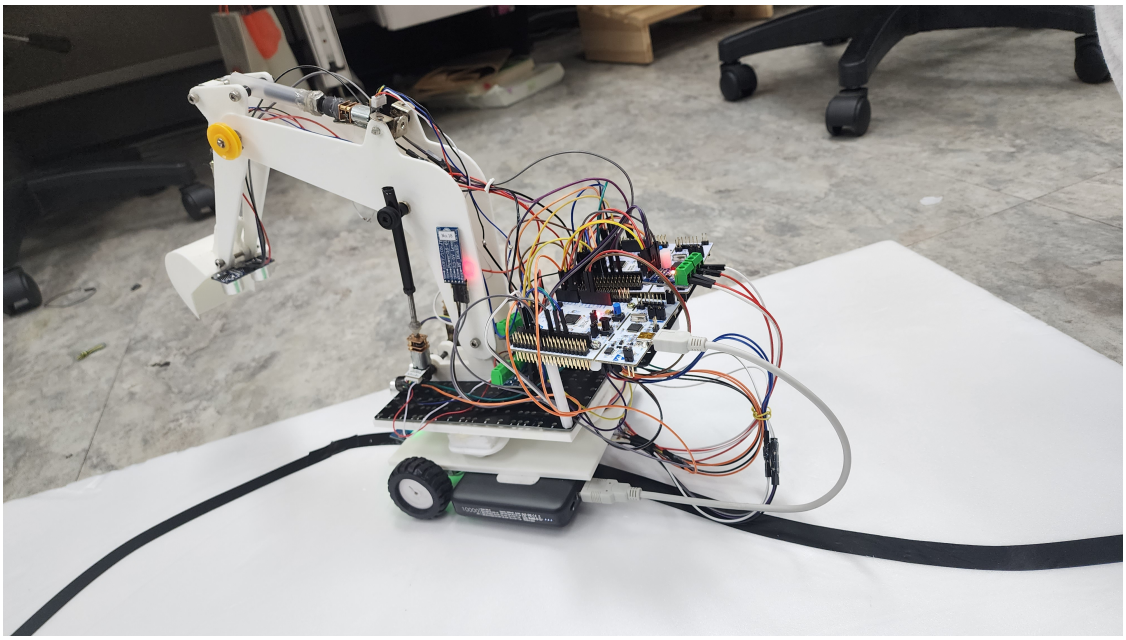
```
// Initialiization
void setup(void) {
    RCC_PLL_init();

    // SYSTICK
    SysTick_init();

    // Bluetooth serial init
    UART1_init();
    UART1_baud(BAUD_9600);
    UART2_init();

    // DIR1
    GPIO_init(GPIOC, DIR_PIN1, OUTPUT);
    GPIO_otype(GPIOC, DIR_PIN1, EC_PUSH_PULL);
    // DIR2
    GPIO_init(GPIOC, DIR_PIN2, OUTPUT);
    GPIO_otype(GPIOC, DIR_PIN2, EC_PUSH_PULL);
    // DIR3
    GPIO_init(GPIOC, DIR_PIN3, OUTPUT);
    GPIO_otype(GPIOC, DIR_PIN3, EC_PUSH_PULL);

    // Stepper motor
    Stepper_init(GPIOA, A, GPIOB, B, GPIOB, NA, GPIOB, NB);
    Stepper_setSpeed(RPM);

    // TIM
    TIM_UI_init(TIM3, 1);

    // ARM Part
    // PWM1
    PWM_init(PWM_PIN1);
    PWM_period_ms(PWM_PIN1, period);
    // PWM2
    PWM_init(PWM_PIN2);
    PWM_period_ms(PWM_PIN2, period);
    // PWM3
    PWM_init(PWM_PIN3);
    PWM_period_ms(PWM_PIN3, period);

    // Input Capture configuration ------------------------------------------------
-------------------------
    // PWM:  TIM4_CH2 (PA_6 AFmode)
   PWM_init(TRIG);
   GPIO_otype(GPIOA, 6, EC_PUSH_PULL);   //PWM Pin Push-Pull
   GPIO_pupd(GPIOA, 6, EC_NONE);         //PWM Pin NO Pull-up, Pull-down
   GPIO_ospeed(GPIOA, 6, EC_FAST);       //PWM Pin Fast
   PWM_period_us(TRIG, 50000);        // 50 msec PWM period
   PWM_pulsewidth_us(TRIG,10);        // 10 usec PWM pulse width

   // input Capture:  TIM4_CH1 (PB_6 AFmode)
   ICAP_init(ECHO);
   GPIO_pupd(GPIOB, 6, EC_NONE);         //PWM Pin NO Pull-up, Pull-down
   ICAP_counter_us(PB_6, 10);            //Counter Clock : 0.1MHz (10us)
   ICAP_setup(PB_6, 1, IC_RISE);         //TI4 -> IC1 (rising edge)
   ICAP_setup(PB_6, 2, IC_FALL);         //TI4 -> IC2 (falling edge)
}
```

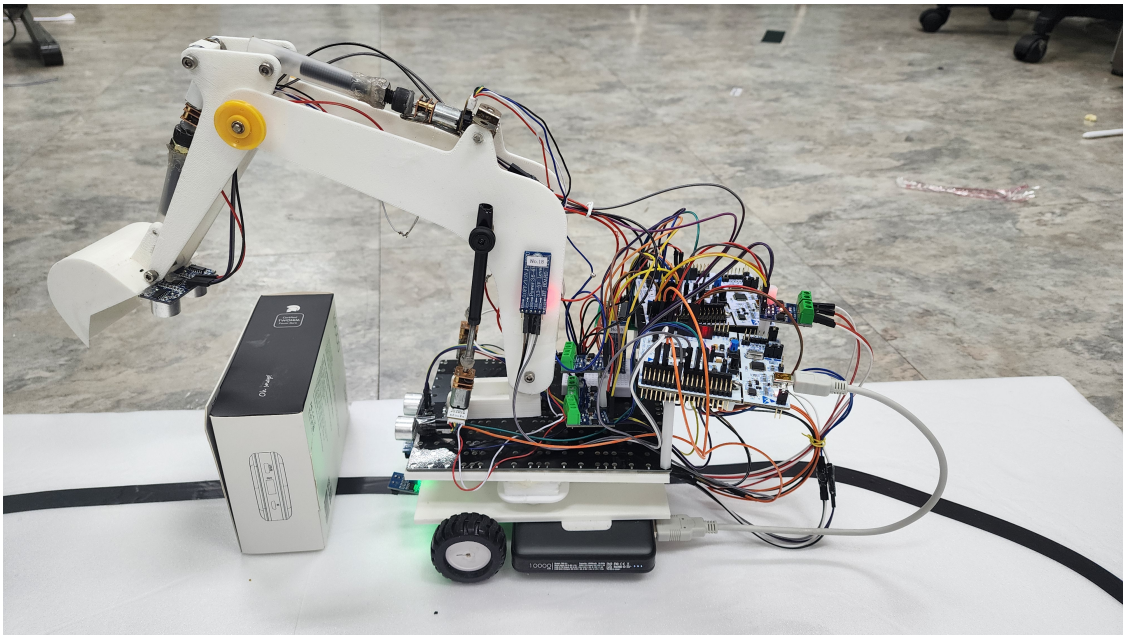
This function initializes specified pins for motors, sensors, timers, clocks, Systick, etc. The detailed setting methods for these can be found in the MCU configuration table provided above.

VI. Results and Demo

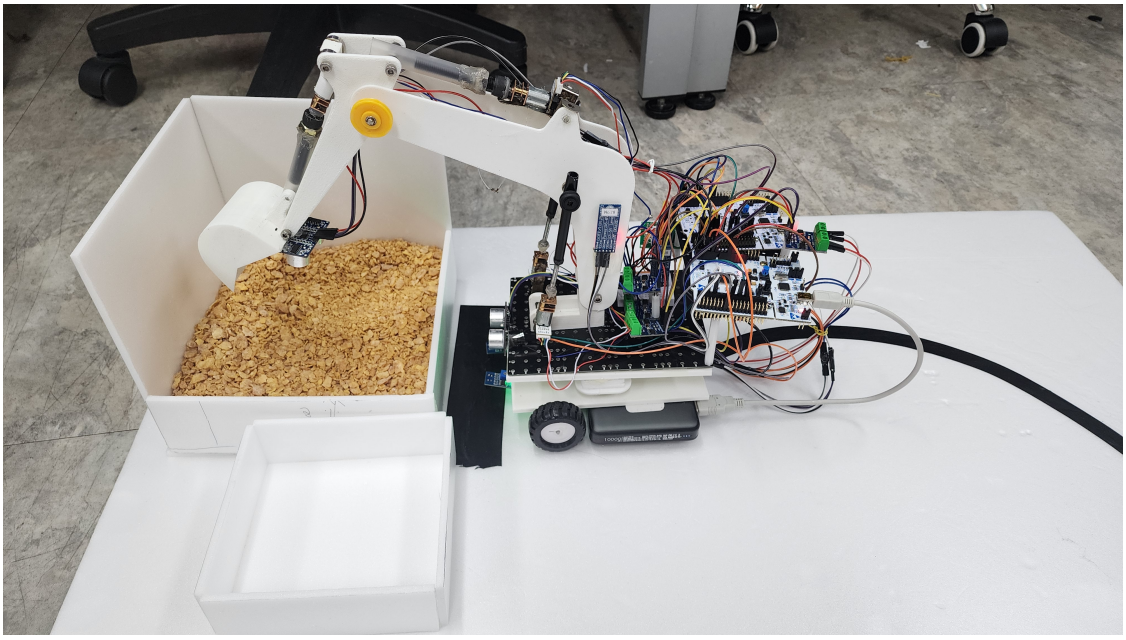
Experiment images and results



> It is autonomously moving along the black line.
>
> 
>
> It detects an object in front, stops, and activates the buzzer.



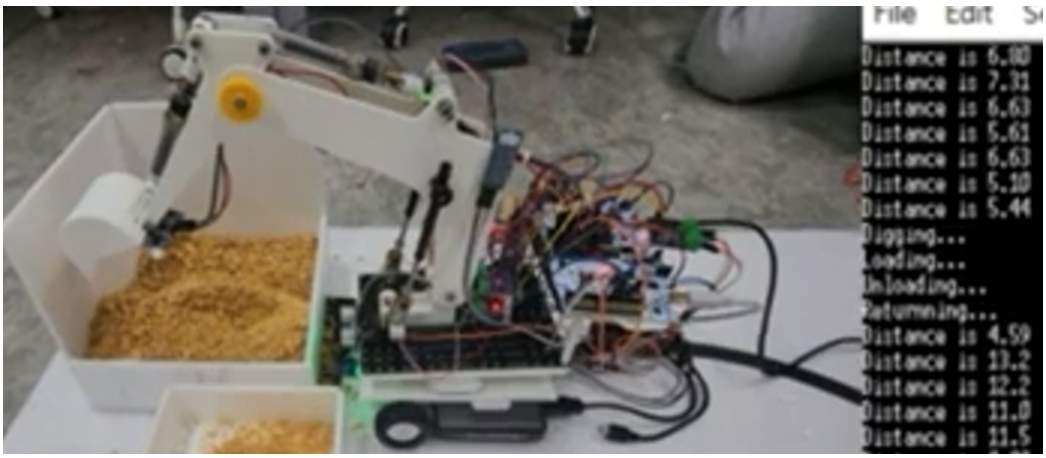
It detects two black lines using two ADC sensors and comes to a stop and alarm by the buzzer.



It starts digging the ground when the depth is below 4.5cm.

The stepper motor rotates to transfer the dispensed material.

It measures the distance.



It stops.(over 13cm)

Demo Video : [https://www.youtube.com/watch?v=uYmLbaKMJFY]

Analysis

The conventional excavator operation relies on the experience of the operator, leading to variations in excavation results. Additionally, the simultaneous measurement and excavation were not possible, requiring a labor-intensive process where a worker had to measure and confirm whether the target values were reached.

To address these challenges, the concept of MG/MC excavators has emerged. These excavators utilize embedded controllers to receive feedback on the excavator's current position and state through GPS, gyro sensors, and other technologies. Once the initial settings are configured by the operator, the excavator can accurately reach the desired target without constant manual adjustments.

In implementing this technology, we utilized embedded controllers for autonomous navigation to the excavation site, employed ultrasonic sensors for measuring and visualizing the distance to the excavation point, and incorporated automatic excavation features. This innovation eliminates the need for operator experience, ensuring precise excavation.

The autonomous navigation feature not only prevents industrial accidents resulting from collisions with heavy machinery at construction sites but also enables the operation of excavators with minimal human resources. This reduces labor costs and allows for reaching target values in a shorter timeframe. Moreover, this advancement contributes to resource efficiency in construction projects.

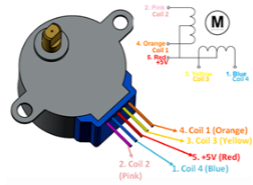
In summary, the incorporation of embedded controllers and advanced sensors in MG/MC excavators enhances accuracy, efficiency, and safety in excavation operations, offering benefits such as accident prevention, reduced manpower requirements, and

VII. Reference

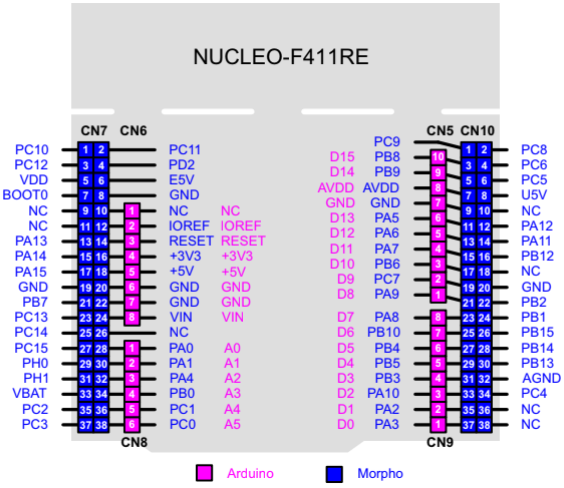
EARTHMOVING EQUIPMENT AUTOMATION: A REVIEW OF TECHNICAL ADVANCES AND FUTURE OUTLOOK by Ehsan Rezazadeh Azar, Assistant Professor Department of Civil Engineering, Lakehead University

How To Make a Remote Control JCB Excavator at Home - YouTube

VIII. Appendix



Rated voltage :	5VDC
Number of Phase	4
Speed Variation Ratio	1/64
Stride Angle	5.625°/64
Frequency	100Hz
DC resistance	50Ω±7%(25℃)
Idle In-traction Frequency	> 600Hz
Idle Out-traction Frequency	> 1000Hz
In-traction Torque	>34.3mN.m(120Hz)
Self-positioning Torque	>34.3mN.m
Friction torque	600-1200 gf.cm
Pull in torque	300 gf.cm
Insulated resistance	>10MΩ(500V)
Insulated electricity power	600VAC/1mA/1s
Insulation grade	A
Rise in Temperature	<40K(120Hz)
Noise	<35dB(120Hz,No load,10cm)
Model	28BYJ-48 – 5V



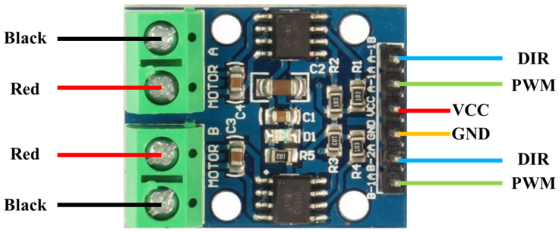
Timer PinOut Map

Advanced Timer

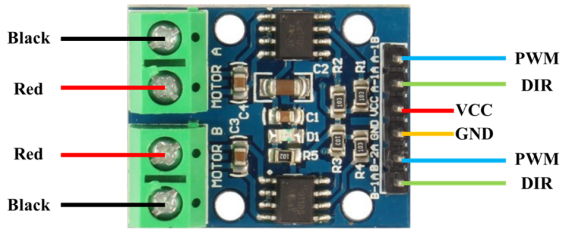
Timer	Channel	Port	Pin
1	1	A	8
	1N	A	7
		B	13
	2	A	9
	2N	B	0
		B	14
	3	A	10
	3N	B	1
		B	15
	4		
	4N		

General Purpose Timer

Timer	Channel	Port	Pin
2	1	A	0
		A	5
		A	15
	2	A	1
		B	3
	3	B	10
	4		
3	1	A	6
		B	4
		C	6
	2	B	5
		C	7
	3	C	8
	4	C	9
4	1	B	6
	2	B	7
	3	B	8
	4	B	9



Case 1



Case 2

Case	Motor	MCU	DIR	Duty ratio: 0→1	Direction
Case 1	Black Red	DIR PWM	LOW	Velocity ↑	CCR
			HIGH	Velocity ↓	CR
Case 2	Red Black	PWM DIR	LOW	Velocity ↑	CR
			HIGH	Velocity ↓	CCR

IX. Troubleshooting

There was an issue with using multiple timers and PWM signals on a single STM board, where the insufficient input voltage led to inadequate motor operation. Additionally, the drawback of using DC motors was the difficulty in obtaining feedback, which could be addressed by using stepper motors or attaching gyro sensors to accurately determine the position of each joint, enabling the implementation of feedback for more precise control.

X. Other Appendix

How To Make a Remote Control JCB Excavator at Home - YouTube