

# [LAB#1] Smart mini-fan with STM32-duino

Date: Sep.15.2023

Author: 21900416 Gyeonheal An

Github: [https://github.com/AnGyeonheal/Embedded\\_Control](https://github.com/AnGyeonheal/Embedded_Control)

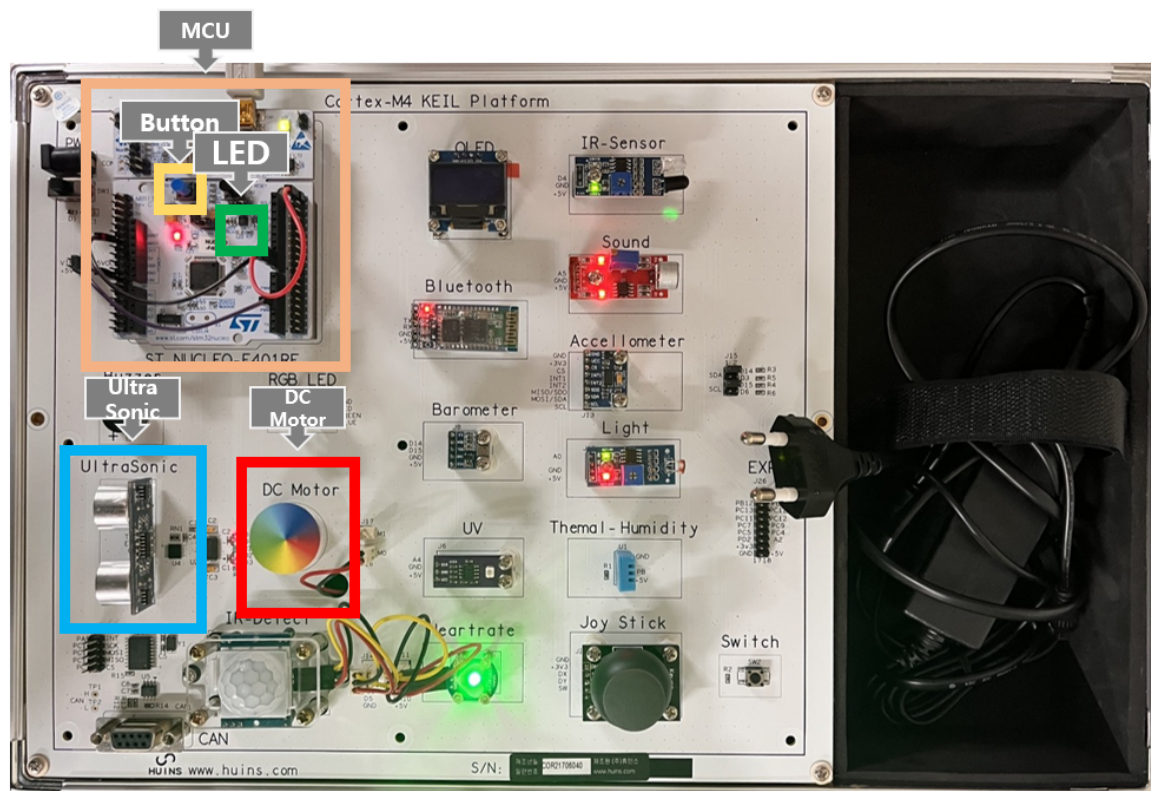
Demo Video: <https://youtu.be/PtdUisu286M?feature=shared>

## I. Introduction

In this lab, we are required to create a simple program that uses arduino IDE for implementing a simple embedded digital application. The report designed a smart fan algorithm that recognizes the distance from the user and can change the mode using a button.

## II. Requirement

### i. Hardware



- **MCU**
  - NUCLEO-F401RE
- **Sensor**
  - Ultrasonic distance sensor(HC-SR04)
  - Button
- **Actuator/Display**
  - LED

- DC motor (RK-280RA)

## ii. Software

- Arduino IDE
- Tera Term

## III. Problem

---

### i. Procedure

The program needs to run the Fan only when the distance of an object is within a certain value.

Example: An automatic mini-fan that runs only when the face is near the fan. Otherwise turns off.

- As the button **B1** is pressed, change the fan velocity. The MODE(states) are
  - MODE(state): **OFF(0%), MID(50%), HIGH(100%)**
- When the object(face) is detected about 50 mm away, then it automatically pauses the fan temporarily.
  - Even the fan is temporarily paused, the MODE should be changed whenever the button **B1** is pressed
- When the object(face) is detected within 50mm, then it automatically runs the fan
  - It must run at the speed of the current MODE
- LED(**LED1**): Turned OFF when MODE=OFF. Otherwise, blink the LED with 1 sec period (1s ON, 1s OFF)
- Print the distance and PWM duty ratio in Tera-Term console (every 1 sec).
- Must use Mealy FSM to control the mini-fan
  - Draw a FSM(finite-state-machine) table and state diagram
  - Example Table. See below for example codes

### ii. Configuration

#### Ultrasonic distance sensor

Trigger:

- Generate a trigger pulse as PWM to the sensor
- Pin: **D10** (TIM4 CH1)
- PWM out: 50ms period, 10us pulse-width

Echo:

- Receive echo pulses from the ultrasonic sensor
- Pin: **D7** (Timer1 CH1)
- Input Capture: Input mode
- Measure the distance by calculating pulse-width of the echo pulse.

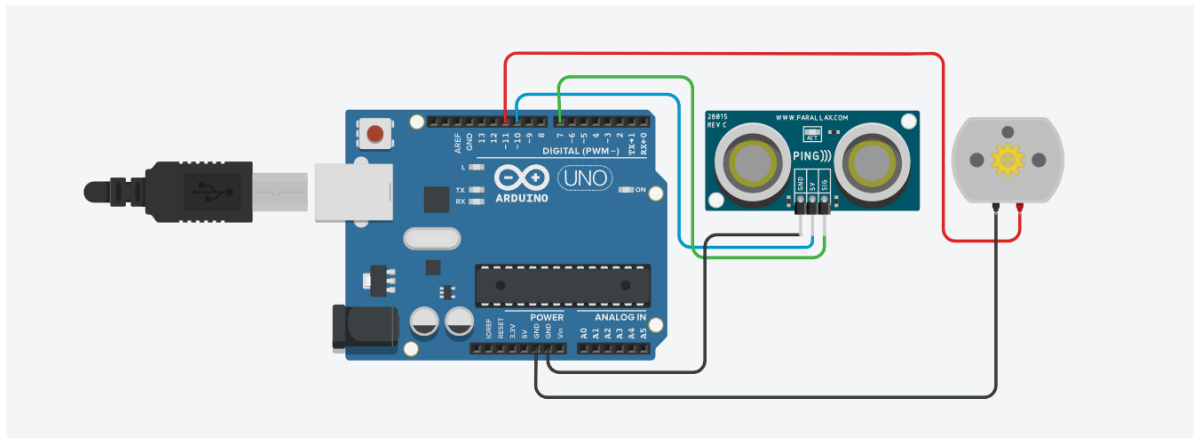
#### USART

- Display measured distance in [cm] on serial monitor of Tera-Term.
- Baudrate 9600

#### DC Motor

- PWM: PWM1, set 10ms of period by default
- Pin: **D11** (Timer1 CH1N)

### iii. Circuit/Wiring Diagram



## IV. Algorithm

### i. Overview

#### Define States

State	Description
S0	Mode: Off
S1	Mode: MID
S2	Mode: HIGH

#### Define Inputs

Button	Object Detection
0: Not Pushed / 1: Pushed	F: Not Detected / T: Detected

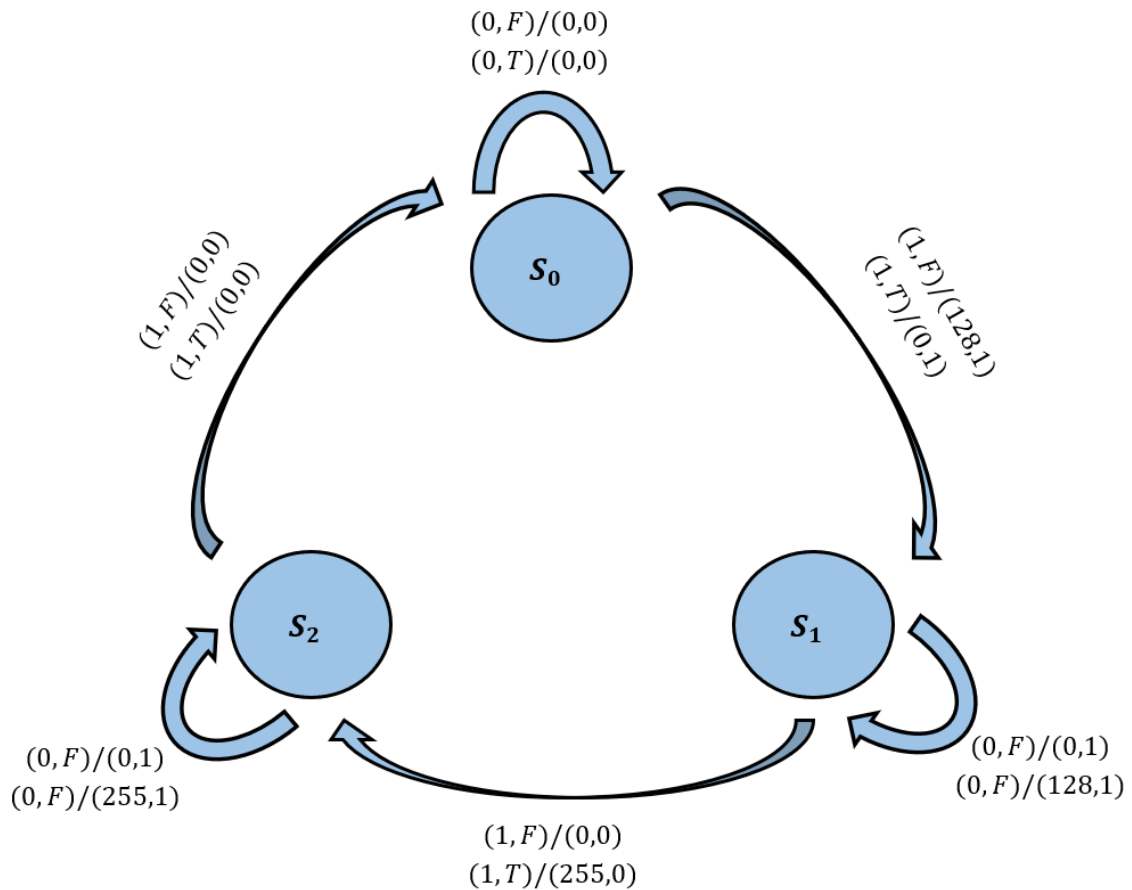
#### Define Outputs

- L: LED (0 / 1)
- V: Speed of Motor (0 / 128 / 255)

### ii. Mealy FSM Table

Present State	Next State (X, Y)				Output Z			
	(0, F)	(0, T)	(1, F)	(1, T)	(0, F)	(0, T)	(1, F)	(1, T)
S <sub>0</sub>	S <sub>0</sub>	S <sub>0</sub>	S <sub>1</sub>	S <sub>1</sub>	V=0 L=0	V=0 L=0	V=0 L=1	V=50 L=1
S <sub>1</sub>	S <sub>1</sub>	S <sub>1</sub>	S <sub>2</sub>	S <sub>2</sub>	V=0 L=1	V=50 L=1	V=0 L=1	V=100 L=1
S <sub>2</sub>	S <sub>2</sub>	S <sub>2</sub>	S <sub>0</sub>	S <sub>0</sub>	V=0 L=1	V=100 L=1	V=0 L=0	V=0 L=0

### iii. Mealy State Graph



## Vi. Description with Code

- Lab source code

```
// State definition
#define S0 0
#define S1 1

// Address number of output in array
#define PWM 0
#define LED 1

const int ledPin = 13;
const int pwmPin = 11;
const int btnPin = 3;

unsigned char state = S0;
unsigned char nextstate = S0;
unsigned char input = 0;
unsigned char ledOut = LOW;
unsigned char pwmOut = 0;

// State table definition
typedef struct {
    uint32_t out[2][2];    // output = FSM[state].out[input][PWM or LED]
    uint32_t next[2];      // nextstate = FSM[state].next[input]
} State_t;

State_t FSM[2] = {
```

```

    { {{0  , LOW }}, {160, HIGH}}, {S0, S1} },
    { {{160, HIGH}, {0  , LOW }}, {S1, S0} }
};

void setup() {
    // initialize the LED pin as an output:
    pinMode(ledPin, OUTPUT);

    // Initialize pwm pin as an output:
    pinMode(pwmPin, OUTPUT);

    // initialize the pushbutton pin as an interrupt input:
    pinMode(btnPin, INPUT_PULLUP);
    attachInterrupt(digitalPinToInterrupt(btnPin), pressed, FALLING);
}

void loop() {
    // Calculate next state. then update State
    nextState();

    // Output
    analogWrite(pwmPin, pwmOut);
    digitalWrite(ledPin, ledOut);

    delay(1000);
}

void pressed(){
    input = 1;
}

void nextState(){

    pwmOut = FSM[state].out[input][PWM];
    ledOut = FSM[state].out[input][LED];

    nextstate = FSM[state].next[input];

    state = nextstate;
    input = 0;
}

```

- **Variable Declaration**

```

// State definition
#define S0 0
#define S1 1
#define S2 2

// Address number of output in array
#define PWM 0
#define LED 1

const int ledPin = 13;
const int pwmPin = 11;
const int btnPin = 3;
const int trigPin = 10;    // Trigger pin : PWM out (ultraSonic)

```

```

const int echoPin = 7;    // Echo pin : Interrupt in (UltraSonic)

float distance;
unsigned long duration;
unsigned char btn_input = 0;
unsigned char state = S0;
unsigned char nextstate = S0;
unsigned char input = 0;
unsigned char ledOut = LOW;
unsigned char fanOut = 0;
unsigned long currentMillis;

```

Pin numbers for input devices and output devices that can be identified in 3-2 were declared as variables, and variables with changing specific values were initialized.

- **Structure Declaration**

```

typedef struct {
    unsigned int next[4];    // nextstate = FSM[state].next[input]
    unsigned int led[4];     // output = FSM[state].out[input]
    unsigned int fan[4];
} State_t;

State_t FSM[3] = {
    { {S0, S0, S1, S1},{LOW, LOW, LOW, HIGH},{0, 0, 0, 128} },
    { {S1, S1, S2, S2},{HIGH, HIGH, HIGH, HIGH},{0, 128, 0, 255} },
    { {S2, S2, S0, S0}, {HIGH, HIGH, LOW, LOW}, {0, 255, 0, 0} }
};

```

The structure is a code that implements the FSM table obtained above. Components of the structure include next[4], led[4], and fan[4]. It has the next state, the presence or absence of an LED, and the speed value of the motor, respectively.

- **setup() Function Description**

```

void setup() {
    Serial.begin(9600);
    // initialize the LED pin as an output:
    pinMode(ledPin, OUTPUT);
    // initialize the pushbutton pin as an interrupt input:
    pinMode(btnPin, INPUT_PULLUP);
    attachInterrupt(digitalPinToInterrupt(btnPin), pressed, FALLING);
    // initialize the UnltraSonic sensor as an input
    pinMode(trigPin, OUTPUT);
    pinMode(echoPin, INPUT);
    // initialize the DC motor pwm signal as an output
    pinMode(pwmPin, OUTPUT);
}

```

In the setup() function, initialization was performed for verification in the Serial Monitor, and LED, Button, UltraSonic, and Motor pins were initialized.

- **loop() Function Description (1)**

```

// First, Output of current State. Then Update next state. Repeat
digitalWrite(trigPin, HIGH);

```

```

delayMicroseconds(10);
digitalWrite(trigPin, LOW);
delayMicroseconds(10);
duration = pulseIn(echoPin, HIGH);
distance = (float)duration / 58.0;
// Has different input variable values depending on two inputs
if(distance < 10){
    if(btn_input == 1)
        input = 3;
    else
        input = 1;
}
else{
    if(btn_input == 1)
        input = 2;
    else
        input = 0;
}
// 1. Output State
stateOutput();

```

In the loop() function, the distance detection by the Ultra Sonic sensor was first expressed in cm. In addition, a code is written that determines the input of what value to put into the above structure according to the button and ultrasonic sensor.

- **loop() Function Description (2)**

```

if(state != S0){
    currentMillis = millis();
    digitalWrite(ledPin, ledOut);
    while(1){
        if(millis()-currentMillis >= 1000){
            ledOut ^= 1;
            break;
        }
    }
}
else{
    digitalWrite(ledPin, ledOut);
}
currentMillis = millis();
digitalWrite(ledPin, ledOut);
analogWrite(pwmPin, fanOut);
// 2. Update State <-- Next State
nextState();
while(1){
    if(millis()-currentMillis >= 1000) break;
}

```

The loop() function has a code that glitters every second even if no object is detected when State 1 and State 2 are the second. Among the built-in functions, millis() is used, which outputs how long has passed since Arduino was executed in 1/1000 seconds.

It also invokes and executes the nextState() function that updates the nextState.

- **loop() Function Description (3)**

```

Serial.println("-----");
Serial.printf("Distance : ");
Serial.print(distance);
Serial.println(" [cm]");
Serial.printf("PWM duty ratio = ");
Serial.println(fanOut);

```

At the end of the loop() function, there is a code in Serial Monitor indicating the distance from an object in cm and a code in Fan's PWM duty ratio. It is possible to check the corresponding value in real time on the user's monitor through the Tera Term program.

- **pressed() Function**

```

void pressed() {
    btn_input = 1;
}

```

The pressed() function is a function executed through attachInterrupt when a button is pressed, so that the btn\_input variable has a value of 1.

- **nextState() Function**

```

void nextState() {
    nextstate = FSM[state].next[input];
    state = nextstate;

    // Intialize button pressed
    btn_input = 0;
}

```

The nextState() function is a function that executes in the loop() function and stores the state change according to the sensor input value in the current state, and finally initializes the btn\_input.

- **stateOutput() Function**

```

void stateOutput() {
    ledOut = FSM[state].led[input];
    fanOut = FSM[state].fan[input];
}

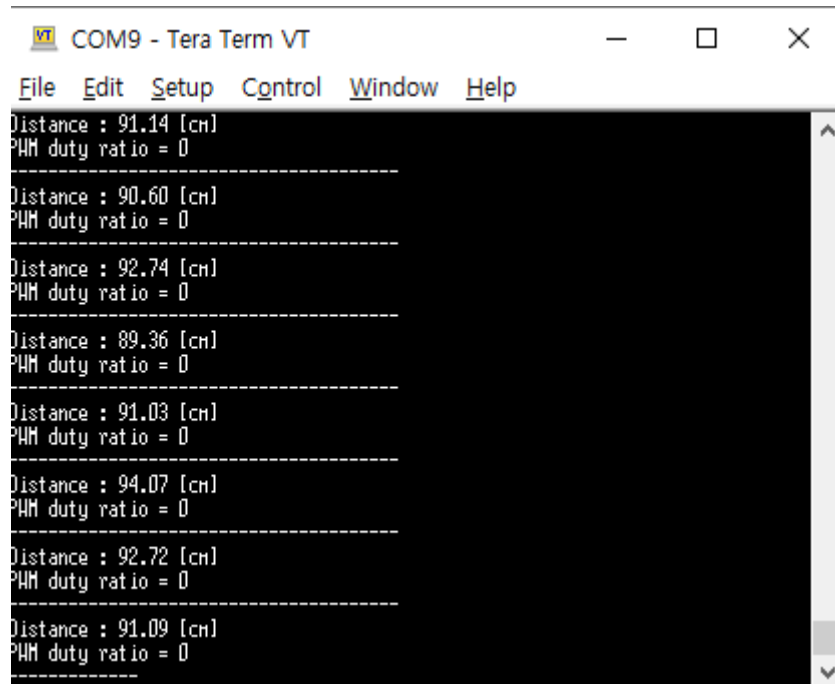
```

The stateOutput() function inputs the value of the input variable determined according to the sensor input value and outputs the speed of the LED and the fan according to the structure element.

## V. Results and Analysis

### i. Results



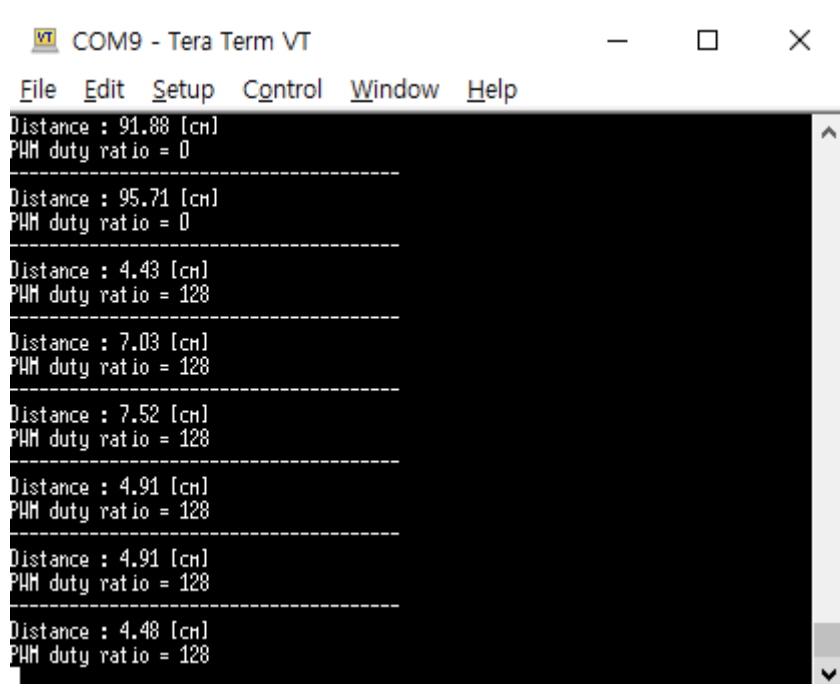


VT COM9 - Tera Term VT

File Edit Setup Control Window Help

```
Distance : 91.14 [cm]
PWM duty ratio = 0
-----
Distance : 90.60 [cm]
PWM duty ratio = 0
-----
Distance : 92.74 [cm]
PWM duty ratio = 0
-----
Distance : 89.36 [cm]
PWM duty ratio = 0
-----
Distance : 91.03 [cm]
PWM duty ratio = 0
-----
Distance : 94.07 [cm]
PWM duty ratio = 0
-----
Distance : 92.72 [cm]
PWM duty ratio = 0
-----
Distance : 91.09 [cm]
PWM duty ratio = 0
-----
```

[State = 0] / Object Detection (X) => PWM = 0

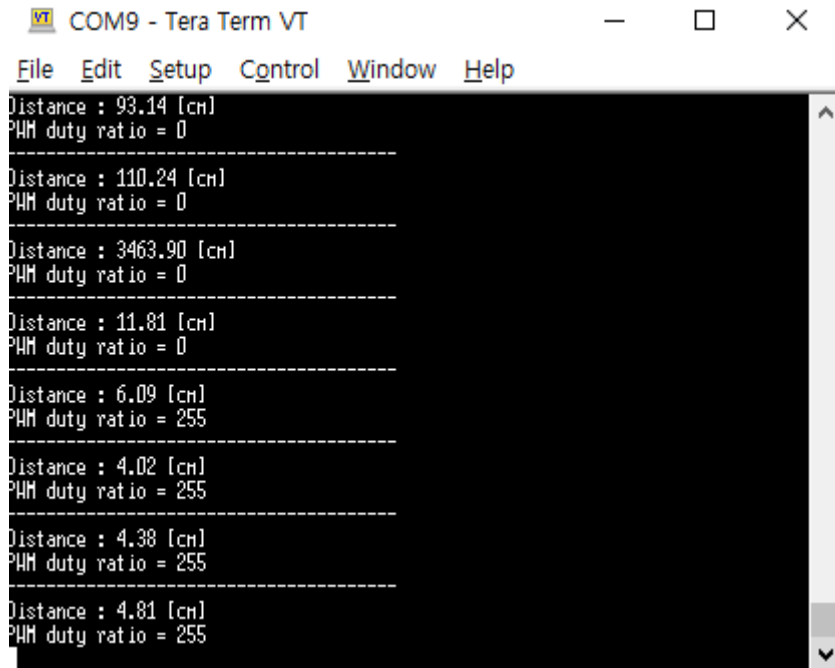


VT COM9 - Tera Term VT

File Edit Setup Control Window Help

```
Distance : 91.88 [cm]
PWM duty ratio = 0
-----
Distance : 95.71 [cm]
PWM duty ratio = 0
-----
Distance : 4.43 [cm]
PWM duty ratio = 128
-----
Distance : 7.03 [cm]
PWM duty ratio = 128
-----
Distance : 7.52 [cm]
PWM duty ratio = 128
-----
Distance : 4.91 [cm]
PWM duty ratio = 128
-----
Distance : 4.91 [cm]
PWM duty ratio = 128
-----
Distance : 4.48 [cm]
PWM duty ratio = 128
-----
```

[State = 1] / Object Detection (O) => PWM = 128



The screenshot shows a Tera Term VT window titled 'COM9 - Tera Term VT'. The window has a menu bar with 'File', 'Edit', 'Setup', 'Control', 'Window', and 'Help'. The main display area shows a series of data points separated by dashed lines. Each point consists of a distance value in centimeters and a PWM duty ratio. The data points are: Distance : 93.14 [cm], PWM duty ratio = 0; Distance : 110.24 [cm], PWM duty ratio = 0; Distance : 3463.90 [cm], PWM duty ratio = 0; Distance : 11.81 [cm], PWM duty ratio = 0; Distance : 6.09 [cm], PWM duty ratio = 255; Distance : 4.02 [cm], PWM duty ratio = 255; Distance : 4.38 [cm], PWM duty ratio = 255; and Distance : 4.81 [cm], PWM duty ratio = 255. A vertical scrollbar is visible on the right side of the window.

```
COM9 - Tera Term VT
File Edit Setup Control Window Help
Distance : 93.14 [cm]
PWM duty ratio = 0
-----
Distance : 110.24 [cm]
PWM duty ratio = 0
-----
Distance : 3463.90 [cm]
PWM duty ratio = 0
-----
Distance : 11.81 [cm]
PWM duty ratio = 0
-----
Distance : 6.09 [cm]
PWM duty ratio = 255
-----
Distance : 4.02 [cm]
PWM duty ratio = 255
-----
Distance : 4.38 [cm]
PWM duty ratio = 255
-----
Distance : 4.81 [cm]
PWM duty ratio = 255
```

[State = 2] / Object Detection (O) => PWM = 255

## ii. Demo Video

<https://youtu.be/PtdUisu286M?feature=shared>

## iii. Analysis

- Designed through FSM, algorithms can be simplified to reduce the speed at which MCU boards operate, making it easier to handle tasks.
- By using the structure, there is an advantage that the State Table designed by the FSM can be imported and transcribed into code.
- To design an MCU board that can perform the desired function, it is most important to define the State in the FSM correctly, and to reduce delay by preventing unnecessary operations within the code

## VI. Reference

<https://ykkim.gitbook.io/ec/ec-course/lab/lab-smart-mini-fan-with-stm32-duino>