Group 30: Som Lee, Trisentosa Wisesa
ECE:3360, 5 May 2022
Final Project Lab Report
Project Name: “Thomas not a Train”

1. Introduction

a. Motivation and Background

The main motivation for this project is to construct a system that uses different components we haven’t used in the previous labs. We realized that in terms of functionalities it doesn’t have a meaningful impact, but it would look cool as a collection item in your home.

b. Goals/Specification

- Implement IR receiver interrupts and command and address decoding by NEC protocols
- Controls the servo motor, LED, and mode via IR commands
- Implement ultrasonic sensor distance measurement and conversion to cm
- Others: mist maker, mp3 module , and body construction using 3D printed modeled parts.

2. Implementation

a. Overview

This project we’re building is an IR controlled rotating turret. The project is split into two microcontrollers. The first microcontroller(B1) controls the IR receiver, the servo motor, LED, as well as sending the mode trigger to the second board. The second microcontroller(B2) controls the ultrasonic sensor and mp3 module. The device starts by activating both microcontrollers, the first mode which is manual mode, is fully controlled in B1, where the users can control the turret using the IR remote. These functions are turning the LED on/off, rotating the turret, resetting the position, and switching the mode. Once the user clicks the auto mode button, B1 will output a high signal to B2 signaling the auto mode. Once in auto mode, B1 will run the servo_auto() function and rotate 180 degrees to right and left repeatedly. B2 will now also output a HIGH signal to B1 if it detects an obstacle <= 15 cm and low otherwise. If the device is in auto mode and the sensor pin is HIGH, then the device will stop its rotation and flash its LED. Once the obstacle is removed, the turret will continue rotating and the LED is turned off. To switch from auto mode to normal mode again, simply click the mode button once again.

Figure 1: IR Remote buttons functionalities

Group 30: Som Lee, Trisentosa Wisesa
ECE:3360, 5 May 2022
Final Project Lab Report
Project Name: “Thomas not a Train”

b. **Hardware description**

Schematic

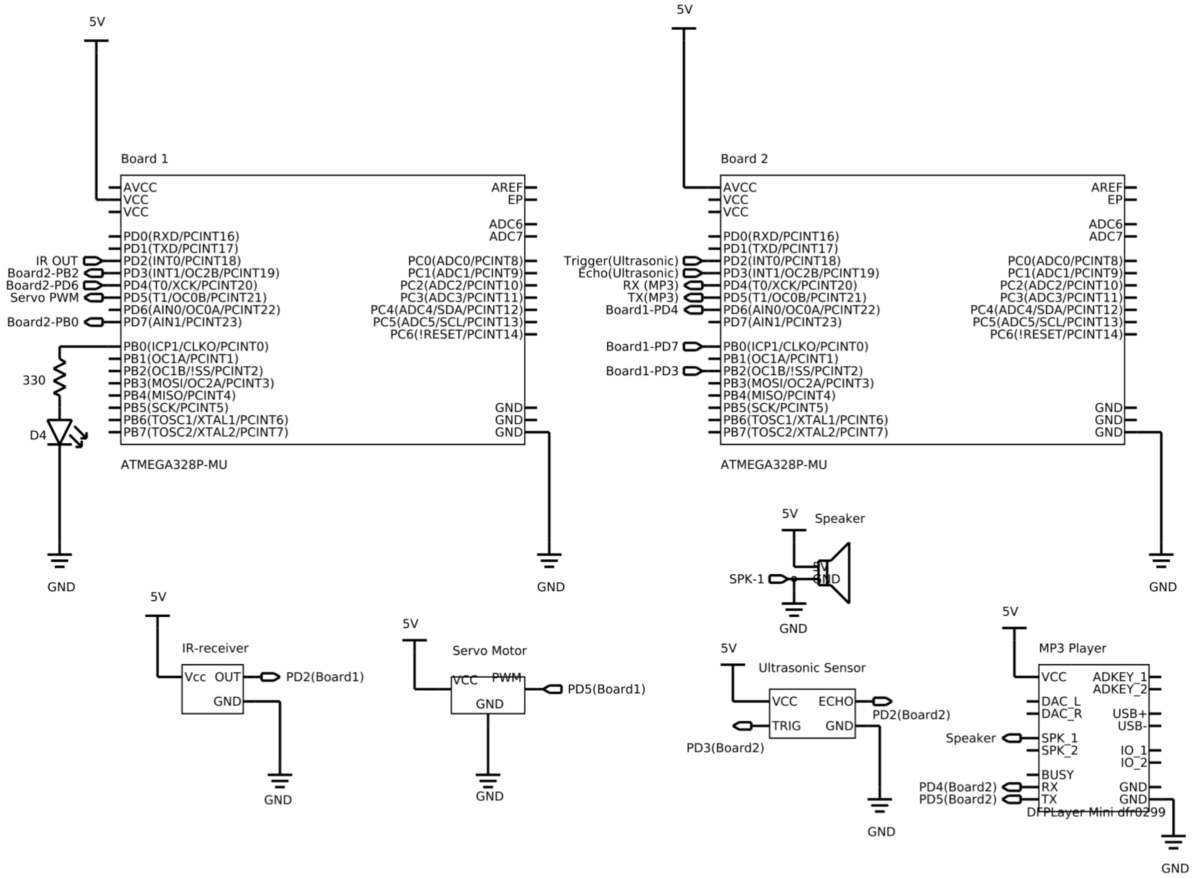

Figure 2: Hardware Schematics

<u>Component Selections:</u>

The components used in this projects are **Ultrasonic sensor (HC-1104), IR-receiver and transmitter set, Servo motor (SG90), MP3 module (DFPlayer mini), Microcontroller (Arduino Uno)**

Group 30: Som Lee, Trisentosa Wisesa
ECE:3360, 5 May 2022
Final Project Lab Report
Project Name: "Thomas not a Train"

IR communication :

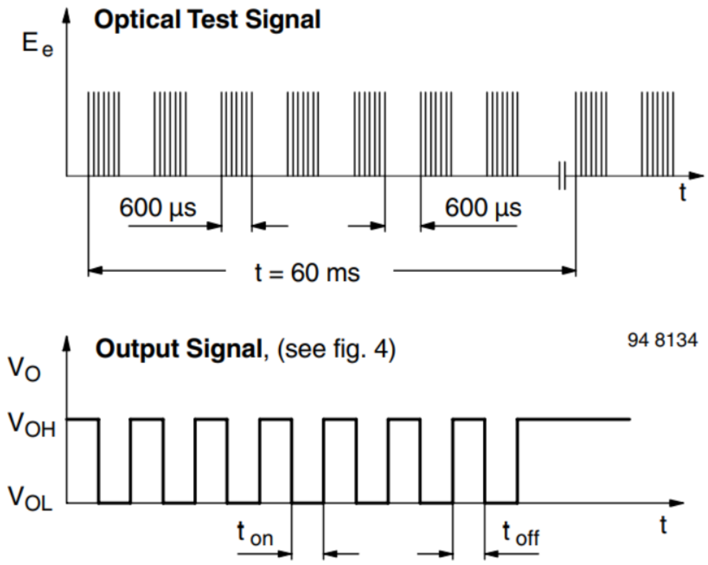

Figure 3: IR Output Signal

Our transmitter uses NEC methode, serieses of IR signals are the reference of each button, and the IR signals(top) are transmitted to the receiver. Then, the microcontroller has to modulate the signals received by the IR receiver (bottom) to be utilized as a digital signal. We used IR decoder library for this modulation to use the digital signal.

Servo motor :

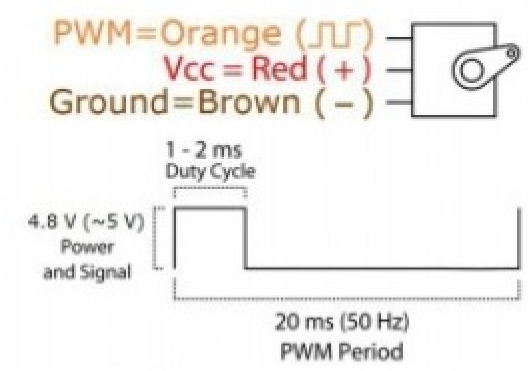

Figure 4: Servo Motor PWM

Servo motor is controlled by duty cycle of PWM signal, between 1~2 ms of 'on' signal set the angle of motor 0~180 degree, 5%~10% duty cycle within 20ms

(50Hz). Since we are using a 16 bit timer for the IR communication already, we used an 8 bit timer for the servo motor PWM.

```
void servo_init(){
	DDRD |= _BV(5); // pin 5 out

	TCCR0A = _BV(WGM00) | _BV(COM0B1) | _BV(COM0B0) ; // mode5 , set on
rising clear on falling

	TCCR0B = _BV(WGM02) | _BV(CS02) | _BV(CS00); //1024 prescalar

	OCR0A = 156; // 156*2=312=20ms period
	OCR0B = 136;
}
```

Figure 5: Snippet of servo motor initialization

We used mode 5 of the 8 bit timer and 1024 prescaler to achieve a 20ms period with 1ms~2ms duty cycle. However, the problem is, the servo can't move at very precise angles, since the timer and OCR range is small. About 15 degrees for each move. It moves like monsters in 80's movies.

Ultrasonic sensor :

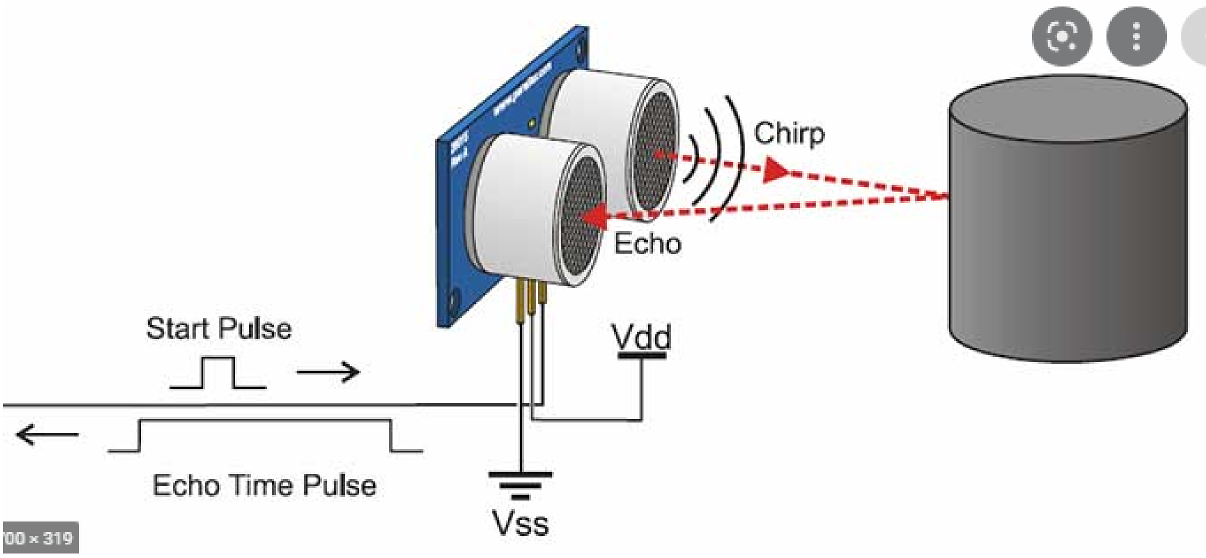

Figure 6: Ultrasonic sensor pulse wave and visualization

The ultrasonic sensor has four pins, Vin, GND, and trigger and echo.
Micro controller sends a single pulse as a start signal, then the sensor shoots ultrasonic wave to forward, and measures time until the reflected ultrasonic

wave is received back. Then the sensor sends the result as a digital signal through the Echo pin.
At first we tried to make it in C language, but the problem is, this function can't avoid using the 16 bit timer which is already in use for the IR receiver. We brought another Uno board. The next problem is, we couldn't find a proper library for the ultrasonic sensor, we tried to build it from the bottom, but it didn't work really well as we expected. We ended up using C++ language and the IDE library for this function. But one good thing for us, since we ended up using cpp as part of our project, we were able to try what we wanted such as MP3 player.

MP3 player :

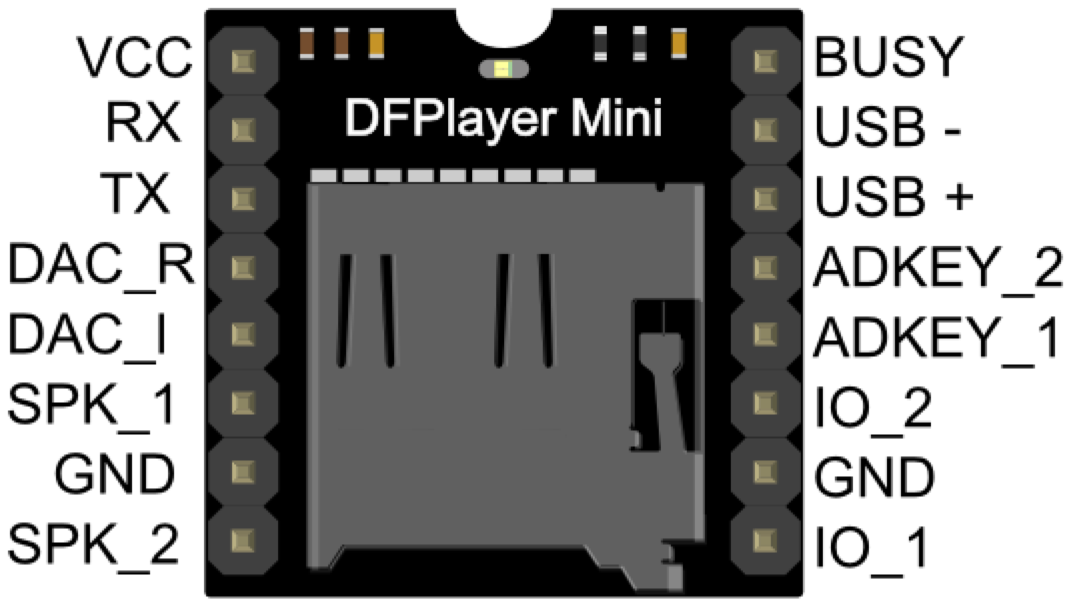

Figure7: MP3 Player Module

This is the MP3 player module we used for our project, Vin and ground and RX and TX and speaker pins are used. This is one type of UART communication, MP3 files are saved in the SD memory card slotted on the MP3 module, and the MP3 module receives the command signals from the controller through RX pin and translate the digital data in the MP3 file as analogue signal to play the music through speaker.

Group 30: Som Lee, Trisentosa Wisesa
ECE:3360, 5 May 2022
Final Project Lab Report
Project Name: “Thomas not a Train”

c. **Software description**

Flow Diagram

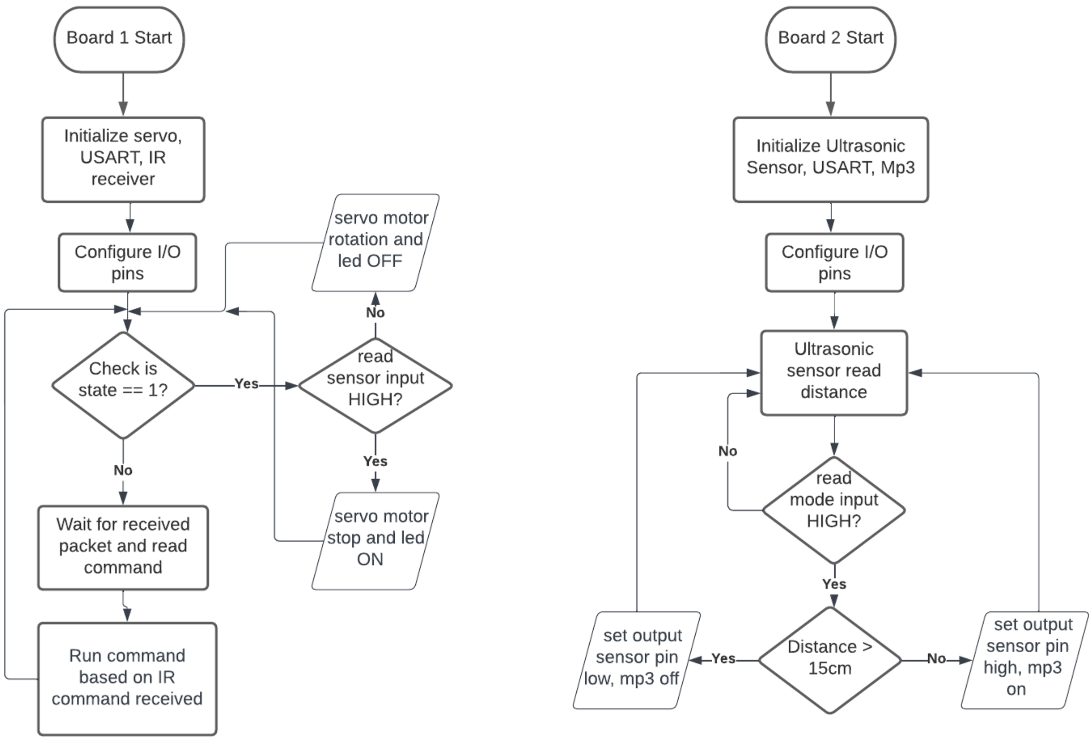

Figure 7: Software Flow Diagram

Major functions:

main(void): this function controls the main logic of the main board. The snippet below shows the loop part of the main function. It first checks for mode, it goes to servo_auto() mode if mode is auto, and will check for incoming packets from the IR remote if it is in control mode.

Group 30: Som Lee, Trisentosa Wisesa
ECE:3360, 5 May 2022
Final Project Lab Report
Project Name: "Thomas not a Train"

```
while (1) {
    cli();
    uint8_t check_result = check_new_packet(&received_packet);
    sei();

    while (state == 1)
    {
        servo_auto();
    }

    if (check_result)
    {
        //char buff[10];
        usart_transmit("\n\r");
        //utoa(received_packet.command, buff, 16);
        //usart_transmit(buff);
        currentCommand = received_packet.command;
        ir_actions(currentCommand);
    }else if(received_packet.repeat > 0){
        ir_actions(currentCommand);
    }
}
```

Figure 8: snippet of main function

servo_auto(): this function runs when the device is in auto mode. It will run a loop that will increment the OCR0B, this controls the rotation of the motor. Within each loop instruction, it checks for a new command and only accepts it if the command is the "mode button", and will return from that function back to main. Otherwise it will keep rotating, and check for the sensor input pin. If the sensor input pin is high, pause the rotation in place and light up the LED.

```
void servo_auto(){
    for (OCR0B = 136; OCR0B < 152; OCR0B++){//1ms to 2ms duty
        _delay_ms(100);
        uint8_t check_result = check_new_packet(&received_packet);
        //check for incoming signal, if signal is "MODE" button then stop
        if (check_result && received_packet.command == 0x46 && received_packet.repeat == 0)
        {
            usart_transmit("new");
            state = 0;
            PORTD |= 0 << PIND3;
            return;
        }

         //if get signal from sonic sensor, light up led and stay in place
        while((PIND & (1 << PIND4)) == (1 << PIND4)){
            PORTB |= 0B000001;
        }
        PORTB &= ~ 0B000001; // turn led back off
    }
```

Figure 9: snippet of servo_auto function

ir_actions(int commandCode): this function takes an integer, it will take the value of a command code from the received packet, and run the function here.

```
void ir_actions(int commandCode){
    switch (commandCode)
        {
            case 0x40:
            usart_transmit("LEFT");
            if (OCR0B > 136)
            {
                OCR0B = OCR0B -1;
            }
            break;
            case 0x43:
            usart_transmit("RIGHT");
            if (OCR0B < 152)
            {
                OCR0B = OCR0B +1;
            }
            break;
            case 0x44:
```

Figure 10: snippet of ir_actions function

3. Experimental Methods

The test procedure for this project is not much different than previous labs. We first tested each of the major components on the board individually. Then, once we got it working we tried to combine the parts together. One of the problems we faced is that three of our major components (servo motor, IR receiver, ultrasonic sensor) required a 16 bit timer to operate, meanwhile our microcontroller only had one. Since we were planning on using two, we ended up using an 8-bit timer to generate the PWM for our servo motor, with the drawback of a slightly less precision and smoothness in movement for the motor. A lot of the debugging is also done using the USART logging system we have made in lab 5. We use this method to debug where the bug is occurring as well as to read the command code of the IR remote.

4. Results

For the demonstration, we have put the video in our project video submission. We didn't put it here because the video file might be too large.

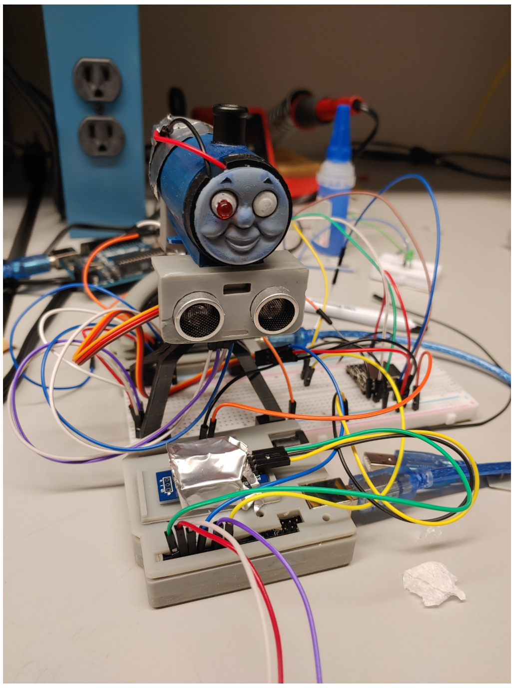

Figure 11: Picture of Thomas

5. Discussion of Results

Overall, the device performs more or less what we were expecting and reached the goal we have set in our project draft submission. However, there is definitely room for improvement. One of the limitations of the project is the responsiveness of the IR sensor when detecting a switch from auto trigger mode to manual mode. This is due to the delay between reading and each rotation of the servo motor. The other limitation is the slight delay after ultrasonic sensors detect an obstacle to a halt. For both of these limitations, an improvement in debouncing (sensor) and handling of IR ISR in the software side can be expected. For our ideas of improvement, a better construction and wiring of the device, as well as more commands for the ir transmitter such as turning on/off mist maker and adding other modes.

6. Conclusion

In this project, we have learned and successfully implemented a rotating "turret" that is controlled by an IR remote, and includes more components such as a servo motor for the body rotation and ultrasonic sensor to measure the distance. This main purpose of the device is to gain even more experience in c programming and on the components we haven't used before. Although it is not the most useful tool, we can definitely use and reference the technologies learned in this project later in our work for something that is substantially more useful.

7. Acknowledgements

We would like to express my gratitude and appreciation toward everyone who gave us the opportunity to complete this project. We would specifically express our thanks to professor Reinhard Beichel for teaching the Embedded System course this semester and the TAs; Dean Farewell, Rys P. Huehnergarth, and Maxwell V. McIntyre for helping and guiding us in this project as well as the previous labs.

8. References

a. Datasheets

- ATMega328P - https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf
- Servo Motor SG90 - http://www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf
- Ultrasonic Sensor HCSR04 - https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf
- IR Receiver - https://www.epitran.it/ebayDrive/datasheet/45.pdf

b. Software Libraries used

- IR-Receiver NEC decoding - https://github.com/biletnikov/avr-nec-ir-decoder

9. Appendix : Source Code

main.c (board 1)

```
// CPU Frequency -> 16 MHz
#ifndef F_CPU
#define F_CPU 16000000UL
#endif

//Set Baudrate
#define USART_BAUDRATE 9600
#define UBRR_VALUE (((F_CPU / (USART_BAUDRATE * 16UL))) - 1)

//Imports
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <avr/pgmspace.h>
#include <util/atomic.h>
#include <avr/power.h>

#include "IR_Receiver.h"

//Functions declarations

void usart_init();
unsigned char usart_receives();
void usart_transmit(char arr[]);
void removeGarbage();
void ir_actions(int commandCode);
void servo_init();
void servo_auto();

//state initialization
int state = 0; //0 = manual, 1 = auto
struct IR_Packet received_packet;

int mist = 0;
int mp3 = 0;
```

```
int main(void)
{
	int currentCommand;

	//servo initialization
	DDRB |= 1 << PINB1; // Set pin 9 on arduino to output

	//ultrasonic sensor signal
	DDRD |= 0 << PIND4;
	PORTD |= 0 << PIND4;

	// Output
	DDRD |= 1 << PIND3; // mode
	PORTD |= 0 << PIND3;
	DDRD |= 1 << PIND6; // mist maker
	PORTD |= 0 << PIND6;
	DDRD |= 1 << PIND7; // mp3 player
	PORTD |= 0 << PIND7;

	DDRB |= 0B000001; // PORTB5, LED

	//usart, receiver initialization
	usart_init();
	sei();
	init_receiver();
	servo_init();

	usart_transmit("START\n");
	while (1) {
		cli();
		uint8_t check_result = check_new_packet(&received_packet);
		sei();

		while (state == 1)
		{
			servo_auto();
		}

		if (check_result)
		{
			//char buff[10];
			usart_transmit("\n\r");
			//utoa(received_packet.command, buff, 16);
			//usart_transmit(buff);
			currentCommand = received_packet.command;
```

```
                    ir_actions(currentCommand);
                }
        }
        return 0;
}

void usart_init() {
        UBRR0H = (unsigned char)(UBRR_VALUE>>8);
        UBRR0L = (unsigned char)UBRR_VALUE;
        // Set frame format to 8 data bits, no parity, 2 start bit, 1 stop bit (11 bits)
        UCSR0C |= (1<<UCSZ01)|(1<<UCSZ00);
        //enable transmission and reception interrupt
        UCSR0B |= (1<<RXEN0)|(1<<TXEN0);
}

unsigned char usart_receives() {
        // Wait for byte to be received
        while(!(UCSR0A&(1<<RXC0))){};
        // Return received data
        return UDR0; // get char from buffer
}

void usart_transmit(char arr[]) {
        int i;
        for (i = 0; i <= strlen(arr)-1; i++) {
                while (!( UCSR0A & (1<<UDRE0))) {}; // wait until buffer empty
                UDR0 = arr[i];
        }
}

void removeGarbage(){
        unsigned char x = usart_receives();
        while(x != 0x0A){
                x = usart_receives();
        }
}

void servo_init(){
        DDRD |= _BV(5); // pin 5 out

        TCCR0A = _BV(WGM00) | _BV(COM0B1) | _BV(COM0B0) ; // mode5 , set on
rising clear on falling

        TCCR0B = _BV(WGM02) | _BV(CS02) | _BV(CS00); //1024 prescalar
```

```
        OCR0A = 156; // 156*2=312=20ms period
        OCR0B = 136;
}

void servo_auto(){
                for (OCR0B = 136; OCR0B < 152; OCR0B++){//1ms to 2ms duty
                        _delay_ms(100);
                        uint8_t check_result = check_new_packet(&received_packet);
                        //check for incoming signal, if signal is "MODE" button then
stop
                        if (check_result && received_packet.command == 0x46 &&
received_packet.repeat == 0)
                        {
                                usart_transmit("new");
                                state = 0;
                                PORTD |= 0 << PIND3;
                                return;
                        }

                         //if get signal from sonic sensor, light up led and stay in place
                        while((PIND & (1 << PIND4)) == (1 << PIND4)){
                                PORTB |= 0B000001;
                        }
                        PORTB &= ~ 0B000001; // turn led back off
                }

                for (OCR0B = 152; OCR0B > 136; OCR0B--){//2ms to 1ms duty
                        _delay_ms(100);
                        uint8_t check_result = check_new_packet(&received_packet);
                        //check for incoming signal, if signal is "MODE" button then
stop
                        if (check_result && received_packet.command == 0x46 &&
received_packet.repeat == 0)
                        {
                                usart_transmit("new");
                                state = 0;
                                PORTD |= 0 << PIND3;
                                return;
                        }

                        // if get signal from sonic sensor, light up led and stay in place
                        while((PIND & (1 << PIND4)) == (1 << PIND4)){
                                PORTB |= 0B000001;
                        }
                        PORTB &= ~ 0B000001; // turn led back off
```

```
		}
}

void ir_actions(int commandCode){
	switch (commandCode)
		{
			case 0x45:
			usart_transmit("Mist Maker");
			if (mist == 1)
			{
				PORTD |= 0 << PIND6;
				mist = 0;
			}else{
				PORTD |= 1 << PIND6;
				mist = 1;
			}
			break;
			case 0x46:
			usart_transmit("SWITCH MODE");
			if (received_packet.repeat == 0)
			{
				if (state == 0)
				{
					state = 1;
					PORTD |= 1 << PIND3;
				}else{
					state = 0;
					PORTD |= 0 << PIND3;
				}
			}
			break;
			case 0x47:
			usart_transmit("MP3 Player");
			if (mp3 == 1)
			{
				PORTD |= 0 << PIND7;
				mp3 = 0;
			}else if (mp3 == 0){
				PORTD |= 1 << PIND7;
				mp3 = 1;
			}
			break;
			case 0x40:
			usart_transmit("LEFT");
			if (OCR0B > 136)
```

```
                {
                        OCR0B = OCR0B -1;
                }
                break;
                case 0x43:
                usart_transmit("RIGHT");
                if (OCR0B < 152)
                {
                        OCR0B = OCR0B +1;
                }
                break;
                case 0x44:
                usart_transmit("SHOOT");
                // turn LED on
                PORTB |= 0B000001; // PORTB5
                _delay_ms(100);

                // turn LED off
                // PORTB5
                PORTB &= ~ 0B000001;
                _delay_ms(100);
                break;
                case 0x09:
                usart_transmit("reset right");
                OCR0B = 152;
                break;
                case 0x15:
                usart_transmit("reset left");
                OCR0B = 136;
                break;
        }
}
```

main.ino(board 2)

```
#include <SoftwareSerial.h>
#include <DFRobotDFPlayerMini.h>
SoftwareSerial MP3Module(4, 5);
DFRobotDFPlayerMini MP3Player;
#define echoPin 2
#define trigPin 3 //sonic sensor pins
#define mode 10 // mode selection
```

```
#define sonicpin 6 // c
#define mpcont 8  // input to control mp3 on/off

long duration; // variable for the duration of sound wave travel
int distance; // variable for the distance measurement
int adv = 3;
int save;
int count;

void setup () {

  Serial.begin (9600);
  MP3Module.begin(9600);
  if (!MP3Player.begin(MP3Module)) { //mp3 module check
    Serial.println(F("MP3 failed"));
    while (true);


  }
  delay(1);
  pinMode(trigPin, OUTPUT); // Set the trigPin as an output
  pinMode(echoPin, INPUT); // Set the echoPin as an input
  pinMode(mpcont, INPUT_PULLUP); // pin between boards

  pinMode(mode, INPUT_PULLUP);
  pinMode(sonicpin, OUTPUT);
  Serial.begin(9600); // baudrate speed

  MP3Player.volume(10);  //mp3 volume
  MP3Player.loop(1);


}

void loop () {

  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH);

  distance = duration * 0.034 / 2;
```

```
  if(distance<800){
    save = save + distance;
    count = count + 1;
  }
  if(count>5){
    count = 0;
    save = 0;
  }
  Serial.print(distance);
  Serial.println(" cm");

  if(distance>15){
   MP3Player.stopAdvertise();
   adv = 3;
   digitalWrite(sonicpin, LOW);
 }
 if(distance<15){
  if(adv==3){
   MP3Player.advertise(1);
   adv = 2;
  }
  digitalWrite(sonicpin, HIGH);
  }

 if(digitalRead(mpcont)==LOW){
   MP3Player.volume(0);
 }
 if(digitalRead(mpcont)==HIGH){
   MP3Player.volume(10);
 }
}
```