RoboVR

SRB WRESTLING (H)

About the game:

In this game, the robot has to kick out the opponent out of the arena. The one who scores more points wins the game. The robot has to be humanoid (i.e. with two hand sand legs). The specifications and components to build the robot are mentioned below.

Components and its Specifications:

Sr. No.	Components	Specifications
1.	Arduino nano - x1	Microcontroller ATmega328
2.	Servo Motor - x16	RKI 2122 OR successor
3.	PCA9685 Servo driver	-
4.	PS2 Wired / Wireless Controller	-
5.	Battery (x2)	i. 9v (for Arduino) ii. LiPo 5v (2200 mAh 1S 25C/50C)
6.	Jumper wires	M-M, M-F
7.	Metal Horn for Servo 25T	-
8.	Metal Gear Standard Servo (or) Servo Economy - Servo Motor	-
9.	Multipurpose Aluminium Standard Servo Bracket	-
10.	Short U Shape Aluminium Servo Bracket	-
11.	Long U Aluminium Servo Bracket	-
12.	L Shaped Interconnect Servo Bracket	-
13.	Large U Beam Aluminium Servo Bracket	-
14.	Robot feet Aluminium Servo Bracket	-
15.	Oblique U Shape Servo Bracket	-
17.	Interconnect Servo Bracket	-

Robot Details:

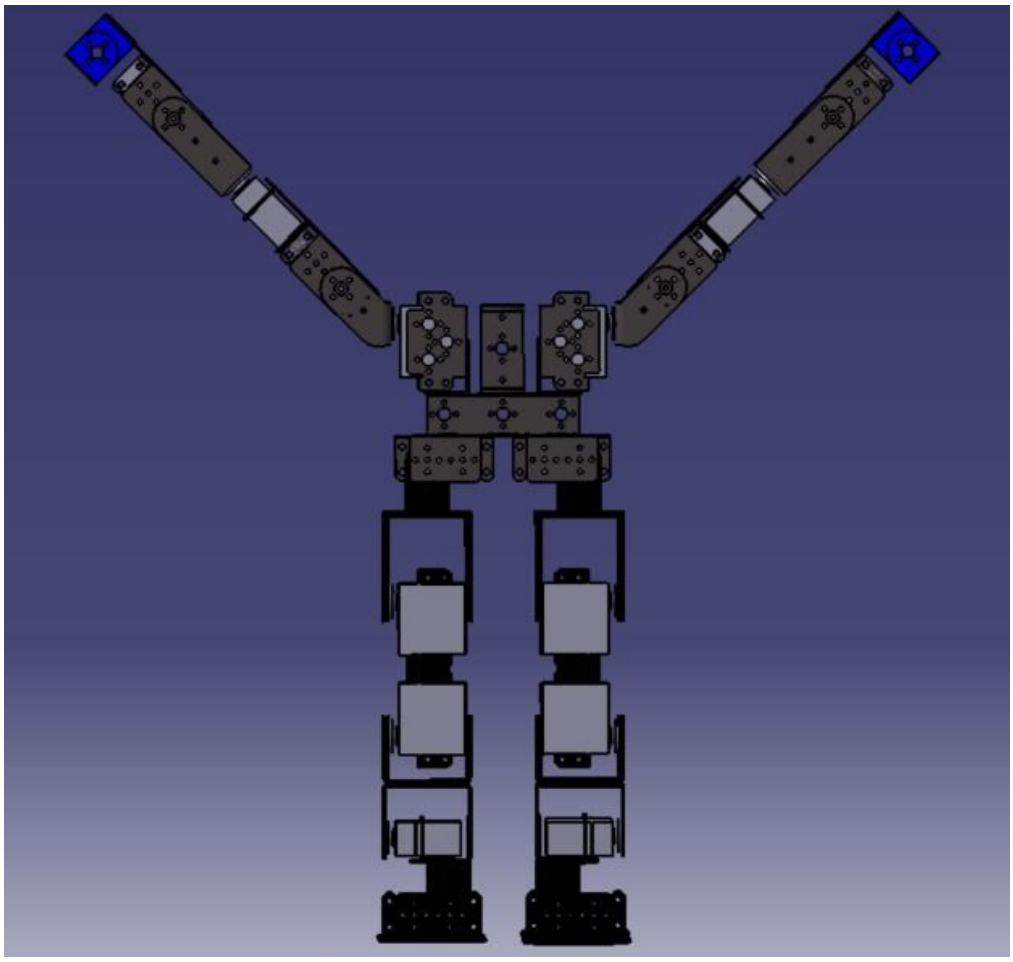
Robot Dimensions: height 15”

Robot Control: Wireless

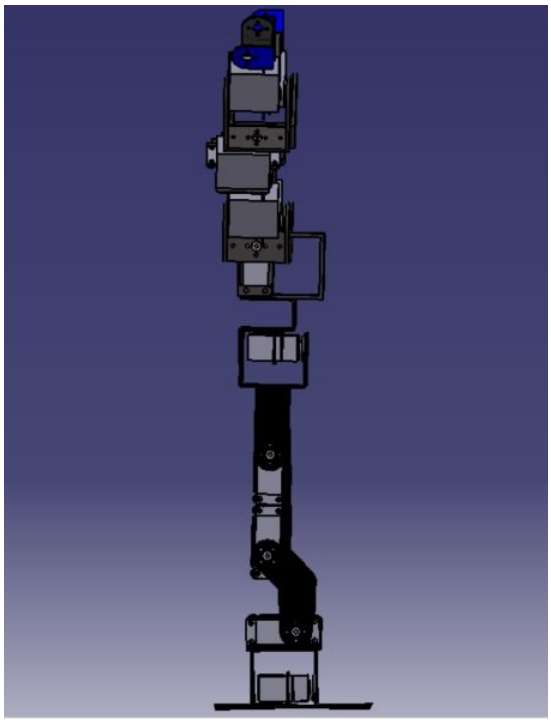
Robot Type: Biped Robot

Mechanical Design:

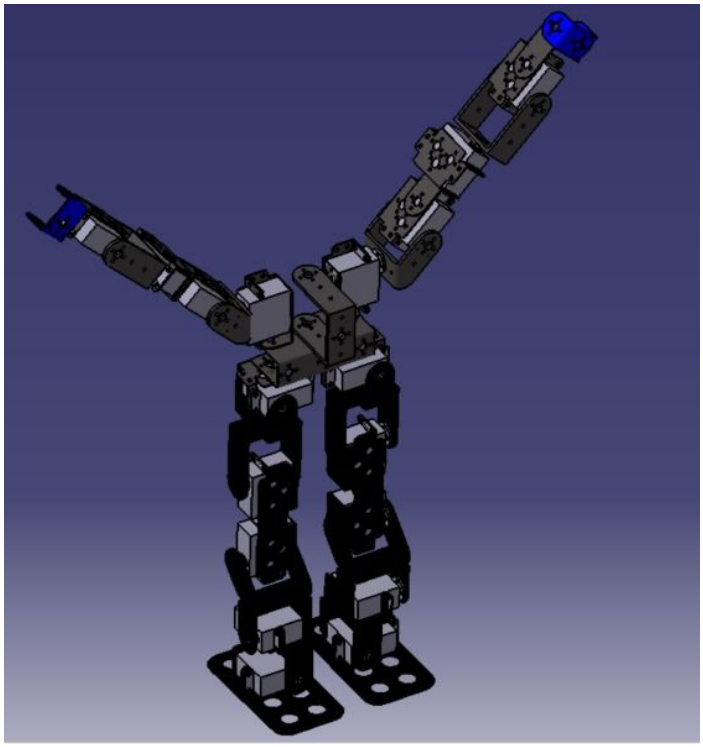
Front View



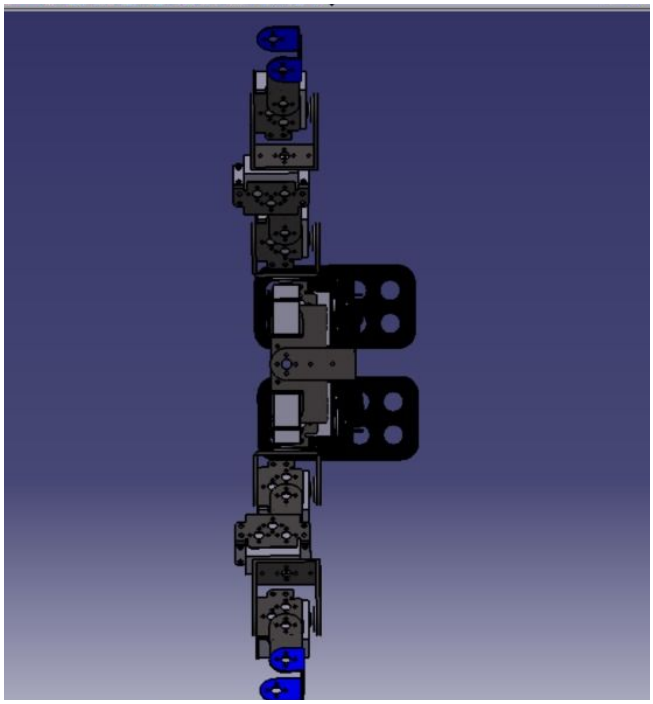
Side View



Isometric View



Bottom View



Working:

16 Servo motors control movement of the bot's arm and leg. 6 Servo motors are for arms mechanism and rest of the 10 servo helps in body and leg motion.

The PCA9685 Servo driver provides power and controls the motion of each servo which is further controlled by an Arduino Nano. Ones the proper code (given below) for the motion of each servo is uploaded in the Arduino, the entire bot can be operated with the PS2 Controller which is also connected to the Arduino wired or wirelessly.

The hooks are provided on the arms to wear the gloves. Metal horns and aluminum brackets are used to construct the body of the Robot.

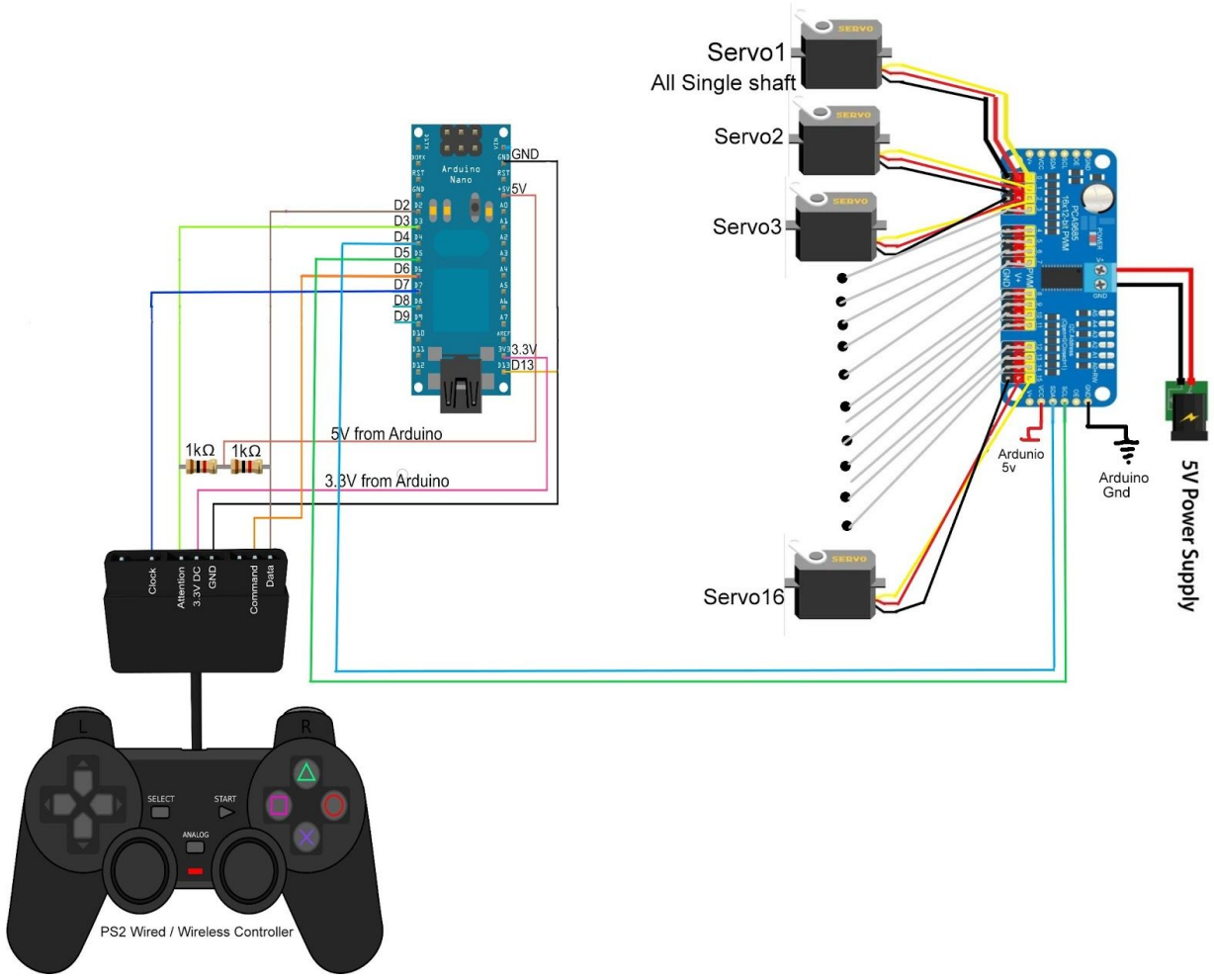
You can use the Arduino USB-Bluetooth Servo Controller software to control the Maximum 18 DOF's servo robot, which will automatically produce the Arduino code to upload in the microcontroller.

Connect the servo motors as below:

- **S1** = Left leg foot
- **S2** = Left leg ankle
- **S3** = Left leg knee
- **S4** = Left leg tight
- **S5** = Left leg pelvis
- **S8** = Left arm hand
- **S9** = Left arm elbow
- **S10** = Left arm shoulder
- **S11** = Head
- **S12** = Right arm shoulder
- **S13** = Right arm elbow
- **S14** = Right arm hand
- **S16** = Right leg pelvis
- **S17** = Right leg tight
- **S18** = Right leg knee
- **S19** = Right leg ankle
- **S20** = Right leg foot

The schematic for the servo motion and connecting the PS2 controller is given below:

Connections:



Codes:

Servo testing Code:

```
#include <SoftwareSerial.h>;
SoftwareSerial usc(19,18); //defining usc32 and rx/tx pins

#include <Vcc.h>
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <ChainableLED.h>
//#include <p9813.h>
#include <ADXL345.h>
#include "ds3231.h"
#include <inttypes.h>
#include <lm75.h>
//Battery Level Indicator Definitions

const float VccMin   = 0.0;           // Minimum expected Vcc level, in Volts.

const float VccMax   = 5.0;           // Maximum expected Vcc level, in Volts.

const float VccCorrection = 1.0/1.0;  // Measured Vcc by multimeter divided by
reported Vcc

Vcc vcc(VccCorrection);
```

```
//Definitions for SSD1306 128x64 OLED Display

#define OLED_RESET                    7

#define OLED_SA0                      8

//Definitions for P9813 RGB LED Driver

#define no_of_LEDs                    1

#define data_pin                      5

#define clk_pin                       6

#define rgb_pwr                       1

//Definitions for Joystick

#define KEY_UP                        A1

#define KEY_DOWN                      A5
```

```
#define KEY_LEFT                        A3

#define KEY_RIGHT                       A2

#define KEY_ENTER                       A4


//Clock Stuffs


#define CLOCK_CENTER_X                  24

#define CLOCK_CENTER_Y                  40

#define H_LENGTH                        10

#define M_LENGTH                        14

#define S_LENGTH                        18


#define buzzer                          11


#define rad_per_degree                0.0174556


Adafruit_SSD1306 oled(OLED_RESET, OLED_SA0);
```

```
ChainableLED RGB_LED(clk_pin, data_pin, rgb_pwr, no_of_LEDs);

TempI2C_LM75 thermo = TempI2C_LM75(0x48, TempI2C_LM75::nine_bits);

ADXL345 accelerometer;



boolean tgl = false;

boolean settings = false;


signed char parameter = 0;

unsigned char max_b = 127;


float t_p = 0.0;

float t_o = -255.0;



struct ts T;



void showBatteryLevel2()
```

```
{
float p = vcc.Read_Perc(VccMin, VccMax);


oled.fillRect(2, 20, 50, 60, BLACK);
oled.setTextColor(WHITE);
oled.setCursor(2, 28);
oled.println("Battery");
oled.setCursor(2, 40);
oled.print(p); //print the voltage to oled
oled.print(" %");
 if (p < 20) //set the voltage considered low battery here
 {
oled.setCursor(25, 40);
// oled.setTextColor(YELLOW);
oled.print("!!!");
oled.setTextColor(WHITE);
 }
```

```
 Serial.print("VCC = ");
 Serial.print(p);
 Serial.println(" %");

}



//read internal voltage
long readVcc() {
 long result;
 // Read 1.1V reference against AVcc
 ADMUX = _BV(REFS0) | _BV(MUX3) | _BV(MUX2) | _BV(MUX1);
 delay(2); // Wait for Vref to settle
 ADCSRA |= _BV(ADSC); // Convert
 while (bit_is_set(ADCSRA, ADSC));
 result = ADCL;
```

```
 result |= ADCH << 8;
 result = 1126400L / result; // Back-calculate AVcc in mV
 return result;
}


void printVolts()
{
oled.fillRect(2, 20, 50, 60, BLACK);
oled.setTextColor(WHITE);
// int sensorValue = analogRead(A0); //read the A0 pin value
// float voltage = sensorValue * (5.00 / 1023.00) * 2; //convert the value to
a true voltage.
 double voltage = double( readVcc() ) / 1000;
 //float voltage = readVcc();
 oled.setCursor(2, 28);
 oled.println("Battery");
 oled.setCursor(2, 40);
 oled.print("=");
 oled.print(voltage); //print the voltage to oled
```

```
 oled.print("v");
 if (voltage < 6.50) //set the voltage considered low battery here
 {
  // digitalWrite(led_pin, HIGH);
 }
}


void draw_background()
{
   unsigned char i = 0;


/*    oled.drawRect(2, 18, 44, 44, WHITE);
   oled.drawLine(24, 19, 24, 20, WHITE);
   oled.drawLine(24, 60, 24, 61, WHITE);
   oled.drawLine(3, 40, 4, 40, WHITE);
   oled.drawLine(44, 40, 45, 40, WHITE);


   oled.fillCircle(106, 58, 4, WHITE);
   oled.drawFastVLine(104, 18, 37, WHITE);
```

```
    oled.drawFastVLine(108, 18, 37, WHITE);
    oled.drawFastHLine(105, 17, 3, WHITE);
*/
    for(i = 18; i <= 53; i += 5)
    {
        oled.drawFastHLine(111, i, 3, WHITE);
    }

    oled.setTextSize(1);
    oled.setTextColor(WHITE);
    for(i = 0; i < 40; i += 10)
    {
        oled.setCursor(116, (48 - i));
        oled.println(i + 10);
    }

    oled.setCursor(12, 0);
    oled.println("Time");
```

```
    oled.setCursor(59, 0);
    oled.println("Date");
    oled.setCursor(96, 0);
    oled.println("Temp.");

    oled.display();
}



float get_T_avg()
{
    unsigned char samples = 20;
    float avg = 0.0;

    while(samples > 0)
    {
        avg += thermo.getTemp();
        delayMicroseconds(100);
        samples--;
```

```
    }


    avg /= 20.0;
    return avg;
}



void show_temperature()
{
    unsigned char bar_length = 0;

    t_p = get_T_avg();

    if(t_p != t_o)
    {
        oled.fillRect(98, 8, 29, 8, BLACK);
        oled.setTextColor(WHITE);
        oled.setCursor(98, 9);
```

```
        oled.println(t_p);



        t_p = constrain(t_p, 0, 40.0);

        bar_length = map(t_p, 0.0, 40.0, 0, 36);

        oled.drawLine(106, 53, 106, 18, BLACK);

        oled.drawLine(106, 53, 106, (53 - bar_length), WHITE);

        t_o = t_p;

    }

}




void show_time()

{

    static unsigned long previous_time;

    static unsigned long present_time;



    oled.fillRect(0, 8, 50, 8, BLACK);



    if(T.hour < 10)
```

```
{
    oled.setCursor(0, 9);
    oled.println("0");
    oled.setCursor(6, 9);
    oled.println(T.hour);
}
else
{
    oled.setCursor(0, 9);
    oled.println(T.hour);
}

if(T.min < 10)
{
    oled.setCursor(18, 9);
    oled.println("0");
    oled.setCursor(24, 9);
    oled.println(T.min);
```

```
}
else
{
    oled.setCursor(18, 9);
    oled.println(T.min);
}

if(T.sec < 10)
{
    oled.setCursor(36, 9);
    oled.println("0");
    oled.setCursor(42, 9);
    oled.println(T.sec);
}
else
{
    oled.setCursor(36, 9);
    oled.println(T.sec);
}
```

```
present_time = millis();
if((present_time - previous_time) > 999)
{
    tgl ^= 1;
    previous_time = present_time;
}

if(tgl)
{
    oled.setCursor(12, 9);
    oled.println(":");
    oled.setCursor(30, 9);
    oled.println(":");
}
else
{
    oled.setCursor(12, 9);
```

```
        oled.println(" ");
        oled.setCursor(30, 9);
        oled.println(" ");
    }


 //  display_analog_clock(T.hour, T.min, T.sec);
}



void show_date()
{
    oled.fillRect(48, 16, 54, 37, BLACK);

    if(T.mday < 10)
    {
        oled.setCursor(60, 18);
        oled.println("0");
        oled.setCursor(66, 18);
        oled.println(T.mday);
```

```
}
else
{
    oled.setCursor(60, 18);
    oled.println(T.mday);
}

oled.setCursor(72, 18);
oled.println("/");

if(T.mon < 10)
{
    oled.setCursor(78, 18);
    oled.println("0");
    oled.setCursor(84, 18);
    oled.println(T.mon);
}
else
```

```
{
    oled.setCursor(78, 18);
    oled.println(T.mon);
}

oled.setCursor(62, 32);
oled.println(T.year);

oled.setCursor(49, 46);
switch(T.wday)
{
    case 1:
    {
        oled.println(" Monday  ");
        break;
    }
    case 2:
    {
        oled.println(" Tuesday ");
```

```
        break;
    }
    case 3:
    {
        oled.println("Wednesday");
        break;
    }
    case 4:
    {
        oled.println("Thursday ");
        break;
    }
    case 5:
    {
        oled.println(" Friday  ");
        break;
    }
    case 6:
```

```
        {
            oled.println("Saturday ");
            break;
        }
        default:
        {
            oled.println(" Sunday  ");
            break;
        }
    }
}



void set_time_and_date()
{
   if(digitalRead(KEY_ENTER) == LOW)
   {
        digitalWrite(buzzer, HIGH);
        delay(20);
```

```
    digitalWrite(buzzer, LOW);
    while(digitalRead(KEY_ENTER) == LOW);
    settings = true;
    parameter = 1;
}


if(settings == true)
{
    switch(parameter)
    {
        case 1:
        {
            RGB_LED.setColorRGB(0, 255, 0, 0);
            T.hour = inc_dec(T.hour, 23, 0);
            break;
        }
        case 2:
        {
```

```
        RGB_LED.setColorRGB(0, 0, 255, 0);
        T.min = inc_dec(T.min, 59, 0);
        break;
    }
    case 3:
    {
        RGB_LED.setColorRGB(0, 0, 0, 255);
        T.sec = inc_dec(T.sec, 59, 0);
        break;
    }
    case 4:
    {
        RGB_LED.setColorRGB(0, 127, 127, 0);
        T.mday = inc_dec(T.mday, 31, 1);
        break;
    }
    case 5:
    {
        RGB_LED.setColorRGB(0, 0, 127, 127);
```

```
        T.mon = inc_dec(T.mon, 12, 1);
        break;
    }
    case 6:
    {
        RGB_LED.setColorRGB(0, 127, 0, 127);
        T.year = inc_dec(T.year, 2100, 1980);
        break;
    }
    case 7:
    {
        RGB_LED.setColorRGB(0, 127, 127, 127);
        T.wday = inc_dec(T.wday, 6, 0);
        break;
    }
    default:
    {
        DS3231_set(T);
```

```
                settings = false;
                break;
            }
        }
    }
}



void clock_stuffs()
{
    if(settings == false)
    {
        DS3231_get(&T);
    }
    show_time();
    show_date();
    set_time_and_date();
}
```

```
void display_analog_clock(signed int h, signed int m, signed int s)
{
   float midHours = 0;
   static signed char hx;
   static signed char hy;
   static signed char mx;
   static signed char my;
   static signed char sx;
   static signed char sy;

   h -= 3;
   m -= 15;
   s -= 15;

   if(h <= 0)
   {
       h += 12;
```

```
  }
  if(m < 0)
  {
      m += 60;
  }
  if(s < 0)
  {
      s += 60;
  }
```

```
  oled.drawLine(CLOCK_CENTER_X, CLOCK_CENTER_Y, (CLOCK_CENTER_X + sx),
(CLOCK_CENTER_Y + sy), BLACK);
  oled.drawLine(CLOCK_CENTER_X, CLOCK_CENTER_Y, (CLOCK_CENTER_X + mx),
(CLOCK_CENTER_Y + my), BLACK);
  oled.drawLine(CLOCK_CENTER_X, CLOCK_CENTER_Y, (CLOCK_CENTER_X + hx),
(CLOCK_CENTER_Y + hy), BLACK);
```

```
  s *= 6;
  sx = (S_LENGTH * cos(s * rad_per_degree));
  sy = (S_LENGTH * sin(s * rad_per_degree));
```

```
   oled.drawLine(CLOCK_CENTER_X, CLOCK_CENTER_Y, (CLOCK_CENTER_X + sx),
(CLOCK_CENTER_Y + sy), WHITE);




   m *= 6;

   mx = (M_LENGTH * cos(m * rad_per_degree));

   my = (M_LENGTH * sin(m * rad_per_degree));

   oled.drawLine(CLOCK_CENTER_X, CLOCK_CENTER_Y, (CLOCK_CENTER_X + mx),
(CLOCK_CENTER_Y + my), WHITE);




   midHours = (T.min / 12);

   h *= 5;

   h += midHours;

   h *= 6;

   hx = (H_LENGTH * cos(h * rad_per_degree));

   hy = (H_LENGTH * sin(h * rad_per_degree));

   oled.drawLine(CLOCK_CENTER_X, CLOCK_CENTER_Y, (CLOCK_CENTER_X + hx),
(CLOCK_CENTER_Y + hy), WHITE);

}




void accelerometer_and_RGB_LED()
```

```
{
   unsigned char r = 0x00;
   unsigned char g = 0x00;
   unsigned char b = 0x00;

   Vector norm = accelerometer.readNormalize();
   r = map(norm.XAxis, -11, 11, 0, max_b);
   g = map(norm.YAxis, -11, 11, 0, max_b);
   b = map(norm.ZAxis, -11, 11, 0, max_b);

   if(settings == false)
   {
     RGB_LED.setColorRGB(0, r, g, b);
   }
}
```

```
signed int inc_dec(signed int value, signed int max_value, signed int
min_value)
```

```
{
    if(digitalRead(KEY_UP) == LOW)
    {
        digitalWrite(buzzer, HIGH);
        delay(20);
        digitalWrite(buzzer, LOW);
        value++;
    }
    if(value > max_value)
    {
        value = min_value;
    }

    if(digitalRead(KEY_DOWN) == LOW)
    {
        digitalWrite(buzzer, HIGH);
        delay(20);
        digitalWrite(buzzer, LOW);
        value--;
```

```
}
if(value < min_value)
{
    value = max_value;
}

if(digitalRead(KEY_RIGHT) == LOW)
{
    digitalWrite(buzzer, HIGH);
    delay(20);
    digitalWrite(buzzer, LOW);
    while(digitalRead(KEY_RIGHT) == LOW);
    parameter++;
}
if(digitalRead(KEY_LEFT) == LOW)
{
    digitalWrite(buzzer, HIGH);
    delay(20);
    digitalWrite(buzzer, LOW);
```

```
        while(digitalRead(KEY_LEFT) == LOW);
        parameter--;
    }
    if((parameter > 7) || (parameter < 1))
    {
        parameter = 0;
    }

    return value;
}
void set_RGB_LED_max_brightness()
{
    unsigned int avg = 0;
    unsigned char samples = 16;
    while(samples > 0)
    {
        avg += analogRead(A0);
        delayMicroseconds(10);
```

```
        samples--;
    };
    max_b = (avg >> 6);
}


void transmit_data()
{
    if(tgl)
    {
        Serial.print(T.mday);
        Serial.print("/");
        Serial.print(T.mon);
        Serial.print(".");
        Serial.print(T.year);
        Serial.print("    ");
        Serial.print(T.hour);
        Serial.print(".");
        Serial.print(T.min);
        Serial.print(".");
```

```
        Serial.print(T.sec);
        Serial.print("    ");
        Serial.print("T/'C: ");
        Serial.println(t_p);
    }
}
void setup()
{
    unsigned char i = 0;
    Serial.begin(9600);
    Serial.flush();

    oled.begin(SSD1306_SWITCHCAPVCC, 0x3C);
    oled.clearDisplay();
    RGB_LED.pwr_set(PWR_ENABLE);
    pinMode(buzzer, OUTPUT);
    digitalWrite(buzzer, LOW);


    DS3231_init(DS3231_INTCN);
```

```
   for(i = A1; i <= A5; i++)
   {
       pinMode(i, INPUT_PULLUP);
   }
   draw_background();
 usc.begin(9600); //usc32 initialization
 usc.println("#12P1800T2000"); // move servo 12 at position 510 in 2 secs
 delay(2000);
}
void updateLcd()
{
   set_RGB_LED_max_brightness();
   show_temperature();
   printVolts();
   showBatteryLevel2();
   clock_stuffs();
   accelerometer_and_RGB_LED();
   transmit_data();
   oled.display();
```

```
}

void loop()

{

   updateLcd          ();
usc.println("#12P1800T2000"); // move servo 12 at position 510 in 2 secs

   delay(1000);

   updateLcd();

   usc.println("#12P1000T1000"); // move servo 12 at position 510 in 2 secs

   delay(2000);

   updateLcd();

   usc.println("#12P1800T1000"); // move servo 12 at position 2400 in 2 secs

   delay(1000);

}
```