Table of Contents

Table of Figures

Table of Tables

I. Introduction

Exploring the world can be dangerous and rough such as in war fields and rugged terrains, which are hard to access with normal vehicles. The objective of this project was to create a robot that has the ability to navigate terrain that is impassable for most other land vehicles. After researching and studying many robot designs, a spider robot style was chosen due to the natural movement of a spider and its capability of moving around regardless of the terrain it's facing. In this design, the Spider Robot legs have the same freedom of movement as those of a biological spider.

The Spider Robot could be used for:

- Exploring dangerous and / or rough areas for humans
- Exploring war zones
- Inspecting unstable buildings after natural disasters such as an earthquake
- Defusing bombs such as land mines

II. Background

Using robots nowadays is very common, and people use them in many applications. The idea here was to have a robot that can be used by individuals to reach their desired goals and to make it easy for them to control. To reach the desired results, two approaches have been considered. First of all, the robots design. Inspired by a biological spider, which has the ability to navigate through different terrains, a spider robot is ideal for the application intended. It was important to come up with a capable mechanical design for the robot so it can freely move around and do the movements needed. The design required a dependable leg design, which had the ability to move in all directions. In this design, safety is a very important thing to consider so most of the robot frame is smooth and have no sharp edges. Also, the electrical parts are reusable such as servos and the microcontroller to minimize the environmental effect this robot might cause. Second, the software part is offering the logic for the spider. Offering the user an instant move with no delay is great. The user is having a robot with a smooth movement and with a good speed. A simple control system is needed to make it easy for the user to steer and control the robot. An android app will be good enough to do that since everybody has a smartphone nowadays. Also, considering consuming energy and draining the battery; the robot should last a long time for it to be reliable. At the end, the robot should be straightforward and easy to assemble when manufacturing. It should be easy to build when entering the production line.

Client requirements:

- The robot should have the ability to walk and rotates smoothly
- An android app is needed to steer the robot via Wi-Fi communication
- The robot must pass the rugged road simulated test
- Sleep mode to save the battery which was replaced with charging pins built it the PCB

III. Design

In order to get the robot going, the robot is attached with 6 legs and flat central panel on its back (the spider body) which will contain the hardware. There are 18 servos to move the robot in all directions like moving forward and backward, left and right as smooth as possible. The robot is mostly RTOS, and Putty & Processing software is used to tune the PID of the servos in real time to get a good result. Also, the robot has several modes. For example, attack mode in which the robot will stand as high as possible, scared mode in which the robot looks scared and will sit as low as possible to the ground, and a mode that will make the thorax stable in a flat level when it is moving. Furthermore, a smartphone application was created to control the robot using Wi-Fi communication. Using RTOS helped to interact with the robot in real time and get a better PID tuning for the servos. That was to get rid of unwanted moves and the inertial that the legs and the body might cause when moving around.

A. Hardware:

The hardware of the spider robot is consisted of two major parts, the electrical parts and the Spider body panels. Below is the list of all electrical parts that been used.

The electronics part of the Spider robot consisted of these main parts:

- STM32F446RE
- Mini Maestro 18-Channel USB Servo Controller (Assembled)
- Customized PCB
- GY-85 Sensor Modules Accelerometer Gyroscope Module, 2.5mm Pin, ITG3205 + ADXL345 + HMC5883L Chip
- Logic Level Converter
- YEP 20A HV (2~12S) SBEC w/Selectable Voltage Output
- Towerpro MG996r Servos Motors
- Node MCU version 3(ESP 8266)
- 5000mAh, 3 Cell 11.1V Lipo Battery

The STM32F446 is the main brain of the robot. It uses the ARM Cortex-M4 core to achieve leading performance. The microprocessor has 180 MHz, 256-Kbyte to 512-Kbyte flash, and 128-Kbyte SRAM [1]. The microcontroller is charged with controlling all the movements and sending and receiving commands. Using nucleo board was great in such an application. Figure 1&2 below show the block diagram of all the connected parts:

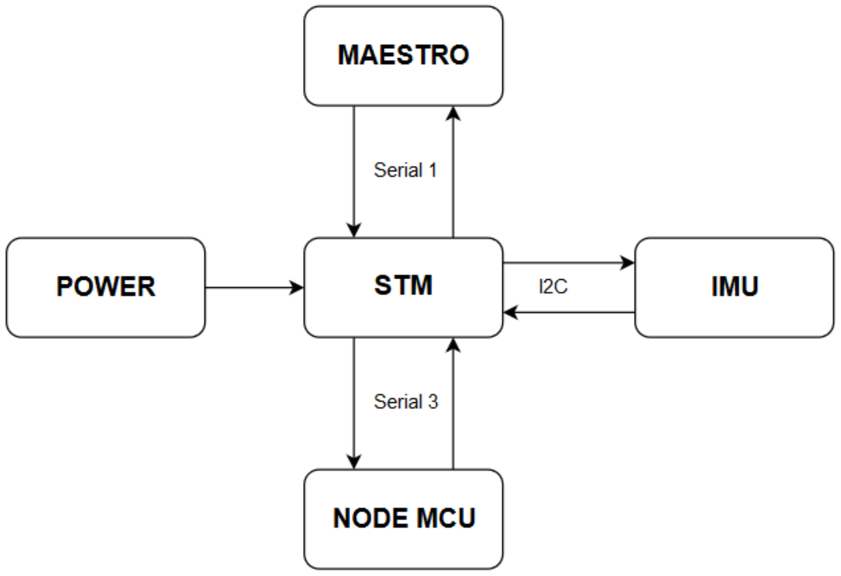


Figure 1: Connections to STM

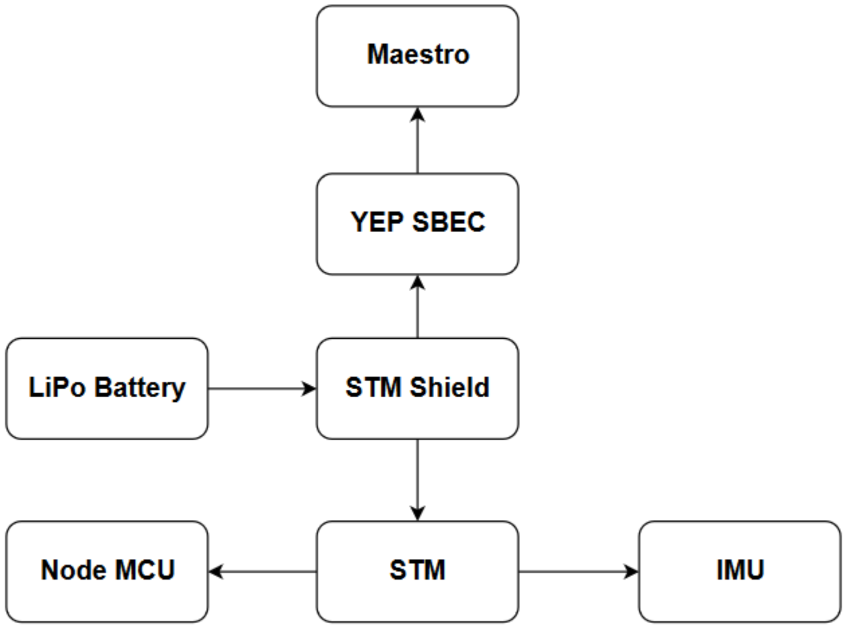


Figure 2: Power Connections

Along with that, a servo controller driver board been used which contains 18 channels to drive all the servos. That was important in helping out managing the servos separately. The Maestro Servo Controller board comes with its own processor to interact with the software that comes with it. The figure 3 below shows the Servo Controller board and figure 4 shows the servos connections to it.

Figure 3: Maestro Servo Controller board [2]

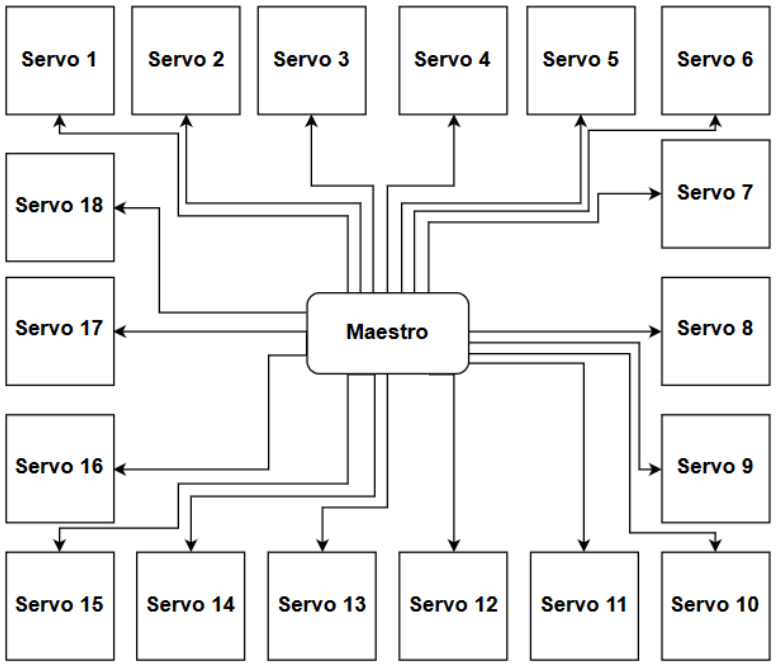


Figure 4: Maestro Connection to Servos

Towerpro MG996r servos been used which have good torque and speed with a dead band width of 1us and it is accurate. There are a lot of counterfeited servo in the internet and they are with a bad quality. The one was found were the originals but they are a little bit more expensive compared to the counterfeited one. Also, YEB(Selectable Voltage Output) was used to reduce the 11.1v from the battery to 6 v that goes to the servo driver board.

A PCB circuit was designed and ordered to eliminate the wiring and to avoid faulty connections. I used Eagle Software to design it. It was designed to fit over the nuclro board, and it is almost the same size as the nucleo board. That was done to reduce the size needed to place all those parts. The PCB contains all major parts: Logic Level Converter, GY-85 Sensor Modules Accelerometer Gyroscope Module, Wi-Fi ESP8266 Module. Also, it has couple of screw terminals pins to connect YEB and the Maestro board to the PCB. The figure 5 and 6 below show the circuit schematic of the PCB and the PCB itself before manufactured respectively.

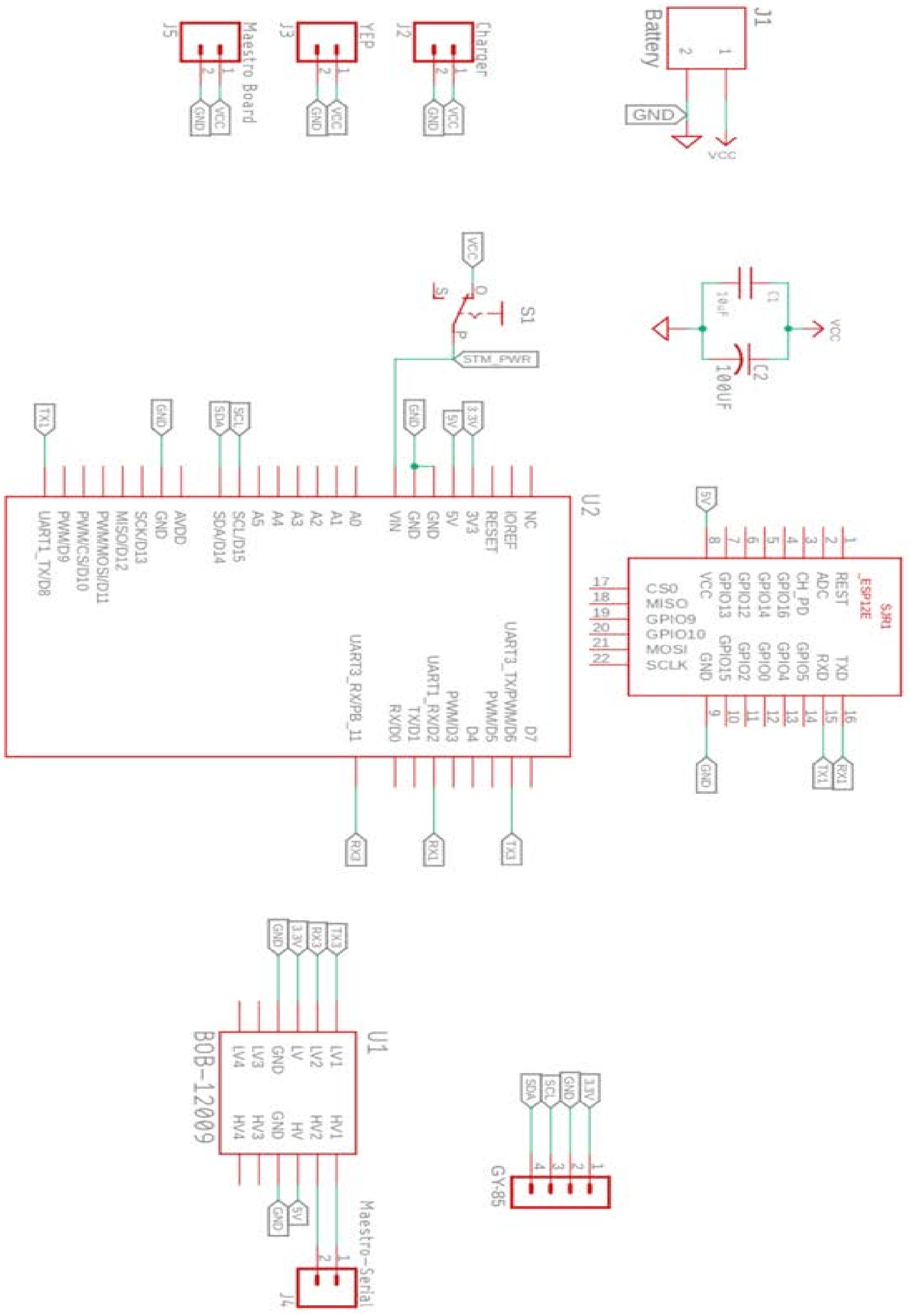


Figure 5: Circuit Schematic

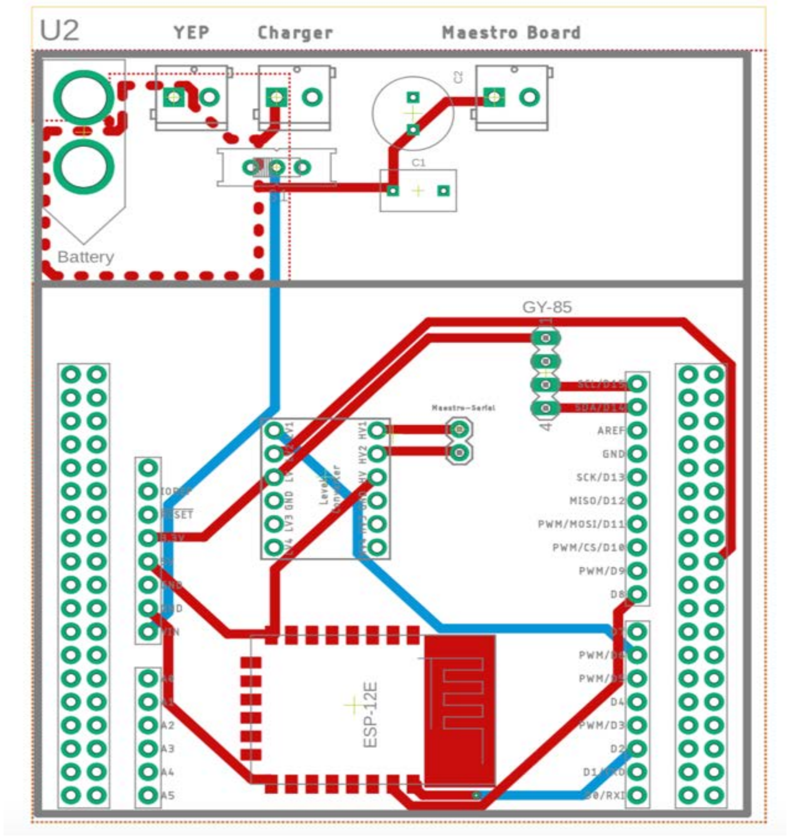


Figure 6: PCB layout

The spider body was designed in SolideWork Software. In this design, each leg has three degree of freedom (x, y, z) so it can move in all directions. The figure 4 below shows the parts needed for one leg. When the first leg came out the way intended for the purpose of this robot, that was after editing it several times, 6 legs were printed. Figure 5 shows the parts for the main body which carry all the electrical parts.

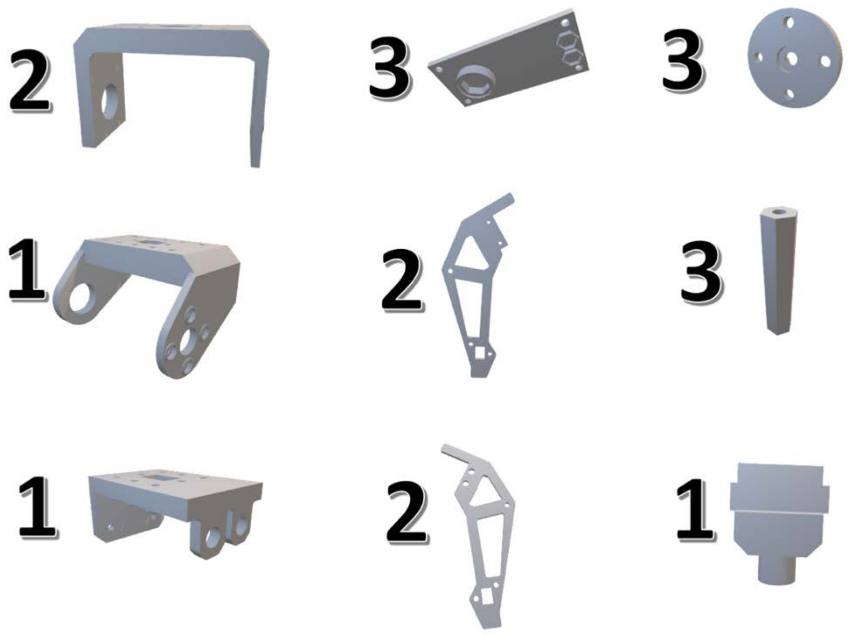


Figure 7: 3D one leg parts

The spider body contain the main body and the legs. In this design there are 6 legs, and each leg has 3 joint to allow the 3DOF. It has total of 18 servos. The robot used some bearing to allow smooth movements and punch of screws and nuts to tighten the parts together. Figure 6 shows the two plates that form the main body.

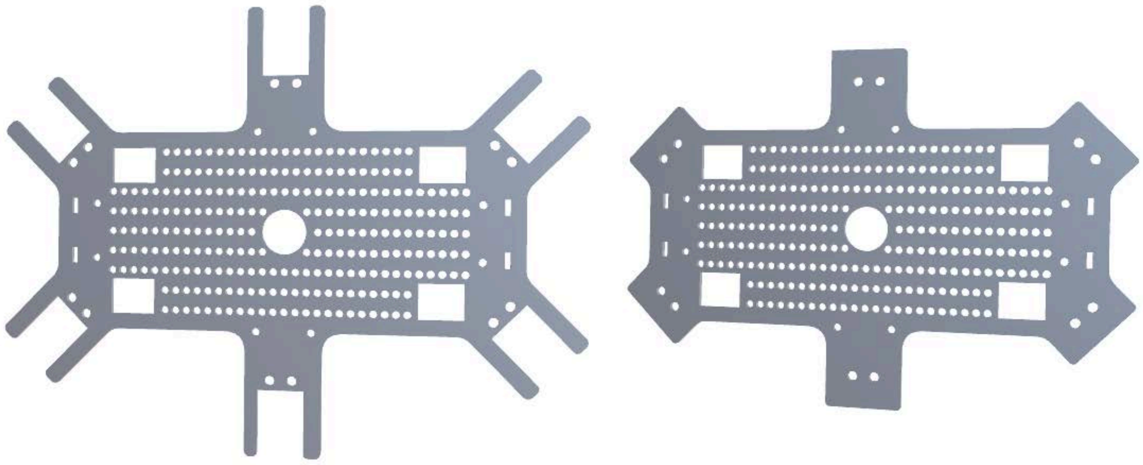


Figure 8: 3D Main body parts

After using couple of bearing, screws, and nuts to tighten everything g together, the final look is shown in figure 6.

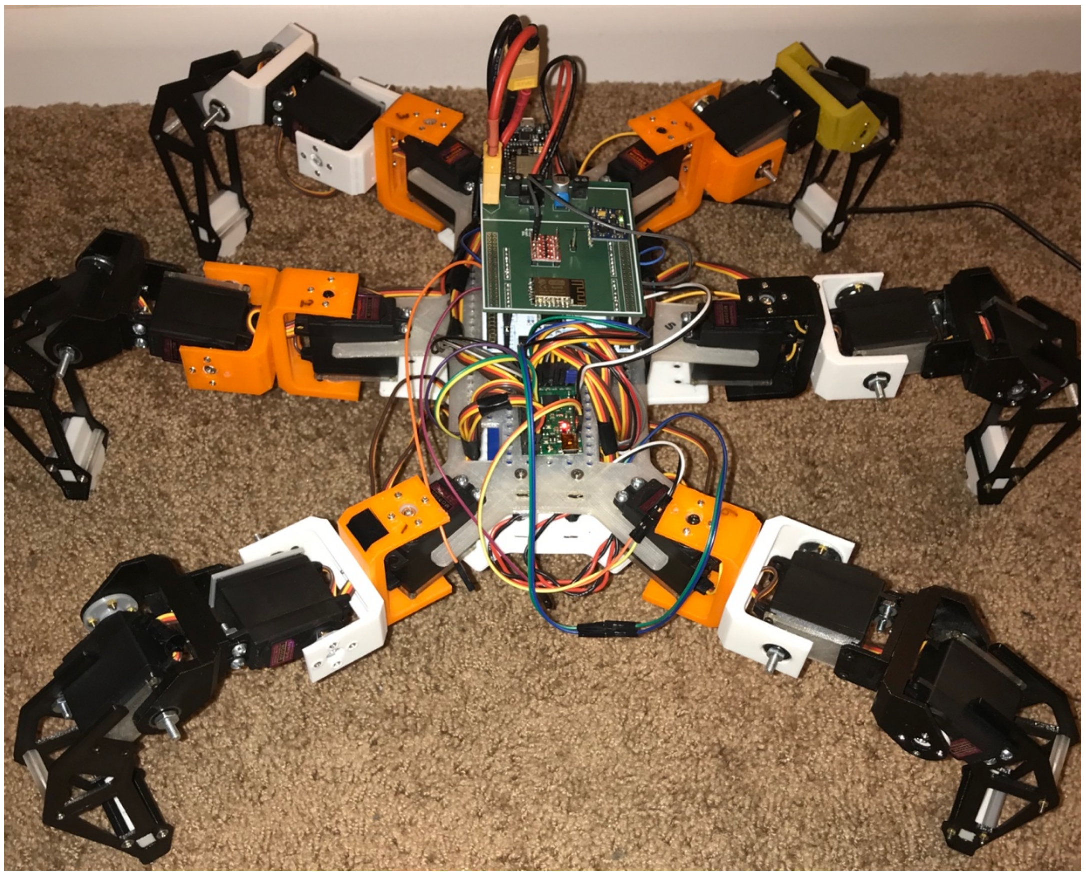


Figure 9: Spider robot final look

B. Software:

The software of this robot consists of two major parts: the microcontroller logic and the smartphone app. For the logic, the servo motors are controlled using Pulse width Modulation (PWM) received from the microcontroller. The direction and position of the servos can be controlled by varying the PWM signal. To do that, each servo is connected to the servo driver board. First of all, the software will identify the 0 position and that is when each leg starts to touch the ground.

That is to make the body level and have more stable movement. After that it will be ready for the commands from the user to be steered and controlled. In figure 3 is a simple schematic of the software process. The system will start with a switch, then it will wait for the user to send a command. If there is a command, then the Wi-Fi module will process that command to the microcontroller board. The board then will send a signal to the servo driver board. Then the servo driver board will send the signal to the involved servos to make that movement and it will run in loop.

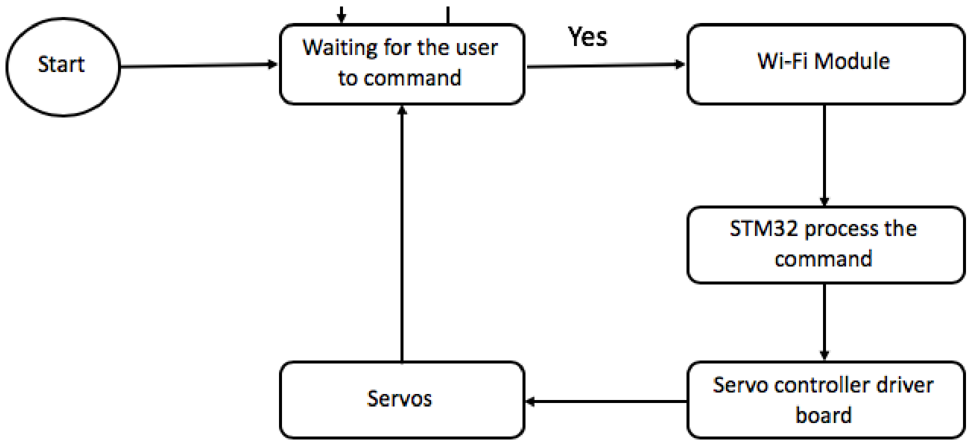

Start
Waiting for the user to command
Yes
Wi-Fi Module
STM32 process the command
Servo controller driver board
Servos

Figure 10: Simple schematic

The gait chosen for this robot was simple 3 legs motion. For Forward, rotation and backward motion the robot will move 3 legs up in the air and move rest in forward or reverse to complete half cycle then in next phase the legs which were in the air comes down while the one which were touching the ground will move up.

Forward motion Example:

Let the bot assembly be as follows:

Table 1: Legs numbers

1	2
3	4
5	6

For this let’s divide the gait on 7 cycles:

Cycle 1: leg 1 , 3 and 5 will move up.

Cycle 2: Leg 1, 3 and 5 will move forward.

Cycle 3: Leg 1, 3 and 5 will move down while leg 4, 6 and 2 will move up.

Cycle 4: Leg 1, 3 and 5 will move backward which will make bot move forward.

Cycle 5: Leg 4, 6 and 2 (which were in air) will move forward.

Cycle 6: Leg 4, 6 and 2 will move down while leg 1, 3 and 5 will move up.

Cycle 7: Leg 4, 6 and 2 will move backward which will make bot move forward.

Repetition of these 7 cycles will generate a whole walking pattern.

Servo motors are connected to Maestro servo board which communicated via serial communication to the STM32Nucleo board. Serial commands were sent in specified orders to control servo channels and target position of servos.

18 channel Polulu Maestro mini was used to control the servos. Serial protocol was used to establish communication between Maestro and STM32Nucleo board.

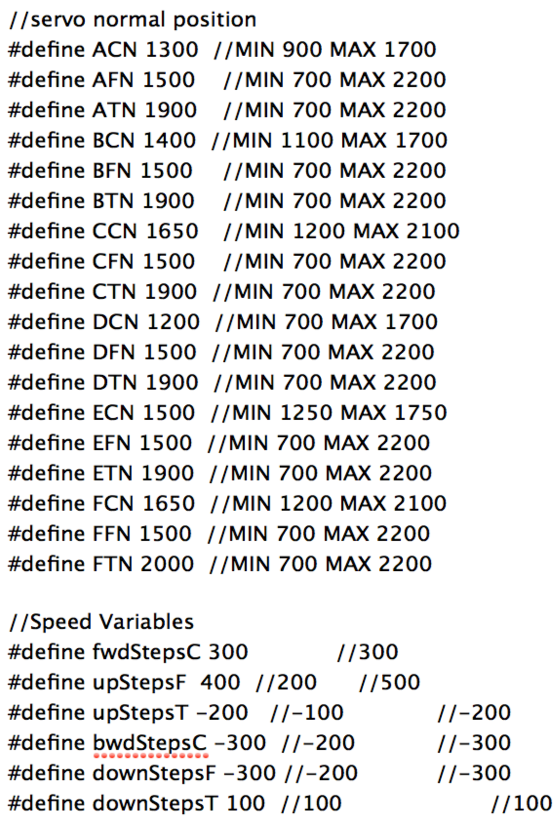
Here are the values of the factory min and max of pulse width that can be given to each servo which shown in table 2. And been used Excel to save and record those values. First was to find the factory min and max to make sure that not to be exceeded. Second was tried to find the range that each servo have and need for each leg individually for the available tolerance. Normal position indicates when the robot will stand up in normal position.

Table 2: Servos values

Leg	Servo	Min	Max	Normal position	Factory Min	Factory Max
1	1	900	1700	1300	600	2300
	2	700	2200	1500	600	2300
	3	700	2200	1900	600	2300
2	1	1100	1700	1400	600	2300
	2	700	2200	1500	600	2300
	3	700	2200	1900	600	2300
3	1	1200	2100	1650	600	2300
	2	700	2200	1500	600	2300
	3	700	2200	1900	600	2300
4	1	700	1700	1200	600	2300
	2	700	2200	1500	600	2300
	3	700	2200	1900	600	2300
5	1	1250	1750	1500	600	2300
	2	700	2200	1500	600	2300
	3	700	2200	1900	600	2300
6	1	1200	2100	1650	600	2300
	2	700	2200	1500	600	2300
	3	700	2200	1900	600	2300

Servo 1 = Attached to the plate
Servo 2 = middle joint
Servo 3 = last joint

After that, the values have been hard coded in the main function for further use. The declaration in shown in the figure 11 below.



```
//servo normal position
#define ACN 1300  //MIN 900 MAX 1700
#define AFN 1500    //MIN 700 MAX 2200
#define ATN 1900    //MIN 700 MAX 2200
#define BCN 1400  //MIN 1100 MAX 1700
#define BFN 1500    //MIN 700 MAX 2200
#define BTN 1900    //MIN 700 MAX 2200
#define CCN 1650    //MIN 1200 MAX 2100
#define CFN 1500    //MIN 700 MAX 2200
#define CTN 1900  //MIN 700 MAX 2200
#define DCN 1200  //MIN 700 MAX 1700
#define DFN 1500  //MIN 700 MAX 2200
#define DTN 1900  //MIN 700 MAX 2200
#define ECN 1500  //MIN 1250 MAX 1750
#define EFN 1500  //MIN 700 MAX 2200
#define ETN 1900  //MIN 700 MAX 2200
#define FCN 1650  //MIN 1200 MAX 2100
#define FFN 1500  //MIN 700 MAX 2200
#define FTN 2000  //MIN 700 MAX 2200

//Speed Variables
#define fwdStepsC 300           //300
#define upStepsF  400  //200      //500
#define upStepsT -200   //-100            //-200
#define bwdStepsC -300  //-200           //-300
#define downStepsF -300 //-200           //-300
#define downStepsT 100  //100                   //100
```

Figure 11: Servos positions

Following are some of the serial commands Maestro used for controlling attached servos:

SetTarget: function void setPos(uint8_t channel, uint16_t target) is the one used to send a servo to specific target. Following are step and set commands:

1. 0x84 send via serial to specify that next serial values will determine channel number and target position.
2. Next command will be to send channel number to Maestro e.g. 0x01 for channel number 1.

3. Next two commands will determine the target position. It needs to send 14 bit data to the Maestro. For example, if a servo needed to go to position 2000, then that position will be multiplied by 4 and then split it into two 7 bit numbers. E.g. target = 2000 * 4

   serialBytes[2] = target & 0x7F;   //lsb

   serialBytes[3] = (target >> 7) & 0x7F; //msb

SetMultiTarget: function void multiTarget() was used to give target position to all servos at one.

1. 0x9F send via serial to specify that next serial values will determine multi target positions.
2. Next command will send 7 bit number to specify total targets.
3. Next command will send 7 bit number to specify first target.
4. Next send consecutive target positions. For example, selecting 4 as total channels and 3 as first target then next 4 14 bit commands will specify the position of channel 3, 4, 5 and 6.

For the balance feature Gyroscope & Accelerometer Module been used which referred as GY-85. GY-85 is an I2C module containing accelerometer, gyro and compass. Only accelerometer been used to get reading of tilt in X and Y axis to stabilize/ balance the robot periodically or before each control command.

As said above GY-85 has gyro, accelerometer and compass all in one module but only accelerometer was used to determine the tilt of the robot and then balancing of the robot using those values. The communication protocol used was I2C. This protocol requires only 2 lines namely SCL and SDA to communicate between two devices. Each device has a unique address which is used to alert the device for incoming and outgoing data. There is one master device and there can be one or more than one slave devices. In this case, STM32Nucleo was master while GY-85 was a slave device.

In the code there are two functions for the Accelerometer communication.

1. Void setupAcc() this function setup the accelerometer for reading value. It sends the device address of 0xA6 and other initial values.
2. Void readAcc() after sending 0x32 to the accelerometer the master can read next 6 bytes of data to determine tilt in x, y and z.

Also, for the balancing feature, it required using the values from accelerometer. The function used was void stabilize() in this function it first get the values of x and y from accelerometer and then if the values deviate from 0 a lot then depending on the values, it will move the femur of the robot to counter the tilt. For Example: If the Y values are less than -20 then that means that front of our bot is higher than the back so we either move femur of back legs down or move femur of front legs up to counter the tilt.

Smartphone application was created to give the ability to remote control the robot and to steer it. For making the smartphone app, MIT AppInventor was used to design the app. MIT AppInventor is an online tool that is simple, but it is capable of making professional app in the same time. In figure 12 below is the app main page of the app where the user has two options.

Either hit start to start playing with the robot or help menu which contain information of how to connect to the robot and what the features the robot has.

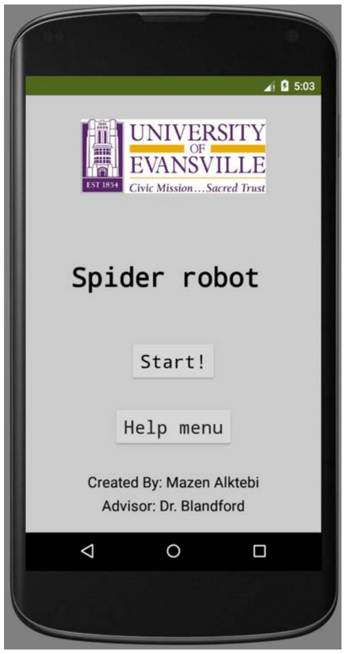


Figure 12: Smartphone App

When pressing the start button, the app will go to the function page where are the features and the functions available for the user. There are three functions as shown in figure 13

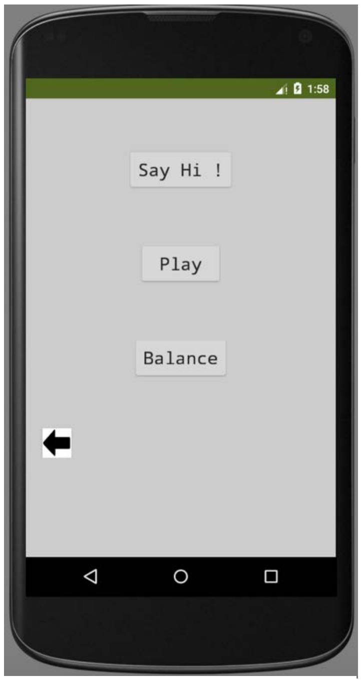


Figure 13: Function menu

Functions are:

1. Say Hi: the leg chosen by the user will go up for 5 seconds and then it will go down.
2. Play option: The user can choose to play sit/stand where the robot will sit to the ground and stand up. Or move option to steer the robot to move in all direction (Forward, Backward, Left, Right)
3. Balance option: There is an Activate button that will balance the Spider main body to the ground flat level (0 degree) when pressed. Whenever the user tries to play a move, the balance option will be deactivated.

In each screen of the movements function there are arrows that determine the directions of where the robot going to move to. The patter here is when a button is pressed and left the button, the move will take place for that complete step. There was a possibility to design in a way that it keeps moving along while the user finger is pressing that button, but it kept storing the incoming message in the stream and it sometime move even when the finger is lifted

For communication between the app and the robot, Nodemcu (ESP8266 12E) was used to get values from android app and send it to STM32Nucleo board via serial communication. It is a low-cost chip that enables the user integrate Wi-Fi in the system and it uses SPI/UART communication. It is shown in figure 14 below.

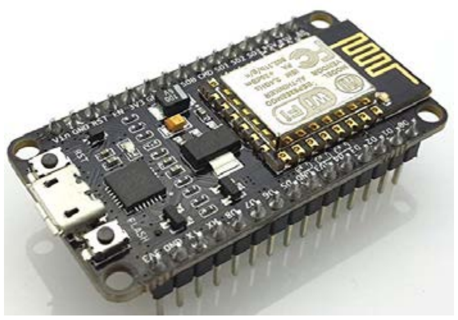
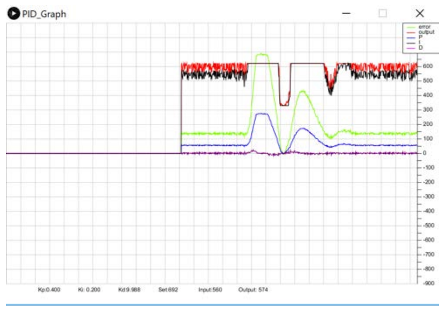


Figure 14: NodeMCU (ESP8266) Wi-Fi module [3]

NodeMCU code had following setup and function:

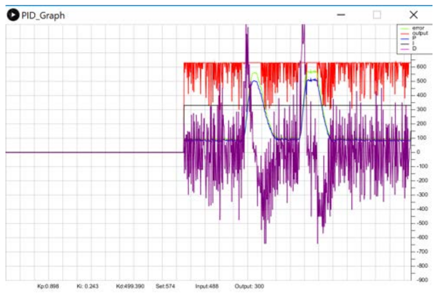
1. Setup NodeMCU in access point mode so that Mobile phone can connect the Wi-Fi of NodeMCU.
2. Make a TCP server on NodeMCU.
3. Android app will act as a client and will send different commands like Move forward, rotate left etc. to the Nodemcu.
4. NodeMCU will then transmit those commands to STM32Nucleo board over serial communication.

Along with that, Processing Software were used to tune the PID of the servos to get rid of any unwanted movements (inertia, frictions …). Figure 4 is an example of the PID tuning. The Processing software uses UART commands via virtual COM port to the Nucleo board. The figures show the time-domain graphs of input and output values, input error, and separately promotional, integral and differential components of output value. In figures 4 & 5 are examples of the PID tuning of a dc motor. Notice the gap in figure 4 where the inertia effecting the motor, and the final results in figure 5 where the motor is more accurate. There were not too much tuning because luckily the servos are accurate.



Kp:0.400 Ki: 0.200 Kd:9.988 Set:692 Input:560 Output: 574

Figure 15: Initial values



Kp:0.898 Ki: 0.243 Kd:499.390 Set:574 Input:488 Output: 300

Figure 16: Final values

C. Standards:

While building this project, several considerations were taken onto account. In this robot, safety is number one priority. The robot will have safety shut off in case of overloading which will prevent any electric parts from burning. The body of the robot will not have any sharp edges that might cause injury. All small parts will be tightened and hidden away, that is to prevent children under 3 years old form swallow them. Moving to manufacturability and sustainability, rebuilding the robot now is easier than before since all weak points was known, and been left with best choice of any option that can be used. The ability to manufacture is very high. The robot would require several specific parts which can be put together to form the final robot. Also, the robot should last long due the material and parts that will be built of. For sustainability, all servos and other components is with good quality. The robot is last long before it stops and that is because of the way the robot is designed. For health impact, since the robot is remote controlled, there is no need for human to make direct contact with the robot which will prevent any injury the servos or legs movements might cause.

IV. Cost

The project cost will be affordable. The only one thing that will have the major cost are the servos, where each servo costs $6.95. There are much cheaper servos in the internet but those won’t be efficient for this application of this robot because a servo with high torque and speed were required. In the robot, 18 servos are required which makes it the costliest thing in this project. Other items are fairly cheap and won’t cost too much.

Table 3: Cost

Description	Quantity	Unit price	Price
Sparkfun Logic Level Converter - Bi Directional	1	$2.95	$2.95
ESP8266 SMT Module	1	$6.95	$6.95
Screw Terminal 3.5 mm Pitch	3	$1.9	$5.70
XT-60 Male Connector	1	$1.5	$1.5
100uF Electrolytic Capacitor	1	$1.49	$1.49
10uF Ceramic Capacitor	1	$2.31	$2.31
Mini Maestro 18-Channel USB Servo Controller	1	$39.95	$39.95
Customized PCB	1	$50	$50
GY-85 Sensor Modules Accelerometer Gyroscope Module, 2.5mm Pin, ITG3205 + ADXL345 + HMC5883L Chip	1	$18.66	$18.66
Turnigy 5000mAh 3S 11.1V 20C 30C Lipo Battery XT60 RC Traxxas Car Truck Plane US	1	$37.99	$37.99
YEP 20A HV (2~12S) SBEC w/Selectable Voltage Output	1	$15.65	$15.65
Original TpwerPro MG996r Servos	18	$6.95	$124.2
10 pieces of 62R-2RS bearing	2	$9.99	$19.98
Bunch of M2x7mm , M2x10mm, M2x32mm, M2x35mm, M2x40mm, M3x16mm,, M2-0.4 Hex, M2.5x6mm flat washer ≅ $35		Total = $279.5 + $35 = $314.5	

V. Results

This project is successfully satisfied the minimum client requirements. The android app is steering the robot via Wi-Fi communication. The PID tuning of servos is well working, and the robot is smoothly walking and rotating, and it is doing the required movements. The robot passed the test of walking through rugged road. Also, the saving battery feature was replaced with charging pins that is built in the PCB which is more efficient for having the voltage needed for the robot when the battery near to drain out. It is ready for manufacturing with height end materials.

References:

[1] “STM32F446”. STM32. Web. 19 Nov. 2018.

https://www.st.com/en/microcontrollers/stm32f446.html?querycriteria=productId=LN1875

[2] “Servo controller driver board.” adafruit. Web. 19 Nov. 2018.

https://www.adafruit.com/product/1429?gclid=Cj0KCQiA28nfBRCDARIsANc5BFA9MIQk1lR5b67V2hq0F3uAppjV-sZuoiG1pOXaDOElZV7JIlcy67AaAqwQEALw_wcB

[3] “Wifi module ESP-WROOM-32.” Mouser Electronics. Web. 19 Nov. 2018.

https://www.mouser.com/ProductDetail/Espressif-Systems/ESP32-WROOM-32?qs=chTDxNqvsyltcwz%2FUUJDtQ%3D%3D&gclid=Cj0KCQiA28nfBRCDARIsANc5BFDXvwVnRgCjtgVGw03gnIr0m2R4UiTIxjaKtrcMqSv2DUP456dkM7caAqlGEALw_wcB

Appendix:

```
#include "main.h"
#include "i2c.h"
#include "usart.h"
#include "gpio.h"


// servo channels
#define AC 0
#define AF 1
#define AT 2
#define BC 3
#define BF 4
#define BT 5
#define CC 6
#define CF 7
#define CT 8
#define DC 9
#define DF 10
#define DT 11
#define EC 12
#define EF 13
#define ET 14
#define FC 15
#define FF 16
#define FT 17

//servo normal positions
#define ACN 1300  //MIN 900 MAX 1700
#define AFN 1500    //MIN 700 MAX 2200
#define ATN 1900    //MIN 700 MAX 2200
#define BCN 1400  //MIN 1100 MAX 1700
#define BFN 1500    //MIN 700 MAX 2200
#define BTN 1900    //MIN 700 MAX 2200
#define CCN 1650    //MIN 1200 MAX 2100
#define CFN 1500    //MIN 700 MAX 2200
#define CTN 1900  //MIN 700 MAX 2200
#define DCN 1200  //MIN 700 MAX 1700
#define DFN 1500  //MIN 700 MAX 2200
#define DTN 1900  //MIN 700 MAX 2200
#define ECN 1500  //MIN 1250 MAX 1750
#define EFN 1500  //MIN 700 MAX 2200
#define ETN 1900  //MIN 700 MAX 2200
#define FCN 1650  //MIN 1200 MAX 2100
#define FFN 1500  //MIN 700 MAX 2200
#define FTN 2000  //MIN 700 MAX 2200
```

```
//Speed Variables
#define fwdStepsC 300             //300
#define upStepsF  400  //200      //500
#define upStepsT -200   //-100          //-200
#define bwdStepsC -300  //-200          //-300
#define downStepsF -300 //-200          //-300
#define downStepsT 100  //100           //100


void SystemClock_Config(void);

void stablize();
void setPos(uint8_t, uint16_t);  // This Functions send 14 bit
data to maestro arguments are ( channel, target Pos)
void walkFwd(); //Bot Will complete one gait cycle of forward
motion
void d2Down();  //Femer and tibia of LEG F, d and B will move to
normal position
void d2Up();    //Femer of LEG F, D and B will move to upward
upStepsF and tibia to upStepsT from normal position
void d1Down();  //Femer and tibia of LEG A, C and E will move to
normal position
void d1Up();    //Femer of LEG A, C and E will move to upward
upStepsF and tibia to upStepsT from normal position
void d1Fwd();       //Coxa of LEG A, C and E will move
fwdStepsC forward from normal postion
void d2Fwd();       //Coxa of LEG B, F and D will move
fwdStepsC forward from normal postion
void d1NC();    //Cox of LEG A, C and E will move to normal
position
void d2NC();     //Cox of LEG F, D and B will move to normal
position


void d1Normal();  //Coxa, Femer and Tibia of Legs A, C and E
will move to normal positions
void d2Normal();  //Coxa, Femer and Tibia of Legs B, D and F
will move to normal positions
void testCode();  // NONE
void rotateRight();  //Bot Will complete one gait cycle of right
rotation
void rotateLeft();   //Bot Will complete one gait cycle of left
rotation
```

```
void d1RR();    //Coxa of Legs A and C will move backwards by
"fwdStepsC" and E will move forwards by "fwdStepsC" from normal
postion
void d2RR();    //Coxa of Legs F and D will move forward by
"fwdStepsC" and B will move backwards by "fwdStepsC"  from
normal postion
void d1RL();        //Coxa of Legs A and D will move forward by
"fwdStepsC" and E will move backwards by "fwdStepsC"  from
normal postion
void d2RL();    //Coxa of Legs F and D will move backward by
"fwdStepsC" and B will move forward by "fwdStepsC"   from normal
postion


void walkRev(); //Caution Not working properly //Bot Will
complete one gait cycle of right rotation
void d1Back();  //Coxa of LEG A, C and E will move fwdStepsC
backward from normal postion
void d2Back();  //Coxa of LEG B, F and D will move fwdStepsC
backward from normal postion
void walkRevG2(); //Caution: Not working properly //Bot Will
complete one gait cycle of right rotation
void walkRevG3(); //Caution: Not working properly //Bot Will
complete one gait cycle of right rotation
void sitDown();   //Caution: Not working properly //Bot will
sitdown from normal postion
void standUp();   //Caution: Not working properly //Bot will
standUP from sitDown Position
void standUpV2(); //Caution: Not working properly //Bot will
standUP from sitDown Position
void standUpPrep(); //Caution: Not working properly //Bot will
take standUP stance from sitDown Position
void standUpV3();   //Caution: Not working properly //Bot will
standUP from sitDown Position


void leg1Play();    //Femer and Tibia of described leg will move
up by upStepsF and upStepsT from normal position respectively
void leg2Play();    //Femer and Tibia of described leg will move
up by upStepsF and upStepsT from normal position respectively
void leg3Play();    //Femer and Tibia of described leg will move
up by upStepsF and upStepsT from normal position respectively
void leg4Play();    //Femer and Tibia of described leg will move
up by upStepsF and upStepsT from normal position respectively
void leg5Play();    //Femer and Tibia of described leg will move
up by upStepsF and upStepsT from normal position respectively
```

```
void leg6Play();    //Femer and Tibia of described leg will move
up by upStepsF and upStepsT from normal position respectively
void write14Bit(uint16_t);  //Send 14Bit Value over I2C to
Maestro // used it for MultiTargetfunction
void multiTarget(); //Send Target position to multiple servos at
once// tried to use it for standUp function
uint8_t getMovState();  //Caution: Not Working //tried to get
servo position from maestro.
void fullNormal();
void readAcc();               //Read values of Accelerometer
void setAcc();           //Ready the accelerometer
static float acc_x=0; //x Axis value of accelerometer
static float acc_y=0; //y Axis value of accelerometer
static float acc_z=0; //z Axis value of accelerometer
uint8_t i2cData[2]={0,0}; //used for I2C data transfer


int main(void)
{

  HAL_Init();

  SystemClock_Config();

  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_USART1_UART_Init();
  MX_USART3_UART_Init();
  MX_I2C1_Init();


     uint8_t serialBytes[4];
     d1Normal();
     HAL_Delay(2000);
     d2Normal();
     HAL_Delay(2000);
     uint8_t serialCont [1];
  while (1)
  {

       HAL_UART_Receive(&huart3, &serialCont[0], 1, 100);
          HAL_Delay(200);

          if (serialCont[0] == 48){
               fullNormal();
               HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
            walkFwd();
               serialCont[0]=0;
          }
          else if (serialCont[0] == 49){
               fullNormal();
                    walkRev();
               HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
               serialCont[0]=0;
          }
          else if (serialCont[0] == 50){
               fullNormal();
                    rotateLeft();
               HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
               serialCont[0]=0;
          }
          else if (serialCont[0] == 51){
               fullNormal();
                    rotateRight();
               HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
               serialCont[0]=0;
          }
          else if (serialCont[0] == 52){
               fullNormal();
                    leg1Play();
               HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
               serialCont[0]=0;
          }
          else if (serialCont[0] == 53){
               fullNormal();
                    leg2Play();
               HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
               serialCont[0]=0;
          }
          else if (serialCont[0] == 54){
               fullNormal();
                    leg3Play();
               HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
               serialCont[0]=0;
          }
          else if (serialCont[0] == 55){
               fullNormal();
                    leg4Play();
               HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
               serialCont[0]=0;
          }
          else if (serialCont[0] == 56){
               fullNormal();
                    leg5Play();
```

```
               HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
               serialCont[0]=0;
          }
          else if (serialCont[0] == 57){
               fullNormal();
                    leg6Play();
               HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
               serialCont[0]=0;
          }
          else if (serialCont[0] == 58){
               fullNormal();
               stablize();
               HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
               serialCont[0]=0;

          }
          else if (serialCont[0] == 59){
               fullNormal();
               sitDown();
               HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
               serialCont[0]=0;
          }
          else if (serialCont[0] == 60){
               fullNormal();
               standUp();
               HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);

               serialCont[0]=0;
          }
  }

}


void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};


  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE3);
  /**Initializes the CPU, AHB and APB busses clocks
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue =
RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 16;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 2;
  RCC_OscInitStruct.PLL.PLLR = 2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /**Initializes the CPU, AHB and APB busses clocks
  */
  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK

|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2)
!= HAL_OK)
  {
    Error_Handler();
  }
}

/* USER CODE BEGIN 4 */


void fullNormal(){
     d1Normal();
     HAL_Delay(200);
     d2Normal();
     HAL_Delay(200);
}
int getPos(uint8_t channel){

     uint8_t serialBytes [4];
     serialBytes[0] = 0x90;
     serialBytes[1] = channel;
     HAL_UART_Transmit(&huart1, &serialBytes[0], 1, 1000);
     //HAL_Delay(20);
     HAL_UART_Transmit(&huart1, &serialBytes[1], 1, 1000);
     HAL_UART_Receive(&huart1, &serialBytes[0], 1, 1000);
     HAL_UART_Receive(&huart1, &serialBytes[1], 1, 1000);
     return ((serialBytes[0] << 8) | (serialBytes[1] & 0xFF));
     //HAL_Delay(20);
}

uint8_t getMovState(){

     uint8_t serialBytes [4];
     serialBytes[0] = 0x93;
     HAL_UART_Transmit(&huart1, &serialBytes[0], 1, 1000);

     HAL_UART_Receive(&huart1, &serialBytes[0], 1, 1000);

     return serialBytes[0];
     //HAL_Delay(20);
}
void write14Bit( uint16_t target){
     target=target *4;
     uint8_t serialBytes [4];
  serialBytes[2] = target & 0x7F; //lsb
     serialBytes[3] = (target >> 7) & 0x7F;//msb
     HAL_UART_Transmit(&huart1, &serialBytes[2], 1, 1000);
     //HAL_Delay(20);
     HAL_UART_Transmit(&huart1, &serialBytes[3], 1, 1000);
     //HAL_Delay(20);
}
void setPos(uint8_t channel, uint16_t target){
     target=target *4;
     uint8_t serialBytes [4];
     serialBytes[0] = 0x84;
     serialBytes[1] = channel;
  serialBytes[2] = target & 0x7F; //lsb
     serialBytes[3] = (target >> 7) & 0x7F;//msb
     HAL_UART_Transmit(&huart1, &serialBytes[0], 1, 1000);
     //HAL_Delay(20);
     HAL_UART_Transmit(&huart1, &serialBytes[1], 1, 1000);
     //HAL_Delay(20);
     HAL_UART_Transmit(&huart1, &serialBytes[2], 1, 1000);
     //HAL_Delay(20);
     HAL_UART_Transmit(&huart1, &serialBytes[3], 1, 1000);
     //HAL_Delay(20);
}
```

```
     /*   A  F
          B  E
          C  D */

uint16_t smallDelay =50; //100
uint16_t bigDelay = 100;  //300
void stablize(){
     int CFS= CFN; int DFS = DFN; int AFS =AFN; int FFS=FFN;
     setAcc();
     readAcc();
     while (acc_x >5 || acc_x<=-5 || acc_y >5 || acc_y <= -5){
          readAcc();
          if (acc_x <-5 && acc_y <-5){
               if (DFS -10 > 700)
                    DFS=DFS-10;
               if (AFS +10 < 2200)
                 AFS=AFS+10;
               if (DFS -10 < 700 && AFS +10 > 2200)
                    break;
               setPos(DF, DFS);
               HAL_Delay(smallDelay);
               setPos(AF, AFS);
               HAL_Delay(smallDelay);
          }
          else if (acc_x >5 && acc_y<-5){
          if (CFS -10 > 700)
                    CFS=CFS-10;
               if (FFS +10 < 2200)
                 FFS=FFS+10;
               if (CFS -10 < 700 && FFS +10 > 2200)
                    break;
               setPos(CF, CFS);
               HAL_Delay(smallDelay);
               setPos(FF, FFS);
               HAL_Delay(smallDelay);
          }
          else
               break;
     }
}
void walkFwd(){
     d1Up();
     d1Fwd();

     d1Down();
     d2Up();
     d1NC();
     d2Fwd();

     d2Down();
     d1Up();

     d2NC();
     d1Down(); //
     //d1Fwd();
}

void walkRev(){
     d1Up();
     d1Back();

     d1Down();
     d2Up();

     d1NC();
     d2Back();

     d2Down();
     d1Up();

     d2NC();
     d1Down();//
     //d1Back();
}

void rotateLeft (){
  d1Up();
     d1RL();

     d1Down();
     d2Up();

     d1NC();
     d2RL();

     d2Down();
     d1Up();

     d2NC();
     d1Down();//
     //d1RL();
}
```

```
void rotateRight (){
  d1Up();
     d1RR();

     d1Down();
     d2Up();

     d1NC();
     d2RR();

     d2Down();
     d1Up();

     d2NC();
     d1Down();//
     //d1RR();
}

void d2Down(){
     setPos(BF, BFN);    //LEG B DOWN
     HAL_Delay(smallDelay);
     setPos(BT, BTN);
     HAL_Delay(smallDelay);
     setPos(DF, DFN); //LEG D DOWN
     HAL_Delay(smallDelay);
     setPos(DT, DTN);
     HAL_Delay(smallDelay);
     setPos(FF, FFN);    //LEG F DOWN
     HAL_Delay(smallDelay);
     setPos(FT, FTN);
     HAL_Delay(smallDelay);
}

void d2Up(){
     setPos(FF, FFN+upStepsF);     //LEG F UP
     HAL_Delay(smallDelay);
     setPos(FT, FTN+upStepsT);
     HAL_Delay(smallDelay);
     setPos(DF, DFN+upStepsF);     //LEG D UP
     HAL_Delay(smallDelay);
     setPos(DT, DTN+upStepsT);
     HAL_Delay(smallDelay);
     setPos(BF, BFN+upStepsF);     //LEG B UP
     HAL_Delay(smallDelay);
     setPos(BT, BTN+upStepsT);
     HAL_Delay(smallDelay);
}

void d1Up(){
     //LEG A UP
     setPos(AF, AFN + upStepsF);
     HAL_Delay(smallDelay);
     setPos(AT, ATN + upStepsT);
     HAL_Delay(smallDelay);
     //LEG C UP
  setPos(CF, CFN + upStepsF);
     HAL_Delay(smallDelay);
     setPos(CT, CTN + upStepsT);
  HAL_Delay(smallDelay);
     //LEG E UP
     setPos(EF, EFN + upStepsF);
     HAL_Delay(smallDelay);
     setPos(ET, ETN + upStepsT);
     HAL_Delay(smallDelay);
}



void d1Down(){
     setPos(AF, AFN);    //LEG A DOWN
     HAL_Delay (smallDelay);
     setPos(AT, ATN);
     HAL_Delay(smallDelay);
     setPos(CF, CFN);    //LEG C DOWN
     HAL_Delay(smallDelay);
     setPos(CT, CTN);
     HAL_Delay(smallDelay);
     setPos(EF, EFN);    //LEG E DOWN
     HAL_Delay(smallDelay);
     setPos(ET, ETN);
     HAL_Delay(smallDelay);


}

void d1Fwd(){
     setPos(AC, ACN+fwdStepsC);    //LEG A FWD
     HAL_Delay(bigDelay);
     setPos(CC, CCN+fwdStepsC);    //LEG C FWD
     HAL_Delay(bigDelay);
     setPos(EC, ECN-fwdStepsC);    //LEG E FWD
     HAL_Delay(bigDelay);
}
```

```
void d1Back(){
     setPos(AC, ACN-fwdStepsC);    //LEG A FWD
     HAL_Delay(bigDelay);
     setPos(CC, CCN-fwdStepsC);    //LEG C FWD
     HAL_Delay(bigDelay);
     setPos(EC, ECN+fwdStepsC);    //LEG E FWD
     HAL_Delay(bigDelay);
}

void d1RL(){
     setPos(AC, ACN+fwdStepsC);    //LEG A FWD
     HAL_Delay(bigDelay);
     setPos(CC, CCN+fwdStepsC);    //LEG C FWD
     HAL_Delay(bigDelay);
     setPos(EC, ECN+fwdStepsC);    //LEG E FWD
     HAL_Delay(bigDelay);
}

void d1RR(){
     setPos(AC, ACN-fwdStepsC);    //LEG A FWD
     HAL_Delay(bigDelay);
     setPos(CC, CCN-fwdStepsC);    //LEG C FWD
     HAL_Delay(bigDelay);
     setPos(EC, ECN-fwdStepsC);    //LEG E FWD
     HAL_Delay(bigDelay);
}

void d2Fwd(){
     setPos(DC, DCN-fwdStepsC);    //LEG D FWD
     HAL_Delay(bigDelay);
     setPos(FC, FCN-fwdStepsC);    //LEG F FWD
     HAL_Delay(bigDelay);
     setPos(BC, BCN+fwdStepsC);    //LEG B FWD
     HAL_Delay(bigDelay);
}

void d2Back(){
     setPos(DC, DCN+fwdStepsC);    //LEG D FWD
     HAL_Delay(bigDelay);
     setPos(FC, FCN+fwdStepsC);    //LEG F FWD
     HAL_Delay(bigDelay);
     setPos(BC, BCN-fwdStepsC);    //LEG B FWD
     HAL_Delay(bigDelay);
}
```

```
void d2RL(){
     setPos(DC, DCN+fwdStepsC);    //LEG D FWD
     HAL_Delay(bigDelay);
     setPos(FC, FCN+fwdStepsC);    //LEG F FWD
     HAL_Delay(bigDelay);
     setPos(BC, BCN+fwdStepsC);    //LEG B FWD
     HAL_Delay(bigDelay);
}

void d2RR(){
     setPos(DC, DCN-fwdStepsC);    //LEG D FWD
     HAL_Delay(bigDelay);
     setPos(FC, FCN-fwdStepsC);    //LEG F FWD
     HAL_Delay(bigDelay);
     setPos(BC, BCN-fwdStepsC);    //LEG B FWD
     HAL_Delay(bigDelay);
}
void d1NC(){
     setPos(AC, ACN);    //LEG A NEU
     HAL_Delay(bigDelay);
     setPos(CC, CCN);    //LEG C NEU
     HAL_Delay(bigDelay);
     setPos(EC, ECN);    //LEG E NEU
     HAL_Delay(bigDelay);
}

void d2NC(){
     setPos(DC, DCN);  //LEG D NEU
     HAL_Delay(bigDelay);
     setPos(FC, FCN);  //LEG F NEU
     HAL_Delay(bigDelay);
     setPos(BC, BCN);  //LEG B NEU
     HAL_Delay(bigDelay);
}

void d1Normal(){
     setPos(AC, ACN);    //LEG A NEU
     HAL_Delay(bigDelay);
     setPos(CC, CCN);    //LEG C NEU
     HAL_Delay(bigDelay);
     setPos(EC, ECN);    //LEG E NEU
     HAL_Delay(bigDelay);
     setPos(AF, AFN);
     HAL_Delay(bigDelay);
     setPos(CF, CFN);
     HAL_Delay(bigDelay);
     setPos(EF, EFN);
     HAL_Delay(bigDelay);
     setPos(AT, ATN);
     HAL_Delay(bigDelay);
     setPos(CT, CTN);
     HAL_Delay(bigDelay);
     setPos(ET, ETN);
     HAL_Delay(bigDelay);
}

void d2Normal(){
     setPos(FC, FCN);
     HAL_Delay(bigDelay);
     setPos(DC, DCN);
     HAL_Delay(bigDelay);
     setPos(BC, BCN);
     HAL_Delay(bigDelay);
     setPos(FF, FFN);
     HAL_Delay(bigDelay);
     setPos(DF, DFN);
     HAL_Delay(bigDelay);
     setPos(BF, BFN);
     HAL_Delay(bigDelay);
     setPos(FT, FTN);
     HAL_Delay(bigDelay);
     setPos(DT, DTN);
     HAL_Delay(bigDelay);
     setPos(BT, BTN);
     HAL_Delay(bigDelay);
}


void walkRevG2(){

     //LEG1
     setPos(AF, AFN + upStepsF);
     HAL_Delay(smallDelay);
     setPos(AT, ATN + upStepsT);
     HAL_Delay(smallDelay);
     setPos(AC, ACN-fwdStepsC);
     HAL_Delay(bigDelay);
     setPos(AF, AFN);    //LEG A DOWN
     HAL_Delay (smallDelay);
     setPos(AT, ATN);
     HAL_Delay(smallDelay);

          //LEG6
     setPos(FF, FFN+upStepsF);     //LEG F UP
```

```
    HAL_Delay(smallDelay);
    setPos(FT, FTN+upStepsT);
    HAL_Delay(smallDelay);
    setPos(FC, FCN+fwdStepsC);    //LEG F FWD
    HAL_Delay(bigDelay);
    setPos(FF, FFN);    //LEG F DOWN
    HAL_Delay(smallDelay);
    setPos(FT, FTN);
    HAL_Delay(smallDelay);

         //LEG5
    setPos(EF, EFN + upStepsF);
    HAL_Delay(smallDelay);
    setPos(ET, ETN + upStepsT);
    HAL_Delay(smallDelay);
    setPos(EC, ECN+fwdStepsC);
    HAL_Delay(bigDelay);
    setPos(EF, EFN);    //LEG E DOWN
    HAL_Delay(smallDelay);
    setPos(ET, ETN);
    HAL_Delay(smallDelay);


    //LEG2
    setPos(BF, BFN+upStepsF);
    HAL_Delay(smallDelay);
    setPos(BT, BTN+upStepsT);
    HAL_Delay(smallDelay);
    setPos(BC, BCN-fwdStepsC);
    HAL_Delay(bigDelay);
    setPos(BF, BFN);
    HAL_Delay(smallDelay);
    setPos(BT, BTN);
    HAL_Delay(smallDelay);

    //LEG3
    setPos(CF, CFN + upStepsF);
    HAL_Delay(smallDelay);
    setPos(CT, CTN + upStepsT);
 HAL_Delay(smallDelay);
    setPos(CC, CCN-fwdStepsC);
    HAL_Delay(bigDelay);
    setPos(CF, CFN);    //LEG C DOWN
    HAL_Delay(smallDelay);
    setPos(CT, CTN);
    HAL_Delay(smallDelay);
```

```
     //LEG4
     setPos(DF, DFN + upStepsF);
     HAL_Delay(smallDelay);
     setPos(DT, DTN + upStepsT);
     HAL_Delay(smallDelay);
     setPos(DC, DCN+fwdStepsC);
     HAL_Delay(bigDelay);
     setPos(DF, DFN); //LEG D DOWN
     HAL_Delay(smallDelay);
     setPos(DT, DTN);
     HAL_Delay(smallDelay);



     setPos(AC, ACN);
     HAL_Delay(bigDelay);
     setPos(BC, BCN);
     HAL_Delay(bigDelay);
     setPos(CC, CCN);
     HAL_Delay(bigDelay);
     setPos(DC, DCN);
     HAL_Delay(bigDelay);
     setPos(EC, ECN);
     HAL_Delay(bigDelay);
     setPos(FC, FCN);
     HAL_Delay(bigDelay);

}

void walkRevG3(){


     //LEG6
     setPos(FF, FFN+upStepsF);     //LEG F UP
     HAL_Delay(smallDelay);
     setPos(FT, FTN+upStepsT);
     HAL_Delay(smallDelay);
     setPos(FC, FCN+fwdStepsC);    //LEG F FWD
     HAL_Delay(bigDelay);
     setPos(FF, FFN);    //LEG F DOWN
     HAL_Delay(smallDelay);
     setPos(FT, FTN);
     HAL_Delay(smallDelay);

     //LEG1
     setPos(AF, AFN + upStepsF);
     HAL_Delay(smallDelay);
     setPos(AT, ATN + upStepsT);
     HAL_Delay(smallDelay);
     setPos(AC, ACN- fwdStepsC);
     HAL_Delay(bigDelay);
     setPos(AF, AFN);    //LEG A DOWN
     HAL_Delay (smallDelay);
     setPos(AT, ATN);
     HAL_Delay(smallDelay);


     setPos(AC, ACN);
     HAL_Delay(bigDelay);
     setPos(FC, FCN);
     HAL_Delay(bigDelay);

}
void sitDownV2(){
     int afsd=AFN; int bfsd=BFN; int cfsd=CFN;
     int dfsd=DFN; int efsd=EFN; int ffsd=FFN;

     while (1){
     if (afsd == 1950 && bfsd ==2050 && cfsd==2050 && dfsd
==2100 && efsd ==2100 && ffsd ==2100)
          break;
  if(afsd +1 <1950)
               afsd =afsd +1;
     if (bfsd + 1 < 2050)
               bfsd = bfsd +1;
     if (cfsd + 1 < 2050)
               cfsd = cfsd +1;
     if (dfsd + 1 < 2100)
               dfsd = dfsd +1;
     if (efsd + 1 < 2100)
               efsd = efsd +1;
     if (ffsd + 1 < 2100)
               ffsd = ffsd +1;
     setPos(AF, afsd);   //LEG A DOWN
     HAL_Delay(20);
     setPos(EF, efsd);   //LEG E DOWN
     HAL_Delay(20);
     setPos(DF, dfsd); //LEG D DOWN
     HAL_Delay(20);
     setPos(BF, bfsd);   //LEG B DOWN
     HAL_Delay(20);
     setPos(CF, cfsd);   //LEG C DOWN
     HAL_Delay(20);
     setPos(FF, ffsd);   //LEG F DOWN
```

```
          HAL_Delay(20);
          }

          setPos(AT, 2200); HAL_Delay(smallDelay);
          setPos(BT, 2200);   HAL_Delay(smallDelay);
          setPos(CT, 2200); HAL_Delay(smallDelay);
          setPos(DT, 2200); HAL_Delay(smallDelay);
          setPos(ET, 2200); HAL_Delay(smallDelay);
          setPos(FT, 2200); HAL_Delay(smallDelay);
     }
     void sitDown(){

          setPos(AF, 1950);   //LEG A DOWN
          HAL_Delay (bigDelay);
          setPos(EF, 2100);   //LEG E DOWN
          HAL_Delay(bigDelay);
          setPos(DF, 2100); //LEG D DOWN
          HAL_Delay(bigDelay);
          setPos(BF, 2050);   //LEG B DOWN
          HAL_Delay(bigDelay);
          setPos(CF, 2050);   //LEG C DOWN
          HAL_Delay(bigDelay);
          setPos(FF, 2100);   //LEG F DOWN
          HAL_Delay(bigDelay);

          setPos(AT, 2200); HAL_Delay(smallDelay);
          setPos(BT, 2200);   HAL_Delay(smallDelay);
          setPos(CT, 2200); HAL_Delay(smallDelay);
          setPos(DT, 2200); HAL_Delay(smallDelay);
          setPos(ET, 2200); HAL_Delay(smallDelay);
          setPos(FT, 2200); HAL_Delay(smallDelay);
     }

     void standUpV2(){

          int afsd=AFN; int bfsd=BFN; int cfsd=CFN;
          int dfsd=DFN; int efsd=EFN; int ffsd=FFN;

          setPos(AT, ATN); HAL_Delay(smallDelay);
          setPos(BT, BTN); HAL_Delay(smallDelay);
          setPos(CT, CTN); HAL_Delay(smallDelay);
          setPos(DT, DTN); HAL_Delay(smallDelay);
          setPos(ET, ETN); HAL_Delay(smallDelay);
          setPos(FT, FTN); HAL_Delay(smallDelay);


          while (1){
```

```
     if (afsd >= AFN && bfsd >= BFN && cfsd>= CFN && dfsd >= DFN
&& efsd >= EFN && ffsd >=FFN){
               setPos(AF, AFN);    //LEG A DOWN
               HAL_Delay(20);
               setPos(EF, EFN);    //LEG E DOWN
               HAL_Delay(20);
               setPos(DF, DFN); //LEG D DOWN
               HAL_Delay(20);
               setPos(BF, BFN);    //LEG B DOWN
               HAL_Delay(20);
               setPos(CF, CFN);    //LEG C DOWN
               HAL_Delay(20);
               setPos(FF, FFN);    //LEG F DOWN
               HAL_Delay(20);
               break;
     }

  if(afsd + 20 <AFN)
               afsd =afsd +20;
     if (bfsd + 20 < BFN)
               bfsd = bfsd +20;
     if (cfsd + 20 < CFN)
               cfsd = cfsd +20;
     if (dfsd + 20 < DFN)
               dfsd = dfsd +20;
     if (efsd + 20 < EFN)
               efsd = efsd +20;
     if (ffsd + 20 < FFN)
               ffsd = ffsd +20;

     setPos(AF, afsd);
     HAL_Delay(20);
     setPos(EF, efsd);
     HAL_Delay(20);
     setPos(DF, dfsd);
     HAL_Delay(20);
     setPos(BF, bfsd);
     HAL_Delay(20);
     setPos(CF, cfsd);
     HAL_Delay(20);
     setPos(FF, ffsd);
     HAL_Delay(20);
     }

}
void standUpV3(){
     uint8_t serialBytes [4];
```

```
	serialBytes[0] = 0x9f;
  serialBytes[2] = 0x12 & 0x7F; //lsb
	serialBytes[3] =  0x00 & 0x7F;
	HAL_UART_Transmit(&huart1, &serialBytes[0], 1, 1000);
	//HAL_Delay(20);
	HAL_UART_Transmit(&huart1, &serialBytes[2], 1, 1000);
	//HAL_Delay(20);
	HAL_UART_Transmit(&huart1, &serialBytes[3], 1, 1000);
	//HAL_Delay(20);
	write14Bit(ACN); write14Bit(AFN); write14Bit(ATN);
	write14Bit(BCN); write14Bit(BFN); write14Bit(FTN);
	write14Bit(CCN); write14Bit(CFN); write14Bit(CTN);
	write14Bit(DCN); write14Bit(DFN); write14Bit(DTN);
	write14Bit(ECN); write14Bit(EFN); write14Bit(ETN);
	write14Bit(FCN); write14Bit(FFN); write14Bit(FTN);
}
void standUpPrep(){
	int delaytemp = 500;
	setPos(AT, 2200); HAL_Delay(delaytemp);
	setPos(BT, 2200); HAL_Delay(delaytemp);
	setPos(CT, 2200); HAL_Delay(delaytemp);
	setPos(DT, 2200); HAL_Delay(delaytemp);
	setPos(ET, 2200); HAL_Delay(delaytemp);
	setPos(FT, 2200); HAL_Delay(delaytemp);

	setPos(AF, 2050); HAL_Delay(delaytemp);
	setPos(BF, 2100); HAL_Delay(delaytemp);
	setPos(CF, 2100); HAL_Delay(delaytemp);
	setPos(DF, 2100); HAL_Delay(delaytemp);
	setPos(EF, 2100); HAL_Delay(delaytemp);
	setPos(FF, 2100); HAL_Delay(delaytemp);
}
void standUp(){

  int     smallDelay = 200;
	setPos(AT, ATN); HAL_Delay(smallDelay);
	setPos(BT, BTN); HAL_Delay(smallDelay);
	setPos(CT, CTN); HAL_Delay(smallDelay);
	setPos(DT, DTN); HAL_Delay(smallDelay);
	setPos(ET, ETN); HAL_Delay(smallDelay);
	setPos(FT, FTN); HAL_Delay(smallDelay);

	setPos(AF, AFN);    //LEG A DOWN
	HAL_Delay (smallDelay);
	setPos(FF, FFN);    //LEG F DOWN
	HAL_Delay(smallDelay);
	setPos(EF, EFN);    //LEG E DOWN
```

```
     HAL_Delay(smallDelay);
     setPos(BF, BFN);    //LEG B DOWN
     HAL_Delay(smallDelay);
     setPos(CF, CFN);    //LEG C DOWN
     HAL_Delay(smallDelay);
     setPos(DF, DFN); //LEG D DOWN
     HAL_Delay(smallDelay);



}

void setAcc(){
     i2cData[0]=0x31;  // G_PWR_MGM
     i2cData[1]=0x01;
     HAL_I2C_Master_Transmit(&hi2c1, 0xA6, i2cData, 2, 10);
     i2cData[0]=0x2D;  //G_SMPLRT_DIV
     i2cData[1]=0x08;
     HAL_I2C_Master_Transmit(&hi2c1, 0xA6, i2cData, 2, 10);
     HAL_Delay(10);
}
void readAcc(){
     static int axis[3];
  uint8_t buff[6];
     i2cData[0]=0x32;
     HAL_I2C_Master_Transmit(&hi2c1, 0xA6, i2cData, 1, 50);
     HAL_Delay(10);
     //HAL_I2C_Master_Sequential_Receive_IT(&hi2c1, 0xD0, buff,
8, 100);
     HAL_I2C_Master_Receive(&hi2c1, 0xA6, buff, 6, 100);
     HAL_Delay(100);

  axis[0] = (((uint8_t)buff[1] << 8) | (uint8_t)buff[0]) ;
     axis[1] = (((uint8_t)buff[3] << 8) | (uint8_t)buff[2]) ;
     axis[2] = (((uint8_t)buff[5] << 8) | (uint8_t)buff[4]) ;

     acc_x= axis[0];
     acc_y= axis[1];
     acc_z= axis[2];
     if(acc_x > 255)
          acc_x=acc_x-65536;
     if(acc_y > 255)
          acc_y=acc_y-65536;
  if(acc_z > 255)
          acc_z=acc_z-65536;
}
```

```
void multiTarget(){
     uint8_t serialBytes [4];
     serialBytes[0] = 0x9f;
  serialBytes[2] = 0x12 & 0x7F; //lsb
     serialBytes[3] =  0x00 & 0x7F;
     HAL_UART_Transmit(&huart1, &serialBytes[0], 1, 1000);
     //HAL_Delay(20);
     HAL_UART_Transmit(&huart1, &serialBytes[2], 1, 1000);
     //HAL_Delay(20);
     HAL_UART_Transmit(&huart1, &serialBytes[3], 1, 1000);
     //HAL_Delay(20);
     write14Bit(ACN); write14Bit(AFN+upStepsF);
write14Bit(ATN+upStepsT);
     write14Bit(BCN); write14Bit(BFN+upStepsF);
write14Bit(FTN+upStepsT);
     write14Bit(CCN); write14Bit(CFN+upStepsF);
write14Bit(CTN+upStepsT);
     write14Bit(DCN); write14Bit(DFN+upStepsF);
write14Bit(DTN+upStepsT);
     write14Bit(ECN); write14Bit(EFN+upStepsF);
write14Bit(ETN+upStepsT);
     write14Bit(FCN); write14Bit(FFN+upStepsF);
write14Bit(FTN+upStepsT);

}

void leg1Play(){
     //LEG A UP

     setPos(AF, AFN + upStepsF);
     HAL_Delay(500);
     setPos(AT, ATN + upStepsT);
     HAL_Delay(5000);
     setPos(AF, AFN);    //LEG A DOWN
     HAL_Delay (500);
     setPos(AT, ATN);
     HAL_Delay(1500);

}
void leg2Play(){
     setPos(BF, BFN+upStepsF);     //LEG B UP
     HAL_Delay(500);
     setPos(BT, BTN+upStepsT);
     HAL_Delay(5000);
     setPos(BF, BFN);    //LEG B DOWN
     HAL_Delay(500);
     setPos(BT, BTN);
     HAL_Delay(500);
     HAL_Delay(1000);
}
void leg3Play(){
     //LEG C UP
  setPos(CF, CFN + upStepsF);
     HAL_Delay(500);
     setPos(CT, CTN + upStepsT);
  HAL_Delay(5000);
     setPos(CF, CFN);    //LEG C DOWN
     HAL_Delay(500);
     setPos(CT, CTN);
     HAL_Delay(1500);
}
void leg4Play(){
     setPos(DF, DFN+upStepsF);     //LEG D UP
     HAL_Delay(500);
     setPos(DT, DTN+upStepsT);
     HAL_Delay(5000);
     setPos(DF, DFN); //LEG D DOWN
     HAL_Delay(500);
     setPos(DT, DTN);
     HAL_Delay(1500);

}
void leg5Play(){
     //LEG E UP
     setPos(EF, EFN + upStepsF);
     HAL_Delay(500);
     setPos(ET, ETN + upStepsT);
     HAL_Delay(5000);
     setPos(EF, EFN);    //LEG E DOWN
     HAL_Delay(500);
     setPos(ET, ETN);
     HAL_Delay(1500);
}
void leg6Play(){
     setPos(FF, FFN+upStepsF);     //LEG F UP
     HAL_Delay(500);
     setPos(FT, FTN+upStepsT);
     HAL_Delay(5000);
     setPos(FF, FFN);    //LEG F DOWN
     HAL_Delay(500);
     setPos(FT, FTN);
     HAL_Delay(1500);
}
#endif
```