Motion Controlled Robotic Arm

ECE 395 Spring 2017 Project

Sanchit Anand & Yujia Qiu

Introduction

For our project in ECE 395, we decided to make a motion controlled robotic arm that is capable of detecting the orientation of a person's hand, and attempting to replicate the motion. Our project was based on another project from the Fall of 2013 - Human Controlled Robotic Arm and Legs Movement, and we attempted to extend the degrees of motion of the arm.

At the end of the project, major changes were made to the motion filtering algorithms, to support a new system of kinematics. This report is written in the hope that future students will be able to expand upon our work.

Schematic

The schematic files we used include a different version of the LPC ARM processor, with almost identical functionality.

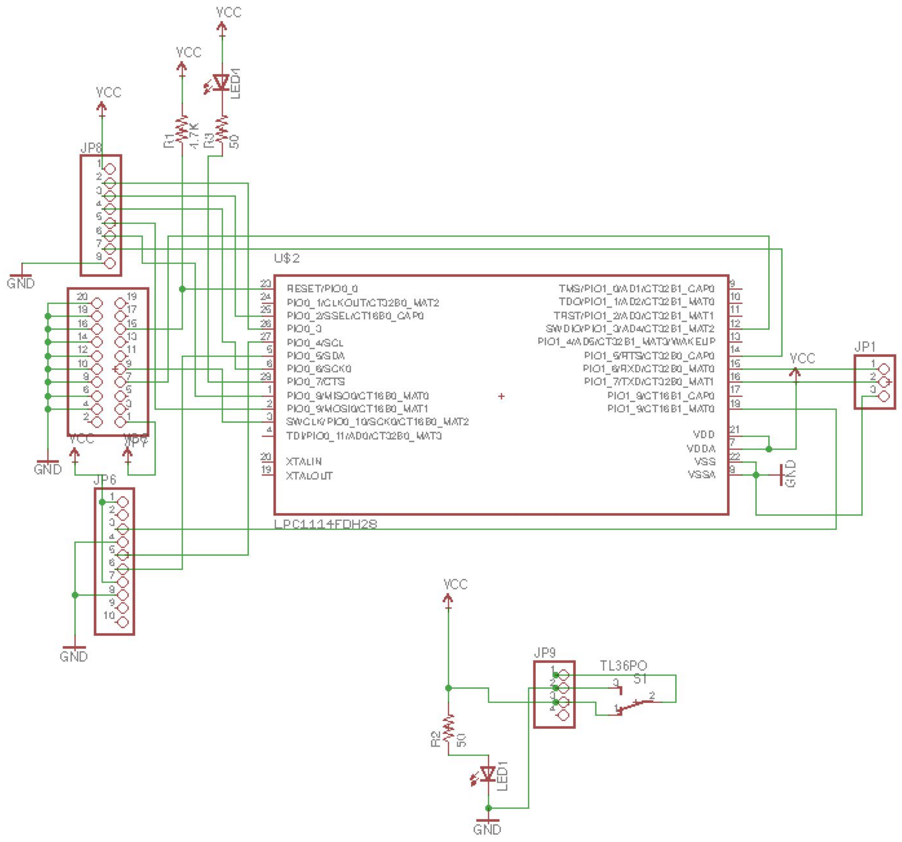


Figure: Schematic Diagram

The 8 pins jumper is to connect the bluetooth breakout, the 10 pins jumper is to connect sensor breakout. The 20 pins jumper is to connect JTAG. The “JP1” is for switch which controls the power on or off. The “JP9” is connect to the power cell.

PCB

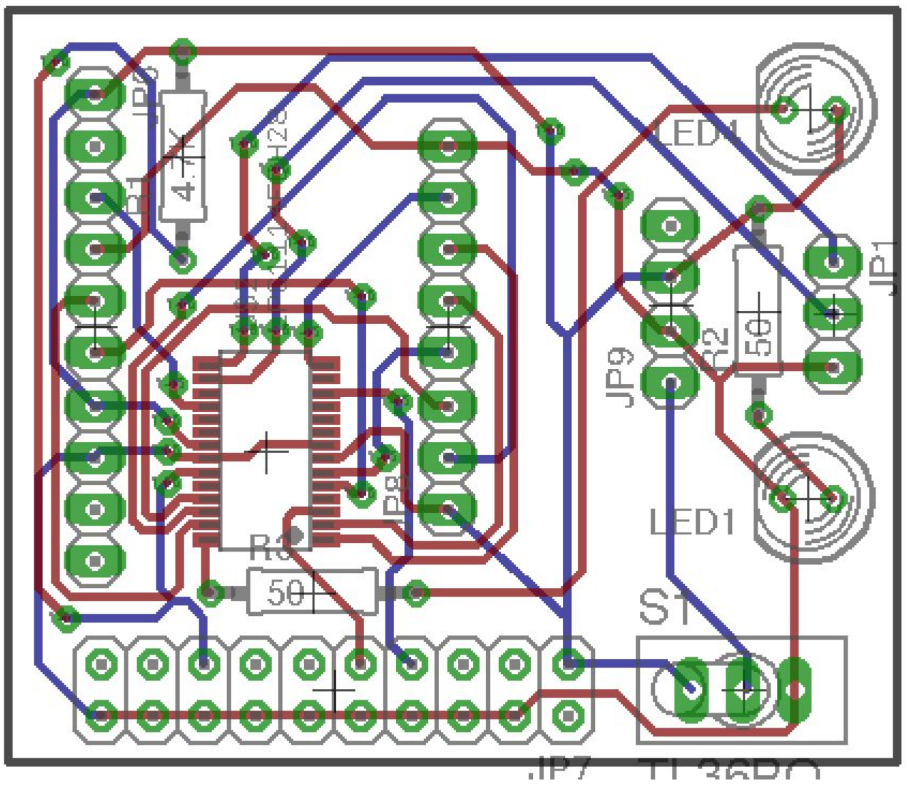


Figure: PCB

This brd file is transferred from schematic. And the figure below is the actual picture of the pcb board.

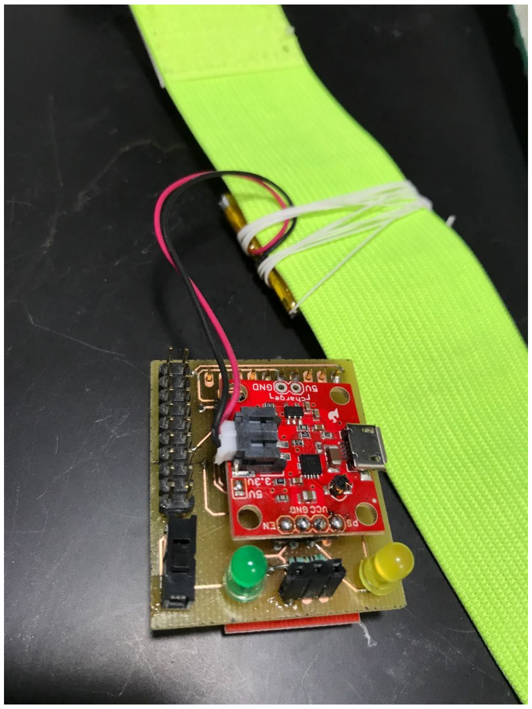


Figure: pcb board picture

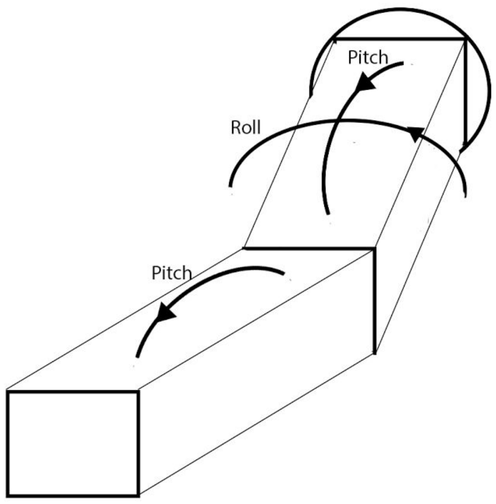
Operation

The project hardware is split up between two different boards. There is a main prototype board that controls the motors, via USART protocol. This main board contains a bluetooth module(NRF24l01+) that polls for data from the motion pads at regular intervals. This angular data is transformed into motor values, and transmitted via USART. There is an LED attached for debugging purposes, as well as connections for downloading code via JTAG module.

The motion pads consist of an battery pack, connected to a power chip that regulates voltage at 3.3V. It also contains a six axis IMU unit, the MPU-6050, for taking measurements of motion along the pitch and roll axes. The microprocessor performs complementary filtering on the data. Finally the bluetooth module (NRF24l01+), transmits the pitch and roll angles to the main board.

There are 4 motion pads in total, one for each shoulder joint, and one for each elbow joint. The 2 shoulder pads measure angles along pitch and roll axes, and transmit to the main board. Then the two separate motors in the shoulder joint get actuated by their respective angle. The 2 elbow pads measure motion along the pitch direction and transmit it to the main board, where the pitch angle from the shoulder joint is subtracted from it. The elbow motor is actuated by the resulting angle.

A short diagram is presented below to help understand the different angles used.



Conclusion

The project yielded satisfactory results as determined by us. Lots of improvements can still be made on it. First of all, by using a 9 degree of freedom IMU, instead of a 6 degree of freedom IMU (The MPU-6050 used for this project), one would be able to measure all 3 axes of motion, instead of just two, and with a higher precision as well. One of the main limitations of using just 6 degrees of freedom is that the error in angles increases as the combined pitch and roll angles get higher. Secondly, by learning how to use the inbuild proprietary DMP software on the MPU-6050, one would be able to offload all the filtering onto the IMU itself, and obtain much more accurate data in the process.

Finally, We found that the servos used in this project were liable to decreased performance after operating for some time. We are unsure whether it is the result of wear and tear, whether it is a fundamental property of the specific brand we used or an issue with the code being used. In either case, it is recommended that more research is done while selecting and using servos and motors to use in future projects.

Appendix : Hardware

NXP LPC1114 Cortex M0 Microprocessor (x5)
The processors that perform all of the filtering and calculations, as well as communications with various peripheral devices.
http://www.nxp.com/documents/data_sheet/LPC111X.pdf

MPU6050 6-Axis Accelerometer/Gyroscope
These are the main sensors. The inbuilt gyroscope and accelerometers measures the orientation of the board with respect to gravity. These measurements are stored in registers of the chip, and can be accessed by microprocessors via I2C protocol. The communication rate is 100Kbits/s. Documentation for this chip is available at
http://www.invensense.com/mems/gyro/mpu6050.html

NRF24L01+ Transceiver
This is a 2.4Ghz radio frequency transmitter/receiver. It is present on both the satellite and the main board, providing a form of wireless communication between the boards. The transceiver operates in two modes, one for receiving (RX) and one for transmission (TX). This can be configured by commands from the microprocessor through Serial Peripheral Interface (SPI) protocol.
http://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01

Dynamixel AX-12a Servos
The AX12 servos are the actuators of the robot, serving as rotational joints based on the commands sent by the microprocessor. These servos work on 9-12V power source and communicate with the microprocessor via a single line. The ID of the servos determines which servo along the series line to receive the instruction as passed from the processor. The servos utilize half-duplex communication protocol.

http://www.crustcrawler.com/products/bioloid/docs/AX-12.pdf

Appendix : Code

complementary.c : C file containing the majority of filtering code.

```
#include "complementary.h"

float x_filter=0, y_filter=0, z_filter=0, angle_x=0, angle_y=0, angle_z=0 ;
float pitch , yaw,  roll ;
float sign;
void complementary_filter(float * Buf,float accel_x, float accel_y, float accel_z, float gyro_x, float gyro_y, float
gyro_z,float dt)
{


          x_filter = 0.9*x_filter +  0.1*accel_x;
          z_filter = 0.9*z_filter + 0.1*accel_z;
          y_filter = 0.9*y_filter + 0.1*accel_y;
          //acc_angle_y = atan2(x_filter, -z_filter) * 180/3.14159265358979323 ;
          //angle_y = 0.98*(gyro_y*dt + angle_y) + 0.02*acc_angle_y;

          //acc_angle_x = atan2(y_filter, -z_filter) * 180/3.14159265358979323 ;
          //angle_x = 0.98*(gyro_x*dt + angle_x) + 0.02*acc_angle_x;

          //acc_angle_z = atan2(x_filter,y_filter) * 180/3.14159265358979323 ;
          //angle_z = 0.98*(gyro_z*dt + angle_z) + 0.02*acc_angle_z;
          angle_z = angle_z + gyro_z * dt;
          sign = z_filter > 0?-1:1;
          pitch = atan2(-x_filter,sqrt(y_filter*y_filter + z_filter*z_filter)) * 180/3.14159265358979323 ;
      roll =  atan2(y_filter, sign* sqrt(z_filter*z_filter + 0.01*x_filter*x_filter))* 180/3.14159265358979323;
          angle_x = 0.98*(gyro_x*dt + angle_x) + 0.02*roll;
          angle_y = 0.98*(gyro_y*dt + angle_y) + 0.02*pitch;
          Buf[0] = angle_x;
          Buf[1] = angle_y;
          //Buf[2] = angle_z;
}
```

main.c : Code that handles the major initialization and communication (Only including some of the code for brevity).

```
#define TRANSMITTER 0x0
#define RECEIVER 0x1
#define SET_ID      0x2


/*** CHOOSE BOARD ***/
/********************/

//#define BOARD TRANSMITTER
#define BOARD RECEIVER
//#define BOARD SET_ID
/********************/

int main(void) {


    delay32Ms(1, 2000);//allow time for devices to power up
    LED_DIR_OUT;
    LPC_GPIO1->DIR &= ~(1<<5); //input mode
    SER_init();
    Led_Blink();
    //initialize SPI protocol
    SSP_IOConfig(0);
    SSP_Init(0);

    Led_Blink();

#if BOARD == TRANSMITTER
    AX12_begin(_1MHZ);
    NRF24L01_Init(_RX_MODE, _CH1, _1Mbps, Address, _Address_Width, _Buffer_Size);
              while(1)
    {




                    NRF24L01_Set_TX_Address(Address, _Address_Width);
                    NRF24L01_Set_RX_Pipe(0, Address, _Address_Width, _Buffer_Size);

                                      for(i=0; i < _Buffer_Size; i++) Buf[i] = '\0';
                             strcpy(Buf,"HELLO\0");
                    //Transmit Request
                    NRF24L01_Send(Buf);
                    delay32Ms(1,5);
                    //Receive Response
                    NRF24L01_Receive(Buf);

                     value = (float*) Buf;

                    if(value[1] == 0.0 ) {}
                    else{
                    temp = value[1];
                    if(temp <-60) temp = -60;
                    if(temp > 60) temp = 60;
                    //value[1] = value[1];
                    temp = (150 + temp)*1023/300;
                    i = temp;
                    //printf("%d,",i);
                    AX12_move(14,i);

                    }
                    delay32Ms(1,1);

                    if(value[0] == 0.0) {}

                    else{
                    pitch = value[0];
                    temp = value[0];
                    if(temp <-90) temp = -90;
                    if(temp > 90) temp = 90;
                    //value[1] = value[1] - 30;
                    temp = ( 150 +  temp )*1023/300;
                    i = temp;
          AX12_move(15,i);
                     }

   }
#endif

#if BOARD == RECEIVER
    NRF24L01_Init(_RX_MODE, _CH1, _1Mbps, Address3, _Address_Width, _Buffer_Size);
    NRF24L01_DRint_Init();
    I2CInit(I2CMASTER);
```

```
                while(MPU6050_init() != 0)
    {
          for(i=0;i<100000;i++);
    }

          delay32Ms(1,500);
    MPU6050_setZero();
    init_timer32(0, 48); //us
    enable_timer32(0);
    while(1)
    {

          gyroy =  MPU6050_getGyroRoll_degree();
          gyrox =  MPU6050_getGyroX();
          gyroz = MPU6050_getGyroZ();
          dt = (float)read_timer32(0) / 1000000;

          //printf("%f\r\n",gyroz);
          //gyro = gyro;
          reset_timer32(0);
          enable_timer32(0);
          x = MPU6050_getAccel_x();
          z = MPU6050_getAccel_z();
          y = MPU6050_getAccel_y();
          complementary_filter(list,x,y,z,gyrox,gyroy, gyroz, dt);
                   cvalue = (char*) list;
                   //Receive Response
                   if(LPC_GPIO1->MASKED_ACCESS[1<<5] == 0)
                   {
                   //NRF24L01_Print_Register();
                   NRF24L01_Receive(Buf);
                   if(strncmp(Buf,"HELLO",_Buffer_Size)==0)
                   {
                            memset(Buf,0,_Buffer_Size);
                            for(i=0;i<sizeof(float)*2;i++)
                                     Buf[i] = cvalue[i]

                            NRF24L01_Send(Buf);
                            }
                   }
          }

#endif
#if BOARD == SET_ID
    AX12_begin(_1MHZ);
    AX12_setID(BROADCAST_ID, ID);
    LED_ON;
    AX12_ledStatus(ID, ON);      //Check if loaded
    AX12_move(ID, 512);
#endif

    }
```

mpu6050.c: added more functions on top of existing class.

```
void MPU6050_setZero(){

    uint16_t i;
    uint16_t n = 1000;
    int32_t tmp_y= 0,tmp_x = 0, tmp_z = 0;
    int32_t ac_x=0, ac_y = 0, ac_z = 0;
    for(i=0;i<n;i++){
          tmp_y += MPU6050_getGyroRoll_raw();
    }
    for (i=0;i<n;i++){
          tmp_z += MPU6050_getGyroZ_raw();
    }
    for(i=0;i<n;i++){

          tmp_x += MPU6050_getGyroX_raw();
    }
    for (i=0;i<n;i++){
          ac_x += MPU6050_getAccel_x_raw();
    }
    for (i=0;i<n;i++){
          ac_y += MPU6050_getAccel_y_raw();
    }
    for (i=0;i<n;i++){
          ac_z += MPU6050_getAccel_z_raw();
    }
    tmp_y /= n;
    tmp_x /= n;
    tmp_z /= n;
    ac_y /= n;
    ac_x /= n;
    ac_z /= n;
    zero_gyro_roll = tmp_y;
    zero_X = tmp_x;
    zero_Z = tmp_z;
    zero_acc_x = ac_x;
    zero_acc_z = ac_z + 2048;
    zero_acc_y = ac_y;

}

int16_t MPU6050_getAccel_y_raw(){


          int16_t tmp;
          uint8_t state;

          I2CWriteLength               = 2;
          I2CReadLength      = 2;   //1;
          I2CMasterBuffer[0]     = MPU6050_ADDRESS;
          I2CMasterBuffer[1]     = MPU6050_RA_ACCEL_YOUT_H;
          I2CMasterBuffer[2]     = MPU6050_ADDRESS | RD_BIT;

          state = I2CEngine();
          if(state != I2C_OK) return 1;

          tmp = (I2CSlaveBuffer[0] << 8) | I2CSlaveBuffer[1];

    return tmp - zero_acc_y;
}

float MPU6050_getAccel_y(){

    int16_t tmp;
    float y;
    tmp = MPU6050_getAccel_y_raw();
    y = (float) tmp/2048;
    return y;

}

int16_t MPU6050_getGyroX_raw(void)
```

```
{
    int16_t tmp;
    uint8_t state;

    I2CWriteLength           = 2;
    I2CReadLength            = 2; //1;
    I2CMasterBuffer[0]     = MPU6050_ADDRESS;
    I2CMasterBuffer[1]     = MPU6050_RA_GYRO_XOUT_H;
    I2CMasterBuffer[2]     = MPU6050_ADDRESS | RD_BIT;

    state = I2CEngine();
    if(state != I2C_OK) return 1;

    tmp = (I2CSlaveBuffer[0]<<8) | I2CSlaveBuffer[1];

    return tmp - zero_X;

}
float MPU6050_getGyroX(void)
{
    int16_t tmp;
    float roll;

    tmp = MPU6050_getGyroX_raw();
    roll = (tmp  / 65.536 );
    return roll;
}

int16_t MPU6050_getGyroZ_raw(void)
{
    int16_t tmp;
    uint8_t state;

    I2CWriteLength           = 2;
    I2CReadLength            = 2; //1;
    I2CMasterBuffer[0]     = MPU6050_ADDRESS;
    I2CMasterBuffer[1]     = MPU6050_RA_GYRO_ZOUT_H;
    I2CMasterBuffer[2]     = MPU6050_ADDRESS | RD_BIT;

    state = I2CEngine();
    if(state != I2C_OK) return 1;
    tmp = (I2CSlaveBuffer[0]<<8) | I2CSlaveBuffer[1];

    return tmp - zero_Z;

}

float MPU6050_getGyroZ(void)
{
    int16_t tmp;
    float roll;

    tmp = MPU6050_getGyroZ_raw();
    roll = (tmp  / 65.536 );
    return roll;
}
```

nrf24l01.c: added one function on top of existing class

```
void NRF24L01_Init_Auto_Ack(char Device_Mode, char CH, char DataRate,
          char *Address, char Address_Width, char Size_Payload) {


    NRF24L01_CE_OUT; // Set Port DIR out
    // Disable Enhanced ShockBurst
          NRF24L01_WriteReg(W_REGISTER | EN_AA, 0x1);
          NRF24L01_WriteReg(W_REGISTER | SETUP_RETR, 0x1);
    // RF output power in TX mode = 0dBm (Max.)
    // Set LNA gain
    NRF24L01_WriteReg(W_REGISTER | RF_SETUP, 0x07 | DataRate); //0b00000111 | DataRate);

    NRF24L01_Set_Address_Width(Address_Width);

    NRF24L01_Set_RX_Pipe(0, Address, Address_Width, 0);
    NRF24L01_Set_RX_Pipe(1, Address, Address_Width, Size_Payload);
    NRF24L01_Set_CH(CH);

    NRF24L01_Set_TX_Address(Address, Address_Width); // Set Transmit address

    // Bits 4..6: Reflect interrupts as active low on the IRQ pin
    // Bit 3: Enable CRC
    // Bit 2: CRC 1 Byte
    // Bit 1: Power Up
    NRF24L01_WriteReg(W_REGISTER | CONFIG, 0x0A | Device_Mode);//0b00001010 | Device_Mode);

    delay32Us(1, 1500);
    if(Device_Mode == _RX_MODE) //take it out of standby
    {    NRF24L01_CE_HIGH;
          delay32Us(1, 130);
    }
    //Delay_us(1500);
}
```