REPORT ON ROBOCON 2019

MUFFAKHAM JAH COLLEGE OF ENGINEERING AND TECHNOLOGY

INDEX

|  |  |  |
| --- | --- | --- |
| Sl.No | Particulars | Page |
| 1 | Abstract,Components | 1 |
| 2 | Wireless Ps2 | 2 |
| 3 | Messenger robot-1 | 20 |
| 4 | Messenger robot-2 | 30 |
| 5 | Eagle software | 35 |
| 6 | MPU6050,OBLU sensor | 40 |
| 7 | Matlab and simulink | 69 |

MESSENGER ROBOT-1

**Abstract:**

* The task of MR1 was to pass some hurdle carrying gerege,then pass on that gerege toMR2 which will carry on.MR1 should thenpick up the shaghai and throwit at certain distance, depending on the orientation of shaghai afterlanding, points aregiven to the teams.

As soon as the problem statement was released we divided the problem statement into the following tasks:

1. Locomotion
2. Grabbing
3. Picking
4. Launching

**Control:**

* A ps2 controller can be used to control a wheeled robot after interfacing it with a microcontroller.
* Interfacing the PS2console with a microcontroller gives the user full control to program and use the buttons as desired.
* The range of wireless ps2s can be optimized by using differentwireless protocol between the Ps2 and the microcontroller.
* Here in our project the PS2 was interfaced with atmega8 microcontroller and then it was converted to wireless using a Bluetooth module calling it as a transmitter.
* On the other side another microcontroller was used as a receiver.
* The communicating protocols involved were UART and SPI.

**COMPONENTS**

1. **Motors**

* 4 Planetary gear motors(500rpm) for locomotion
* 2 PW-TYT motors for lifting and loading

1. **Pistons**

* 2 of stroke length 10cm each.

1. **Motor drivers**

* 6 Hercules single drive motor drivers with current rating of 13Amp.
* 2 L298N dual drive motor drivers of current rating 2A.

1. **Encoder** – 2 rotary incremental encoders with precision of 1500 counts per rotation
2. **Arduino Mega:** 54 pins,atmega 2560, dc power jack,pwm pins of 15, current rating of 20mA at i/o pins.

**PS2 CONTROLLER**

Mr1 was controlled using ps2 controller. Different buttons were assigned for different tasks based on the requirement i.e analog buttons for speed control and locomotion, pad buttons for other mechanisms controlling etc.

**Wireless Ps2 from wired Ps2:**

# wired ps2 controller

A ps2 controller DualShock was used to control the Robot i.e. MR1

The requirement was choosing a controller with good no. buttons which can be interfaced with our bot easily. And the Ps2 controller was one of the best thing matching out requirements and is easily available in the market.

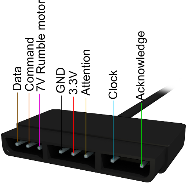
The very first task was interfacing the controller with a microcontroller so as to use it as we desired.

The microcontroller used is ATMega16 which is a 40 pin IC manufactured by ATMEL. With 16KB of size for the program

# interfacing with the microcontroller

After a proper research on the PS2 joystick, it was found that the ps2 uses SPI communication to communicate with the microcontroller

The pin out of the PS2 controller comprises of 8pins. Namely,

1) Data

2) Command

3) Rumble Motor

4) Ground

5) VCC

6) Attention

7) Clock

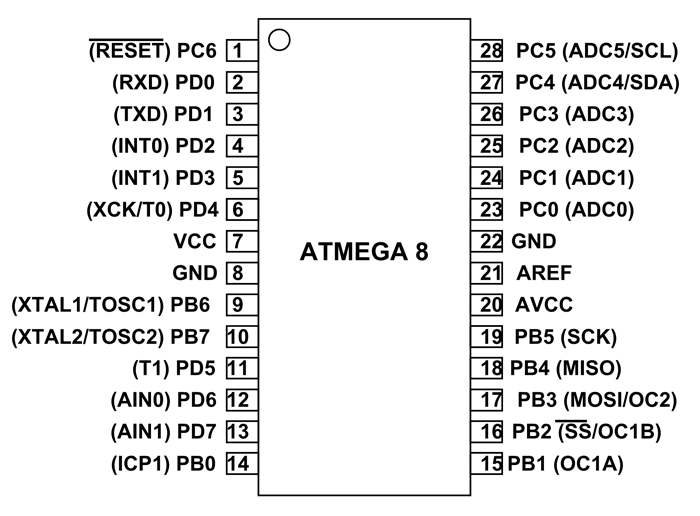
8) Acknowledgement

Out of all these pins the Data, Command, Attention and Clock pin are used in the SPI communication with the microcontroller, VCC and GND pin are used for powering up the controller

Interfacing the controller with Arduino boards became much easy because of the libraries available on the internet. The requirement in our case was making the size as small as possible which made us to go for the microcontroller without onboard components and dropped the option of using the Arduino

# microcontroller

ATmega 16 microcontroller was used for the communication, later it was replaced by Atmega8 looking upon the size of the circuit we desired

Atmega 8 is a 28 pin IC which provides 8KB in system programmable flash memory.

There are 3 ports available with the Microcontroller i.e. PORT B, PORT C, PORT D out of which PORT B has got the pin used for the SPI communication i.e. the communication between the PS2 controller and the microcontroller.

# interfacing

SPI stands for SERIAL PERIPHERAL INTERFACE which is one of the protocols inside a microcontroller that involves the transmit and receive of data serially.

Now that we were familiar with the type of communication, we needed to find a way to connect it to the microcontroller. Soldering it directly to the microcontroller is not an effective way so while finding the means of connection we decided to go for the PS2 Hubs mounted on a circuit with FRC bases soldered. Thus, making the connections secure and easy

The pins uses are MISO, MOSI, CLK, SS of Atmega16

To have a better check on the program output a 16x2 LCD was used as an output along with the Ps2 controller.

The interface used is CVAVR to program the microcontroller .

The controller sends a byte at the same time as it receives one (full duplex) via serial communication. The clock is held high until a byte is to be sent. It then drops low (active low) to start 8 cycles during which data is simultaneously sent and received. When the clock edge drops low, the values on the line start to change. When the clock goes from low to high, value are actually read. Bytes are transferred LSB (least significant bit) first, so the bits on the left (earlier in time) are less significant. Data to be entered as the command to turn on the analog mode 0x44 in hexadecimalformat is shown. It can be seen through the program clearly for a better understanding.

#### Turn on analog mode

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| byte # | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| **Command (hex)** | 01 | 44 | 00 | **0x01** | **0x03** | 00 | 00 | 00 | 00 |
| **Data (hex)** | FF | F3 | 5A | 00 | 00 | 00 | 00 | 00 | 00 |
| **Section** | header | | | config parameters | | | | | |

The signals coming from the PS2 controller were decoded and were stored in 6 variables in the program were named as data0, data1, data2, data3, data4, data5 according to the commands given in the form of Hexadecimal format and using the clock as the reference.

The PS2 was successfully connected but was not interfaced with the microcontroller. After a good look up on the circuit again it was found that **PULL UP** and **PULL DOWN** resisters are necessary in SPI communication.

Later MISO pin of the microcontroller was pulled up using a PULL UP resister of 10K ohms and the SS pin of the microcontroller was pulled down using a PULL DOWN resister of same value i.e. 10K ohms

The values from the controller were printed on a 16x2 LCD. Which showed the following results from various variables written in the program.

Data0:

Keys Set:PAD\_UP, PAD\_DOWN, PAD\_LEFT, PAD\_RIGHT, START, SELECT

Data1:

Keys set: R1, R2, L1, L2, PAD\_CROSS, PAD\_SQUARE, PAD\_TRIANGLE, PAD\_CIRCLE

Data2:

Keys set: RIGHT STICK \_X AXIS

Data3:

Keys set: RIGHT STICK\_Y AXIS

Data4:

Keys set: LEFT STICK\_X AXIS

Data5:

Keys set: LEFT SICK\_YAXIS

And thus by determining the values from each key the pss2 was successfully interfaced with the microcontroller which can be observed by the indication LED on the PS2 controller which never goes off when the controller is interfaced.

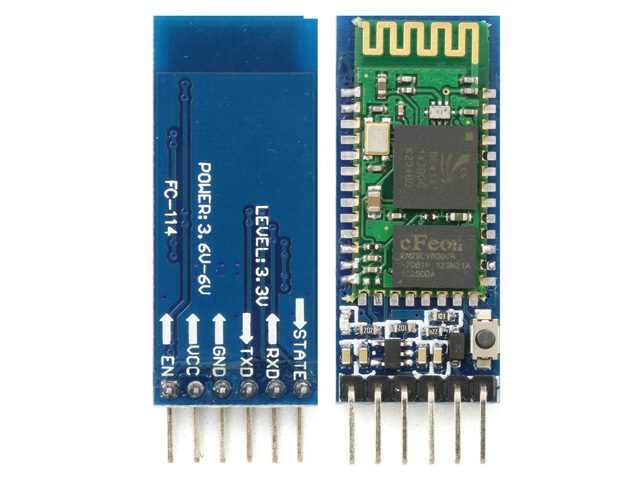
The controller can now be used to control anything that can be attached to our microcontroller .

# wireless transmission of data:

Now that the controller is successfully interfaced with our microcontroller, the task that is left was transmitting the values wirelessly to another microcontroller which was decided to be the ARDUINO MEGA 2560.

Doing a research for the best available wireless modules it was found that RF module has got a good range and stability but the values that can be transmitted were limited. This made us search for another module and looking upon the specifications and our requirements **BLUETOOTH MODULE HC05** was finalized as it fits good.

There are other modules like **HC06** as well but the restriction is that they can’t be the transmitter and will only act like a slave to other Bluetooth enabled devices be it your mobile phone or PC.

While HC05 can act like both master and slave i.e. transmitter and the receiver which needs to be initialized before actually using it in our project.

Taking a look at the pinout of the module it is a module with 6 pins .

1. **State** pin decides whether the module is acting like a master or a slave which can also be programmed during the initialization
2. **Rxd**  pin is the receiver pin which is connected to the Tx pin of the microcontroller.
3. **Txd**  pin is the transmitter pin which is connected to the Rx pin of the microcontroller.
4. **GND** is the ground pin which is grounded with the microcontroller so that the module and the microcontroller both shares a common ground
5. **VCC** pin is for giving the DC supply to the module it can be between 3V to 6V for the Bluetooth module to function.
6. **EN**  is the ENABLE pin which is used during the initialization of the module which is left unused during the transmission of data between the two modules once they are initialized.

**INITIALIZING THE BLUETOOOTH MODULE:**

The Bluetooth module needs to be initialzed before actually using it..

The procedure is called as the **AT COMMAND MODE** in which the module takes in the commands and works as desired.

The interface used to give the input can be Arduino or any other programming software .Arduino is preferable because of its user friendly interface and ease for programming the module.

To get into the AT COMMAND MODE an arduino board is required which has a UART ie RX and TX pins.

Now after making the connections as listed

*Bluetooth moduleArduino uno:*

Txd -----------------🡪TX

Rxd ------------------🡪Rx

Vcc ------------------🡪 5V

Gnd -----------------🡪 GND

EN -------------------🡪5V

Now by holding the reset switch on the module and simultaneously powering up the module it redirects us to the AT COMMAND MODE indicating the blink of the light at regular intervals which is once in 2 seconds .

After buring an empty program in the Arduino and opening the Serial monitor the commands can be entered.The commands to be entered are listed below:

MODULE 1:

**AT+ROLE=1** *which configures the module as a master*

**AT+NAME=master** which names the module as master

**AT+ADDR?***which is entered to display the address of the module*

**AT+UART=9600,1,0**  *which is the BAUD RATE,stop bit and the parity involved in the data transmission.*

**AT+PSWD=*xxxxxx***  *is the password to secure the module from other modules.*

**AT+CMODE=0***which means that the module can connect to only one module whose adderss is specified*

**AT+BIND=xxxx,xx,xxxx***which will be the address of the module acting like the slave* .

MODULE 2:

**AT+ROLE=0** *which configures the module as a slave*

**AT+NAME=slave** which names the module as slave

**AT+ADDR?***which is entered to display the address of the module*

**AT+UART=9600, 1, 0**  *which is the BAUD RATE, stop bit and the parity involved in the data transmission.*

**AT+CMODE=0***which means that the module can connect to only one module whose adderss is specified*

**AT+BIND=xxxx,xx,xxxx***which will be the address of the module acting like the slave*.

**TRANSMISSION OF DATA:**

Now that the Bluetooth modules are initialized it is connected to both the circuits i.e. on the master and the slave. The transmitter is Atmega8 and the receiver is Arduino Mega2560 which was later replaced with arduino Due.

The transmitter is programmed using CVAVR software in which UART and SPI was used for the PS2 controller and the Bluetooth module HC05.

Below attached is the link to the program written in CVAVR.

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 This program was created by the CodeWizardAVR V3.33

 Automatic Program Generator

 © Copyright 1998-2018 Pavel Haiduc, HP InfoTech s.r.l.

 http://www.hpinfotech.com

 Project :Ps2withatmega8

 Version :18.0.5

 Date    : 9/19/2018

 Author  :AsadZeeshan

 Company :

 Comments:

 Chip type               : ATmega8A

 Program type            : Application

 AVR Core Clock frequency: 1.000000 MHz

 Memory model            : Small

 External RAM size       : 0

 Data Stack size         : 256

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*` \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<mega8.h>

#include<spi.h>

#include<alcd.h>

#include<io.h>

#include<stdio.h>

#include<delay.h>

#include<stdlib.h>

intdata0,data1,data2,data3,data4,data5,a,b,c,d;

unsignedcharx[10],y[10],z[10],g[10],h[10],f[10];

#defineDATA\_REGISTER\_EMPTY(1<<UDRE)

#defineRX\_COMPLETE(1<<RXC)

#defineFRAMING\_ERROR(1<<FE)

#definePARITY\_ERROR(1<<UPE)

#defineDATA\_OVERRUN(1<<DOR)

#pragmaused-

#define\_ALTERNATE\_PUTCHAR\_

#pragmaused+

voidputchar(charc)

{

 while((UCSRA&DATA\_REGISTER\_EMPTY)==0);

 UDR=c;

}

voidmain(void)

{

 // Input/Output Ports initialization

 // Port B initialization

 // Function: Bit7=In Bit6=In Bit5=Out Bit4=In Bit3=Out Bit2=Out Bit1=In Bit0=In

 DDRB=(0<<DDB7)|(0<<DDB6)|(1<<DDB5)|(0<<DDB4)|(1<<DDB3)|(1<<DDB2)|(0<<DDB1)|(0<<DDB0);

 // State: Bit7=T Bit6=T Bit5=0 Bit4=T Bit3=0 Bit2=0 Bit1=T Bit0=T

 PORTB=(0<<PORTB7)|(0<<PORTB6)|(0<<PORTB5)|(0<<PORTB4)|(0<<PORTB3)|(0<<PORTB2)|(0<<PORTB1)|(0<<PORTB0);

 // Port C initialization

 // Function: Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=In

 DDRC=(0<<DDC6)|(0<<DDC5)|(0<<DDC4)|(0<<DDC3)|(0<<DDC2)|(0<<DDC1)|(0<<DDC0);

 // State: Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T

 PORTC=(0<<PORTC6)|(0<<PORTC5)|(0<<PORTC4)|(0<<PORTC3)|(0<<PORTC2)|(0<<PORTC1)|(0<<PORTC0);

 // Port D initialization

 // Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=In

 DDRD=(0<<DDD7)|(0<<DDD6)|(0<<DDD5)|(0<<DDD4)|(0<<DDD3)|(0<<DDD2)|(0<<DDD1)|(0<<DDD0);

 // State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T

 PORTD=(0<<PORTD7)|(0<<PORTD6)|(0<<PORTD5)|(0<<PORTD4)|(0<<PORTD3)|(0<<PORTD2)|(0<<PORTD1)|(0<<PORTD0);

 // Timer/Counter 0 initialization

 // Clock source: System Clock

 // Clock value: Timer 0 Stopped

 TCCR0=(0<<CS02)|(0<<CS01)|(0<<CS00);

 TCNT0=0x00;

 // Timer/Counter 1 initialization

 // Clock source: System Clock

 // Clock value: Timer1 Stopped

 // Mode: Normal top=0xFFFF

 // OC1A output: Disconnected

 // OC1B output: Disconnected

 // Noise Canceler: Off

 // Input Capture on Falling Edge

 // Timer1 Overflow Interrupt: Off

 // Input Capture Interrupt: Off

 // Compare A Match Interrupt: Off

 // Compare B Match Interrupt: Off

 TCCR1A=(0<<COM1A1)|(0<<COM1A0)|(0<<COM1B1)|(0<<COM1B0)|(0<<WGM11)|(0<<WGM10);

 TCCR1B=(0<<ICNC1)|(0<<ICES1)|(0<<WGM13)|(0<<WGM12)|(0<<CS12)|(0<<CS11)|(0<<CS10);

 TCNT1H=0x00;

 TCNT1L=0x00;

 ICR1H=0x00;

 ICR1L=0x00;

 OCR1AH=0x00;

 OCR1AL=0x00;

 OCR1BH=0x00;

 OCR1BL=0x00;

 // Timer/Counter 2 initialization

 // Clock source: System Clock

 // Clock value: Timer2 Stopped

 // Mode: Normal top=0xFF

 // OC2 output: Disconnected

 ASSR=0<<AS2;

 TCCR2=(0<<PWM2)|(0<<COM21)|(0<<COM20)|(0<<CTC2)|(0<<CS22)|(0<<CS21)|(0<<CS20);

 TCNT2=0x00;

 OCR2=0x00;

 // Timer(s)/Counter(s) Interrupt(s) initialization

 TIMSK=(0<<OCIE2)|(0<<TOIE2)|(0<<TICIE1)|(0<<OCIE1A)|(0<<OCIE1B)|(0<<TOIE1)|(0<<TOIE0);

 // External Interrupt(s) initialization

 // INT0: Off

 // INT1: Off

 MCUCR=(0<<ISC11)|(0<<ISC10)|(0<<ISC01)|(0<<ISC00);

 // USART initialization

 // Communication Parameters: 8 Data, 1 Stop, No Parity

 // USART Receiver: On

 // USART Transmitter: On

 // USART Mode: Asynchronous

 // USART Baud Rate: 9600 (Double Speed Mode)

 UCSRA=(0<<RXC)|(0<<TXC)|(0<<UDRE)|(0<<FE)|(0<<DOR)|(0<<UPE)|(1<<U2X)|(0<<MPCM);

 UCSRB=(0<<RXCIE)|(0<<TXCIE)|(0<<UDRIE)|(1<<RXEN)|(1<<TXEN)|(0<<UCSZ2)|(0<<RXB8)|(0<<TXB8);

 UCSRC=(1<<URSEL)|(0<<UMSEL)|(0<<UPM1)|(0<<UPM0)|(0<<USBS)|(1<<UCSZ1)|(1<<UCSZ0)|(0<<UCPOL);

 UBRRH=0x00;

 UBRRL=0x0C;

 // Analog Comparator initialization

 // Analog Comparator: Off

 // The Analog Comparator's positive input is

 // connected to the AIN0 pin

 // The Analog Comparator's negative input is

 // connected to the AIN1 pin

 ACSR=(1<<ACD)|(0<<ACBG)|(0<<ACO)|(0<<ACI)|(0<<ACIE)|(0<<ACIC)|(0<<ACIS1)|(0<<ACIS0);

 SFIOR=(0<<ACME);

 // ADC initialization

 // ADC disabled

 ADCSRA=(0<<ADEN)|(0<<ADSC)|(0<<ADFR)|(0<<ADIF)|(0<<ADIE)|(0<<ADPS2)|(0<<ADPS1)|(0<<ADPS0);

 // SPI initialization

 // SPI Type: Master

 // SPI Clock Rate: 2\*15.625 kHz

 // SPI Clock Phase: Cycle Half

 // SPI Clock Polarity: High

 // SPI Data Order: LSB First

 SPCR=(0<<SPIE)|(1<<SPE)|(1<<DORD)|(1<<MSTR)|(1<<CPOL)|(1<<CPHA)|(1<<SPR1)|(0<<SPR0);

 SPSR=(1<<SPI2X);

 // TWI initialization

 // TWI disabled

 TWCR=(0<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|(0<<TWEN)|(0<<TWIE);

 // Alphanumeric LCD initialization

 // Connections are specified in the

 // Project|Configure|C Compiler|Libraries|Alphanumeric LCD menu:

 // RS - PORTB Bit 6

 // RD - PORTB Bit 7

 // EN - PORTD Bit 5

 // D4 - PORTD Bit 6

 // D5 - PORTD Bit 7

 // D6 - PORTB Bit 0

 // D7 - PORTB Bit 1

 // Characters/line: 16

 lcd\_init(16);

 while(1)

 {

   while(d!=0x73)

   {

     PORTB|=(1<<PORTB3)|(1<<PORTB5);//set

     PORTB&=~(1<<PORTB2);//clr

     //config mode

     spi(0x01);

     spi(0x43);

     spi(0x00);

     spi(0x01);

     spi(0x00);

     PORTB|=(1<<PORTB3);

     delay\_ms(1);

     PORTB|=(1<<PORTB2);

     delay\_ms(10);

     PORTB|=(1<<PORTB3)|(1<<PORTB5);

     PORTB&=~(1<<PORTB2);

     spi(0x01);

     spi(0x44);

     spi(0x00);                                 //analogmode

     spi(0x01);

     spi(0x03);

     spi(0x00);

     spi(0x00);

     spi(0x00);

     spi(0x00);

     PORTB|=(1<<PORTB3);

     delay\_ms(1);

     PORTB|=(1<<PORTB2);

     delay\_ms(10);

     PORTB|=(1<<PORTB3)|(1<<PORTB5);

     PORTB&=~(1<<PORTB2);

     spi(0x01);

     spi(0x43);

     spi(0x00);

     spi(0x00);

     spi(0x5A);

     spi(0x5A);                               //exitconfig

     spi(0x5A);

     spi(0x5A);

     spi(0x5A);

     PORTB|=(1<<PORTB3);

     delay\_ms(1);

     PORTB|=(1<<PORTB2);

     delay\_ms(10);

     PORTB|=(1<<PORTB3)|(1<<PORTB5);

     PORTB&=~(1<<PORTB2);

     spi(0x01);

     d=spi(0x42);

     spi(0x00);

     spi(0x00);

     spi(0x00);

     spi(0x00);                                   //ack

     spi(0x00);

     spi(0x00);

     spi(0x00);

     PORTB|=(1<<PORTB3);

     delay\_ms(1);

     PORTB|=(1<<PORTB2);

     delay\_ms(10);

   }

   while(d==0x73)

   {

     PORTB|=(1<<PORTB3)|(1<<PORTB5);

     PORTB&=~(1<<PORTB2);

     a=spi(0x01);

     b=spi(0x42);

     c=spi(0x00);

     data0=spi(0x00);//buttons set 1 8

     data1=spi(0x00);//button set 2  8

     data2=spi(0x00);//  rx     0 right 255

     data3=spi(0x00);//  ry     0 down  255

     data4=spi(0x00);//  lx

     data5=spi(0x00);//  ly

     delay\_us(10);

     PORTB|=(1<<PORTB3);

     delay\_us(10);

     PORTB|=(1<<PORTB2);

     /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

     //       //data0 for arrow keys set

     //       //data1 for square,circle,triangle,cross,l1,l2,r1,r2

     //       //data2 right stick(x)

     //       //data3 left stick(y)

     //       //data4 right stick(y)

     //       //data5 left stick(x)

     if(data0==0xDF)

     {

       putchar('R');     //right arrow

     }

     elseif(data0==0x7F)//left arrow

       putchar('L');

     elseif(data0==0xEF)//arrow up

       putchar('U');

     elseif(data0==0xBF)//arrow down

       putchar('D');

     elseif(data1==0xDF)

     {

       putchar('O');     //circle

     }

     elseif(data1==0x7F)//square

       putchar('Q');

     elseif(data1==0xEF)//arrow up

       putchar('T');

     elseif(data1==0xBF)//cross

       putchar('X');

     elseif(data1==0xF7)

     {

       putchar('r');     //R1

     }

     elseif(data1==0xFD)//R2

       putchar('p');

     elseif(data1==0xFB)

     {

       putchar('l');     //L1

     }

     elseif(data1==0xFE)//L2

       putchar('q');

     //sticks

     elseif(data3<81)

     {

       itoa(data3,x);      //right joystick up

       puts(x);

     }

     elseif(data3>174)//right joystick down

     {

       itoa(data3,x);

       puts(x);

     }

     elseif(data4<81)//left joystick left

     {

       data4=data4+625;

       itoa(data4,x);

       puts(x);

     }

     elseif(data4>174)

     {

       data4=data4+540;  //left joystick right

       itoa(data4,x);

       puts(x);

     }

     elseif(data2<81)

     {

       data2=data2+265;

       itoa(data2,x);

       puts(x);            //right joystick left

     }

     elseif(data2>174)//right joystick right

     {

       data2=data2+180;

       itoa(data2,x);

       puts(x);

     }

     elseif(data5<81)

     {

       data5=data5+445;

       itoa(data5,x);

       puts(x);            //left joystick up

     }

     elseif(data5>174)//left joystick down

     {

       data5=data5+360;

       itoa(data5,x);

       puts(x);

     }

   }

 }

}

-----------------------------------------------------------------------------

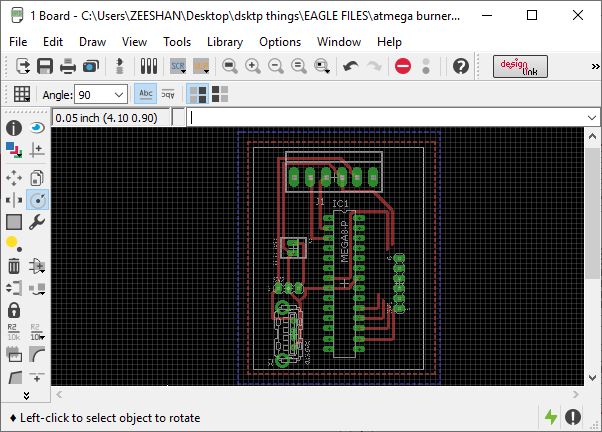
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

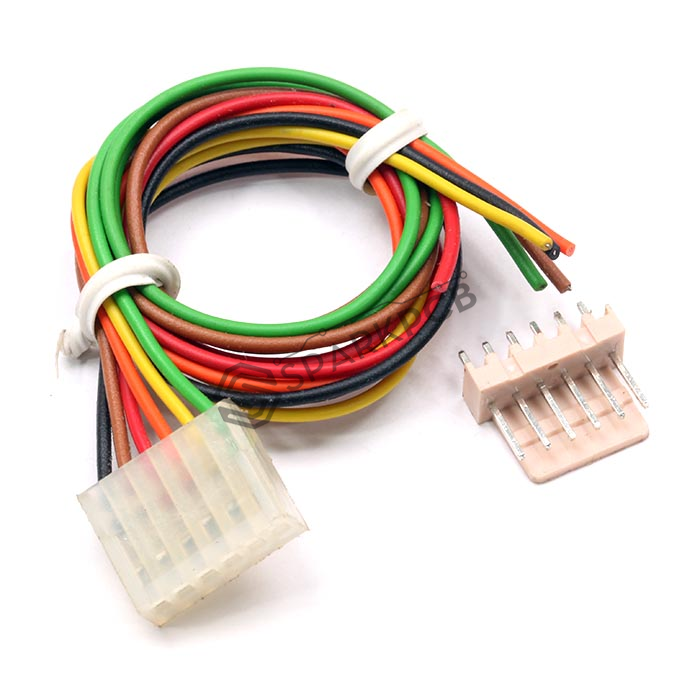
The task to be done when a particular button is pressed is handled by assigning the characters i.e. a different character for different button. The characters are transmitted through the UART using the function putchar('<ASSIGNED CHARACTER>')

The Analog values through the sticks or the gears are ranging from 0 to 255 now the issue arises. There are two gears giving up the same value and when the values are transmitted it takes the value coming from only on stick hence there cannot be a unique task using the values of two sticks. Hence to overcome the issue the values are mapped by adding a high value so that it is transmitted within a particular range and hence can be used for two different tasks. The data received at the receiver side was used by reading using Serial.read()and the data was stored in a variable to compare with the other conditions.

# 

# circuit and connections:

At the transmitter side the ps2 controller was made wireless by attaching a circuit designed in **CADSOFT EAGLE.**The circuit is designed so that it can be powered by various ways using a 12V battery and also by using a powerBank through a USB port attached onto the circuit. The circuit can also be powered by using a 5V Lipo battery so that the weight and size is minimized to its best . the transmitter fixed with a draft circuit is shown in the picture above. The cables of the PS2 were made short and soldered to multistand cables so that it can be easily pulled onto the **Relimate** base mounted onto the circuit.



****



The circuits were designed and manufactured. The components were hand soldered onto the PCB .

# issues faced:

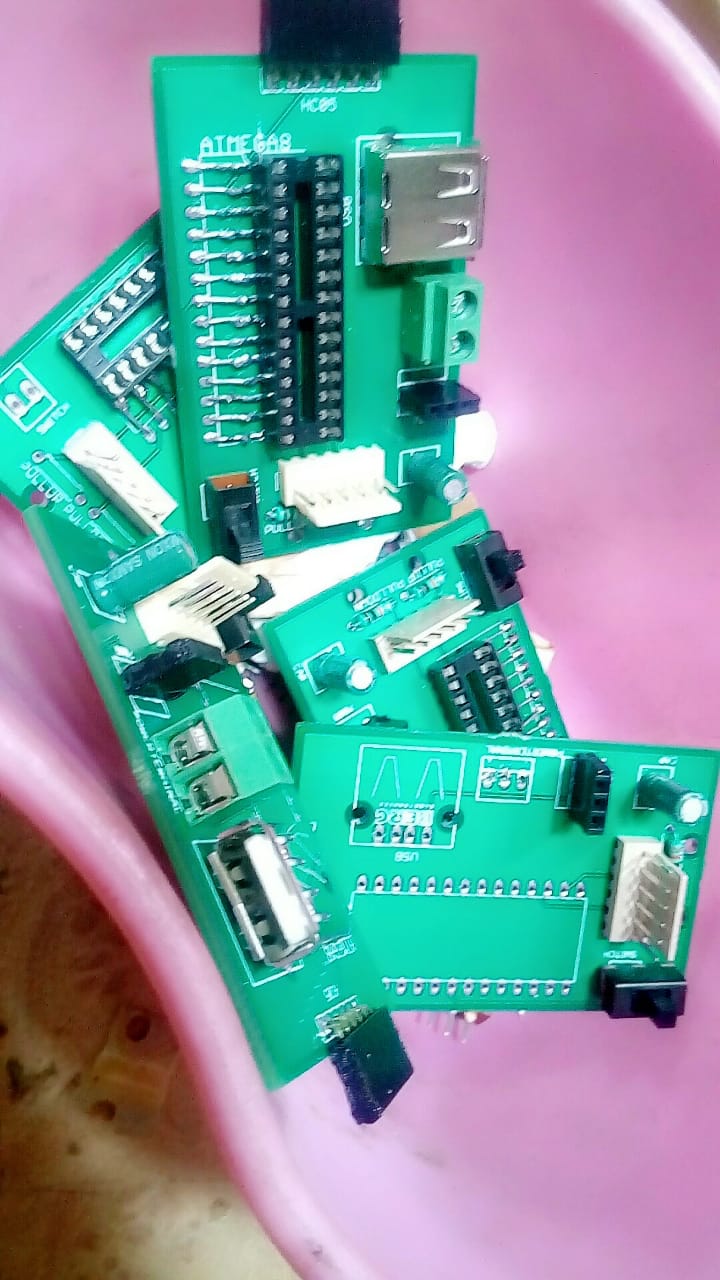
UART allows only string and char datatypes to be transmitted hence the analog values are to be converted by storing it in a variable whose datatype is unsignedcharx[10];

Now that the values are successfully transmitted and received successfully but the issue was the errors in the receiving side. The value received is not as desired, the value which was expected changes because of the very next value mixed with the present value

In other words, If 255 is transmitted then it is received as 2, 55, 2552 at a time which makes the value completely unusable in the code.

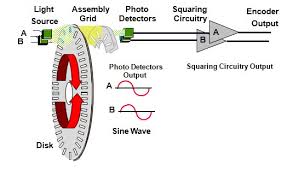
This problem was overcome by writing a simple code at the receiving side . it was observed that by sending a character ‘\0’ or ’ \n’ after a successful transmission can be used as a separating element between two successive values.

The controller developed can be further modified by using micorcontrollers of as minimum size as possible. Further the BLUETOOTH MODULE HC05 can be replaced by using BLUETOOTH LOW ENERGY (BLE) which will consume less power.



**Encoder**

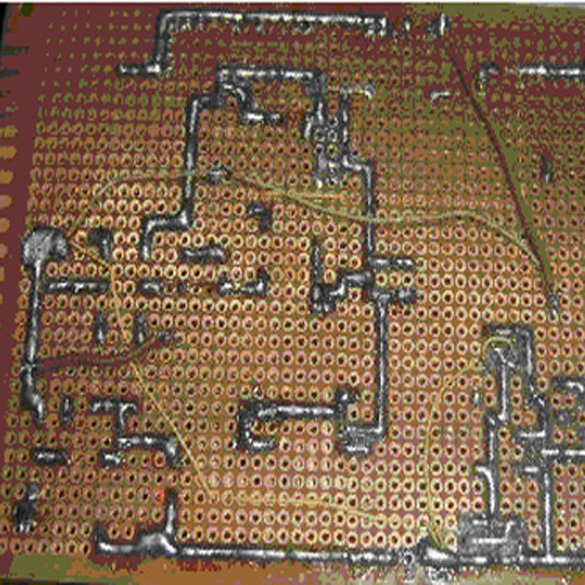
Encoder is a circit or a network that acvts like a feedback mechanism from motor. Encoders converts rotating motion of motor into an electrical signal which is taken as an input by a mictrocontroller as a analog signal and converts it into digital signal and from that signal readings are calculated via programming and for each rotation of motor an encoder gives fixed value of counts (say 1500) and based on this counts and wheel dia the distance of the robot can be calculated.

****

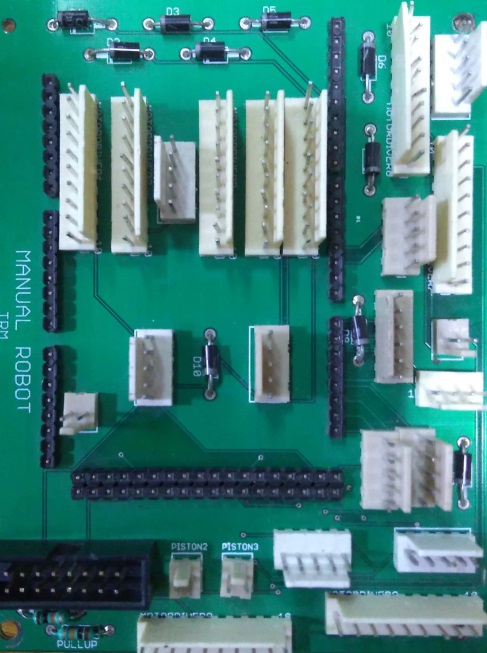
Encoder working principle

**Circuit**

The ciruits for the prototyping was designed and manufactured manually and rectifications were done on that circuits and the final circuit of the robot was designed on cadsoft eaglesoftware and printing of the circuit was donr from the vendor.



Testing circit of MR1



FINAL PRINTED CIRCUIT OF MR1

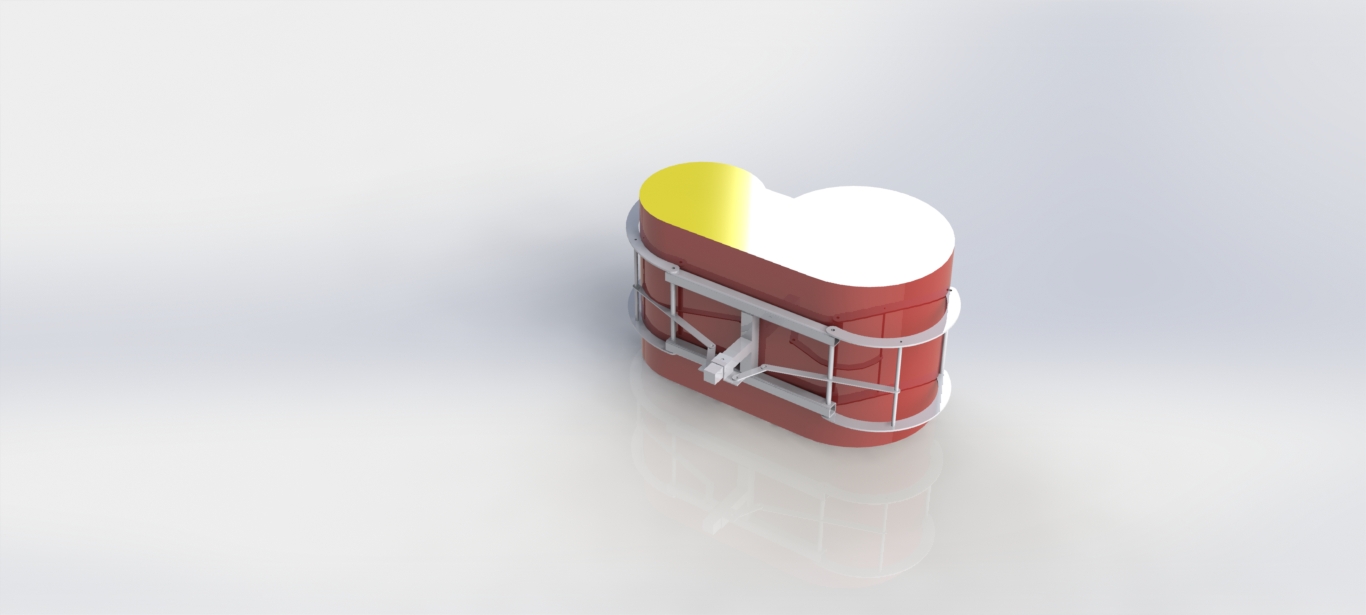
**THE FINAL MESSENGER ROBOT 1**

**Locomotion**

* To finish the task as soon as possible we need to pass the Gerege to MR 2 and then throw shagai.
* In order to obtain the following requirement we need an efficient chassis. So holonomic chassis with 4 Omni wheels is used.
* A holonomic chassis has motion with 3 degrees of freedom, unlike differential drives, which have only 2 degrees of freedom. As a result, a holonomic drive can shift from side to side or diagonally without changing the direction of its wheels.
* Omni wheels are multidirectional wheel ,they have three degrees of freedom ,itcan also rotate around the motorized wheel axis, around the rollers, and around the point of contact.
* Omni wheels are used to reduce the friction.
* Locomotion of MR1 was controlled by a ps2 controller.
* It was programmed to move in a plus or omnidirectional mode**.**
* Ps2 controller has two analog sticks which were used for moving in forward, backward, right and left directions.
* For forward and backward motion the right analog stick should be moved upwards or downwards respectively.
* For left and right motion the left analog stick should be moved left or right respectively.
* The analog stick gives values from 0 to 255 at the extremities with 127 as the center value.
* These values are mapped with the OCRi.e. as the stick is moved away from the center (rest position)the speed increases accordingly.
* The pad left button was used to rotate about point in counter clock wise direction and pad right was used to rotate about point in clock wise direction.

**Grabbing & Lifting**

* Grabbing the shagai of 540gm from the grloungd is not that easy task.
* seveeal mechanisms were tested for picking the shagai from the ground and placing on to the robot.
* First mechanism was with a horizontal plate that is motor actuated with a rack and pinion is used and it was not giving the statisfactory results, the disadvantage of that mechanism was the motor fails to pick that load.
* The drawback of the above mechanism is overcomed by using a worm gear PW motor which has stall torque of infinite theoretically,i.e it doesn’t move unless it is powered.and it have self locking mechanism.
* As shown in fig. the improvised mechanism has links to hold and pick up the shagai from the ground.
* Holding and releasing of the shagai is done using links that are actuated using piston.
* The grabber is made by using 20x20 aluminum blocks and polypropylene. The links are made of aluminum plate.

****

* The grabbing mechanism used piston to grab and release the shaghai.
* For lifting the shaghai, the motor was connected with encoderfor precise movement.
* The encoder counts were used to know the position of grabber and thus grab the shaghai and place it on the launching pad precisely.
* While lifting the shagai, to counteract the gravitational pull, more OCR was given to the motor.
* Similarly less OCR was given while coming down.
* Different buttons of the ps2 controller were assigned for grabbing and releasing the shaghai, moving the grabber up and down

**Loading shagai and shooting:**

* Shooting the shagai of 540g at a distance of around 2m requires high power and accuaracy.
* Mechanism using pistons: This mechanism uses a pneumatic piston and an aluminum frame casted in a semicircular shape. setup was an incline surface with a slot in the center. A piston is placed beneath the incline surface. And the frame was connected to the piston through the slot made at the center. When the piston opens, it gives a push to shagai and launching.The unfavorable part is that the threshold generated was not sufficient enough.
* Catapult mechanism:This mechanism is made by using aluminum bocks and a pneumatic piston. So, the shagai was placed on one end of the link and piston is connected o the other end. When the piston is closed, the shagai launches. The negative part was the projectile was uncertain and range is less, so this mechanism was failed.

A close up of a sign

Description automatically generated

* Mechanism using TheraBand:TheraBand is an elastic material that can stretch up to twice of its length when gradual load is applied. The arrangement is the TheraBand is fixed on the either side of the launching surface. The band is inserted in a sandwich of aluminum and acrylic sheet.The band is pulled and shagai is to be placed. The frame is released at the desired time. The problem was TheraBandwas breaking from the ends. So, we had to look for different options.
* Mechanism using springs: In the setup, a surface is placed on the top of chassis at an angle of 12 degree with the horizontal. On the either side of the incline surface are two springs. The other ends of the springs are connected to a frame that launches the shagai after the springs are loaded and shagai is placed on the inclined surface. To load the springs, we have used a heavy-duty clutch wire which will be rolled on a pulley. The pulley is on a PW motor. While pulling the spring, the main issue was to lock the springs until the bot reaches in the throwing area. So, for that we used a locking mechanism using a polypropylene which locks the frame connected to motor and springs when the springs are completely loaded. This is done by using a pneumatic piston fixed on the inclined surface. Opening of piston leads tolocking of the frame. So we chose this mechanism because it gave use the desired output.
* To shoot the shaghai there was a spring loaded mechanism.
* For pulling the spring to an exact distance a motor attached with an encoder was used.
* Afterloading the spring, it was locked using a piston controlled by ps2 controller.
* After locking the spring, shaghai is placed on the launching pad.
* After aligning MR1 in the correct direction the lock is released with the piston and due to spring force the shaghai is shot in desired angle and distance.



**Code**

#include<PS2X\_lib.h>

#definePS2\_DAT       53

#definePS2\_CMD       51

#definePS2\_SEL       49

#definePS2\_CLK       47

#definepressures   false

#definerumble      false

**PS2X**ps2x;

int  w=0;

interror=0;

bytetype=0;

bytevibrate=0;

intencoderPinR1=18;//Channel A

intencoderPinR2=19;//Channel B

intencoderPin1=20;//Channel A

intencoderPin2=21;//Channel B

volatileintlastEncoded=0;

volatilelongev=0,count=0;

voidsetup()

{

 // put your setup code here, to run once:

**Serial**.begin(57600);

 error=ps2x.config\_gamepad(PS2\_CLK,PS2\_CMD,PS2\_SEL,PS2\_DAT,pressures,rumble);

 type=ps2x.readType();

 digitalWrite(encoderPin1,HIGH);//turn pullup resistor on

 digitalWrite(encoderPin2,HIGH);//turn pullup resistor on

 digitalWrite(encoderPinR1,HIGH);//turn pullup resistor on

 digitalWrite(encoderPinR2,HIGH);//turn pullup resistor on

 attachInterrupt(digitalPinToInterrupt(18),pull,CHANGE);

 attachInterrupt(digitalPinToInterrupt(19),pull,CHANGE);

 attachInterrupt(digitalPinToInterrupt(20),updateEncoder,CHANGE);

 attachInterrupt(digitalPinToInterrupt(21),updateEncoder,CHANGE);

 pinMode(30,OUTPUT);

 pinMode(31,OUTPUT);

 pinMode(32,OUTPUT);

 pinMode(33,OUTPUT);

 pinMode(34,OUTPUT);

 pinMode(36,OUTPUT);

 pinMode(38,OUTPUT);

 pinMode(39,OUTPUT);

 pinMode(40,OUTPUT);

 pinMode(41,OUTPUT);

 pinMode(42,OUTPUT);

 pinMode(43,OUTPUT);

 pinMode(44,OUTPUT);

 pinMode(45,OUTPUT);

 pinMode(46,OUTPUT);

 pinMode(48,OUTPUT);

 pinMode(7,OUTPUT);

 pinMode(8,OUTPUT);

 pinMode(9,OUTPUT);

 pinMode(10,OUTPUT);

 pinMode(11,OUTPUT);

 pinMode(12,OUTPUT);

}

voidloop()

{

 if(error==0)//skip loop if no controller found

 {

   ps2x.read\_gamepad(false,vibrate);

   inty=ps2x.Analog(PSS\_RY);

   intx=ps2x.Analog(PSS\_LX);

   intr=map(x,127,255,0,255);

   intb=map(y,127,255,0,255);

   intl=map(x,0,126,255,0);

   intf=map(y,0,126,255,0);

   if(x>=180)//right

   {

**Serial**.println(r);

**Serial**.println("right");

     mot1(0,1,r);

     mot2(0,1,r);

     mot3(1,0,r);

     mot4(1,0,r);

   }

   elseif(x<=100)//left

   {

**Serial**.println(l);

**Serial**.println("left");

     mot1(1,0,l);

     mot2(1,0,l);

     mot3(0,1,l);

     mot4(0,1,l);

   }

   elseif(y<=100)//forward

   {

**Serial**.println(f);

**Serial**.println("forward");

     mot1(0,1,f);

     mot2(1,0,f);

     mot3(1,0,f);

     mot4(0,1,f);

   }

   elseif(y>180)//backward

   {

**Serial**.println(b);

**Serial**.println("back");

     mot1(1,0,b);

     mot2(0,1,b);

     mot3(0,1,b);

     mot4(1,0,b);

   }

   else  if(ps2x.Button(PSB\_PAD\_RIGHT))

   {

     mot1(0,1,125);

     mot2(0,1,125);

     mot3(0,1,125);

     mot4(0,1,125);

   }

   elseif(ps2x.Button(PSB\_PAD\_LEFT))

   {

     mot1(1,0,125);

     mot2(1,0,125);

     mot3(1,0,125);

     mot4(1,0,125);

   }

   elseif(ps2x.Button(PSB\_PAD\_UP))//lift

     mot\_e(0,1,210);

   else  if(ps2x.Button(PSB\_PAD\_DOWN))//return

     mot\_e(1,0,160);

   else  if(ps2x.Button(PSB\_L1)&&w==1)//lift

   {

     mot\_p(1,0);

     while(ev<=0&&ev>-810)

     {

       while(ev<=0&&ev>-680)

       {

         mot\_e(0,1,210);

       }

       while(ev<-670&&ev>-810)

       {

         mot\_e(0,1,40);

       }

       w=0;

     }

   }

   else  if(ps2x.Button(PSB\_R1)&&w==0)//return

   {

     while(ev<840&&ev>=0)

     {

       while(ev>=0&&ev<200)

       {

         mot\_e(1,0,90);

       }

       while(ev>200&&ev<850)

       {

         mot\_e(1,0,40);

       }

       w=1;

     }

   }

   else  if(ps2x.Button(PSB\_CROSS)

 {

   while(count>=-13100)

     {

       mot\_r(0,1,255);//wind

     }

   }

   else  if(ps2x.Button(PSB\_TRIANGLE))

 {

   while(count<=13200)

     {

       mot\_r(1,0,255);//unwwind

     }

   }

   else  if(ps2x.Button(PSB\_CIRCLE))

   mot\_r(0,1,255);//wind

   else  if(ps2x.Button(PSB\_SQUARE))

     mot\_r(1,0,255);//unwind

     else  if(ps2x.Button(PSB\_R2))

       mot\_t(1,0);

       else  if(ps2x.Button(PSB\_L2))

         mot\_t(0,1);

         else//stop

         {

           count=0;

           ev=0;

           mot1(1,1,255);

           mot2(1,1,255);

           mot3(1,1,255);

           mot4(1,1,255);

           mot\_e(0,0,255);

           mot\_r(0,0,255);

           mot\_p(0,0);

           mot\_t(0,1);

          }

 }

}

voidmot1(intp,intq,intr)

{

 digitalWrite(48,p);

 digitalWrite(46,q);

 analogWrite(12,r);

}

voidmot2(intp,intq,intr)

{

 digitalWrite(44,p);

 digitalWrite(42,q);

 analogWrite(11,r);

}

voidmot3(intp,intq,intr)

{

 digitalWrite(40,p);

 digitalWrite(38,q);

 analogWrite(10,r);

}

voidmot4(intp,intq,intr)

{

 digitalWrite(36,p);

 digitalWrite(34,q);

 analogWrite(9,r);

}

voidmot\_e(intp,intq,intr)

{

 digitalWrite(32,p);

 digitalWrite(30,q);

 analogWrite(8,r);

}

voidmot\_p(intp,intq)

{

 digitalWrite(45,p);

 digitalWrite(43,q);

}

voidmot\_t(intp,intq)

{

 digitalWrite(41,p);

 digitalWrite(39,q);

}

voidmot\_r(intp,intq,intr)

{

 digitalWrite(31,p);

 digitalWrite(33,q);

 analogWrite(7,r);

}

voidupdateEncoder()

{

 intMSB=digitalRead(encoderPin1);//MSB = most significant bit

 intLSB=digitalRead(encoderPin2);//LSB = least significant bit

 intencoded=(MSB<<1)|LSB;//converting the 2 pin value to single number

 intsum  =(lastEncoded<<2)|encoded;//adding it to the previous  value

 if(sum==0b1101||sum==0b0100||sum==0b0010||sum==0b1011)ev++;

 if(sum==0b1110||sum==0b0111||sum==0b0001||sum==0b1000)ev--;

 lastEncoded=encoded;//store this value for next time

}

voidpull()

{

 intMSB=digitalRead(encoderPinR1);//MSB = most significant bit

 intLSB=digitalRead(encoderPinR2);//LSB = least significant bit

 intencoded=(MSB<<1)|LSB;//converting the 2 pin value to single number

 intsum  =(lastEncoded<<2)|encoded;//adding it to the previous value

 if(sum==0b1101||sum==0b0100||sum==0b0010||sum==0b1011)count++;

 if(sum==0b1110||sum==0b0111||sum==0b0001||sum==0b1000)count--;

 lastEncoded=encoded;//store this value for next time

}

MESSENGER ROBOT-2

# Problem Statements for MR2

# the robot must be an autonomous robot. It must be a Quadruped robot,i.e.its locomotion cannot be on wheels but can only move on four legs, like any 4-legged mammal. It Must bring the gerege to uukai zone after receivingit in gobiurtuu from MR1. It must carry the gerege below its maximum body height and raise only at the uukai zone (mountain).



# Bot Features

Based on above statements we decided that the bot will have the following functions:

* **Gait pattern**: Shall mimic gait pattern of either a horse or dog.
* **Autonomous track completion and Obstacle bypass**: The path will be hard-fed in the program via delays.
* **GeregeTranport:**Should be able to grab, carry and translate the gerege as per requirement.
* **Instantaneous Turning:** The robot should be able to take sharp turns without having to change its orientation.

# Gait Pattern:

A Gait pattern is the consecutive movement of limbs of any being during locomotion over solid surface.

## Types

There are 4 primary Gait patterns for Dog classified as followed:

* Walk:

The walk is a four-beat gait. When walking, the legs follow this sequence: left hind leg, left front leg, right hind leg, right front leg, in a regular 1-2-3-4 beat. At the walk, the dog will alternate between having three or two feet on the ground.

* Trot:

The trot is a two-beat gait that has a wide variation in possible speeds. In this gait, the horse moves its legs in unison in diagonal pairs. From the standpoint of the balance, this is a very stable gait.

* Canter:

The canter is a controlled, three-beat gait that is usually a bit faster than the average trot, but slower than the gallop. Dogs use the cross canter gait, also called rotary or diagonal canter, in which they run with one front leg ahead of the other and the opposite hind leg ahead. This gait allows to turn sharply.

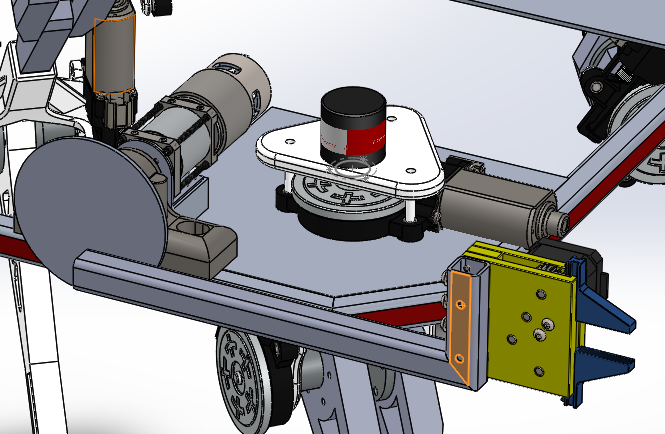
* Gallop:

The gallop is very much like the canter, except that it is faster, more ground-covering, and the three-beat canter changes to a four-beat gait. It is the fastest gait. When dogs gallop, they usually use different lead legs in front and rear, unlike horses, which usually gallop with the same leads in front and rear.

Among the four trot is the most efficient since it covers more area at minimum energy cost.

# Gerege Transport:

This feature alone require 2 necessary functions that are to be executed in the complete carry task.

* Receiving and Holding the gerege:

The constraint in this task is the rule which implies that there must not be any contact or communication between the two bots other than the gerege transfer and that the MR2 must carry the gerege below its max height. Hence we devised a stepper powered gear based gripper with rubber grips as shown in the figure. The gerege will be passed in horizontal manner.

* Rising the gerege:

The gripper assembly will be kept initially in horizontal orientation with the help of mechanical restrictions and will later be raised to vertical orientation by operating the planetary geared encoder motor.

# Bot structure

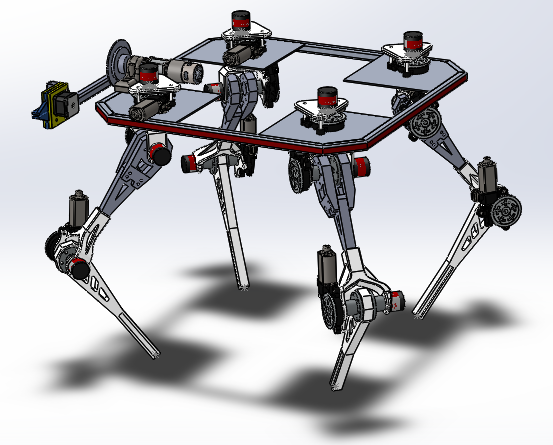
Considering the number of components to be used in the bot, it was necessary to use a light weight material. Ofcourse Aluminum can be used but using plastics will further decrease the bot weight. Hence, we decided to use the material called Polypropylene (PP) which has good machining properties.

The Body

Considering the size of the robot, it was decided that the body will have to be made out of aluminum in order to counter the large bending moments.

The Legs

The model of legs was initially simple, with motor-encoder couple at each joint. Worm-geared motor was used considering the worm-gear mechanism’s self-locking feature. Each leg consists of 3 articulated joints, meaning each leg has 3 DOF. The topmost joint at the body will enable the robot to turn all legs instantaneously without having to lift any leg, thus facilitating in sharp turns.



# Electronics Description

The Speciality of the 4 legged Robot that was designed is that it has 3 degrees of freedom in each leg which has allowed us to program the robot to follow any gait pattern that is suitable for a particular condition. This is achieved by having 3 PW motors on each leg. To control the position optical incremental external encoders (with 1500 PPR (pulse per rotation))are used. The first encoder is placed at the hip joint, the second one at the thigh and the last at the knee which may also be referred as ankle. For example as shown in below figure:

---- --->Hip

\* --->Thigh

\

\

\

* -->ankle

/

/

/

The hip joint can rotate the leg 3600 about a point, clockwise or anticlockwise. Thigh motor can rotate 1800 about its point of rotation, thus translating to back or front parallel to ground. The ankle motor can rotate approximately 2700 max, back and front parallel to ground. The design is very simple, making the robot flexible.

Moving towards electronics part each leg has 3 motors, in order to operate each motor individually a separate motor driver is to be used. “Hercules” motor driver of current rating 13A suits best as per the specs of the motor. 3 Arduino due are used as they have ARM cortex processor which supports all pins as interrupt and PWM.

processors in them making them very fast when compared to other Arduinos. Furthermore, all of their digital pins can be used as external interrupts. 2 Arduinos were used for controlling the legs, each controlled 2 diagonal legs. The middle/Central Microcontroller (Arduino due) was used to control and synchronize the functions of the other 2 Arduino Dues. Communication was established between these Arduinos with the help I2C and Interrupts.

Two Arduinos (This was then extended to be operational for the tri-Arduino setup) have been used, “Wire.h” library and “PID\_v1.h” library to do most of the work.

**Circuit Design for 4 Legged Robot / Quadrobot**

****

**Eagle:**

EAGLE is an electronic design automation software application with schematic capture, printed circuit board layout, auto router and computer aided manufacturing features.

EAGLE stands for Easily Applicable Graphical Layout Editor.

This software is compatible with Windows , Linux, Mac OS X, providing an easy availability and flexibility of options for the User. It is also available in different languages like German , Chinese ,English etc.

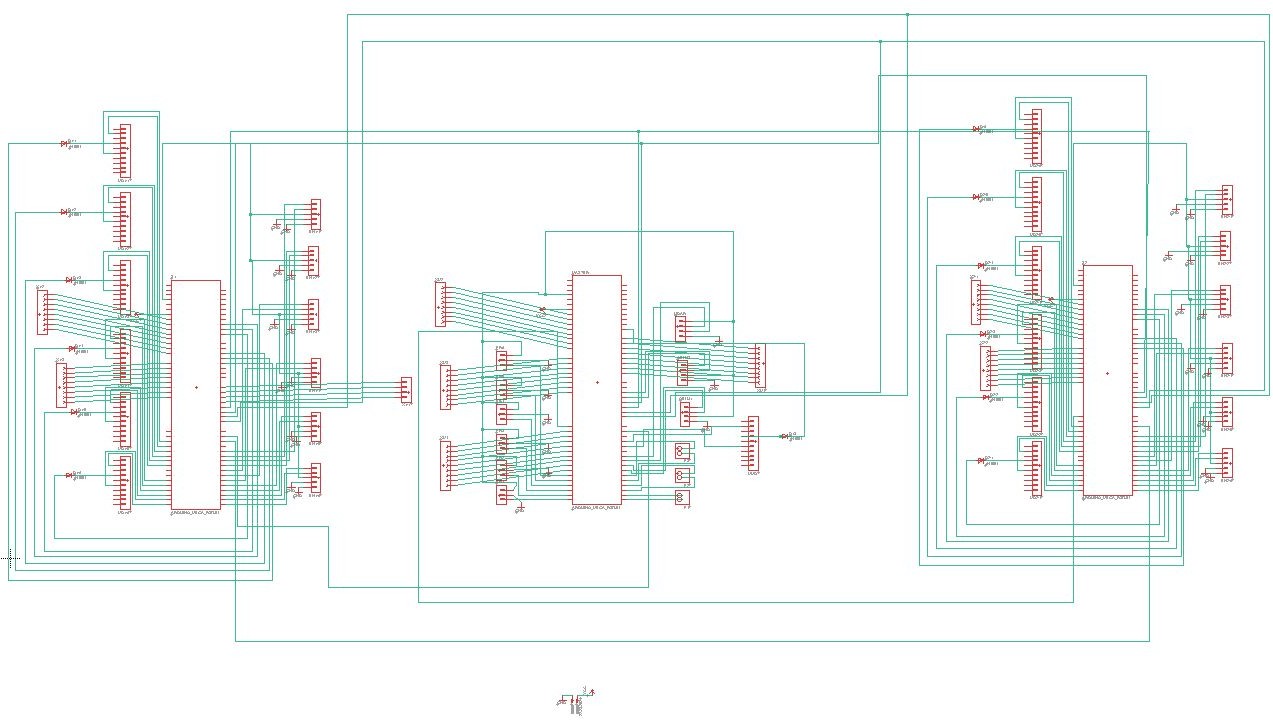
EAGLE is a user friendly software in which there is an opportunity for the user to design and fabricate the PCB. It provides a Schematic Mode, PCB Layout Mode, CAM Processor Mode, etc in order to make the task of PCB designing easy. EAGLE is available in different versions depending on the user like Teacher, Student, Professional, etc.

EAGLE 7.6.0 is a free version for Students, Teachers and also Beginners, EAGLE 7.6.0 was used to design and to fabricate the PCB.

**IMPLEMENTATION AND DESCRIPTION OF EAGLE:**

### SCHEMATIC MODE:

This model helps the user to connect the components that are required to be pasted on PCB.

This software contains predefined library of components and also provides a facility to download from the internet

The Schematic Mode is specially designed in such a way that user can easily access the schematic.

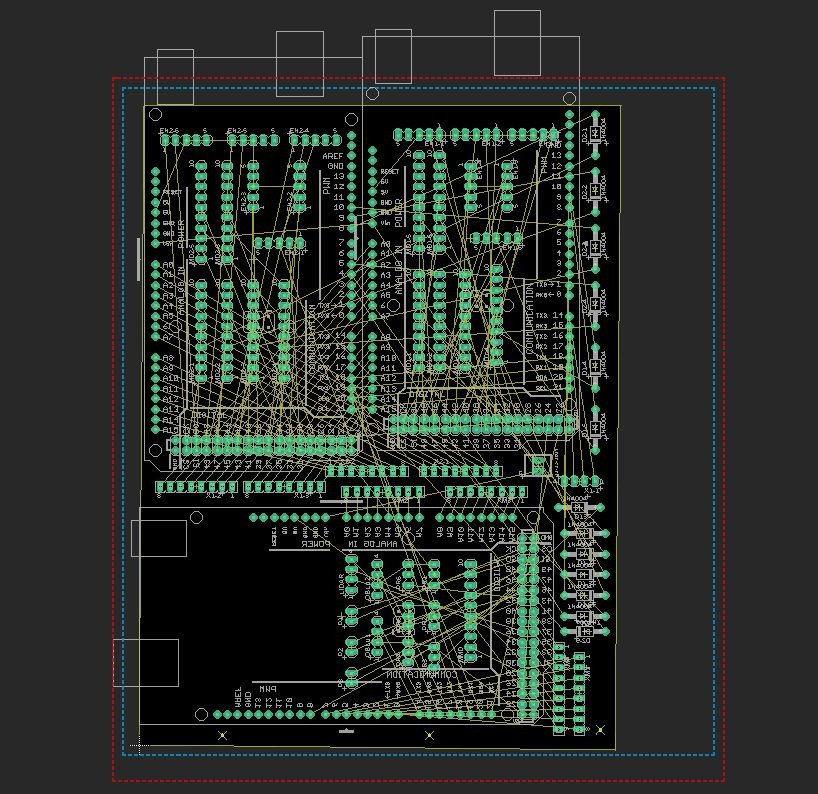
We easily mastered the simple operations by making sample circuits, We developed the skills of designing the circuits easily. Then we started to design our basic circuits on the Schematic Mode,The above image shows one of our Circuit’s Schematic Capture.

EAGLE has very vast range of Libraries of Components for ex TEXAS INSTRUMENTS, ATMEL, and many other Companies etc which helps the user in finding a Component in its wide range of Component’s. While we were designing the circuits there was a problem of some Components, Even after having a large number of Components in EAGLE, ATMEGA 16 was not available in the Components list, It turns out that we need to download the ATMEGA 16 library and include it in the main set of libraries in the EAGLE 7.6.0 Folder.

### PCB LAYOUT :

This layout provides the user with a basic layer that resembles a PCB. The user can arrange the components on PCB so that he can correct the overlapping junctions and also adjust the area.

We have an additional feature as auto-route which routes the connections giving preference to vcc and ground giving a non-overlapping PCB design.



After completely designing the circuit in the Schematic Mode, Now we need to make PCB Layout out of the schematic circuit, opt the schematic to PCB option on the Control

Panel, we placed each and every component on the PCB in such a way that no air-wire collide, This is to make the PCB in only one layer i.e Top or Bottom Layer.

If the circuit is very big then we can also do it in two Layers ie both Top and bottom Layers. After the Components are well placed we need Route the tracks which can be done either by manual routing or auto routing. As manual routing takes time we can auto-route it in just a few clicks but before routing but before routing we need to set the Trace Width of the Tracks, diameter of Drill holes, Pads, setting the Trace Width is an important factor in the Electronic circuits, setting wrong trace width can lead to the in-efficient circuit which in-turn results in failure of the circuit.

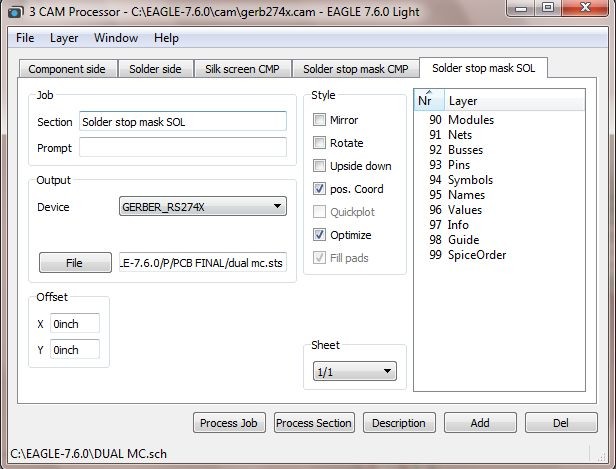
After auto routing the circuit there will be Traces visible in place of that air-wires, Now we did the Ratsnest the circuit to make the circuit complete. But this was not the end we successfully made the circuit, Now we need to fabricate the PCB so we need to make the files which is used by the PCB manufacturing Company to fabricate our PCB. That special file are called as “Gerber files”.

Creating of Gerber files is shown in next side heading CAMPROCESSOR.

#### CAM PROCESSOR:

This converts the PCB layout files to the required Extensions like Gerber and DRL etc which are used by different manufacturing companies to provide information to the CNC machines.

A general extension provided is Gerber that converts the pcb layout into a 2D plane with exact dimensions i.e. space and leisure between two different paths etc.



Gerber Files can be made by going into the CAM PROCESSOR, It can be done either by downloading the ‘Gerber Job’ directly or by setting the parameters manually. We used the manual method by setting all the required values to the Gerber, We didn’t opted the Gerber Job method because in the Gerber job it contains the standard values which may not be suitable for our circuits, so we did manage to set each parameter value manually to make the circuits work efficiently, There are many parameters which plays an important role in making a good PCB such as Trace Width, Cu thickness, Space between two Traces, Pads, Drill holes diameter, etc. After that, We provided the Gerber files to a PCB Manufacturing Company to fabricate the PCB.

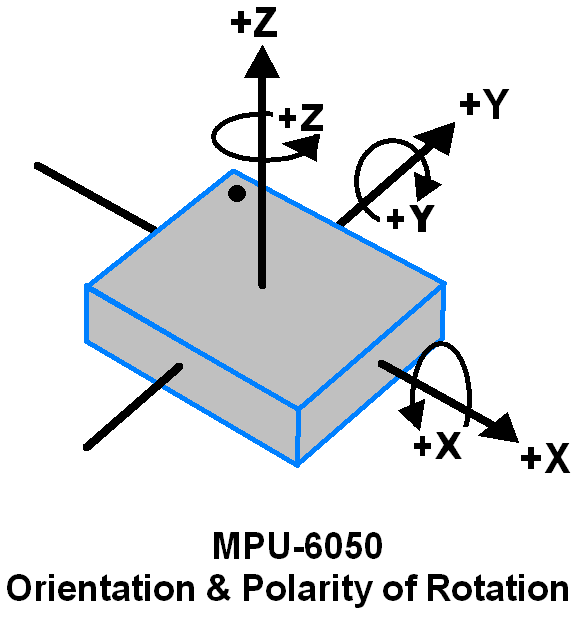
**MPU6050 (Gyroscope + Accelerometer + Temperature) Sensor Module**

MPU6050 sensor module is complete 6-axis Motion Tracking Device. It combines 3-axis Gyroscope, 3-axis Accelerometer and Digital Motion Processor all in small package. Also, it has additional feature of on-chip Temperature sensor. It has I2C bus interface to communicate with the microcontrollers.

It has Auxiliary I2C bus to communicate with other sensor devices like 3-axis Magnetometer, Pressure sensor etc.

3-Axis Gyroscope

The MPU6050 consist of 3-axis Gyroscope with Micro Electro Mechanical System(MEMS) technology. It is used to detect rotational velocity along the X, Y, Z axes as shown in below figure.



- When the gyros are rotated about any of the sense axes, the Coriolis Effect causes a vibration that is detected by a MEM inside MPU6050.

- The resulting signal is amplified, demodulated, and filtered to produce a voltage that is proportional to the angular rate.

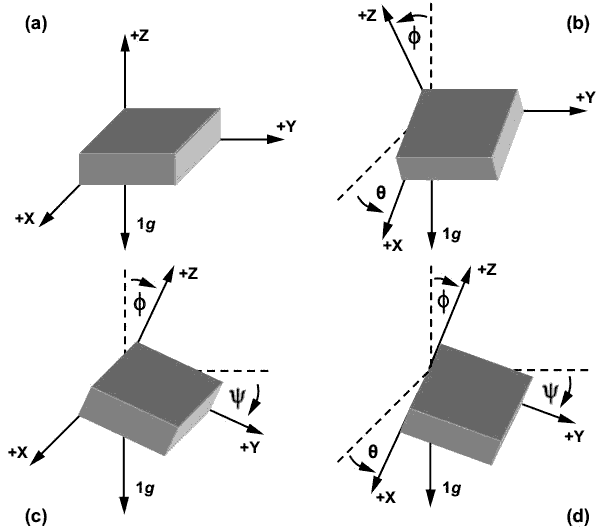
- This voltage is digitized using 16-bit ADC to sample each axis.

- The full-scale range of output are +/- 250, +/- 500, +/- 1000, +/- 2000.

- It measures the angular velocity along each axis in degree per second unit.

3-Axis Accelerometer

The MPU6050 consist 3-axis Accelerometer with Micro Electro Mechanical (MEMs) technology. It used to detect angle of tilt or inclination along the X, Y and Z axes as shown in below figure.



- Acceleration along the axes deflects the movable mass.

- This displacement of moving plate (mass) unbalances the differential capacitor which results in sensor output. Output amplitude is proportional to acceleration.

- 16-bit ADC is used to get digitized output.

- The full-scale range of acceleration are +/- 2g, +/- 4g, +/- 8g, +/- 16g.

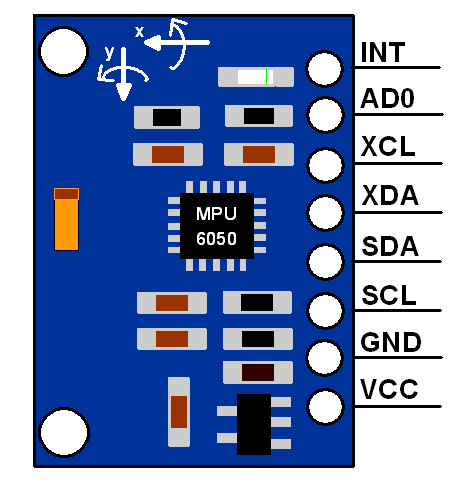
- It measured in g (gravity force) unit.

- When device placed on flat surface it will measure 0g on X and Y axis and +1g on Z axis.

DMP (Digital Motion Processor)

The embedded Digital Motion Processor (DMP) is used to compute motion processing algorithms. It takes data from gyroscope, accelerometer and additional 3rd party sensor such as magnetometer and processes the data. It provides motion data like roll, pitch, yaw angles, landscape and portrait sense etc. It minimizes the processes of host in computing motion data. The resulting data can be read from DMP registers.

MPU-6050 Module



The MPU-6050 module has 8 pins,

**INT:** Interrupt digital output pin.

**AD0:** I2C Slave Address LSB pin. This is 0th bit in 7-bit slave address of device. If connected to VCC then it is read as logic one and slave address changes.

**XCL:** Auxiliary Serial Clock pin. This pin is used to connect other I2C interface enabled sensors SCL pin to MPU-6050.

**XDA:** Auxiliary Serial Data pin. This pin is used to connect other I2C interface enabled sensors SDA pin to MPU-6050.

**SCL:** Serial Clock pin. Connect this pin to microcontrollers SCL pin.

**SDA:** Serial Data pin. Connect this pin to microcontrollers SDA pin.

**GND:** Ground pin. Connect this pin to ground connection.

**VCC:** Power supply pin. Connect this pin to +5V DC supply.

**Calculations acceleration and angular velocity**

Gyroscope and accelerometer sensor data of MPU6050 module consists of 16-bit raw data in 2’s complement form.

- Accelerometer full scale range of +/- 2g with Sensitivity Scale Factor of 16,384 LSB(Count)/g.

- Gyroscope full scale range of +/- 250 °/s with Sensitivity Scale Factor of 131 LSB (Count)/°/s.

To get sensor raw data, we need to first perform 2’s complement on sensor data of Accelerometer and gyroscope.

After getting sensor raw data we can calculate acceleration and angular velocity by dividing sensor raw data with their sensitivity scale factor as follows,

**Accelerometer values in g (g force)**

Acceleration along the X axis = (Accelerometer X axis raw data/16384) g.

Acceleration along the Y axis = (Accelerometer Y axis raw data/16384) g.

Acceleration along the Z axis = (Accelerometer Z axis raw data/16384) g.

**Gyroscope values in °/s (degree per second)**

Angular velocity along the X axis = (Gyroscope X axis raw data/131) °/s.

Angular velocity along the Y axis = (Gyroscope Y axis raw data/131) °/s.

Angular velocity along the Z axis = (Gyroscope Z axis raw data/131) °/s.

Getting displacement and heading using mpu6050 is very complicated and not

accurate because it has only 1 IMU. Oblu is chosen because of onboard floating-point

processing capability, along with four IMUs, making sensor fusion and motion processing

possible within the module itself and in turn resulting in very accurate motion sensing.

**OBLU**

Oblu is Multi-IMU based inertial navigation module. It has on-board four 6 axis

IMUs for motion sensing. The sensors’ data is calibration compensated and fused (averaged) to

get single ax, ay,az, gx, gy and gz. Oblu has an onboard Bluetooth module (BLE

4.1)preprogrammed for data transfer.

Oblu comes pre-programmed as a Pedestrian Dead Reckoning or PDR

sensor. Pedestrian dead reckoning is the process of calculating one's current position by using a

previously determined position and advancing that position based upon speed over elapsed

time and course. Oblu has ZUPT(zero velocity update) aided inertial navigation , which

generates displacement and heading change data between two successive ZUPT instances

i.e .when velocity becomes zero. This way oblu detects 'steps' and corrects some internal errors.

And this frequent correction of errors, results in great tracking performance.

Note: Oblu can be only if the bot moves in regular zero and non zero velocity moments.

The PDR system consists of "oblu" as a PDR sensor (IMU) and a PDR

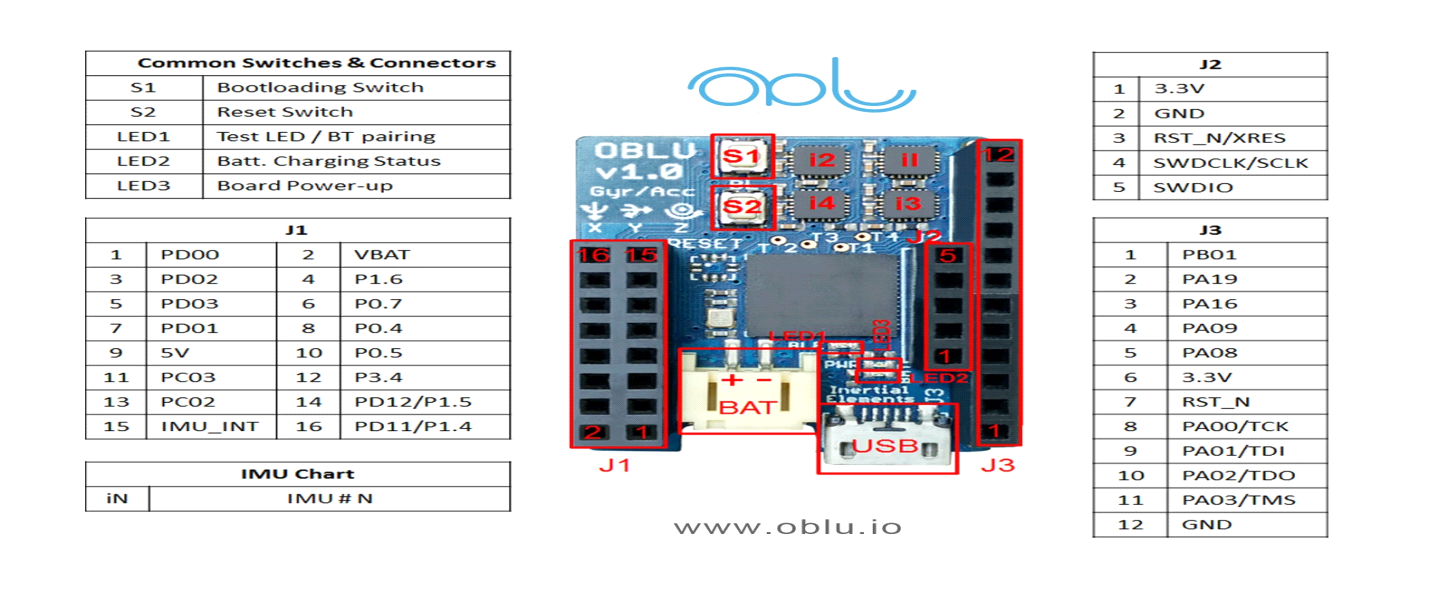
application, Xoblu Android App. Data from on-board multiple IMUs are sampled, combined and

undergoes computation to produce displacement and orientation information at every step. The

oblu transmits displacement and orientation data at every step to Xoblu, in response to an

appropriate command. Xoblu then, using this PDR data from the oblu, constructs and displays

the tracked path.



**Command Flow Sequence**

**1. Xoblu sends START command to oblu**

For example, to get stepwise dead reckoning data the command [52 0 52] has to be sent

**2. oblu sends 4 bytes acknowledgment in response to the START command from Xoblu.**

Example

"A0 34 00 D4"

A0: Acknowledgement state

34 = Start command state

00 D4 = Checksum

**3.Oblu sends data packet containing displacement and orientation information at every**

**step, to Xoblu.**

Total data packet size is 64 bytes.

· There are 4 header bytes - one for state, two for data packet number and one for

Payload size.

· Number of bytes in Payload is 56-Payload has 14 elements of float type (14\*4=56)First

4 elements are the most important ones which consist of the delta vector i.e. the change

in position x, y, z and the angle of rotation around the z-axis. The remaining 10 elements are

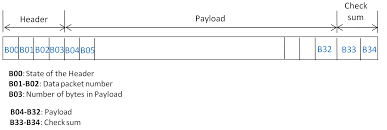
not used in the step-wise dead reckoning. These are used in data fusion from two oblu

devices in case of a dual foot mounted system.

· Next 2 bytes consisted of step counter, which is a counter taking record of ZUPT

instances observed.

· The last 2 bytes are for checksum.



**4. Xoblu acknowledges receiving the last data packet by sending an appropriate**

**acknowledgment to oblu.**

ACK consists of 5 bytes:

· 1st byte: 01

· 2nd byte: P1 (First byte of data packet number of last data packet received)

· 3rd byte: P2 (Second byte of data packet number of last data packet received)

· 4th byte: Quotient of {(1+P1+P2) div 256}

· 5th byte: Remainder of {(1+P1+P2) div 256}

**5: Xoblu sends stop command**

· Stop Processing in Oblu by sending 3 bytes command: {0x32, 0x00, 0x32}

· Stop all outputs in Oblu by sending 3 bytes command : {0x22, 0x00, 0x22}

Note: These commands can be sent only by Xoblu.To start and stop using Arduino the BLE has

to be disabled. It can be done

**CODE**

#include <SoftwareSerial.h>

#include <math.h>

#define packet\_size 68

#define track\_error\_percent 5

#define theta\_error\_percent 7

unsigned int stepsw = 0;

double sin\_phi, cos\_phi;

double delta [4] = {0.0, 0.0, 0.0, 0.0};

double final\_data [5];

double distance, distance1, track\_length, heading = 0.0;

double dx [5];

double x\_sw [5]={0,0,0,0,0};

int bytedata, bytedata\_usb;

byte ack [5] = {0, 1, 2, 3, 4};

int line[packet\_size];

uint8\_t header[4]={0,0,0,0};

int newbyte = -1;

int x [4]={0,0,0,0};

int y [4]={0,0,0,0};

int z [4]={0,0,0,0};

int h [4]={0,0,0,0};

int steps[2]={0,0};

bool flag\_f, flag\_b = false;

unsigned int j, k, l, package\_number, package\_number\_old, datacount = 0;

int nturn=0;

float theta;

double track;

double data\_array[]= {0.5,70,0.5,70,0.5,70,0.5,70,0,0};

int data\_array1[]= {};

int r, count=0;

typedef unsigned char uchar;

const int w\_cmd[3] = {0x34, 0x00, 0x34};

//int serialdata(byte data);

#define pwm1 2

#define pwm2 3

#define dr1 42

#define dr2 43

#define an1 4

#define an2 5

/\*-----------------------------

\* Function Definations

-------------------------------\*/

void serialdata(int data) ;//1

byte createAck(byte ack [], int packet\_number\_1, int packet\_number\_2);//2

float bytesToFloat(uchar b0, uchar b1, uchar b2, uchar b3);//3

void stepwise\_dr\_tu();//4

void move\_forward(int mdelay);//5

void move\_backward(int mdelay);//6

void turn\_left(int mdelay);//7

void turn\_right(int mdelay);//8

void stop\_m();//9

void motorcontrols();//10

/\*-----------------------------

\* SETUP

------------------------------\*/

void setup()

{

// put your setup code here, to run once:

Serial.begin(115200);

Serial3.begin(115200);

pinMode(pwm1,OUTPUT);

pinMode(pwm2,OUTPUT);

pinMode(dr1,OUTPUT);

pinMode(dr2,OUTPUT);

for (int i = 0; i < packet\_size; i++)

{

line[i] = '\0';

}

}

/\*------------------------------

\* LOOP

--------------------------------- \*/

void loop()

{

// put your main code here, to run repeatedly:

// when characters arrive over the serial port...

if (Serial3.available())

{

// wait a bit for the entire message to arrive

delay(100);

// read all the available characters

while (Serial3.available() > 0)

{

bytedata = Serial3.read();

if (newbyte == -1)

{

if (bytedata == 0xA0)

{

datacount = 0;

}

if (bytedata == 0xD4)

{

datacount++;

newbyte = 0;

break;

}

}

else

{

if (datacount)

{

serialdata(bytedata);

}

}

}

}

}

void serialdata(int data)

{

line[j] = data;

j++;

if (j == 63) //line array will not fill, this condition can not be executed

{

//////////////////////////////////header assign///////////////

for (int i = 0; i < 4; i++) //the first 4 bytees are of header

{

header[i] = line[i];

}

//////////////////////////////////// payload //////////////////////////

for (int i = 0; i < 4; i++) /////////////next 4 bytes are of dx////////////

{

x[i] = line[i + 4];

dx[0] = bytesToFloat(x[0], x[1], x[2], x[3]);

}

for (k = 0; k < 4; k++) //////////////next 4 byte are of dy////////

{

y[k] = line[k + 8];

dx[1] = bytesToFloat(y[0], y[1], y[2], y[3]);

}

for (k = 0; k < 4; k++) /////////////next 4 byte are of dz////////////////

{

z[k] = line[k + 12];

dx[2] = bytesToFloat(z[0], z[1], z[2], z[3]);

}

for (k = 0; k < 4; k++) /////////////next 4 byte are of da(heading)////////////////

{

h[k] = line[k + 16];

dx[3] = bytesToFloat(h[0], h[1], h[2], h[3]);

}

for (k = 0; k < 1; k++) /////////////step count 2 byte////////////////

{

steps[k] = line[k + 61];

}

int package\_number1 = header[1];

int package\_number2 = header[2];

//////////////////////////////// Create ack for next step///////////////////////////////

ack[0] = createAck(ack, package\_number1, package\_number2); // Acknowledgement created

ack[1] = createAck(ack, package\_number1, package\_number2); // Acknowledgement created

ack[2] = createAck(ack, package\_number1, package\_number2); // Acknowledgement created

ack[3] = createAck(ack, package\_number1, package\_number2); // Acknowledgement created

ack[4] = createAck(ack, package\_number1, package\_number2); // Acknowledgement created

package\_number = package\_number1 \* 256 + package\_number2; //PACKAGE NUMBER ASSIGNED

if (package\_number\_old != package\_number) {

stepsw++;

stepwise\_dr\_tu();

package\_number\_old = package\_number;

}

motorcontrols();

j = 0;

}

}

byte createAck(byte ack [], int packet\_number\_1, int packet\_number\_2)

{

ack[0] = 0x01; // 1st byte

ack[1] = (byte)packet\_number\_1; // 2nd byte

ack[2] = (byte)packet\_number\_2; // 3rd byte

ack[3] = (byte)((1 + packet\_number\_1 + packet\_number\_2 - (1 + packet\_number\_1 + packet\_number\_2) % 256) / 256); // 4th byte – Quotient of {(1+P1+P2) div 256}

ack[4] = (byte)((1 + packet\_number\_1 + packet\_number\_2) % 256); // 5th byte- Remainder of {(1+P1+P2)div 256

return ack[4];

}

float bytesToFloat(uchar b0, uchar b1, uchar b2, uchar b3){

union

{

float f;

uchar b[4];

} u;

u.b[3] = b0;

u.b[2] = b1;

u.b[1] = b2;

u.b[0] = b3;

return u.f;

}

void stepwise\_dr\_tu()

{

sin\_phi = (float) sin(x\_sw[3]);

cos\_phi = (float) cos(x\_sw[3]);

delta[0] = cos\_phi \* dx[0] - sin\_phi \* dx[1];

delta[1] = sin\_phi \* dx[0] + cos\_phi \* dx[1];

delta[2] = dx[2];

delta[3] += dx[3] \* 57.3;

x\_sw[0] += delta[0];

x\_sw[1] += delta[1];

x\_sw[2] += delta[2];

x\_sw[3] += dx[3];

final\_data[0] = x\_sw[0];

final\_data[1] = x\_sw[1];

final\_data[2] = x\_sw[2];

final\_data[3] = delta[3];

distance1 = sqrt((delta[0] \* delta[0] + delta[1] \* delta[1] + delta[2] \* delta[2]));//sqrt(x^2+y^2+z^2)

heading += dx[3] \* 57.3;

if(flag\_f)

{

track\_length += distance1;

distance += distance1;

//Serial.print(distance);

}

if(flag\_b)

{

track\_length -= distance1;

distance -= distance1;

}

}

void move\_forward(int mdelay){

digitalWrite(pwm1, HIGH);

digitalWrite(dr1, LOW);

digitalWrite(pwm2, HIGH);

digitalWrite(dr2, LOW);

analogWrite(an1,140);

analogWrite(an2,125);

delay(mdelay);

Serial.println("forward");

stop\_m();

flag\_f = true;

flag\_b = false;

}

void move\_backward(int mdelay){

digitalWrite(pwm1, LOW);

digitalWrite(dr1, HIGH);

digitalWrite(pwm2, LOW);

digitalWrite(dr2, HIGH);

analogWrite(an1,140);

analogWrite(an2,125);

delay(mdelay);

Serial.println("backward");

stop\_m();

flag\_b = true;

flag\_f = false;

}

void turn\_left(int mdelay){

digitalWrite(pwm1, HIGH);

digitalWrite(dr1, LOW);

digitalWrite(pwm2, LOW);

digitalWrite(dr2, HIGH);

analogWrite(an1,140);

analogWrite(an2,125);

Serial.println(heading);

delay(mdelay);

Serial.println("left");

stop\_m();

flag\_f = false;

flag\_b = false;

}

void turn\_right(int mdelay){

digitalWrite(pwm1, LOW);

digitalWrite(dr1, HIGH);

digitalWrite(pwm2, HIGH);

digitalWrite(dr2, LOW);

analogWrite(an1,140);

analogWrite(an2,125);

Serial.println(heading);

delay(mdelay);

Serial.println("right");

stop\_m();

flag\_f = false;

flag\_b = false;

}

void stop\_m(){

digitalWrite(pwm1, LOW);

digitalWrite(dr1, LOW);

digitalWrite(pwm2, LOW);

digitalWrite(dr2, LOW);

analogWrite(an1,0);

analogWrite(an2,0);

Serial.println(heading);

Serial.println("stop");

//delay(mdelay);

}

void motorcontrols() {

bool turn = false;

float dis\_error = 0;

float theta\_error = 0;

int forw\_delay, back\_delay = 0;

int left\_delay, right\_delay = 0;

track = data\_array[nturn];

theta = (float) data\_array[nturn+1];

if(heading>=360)

{heading=heading-360;}

else if(heading<=-360)

{heading =heading+360;}

double delta\_track = fabs(track - track\_length);

float delta\_theta = fabs(theta - heading);

//if (nturn==0){

dis\_error = 0.015;

theta\_error = 3;

/\* }

else {

dis\_error = (track \* track\_error\_percent)/200;

theta\_error = (theta \* theta\_error\_percent)/200;

}

\*/

if (delta\_track < 0.2)

{

forw\_delay = 300;

}

else if (delta\_track >= 0.2 && delta\_track < 0.5){

forw\_delay = 500;

}

else if (delta\_track >= 0.5 && delta\_track < 1){

forw\_delay = 600;//700

}

else if (delta\_track >= 1){

forw\_delay = 700;//800

}

back\_delay = forw\_delay;

if (delta\_theta < 15){

left\_delay = 80;

}

else if (delta\_theta >= 15 && delta\_theta < 30){

left\_delay = 140;//140//

}

else if (delta\_theta >= 30 && delta\_track < 60){

left\_delay = 150;//160/

}

else if (delta\_theta > 60)

{

left\_delay = 170;//180/

}

right\_delay = left\_delay;

if ((track\_length >= (track - dis\_error)) && (track\_length <= (track + dis\_error)))

turn = true;

else

turn = false;

if (!turn)//if turn=false,i.e,(track\_length <= (track - dis\_error)) && (track\_length >= (track + dis\_error)

{

if (heading >= 3)//-theta+theta\_error

{

turn\_right(50);

}

else if (heading < -3) //theta-theta\_error 1.5

{

turn\_left(50);//40

}

else if ((track\_length < (track - dis\_error)) )

{

move\_forward(forw\_delay);

}

else if ((track\_length >= (track + dis\_error)))

{

move\_backward(150);//100

}

}

else

{

if(theta == 0)

{

stop\_m();

j = 0;

nturn = 0;

datacount = 0;

newbyte = -1;

delay(5000);

}

else if (heading < (theta - theta\_error))

{

turn\_left(right\_delay);

}

else if (heading > (theta + theta\_error))

{

turn\_right(left\_delay);

}

else

{

track\_length = 0;

heading = 0;

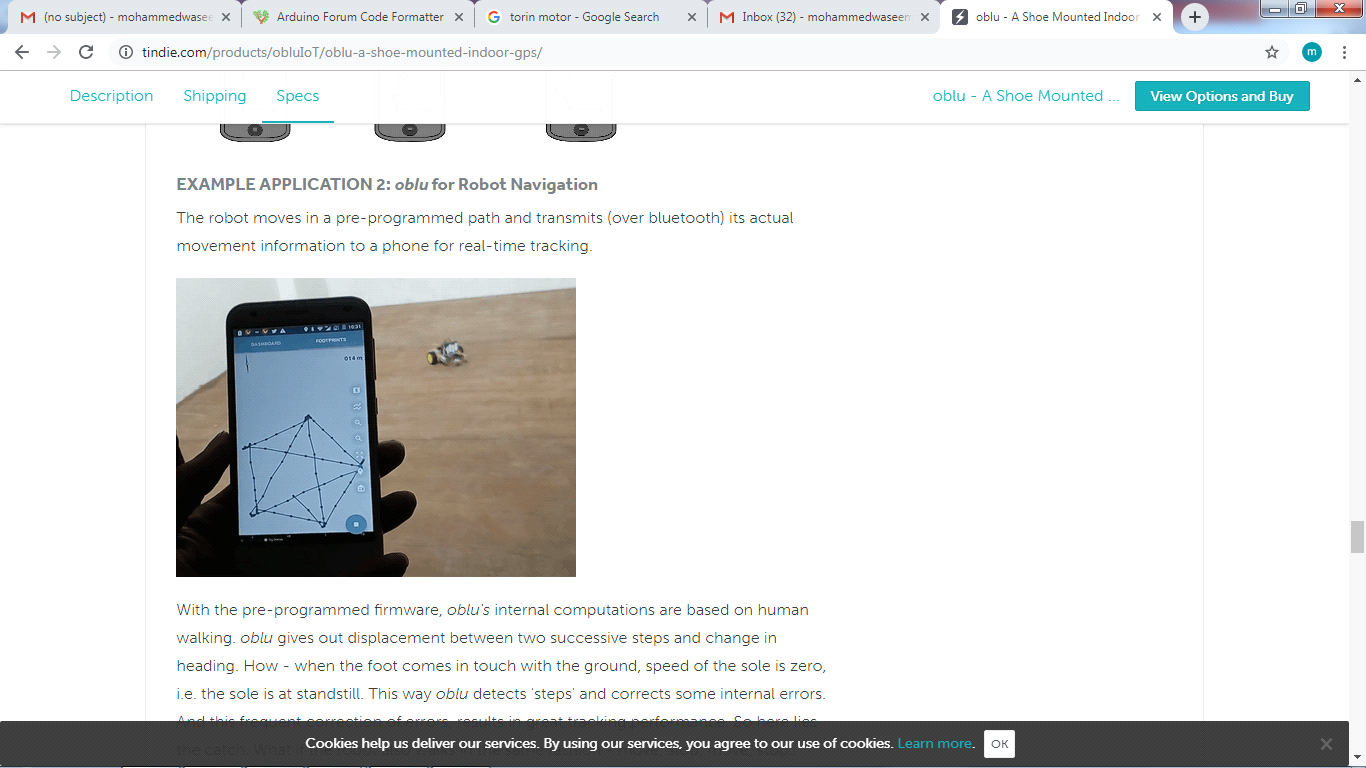
nturn=nturn+2;

}

}

}

OUTPUT:



## Programming 4 Legged Robot / Quadrobot

**Walking Mechanism:**

The aim was to create very flexible code that could be used as a horse, dog, or a spider, etc.

Spider:



In this, There are two types of gait pattern:

* 1. One of them is the gait start by raising the thigh, then rotate the hip motor forward, drops the thigh to touch the ground and finally rotating the hip motor backward.
  2. The other is the one in which the ankle motor is involved to pull the body forward.

Dog:



The speciality of dog gait is that, it is (to be implemented and) faster than a spider.

But horse gait pattern is faster than dog. The only difference between horse and dog is that, front legs are inverted inside in case of dog but for horse the front legs are inverted outside.

The reason for choosing dog over horse is that, there are many tutorials/ open sources are available on the Internet for it.

When it comes to dog, rotating it or turning it is hard. But the bot had a degree of freedom on the hip joint which would allow it’s rotation about the center of the body of chassis.

For this gait pattern, the hip motors are kept constant such that when thighs are rotated there should be parallel to each other and perpendicular to the chassis.

Based on the lengths of the legs, the calculated angle between the thigh and ankle should be 600 and the thigh and chassis be 600 in order for the bot to stand stably on the ground like a dog.

Steps to be followed for programming are:

1. For 1 leg movement, first ankle must be raised from the ground, then thigh must move upwards (in forward direction), then ankle should drop to touch the ground, and finally thigh should push the ground backwards for the body to move forward.
2. For 2 legs:

Adjacent Legs on Arduino1:

* 1. Moving front leg
  2. Communicate with other Arduino
  3. Move that Arduino’s back leg
  4. Move that Arduino’s front leg
  5. Then again communicate
  6. Move this Arduino’s back leg And this loop will continue.

Diagonal Legs on Arduino1:

This option is better than previous, as:

1. Moving front Leg
2. Moving Back Leg
3. Communicate
4. Moving that Arduino’s front leg
5. Moving that Arduino’s back leg
6. Communicateaand this loop will continue.

#### Synchronizing Legs on Arduino1:

Synchronizing here means moving two legs at a time on one Arduino, that is, first rising ankle of front leg and back leg, then moving thigh of front leg and back leg forward, then dropping the ankle of front leg and back leg, and then moving thigh of front leg and back leg backward.

About circuit: 1 pin of each Arduino is connected to other pin of the other Arduino in such a way that it forms a triangle in the form of interrupt pins:

Left MC Right MC

\ /

\ /

Master MC

In a similar way, I2C pins of Arduino, that is, SDA and SCL are connected.

Since I haven’t gone for the programming of master, I thought of communicating between left MC and right MC.

--------Interrupt pin-------

Left MC----------SDA Right MC

-----------SCL----------

#### Interrupt Communication:

One way communication using Interrupt and flag was easy. But in Arduino Due,two way communications involving one pin was creating an ambiguity, as the pin’s mode in master and slave had to be INPUT at one instance and OUTPUT at another instant.

Then, 2 pins were used. Its function was the same as the other but when right gets interrupt it gives an interrupt signal to left MC and resets its flag. When left MC is interrupted it sets the flag in ISR, enters the condition in void loop and sends an interrupt signal o right MC then reset its flag. This process continues in a loop. It was working well on Arduino Mega. But when shifted to Arduino Due, Serial monitor was crashing, mostly because of the speed at which Arduino Due works. So, this approach failed.

#### Digital pins:

This indicates slave to start its function. When slave is done it makes the other pin high. This will indicate master that slave is done and master should start its work.This was not efficient approach but working.

#### Two wire (I2C):

I2C communication was simple in one way, that is, master to slave but when it comes to other way, whenever master is requesting data from slave, it is acting as interrupt in case of slave. So, pulling SCL down until slave is ready to send data would work

But on Arduino Due, same as the previous problem was faced.

#### Three Wire (I2C+Interrupt):

1. Initially flag in Master is set, Master will execute forward and slave will be adjusting.
2. After finishing master’s work it will use I2C for sending a character indicating that it has completed its forward motion, resets its flag and start adjusting.
3. On receiving information from master, slave will stop adjusting legs, and start executing forward motion.
4. When slave finish its forward function, it makes the interrupt pin high and low. This sets the flag I master, indicating slave has finished executing forward function and master should stop adjusting and start its forward.

## Moving 4 Legs:

The work that was left is interfacing above steps, so the following logic was used:

* 1. Master forward -> front Leg then back leg complete -> I2C
  2. Slave forward -> front Leg then back leg complete -> Interrupt

### Two leg at a time:

Moving two legs is also same like above logic except front and back leg is synchronize.

For observing in air the above code will work. When kept on ground it won’t work but the one leg at a time can be used for forming a walking logic and two leg at a time can be used for forming running logic

### Walking:

The logic would be:

1. Raise front ankle.
2. Take step, that is, move the back leg’s thigh backwards, and front leg’s thigh forward.
3. Then drop the front ankle.
4. Similarly for back leg also should be done.
5. While this is happening the other two legs at other Arduino will be moving backward, that is, adjusting.
6. And then communicate with other Arduino and follow the same procedure as above will make a walking algorithm.

### Reset and Set:

Where should counts start, that is, what should be the reset position?

A reset position must always be a mechanical restriction, that is counts must increment or decrement from that position.

Choosing reset position for ankle and thigh was easy. Mechanical restriction in case of thigh was chassis and for ankle it was the joint/the connecting part of ankle and thigh. But for hip, there was no mechanical restriction.

Reset position of dog is:

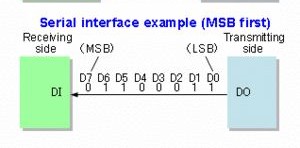
1. Ankle must rise until it touches the joint.
2. Thigh must rotate backward until it touches the chassis.

In this the front legs will be inside the chassis and the back legs will be outside the chassis and on ground it would be like sitting. These rotations must be independent of counts and PID. The reason for counts independent is that, no knows practically what will happen to dog robot when it is walking for example, it might fall, legs will rotate in other direction while walking/testing.

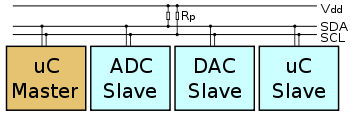
Counts for set position must be chosen in such a way that the angle between thigh and ankle is 600 and thigh and chassis is also 600.

The program had to rotate motors simultaneously (front ankle and thigh & back ankle and thigh).

## Communication in 4 Legged Robot / Quadrobot

**Serial Communication:**

[In telecommunication](https://en.wikipedia.org/wiki/Telecommunication) and [data transmission](https://en.wikipedia.org/wiki/Data_transmission), **serial communication** is the process of sending [data](https://en.wikipedia.org/wiki/Data) one [bit](https://en.wikipedia.org/wiki/Bit) at a time, sequentially, over a [communication channel](https://en.wikipedia.org/wiki/Communication_channel) or [computer](https://en.wikipedia.org/wiki/Computer_bus) [bus](https://en.wikipedia.org/wiki/Computer_bus). This is in contrast to [parallel communication](https://en.wikipedia.org/wiki/Parallel_communication), where several bits are sent as a whole, on a link with several parallel channels.

**I2C:**

I2C is a serial protocol for two-wire interface to connect low-speed devices like microcontrollers, EEPROMs, A/D and D/A converters, I/O interfaces and other similar peripherals in embedded systems. It was invented by Philips and now it is used by almost all major IC manufacturers. Each I2C slave device needs an address – they must still be obtained from [NX](http://www.nxp.com/campaigns/i2c-bus/)P (formerly Philips semiconductors).

**Electronic Components in 4 Legged Robot / Quadrobot Arduino Due:**



The **Arduino Due** is a microcontroller board based on the [Atmel SAM3X8E ARM](http://www.atmel.com/Images/Atmel-11057-32-bit-Cortex-M3-Microcontroller-SAM3X-SAM3A_Datasheet.pdf) [Cortex-M3 CPU](http://www.atmel.com/Images/Atmel-11057-32-bit-Cortex-M3-Microcontroller-SAM3X-SAM3A_Datasheet.pdf). It is the first Arduino board based on a 32-bit ARM core microcontroller. It has 54 digital input/output pins (of which 12 can be used as PWM outputs), 12 analog inputs, 4 UARTs (hardware serial ports), an 84 MHz clock, a USB OTG capable connection, 2 DAC (digital to analog), 2 TWI, a power jack, an SPI header, a JTAG header, a reset button and an erase button.

**Warning: Unlike most Arduino boards, the Arduino Due board runs at 3.3V. The maximum voltage that the I/O pins can tolerate is 3.3V. Applying voltages higher than 3.3V to any I/O pin could damage the board.**

But, the main reason for choosing Due was that, all the pins are interrupt pins which was essential for Encoders.

**External Encoders:**

A rotary encoder is a type of position sensor which is used for determining the angular position of a rotating shaft. It generates an electrical signal, either analog or digital, according to the rotational movement.

Orange 600 PPR Incremental Optical Rotary Encoder is hi-resolution optical encoder with quadrature outputs for increment counting. It will give 2400 transitions per rotation between outputs A and B. A quadrature decoder is required to convert the pulses to an up count. The Encoder is build to Industrial grade.

12 encoders are required for the positioning of 12 motor.

## Hercules Motor Driver:

: 

Hercules 6V-36V, 16Amp Motor Driver can take up to 30A peak current load and can be operated up to 3 KHz PWM. Motor driver can be interfaced with 3.3V and 5V logic levels. Motor driver has built-in protection from under / over voltage, over temperature and short. The Motor driver has terminal block as power connector and 7 pin 2510 type.

relimate connector for the logic connection. It is suitable for high performance robots, Robocon, Robo-cup, US First, Battle robots etc.

**MATLAB AND SIMULATION**

TASK:

The task was to get the angle calculations and gait pattern calculations of the four-legged bot.

OVERVIEW:

The bot was designed in solid works and then imported in the simulation workspace by converting it into .STEP file.The parts were again assembled in the Simulink using Simscape multi bodies.

Each and every part of the bot was a separate Simscape multi body.

A natural environment was created in the Simulink space by changing the friction and damping condition of the floor, where the bot was supposed to walk after the production.

Steps followed:

Firstly, we decided to make the whole bot in Simulink itself which is very difficult and hectic task and we were restricted to limited shapes so we decided to find some way that would be easy so we ended that we can import solid works model in MATLAB by installing a plugin in MATLAB and enable that plugin in solid works

A basic bot with two legs was first used for the simulation. After getting the satisfactory outputs,upgrade and stimulation was done on four-legged bot with less parts and simple design for easy simulation.

This was followed by making the bot follow a specific predefined path and reach the goal.

After achieving this, simulation was done on the final design of robot which included sensors, motors, encoders, motor drivers and required circuitry.

Different libraries and functions were used while simulating viz., sphere to plane, Delay function, smooth trajectory,6 DOF etc.

Problems faced:

1. Parts getting dismantled after running for simulation.
2. The legs were passing through the surface, due to incorrect friction values and damping conditions.

Problem Solved:

The problem of parts getting dismantled was solved by using revolute joint instead of cylindrical joint between different parts.

The placement of bot on ground was improvedby changing the damping conditions and friction values.

