Gesture Controlled BOT

NEELMANI GAUTAM
11640590

***Abstract*—This manual shows how to build a gesture controlled bot. It is controlled with hand gestures. Its controller is a glove over which a breadboard is mounted and a flex sensor is attached to forefinger of the glove. To move the bot forward, backward, right or left user need to tilt the breadboard forward, backward, right or left. To increase the speed of bot user need to bend his/her forefinger.**

1. INTRODUCTION

Bot is programed using ATMega328p. The main program is written in 'C' while some functions to control peripherals of microcontroller (eg. ADC, Timer) are written in 'AVR-ASM' and called in 'C' program file. This project can be divided into two parts:

- **Controller Side:** An accelerometer is used to detect hand gesture and thus control the movement of bot. It communicate with bot wirelessly using bluetooth module. Both are connected with ATMega328p and mounted over a hand glove. A flex sensor is also used to control the speed of bot. It is connected with microcontroller and attached to the forefinger of glove. It maps the amount of deflection or bending to some analog value.
- **Bot Side:** Here a bluetooth module is connected to ATMega328p on the bot side and paired with the bluetooth module on controller side. Its receives the data from controller and transfer it to ATMege328p that process it and move bot accordingly.

2. COMPONENTS REQUIRED

A. *Bot Side*

- Arduino UNO × 1
- HC-05 × 1 (Bluetooth Module)
- Chassis × 1
- Wheels × 2
- DC Motor × 2
- L293D × 1 (Motor Driver IC)
- Breadboard × 1

B. *Controller Side*

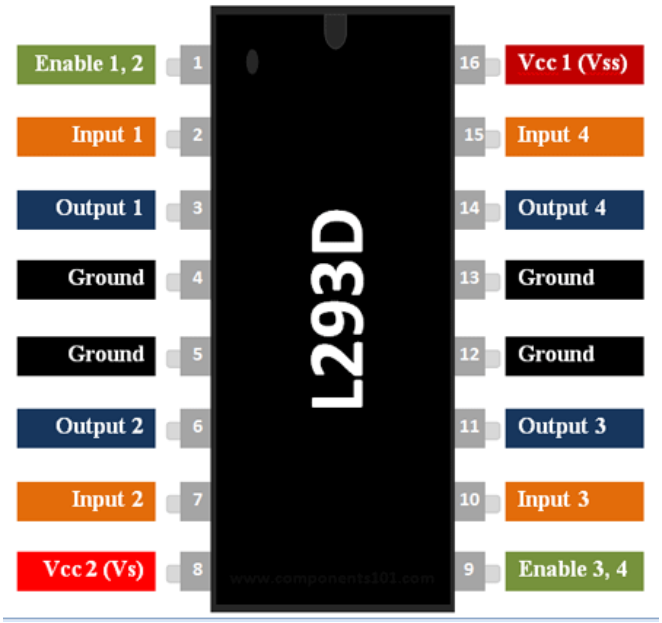
- Arduino UNO × 1
- HC-05 × 1 (Bluetooth Module)
- ADXL335 × 1 (Accelerometer Sensor)
- Flex Sensor 2.2" × 1
- Breadboard × 1

C. *Tools and Other Things*

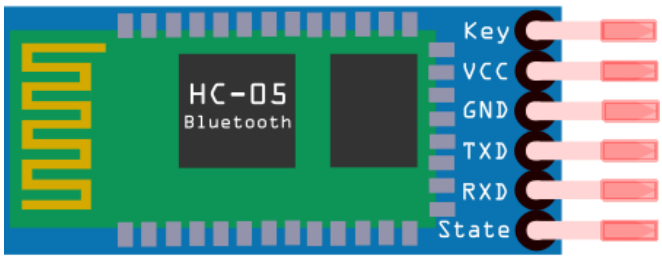
- Soldering Iron
- Insulation Tape
- Double Sided Tape
- Jumper Wires
- Single Strand Wires
- Wire Stripper/Cutter
- Screwdriver Set

3. CONNECTIONS

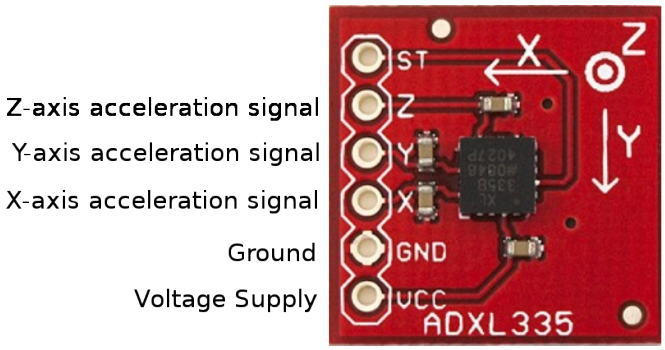
L293D IC Pinout



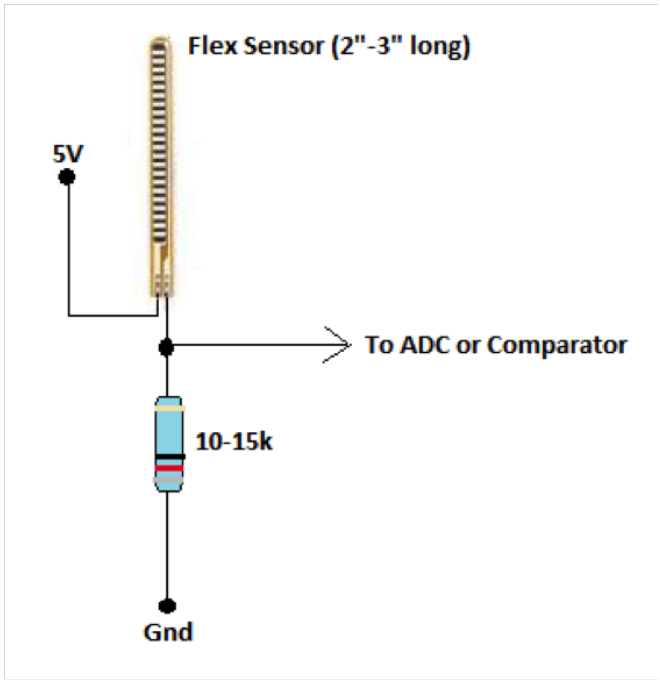
HC-05 Pinout



ADXL335 Pinout



Flex Sensor Connection



A. *Bot Side*

For L293D IC:

- Pin 1, Pin 8, Pin 9, Pin 16 to V_{cc}.
- Pin 4, Pin 5, Pin 12, Pin 13 to GND.
- Pin 2 to Digital Pin 3 of Arduino UNO.
- Pin 7 to Digital Pin 5 of Arduino UNO.
- Pin 10 to Digital Pin 6 of Arduino UNO.
- Pin 15 to Digital Pin 11 of Arduino UNO.
- Pin 3, Pin 6 to left dc motor of bot.
- Pin 11, Pin 14 to right dc motor of bot.

For HC-05:

- VCC of HC-05 to V_{cc} of Arduino UNO.
- GND of HC-05 to GND of Arduino UNO.
- RXD of HC-05 to Tx of Arduino UNO.
- TXD of HC-05 to Rx of Arduino UNO.

B. *Controller Side*

For ADXL335:

- VCC of ADXL335 to V_{cc} of Arduino UNO.
- GND of ADXL335 to GND of Arduino UNO.
- X of ADXL335 to A0 of Arduino UNO.
- Y of ADXL335 to A1 of Arduino UNO.

For Flex Sensor:

- One pin of flex sensor to 5V.
- Other pin to GND through a 10kΩ resistor.
- Pin connected to GND to A2 of Arduino.

For HC-05:

- VCC of HC-05 to V_{cc} of Arduino UNO.
- GND of HC-05 to GND of Arduino UNO.
- RXD of HC-05 to Tx of Arduino UNO.
- TXD of HC-05 to Rx of Arduino UNO.

4. Interfacing Accelerometer and Flex Sensor

Accelerometer and flex sensor output some analog value. But our microcontroller can deal with digital only. Thus we need to convert analog values to some digital values. This is done using ADC.

A. *ADC - Analog-to-Digital Converter*

Theory: The ATMega328p features a 10-bit successive approximation ADC. The ADC is connected to an 8-channel analog multiplexer which allows eight single-ended voltage inputs constructed from the pins of Port A. The single-ended voltage inputs refer to 0V (GND). The ADC converts an analog input voltage to a 10-bit digital value through successive approximation. The minimum value represents GND and the maximum value represents the voltage on the AREF pin minus 1 LSB. Voltage reference and input channel selections will not go into effect until ADEN is set. The ADC generates a 10-bit result which is presented in the ADC data registers, ADCH and

ADCL. By default, the result is presented right adjusted, but can optionally be presented left adjusted by setting the ADLAR bit in ADMUX. If the result is left adjusted and no more than 8-bit precision is required, it is sufficient to read ADCH. Otherwise, ADCL must be read first, then ADCH, to ensure that the content of the data registers belongs to the same conversion. Once ADCL is read, ADC access to data registers is blocked. This means that if ADCL has been read, and a conversion completes before ADCH is read, neither register is updated and the result from the conversion is lost. When ADCH is read, ADC access to the ADCH and ADCL registers is re-enabled.

B. Implementation in Program

I have written a function **analogRead** in 'AVR-ASM' which takes one argument (0-5) denoting the analog pin number (A0-A5) of Arduino UNO from which to read and returns a 16-bit value in right adjusted form. I am calling this function from 'C' program file. The following code snippet shows analogRead in 'AVR-ASM':

```
analogRead:
        ORI R24, 0B01000000
        STS ADMUX, R24
        LDI R24, 0B11000111
        STS ADCSRA, R24
wait:
        LDS R24, ADCSRA
        SBRS R24, ADIF
        RJMP wait
        LDI R24, 0B00010000
        STS ADCSRA, R24
        LDS R24, ADCL
        LDS R25, ADCH
        RET
```

5. Programing Bluetooth

In order to communicate between two bluetooth modules we need to configure them into master-slave mode. One bluetooth module is configured as master while other is configured as slave. Slave wait for a master to initiate a connection. We also configure them to connect to each other only by setting the other module address into the bind address of them. So as soon as modules are powered slave wait for master to connect to it while master search for slave and connect to it. To configure bluetooth modules we need to enter into AT Command Mode of bluetooth module and then set these settings. We also set the Baud Rate of the communication to 9600 in both the modules.

Now, in order to transfer data between microcontroller and bluetooth module we need to use USART of microcontroller.

A. USART - Universal Synchronous & Asynchronous Serial Receiver & Transmitter

Theory: It is a serial communication protocols used for transmitting and receiving data bit by bit with respect to clock pulses using only two wire. The AVR microcontroller has two pins: TXD and RXD, which are specially used for transmitting and receiving the data serially.

Setting up the BAUD rate: To establish a serial communication we need to specify the rate at which data is transmitted and received. This is known as the BAUD rate. The AVR microcontrollers do not take the BAUD rate directly but through a register **UBRR0 - USART0 Baud Rate Register**. The BAUD rate value is converted to this register value using the equation:

$$UBBR0 = \frac{ClockFrequency}{16BAUD} - 1$$

This equation converts BAUD rate into a 12-bit number. The High 4 bits is stored in the **UBRR0H** register and the low 8 bits is stored in the **UBRR0L** register.

Initializing the USART0: To be able to receive and transmit we need to work with the registers:

- **UCSR0B - USART0 control and status register B**
- **USCR0C - USART0 control and status register C**

We set the 4th bit **RXEN0 - Receiver Enable** to one and the 3rd bit **TXEN0 - Transmit Enable** to 1 of the UCSR0B to enable trasmitting and receiving of serial data. We also need to determine the number of data bits to send per frame. To do this we enable the bit 2 **USCZ01** to 1 and bit 1 **UCSZ00** to 1 of the UCSR0C register. Setting these two bits to 1 sets up a 8-bit data frame.

Date Buffer Register 0: The receiving and transmitting data buffer register share the same I/O address. It is the **UDR0 - USART0 Data Register 0**. It is a 16 bit register. The received data goes into the upper 8 bits and the transmit data goes into the lower 8 bits. When data is being sent store the data in the UDR0 register and send it serially. When data is being read we just return the value in the UDR0 register.

B. Implementation in Program

I have written a function **USART0_Init** in 'C' which when called initialize the USART0 and begin serial communication between Arduino UNO and Bluetooth Module. I have implemented a function **SerialWrite** which takes a 8-bit value and send it to the Rx pin of Arduino UNO serially. I have also written a function **SerialRead** which reads 8-bit data send serially to the Tx pin of Arduino UNO and return it. All these functions are combined and placed in a 'C' header file (usart.h) as shown in the following code snippet:

```
#ifndef F_CPU
#define F_CPU 16000000UL
#endif

#ifndef BAUD
#define BAUD 9600
#endif

void USART0_Init(void) {
        uint16_t UBRRVAL = ((F_CPU
            )/(16UL*BAUD) - 1);
        UBRR0H = (uint8_t)(UBRRVAL
            >>8);
        UBRR0L = (uint8_t)UBRRVAL;
        UCSR0C = (1<<UCSZ01)|(1<<
            UCSZ00);
        UCSR0B=(1<<RXEN0)|(1<<
            TXEN0);
}

void SerialWrite(uint8_t u8Data) {
    while(!(UCSR0A&(1<<UDRE0)));
    UDR0 = u8Data;
}

uint8_t SerialRead(void) {
        while(!(UCSR0A&(1<<RXC0)))
            ;
        return UDR0;
}
```

6. Controlling Bot Speed

Analog values send by flex sensor are converted into digital values using ADC at controller side and then send to Bot wirelessly using bluetooth. These values there again converted into analog voltage to control the speed of DC motor. This is done using PWM.

A. PWM - Pulse Width Modulation

Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the pulse width. To get varying analog values, you change, or modulate, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 5v controlling the brightness of the LED. The ratio of the time for which pulse is high to the total time interval is called **duty cycle** of pulse.

B. Timer

In avr microcontroller we can generate pulse of varying duty cycles using timer. ATMega328p timers can be programmed in **phase-correct pwm** mode to generate a specific voltage between 0V-5V on digital pins with pwm support, that is, those which are directly connected to output compare pin of timer.

ATMega328p timer can run in 4 modes:

- Normal Mode
- CTC Mode
- Fast PWM Mode
- Phase Correct PWM Mode

Here we are using Timer0 and Timer2, both 8-bit timer, in Phase Correct PWM mode as it provide

large range of operation and thus high precision, which is required to accurately control the speed of DC motor.

C. *Implementation in Program*

I have written a two functions **pwmSetup** and **analogWrite** in 'AVR-ASM' and called it in 'C'. pwmSetup is called in the beginning of program to setup Timer0 and Timer2 in phase correct pwm mode. analogWrite function takes two arguments. First one denote the digital pin number (3, 5, 6 or 11) on which to produce analog value and second one is a 8-bit value (0 corresponds to 0V while 255 corresponds to 5V).

```
pwmSetup:
        LDI R24, 0B10100001
        OUT TCCR0A, R24
        STS TCCR2A, R24
        LDI R24, 0b00000001
        OUT TCCR0B, R24
        STS TCCR2B, R24
        RET

analogWrite:
        CPI R24, 3
        BREQ 13
        CPI R24, 5
        BREQ 15
        CPI R24, 6
        BREQ 16
        CPI R24, 11
        BREQ 111
        RET
13:
        STS OCR2B, R22
        RET
15:
        OUT OCR0B, R22
        RET
16:
        OUT OCR0A, R22
        RET
111:
        STS OCR2A, R22
        RET
```

7. Instructions to dump the code

After connecting the bot and controller as explained in the above sections, first connect the bot side Arduino UNO to computer and type in terminal make. This make file build the project and also upload the hex file to ATMega328p. Similarly do it for controller.

8. References

- Atmel AVR ATmega328P Datasheet
- AVR Freaks
- "Microcontroller and Embedded System" by Muhammad Ali Mazidi and Sarmad Naimi