EE318 Electronic Design Lab Report

Title of Project:

Hand Gesture Controlled Wireless Vehicle

Group D11

Ashay Awate(06D07012),
Milind Kothekar(06D07017),
Neha Rambhia(06D07011)

EDL Guide: Prof. Apte
Course Instructor: Prof. Vivek Agarwal

Abstract

The project is to make a vehicle that can be controlled wirelessly by hand gestures. This involves two modules. One is the glove that translates the motions of the hand using an accelerometer and wirelessly through RF modules transmits it to the vehicle in question. The vehicle here being the second module that interprets the signal sent in by the accelerometer and modifies its motion with respect to the signal given by the hand gesture. The car itself simulates the motion of a vehicle using a servo for its steering mechanism and steppers for its movement, speed control.

At the end of 11 weeks, the project has achieved its basic objective and can make extremely basic turns as well as modify the speed of the vehicle. More complicated motion can definitely be extrapolated from this basic behavior.

1) Introduction:

Everyone has imagined being able to move an object with a flick of their wrist at their whim at some point or the other. A very natural tendency to move objects with the movement of ones hand is something of great convenience, and to be able to control a car, its speed, direction with the same motions is every child's dream.

In our project we have put together a car, which is a model of the cars we drive today, with servo motors controlling the direction of the car's motion, which is controlled by a glove. The glove is worn by the user, is mounted by accelerometers and it transmits data wirelessly to the car.

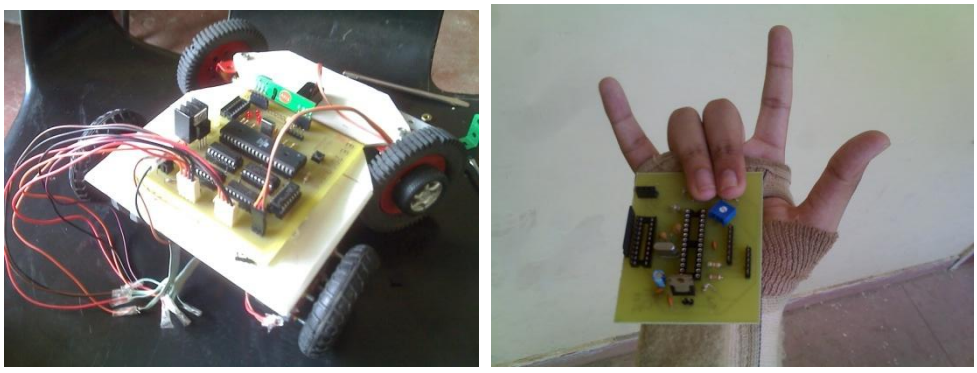


Fig 1. The car and the glove

2) Project Modules

A. The glove

This module has the accelerometers and an atmega 8 that helps the flow of data in the circuit of the glove, or is the brain of the glove.

B. The Car

In the car module we have an atmega16 that controls in the received signals, interprets it and controls the motions of the car. It is also connected to the stepper and servo motors of the car

C. The RF circuit

This is the transmitting and receiving module of the entire setup. An ASK transmitter is fitted onto the glove, which facilitates transmission of the accelerometer data from thr glove to the car. The car is fitted with the receiver.

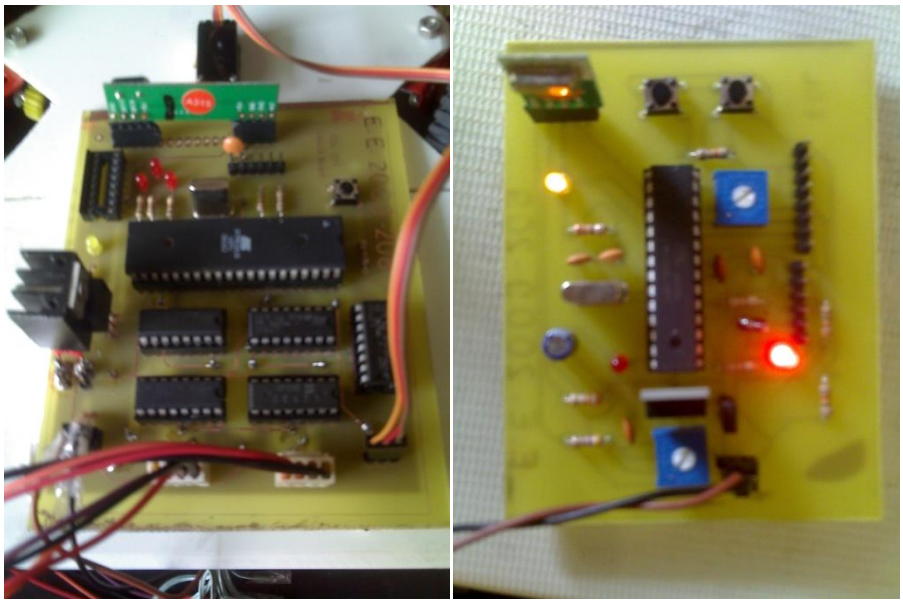


Fig 2. PCB boards of the car and the glove respectively.

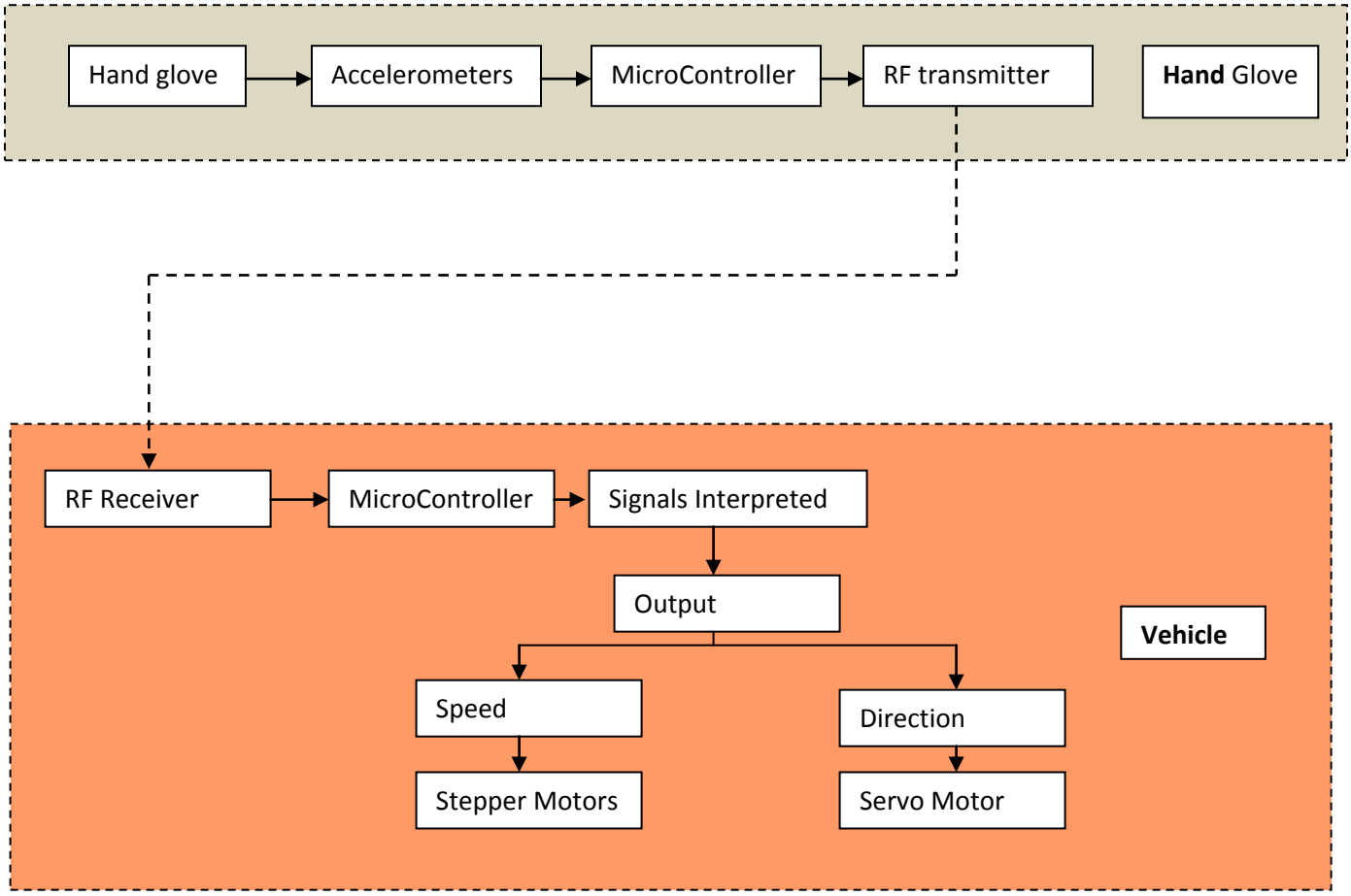


Fig 3. Block Diagram of the entire project

3) Components of the circuit

a. Accelerometers

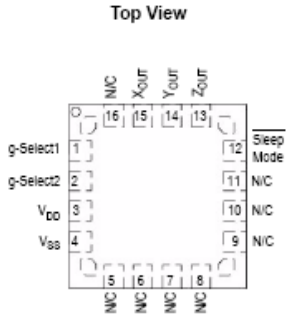


Fig.4 top view of MMA7260

We have to measure acceleration in two directions (X and Y) to change the speed and direction of motion. The accelerometers help us measure the direction and the magnitude of acceleration in that direction, hence recognizes the hand gestures. As to which direction and how fast we are moving the hand. Therefore we decided to use a 3 axis accelerometer for the purpose.

After some searching we narrowed down on a product from Freescale- MMA7260. This is a 3 axis accelerometer. The IC package is 16 pin QFN package. A PCB has been designed to test the accelerometer. The accelerometer has been tested, and the corresponding readings have been calibrated in the code, to run the stepper motor accordingly.

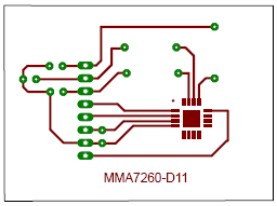


Fig.5 PCB for accelerometer testing

The features of this sensor include:

- Variable sensitivity selection: 1.5-6g
- Sleep mode
- Low power consumption: voltage: 2.2-3.6V current: 550uA
- Low cost

Principle of working:

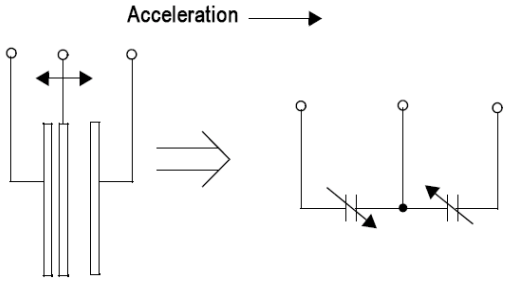


Fig6. Simplified Transducer Physical Model

The accelerometer uses switched capacitor techniques to measure the g-cell capacitors and extract the acceleration data from the difference between the two capacitors. The ASIC also signal conditions and filters (switched capacitor) the signal, providing a high level output voltage that is ratiometric and proportional to acceleration.

Problems we faced:

Soldering the IC was very difficult because of the package.
After soldering, the accelerometer was not found to work. AT the X,Y and Z outputs, the voltage observed was zero.
Possible cause of damage: soldering may not be proper, accelerometer may have been overheated while soldering, while doing the continuity test, the currents entering the device may have damaged it.
Another board and IC was used and worked. We took a few readings of the accelerometer at in various tilts. The second IC on further testing also stopped working. Cause: spike in the IC probably.

b. Atmega microcontrollers

In the project, there are two microcontrollers that we need. One for the handglove and the second for the car. We chose microcontroller of the Atmel's AVR series because of the following reasons:

- Some members of our group were familiar with AVR coding.
- The ISP programmer (parallel port and USB) for the AVR microcontroller are easy enough to make.
- There are good development tools available for writing and debugging programs including WinAVR and AVR Studio.
- They are easily available.
- The microcontrollers available in the series suited our required specifications.

Hand Glove Controller:

The microcontroller for the hand held controller had to have the following specifications:

- It had to be small since it was to be mounted on a hand.
- Since it would take in the analog voltages of the accelerometer it would be preferred that it would have ADC's.
- It should work on the voltage range of 2.7v to 3.6V.
- It should have inbuilt protocol to help in transmitting of data.

The ATMEGA8L microcontroller was found to suit all these specifications.

Vehicle Microcontroller:

The microcontroller for the vehicle had to have the following specifications:

- It had to have some inbuilt protocol to receive the data sent by the hand glove(USART)
- It should have some ports available to drive a stepper motor and a servo motor.
- It should have a low power consumption.
- Working on 5 volts was preferred.
- It should have a good interrupt service routine, since we had planned to use interrupts.
- It should have a good crystal frequency.

A lot of Microcontrollers satisfy these criterion, but we chose the ATMEGA16 microcontroller.

c. ASK transmitters and receivers

We are using ASK (Amplitude Shift Keying) transmitter and Receiver circuits with USART protocol.

Configuration:

- 433 MHz,
- ASK modulation
- 1 kbps nominal data rate
- Maximum 10 kbps.

The Tx and Rx were chosen because of the simplicity of use, relevant data rate. The range is 2m without antennas, which is relevant and suits the wireless handling we need.

The minimum data rate we required was 720 bps as we wanted to sample each of the axis readings 30 times a second.

Data rate = 30(sampling rate of each axis) x 3(no of axes) x 8(no of bits per packet of data)

= 720 bps

The data is transmitted serially in the form of data packets.

4) The vehicle

Fig 7. Above side view of the car, left is the top view of the car.

The vehicle is a 4 wheel vehicle that is driven by synchronized stepper motors. The turning of the same is dependant on servo motors.

Specifications of stepper motors:

We decided to use stepper motors to provide accurate control for our vehicle. We tested a bipolar stepper motor. Made a test circuit of the stepper motor drivers using IC L293D.

The circuit worked, and motor runs at different speeds, I-V characteristics were also measured.

Specifications of Servo Motor:

A servo motor, gives a particular angle of the shaft for a certain time period (the duty cycle) of a square wave. It gives very high torque and holds the angle till it is enabled or is reset to original duty cycle. The servo motor we are using has the following specifics:

4.8-6V 0.12/ 0.10s / 60^{o}

The servo motor is connected to a shaft in the car, the axle which turns the wheels.

Fig 8. The Image shows left and right turning respectively of the shaft. The dark blue block is the servo motor.

5) The software

a. Coding for the glove

On the glove, we need to code the timing of the entire circuit to ensure synchronous operations. Besides which we need to take in analog output of the accelerometer and not only convert it digitally, but also send it though the transmitter.
Here the data is coded in the form of data packets. When the circuit is enabled, that is a valid hand gesture is being given.
The data packet is of the form:
Start byte1
Start byte2
X-axis reading
Y –axis reading
Stop byte
This format also helps us ensure that we have proper data transmission and helps us check for errors. Besides this is helps us save power by transmitting only zero in all other cases.

b. Coding for the car

Coding for the car starts with synchronizing all the circuitry on the vehicle.
We receive data bytes from the receiver. These data packets which some in serially byte at a time are decoded as x-axis reading, y-axis reading provided the start bytes and stop byte are correct i.e. we are sure that the data bytes are being correctly transmitted.
These x and y axis readings are then further used to calculations for speed and angle changes.

The x axis here represents the forward backward motion i.e. the speed of the vehicle.
It is neutral while the reading lies between 140 to 160.
Greater than 160, we increase the speed by 1 step and less than 140 we decrease it by 1 step. We have 10 possible such speeds to be worked upon.

The y axis represents the turning.
While we get reading from the y-axis accelerometer, the servo will hold a fixed angle (30 deg) for a time duration proportional to the duration of the signal coming in from the accelerometer. Valid readings again lie in the range >160 and <140

6) Test Procedure:

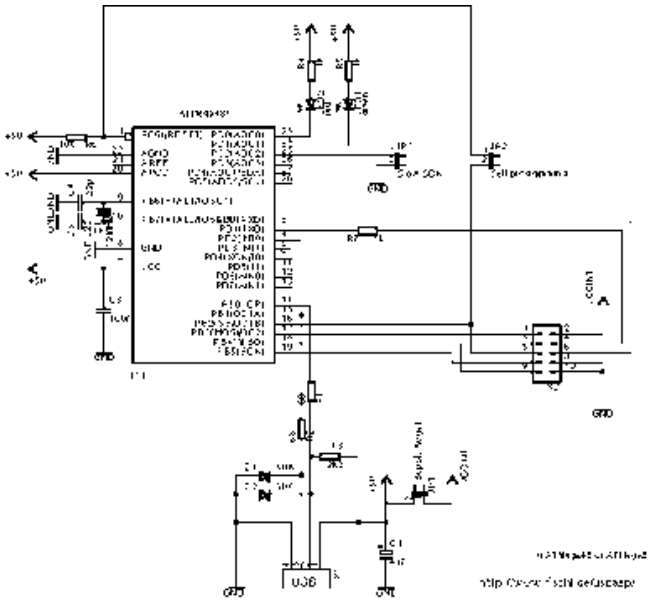
A programmer board and application board has been made on a breadboard.
Various codes have been tested successfully on this microcontroller including:

- Testing of Timers
- Testing of ADC's and they multiplexing
- Testing of USART protocol to transmit as well as receive.

It is being used to take in the input of the accelerometer and transmit it through USART transmitter.

Programmer Board:

Initially we were programming our Microcontrollers on the Universal Programmer available in the lab. However to make it convenient for us to test our codes, we have built an USB programmer to program our AVR microcontrollers. It is an In-system Programmer, so it saves us the hassle of constantly removing the microcontroller each time we want to test a code. We soldered the circuit of the programmer on a general purpose PCB.



a. The motors:

The motors were individually tested

- Testing of codes to drive both Unipolar as well as Bipolar Stepper Motors.
- Code to receive data and accordingly modify speed of stepper motor.
- Testing the various angle of a servo motor
- Dynamically changing angles of the servo motor.

In the demo, the Microcontroller is being used to receive data from a USART receiver, process the data and modify the speed of a stepper motor accordingly. The motor has been calibrated to work at 10 speeds, and will increase or decrease in steps depending on the accelerometer reading.

And the servo motor is set to turn at 2 angles besides neutral +/- 30deg

8) Conclusion and suggestions for further improvement.

The project gave us insight into the uses or sensor like the accelerometer
It has also taught us communication theory using RF modules. Giving us insight into programming of AVR and mechanical skill when it comes to making an integrated aesthetically sound system

It at the end has turned out to be a very basic model where the hand gesture and corresponding response from the car can be refined as well as we can add meaning to the third axis which we are not currently using.

The applications of this kind of system are varied. They can be used to manipulate vehicles and objects, detectors in places where fine control is required as well as manual handling is not possible.

Appendix

User manual

For the user:
In order to use the mechanism.
Reset car.
For the handglove. Press the button when you want to give a valid command.
While keeping the button pressed,
Moving hand forward and releasing button: increases speed
Moving hand backwards and releasing button: decreased speed
Moving hand to right: turns car to right
Moving hand to left: turns car left
(right and left turning depends on the duration of the hand gesture)

Codes for the hand glove

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#define F_CPU 4000000UL  // 4 MHz
//#include <UART.c>

//Bit Function definitions
# define tbi(x,y) ( (x) ^= (1<<(y)) )
# define sbi(x,y) ( (x) |= (1<<(y)) )
# define cbi(x,y) ( (x) &= ~(1<<(y)) )
```

```
# define isHigh(x,y) (x & (1<<y))

#define LED_port PORTD

//USART definitions
#define BAUD 1200
#define MYUBRR F_CPU/16/BAUD-1

#define START_BYTE1 0x55
#define START_BYTE2 0xAA
#define STOP_BYTE   0x44

char readX(void)
{
        char a;
        ADMUX = (0<<REFS1) | _BV(REFS0) | _BV(ADLAR) | (0<<MUX3) | (0<<MUX3)
|(0<<MUX1) |(0<<MUX0);
        ADCSRA = _BV(ADEN) | _BV(ADSC) | (0<<ADFR) | _BV(ADIE) | (0<<ADPS2) | (0<<ADPS1)
| (0<<ADPS0);
        _delay_us(1);
        while(!(isHigh(ADCSRA,ADIF)));          //Wait for conversion to complete
        a = ADCH;
        ADCSRA |= _BV(ADIF);
        return (a);
}

char readY(void)
{
        char a;
        ADMUX = (0<<REFS1) | _BV(REFS0) | _BV(ADLAR) | (0<<MUX3) | (0<<MUX3)
|(0<<MUX1) |_BV(MUX0);
        ADCSRA = _BV(ADEN) | _BV(ADSC) | (0<<ADFR) | _BV(ADIE) | (0<<ADPS2) | (0<<ADPS1)
| (0<<ADPS0);
        _delay_us(1);
        while(!(isHigh(ADCSRA,ADIF)));          //Wait for conversion to complete
        a = ADCH;
        ADCSRA |= _BV(ADIF);
        return (a);
```

```
}

char readZ(void)
{
        char a;
        ADMUX = (0<<REFS1) | _BV(REFS0) | _BV(ADLAR) | (0<<MUX3) | (0<<MUX3) |
_BV(MUX1) | (0<<MUX0);
        ADCSRA = _BV(ADEN) | _BV(ADSC) | (0<<ADFR) | _BV(ADIE) | (0<<ADPS2) | (0<<ADPS1)
| (0<<ADPS0);
        _delay_us(1);
        while(!(isHigh(ADCSRA,ADIF)));          //Wait for conversion to complete
        a = ADCH;
        ADCSRA |= _BV(ADIF);
        return (a);
}

char readREF(void)
{
        char a;
        ADMUX = (0<<REFS1) | _BV(REFS0) | _BV(ADLAR) | (0<<MUX3) | (0<<MUX3) |
_BV(MUX1) | _BV(MUX0);
        ADCSRA = _BV(ADEN) | _BV(ADSC) | (0<<ADFR) | _BV(ADIE) | (0<<ADPS2) | (0<<ADPS1)
| (0<<ADPS0);
        _delay_us(1);
        while(!(isHigh(ADCSRA,ADIF)));          //Wait for conversion to complete
        a = ADCH;
        ADCSRA |= _BV(ADIF);
        return (a);
}


//===============USART FUNCTIONS=====================//
 void Init_UART_TX( unsigned int baudrate )
  {
    DDRD |= (_BV(PD1));          //setting output port (ATMEGA16)
        PORTD |= 0x02;

        UBRRH= (unsigned char) (baudrate>>8);
```

```
   UBRRL= (unsigned char) baudrate;

        //USART control and status register
        //                rx int en                                   data size
        UCSRB =   _BV(TXEN) | (0<<UCSZ2);//| _BV(TXEN) ;_BV(RXCIE) |
        //                write UCSRC             asynch                  //even parity    stop    bits(rx
ignor)  8 bits of data
        UCSRC  =   _BV(URSEL)  |  (0<<UMSEL)  |  _BV(UPM1)  |  (0<<UPM0)  |  _BV(USBS)     |
_BV(UCSZ1)| _BV(UCSZ0) | (0<<UCPOL);
 }


 void TransmitByte( unsigned char data )
 {
        while ( !(UCSRA & (_BV(UDRE))) );       // Wait for empty transmit buffer
                UDR =  data;  // Start transmition
        tbi(PORTD,PD7);
 }


int main(void)
{
        DDRD = 0xff;            //Configure PortD for output
        PORTD = 0xff;

        _delay_ms(800);
        char read_data;
        char read_data_X,read_data_Y,read_data_Z,read_data_REF;

        //tbi(PORTD,PD3);

        Init_UART_TX(BAUD);
        //sei();


        //DDRB= 0b00011111;
        //PORTB=0b00011111;
```

```
	DDRC=0x00;			//Port C- ADC as input
	PORTC=0x00;

	//int j;

	//	AVCC selected(Aref = cap)	Selects ADC0 as source

	//ADMUX = (0<<REFS1) | _BV(REFS0) | _BV(ADLAR) | (0<<MUX3) | (0<<MUX3)
|(0<<MUX1) |(0<<MUX0);

	//		enable ADC	Start conv  no free run   int enable
Prescalar = 2
	//ADCSRA = _BV(ADEN) | _BV(ADSC) | (0<<ADFR) | _BV(ADIE) | (0<<ADPS2) |
(0<<ADPS1) | (0<<ADPS0);

/*
	while(isHigh(ADCSRA,ADIF));  //Wait for conversion to complete

	ADCSRA |= _BV(ADSC);
*/
	PORTD = 0x0e;
	_delay_ms(800);
	_delay_ms(800);

	PORTD = 0x00;


	while(1)
	{

		//tbi(PORTD,PD6);
		//_delay_ms(800);
		//ADCSRA |= _BV(ADSC);		//start conversion
		//_delay_us(10);

		//while(!(isHigh(ADCSRA,ADIF)));	//Wait for conversion to complete
		read_data_X =readX();
		read_data_Y =readY();
```

```
read_data_Z =readZ();
read_data_REF =readREF();
PORTD = (read_data&0xfe)|(PORTD&0x01);
_delay_ms(100);
TransmitByte(START_BYTE1);
TransmitByte(START_BYTE2);
TransmitByte(read_data_Z);
TransmitByte(0x00);                                    //Servo control
TransmitByte(STOP_BYTE);
//tbi(PORTD,PD5);
_delay_ms(10);
//PORTD=0xff;
//ADCSRA |=  _BV(ADIF);       //Clear the interrupt flag
//_delay_ms(100);
//
//_delay_ms(800);
//read_data =readY();
//PORTD = (read_data&0xfe)|(PORTD&0x01);
//tbi(PORTD,PD4);
//_delay_ms(800);
//_delay_ms(800);
//_delay_ms(400);



//for(int i=0; i<4; i++)
//{
        //int p=1;

        //for(j=0; j<=i;j++)
        //        {p=p*2;}
        //ADMUX=0b00100000;

//ADCSRA=(1<<ADEN)|(1<<ADSC)|(1<<ADFR)|(0<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<
ADPS0);


        //while((ADCSRA&(1<<ADIF))==(1<<ADIF));
```

```
                        //int k = ADCH;
                        //PORTB |= (k&(0b00001111));
                        //tbi(PORTB,PB4);
                        //tbi(PORTB,PB1);
                        //tbi(PORTB,PB2);
                        //tbi(PORTB,PB3);

        //ADCSRA=(0<<ADEN)|(0<<ADSC)|(0<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
                        //_delay_ms(400);
                //}
        }

}
```

Codes for the car

```
//Include files
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>


//Bit Function definitions
# define tbi(x,y) ( (x) ^= (1<<(y)) )
# define sbi(x,y) ( (x) |= (1<<(y)) )
# define cbi(x,y) ( (x) &= ~(1<<(y)) )
# define isHigh(x,y) (x & (1<<y))

#define Motor_port PORTC
#define m_a1 PC0
#define m_a2 PC1
#define m_b1 PC2
#define m_b2 PC3



#define XTAL            4000000L    // Crystal frequency in Hz
#define TIMER_PRE    1024     // Timer prescaler
```

```
//usart coding

#define BAUD 1200
#define MYUBRR F_CPU/16/BAUD-1

int max1=170, max2=190, min1=140, min2=120;

//usart initialization

 void Init_UART_RX( unsigned int baudrate )
 {
   DDRD &= ~(_BV(PD0));       //setting input port (ATMEGA16)
          PORTD |= 0x01;

          UBRRH= (unsigned char) (baudrate>>8);
   UBRRL= (unsigned char) baudrate;

          //USART control and status register
          //                  rx int en                                                  data size
          UCSRB =  _BV(RXCIE) | _BV(RXEN) | (0<<UCSZ2);//| _BV(TXEN) ; ???????????
          //                  write UCSRC              asynch                    //even parity     stop     bits(rx
ignor)  8 bits of data
          UCSRC  =    _BV(URSEL)  |  (0<<UMSEL)  |  _BV(UPM1)  |  (0<<UPM0)  |  _BV(USBS)      |
_BV(UCSZ1)| _BV(UCSZ0) | (0<<UCPOL);
 }



 /*-------------------------------------------------------------
------------FUNCTIONS TO READ UART------------------------------
-------------------------------------------------------------
 unsigned char ReceiveByte( void )
 {
  while ( !(UCSRA &  (_BV(RXC))) );     /  Wait for incomming data
  return UDR;// Return the   data
 }
*/
```

```
int speed;

void configure_speed(int a)
{
	int increment=0;


//AT 3 VOLTS
//NOMINAL A = 0X96  150
//lOW A = 0X3C		60
//HIGH A = 0XD2		210

	if(a>160)
	increment=1;
	//if(a<175&&a>160)
	//increment=2;
	//if(a<1.5&&a>1.25)
	//increment=1;
	//if(a<1.2&&a>1)
	//increment=-1;
	if((a<140)&&(a>90))
	increment=-1;
	//if((a<90)&&(a>60))
	//increment=-3;
	speed=speed+increment;
	if(speed>10)
		speed=10;
	if(speed<0)
		speed=0;

}


int max;
void update(unsigned char b, int f)
{
	int a;
```

```
	a=(int)b;
	if(f==0)
	{
		a=max;
		configure_speed(a);
		max=0;

	}
	if(f==1)
	{
		if(a>max)
		max=a;
	}

}


// Interrupt Routine for receiving data

ISR(USART_RXC_vect)
{
	tbi(PORTA,PA7);
	unsigned char a,status;
	status = UCSRA;
	a = UDR;	//Reads the UART data reg
	//1st bit indicates which accelerometer
	if (!( status&(_BV(FE)|_BV(DOR)|_BV(PE)) )) //If ther is no error
	{
		int f=0;
		if((a&0x80)==0x80)
			f=1;
		//PORTA = a;
		update(a,f);
	}



}
```

```
//Using Interrupts
void Timer_Init(void)
{

	//Timer/Counter Control Register – TCCR0
	cli();
	OCR0 = 0x60;
	//		force o/p comp bit		need to set CTC mode
	toggle OC0 on compare match		clock is now fclk/1024
	TCCR0 |= _BV(WGM01) | _BV(COM00) | _BV(CS01) | _BV(CS00);//|
_BV(CS02);//(FOC0<<0) | (WGM00<<0) | (COM01<<0) |

	//TCCR1A = (COM1A1<<0);//; | _BV(COM1A0); //| (WGM11<<0)| (WGM10<<0);//|
_BV(FOC1A)
	//TCCR1B |= _BV(WGM12) ;// | (CS11<<0) | _BV(CS10);//| _BV(CS12) |(WGM13<<0)|


	//Timer/Counter Interrupt Mask Register
	//Note: This reg contains controls for all timers so be careful
	//		int enable what		???????
	//Interrupt enabled
	TIMSK |= _BV(OCIE0) | (TOIE0<<0)| _BV(OCIE1A);
	//sei();
}

#define servo_pin PD4

void servo_enable(void)
{
	DDRD |= _BV(servo_pin);
	sbi(PORTD,servo_pin);

}
```

```
void servo_disable(void)
{
	DDRD |= _BV(servo_pin);
	cbi(PORTD,servo_pin);
	cbi(PORTD,PD5);
}



void PWM_Timer_Init(void)
{
	sbi(DDRD,PD5);		//set OCR1a pin for output
	sbi(DDRD,PD7);
	//time = 0.1 ms = 10-4s
	//freq = 10kHz=10000

	ICR1=12000;

	if(speed%3==0)
		OCR1A =350;
	else if(speed%3==1)
		OCR1A =500;
	else
		OCR1A =1000;

	//650 angle=0


	//=========================================================================
=========================================
	//T		ode....

	//TImer1 control reg
	//			clear OC1A on comp			OC1B:ignore
	not needed in non PWM mode			fast PWM ICR1 as OP
	TCCR1A		= (0<<COM1A0) |_BV(COM1A1)	|	(0<<COM1B0)	|(0<<COM1B1)	|
(0<<FOC1A) | (0<<FOC1B)	| _BV(WGM11)|(0<<WGM10);
```

```
	//			do not use			mode setting
	clock=1/256
	TCCR1B		=
(0<<ICNC1)|(0<<ICES1)|(1<<WGM13)|(1<<WGM12)|(0<<CS12)|(1<<CS11)|(0<<CS10);



	TIMSK  |= (0<<TICIE1) | _BV(OCIE1A);

	//Calculations
	//cs00=== 010
	//prescalar =8
	//fer = 8 Mhz.
	//scaled Freq = 0.5 MHz
	//time wanted = 20 ms
	//freq wanted = 1/ 20 /10-3= 1000/20= 50Hz
	//ICR= 500000/50= 10000 in dec
	//for 1 ms control signal
	//freq wanted = 1/10-3= 1000
	//OCR= 500000/1000= 500



}

void PWM_timer_stop(void)
{
	//select no clock source..
	//stops timer
	TCCR1B                                                                  =
(0<<ICNC1)|(0<<ICES1)|(1<<WGM13)|(1<<WGM12)|(0<<CS12)|(0<<CS11)|(0<<CS10);
}


int motor_state = 0;		//This is the state of which coil is on.
		//This goes from 0 to 3
```

```
ISR(TIMER1_COMPA_vect)
{
	tbi(PORTA,PA7);
	//tbi(PORTD,PD5);
}

ISR(TIMER0_COMP_vect)
{
	//tbi(PORTA,PA1);
//tbi(PORTD,PD5);

	//sbi(PORTC,PC5);
		if(motor_state==0)
	{	Motor_port = 0b00000001;
		motor_state=1;
		sbi(PORTC,PC5);
	}
	else if(motor_state==1)
	{	Motor_port = 0b00000010;
		motor_state=2;
		sbi(PORTC,PC6);
	}
	else if(motor_state==2)
	{	Motor_port = 0b00000100;
		motor_state=3;
		sbi(PORTC,PC7);
	}
	else if(motor_state>=3)
	{	Motor_port = 0b00001000;
		motor_state=0;
		//sbi(PORTD,PD5);
	}
};

void update_speed(void)
{
	if(speed==1)
	{	OCR0 = 0xD0; }//((XTAL/2/TIMER_PRE/50) - 1 );   }//20 ms delay
```

```
	if(speed==2)
	{	OCR0 = 0xC0; }
	if(speed==3)
	{	OCR0 = 0xB0; }
	if(speed==4)
	{	OCR0 = 0xA0; }
	if(speed==5)
	{	OCR0 = 0x90; }
	if(speed==6)
	{	OCR0 = 0x80; }
	if(speed==7)
	{	OCR0 = 0x70; }
	if(speed==8)
	{	OCR0 = 0x60; }
	if(speed==9)
	{	OCR0 = 0x50; }
	if(speed==10)
	{	OCR0 = 0x40; }//=64
	if(speed==11)
	{	OCR0 = 0x35; }

}

int main(void)
{
	//Seeting prot D to output to run motors
	DDRC = 0xff;
	motor_state = 0;
	DDRA = 0xff;
	PORTA=0x00;
	_delay_ms(800);


	//tbi(PORTA,PA4);
	//int speedy=500;
	//int a = 1000/speedy;
	//Init_UART_RX(BAUD);
	Timer_Init();
```

```
PWM_Timer_Init();
//int i;


sei();
//DDRD = 0xff;

int i;
speed=0;

while(1)
{
        //tbi(PORTA,PA5);

        for(i=1;i<=10;i++)
        {
        speed=i;
        //update_speed();

        PORTA=(((char)speed)|((PORTA)&0b11100000));
        PWM_Timer_Init();
        servo_enable();
        _delay_ms(500);
        _delay_ms(500);
        _delay_ms(500);
        _delay_ms(500);
        //servo_disable();
        //PWM_timer_stop();
        servo_disable();
        _delay_ms(500);
        _delay_ms(500);
        _delay_ms(500);
        _delay_ms(500);
        _delay_ms(500);



        }
```

```
        }
        return 0;
}
/*
20ms = 20 *10-3 s
4 Mhz => 1/ 4 / 106= 2.5 * 10-7ses
OCR 0 = 8000 in decimal = 1f40
There is a factor of 2
so
OCR0 = 4000 = 0x0FA0
ftoggle=1000/20=50
1/18
*/
```