Ping–Pong Player

University of Florida
EEL 5666
Intelligent Machine Design Lab

Student Name: Sanjay Solanki
Date: 9th December 2002
Instructor: Dr.A.A.Arroyo

Table Of Contents

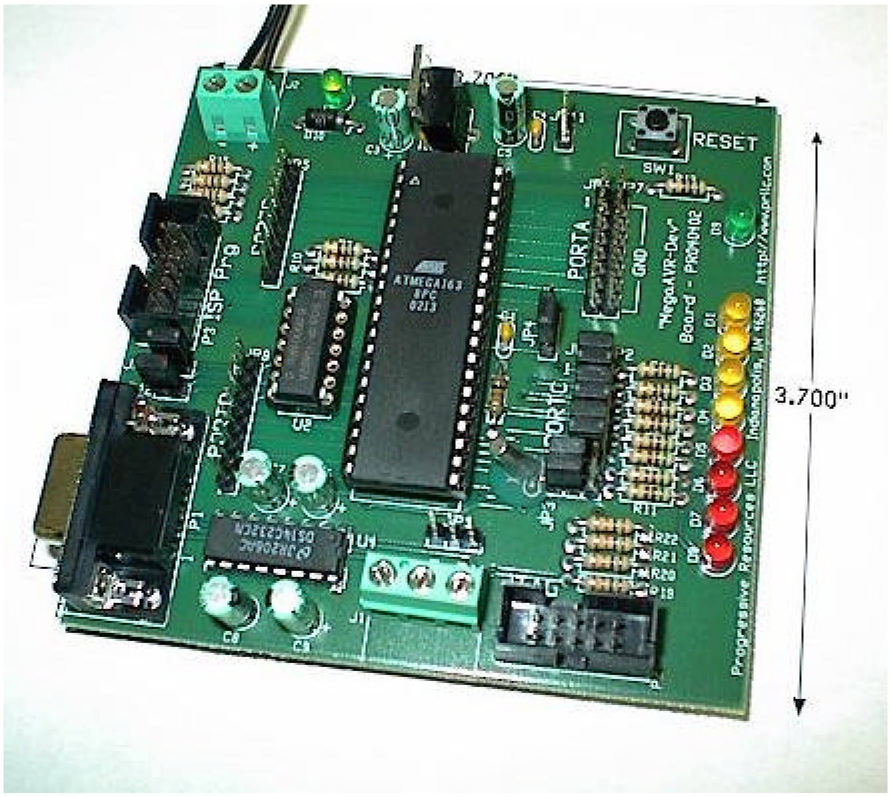
Abstract

Ping Pong Player is a mobile robot to help you practice Ping Pong. It shoots balls on the other end of the net as it moves on the table. There is a space provided on the robot, if the player aims the ball right into the space the robot will use the same ball again and a Buzzer will sound.

Introduction

The Ping Pong player has a total of 5 actuators. 2 actuators for the mobile platform and 3 for the arm to pick and shoot the ball. It uses the Atmaega323 microprocessor and has IR sensors, Phototransistors, Limit switch and Flex sensor

Microprocessor
The microcontroller used is Atmega323 on the MegaAVR development board from progressive resources.



Features

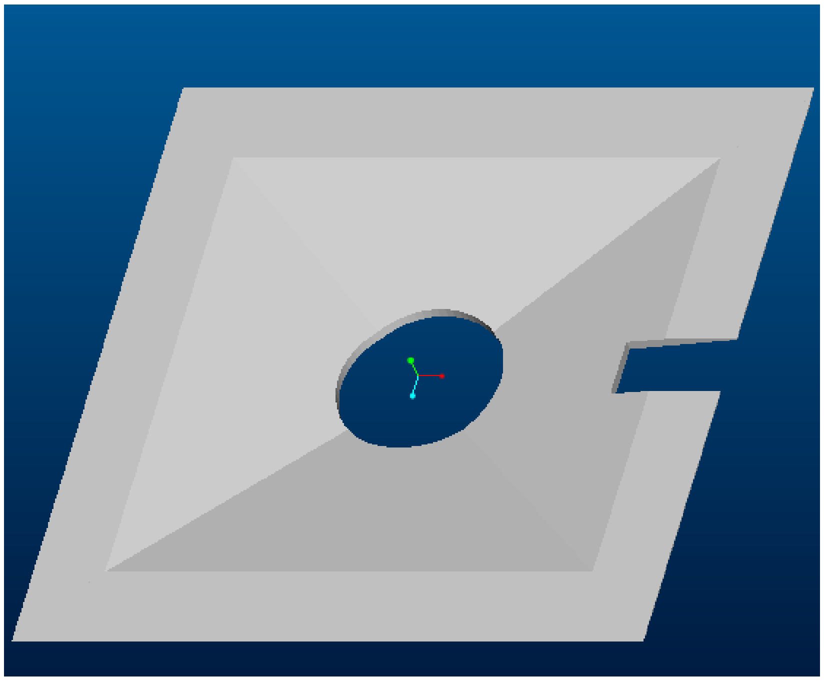
- High-performance, Low-power AVR® 8-bit Microcontroller
- Advanced RISC Architecture
  - 130 Powerful Instructions – Most Single-clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 8 MIPS Throughput at 8 MHz
  - On-chip 2-cycle Multiplier
- Non-volatile Program and Data Memories
  - 32K Bytes of In-System Self-programmable Flash
    Endurance: 1,000 Write/Erase Cycles
  - Optional Boot Code Section with Independent Lock Bits
    In-System Programming by On-chip Boot Program
  - 1K Byte EEPROM
    Endurance: 100,000 Write/Erase Cycles
  - 2K Bytes Internal SRAM
  - Programming Lock for Software Security
- JTAG (IEEE Std. 1149.1 Compliant) Interface
  - Extensive On-chip Debug Support
  - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
  - Boundary-Scan Capabilities According to the JTAG Standard
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Four PWM Channels
  - 8-channel, 10-bit ADC
  - Byte-oriented Two-wire Serial Interface
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated RC Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby
- I/O and Packages
  - 32 Programmable I/O Lines
  - 40-pin PDIP and 44-lead TQFP
- Operating Voltages
  - 2.7 - 5.5V (ATmega323L)
  - 4.0 - 5.5V (ATmega323)
- Speed Grades
  - 0 - 4 MHz (ATmega323L)
  - 0 - 8 MHz (ATmega323)

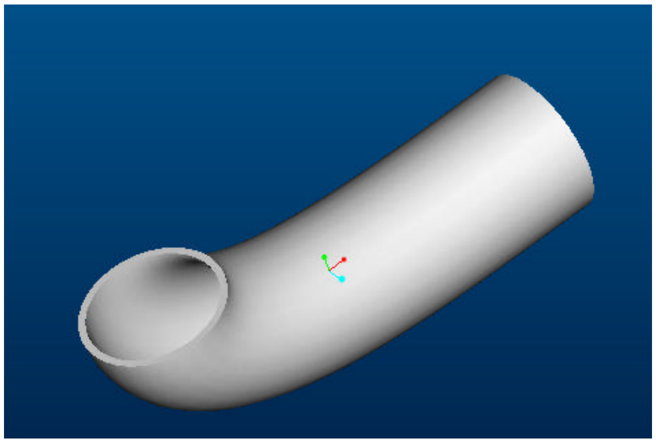
Power Supply

A pack of 8 batteries of Nickel cadmium each of 1.2 volts is used to power the processor. The power supplied to the servos is used from the same batteries but only 5 of the batteries are used. The power to the sensors is supplied from the board.

Body

The bottom of the platform is made funnel shaped with a pipe at the centre thus using gravity feed system for positioning the ball for the arm to pick up. The Fig 1 and 2 shows the bottom funnel and the pipe.

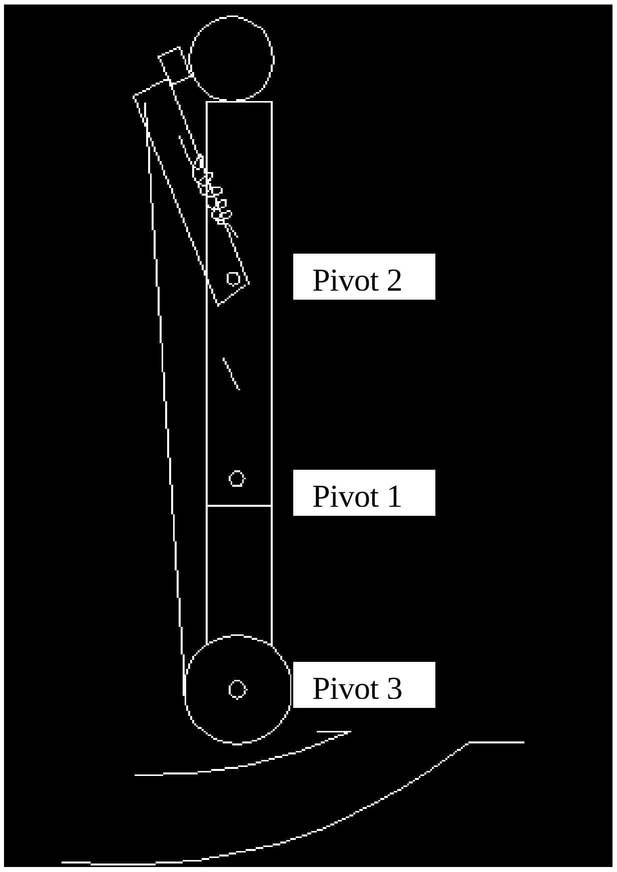




The rest of the body consist of providing a space of 8” by 8” and on the top is mounted the electronics. The body is supported by aluminum frame, which stand on two tires driven by the servos and one small free wheel at the front

Mechanical Design and Actuation

Arm Design



The Fig.1 shows the arm for picking and throwing the ball. The working cycle of picking and throwing the ball is as follows: The hammer rotates about the pivot 2 due to the pulley action about the pivot 3, the rotation of the pivot is against the spring force. Then the whole of the arm rotates along the Pivot 1 to lift the ball from the pipe. After lifting the ball up the servo on the pulley rewinds. Next the hammer action takes place as the lock on it is removed and the ball shoots on the other side of the table.

Sensors

The robot is equipped with four sensors:

1. IR Sensor for avoiding the robot from falling it off the table as it moves on the table.
2. A limit switch for limiting the hammer rotation
3. Phototransistor to detect the ball before the arm will actuate to throw the ball
4. A flex sensor, which senses if the ball is hit back on the robot and turns the buzzer ON.

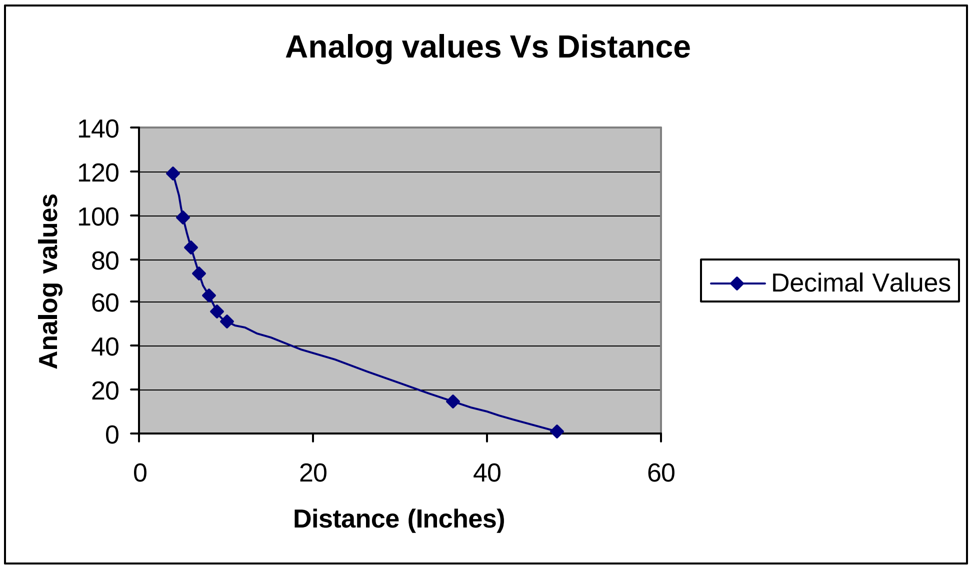
All the above sensors are run my by the Atmega 323 microcontroller.

IR Sensors

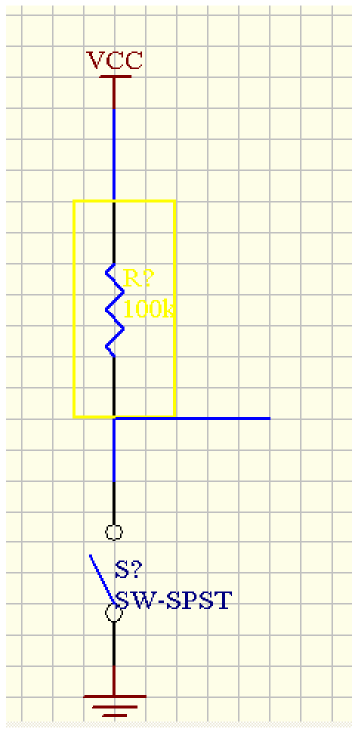
The Near Infrared Proximity sensors are sensitive in the range just below the visible light, often around 880 nm wavelengths. The IR sensors consist of two GP2D12 sharp sensors, one on each side of the robot to detect the end of the table.

The Fig. 1 gives the graph of the decimal value of the corresponding Analog Output Voltage Vs Distance to Reflective object. It can be seen from the graph that for a distance of about 10 to 12 inches the Analog Output Voltage is around 1 Volt (decimal value 51) whereas for a distance above 36 CMS the output voltage is less then 0.4 volts (decimal value 15). This difference in the output voltage is used to detect the end of the table. When the Robot reaches one end of the table the direction of the motors is reversed.

Distance in INCHES	Decimal Values	Binary Output
4	119	1110111
5	99	1100011
6	85	1010101
7	73	1001001
8	63	111111
9	56	111000
10	51	110011
36	15	1111
48	1	1



Limit Switch

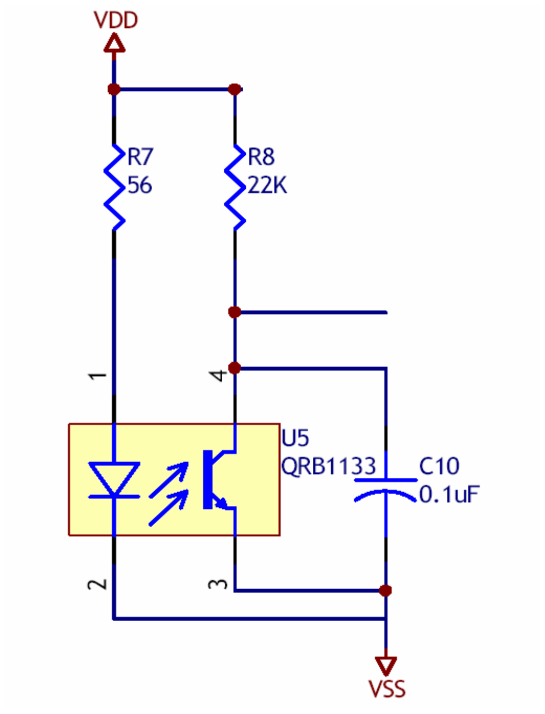


The fig. Below shows the ckt. for the limit switch. A resistance of 100k is added in series. As shown the signal is normally high. Only after the limit switch is triggered the signal reaches Low.

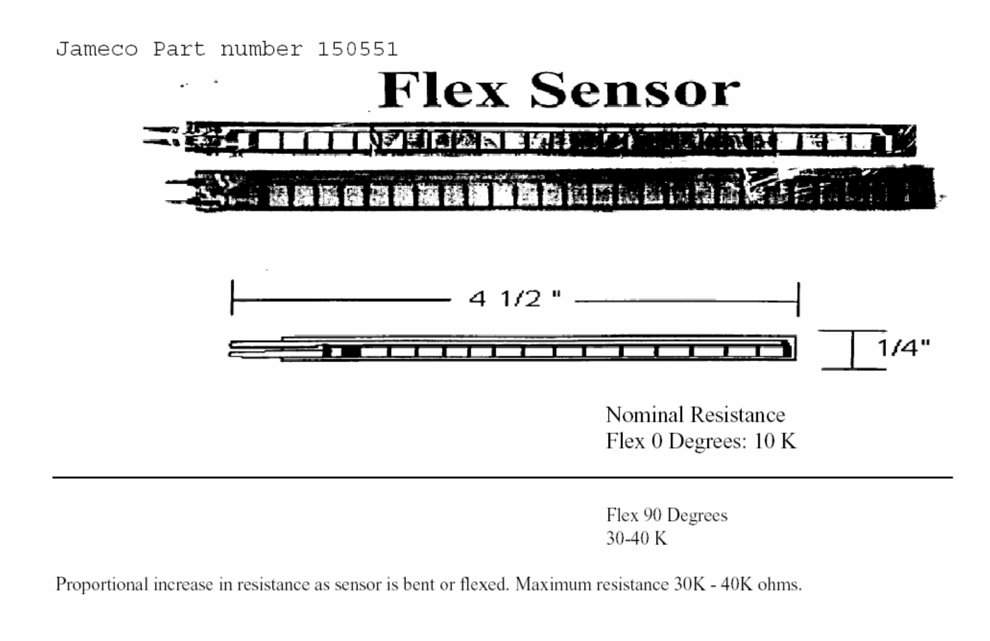
Phototransistors

The QRB1133 consists of an Infrared emitting diode and an NPN silicon Phototransistor mounted side by side. The Phototransistor responds to radiation from the emitting diode only when a reflective object passes within its field of view. I am using this sensor to determine whether the ping-pong ball is in right position for the arm to lift.

The circuit below is used from MarkIII robot kit for the phototransistor

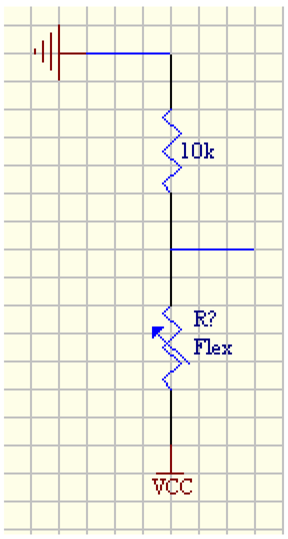


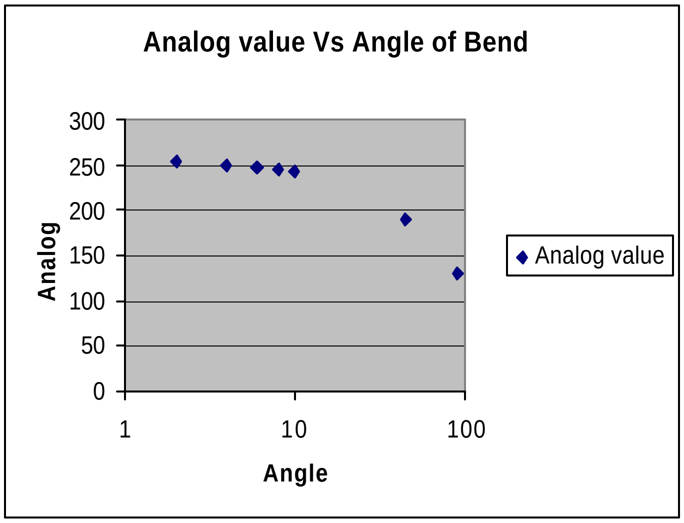
Flex Sensor


Jameco Part number 150551
Flex Sensor
4 1/2 "
1/4"
Nominal Resistance
Flex 0 Degrees: 10 K
Flex 90 Degrees
30-40 K
Proportional increase in resistance as sensor is bent or flexed. Maximum resistance 30K - 40K ohms.

The resistance of the sensor increases as it is bend on one side. The resistance varies from 10k to 40k.

The Fig. Below shows the ckt. For the flex sensor. A resistance of 10k is used in series with the sensor.
It can be seen that the analog value changes from 255 to 130 as the sensor is bent from 0 to 90 degrees.





Conclusion

The Ping-pong player works as expected. It shoots the ball on the other side of the table randomly as it is moving on the table. I enjoy playing with it. However the overall design is not very sturdy, thinking of it as the first prototype the next improvement would be to make it sturdier. Future improvements could be adding a vision sensor so that it can actually locate the ball.

Acknowledgements

My sincere thanks to Dr. A.A.Arroyo, Uriel Rodriguez and Jason Plew for their guidance throughout the project work.
My special thanks to Anirban Dutta for his help in designing the Mechanical System and Amit jayakaran for his help in the code for generating the extra PWM’s.

Sources

- Progressive resources LLC : AVRmega DEV board
- Jameco : Flex sensor
- MarkIII : IR sensors, Phototransistors, Servos
- Radioshack : Buzzer, Battery, Resistors, capacitors

Appendix A

Code:

PingPong.c:

```
/* Program for Ping Pong Player */
/* Created by sanjay solanki */
/* October 6, 2002 */
/* Revised December 05, 2003 */

#include <io.h>
#include <interrupt.h>
#include <math.h>
#include <sig-avr.h>

#define SERVO_ARMBALLPICKUP 172
#define SERVO_ARMBOTTOM 110
#define SERVO_ARMTOP 0
#define SERVO_PULLEYWIND 150
#define SERVO_PULLEYUNWIND 95
#define SERVO_PULLEYSTOP 102
#define SERVO_PULLEYLOCK 95
#define SERVO_PULLEYUNLOCK 180

typedef unsigned short u16;
typedef volatile unsigned char u08;
```

```
#include "tiremotor.h"

u08 pulley, arm, pulleylock;
u08 temp_pulley,temp1_pulley;
u08 m=48,n=98;

void delay(u16 delay_time) {
    do {
      u08 i=0;
      do {
      asm volatile("nop\n\t"
          "nop\n\t"
          "nop\n\t"
          "nop\n\t"
          ::);
      } while(--i);
    } while(--delay_time);
}

u08 ADC_getreading(u08 channel)
 {
       u08 temp_valueH;

       outp((1<<REFS0)|(1<<REFS1)|(1<<ADLAR),
ADMUX);    //use 4.95V as reference voltage

       ADMUX=ADMUX & 0xF8;
       ADMUX=ADMUX | channel;
	if (channel==4)
	{
		outp(4+192,ADMUX);
		sbi(ADMUX,ADLAR);	/* result is left
adjusted */
	}


	if (channel==3)
	{
		outp(3+192,ADMUX);
		sbi(ADMUX,ADLAR);	/* result is left
adjusted */
	}


	sbi(ADCSR, ADSC);

	loop_until_bit_is_set(ADCSR, ADIF);
	//wait till conversion is complete

	temp_valueH = inp(ADCH);

	sbi(ADCSR, ADIF);

	ADMUX=0;
```

```
        return temp_valueH;

}



void flex(void)
{
        if(ADC_getreading(4)<245)
                {

        MOTOR_speed(SERVO_STOP,SERVO_STOP)
;

                temp_pulley = pulley;
                pulley = SERVO_PULLEYSTOP;
                cbi(PORTB,7);
                delay(0x2FF);
                sbi(PORTB,7);
                delay(0x2FF);
                cbi(PORTB,7);
                delay(0x4FF);
                sbi(PORTB,7);
                delay(0x2FF);
                cbi(PORTB,7);
                delay(0x2FF);
                sbi(PORTB,7);
                pulley = temp_pulley;
                }
```

```
}

void obsta(void)
{

   if(ADC_getreading(1)<23)
                  {

      MOTOR_speed(SERVO_STOP,SERVO_STOP)
;
                  delay(0x2);
                  m = 48;   //LEFT MOTOR
FORWARD SPEED
                  n = 98;  //RIGHT MOTOR
FORWARD SPEED

             }

   if(ADC_getreading(0)<23)
             {

      MOTOR_speed(SERVO_STOP,SERVO_STOP)
;
                  delay(0x2);
                  m = 84;
                  n = 35;
             }
```

```
    MOTOR_speed(m,n);
}

void obstadelay(u16 delay_time) {
    do {
      u08 i=0;
      do {
      obsta();
      flex();
      } while(--i);
    } while(--delay_time);
}



SIGNAL (SIG_OUTPUT_COMPARE0)
{
       cli();

       PORTC=PORTC | 7;

       outp(0,TCNT0); //start value of timer variable
       outp(3,TCCR0); //prescale 64
       while(TCNT0<=45)
       {}

       outp(0,TCNT0); //start value of timer variable
```

```
        while (TCNT0<=180)
        {
                if (TCNT0>pulley)
                        PORTC=PORTC & 0xFE;

                if (TCNT0>arm)
                        PORTC=PORTC & 0xFD;

                if (TCNT0>pulleylock)
                        PORTC=PORTC & 0xFB;

        }

        PORTC=PORTC & 0xF8;

        outp(5,TCCR0); //prescale 1024
        outp(0,TCNT0);//Reinitializr value of timer 0

        outp((1<<OCIE0),TIMSK); // Enable interrupt
of timer 0
        sei();

}

u08 throw(void)
{
```

```
while(bit_is_set(PINA,2))
{
    obstadelay(0x1);
    pulley = SERVO_PULLEYWIND;
}


pulleylock = SERVO_PULLEYLOCK;
pulley = SERVO_PULLEYSTOP;


arm = SERVO_ARMBALLPICKUP;
obstadelay(0xFA);


while(ADC_getreading(3) > 250)
{
obstadelay(0x1);
}


arm = SERVO_ARMBOTTOM;
obstadelay(0x3);


pulley = SERVO_PULLEYSTOP;
```

```
            arm = SERVO_ARMTOP;
            obstadelay(0x3);


            pulley = SERVO_PULLEYUNWIND;
            obstadelay(0x8F);

            pulley = SERVO_PULLEYSTOP;

            pulleylock = SERVO_PULLEYUNLOCK;
            obstadelay(0xF);


            return (1);

}

void ADC_init(void)
{
      DDRA=0;
      outp((1<<ADEN) | (1<<ADPS2) | (ADPS1),
ADCSR); //Initialize to use 8bit resolution for all
channels
}
```

```
int main(void)
{

        outp(0xFF,DDRC);
        outp(0x00,DDRA);
        outp(0xFF,DDRB);
        sbi(PORTB,7);
        PORTC=7;



        pulley = 102;
        pulleylock = SERVO_PULLEYUNLOCK;
        Motor_init();
        ADC_init();






        sei(); //Set global interrupt enable

        arm = SERVO_ARMTOP;
        delay(0xFFF);

    u16 cnt;
        for( ; ;)
```

```
		{
			throw();
		}

}
```

tiremotor.h

```
/* Program for running a hacked servo motor */
/* Created by sanjay solanki */
/* October 6, 2002 */

#include <io.h>
#include <math.h>

#define SERVO_STOP 68
#define SERVO_RIGHT 50
#define SERVO_LEFT 100
#define FULL_RIGHT 100
#define FULL_LEFT -100
#define LEFT_MOTOR 0
#define RIGHT_MOTOR 1
#define K 10

void Motor_init(void)
{
	OCR1AL = SERVO_STOP;
	OCR1BL = SERVO_STOP;
	TCCR1A = (1 << COM1A1) | (1 << COM1B1) |
(1 << PWM10) | (1 << PWM11);
	TCCR1B = (1 << CS11) | (1 << CS10);
	sbi(DDRD, PD4);
	sbi(DDRD, PD5);

	//Init timer 0
	outp((1<<OCIE0),TIMSK); // Enable interrupt
of timer 0
	outp(0,TCNT0);//Initial value of timer 0
	outp(5,TCCR0);//Prescale of 1024
	outp(0xBC,OCR0);//Decimal value 94 - equal to
16.04ms - 94*1024/6M
	sbi(DDRC,0);
	sbi(DDRC,1);
	sbi(DDRC,2);
}

void SERVO_speed(int mot_num, int speed)
{
	if(mot_num == 0)
		OCR1BL = speed;



	else if(mot_num == 1)
		OCR1AL = speed;
	return;
```

```
}

void MOTOR_speed(int leftspeed, int rightspeed)
{
	SERVO_speed(LEFT_MOTOR, leftspeed);
	SERVO_speed(RIGHT_MOTOR, rightspeed);
}
```