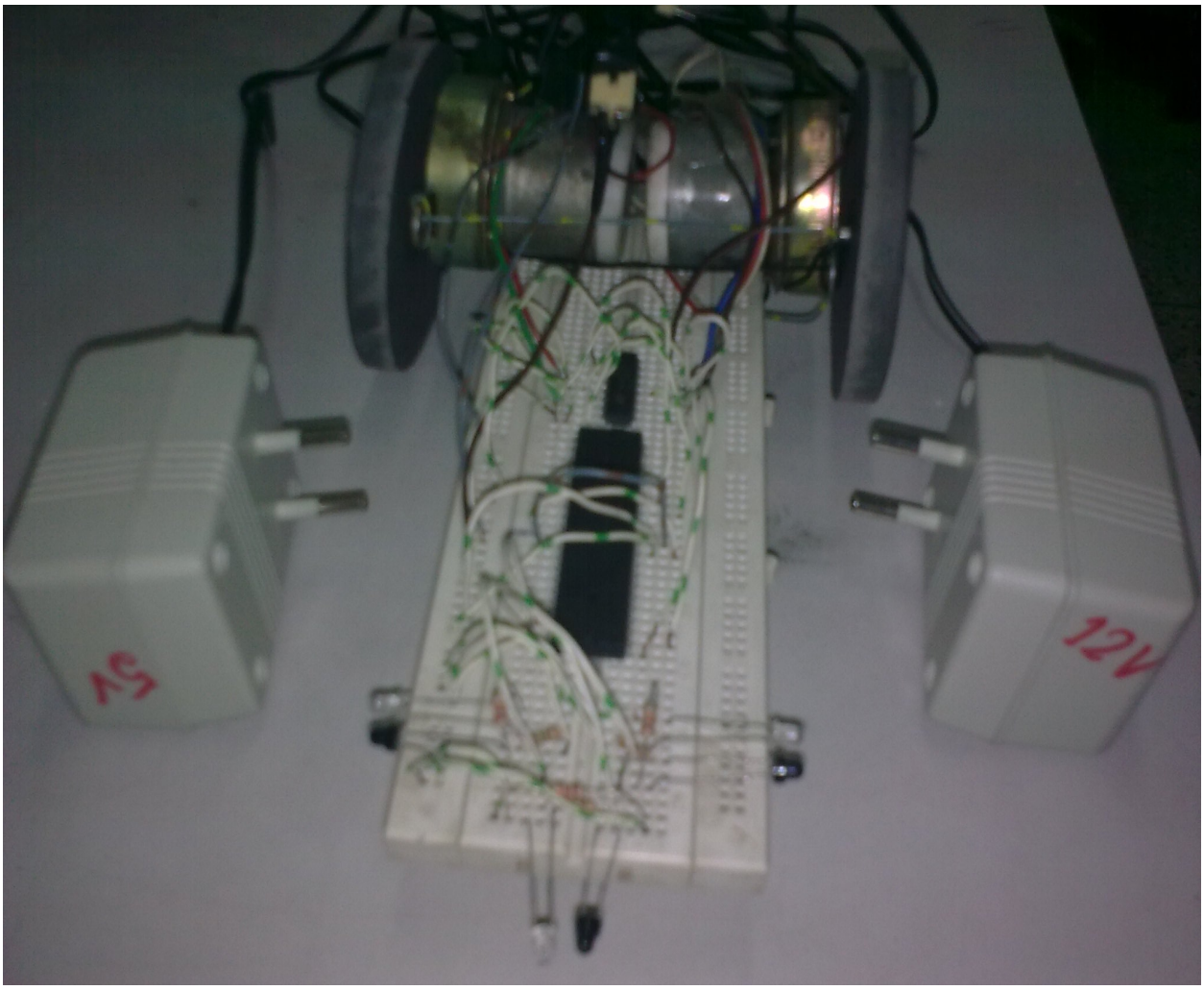
Line Follower ROBOT



Submitted To:

Shajib Kumar Mistry, Lecturer CSEDU

Submitted By:

Md Mushfekur Rahman – Roll # 1501

Department of Computer Science and Engineering

University of Dhaka

8th May, 2012

Table of Content

Introduction

A **robot** is a mechanical intelligent agent which can perform tasks on its own, or with guidance. The term **robot** can also apply to a virtual agent. In practice it is usually an electro mechanical machine which is guided by computer and electronic programming.

Robots can be autonomous or semi autonomous and come in those two basic types: those which are used for research into human like systems, as well as those into more defined and specific roles, and helper robots which are used to make or move things or perform menial or dangerous tasks. Another common characteristic is that, by its appearance or movements, a robot often conveys a sense that it has intent or agency of its own.

The **robot** that has been built in this project is an **Line Follower Robot**. The main purpose of this **robot** is: to follow along a line on the ground, climb onto slopes upto 15 degrees, navigate to all possible directions.

This is a manual containing everything that needs to be known in building an **Line Follower Robot**. This manual contains the Hardware and Software Requirements, along with the estimated price for building the **robot**. Explanation about the algorithm used, along with the circuit diagram(s) and its working procedure. Detailed Explanation of the IC's used, along with the explanation of the source code.

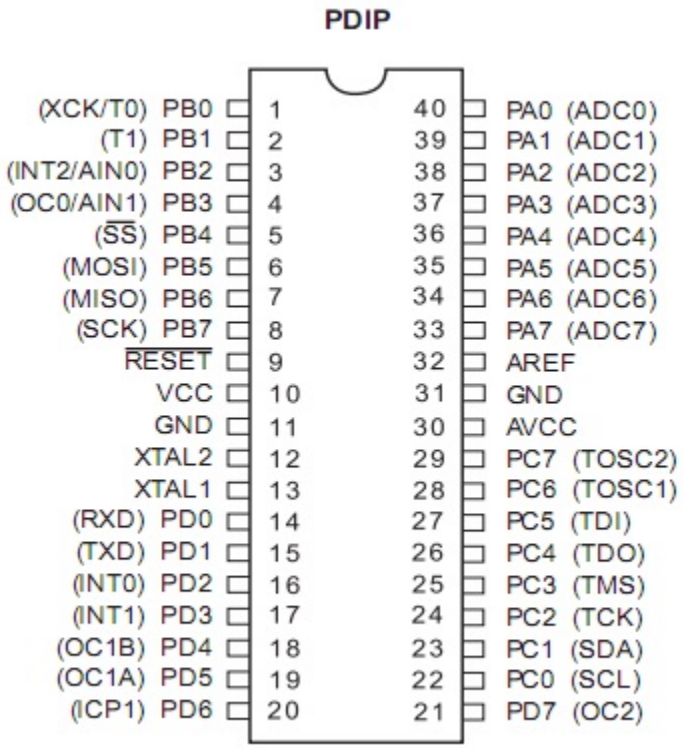
Hardware Requirements with Price Analysis

Component	Quantity	Price (TK)
Atmega32 Microprocessor Chip	1	750
L293D IC Dual H-Bridge Motor Driver	1	50
DC Gear Motor	2	400 * 2 = 800
Infra-red sensors(transmitter + receiver)	3	25 * 3 = 75
15 ohm resistor	3	5 * 3 = 15
22 kohm resistor	3	5 * 3 = 15
12V adapter	1	80
5V adapter	1	80
Standard Breadboard	1	350
Grand Total		2215

Hardware Details

- Atmega32 Microprocessor Chip:

  Pin Configuration:



  Overview:

  The ATmega32 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega32 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

  Pin Description:

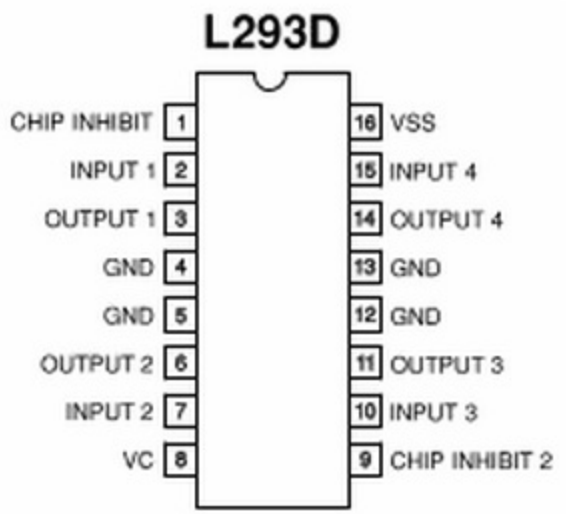
  - ✓ VCC — Digital supply voltage.
  - ✓ GND — Ground.

- ✓ Port A (PA7…PA0) Port A serves as the analog inputs to the A/D Converter. Port A also serves as an 8-bit bi-directional I/O port, if the A/D Converter is not used. Port pins can provide internal pull-up resistors (selected for each bit). The Port A output buffers have symmetrical drive characteristics with both high sink and source capability. When pins PA0 to PA7 are used as inputs and are externally pulled low, they will source current if the internal pull-up resistors are activated. The Port A pins are tri-stated when a reset condition becomes active, even if the clock is not running.
- ✓ Port B (PB7…PB0) Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.
- ✓ Port C (PC7…PC0) Port C is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port C output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running. If the JTAG interface is enabled, the pull-up resistors on pins PC5(TDI), PC3(TMS) and PC2(TCK) will be activated even if a reset occurs. The TD0 pin is tri-stated unless TAP states that shift out data are entered. Port C also serves the functions of the JTAG interface.

✓ Port D (PD7… PD0)	Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.
✓ RESET	Reset Input. A low level on this pin for longer than the minimum pulse length will generate a reset, even if the clock is not running.
✓ XTAL1	Input to the inverting Oscillator amplifier and input to the internal clock operating circuit.
✓ XTAL2	Output from the inverting Oscillator amplifier.
✓ AVCC	AVCC is the supply voltage pin for Port A and the A/D Converter. It should be externally connected to VCC, even if the ADC is not used. If the ADC is used, it should be connected to VCC through a low-pass filter.
✓ AREF	AREF is the analog reference pin for the A/D Converter.

- L293D IC Dual H-Bridge Motor Driver:

  Pin Configuration:



Overview:

L293D is a dual H-Bridge motor driver, so with one IC we can interface two DC motors which can be controlled in both clockwise and counter clockwise direction and if you have motor with fix direction of motion. L293D has output current of 600mA and peak output current of 1.2A per channel.

Pin Description:

- ✓ Chip Inhibit 1 — It is used to enable Input 1, 2 and Output 1, 2.
- ✓ Input 1 — It is used to take the first bit as input from the Atmega32.
- ✓ Output 1 — It is connected to the positive end of the first motor.
- ✓ GND — Ground.
- ✓ Output 2 — It is connected to the negative end of the first motor.
- ✓ Input 2 — It is used to take the second bit as input from the Atmega32.
- ✓ VC — 12V supply voltage.
- ✓ VSS — 5V supply voltage.
- ✓ Input 3 — It is used to take the third bit as input from the Atmega32.

✓	Output 3	It is connected to the positive end of the second motor.
✓	Input 4	It is used to take the fourth bit as input from the Atmega32.
✓	Output 4	It is connected to the negative end of the second motor.
✓	Chip Inhibit 2	It is used to enable Input 3, 4 and Output 3, 4.

- DC Gear Motor:

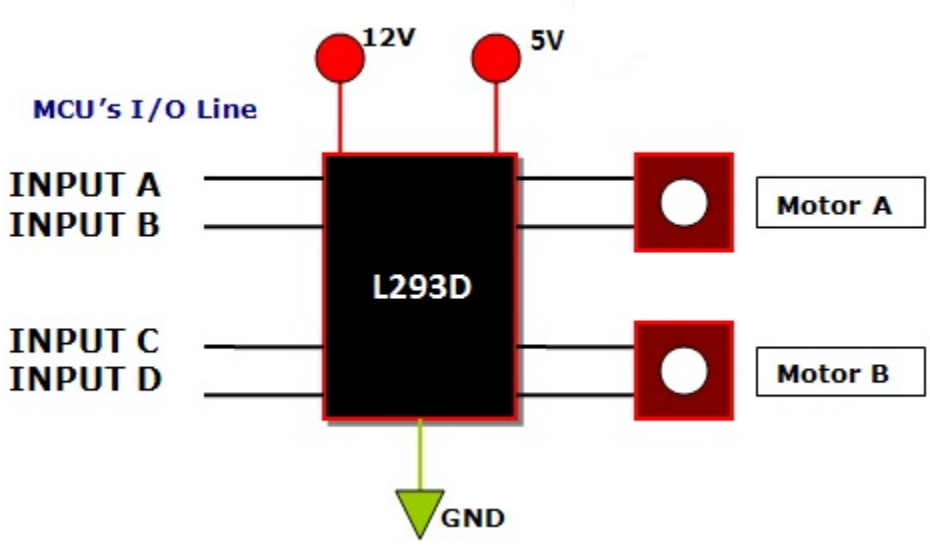
  Overview:

  DC Motor is the most commonly used actuator in robotics applications. An actuator is a device used to produce motion. DC motors are used to drive wheels in order make the robot move. They are also used to power grippers, arms and weapons (fighting robots). Normally a DC Motor with inbuilt gear box is preferred. They normally consume 400ma to 1000ma current and works off 12v DC Supply. They are available in many different RPM (Revolution per Minutes). Example: 60 RPM, 100 RPM and 200 RPM. One such motor is shown below.

Control:

DC Motor Rotates in one direction when you apply power to its terminals. When you reverse the polarity of the supply it will rotate in other direction. As a DC Motor requires current in the range of 400ma to 1000ma we cannot supply them directly from the MCUs I/O PINs. We need some kind of driver circuit that can deliver more current to the motors. Also a MCU generally works off 5v supply but normal motors require a 12v (or 24v) supply. This circuit which is used to control a Motor from MCUs I/O line is called **H-BRIDGE** circuit. Many easy to use H-Bridge ICs are available, like the L293D. One L293D IC has two H-BRIDGE circuits. So it can control 2 DC Motors.

An Example of driving motors by using L293D is given below.



The Motor is controlled in the following way by MCU's I/O lines:

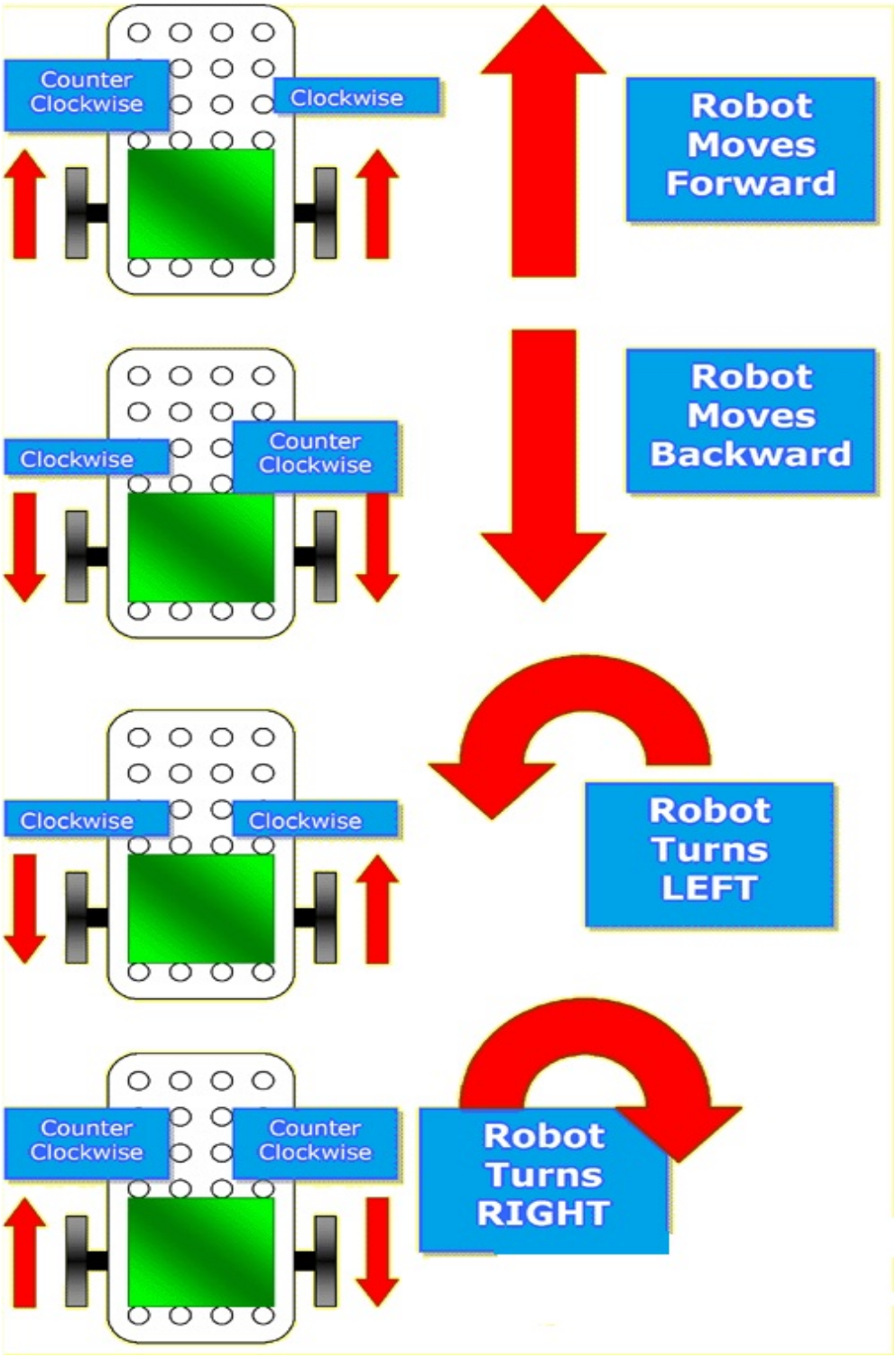
Motor A:

Input A	Input B	Description
0	0	Motor Stops or Breaks
0	1	Motor Runs Anti-Clockwise
1	0	Motors Runs Clockwise
1	1	Motor Stops or Breaks

Motor B:

Input C	Input D	Description
0	0	Motor Stops or Breaks
0	1	Motor Runs Anti-Clockwise
1	0	Motors Runs Clockwise
1	1	Motor Stops or Breaks

The following diagram demonstrates the motion of the robot:



- Infra-red Sensors:

  Overview:

An infrared sensor is an electronic device that emits and/or detects infrared radiation in order to sense some aspect of its surroundings. Infrared sensors can measure the heat of an object, as well as detect motion. The IR Sensor element is made up of an IR Tx, IR Rx and few resistors. The schematic is given in the circuit diagram section. One such IR transmitter and receiver is shown below.

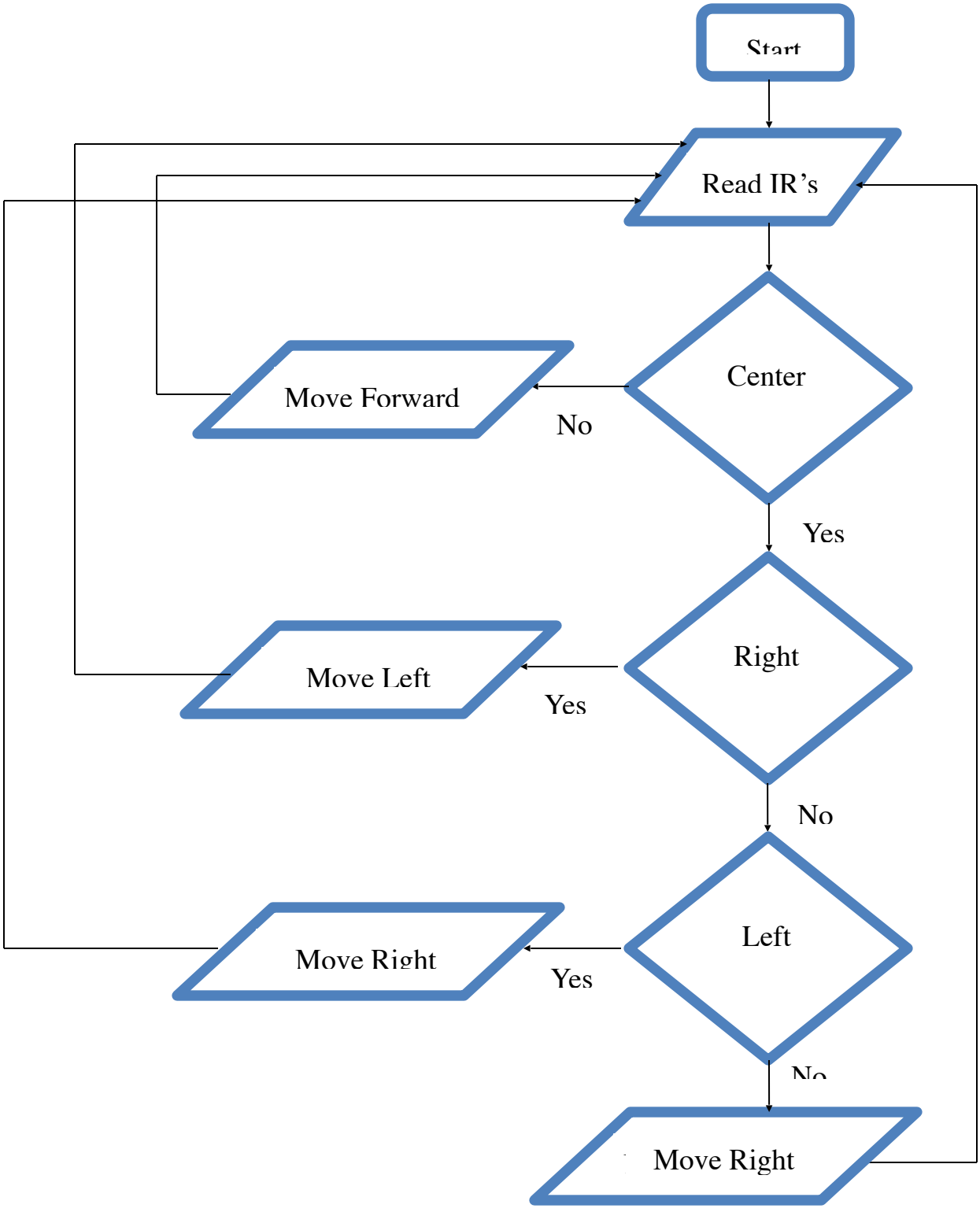
Software Requirements

- WinAVR

- AVR Studio
- AVR Library
- LT48XP_790

Flow Diagram

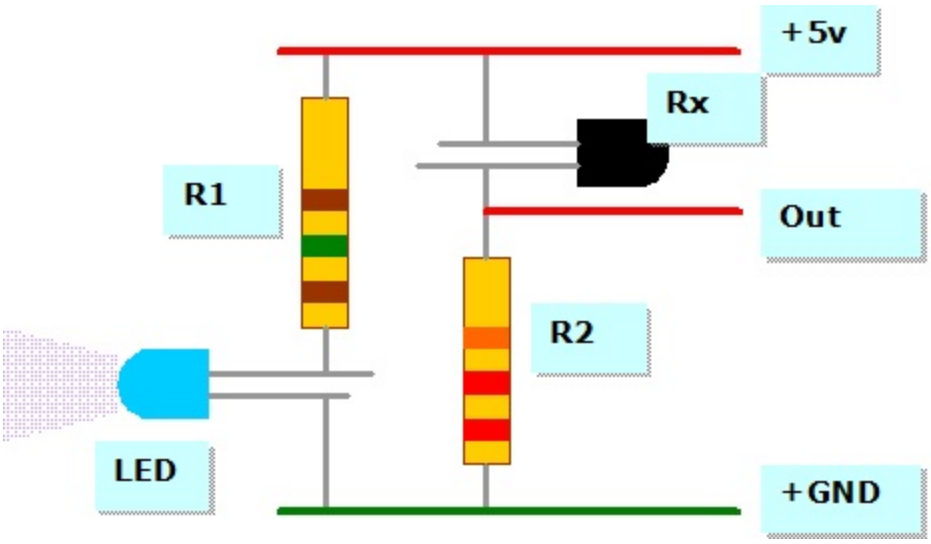
The following flow diagram illustrates the basic structure of the algorithm, on the basis of which the robot can perform appropriate tasks:



Circuit Diagram(s)

- Infra-red Sensor Circuit:

  Diagram:

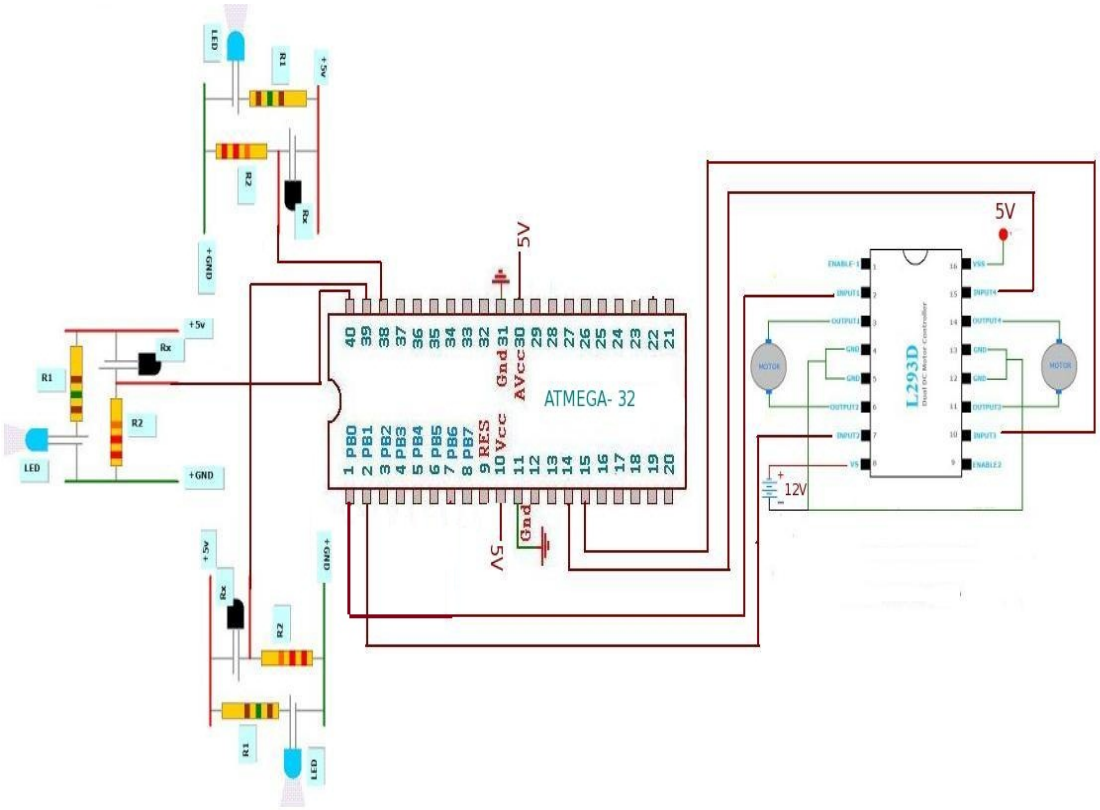


Description:

- LED - is the transmitter.
- R1 - is the 150 ohm resistor.
- R2 - is the 22kohm resistor.
- Rx – is the receiver.
- Out – is the output of the infra-red sensor, which is fed to Atmega32.
- +5V – is the 5V supply voltage.
- +GND – is the ground.

- Main Circuit:

  Diagram:



Description:

- Output from the 3 IR's are connected to the ADC input pins of Atmega32.
- Center, Left and Right IR output is connected to ADC0, ADC1 and ADC2 respectively.
- +5V is fed to both VCC and AVCC with pin numbers 10 and 30 respectively.
- Pin number 11 and 31 are grounded.
- PB0 and PB1 of Atmega32 are connected to Input1 and Input 2 of L293D respectively.
- PD0 and PD1 of Atmega32 are connected to Input3 and Input 4 of L293D respectively.
- VC, pin number 8 of L293D is fed +12V supply voltage.
- VSS, pin number 16 of L293D is fed +5V supply voltage.
- The GND pins L293D are grounded.

- The positive and negative end of the left DC gear motor is connected to pin number 3 and 6 respectively.
- The positive and negative end of the right DC gear motor is connected to pin number 14 and 11 respectively.

Source Code

- Introduction:

  The source code has been divided into two files: robot.c and robotHeader.h.

robotHeader.h is a custom defined header file containing the function details.

robot.c contains the main program.

- robotHeader.h:

```
//function to read the value of ADC
unsigned int readADC(unsigned int ch) {
   //Select ADC Channel ch must be 0-7
   ch = ch & 0b00000111;
   ADMUX &= 0b11100000;
   ADMUX |= ch;
   //Start Single conversion
   ADCSRA |= (1<<ADSC);
   //Wait for conversion to complete
   while(!(ADCSRA & (1<<ADIF)));
   //Clear ADIF by writing one to it
   ADCSRA |= (1<<ADIF);
   return (ADC);
}
void initMotor() {
      DDRB = 0xFF;     //port B initialized as output
      DDRD = 0xFF;     //port D initialized as output
      DDRC = 0xFF;     //port C initialized as output
      PORTB = 0;
      PORTD = 0;
      PORTC = 0;
}
void initADC() {
   ADMUX=(1<<REFS0); // For Aref=AVcc;
```

```
   ADCSRA=(1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|
(1<<ADPS0);             //Rrescalar div factor =128

}

void forwardMotorAB() {

      PORTD = 2; //right motor clockwise

      PORTB = 1; //left motor anticlockwise

}

void leftMotorAB() {

      PORTB = PORTD = 2;    //left and right motor clockwise

}

void rightMotorAB() {

      PORTD = PORTB = 1;    //left and right motor anticlockwise

}

void stopMotorAB() {

      PORTD = PORTB = 0; //both motor stops

}

void backwardMotorAB() {

      PORTD = 1; //right motor anticlockwise

      PORTB = 2; //left motor clockwise

}
```

- robot.c:

```
#include <avr/io.h>

#include <avr/delay.h>

#include "robotHeader.h"

#define CTHRES 475    //center threshold limit

#define RTHRES 575    //right threshold limit

#define LTHRES 575    //left threshold limit
```

```
int main() {
      initMotor();
      initADC();
      unsigned int centerIR, rightIR, leftIR;
      forwardMotorAB();          //initially move forward
      while(1) {
            centerIR = readADC(0);      //Check center Sensor
            rightIR = readADC(1);       //Check right Sensor
            leftIR = readADC(2);        //Check left Sensor
            if(centerIR > CTHRES) {
                  _delay_ms(10000);
                  rightIR = readADC(1);
                  leftIR = readADC(2);
                  //if both blocked
                  if(rightIR > RTHRES && leftIR > LTHRES) {
                        backwardMotorAB();
                        while(rightIR > RTHRES || leftIR > LTHRES){
                        //Check right Sensor
                              rightIR = readADC(1);
                              //Check left Sensor
                              leftIR = readADC(2);
                        _delay_ms(5000);
                        }
                        rightIR = readADC(1);
                        leftIR = readADC(0);
                        if(rightIR < RTHRES)
                              rightMotorAB();
                        else if(leftIR < LTHRES)
```

```
            leftMotorAB();
      }
      if(rightIR < RTHRES) {           //if right clear
            rightMotorAB();
            while(centerIR > CTHRES) {
                  centerIR = readADC(0);
                  _delay_ms(5000);
            }
            forwardMotorAB();
      }
      else if (rightIR > RTHRES){//if right blocked
            leftMotorAB();
            while(centerIR > CTHRES) {
                  centerIR = readADC(0);
                  _delay_ms(5000);
            }
            forwardMotorAB();
      }
      if(leftIR < LTHRES) { //if left clear
            leftMotorAB();
            while(centerIR > CTHRES) {
                  centerIR = readADC(0);
                  _delay_ms(5000);
            }
            forwardMotorAB();
      }
      else if(leftIR > LTHRES) {  //if left blocked
```

```
                    rightMotorAB();
                    while(centerIR > CTHRES) {
                        centerIR = readADC(0);
                        _delay_ms(5000);
                    }
                    forwardMotorAB();
                }
            }
            else {
                if( (rightIR > RTHRES && leftIR > LTHRES) ||
(rightIR < RTHRES && leftIR < LTHRES) )
                    forwardMotorAB();
                else {
            //check if left is blocked and right is clear
                    if(leftIR > LTHRES && rightIR < RTHRES) {
                    rightMotorAB();
                        while(leftIR > LTHRES) {
                            leftIR = readADC(2);
                        _delay_ms(5000);
                        }
                        forwardMotorAB();
                    }
                    else {
                //check if right is blocked and left is clear
                    if(rightIR > RTHRES && leftIR < LTHRES) {
                        leftMotorAB();
                            while(rightIR > RTHRES) {
                                righter = readADC(1);
                                _delay_ms(5000);
```

```
                                        }
                                        forwardMotorAB();
                                    }
                                }
                            }
                        }
                    }
                return 0;
}
```

Future Enhancements

- The existing robot will be given hands and a firm body.
- Existing infra-red sensors will be replaced by more powerful sensor devices.
- A camera will be installed in the robot.
- The camera will be used to detect the type of object in front of the robot.
- The camera along with the sensors will be added to a 360 deg revolving head.