Line Following Robot

By,
Priyank Patil
Department of Information Technology
K. J. Somaiya College of Engineering
Mumbai, India

Contents

Summary

The purpose of this document is to help you build a Line Following Robot.

Starting with an overview of the system the document would cover implementation details like circuits and algorithms, followed by some suggestions on improving the design.

The ‘Reference and Resources’ page has a list of relevant books, websites, electronic shops and commonly used parts & their prices.

Introduction

What is a line follower?

Line follower is a machine that can follow a path. The path can be visible like a black line on a white surface (or vice-versa) or it can be invisible like a magnetic field.

Why build a line follower?

Sensing a line and maneuvering the robot to stay on course, while constantly correcting wrong moves using feedback mechanism forms a simple yet effective closed loop system. As a programmer you get an opportunity to 'teach' the robot how to follow the line thus giving it a human-like property of responding to stimuli.

Practical applications of a line follower : Automated cars running on roads with embedded magnets; guidance system for industrial robots moving on shop floor etc.

Prerequisites:

Knowledge of basic digital and analog electronics.
(A course on Digital Design and Electronic Devices & Circuits would be helpful)
C Programming
Sheer interest, an innovative brain and perseverance!

Background:

I started with building a parallel port based robot which could be controlled manually by a keyboard. On the robot side was an arrangement of relays connected to parallel port pins via opto-couplers.

The next version was a true computer controlled line follower. It had sensors connected to the status pins of the parallel port. A program running on the computer polled the status register of the parallel port hundreds of times every second and sent control signals accordingly through the data pins.
The drawbacks of using a personal computer were soon clear –
It's difficult to control speed of motors
As cable length increases signal strength decreases and latency increases.
A long multi core cable for parallel data transfer is expensive.
The robot is not portable if you use a desktop PC.

The obvious next step was to build an onboard control circuit; the options – a hardwired logic circuit or a uC. Since I had no knowledge of uC at that time, I implemented a hardwired logic circuit using multiplexers. It basically mapped input from four sensors to four outputs for the motor driver according to a truth table. Though it worked fine, it could show no intelligence – like coming back on line after losing it, or doing something special when say the line ended. To get around this problem and add some cool features, using a microcontroller was the best option.

The AVR microcontroller:

> *"Atmel's AVR® microcontrollers have a RISC core running single cycle instructions and a well-defined I/O structure that limits the need for external components. Internal oscillators, timers, UART, SPI, pull-up resistors, pulse width modulation, ADC, analog comparator and watch-dog timers are some of the features you will find in AVR devices.*
>
> *AVR instructions are tuned to decrease the size of the program whether the code is written in C or Assembly. With on-chip in-system programmable Flash and EEPROM, the AVR is a perfect choice in order to optimize cost and get product to the market quickly."*
>
> -http://www.atmel.com/products/avr/

Apart form this almost all AVRs support **In System Programming** (ISP) i.e. you can reprogram it without removing it from the circuit. This comes very handy when prototyping a design or upgrading a built-up system. Also the programmer used for ISP is easier to build compared to the parallel programmer required for many old uCs. Most AVR chips also support **Boot Loaders** which take the idea of In System Programming to a new level. Features like **I^2C** bus interface make adding external devices a cakewalk. While most popular uCs require at least a few external components like crystal, caps and pull-up resistors, with AVR the number can be as low as zero!

Cost: AVR = PIC > 8051 (by 8051 I mean the 8051 family)

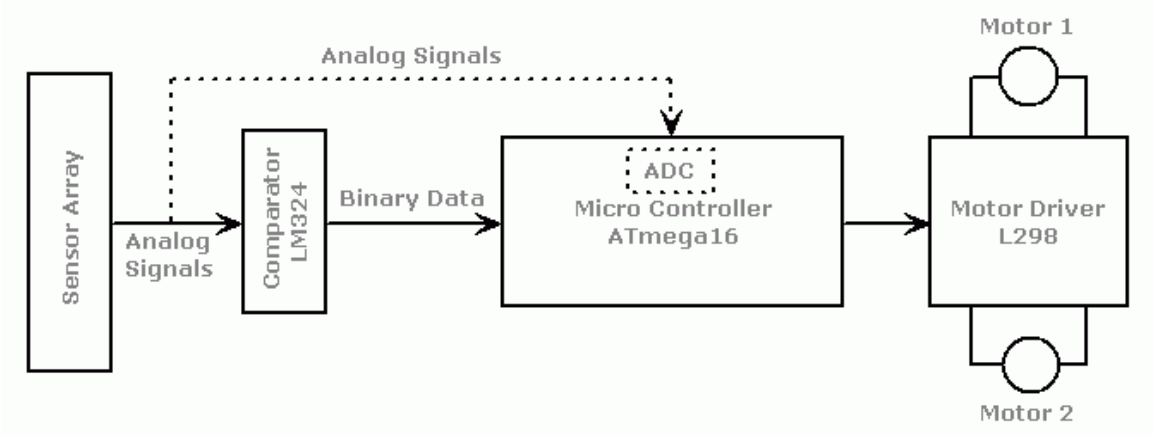
Availability: AVR = PIC <8051

Speed: AVR > PIC > 8051

Built-in Peripherals: This one is difficult to answer since all uC families offer comparable features in their different chips. For a just comparison, I would rather say that for a given price AVR = PIC > 8051.

Tools and Resources: 8051 has been around from many years now, consequently there are more tools available for working with it. Being a part of many engineering courses, there is a huge communitiy of people that can help you out with 8051; same with books and online resources. In spite of being new the AVR has a neat tool chain (See 'References and Resources'). Availability of online resources and books is fast increasing.
Here, 8051 > AVR = PIC

Overview



Block Diagram

The robot uses IR sensors to sense the line, an array of 8 IR LEDs (Tx) and sensors (Rx), facing the ground has been used in this setup. The output of the sensors is an analog signal which depends on the amount of light reflected back, this analog signal is given to the comparator to produce 0s and 1s which are then fed to the uC.

L4	L3	L2	L1	R1	R2	R3	R4

Left Center Right

Sensor Array

Starting from the center, the sensors on the left are named L1, L2, L3, L4 and those on the right are named R1, R2, R3, R4.
Let us assume that **when a sensor is on the line it reads 0 and when it is off the line it reads 1**

The uC decides the next move according to the algorithm given below which tries to position the robot such that L1 and R1 both read 0 and the rest read 1.

L4	L3	L2	L1	R1	R2	R3	R4
1	1	1	0	0	1	1	1

Left Center Right

Desired State L1=R1=0, and Rest=1

Algorithm:

1. L= leftmost sensor which reads 0; R= rightmost sensor which reads 0.
   If no sensor on Left (or Right) is 0 then L (or R) equals 0;
   Ex:

L4	L3	L2	L1	R1	R2	R3	R4
1	0	0	1	1	1	1	1

Left Center Right

Here L=3 R=0

L4	L3	L2	L1	R1	R2	R3	R4
1	1	0	0	0	0	0	0

Left Center Right

Here L=2 R=4

2. If all sensors read 1 go to step 3,
   else,
   If L>R Move Left
   If L<R Move Right
   If L=R Move Forward
   Goto step 4

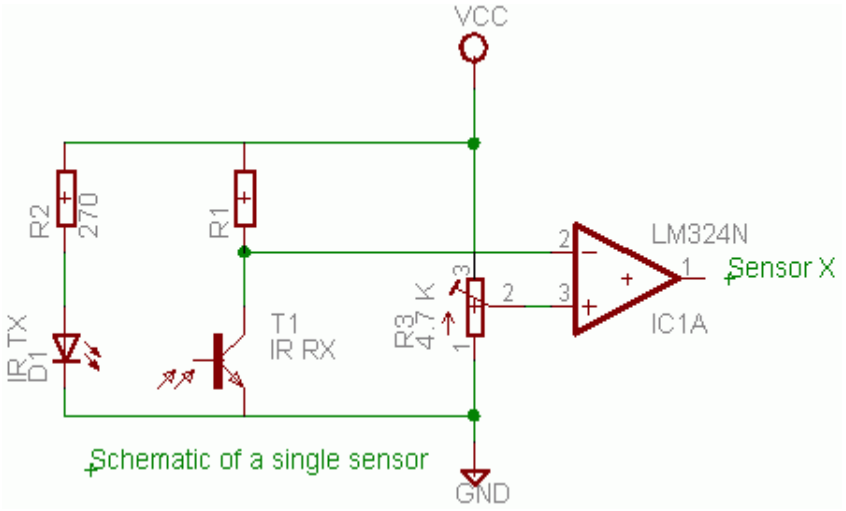
3. Move Clockwise if line was last seen on Right
   Move Counter Clockwise if line was last seen on Left
   Repeat step 3 till line is found.

4. Goto step 1.

Implementation

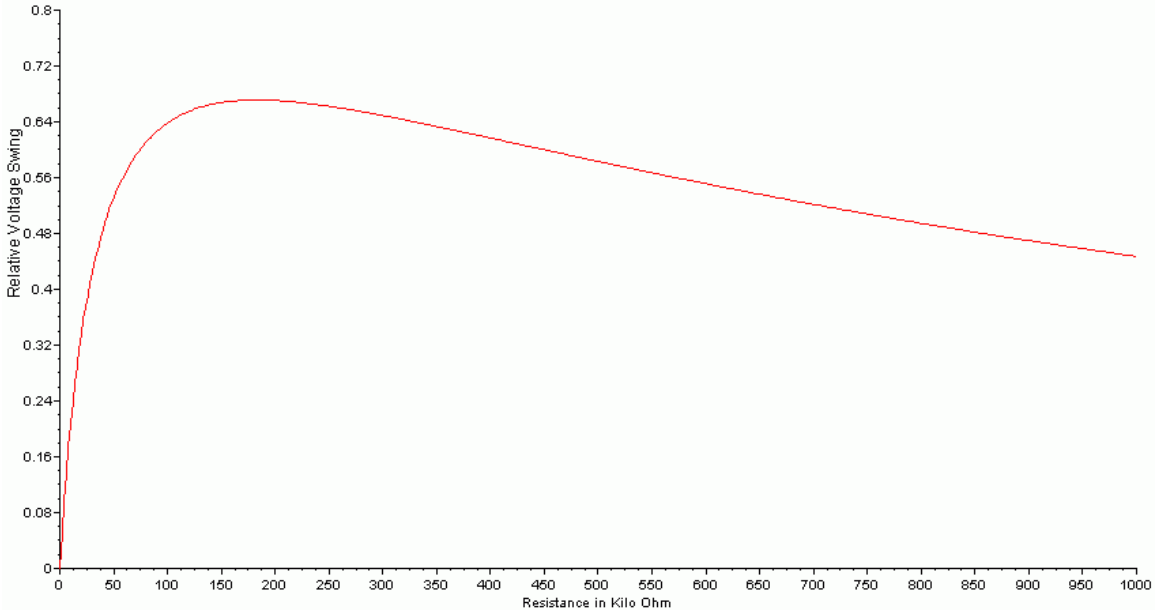
Sensor Circuit:

The resistance of the sensor decreases when IR light falls on it. A good sensor will have near zero resistance in presence of light and a very large resistance in absence of light.
We have used this property of the sensor to form a potential divider. The potential at point '2' is $R_{sensor} / (R_{sensor} + R_1)$.
Again, a good sensor circuit should give maximum change in potential at point '2' for no-light and bright-light conditions. This is especially important if you plan to use an ADC in place of the comparator



To get a good voltage swing , the value of R1 must be carefully chosen. If **R_{sensor} = a when no light falls on it and R_{sensor} = b when light falls on it**. The difference in the two potentials is:

V_{cc} * { a/(a+R1) - b/(b+R1) }

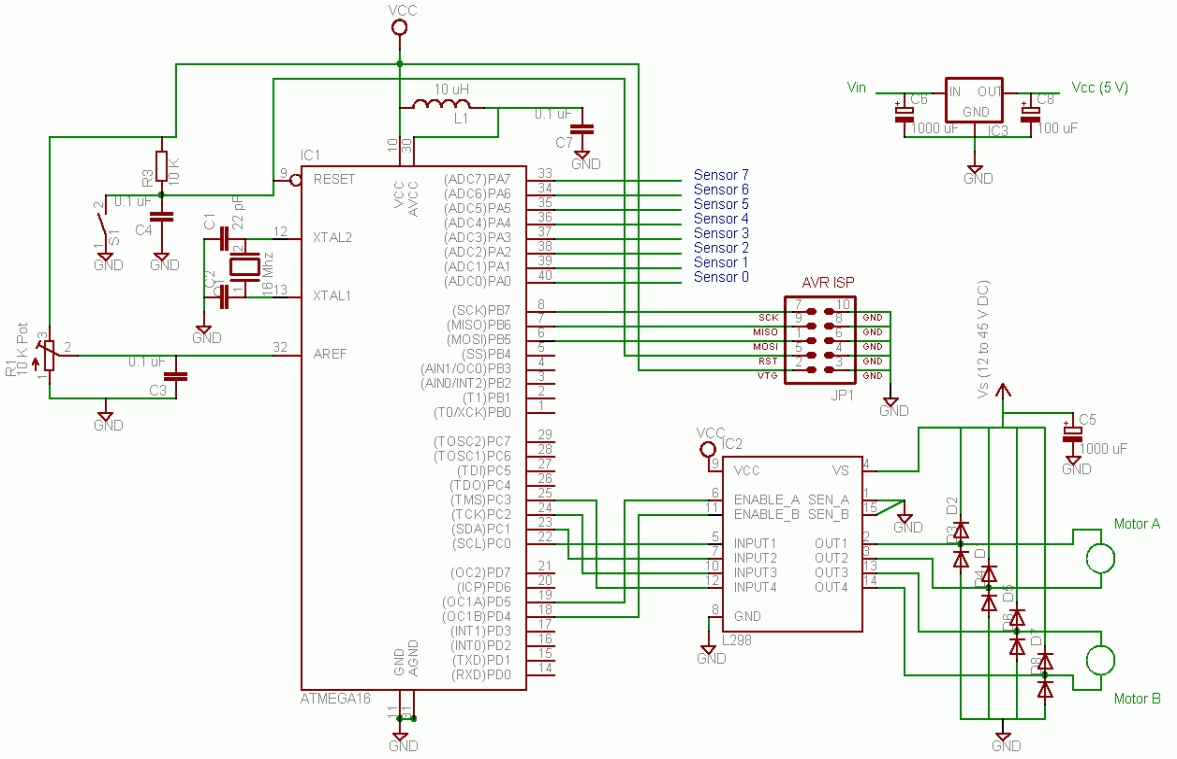


Relative voltage swing = Actual Voltage Swing / V_{cc}
= V_{cc} * { a/(a+R1) - b/(b+R1) } / V_{cc}
= a/(a+R1) - b/(b+R1)

The sensor I used had a = 930 K and b = 36 K. If we plot a curve of the voltage swing over a range of values of R1 we can see that the maximum swing is obtained at R1= 150 K (use calculus for an accurate value).
There is a catch though, with such high resistance, the current is very small and hence susceptible to be distorted by noise. The solution is to strike a balance between sensitivity and noise immunity. I chose value of R1 as 60 K. Your choice would depend on the 'a' and 'b' values of your sensor.

If you found this part confusing, use a 10K resistor straightaway, as long as you are using a comparator it won't matter much.

Motor Interface and Control Circuit:

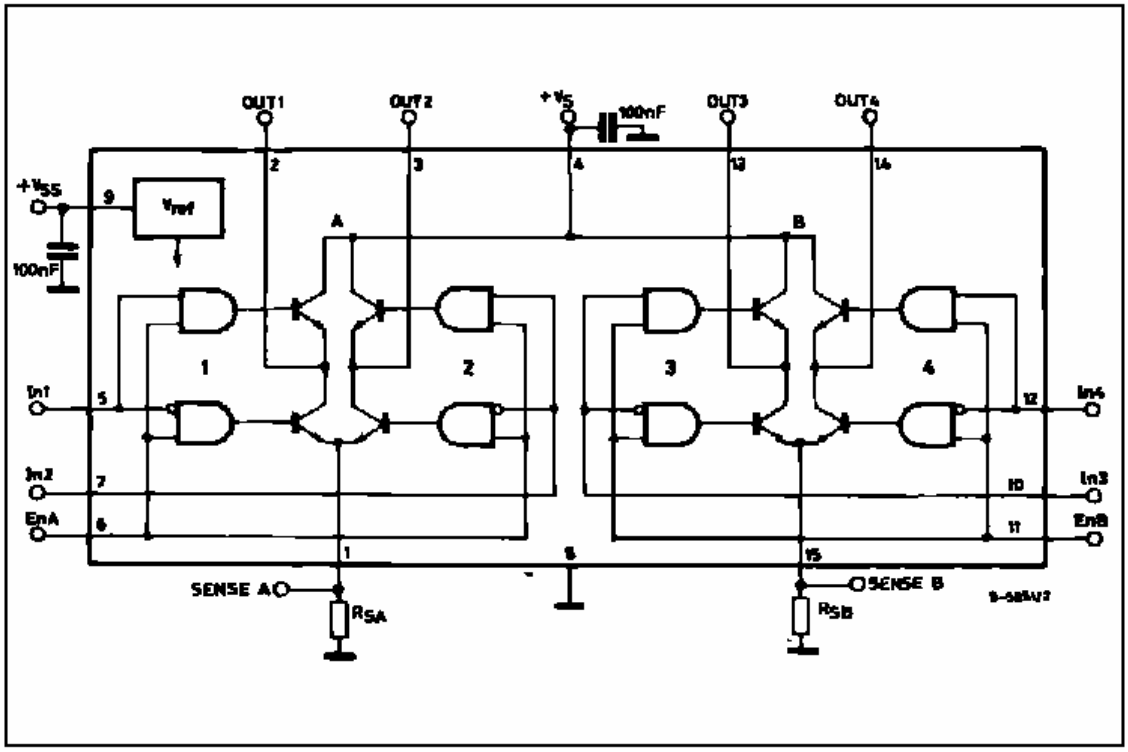


The 8 sensors are connected to PORTA.

You need not connect anything to AVCC and AREF, it is required only if ADC is used.

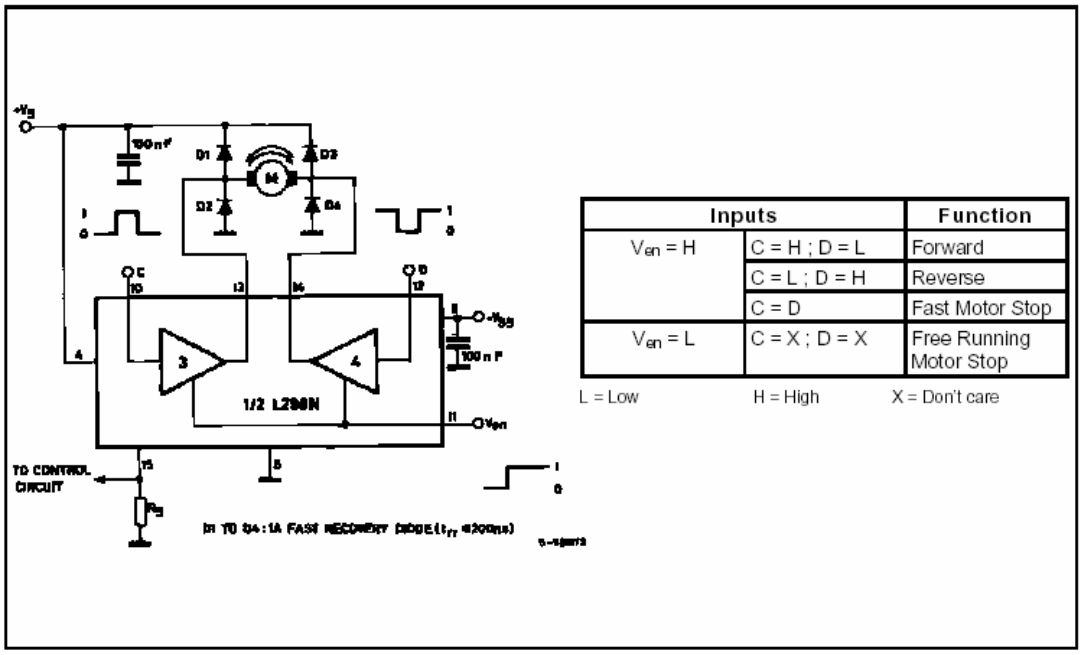
The L298 Motor Driver has 4 inputs to control the motion of the motors and two enable inputs which are used for switching the motors on and off. To control the speed of the motors a PWM waveform with variable duty cycle is applied to the enable pins. Rapidly switching the voltage between Vs and GND gives an effective voltage between Vs and GND whose value depends on the duty cycle of PWM. 100% duty cycle corresponds to voltage equal to Vs, 50 % corresponds to 0.5Vs and so on. The 1N4004 diodes are used to prevent back EMF of the motors from disturbing the remaining circuit. Many circuits use L293D for motor control, I chose L298 as it has current capacity of 2A per channel @ 45V compared to 0.6 A @ 36 V of a L293D. L293D's package is not suitable for attaching a good heat sink, practically you can't use it above 16V

without frying it. L298 on the other hand works happily at 16V without a heat sink, though it is always better to use one.



Internal Schematic of L298

Figure 6 : Bidirectional DC Motor Control.



Inputs		Function
V_{en} = H	C = H ; D = L	Forward
	C = L ; D = H	Reverse
	C = D	Fast Motor Stop
V_{en} = L	C = X ; D = X	Free Running Motor Stop

L = Low H = High X = Don't care

Truth Table for controlling the direction of motion of a DC motor

Source Code

```
/*****************************************************
Project : Line Follower
Version :
Date    : 2/19/2006
Author  : Priyank
Company : Home
Comments:

Chip type           : ATmega16
Program type        : Application
Clock frequency     : 7.372800 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 256
*****************************************************/

//#define debug 1
#include <mega16.h>
#include <delay.h>
#ifdef debug
#include <stdio.h>
#endif

#define FWD 0xAA
#define REV 0x55
#define R 0x22
#define L 0x88
#define CW 0x99
#define CCW 0x66
#define STOP 0x00
#define B 0xFF
#define RSPEED OCR1AL
#define LSPEED OCR1BL
#define SPEED0 255
#define SPEED1 0
#define SPEED2 0
#define SPEED3 0
#define MAX 3
#define HMAX 1

void move (unsigned char dir,unsigned char delay,unsigned char power);
unsigned char i,rdev,ldev,ip,delay,dir,power,dirl,history[MAX],hcount=0,rotpow;

#ifdef debug
unsigned char rep=0,prev=0;
#endif

void main(void)
{

// Input/Output Ports initialization
// Port A initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
```

```
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTA=0x00;
DDRA=0x00;

// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTB=0x00;
DDRB=0x00;

// Port C initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTC=0x00;
DDRC=0xFF;

// Port D initialization
// Func7=In Func6=In Func5=Out Func4=Out Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=0 State4=0 State3=T State2=T State1=T State0=T
PORTD=0x00;
DDRD=0x30;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 921.600 kHz
// Mode: Fast PWM top=00FFh
// OC1A output: Non-Inv.
// OC1B output: Non-Inv.
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0xA1;
TCCR1B=0x0A;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0xFF;
OCR1BH=0x00;
OCR1BL=0xFF;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
```

```
TCNT2=0x00;
OCR2=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// INT2: Off
MCUCR=0x00;
MCUCSR=0x00;

#ifdef debug
// USART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART Receiver: On
// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud rate: 57600
UCSRA=0x00;
UCSRB=0x18;
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0x07;
#endif

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x00;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;

while (1){

#ifdef debug
if(rep<255)
rep++;
if(prev!=PINA) {
prev=PINA;
printf("%u\r",rep);
for(i=0;i<8;i++)
printf("%u\t",(prev>>i)&0x01);
rep=0;
}
#endif

if(PINA!=255){
         rotpow=255;
        ldev=rdev=0;

        if(PINA.3==0)
        rdev=1;
        if(PINA.2==0)
        rdev=2;
        if(PINA.1==0)
        rdev=3;
```

```
        if(PINA.0==0)
        rdev=4;

        if(PINA.4==0)
        ldev=1;
        if(PINA.5==0)
        ldev=2;
        if(PINA.6==0)
        ldev=3;
        if(PINA.7==0)
        ldev=4;

        if(rdev>ldev)
        move(R,0,195+12*rdev);
        if(rdev<ldev)
        move(L,0,195+12*ldev);
        if(rdev==ldev)
        move(FWD,0,200);
        }

else    {
        for(i=0,dirl=0;i<MAX;i++) {
        if(history[i]==L)
        {dirl++;}
        }
         if(rotpow<160) {rotpow=160;}
         if(rotpow<255) {rotpow++;}

        if(dirl>HMAX)
        {move(CW,0,rotpow);}
        else
        {move(CCW,0,rotpow);}
        }
};
}

void move (unsigned char dir,unsigned char delay,unsigned char power) {
PORTC=dir;
if(dir==L || dir==R) {
        hcount=(hcount+1)%MAX;
        history[hcount]=dir;
        }
LSPEED=RSPEED=255;//power;
//delay_ms(delay);
}
```

Possible Improvements:

-Use of differential steering with gradual change in wheel speeds.
-Use of Hysteresis in sensor circuit using LM339
-Use of ADC so that the exact position of the line can be interpolated
-Use of Wheel Chair or three wheel drive to reduce traction.
-General improvements like using a low dropout voltage regulator, lighter chassis etc

References and Resources

Books:

Programming and Customizing the AVR Microcontroller – Dhananjay V. Gadre
Parallel Port Complete – Jan Axelson

Links:

Atmel Corp.
Makers of the AVR microcontroller
http://www.atmel.com

AVRbeginners.net
http://www.avrbeginners.net/

AVR assembler tutorial
Tutorial for learning assembly language for the AVR-Single-Chip-Processors AT90Sxxxx from ATMEL with practical examples.
http://www.avr-asm-tutorial.net/

One of the best sites AVR sites
http://www.avrfreaks.net

WinAVR
An open source C compiler for AVR
http://sourceforge.net/projects/winavr

PonyProg
A widely used programmer. Support for newer chips is added periodically. Can also program PICs and EEPROMS
http://www.lancos.com/prog.html

Basic Electronics
http://www.kpsec.freeuk.com/

Williamson Labs
Nice animated tutorials, articles and project ideas.
http://www.williamson-labs.com/home.htm

Small Robot Sensors
http://www.andrew.cmu.edu/user/rjg/websensors/robot_sensors2.html

Robotics India
An Indian site devoted to robotics. Must see
http://www.roboticsindia.com/

Seattle Robotics Society
http://www.seattlerobotics.org/

Line Follower ROBOT
Award winner from VingPeaw Competition 2543, the robot built with 2051, L293D, and four IR sensors. Simple circuit and platform, quick tracking and
Easy to understand program using C language.
http://www.kmitl.ac.th/~kswichit/LFrobot/LFrobot.htm

Tools: AVR Studio
For writing code in assembly and simulation of code. Current versions has AVR-GCC plug-in to write code in C.

Compilers: IAR, Image Craft , Code Vision AVR, WinAVR

Programmers: Pony Prog, AVR Dude, AVRISP and many more.

Evaluation Boards: STK200, STK500 from Kanda Systems

Shops:

Motors:
1. Mechtex - Mulund, Mumbai
2. Servo Electronics - Lamington road, Mumbai Phone:56346573
3. Bombay Electronics - Lamington Road, Mumbai

Electronics:
1. Visha Electronics - Lamington road, Mumbai
   [Programmer for 8051, PIC and AVR available]
   Phone: 23862650 / 23862622

2. Gala Electronics - Lamington road, Mumbai

3. Chip Components - Lamington road, Mumbai
   Telephone: 56390468 / 56587005

Parts and Prices:

Part	Approximate Price in Indian Rupees
Visible Light Leds White or Bicoloured	1.00 5.00
IR LED IR Sensor	3.00 7.00
Capacitor (small values) Capacitor (large values / electrolytic)	0.25 to 2.00 2.00 to 20.00 Or more
Resistors (1/4 W) Variable Resistor (Preset) Variable Resistor (Pot)	0.25 2.50 8.00
Microcontrollers AT89C2051 (8051 Core) AT89C51 (8051 Core) AT89S52 (8051 Core) PIC16F84A (PIC Core)	40 to 450 40 60 150 120

ATmega8 (AVR Core) ATmega16 (AVR Core) ATmega32 (AVR Core) ATmega128 (AVR Core)	90 150 300 425
Transistors Low power Eg: BC547 Power Transistor Eg: TIP31C	1.50 to 15 or more 1.50 15.00
Connectors	1.00 per pin
Optocoupler (MCT2E)	8.00
Common Tools Soldering Iron Solder metal Solder Flux Desoldering Wick Breadboard Wire Stripper	 150 (typical) to 400 25.00 5.00 5.00 80.00 to 100.00 25.00
Common ICs Voltage Regulators (78XX), LM324,IC555 etc MAX232 ULN 2003 / ULN 2803 TSOP17XX L298, L293, L293D	 5.00 20.00 14.50 17.00 70.00
IC Programmers Homemade (Support fewer devices, support only serial programming, not as rugged) Eg: PonyProg (http://www.lancos.com) Readymade (Support many chips, support parallel programming, easy to use, rugged)	 20.00 to 80.00 2500 to 25000
Wireless Modules	700.00
Parallel Port / Serial Port Add on Card for PC	500.00
Universal PCB	10.00 to 50.00

You may drop your feedback at priyank.patil@gmail.com .

Priyank Patil
KJ Somaiya CoE – Information Technology
VidyaVihar,
Mumbai