Introduction

I always wanted to work on robotics. At the beginning it was just a child's daydream. But when I got introduced with microprocessors as my undergraduate course, I begin to start thinking to make my daydream real. Then I learn microcontroller by myself. And then the hardest part comes through my way. That is, where to start?? I really begin to worry. I don't know what to do. And then one day I saw a video in you tube (thanks to you tube), where I get the idea.

For me, hobby robotics is about creativity. I just didn't want to throw a couple sensors on a robot kit, instead I wanted to build it from the ground up. Looking through catalogs, reading datasheets, and building the robot within the constraints of my limited resources was the ultimate experience. Throughout the process, I kept a couple of concepts in mind:

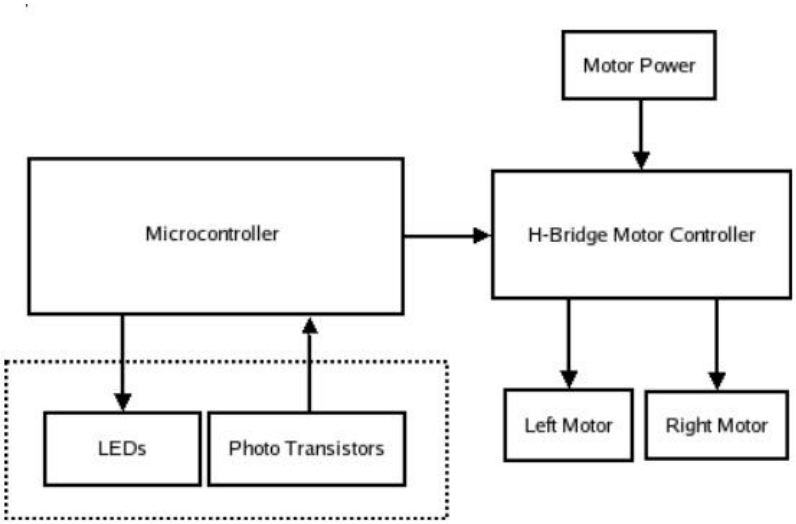
- Close enough is good enough. I'm not a NASA engineer, I'm a hobbyist. It doesn't have to be perfect, it just has to work. This is especially true considering that my "workshop" is merely a bedroom floor. I don't have the facilities to make straight cuts or aligned holes. Improvise.
- The overall goal is to learn and discover. When it felt like I was re-inventing the wheel I just reminded myself that I was learning *how* the wheel was invented.
- Success is a series of failures. It can be very discouraging to spend large amounts of time only to hit a dead-end and start over. The lessons learned made it very much worth while.
- Work within your means. I would often get discouraged by what I see other people building--making molds, welding, etc. I don't have those resources nor a large budget. A roboticist shouldn't be thinking about how it is supposed to be done, but how it *coul*d be done considering the elements.

Line Following Robots

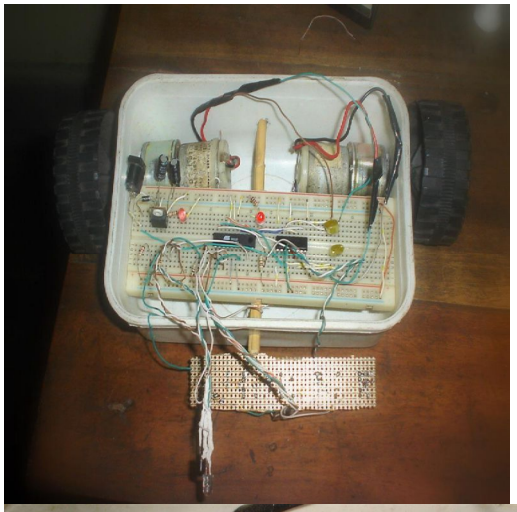
A line-following robot or "line follower" is a pretty common type of robot for hobbyists. Robotics competitions usually have a line-following event. The line is usually a black line about 3/4" wide, such as black electrical tape, on a white surface. Advanced courses may add new challenges such as inclines, tighter turns, intersections, thinner lines, or changing line colors. The robots typically sense the line by measuring light reflected off the ground, where a black line reflects little/no light and the white floor reflects a lot of light back. There are TONS of websites relating to line-following robots as it's a very common beginner project.

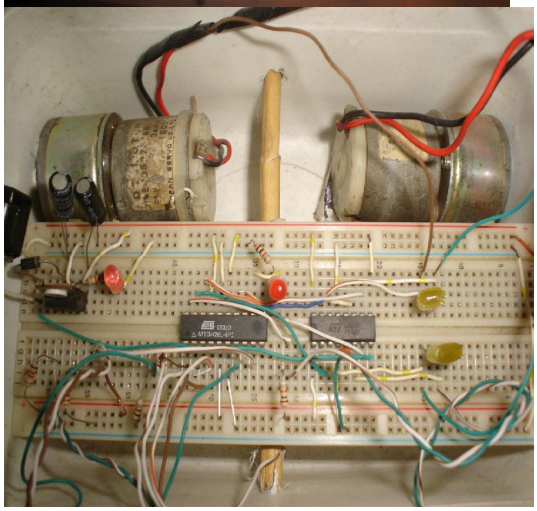
My Robot's Overview

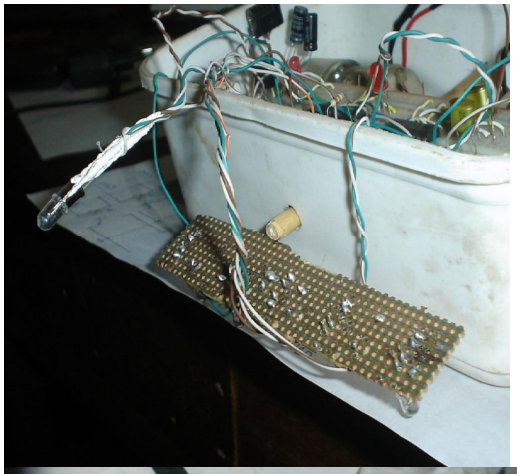
My robot is actually a fairly good robot base which could easily be adapted to maze solving, obstacle avoiding, or other applications of small, 2-wheeled robots. However, at this point in time, it only follows a line and avoid obstacle. The robot consists of 2 DC gearhead motors, a sensor board with 5 photo-transistor/ Infrared LED sensors, an L293D H-Bridge motor controller board, and an Atmel ATtiny26 microcontroller.

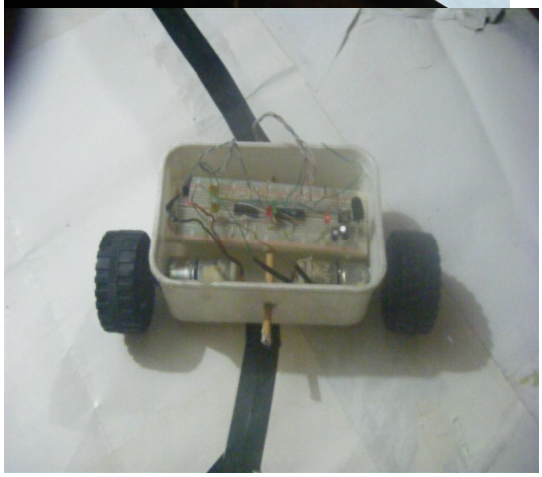


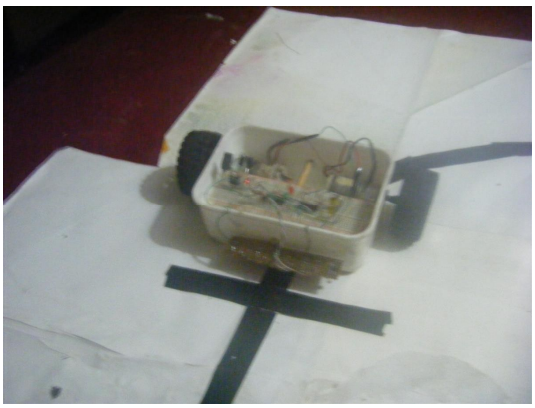
A sensor board with 5 sensors shines light downward at the ground. If the line is underneath the sensor, then the little/no light will be reflected back to a paired phototransistor. If the line is not underneath the sensor, more of the light will be reflected back. A microcontroller measures the output of each of the phototransistors through its analog-to-digital converter (ADC). Based on the position of the line underneath the robot, the microcontroller adjusts the speed/direction of the DC motors to steer the robot.











Building The Robot

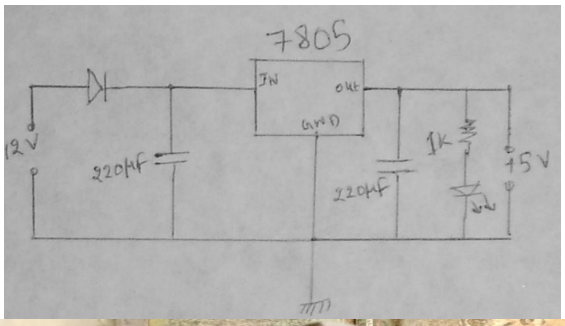
Building the robot was the hardest part for me. I am by no means a mechanical engineer and had a lot to learn. The robot is a 2-wheeled differential-drive robot.

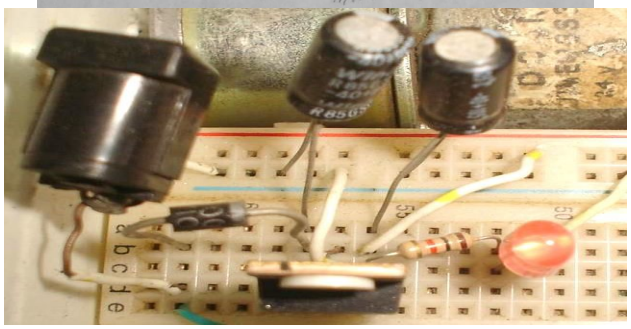
A differential-drive robot works like a tank. The two wheels provide both the drive and the steering. By stopping one motor and not the other, the robot pivots on the stopped motor to turn. By turning one motor forward and the other motor backward, the robot turns in place.

Motors and Wheels

In earlier attempts I had problems with motors not having enough torque to properly move the heavy robot bases I was building. I decided to buy gearhead motors with a very high torque and use larger wheels to make up for the loss in RPMs. I bought these motors from “Dholaikhal”. It cost 350tk each, that mean 700tk total. This is the only expensive part. I collect the tire from my nephew’s toy. These relatively large wheels allow the slower motors to suffice for moving the robot quickly.

Power Supply Ckt

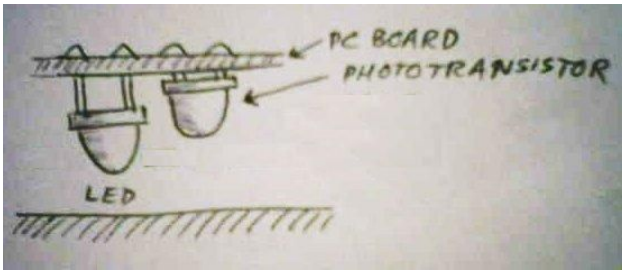


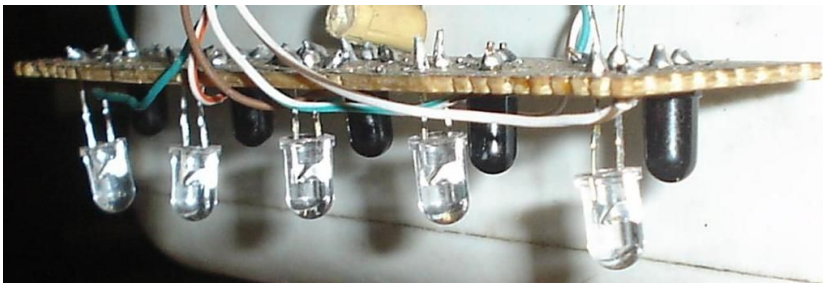


Sensor Board

The sensor board was one of the only things that worked since the very first. There are 5 line sensors. Each line sensor consists of a single infrared LED and a photo-transistor. The electronics of the sensor board will be discussed later. However, the construction here does make a difference. The phototransistors will be sensitive to the ambient light in the environment and thus I want to keep that down to a minumum. I want the light from the LEDs bouncing off the ground back to the phototransistor to be the primary light hitting the phototransistor.

That's why I placed them as the figure shows:





I wasn't too concerned with one sensor being skewed or protruding more/less as I knew I could have each LED independently calibrated via software (discussed later).

Robots Electronics

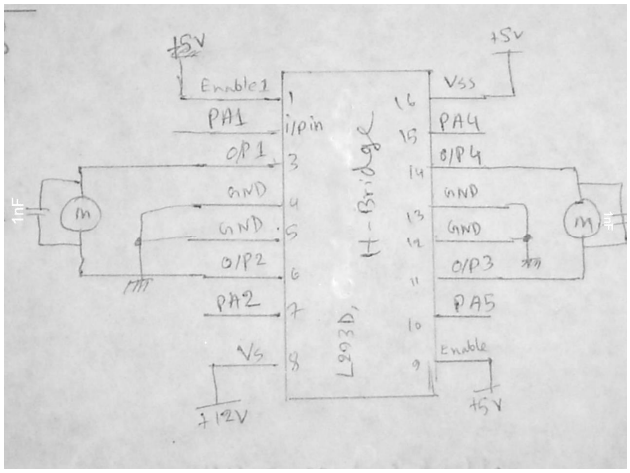
Motor Controller

The motor controller is a simple H-Bridge using the L293D (Tk. 90). 4 lines from the microcontroller are used to control the 2 motors. M1DIRA and M1DIRB is the direction control for the left motor and M2DIRA and M2DIRB is the direction control for the right motor. This is a very common DC motor control concept and thus I'm not going to go into further detail. A Google search will return numerous pages on the topic.

When testing the motors, I found that they pulled 50mA with just the wheel spinning freely and 590mA stalled. The L293D can handle 600mA per channel. Just to be on the safe side, you can put a heatsink on the L293D though it was never necessary.

The most important thing I learned with this part was this: BEWARE OF THE NOISE FROM MOTORS! The motors can add tremendous noise to the supply even when using a separate battery for the motors than from the rest of the electronics

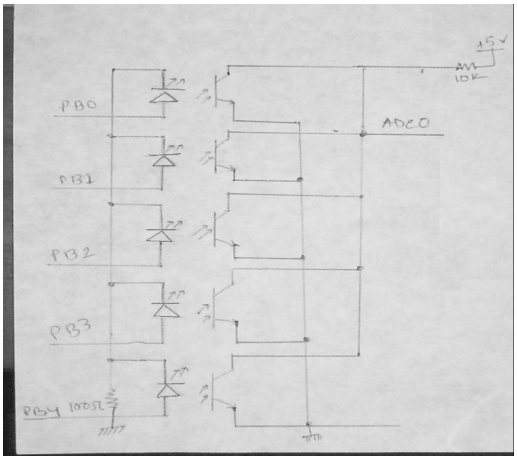
(common ground). The solution was the capacitors on the motors. Three 0.1uF capacitors were used on each motor. One was connected between the motor terminals, and also one from each terminal to the motor's casing. Always twisted the wires. Michael Simpson's "Reduce Motor Noise" discusses these techniques in more detail.



Sensor Board

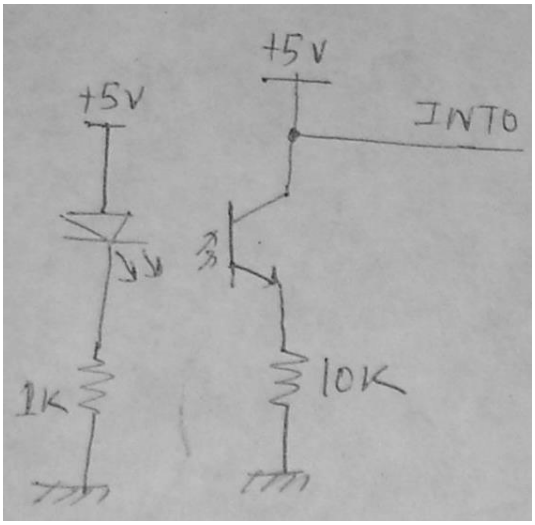
For each sensor, I used an Infrared LED paired with a phototransistor. I did not want to deal with difficult hardware for the sensors. I knew the tools I have at my disposal will not allow perfection in terms of mounting these LEDs and getting each of the 5 sensors to behave the same. Each of the phototransistors outputs are tied together resulting in a single output line fed back into the ADC of the microcontroller. However, only one LED is turned on at a time via software. Each sensor is individually calibrated so that ambient light and variations in how each sensor is mounted are accounted for.

For obstacle detection only one pair of infrared LED and photo transistor is used. It don’t need to calibrate.



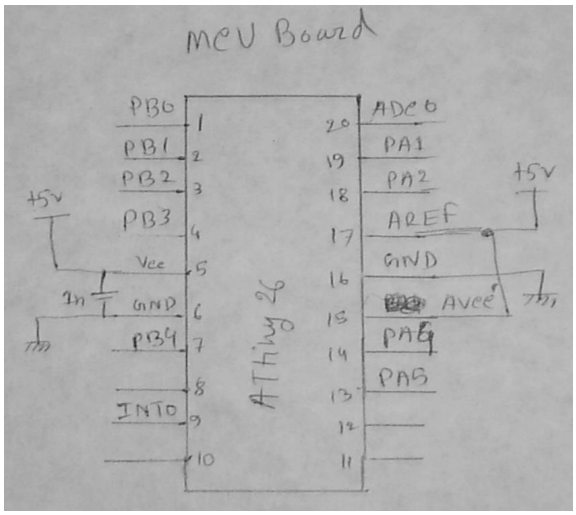
Obstacle Avoiding

For obstacle avoiding I use one pair of infrared LED and phototransistor. Output of the phototransistor is connecting with the “Interrupt 0” pin of the microcontroller.



Microcontroller

This robot has used Atmel's AVR ATtiny26.



Software

The software for the robot is written in C for the Atmel AVR microcontrollers using avr-libc and the AVR Studio compiler. The robot is able to handle 90 degree turns and intersections at a very decent speed. At this point in time, my gear motors have such a high reduction ratio that I do not need to use PWM to change the speed. When turned on, the robot stays still for a couple seconds while it calibrates the sensors. Once this calibration is complete, the robot begins to search the line. When it detects the line it starts to track the line by the line detection algorithm in a loop. If there is any obstacle it'll trigger the external interrupt and it turn over 180 degrees and again track the line.

Calibration

During the calibration, the robot stays still in place above the line in an attempt to "learn" the lighting conditions of the environment when on the line. After a while it goes forward a little to get off the line in an attempt to "learn" the lighting conditions

of the environment when off the line. 2 values are setup for each sensor. A low value indicating the lowest value read from that sensor and one for the highest value for that sensor. At the end of the calibration, the middle of these two values is the "trip point" for the sensor. Anything above this trip point is considered high and anything below is considered low. Each LED is calibrated many times over, one at a time, as follows:

```
Turn on LED.
Measure ADC value from phototransistor.
If the value is lower than the lowest value measured, save this as
the lowest value.
If the value is higher than the highest value measured, save this as
the highest value.
Turn off LED.
```

Then, a trip point is setup as `low value + ((high value - low value) / 2)`.

Steering

There are 6 directions defined for the robot.

`GO_LEFT`	The left motor is stopped and the right motor goes forward. Robot pivots to the left on the left wheel.
`GO_HARD_LEFT`	The left motor goes reverse and the right motor goes forward. Robot turns in place to the left.
`GO_FORWARD`	Both motors go forward and robot goes forward.
`GO_HARD_RIGHT`	The right motor goes reverse and the left motor goes forward. Robot turns in place to the right.
`GO_RIGHT`	The right motor is stopped and the left motor goes forward. Robot pivots to the right on the right wheel.
`GO_BRAKE`	Both motors are stopped.

Line Detection Logic

The robot decides which direction to go based on the 5 sensors. Each sensor is read one at a time and the bit value is stored in a variable. The bits are left shifted into the variable starting with the left sensor. Therefore, the leftmost sensor is the most significant bit in the variable. Based on the final result of all 5 sensors being left-shifted into the variable, the robot decides how to steer.

Bit Pattern	Hex Value	Decision
`00100`	`0x04`	The line is in the center. Go forward.
`01110`	`0x0E`	The line is center but really thick? Go forward.
`11111`	`0x1F`	Possible intersection. Go forward.

00001	0x01	Line on right-most sensor. Go hard right.
00011	0x03	Line on right-most sensor. Go hard right.
00111	0x07	Line on right-most sensor. Go hard right.
00010	0x02	Line on mid-right sensor. Go right.
00110	0x06	Line on mid-right sensor. Go right.
01100	0x0C	Line on mid-left sensor. Go left.
01000	0x08	Line on mid-left sensor. Go left.
10000	0x10	Line on left-most sensor. Go hard left.
11000	0x18	Line on left-most sensor. Go hard left.
11100	0x1C	Line on left-most sensor. Go hard left.
?	N/A	All other conditions, continue in the last direction determined. Line could be between sensors or there could be a problem.

Microcontroller Code:

```
/*
   File:       main.c
   Version:    1.1
   Date:       March. 13, 2010
   KnightBot  Version 1. Schematics and details at www.**.com

**********************************************************************
******
   Copyright (C) 2010 Fahad Mirza   <fahad_1046116@yahoo.com>

   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.

**********************************************************************
*****/

/* defines for line sensors */
#define LED_PORT PORTB
#define LED_DDR DDRB
#define LED1 PB0     /* left-most sensor */
#define LED2 PB1
#define LED3 PB2
#define LED4 PB3
#define LED5 PB4     /* right-most sensor */

/* defines for motor control */
#define MOTOR_PORT PORTA
#define MOTOR_DDR DDRA
#define M1DIRA PA1
```

```
#define M1DIRB PA2
#define M2DIRA PA4
#define M2DIRB PA5



#define GO_LEFT 0
#define GO_HARD_LEFT 1
#define GO_FORWARD 2
#define GO_HARD_RIGHT 3
#define GO_RIGHT 4
#define GO_BRAKE 5
#define GO_REVERSE 6


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

unsigned char midpoint[5] = { 0,0,0,0,0 };    /* sensor trip points */
unsigned int k;

/* function prototypes */
void init_io();
void init_adc();
void steer_robot(unsigned char direction);
unsigned char get_line_sensor_value(unsigned char led_index);
void calibrate_line_sensors();


/* INT0 Interrupt Service Routine (ISR) */
ISR(INT0_vect)
{

        unsigned char i;
        unsigned char adc_value;

        steer_robot(GO_REVERSE);
        for(k=0;k<65535;k++){;}    /* Delay Loop */
        LED_PORT &= ~(1<<LED1) & ~(1<<LED2) & ~(1<<LED4) &
~(1<<LED5);
        steer_robot(GO_HARD_LEFT);

        for(i=0;i<3;i++)
        {
                for(k=0;k<65535;k++){;}
        }

      do
      {
         adc_value = get_line_sensor_value(3);
```

```
    }while(adc_value < midpoint[3]);

    steer_robot(GO_BRAKE);

    // clear pending interrupts
    GIFR |=  (1<<INTF0);
}


int main (void)
{
    unsigned char adc_value;                                /* ADC value */
    unsigned char last_direction = GO_FORWARD; /* last direction steered */
    unsigned char i;                                        /* loop counter */
    unsigned char sensor_bits;                              /* sensor bit values */

    /* initializations */
    init_io();
    init_adc();

    /* calibration */
    calibrate_line_sensors();


    while (1)
    {
        /* clear previous value */
        sensor_bits = 0;

        /*
        create bit patter for sensors where a 1 represents the line
        under the sensor and a 0 represents no line
        */
        for (i=0; i<5; i++)
        {
            sensor_bits = sensor_bits << 1;
            adc_value = get_line_sensor_value(i);
            if (adc_value >= midpoint[i])
            {
                sensor_bits |= 0x01;
            }
            else
            {
                sensor_bits &= 0xFE;
            }
        }

        /* hexidecimal representations for various bit patterns of lines */
```

```
            if (sensor_bits == 0x04 || sensor_bits == 0x0E || sensor_bits == 0x1F)
            {
                steer_robot(GO_FORWARD);
                last_direction = GO_FORWARD;
            }
            else if (sensor_bits == 0x0C || sensor_bits == 0x08)
            {
                steer_robot(GO_LEFT);
                last_direction = GO_LEFT;
            }
            else if (sensor_bits == 0x02 || sensor_bits == 0x06)
            {
                steer_robot(GO_RIGHT);
                last_direction = GO_RIGHT;
            }
            else if (sensor_bits == 0x10 || sensor_bits == 0x18 || sensor_bits == 0x1C)
            {
                steer_robot(GO_HARD_LEFT);
                last_direction = GO_HARD_LEFT;
            }
            else if (sensor_bits == 0x01 || sensor_bits == 0x03 || sensor_bits == 0x07)
            {
                steer_robot(GO_HARD_RIGHT);
                last_direction = GO_HARD_RIGHT;
            }
            else
            {
                 /* If the robot does not see a line at all, it's possible that the line is
between
                sensors. Continue going the direction previously determined until the
line is
                found.*/
                steer_robot(last_direction);
            }
        }
        return (0);
}

void init_io()
{
        /* setup input/output */
        LED_DDR |= (1<<LED1) | (1<<LED2) | (1<<LED3) | (1<<LED4) | (1<<LED5);
        MOTOR_DDR |= (1<<M1DIRA) | (1<<M1DIRB) | (1<<M2DIRA) |
(1<<M2DIRB);

        GIMSK= (1<<INT0);    /* set INT0 interrupt enable bit */
        MCUCR |= (1<<ISC01) | (1<<ISC00);  // INT0 to rising edge

        sei();  /* enable interrupts */
}
```

```
void init_adc(void)
{
	ADMUX |=  (1<<REFS0) | (1<<ADLAR); // channel 0, left-justified result
	ADCSR |= (1<<ADEN) | (1<<ADPS1); // Prescaler 4
}

void steer_robot(unsigned char direction)
{
	switch (direction)
	{
		case GO_LEFT:
			MOTOR_PORT &= ~(1<<M1DIRA);    /* clear M1 */
			MOTOR_PORT &= ~(1<<M1DIRB);

			MOTOR_PORT |= (1<<M2DIRA);       /* set M2 */
			MOTOR_PORT &= ~(1<<M2DIRB);
			break;

		case GO_HARD_LEFT:
			MOTOR_PORT &= ~(1<<M1DIRA);
			MOTOR_PORT |= (1<<M1DIRB);       /* reverse M1*/

			MOTOR_PORT |= (1<<M2DIRA);       /* set M2 */
			MOTOR_PORT &= ~(1<<M2DIRB);
			break;

		case GO_FORWARD:
			MOTOR_PORT |= (1<<M1DIRA);       /* set M1 */
			MOTOR_PORT &= ~(1<<M1DIRB);

			MOTOR_PORT |= (1<<M2DIRA);       /* set M2 */
			MOTOR_PORT &= ~(1<<M2DIRB);
			break;


		case GO_HARD_RIGHT:
			MOTOR_PORT |= (1<<M1DIRA);       /* set M1 */
			MOTOR_PORT &= ~(1<<M1DIRB);

			MOTOR_PORT &= ~(1<<M2DIRA);
			MOTOR_PORT |= (1<<M2DIRB);       /* reverse M2 */
			break;

		case GO_RIGHT:
			MOTOR_PORT |= (1<<M1DIRA);       /* set M1 */
			MOTOR_PORT &= ~(1<<M1DIRB);

			MOTOR_PORT &= ~(1<<M2DIRA);      /* clear M2 */
			MOTOR_PORT &= ~(1<<M2DIRB);
			break;
```

```
            case GO_BRAKE:
                MOTOR_PORT &= ~(1<<M1DIRA);      /* clear M1 */
                MOTOR_PORT &= ~(1<<M1DIRB);

                MOTOR_PORT &= ~(1<<M2DIRA);      /* clear M2 */
                MOTOR_PORT &= ~(1<<M2DIRB);
                break;

            case GO_REVERSE:
                MOTOR_PORT &= ~(1<<M1DIRA);      /* clear M1 */
                MOTOR_PORT |= (1<<M1DIRB);

                MOTOR_PORT &= ~(1<<M2DIRA);      /* clear M2 */
                MOTOR_PORT |= (1<<M2DIRB);
                break;

            break;
        }
}

unsigned char get_line_sensor_value(unsigned char led_index)
{
        unsigned char adc_value;

        switch (led_index)
        {
            case 0: LED_PORT |= (1<<LED1); break;
            case 1: LED_PORT |= (1<<LED2); break;
            case 2: LED_PORT |= (1<<LED3); break;
            case 3: LED_PORT |= (1<<LED4); break;
            case 4: LED_PORT |= (1<<LED5); break;
        }

        for (k=0;k<5000;k++){;}  /* Delay Loop */

        /* read output from ADC */
        ADCSR |= (1<<ADSC);
        while (!(ADCSR & (1<<ADIF)));
        adc_value = ADCH;
        ADCSR |= (1<<ADIF);

        /* turn off LED */
        switch (led_index)
        {
            case 0: LED_PORT &= ~ (1<<LED1); break;
            case 1: LED_PORT &= ~ (1<<LED2); break;
            case 2: LED_PORT &= ~ (1<<LED3); break;
            case 3: LED_PORT &= ~ (1<<LED4); break;
            case 4: LED_PORT &= ~ (1<<LED5); break;
        }
```

```
	return adc_value;
}

void calibrate_line_sensors()
{
	unsigned char adc_value;	/* ADC value */
	unsigned char i, j;		/* loop counter */
	unsigned char thresh_high[5] = { 0,0,0,0,0 };
	unsigned char thresh_low[5] = { 255,255,255,255,255 };

	for (i=0; i<50; i++)
	{
		for (j=0; j<5; j++)
		{
			adc_value = get_line_sensor_value(j);
			if (adc_value < thresh_low[j]) thresh_low[j] = adc_value;
			if (adc_value > thresh_high[j]) thresh_high[j] = adc_value;
		}


		if(i==25)
		{
			steer_robot(GO_FORWARD);
			for(k=0;k<65535;k++){;}
			steer_robot(GO_BRAKE);
		 }

	}

	for (i=0; i<5; i++)
	{
		midpoint[i] = (thresh_low[i] + (thresh_high[i] - thresh_low[i]) / 2);
	}

}
```