DEPARTMENT OF ELECTRICAL ENGINEERING, IIT BOMBAY

EDL Final Report

Roomba: The Robo-vac

D4: Sunny Mahajan, Himesh Joshi, Arun Mukundan and Ankit Mehta

1st Dec '09

The Objective of this project was to design a vacuum cleaner that can pick up dirt autonomously, while intelligently avoiding obstacles as well as cliffs and covering the entire room.

Final system comprises of the following blocks

- Sensors for wall, cliff and obstacle detection
- Differentially Driven DC Motors
- An LCD interface for scheduling tasks
- The vacuum mechanism
- An intelligent algorithm to manage the allotted task.

Individual contributions:

- *Sunny Mahajan* – Cliff Sensor, Wall Sensor, Obstacle Sensor, Bump Sensor, Vacuum Unit, Motor Control Unit, Navigation Algorithm, Bot Design and ergonomics
- *Himesh Joshi* – Cliff Sensor, Bot Design and Motor Control Unit.
- *Arun Mukundan* – LCD Display Unit and Bot Design.
- *Ankit Mehta* – Cliff Sensor.

Bill of Materials

So far, we have only purchased bump sensors and a cheap USB powered vacuum. All other components were designed using the components available in the laboratory.

IEEE Code of Ethics

In our designs, we have not violated any patent or trade secret to the best of our knowledge. All our designs and testing has been completed indigenously.

Hardware Accomplishments

1. **Sensors**

- Cliff sensors
- Obstacle detection
- Bump Sensors

2. **Motors**

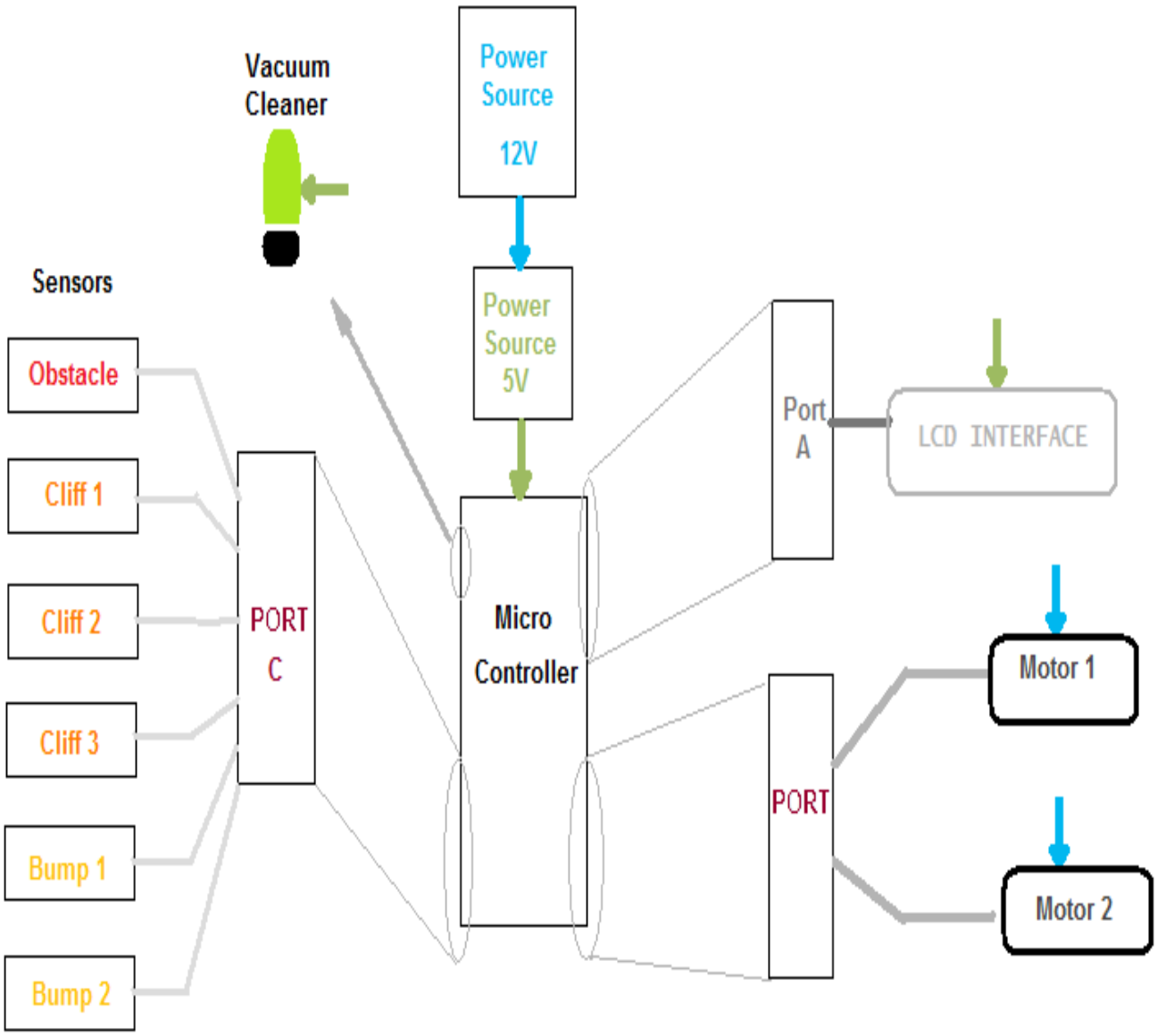
- Basic working
- Routines

3. **LCD Interface and System Scheduling Unit.**
4. **Vacuum Interface(Remotely Controllable)**
5. **Control Algorithm**

Summary

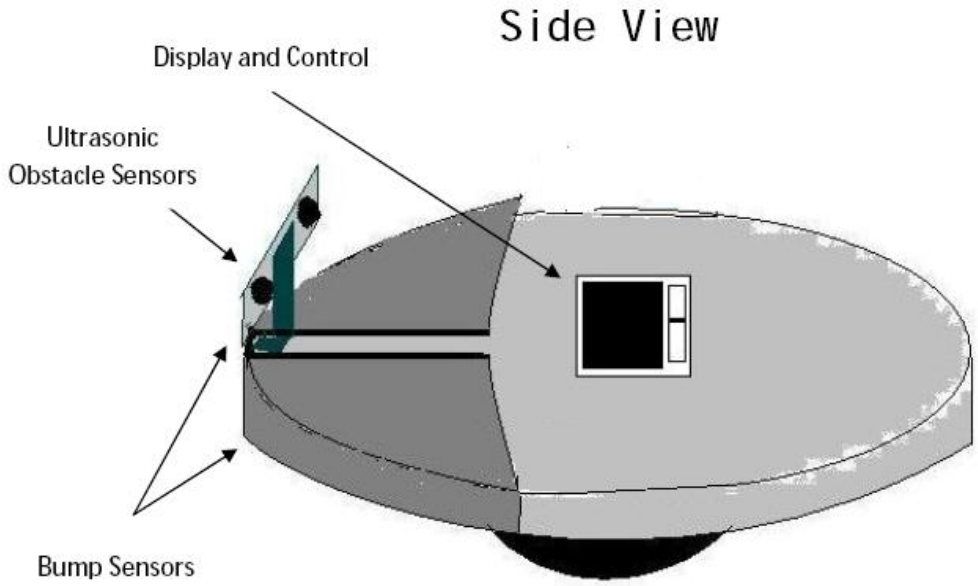
With this project, we intend to learn about making a deliverable, and also to face the technical and design challenges that go into this process. We chose to make an indigenous autonomous robotic vacuum cleaner with the objective of usage in a real world environment replete with static and moving obstacles, walls, cliffs, etc.

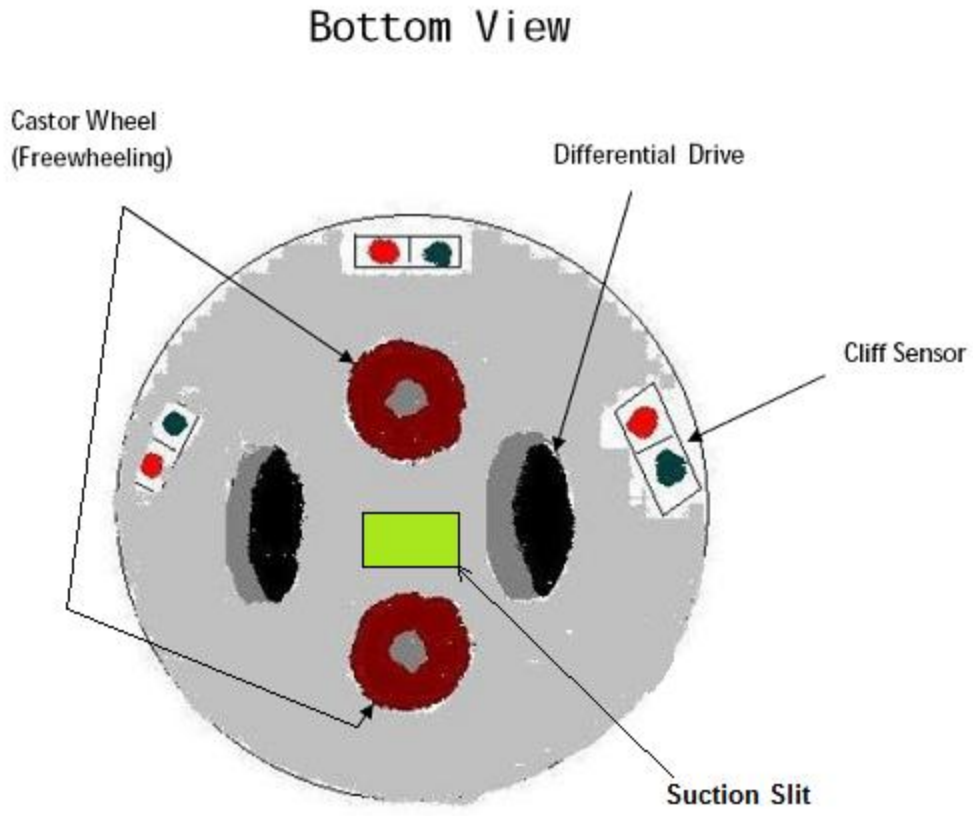
Block Diagram



1. Sensors

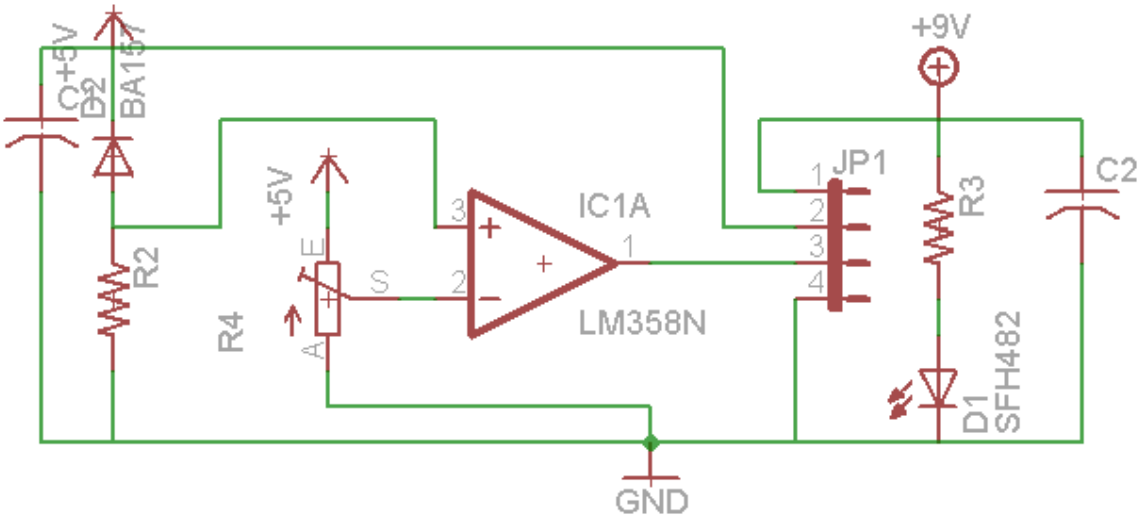
The designs have been tested and the boards fabricated. The sensor placement and the final design we've envisioned is depicted in the followingimage:

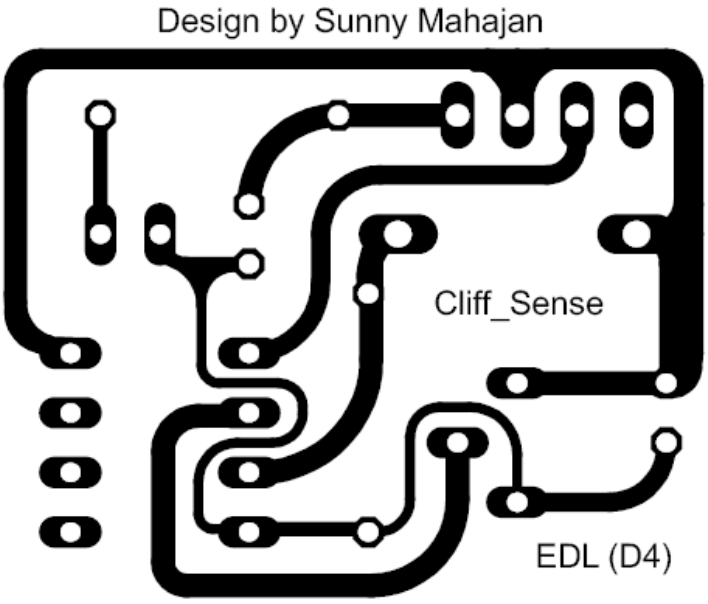


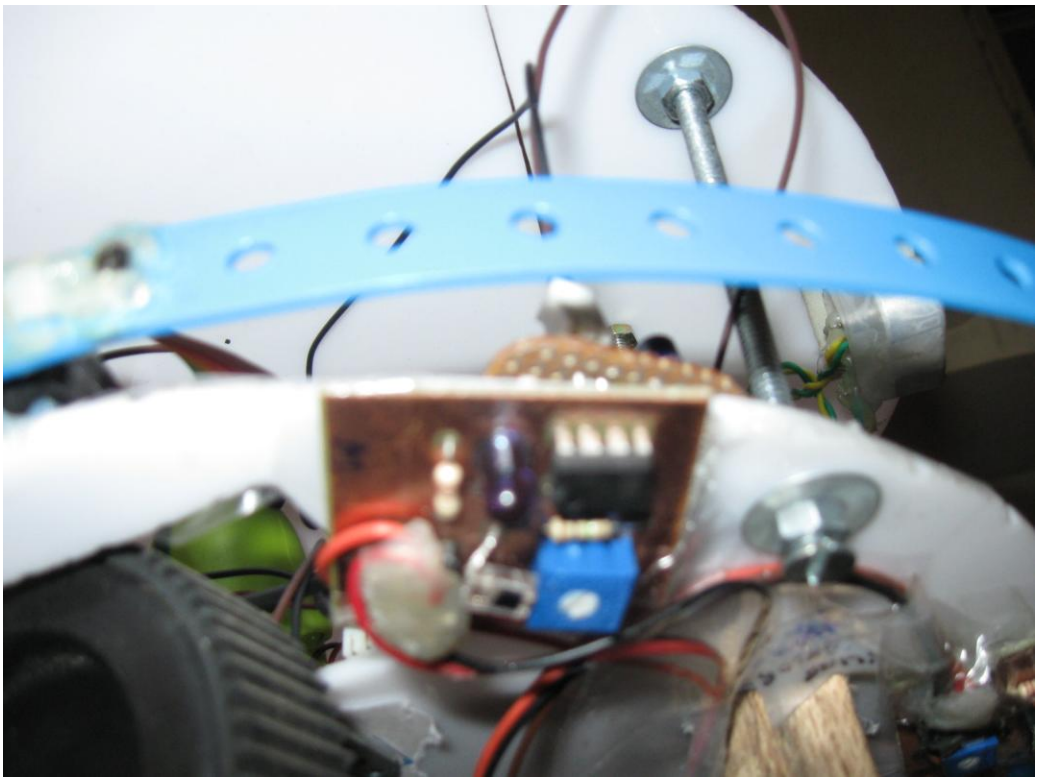


1. Cliff Sensor

This sensor adds the capability to avoid cliffs such as stairs in a house and makes our bot more applicable to real world situations. As this sensor is located at the bottom of the bot, there is no issue of ambient light based false detections and so, IR led/photodiode pair was used to keep the cost at a minimum.





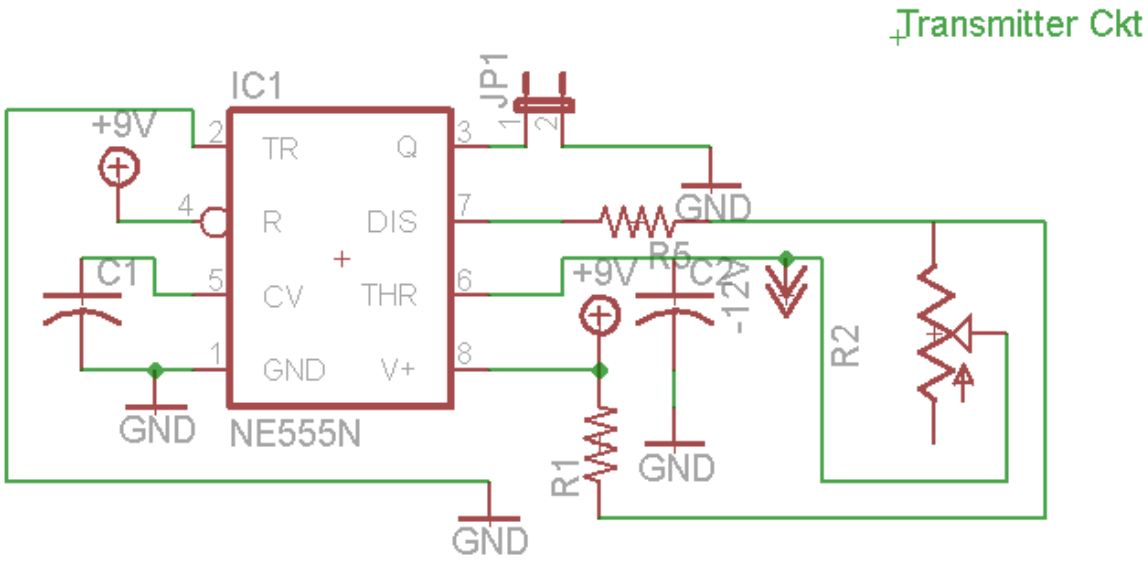


2.Obstacle Sensor

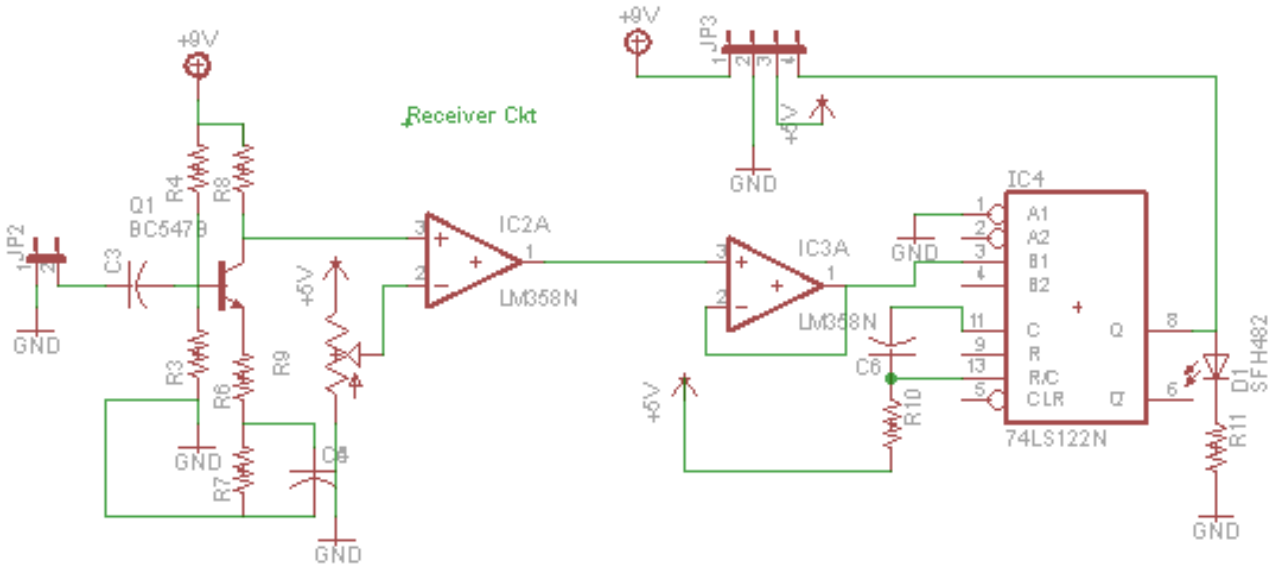
This gives the ability to decide when the bot is on a collision course and make adaptive decisions to navigate accordingly. For this purpose, ultrasonic transceiver pair was utilized as audio waves are known to have maximum configurable range and are reflected by most materials as compared to IR which is absorbed easily by objects. This enhances the applicability of our robot.

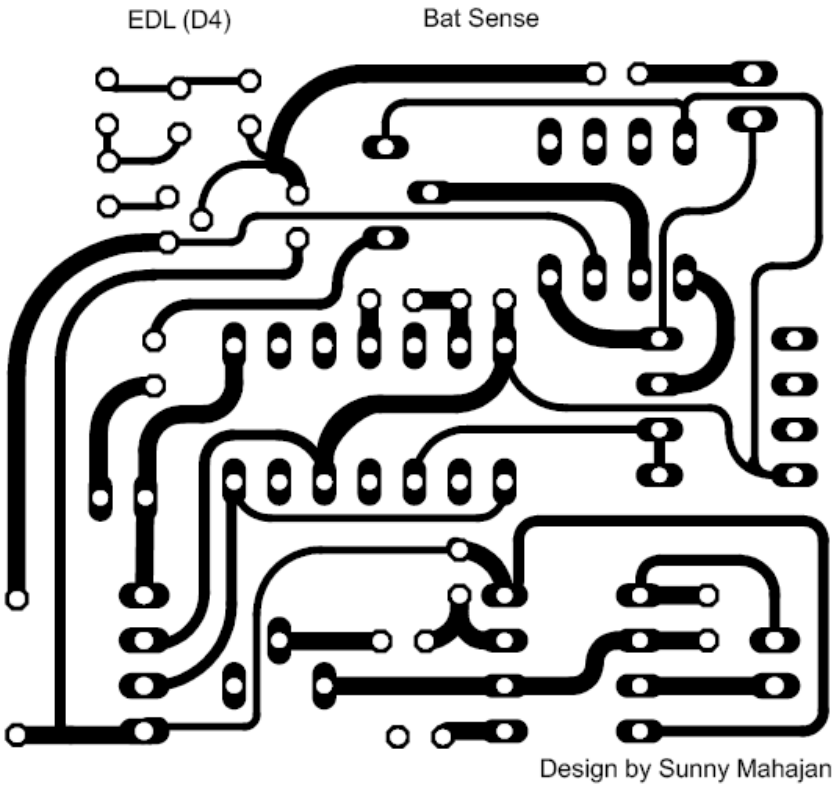
The sensor design comprises of a Piezoelectronic Ultrasonic Transmitter fed off a 40kHz 555. The reflected waves are received by an Ultrasonic receiver and our amplified by a CE amplifier(bypassed emitter resistor) and the output is buffered and passed on to the next comparator stage where the sinusoid is converted to a pulse train. But since, we need uniform output on detection, we added a retriggerable monostable which gives positive dc on detection.

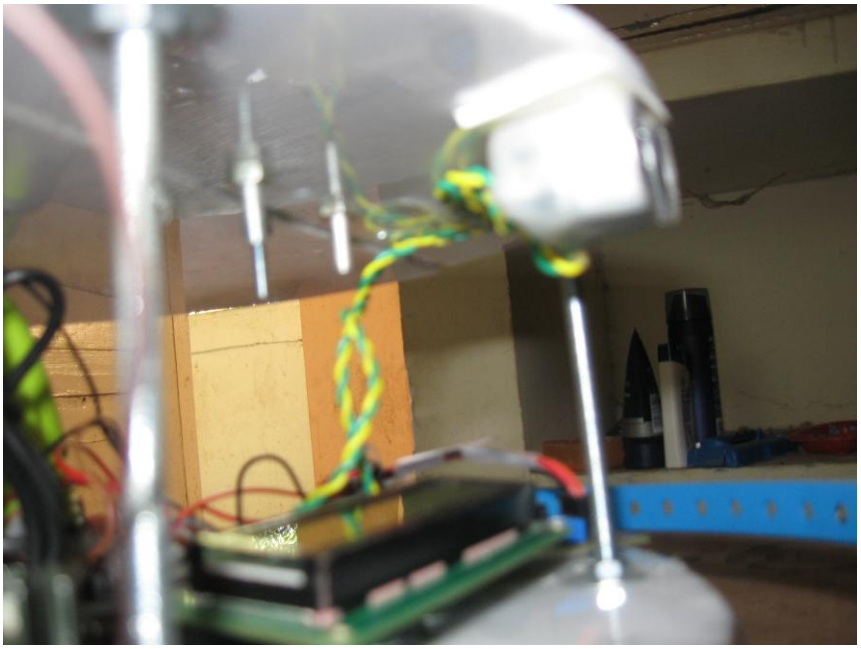
Transmitter Circuit



Receiver Circuit

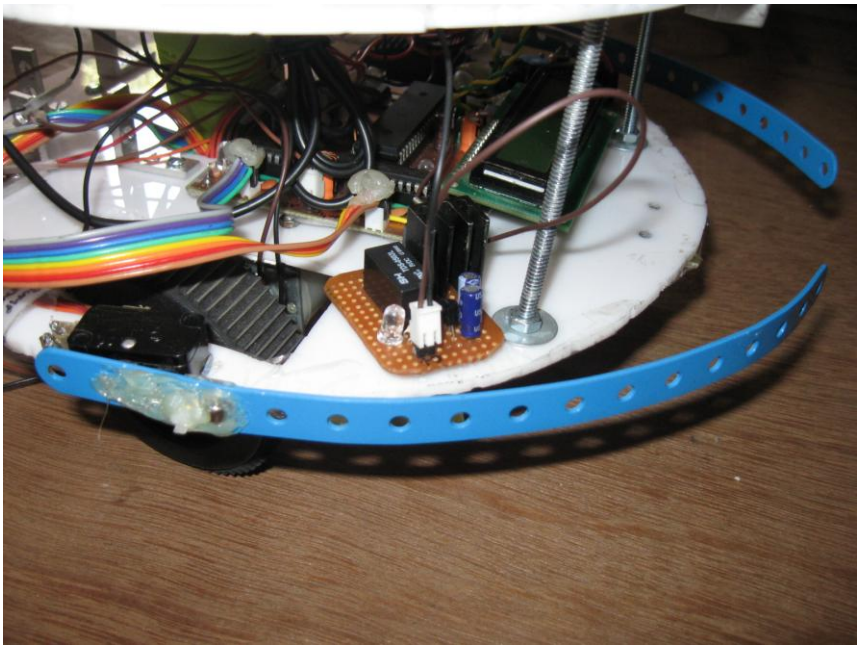



EDL (D4)
Bat Sense
Design by Sunny Mahajan



3. Bump Sensor

These are used in case the bot actually collides. For this, we used leaf switches which close on contact on a throw condition. The `bumps' are made using Lego builder strips connected to tactile leaf switches, which trigger on contact. Mechanical bounce is taken care of.



2. Motor Control Unit

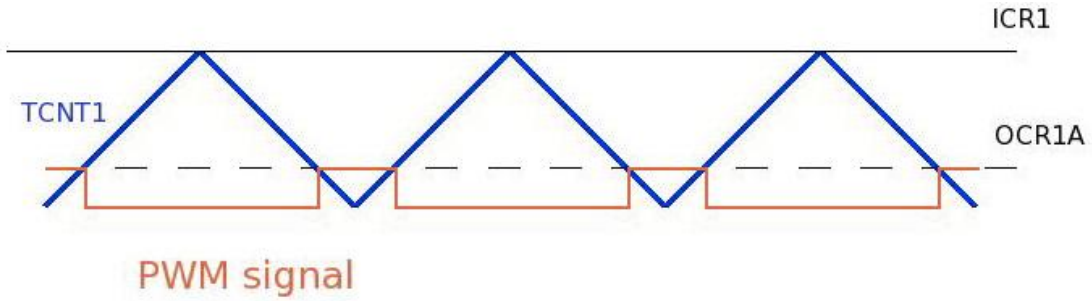
The bot employs the use of DC motors which are being interfaced with the help of a microcontroller, ATMega32.

The programming has been done in order to provide for different modes of motion of the bot. For example, for turning, the concept of differential drive has been used where one of the side wheels is turned off, enabling the bot to rotate at its own position.

Pulse Width Modulation technique has been incorporated whenever spiraling is required, i.e. motion in a circle of a given

radius, which is essential to the Roomba's room filling algorithm that shall be employed later on.

For employing pulse width modulation, the output compare register OCR1A is used. Basically, the value set at the OCR1A register is used as a means of comparison and compared with a uniformly ramped waveform, as shown in the figure below



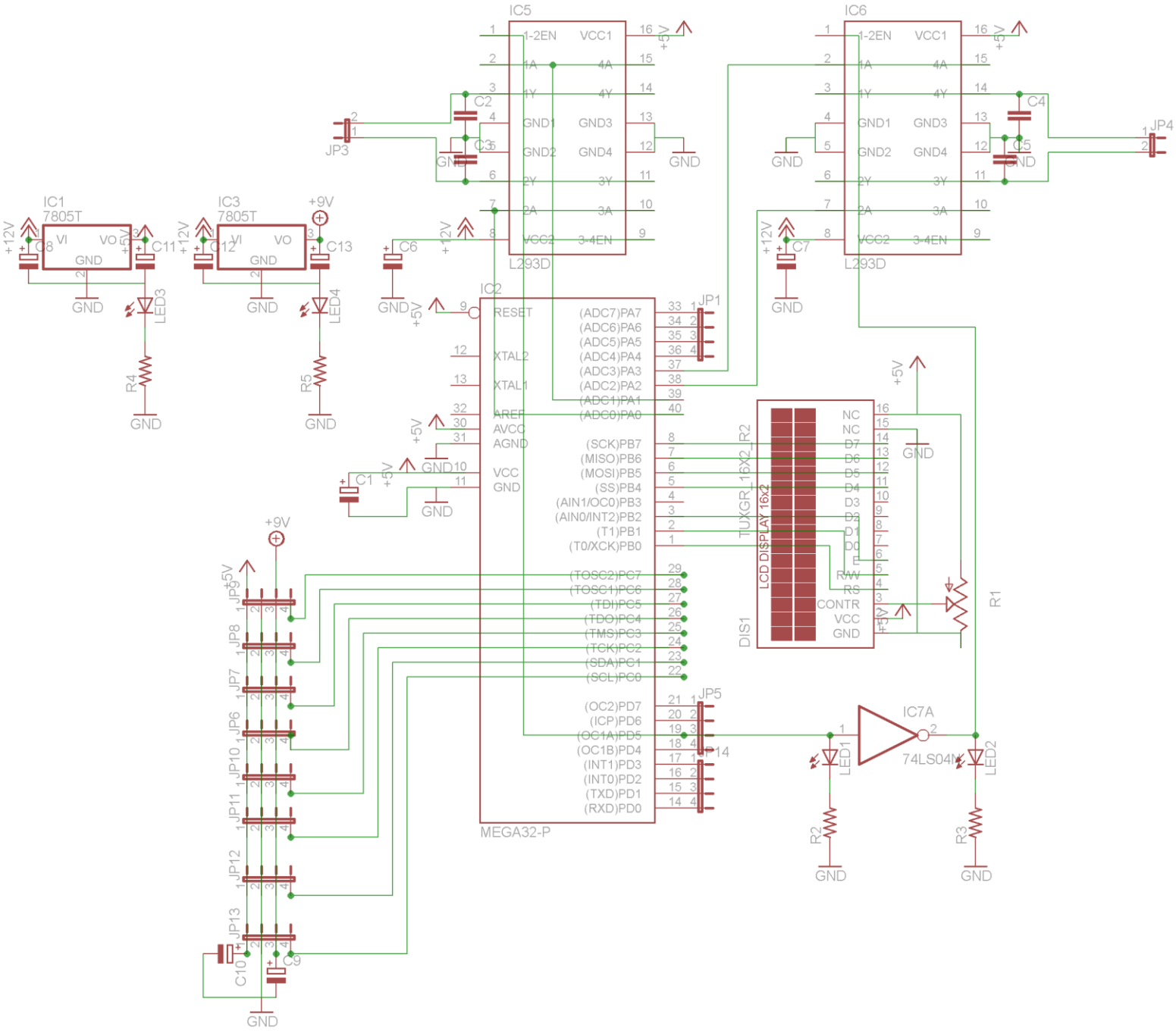
Thus, OCR1A can be set anywhere between 0 and 256, which happen to be the top and bottom values.
To get a duty cycle of 50%, the value is set at 128, which is the middle value. The signal thus obtained is fed to the enable pin of the first motor driver IC (L293D) and complement is fed to the second motor driver IC.
With the help of such a setup at the enable pins, the spiral motion of the bot becomes simple where a change in the value of the compare register will cause opposite shifts in the duty cycles for the two motors.
The advantage of this system is that the calibration is much easier because the sum of both the duty cycles is unity.

We have defined the following routines for moving the bot

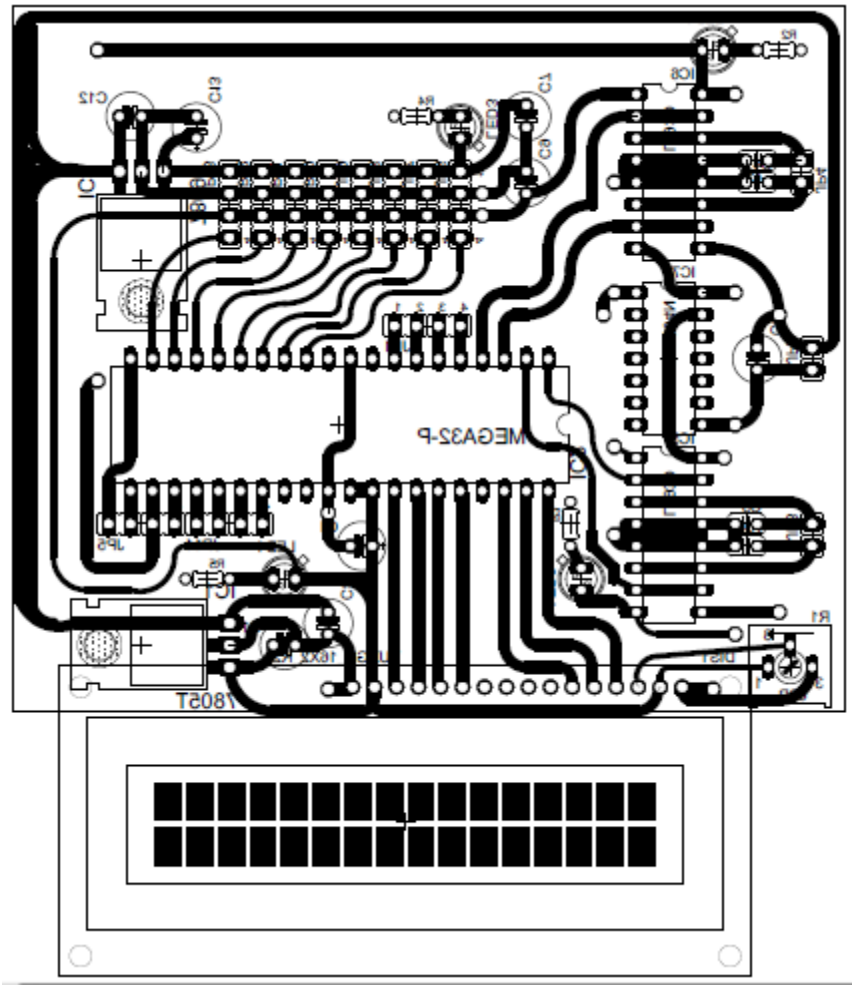
- Move_straight
  The bot moves straight. This is done by balancing the duty cycle for both wheels.
- Turn_left, turn_right
  Takes a 90 degree turn to the left by stopping one wheel and letting the other one run.

- Spiral_out_left, spiral_out_right
  This movement is to take slight turns, by slowing down one wheel in steps.
  It is needed as turns near walls should be gentle, and also our room filling algorithm requires a spiral motion.

Schematic: consists of an Atmega32, two L293D's,a 7805, a 7809 and an LCD display interface.

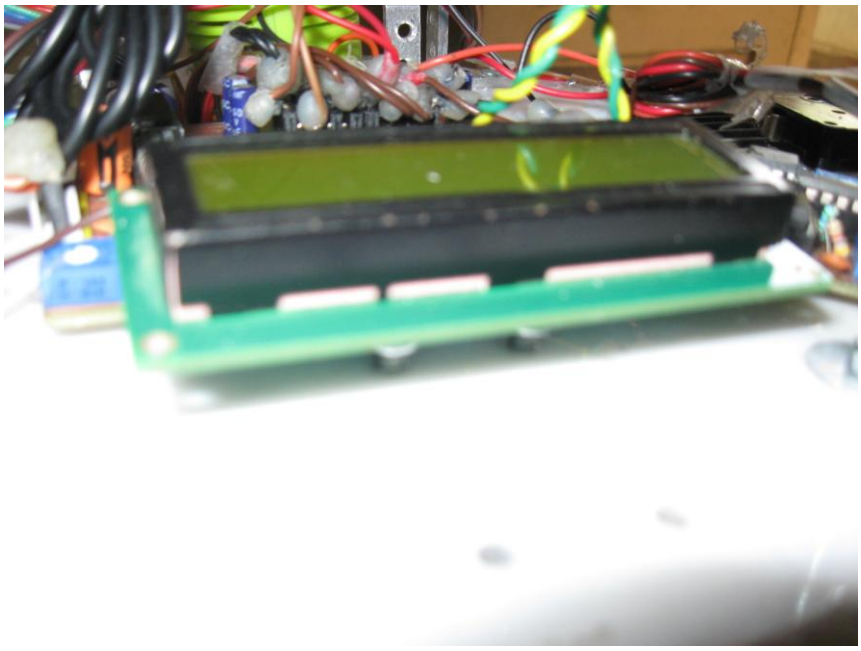


Motor Control Unit Board :



3.LCD Interface

The LCD we are using is a 16X2 character display LCD. We are using this, as it is only for debugging purposes.
The refresh rate is fast enough compared to the times the bot takes to react to and stimulus to observe the necessary stimulus on the screen.



4. Navigation Algorithm

The control algorithm receives inputs from the different sensors and runs the appropriate motor routine.
In order to get the inputs, the outputs from the various sensors are fed to the Port C which is continuously polled to detect the presence of any obstacle/cliff. Once it is known that which sensor was activated, appropriate decisions can be taken to avoid the erroneous path.

The following is the control we use

1. Obstacle Sensor
   Bot should reverse and then pursue a path in either direction. In order to make the decision of which path to pursue a fair one, we introduce some randomness so that the decision will be left or right with probability .5
2. Cliff Sensor
   Bot should figure out which of the cliff sensors fired use the information to avoid the cliff
   - Left sensor – Reverse and turn right to avoid
   - Right sensor – Reverse and turn left to avoid
   - Centre sensor – Reverse and again turn in any direction
3. Bump sensor
   In case the bot does bump into something, it must reverse and turn away. This decision is similar to the one in case of the cliff sensor.

5. Vacuum Control Unit

In order to take care of the vacuum action, we procured a local handheld USB powered vacuum. We modified it in order to interface it to the microcontroller by using one pin to activate/deactivate the vacuum, and used a magnetic relay to open or short the switch that powers the vacuum.

The vacuumed dust can then be cleaned by opening the flap of the bot and emptying out the garbage cache.

6. Power

Our bot is powered by a 12V 1.2A-hr lead acid battery.

7. Testing and calibration

The bot needed the following calibration to be done.

1. Sensors
   The sensor ranges needed to be calibrated. This was done while testing as there was a provision by means of pots to change the ranges.
2. Motors
   The motors needed to be calibrated to ensure that the time for the different operations described was optimal, such as turning and spiraling.
   In order to do this, while testing, we changed the delay in the code in order to get the timing correct.

Finally, we tested the bot in a variety of situations

1. Straight line motion – Negligible curve observed
2. In presence of cliffs – The bot appropriately turned away and did not fall
3. In presence of obstacles – The bot avoids static obstacles well. Due to a provision in the code, that incorporates a counter, we can tell if an obstacle is static or moving, thus moving obstacles are not accounted for, and only static ones are reacted to. Therefore, in a household situation, the bot will not be disturbed from its route by people walking about, or the presence of pets, etc.
4. Real world testing – We let the bot run in our wing, in the presence of stairs, obstacles and moving objects, and found that it behaves in a satisfactory manner.

End of Report