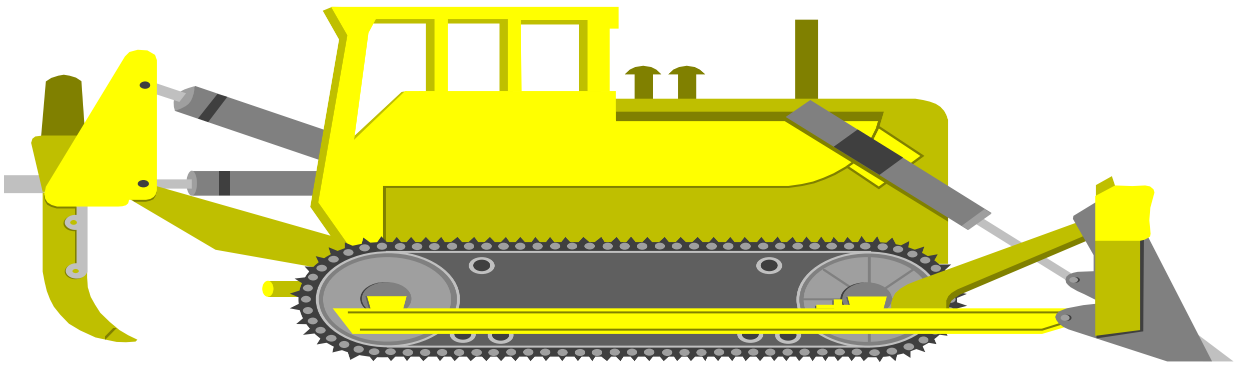


Dickens Rover: Development of a Hole Filling Robot

Thomas Cosenza
EEL 5666
Dr. Keith L. Doty
TA: Scott Jantz
Final Report, 12/12/97

DEDICATION 4

ABSTRACT 5

EXECUTIVE SUMMARY 6

INTRODUCTION 7

THE PHYSICAL 8

Overview of the Physical 8

Parts of the Overall Robot. 8

Power source: 8

Motors: 8

Servos: 9

Sensors: 9

Inferred Detectors (IR) 9

Bump Sensors 9

Cadmium Sulfide Cells (CDS) 9

Collimated IR: 16

THE METAPHYSICAL 17

Overview 17

Behaviors 17

Collision Avoidance 17

Darkness Finding 17

Hole Finding 18

Hole Filling 18

Hole Checking/Flattening 18

Arbitration System **18**

CONCLUSION **18**

WORKS CITED **19**

APPENDIX A- SOFTWARE **20**

Dedication

To a fellow Engineer that was taken before his time. Dickens was a graduate of University of South Florida in the Fall of 1996. Four weeks before graduation he was diagnosed with Lymph node cancer. I meet him in February when he was told that the cancer was in remission. We worked together on a project and every day I saw a little bit of him die every day but he was never down and never shook his fists at god and asked the question why me he just did his job. Instead of lying down and giving up, he faced each day with honor and courage. Dickens finally past away in August of 1997He showed me that nothing was impossible if we were to fight and that life is worth living and not to worry about tomorrow(even if the robot is due). I will always remember him and his big ol' ear to ear smile. Good bye my friend I will see you at some point.

Abstract

This paper is on the design and construction of an autonomous robotic platform for EEL5666 - Intelligent Machine Design Lab at the University of Florida. The goal of the project was to create a robot that will act as an autonomous agent that will go around and fill holes in the ground. The robot will run around and search out dark areas on the ground. If it finds a "dark area" it will then head towards that area and see if it can pick up anything. Once at the hole it has several routines that have been built to do the hole filling and flattening.

Executive Summary

The motivation for this project was the creation of an autonomous agent capable performing a very dangerous task that is normal preformed by a human. In particular, the robot is designed to repair an airport runway after an enemy has bombed it.

The critical part of this project was the robot's ability to find the holes in the ground. By using two sets of sensors in conjunction the robot can complete this task.

The first way that I approached this problem was by looking at the distinguishing aspects of a hole. The first one is that a hole has shadow on the inside of it. The other is depth.

Using collimated IR the robot can see the depth of the hole. Finding the darkness of the hole one must come up with another approach. I developed a way of switching resistance's on a stander Cadmium Sulfide Resistor (CDS Cell). The design is easy to build but the software gets quite complicated.

Along with the above mentioned sensor this autonomous agent has a complete sensor suite consisting of four analog IR sensors and bump sensors used for obstacle avoidance.

The "brain" of the agent is a memory and I/O expanded MC68HC11E9 evaluation board from Motorola. The software, including both drivers and behavior code is written entirely in the Image Craft 'C' compiler for the HC11. Includes a master process that occurs at a regular interval to insure consistency in sensor reading and actuation. A variety of behaviors are implemented in a subsumption architecture. The following simple behaviors are implemented: Hole Finding, Hole Filling, Hole Checking, collision avoidance, and Hole Positioning.

Introduction

During war time airfields may be bombed and suffer damage. People would need to put their lives in jeopardy to go out and repair these runways. The main task of the Dickens Rover will be to go out and find those holes and fill them with sand.

The basic problems with this design is can we answer the problem, "Is there a hole". There will also be the normal amount of avoidance behaviors that are part of any robot and a return to base feature that is still in the brainstorming phase. The hope of this project is to build a robot that can go out and do many tasks that would otherwise cause human fatalities.

There are four main behaviors that are involved with the rover. The first one is collision avoidance. This is a normal behavior that is stated in the syllabus of the class. Well let's be honest, it would be a pretty useless although funny robot without this behavior.

The second behavior is "Find the Hole". This behavior will have the robot roving around looking for holes in the ground. If it sees a hole it will then go to the next behavior "Fill the Hole". The "Fill the Hole" behavior will have the robot dump sand in the hole and then try and smooth out the sand.

The final behavior is "Return to Base". This behavior will be used to have the robot return after it has run out of sand, more about this later.

The platform of the Dickens Rover is a combination of different things. The robot's base is from a Tonka Bulldozer. It has a front servo that will control the blade on the bulldozer. This will allow the robot to raise and lower the blade to do different tasks. There are also two motors that will drive the right and left treads that are already part of the bulldozer. I am going to use the leads off of those motors/servos and hook them into the HC11 board in some way.

The top of the bulldozer has been removed and I have placed a piece of wood there as a platform. This is where the Hc11 board will go. I will also place some type of container in the front of the bulldozer to hold the sand. This container may be half of a two-liter soda bottle or something to that effect.

Most of the motors/servos are already in place. One place that I will need to add an extra servo is when the robot goes to "Fill the Hole". I am going to add a servo to control the flow of the sand out of the previously mentioned container. When the robot "Finds a Hole" it will then lift some type of door to allow the sand to flow out the container. After a fix period of time the robot will shut off this valve and spread the sand with its blade.

There are several types of sensors that I plan to use. I am going to use IR sensors and bump sensors to do obstacle avoidance. I also plan to have CDS sensors for doing hole detection. It is my hope that I can use the ambient light to detect the holes in the ground. However, since when you smell smoke you still want to be able to see the fire, I plan to use "End of the World" sensors to give my robot a little more accuracy.

Since the robot will need to know when it is out of sand I am going to have some type of pressure sensor to tell when the container of sand is empty. The sensor will most likely go under the storage device. The robot will then need some way of “Finding Base”. I have not yet decided on what to do for this and since it will be one of the last behaviors so I will think about it and let you know at a later date.

The Physical

Overview of the Physical

As a great man once said "A rock is the greatest computer in the world, the I/O is just poorly designed"[1]. The rover is not the greatest computer in the world but it has an A/D port and cool features that make it come alive.

Parts of the Overall Robot.

Power source:

Since the motors of the rover were already set to run off of another 6V Battery I decided that the Rover should have two separate power sources. The first one is the standard 12V supply that is given out in class. I used eight 1.5 V AA Nickel Cadmium rechargeable batteries. This set of batteries was the main power for the processor and the on board 32 K of Ram.

The other battery is a New Bright 6.0 V Nickel Cadmium rechargeable battery. The only job for this powerhouse was to power the motor driver circuit and the motors for the treads of the robot.

Motors:

The bulldozer that I purchased for this project already had the motors installed so I saw no need to change them out. The stall current was .2A which would assure that the motors will not blow up if they are stressed.

I also had to build a motor driver circuit for the motors. It was a design that I received from Scott Jantz. I used a 5V DPDT relay and two 2n2222a npn transistors to build a H-Bridge circuit that will handle any surge.

[1] Dr Kenneth Doty, IMDL lecture (see I was paying attention)

Servos:

There is an Aristo Craft Tracker Servo used to open and close a door on the front of the bulldozer. It is attached to the 12V supply at the point where it is known to be between 5-7V. The servo use used implemented by using routines from the servo.c and servo.icb libraries in Interactive C.

Sensors:

Inferred Detectors (IR)

Dickens has 4 analog-modified Sharp IR sensors where there are two mounted on the front and two others mounted on the rear. These pick up reflections from 40-KHz modulated IR LED's mounted robot in strategic positions. The four LED's are connected in series and driven from a latch at memory location 0x7000 on the ME11.

This sensor arrangement provides necessary information about nearby objects critical to the implementation of a collision avoidance behavior. This is the method used throughout the EEL 5666 class, and has proven quite successful. Given reasonable constraints on the size of the objects (must be higher than the bumper of the robot, and of significant size) and the reflective properties of the objects, the agent is capable of detecting obstacles with more than enough time to prevent collisions.

Bump Sensors

There are two bump sensors on the front of the robot. They are pulled high and go low when they bump some thing.

Cadmium Sulfide Cells (CDS)

While this sensor is not a new one for EEL5666 - Intelligent Machine Design Lab the way in which it was implemented was new and innovated even if I do say so myself

A normal CDS cell has two main properties

- Use Voltage to see what the sensor is actually doing. Use R[2] to regulate the range of the sensor
- As Light Levels increase the CDS cell losses it's resistively

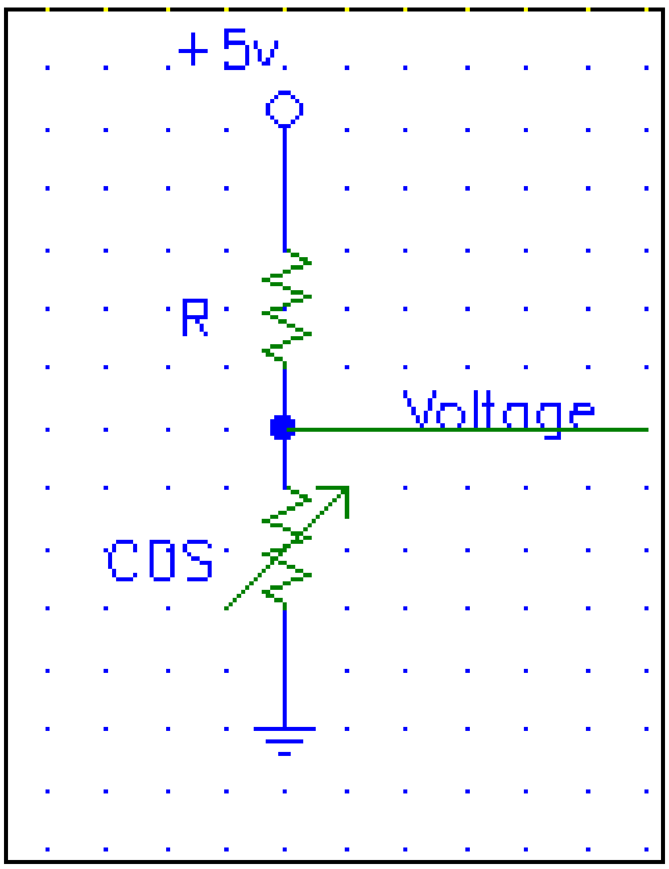


Figure 1

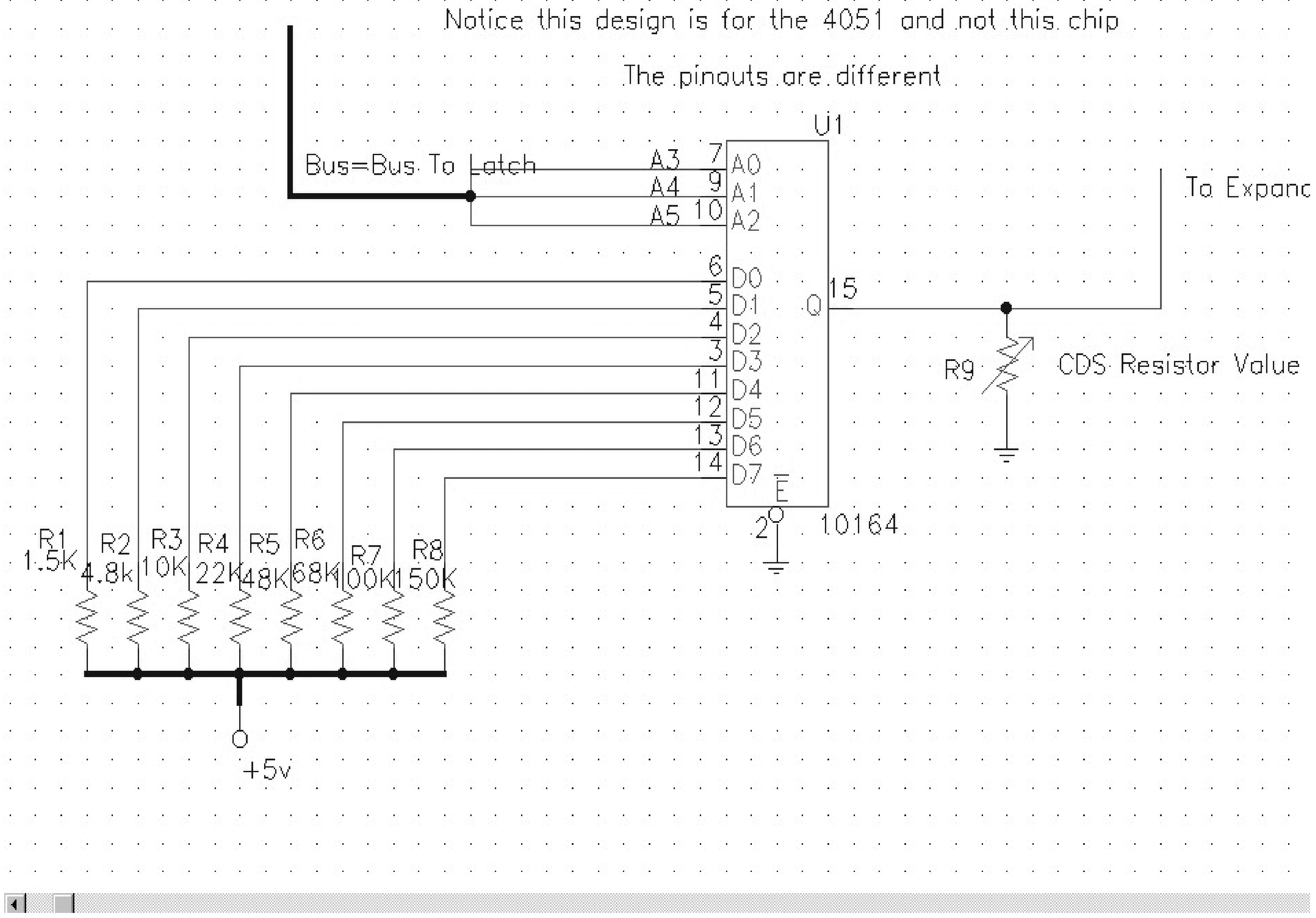
This sensor basically works as a voltage divider where the voltage will be VDD-Voltage drop over R. This in it's self is a very useful device when using the Motorola 68HC11

[2] See Figure 1

board. You can take that voltage and send it to one of the analog pins in port E on the board. I have decided to use pin 50 connected by a another analog multiplexer to read the voltage change. Now there are certain problems towards this approach. The resistance of the CDS cell has a very large swing in resistance. This swing makes it very easy to saturate the sensor so that it is effectively blind. The range of the cell would be any where from:

- Low end 1.5 KΩ
- Normal was about 58 KΩ
- High end 160 KΩ

The next 5 figures illustrate the behavior of this sensor with varying values for R and stable values of resistance of the CDS.



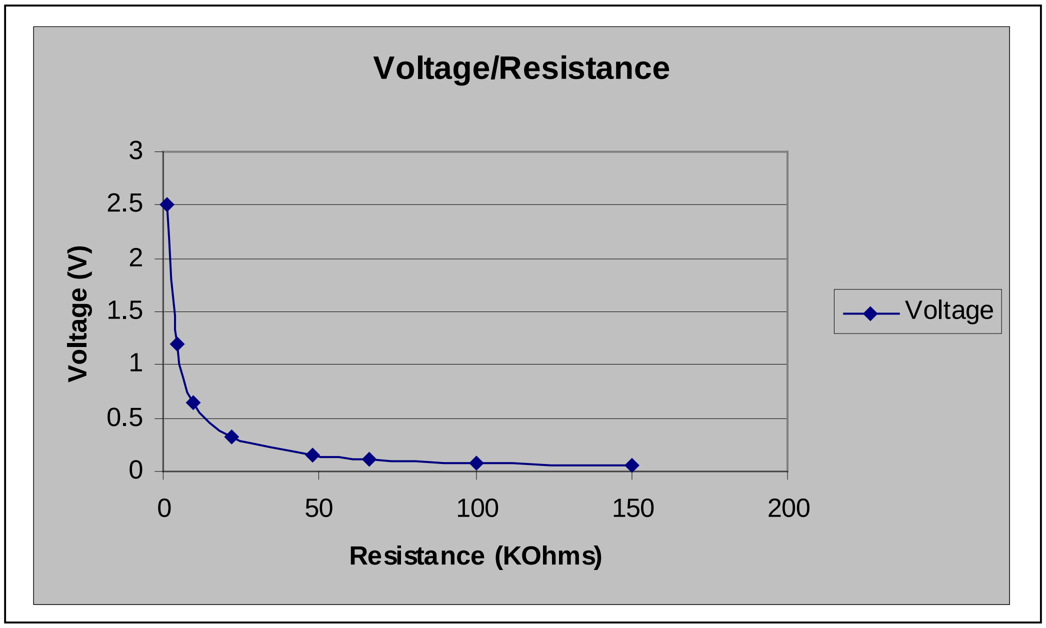


Figure 2, Saturation Findings at 1.5 KΩ

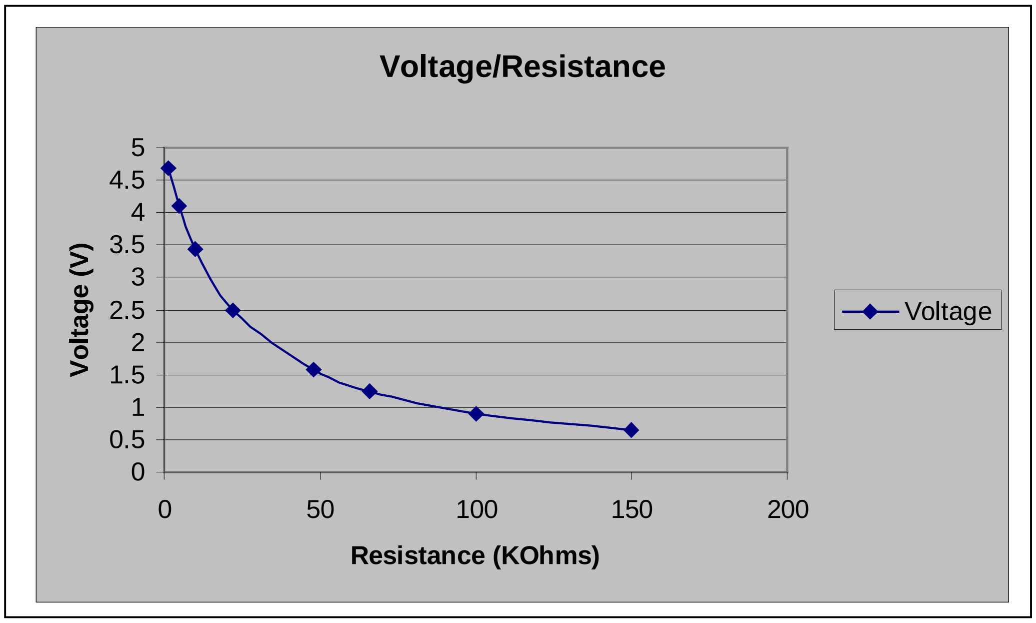


Figure 3, Saturation Findings at 22KΩ

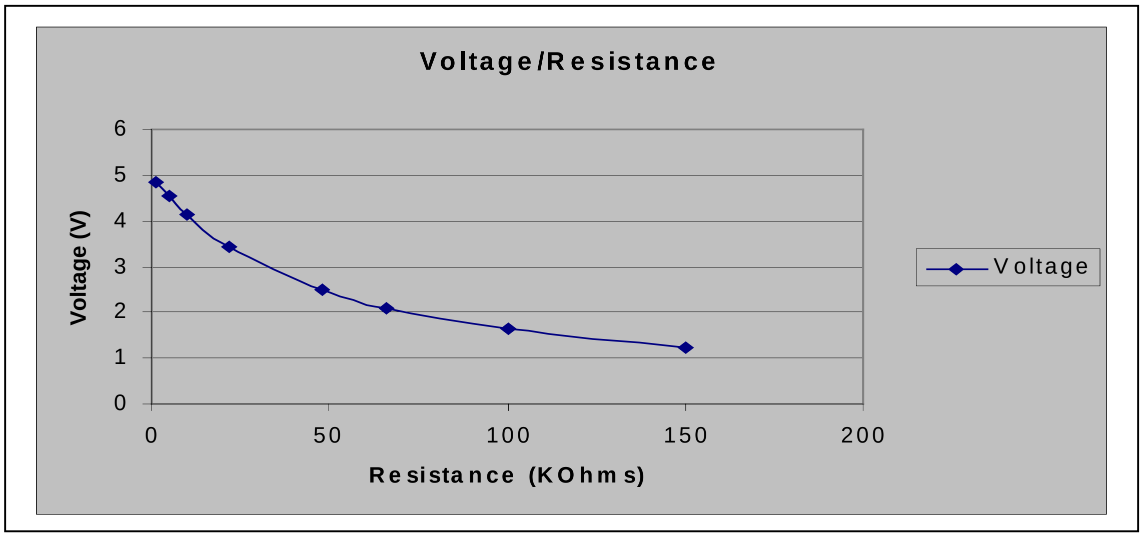


Figure 4, Saturation Findings at 48KΩ

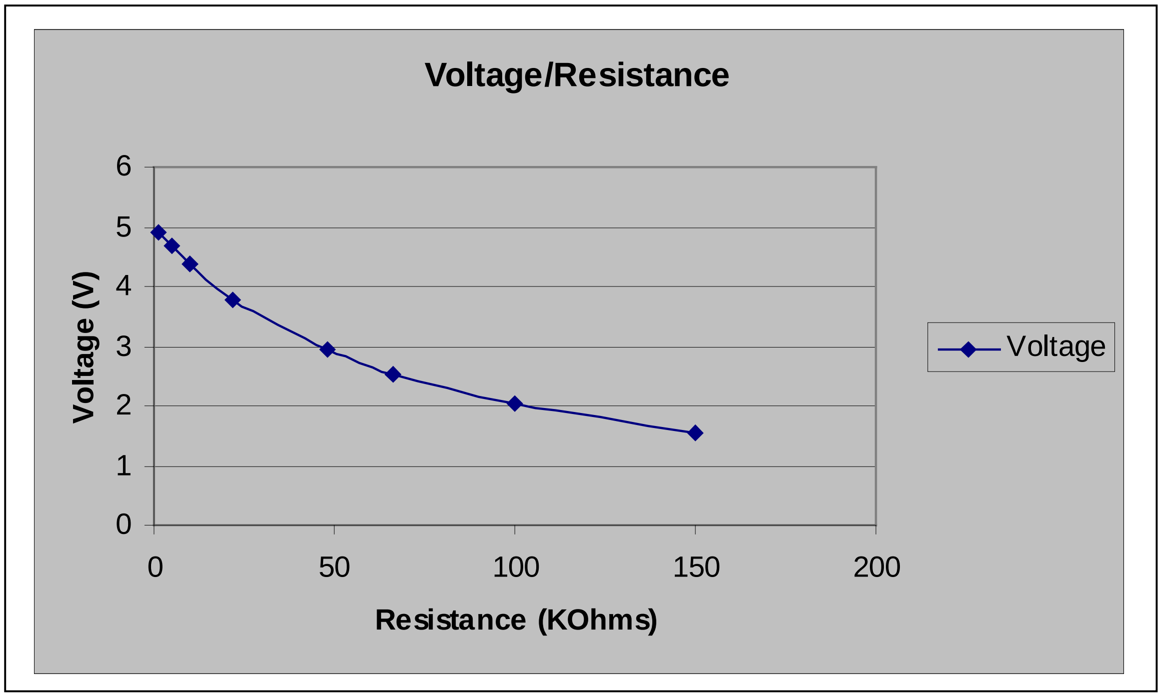


Figure5, Saturation Findings at 68KΩ

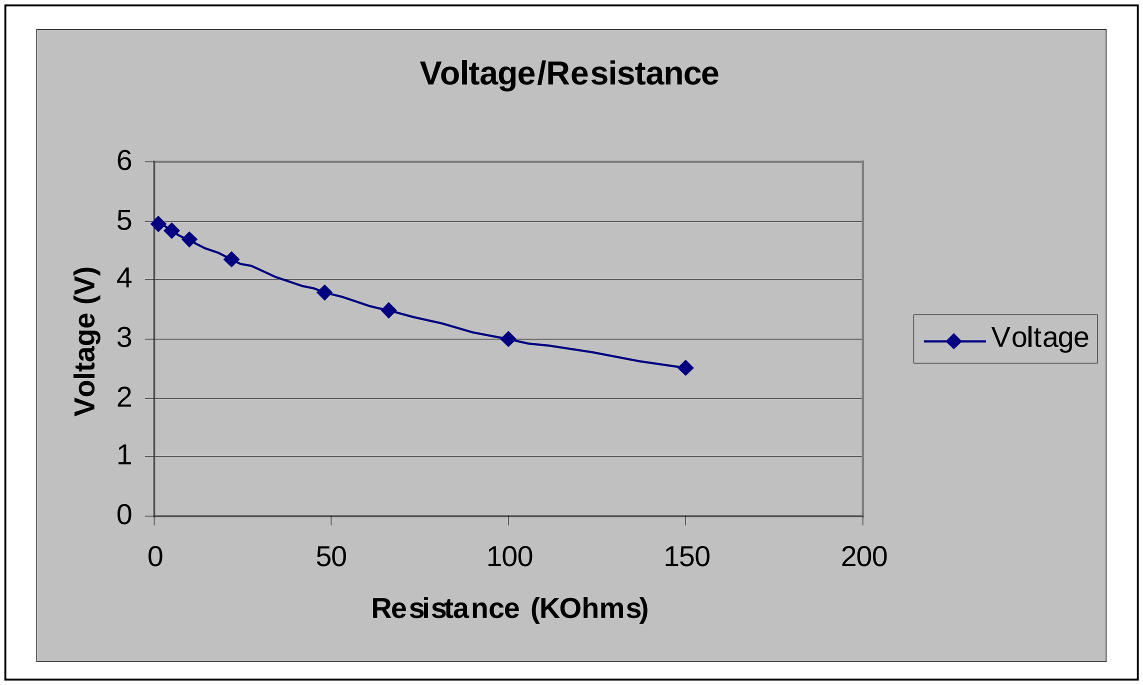


Figure 6: Behavior at 150 K Ω.

Figure 6: Behavior at 150 K Ω.

In Table 1 you can see some of the experimentation that occurred when introducing different amounts of light to the sensor.

Table 1:Observed Effects of CDS Resistor

Resistance (R)	Dark	Normal	Bright
1.5 KΩ	Saturation	Some Saturation	OK
6.8 KΩ	Saturation	Less Saturation	OK
10 KΩ	Saturation	Can start to use	Starting to saturate
22 KΩ	Some Saturation	OK	Saturated
48 KΩ	Can Start to use	OK	Saturated
68 KΩ	Ok	Starts to Saturate	Saturated
100 KΩ	Ok	Saturated	Saturated
150 KΩ	Ok	Saturated	Saturated

The above figure contains the basic design of the CDS sensor. The chip used above is not the same mux that I am using to make the sensor array. I am using the 74HC4051 analog mux. There is a slight amount of resistance to the mux but that should not effect the sensor too much. This sensors basic design is a resistor network that can be switched to select different values of resistance.

The software design of this type of sensor is a more important part of this robot then the actual hardware design.

Flow Chart

Figure 8: Block diagram of code

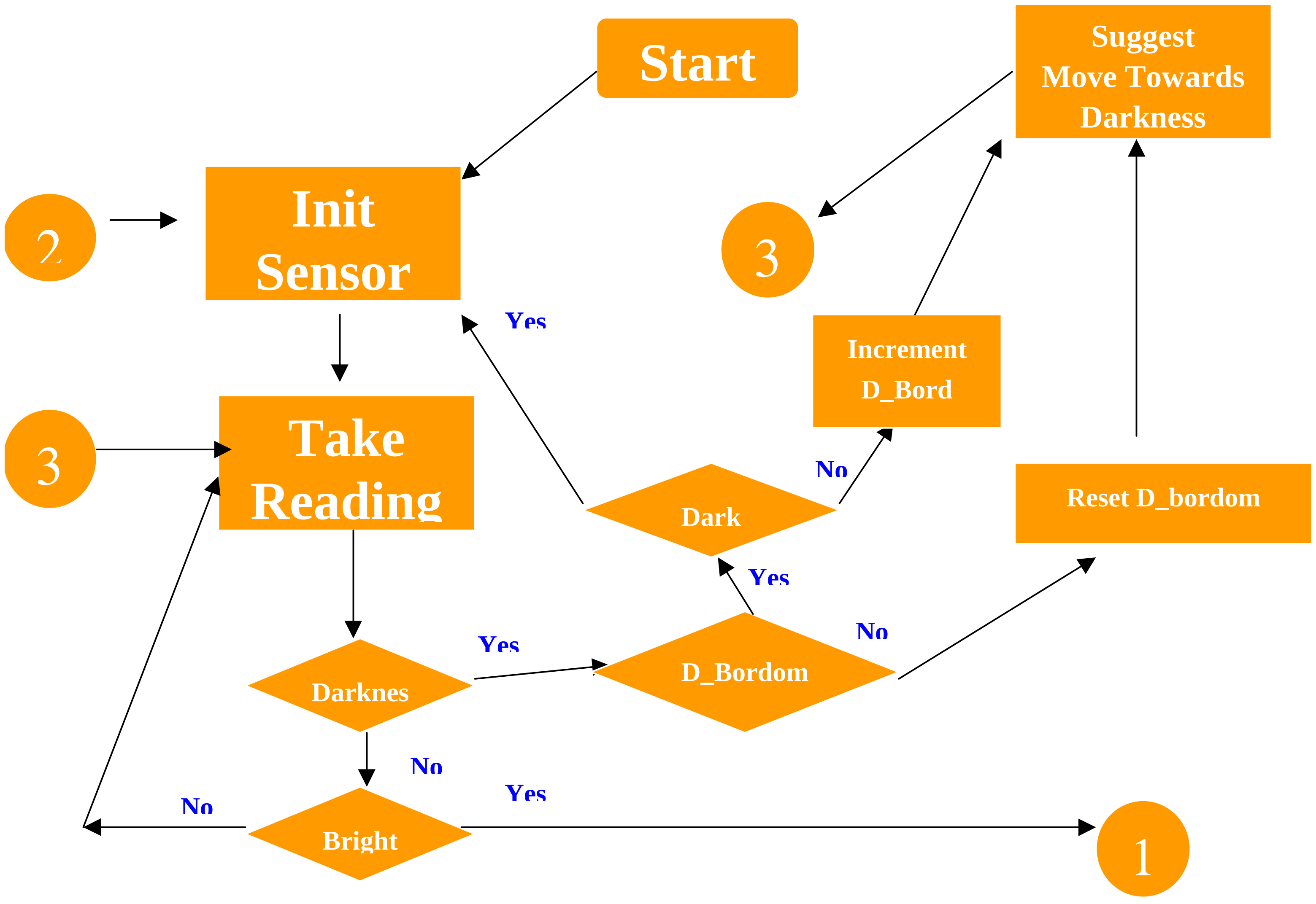
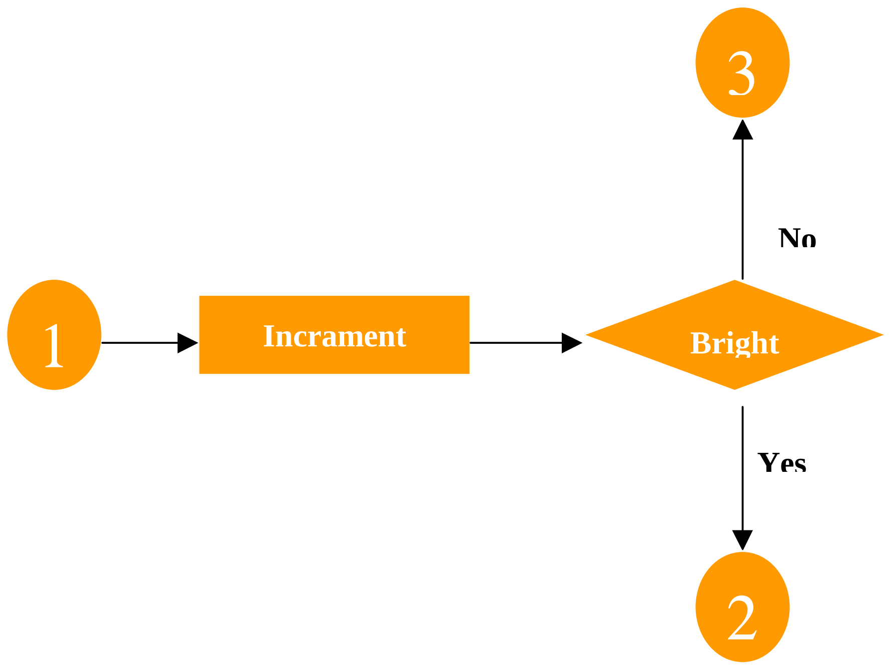


Figure 8 cont.



The beta code[3] for this type has several attributes that you will need to under stand how it works.

1. I have expended the I/O capabilities of my robot by memory mapping another mux at address Ox4000 using the lower three data bits to select which of the cds sensors to select.
2. I am using three different CDS sensors.
3. The version of IC that I am using does not support octal notation.

Collimated IR:

This is just another application of the IR sensor mentioned above. Instead of using a broad beam of 40kHz light the beam is tightened to a dot. The theory that was later proved in testing was that by using this method one could see a drop in the floor. The

[3] Appendix A

hypothisis was that the drop would have to go all the way down to 80% of the normal reading but it was later determined that it was only 95% (a 5% drop).

The Metaphysical

Overview

As in any entity there must not only be an existence but a purpose. The rover has a stated purpose and has behaviors to match. So there has to be some type of coding method that has to be used. I decided to implement the waterfall model when writing the code. This process is slightly longer then just a functional approach and found it to be effective for results but too long to implement for this class. I myself wish that I would have just used the simpler functional approach.

Behaviors

A number of simple behavior routines were written to take advantage of the sensors and actuators implemented on the agent.

Collision Avoidance

This behavior is the essential element of the robots self-preservation instinct. Too actually write this code I went walking on the North Lawn to see how I would look at the world if I could only see three feet or less in front of me. I realized that if I had another task on my mind I would go and avoid the obstacle and then continue. I decided to use hard line rules since they were simple to implement. I use the combination of this behavior and the dark finding behavior to determine direction.

Darkness Finding

This behavior is used to point the robot in the right direction. While you really will not see this part of the code work directly I have proven in the second demo of this year that the behavior or better yet, tendency is there for the robot to head towards darkness. This behavior uses the CDS cells.

Hole Finding

This is a routine that will actually see the change in depth of a hole. The routine is implemented by using the collimated IR. Once the routine sees a hole it returns a Boolean true to the calling function.

Hole Filling

This routine is called after the robot finds a hole. It use two other sub behaviors position and dump. I call this a sub behaviors because without one you can not have the other. The position of the robot takes the last set of data and traces back to the last known position of land. After it considers itself over the hole it then starts to dump material in too the hole using the servo routines.

Hole Checking/Flattening

After the robot fills the hole it goes ahead and smooth out the stuff around the hole. After that is competed it then checks the hole to see if it is full and then takes the proper action if it sees a hole still

Arbitration System

The robot stays in collision avoidance or darkness finding during most of its running time. This gives a preference for moving forward. If the bump sensors were to go off or it were to find a hole the other behaviors. I would like to make this more of a mathematical approach and I believe that it can evolve into that.

Conclusion

The development of Dickens Rover has reached the stage where it satisfies the criterion set forth in the EEL 5666 syllabus. The robot has a complement of 4 sensors: analog IR detectors, Bump sensors, collimated IR. Finally, four main behaviors have been successfully implemented on the platform: collision avoidance, darkness finding, hole filling, and finally hole checking.

A number of improvements could be made to the hardware of the robot. First, a more complete array of IR detector/emitter pairs should be placed on the robot and integrated into the behaviors. These should be mounted allow the robot to make better decisions when roving.

The code written for the agent is not yet fully mature. It could be improved in a number of ways. First, the motor control and collision avoidance routines could be improved to take advantage of the work done by Dr. Keith Doty and Scott Jantz in non-linear dynamics control of small autonomous agents. Further, the arbitration network could be improved by the integration of a neural. Also, implementing a simple local mapping algorithm would dramatically increase the sophistication of the agent's hole finding ability. Also the addition of a home base and a way that the robot would know that it was full could be an improvement.

Works Cited

[2] Drew Bellesar. Final Report: Hacking the Hero.
IMDL Final papers, http://www.mil.ufl.edu. 1996

[3] M68HC11 Reference Manual
Motorola 1991

[4] MC68HC11E9 Technical Data
Motorola 1991

[5] High-Speed CMOS Data
Motorola 1993

*** Special thanks to Ian A Arroyo for his help during crunch time. Also special thanks to the TA's of the class for putting up with me through the first ten weeks.**

Appendix A- Software

```
/* This is the main routine for the obstical avoidence routine

There are four behaviors that are involved with this automaktone:
they are
1) Obstical Avoidence
2) Find hole
3) Fill hole
4) Check to see if the hole is filed
There are 5 files that are included before the
arbatrator is added.
The files are Motor.c,obstical.c,finddark.c, findhole.c
common.c these will all be included in this file due
to failures of IC compiler.  In icc11 I would suggest
breaking up these packets*/
/*arbatrator constants*/
int expanded_IO  = 0x4000;
/*arbatrator variables*/
int motorr[5],motorl[5],urg[5]={0,0,0,0,0},dark_bord=0;
/*Arbatrator routines*/
/*Drive Motors use an Exponental approach to reving up the motors*/
void drivemotors(int left, int right, int urgency) {
  int i;
  lmotor += urgency*(left - lmotor)/100;
  rmotor += urgency*(right - rmotor)/100;
  motor(0,lmotor);
  motor(1,rmotor);
  sleep(0.05);
}
/*Reverse course is a routine that will allow the robot to back up if
there is trouble*/
void reverse_course(){
int i;
drivemotors(0,0,100);
for(i = 0; i<3; i++){
      if(irsensor[1] <  nthresh){
            drivemotors(-30,-90,32);
      }
      else{
            if(irsensor[2] <  nthresh){
                  drivemotors(-90,-30,32);
            }else{
                  drivemotors(-90,90,65);
        }}}}

/* obstical avoidence routines*/
/* Motor control/object avoidance
   by Thomas Cosenza

   Sensors:                  0    3

                       1    4 (Could not fit at2)

   Motors:          0          1

*/
```

```
/* constants */

/* Global Variables */
int irsensor[4]; /*IR Array */
int vfthresh = 105, lowthresh = 95;
int nthresh = 125, fthresh = 115;

/*masks */
int vnl = 0x80, nl = 0x40, fl = 0x20,vfl = 0x10;
int vnr = 0x8, nr = 0x4, fr = 0x2,vfr = 0x1;

/*IR will look at all of the IR sensors
and store in the IR sensor array*/

void ir() {
  int i;
  for (i=1;i<=4;i++) {
    irsensor[i-1] = (analog(i-1));
  }
  irsensor[2] = analog(4);
}

/*Much to the chagrin to one of the TA’s in the class I have decided
that I would use if then else statelments for the robot. Note ic does
not have case statement this is the reason why I did it this way*/
int read_front() {
int i,direction = 0;
ir();
/*check the right side*/
if(irsensor[3] > nthresh) {direction |= vnr;} /*case really close on the
right*/
else if( irsensor[3] <= nthresh && irsensor[3] >= fthresh) {direction
|=nr;} /*case that we are close */
     else if(irsensor[3] <= fthresh && irsensor[3] >= vfthresh)
{direction |= fr;} /*case that we can see something not close*/
        else if(irsensor[3] <= vfthresh && irsensor[3] >= lowthresh)
{direction |= vfr;} /*case unsure how close maybe nothing*/
             else if(irsensor[3] < lowthresh) {direction |= 0;} /*case
of nothing seen*/

/*check the left side*/
if(irsensor[0] > nthresh) {direction |= vnl;} /*case really close on the
right*/
else if( irsensor[0] <= nthresh && irsensor[0] >= fthresh) {direction
|=nl;} /*case that we are close */
     else if(irsensor[0] <= fthresh && irsensor[0] >= vfthresh)
{direction |= fl;} /*case that we can see something not close*/
        else if(irsensor[0] <= vfthresh && irsensor[0] >= lowthresh)
{direction |= vfl;} /*case unsure how close maybe nothing*/
             else if(irsensor[0] < lowthresh) {direction |= 0;} /*case
of nothing seen*/
return(direction);
}

/* changes speed to required values for lmotor and rmotor. Change is
   more gradual when urgency is smaller. */

void object_avoid_init(){
  poke(0x7000,0xff); /*Turn on IR emmiters */
}
void object_avoid_behavior () {
  int i, sensor, next=100;
    sensor = read_front(); /*Read the IR Sensors */
      if(sensor == 0x88)/*Danger to close*/
      {
         drivemotors(0,0,100);
         motorl[0] = -101;
         motorr[0] = -101;
         urg[0]    = 100;
         return;
      }
      if((sensor & vnl)||(sensor == 0x44))/*Case that we are near to
colition on the left*/
      {
         motorl[0] = 100;
         motorr[0] = -35;
         urg[0]    = 65;
         return;
      }
      if(sensor & vnr)/*Case that we are near to colition on the right*/
      {
         motorl[0] = -35;
         motorr[0] = 100;
         urg[0]    = 65;
         return;
      }
      if(sensor & nl)/*Case that we are near on the left*/
     {
         motorl[0] = 100;
         motorr[0] = 10;
         urg[0]    = 45;
         return;
      }
      if(sensor & nr)/*ase that we are near on the right*/
     {
         motorl[0] = 10;
         motorr[0] = 100;
         urg[0]    = 45;
         return;
      }
      if(sensor & fl)/*Case that we see something on the left*/
     {
         motorl[0] = 95;
         motorr[0] = 60;
         urg[0]    = 23;
         return;
      }

      if(sensor & fr)/*Case that we see something on the left*/
       {
         motorl[0] = 60;
         motorr[0] = 95;
         urg[0]    = 23;
         return;
      }
         urg[0]    = 10;
         motorr[0] = 75;
         motorl[0] = 75;
}

/*CDS Routines */
/* This is code that will control the CDS cells */
/* this is code version 1-2 level c*/
/*  Bits 00   000    000
       NA   Sensor Selector

Selector
000 = Right Sensor
010 = Left Sensor
001 = Center

Sensor
000 = 1.5KOhm
001 = 6.5KOhm
010 = 10 KOhm
011 = 22 KOhm
100 = 46 KOhm
101 = 68 KOhm
110 = 100KOhm
111 = 150KOhm

*/

/*Cds Sesors

Front
2 1 0
Rear

*/

/*Constants*/
int shift = 0x8;/*shifting property of the resistornet*/
/*Vars Global*/
int minrange[3]; /* minimum range */
int maxrange[3];/* maximum range */
int cds_value[3];/*Cds resistor */
int current_value[3];/*current value of the sensor*/
int motordelta_left;
int motordelta_right;
int lmotor=0,rmotor=0;
int light_bordom = 0;

/* min takes two integers and returns the lesser of the two unless they
are
equal then it will return the second value*/

int min( int a, int b)
{
 if(a<b){
  return a;
 }
return b;
}
```

```
int max( int a, int b)
{
 if(a>b){
  return a;
 }
return b;
}

/* Init values takes no arguments and initilizes all global variables*/
void init_cds()
{
int current = 0;
int value;
while((current & 3) != 3){/* look through all values of the cds
sensors*/
  poke(expanded_IO,current);
  value = analog(7); /* Get value at the sensor*/
  if(value > 150){  /*Tolerence is from 125- 175 for the start*/
      if((current & 0x38)==0x38) /*Check to see if it is really dark */
      {   cds_value[current&3]= 0x7; /*put the value of the resistor in
the slot*/
          current &= 3;/*clear the reistor value*/
          current++;/*Increment the lower octet which is the cds
selector*/
      }
      else{
         current += shift; /*go to the next highest resistor value*/
        }
      }
  else{
       maxrange[current&3] = (int)((float)(255-value)*.2 +
(float)value);
       minrange[current&3] = (int)((float)value -(float)value * .2);
       cds_value[current&3] = (current&0x38)>>3;
       current++; /* Go to the next Sensor */
       current = current & 0x03; /*Clear the resistor value*/
       }
}
}

/*read cds pokes the location 4000 with the to
switch to each of the cds cell of the given sensor*/
int readcds(int sensor)
{poke(expanded_IO,((cds_value[sensor]<<3)|sensor));/*or the sensor with
the current resistor found by the cds init routine*/
return(analog(7));/*return the value of that sensor*/
}


int find_the_dark()
{
int direction;
 if(maxrange[0] < current_value[0] &&
    maxrange[1] < current_value[1] &&
    maxrange[2] < current_value[2]){
    motordelta_left=100;
    motordelta_right=100;
    light_bordom++;
   return(-1);
 }

light_bordom=0;
 if(minrange[0]<=current_value[0] && maxrange[0]>=current_value[0] &&
    minrange[1]<=current_value[1] && maxrange[1]>=current_value[1] &&
    minrange[2]<=current_value[2] && maxrange[2]>=current_value[2])
 {
  motordelta_left = 75;
  motordelta_right = 75;
  return(1);
 }
 if(maxrange[0] < current_value[0]){
    if(maxrange[1] < current_value[1]){
       direction=decide_direction(0,1);
       if(direction == -1){
             motordelta_left = 0;
             motordelta_right = 100;
             return(1);
             }
       if(direction == 0){
             motordelta_left = 100;
             motordelta_right = 100;
             return(1);
             }
       if(direction == 1){
             motordelta_left = 100;
             motordelta_right = 0;
             return(1);
             }
      }
     else if(maxrange[2] < current_value[2])
         {
             motordelta_left = 100;
             motordelta_right = 100;
             return(1);
          }
          else {
             motordelta_left = 0;
             motordelta_right = 100;
             return(1);
            }}

 if(maxrange[1] < current_value[1]){
   if(maxrange[2] < current_value[2]){
      direction=decide_direction(2,1);
      if(direction ==-1){
             motordelta_left = 10;
             motordelta_right = 100;
             return(1);
             }
     if(direction == 0){
             motordelta_left = 100;
             motordelta_right = 100;
             return(1);
             }
       if(direction == 1){
             motordelta_left = 100;
             motordelta_right = 10;
             return(1);
             }}
     else{
      motordelta_left = 100;
      motordelta_right = 100;
      return(1);
      }
 }
   if(maxrange[2] < current_value[2]){
     motordelta_left = 100;
     motordelta_right = 0;
     return(1);
   }
motordelta_left = 0;
motordelta_right = 0;
return(0);
}


int decide_direction(int sena,int senb){
 int need;
 need = find_change((float)((255-current_value[sena])/(255-
maxrange[sena])),(float)((255-current_value[senb])/(255-
maxrange[senb])));
 return(need);
}

int find_change( float delta_one, float delta_two)
{
  if(delta_one == delta_two)
  {
   return 0;
  }
  if(delta_one < delta_two)
  {
   return 1;
  }
  return -1;
}

void darkness_behavior(){
int i;
int flag;

for(i=0;i<3;i++){/*get all cds values*/
    current_value[i]=readcds(i);
  }
flag=find_the_dark();
if(flag==1){
  motorr[1] = motordelta_right;
  motorl[1] = motordelta_left;
  urg[1]    = 75;
  return;
}
```

```
if(flag == -1)
   {
   if(light_bordom >=1000)
     {
       init_cds();/*reinitialise cds values*/
       light_bordom = 0;
     }
   }
    motorr[1] = 0;
    motorl[1] = 0;
    urg[1]    = 0;

 return;
}

/*Bump Sensor */
/* this is the detection algorithm for a bump detection
I am taking the involentary resonse method to this */
/* bump sensors are wired to the analog ports 5 & 6*/

int bumped_something(){
if(analog(5) > 100 && analog(6) > 100)/*check to see if nothing is hit*/
{
  motorl[3]=motorr[3]=0;/*no danger no urgency*/
  urg[3] = 0;
  return(0);
}

return(1);
}


/* Robot Darkness Code Level 1.1 */
/* Definitions*/
int shift_dark=3;
int mask =7;

/*Global Variables*/
int dark_tol[3];

/* Check to see what the current depth is */
void init_depth(){
int i;
for(i=0;i<3;i++)
{
  poke(expanded_IO,i+shift_dark);
  dark_tol[i] = (int)((float)(analog(7)-80)*.95 + 80.0);
}
}

/*Take a reading of the current depth*/

int read_depth(){
int i,reading=0;

for(i=0;i<3;i++){
  poke(expanded_IO,i+shift_dark);
  if(analog(7)<dark_tol[i]){
    reading = reading | 1 << (2-i);
  }
}
return(reading);
}

/*This is the Stub for depth perception*/
int foundhole(){
return(read_depth());
}



/*arbtrator main*/

void init_robot(){
object_avoid_init();
init_cds();
init_depth();
}

void position(){
while(foundhole()){
drivemotors(-100,-100,10);
}
drivemotors(0,0,100);
}


void wait_for_it(int stop){
   int i;
   for(i=0;i<stop;i++){
   i=i+0;
   }
 }

/*Dump returns no values and opens the door and dumps the pellets.
 You will need to load lib_rw10.c servo.icb servo.c then this code*/
void dump(){
int close = 180;
int open = 0;
servo_on();
servo_deg((float)open);
wait_for_it(5000);
servo_deg((float)close);
wait_for_it(1000);
servo_off();
}

void flatten(){
int i;
for(i=0;i<10;i++){
  drivemotors(10,100,45);
  sleep(.2);
  }
for(i=0;i<10;i++){
  drivemotors(-10,-100,45);
  sleep(.2);
  }
for(i=0;i<10;i++){
  drivemotors(100,10,45);
  sleep(.2);
  }
for(i=0;i<10;i++){
  drivemotors(-100,-10,45);
  sleep(.2);
  }
}
/*rev takes the motors and gets them up to speed gently*/
void rev(int left,int right,int urg,int speed){
 int i;
  for(i=0; i< speed;i++){
       drivemotors(left,right,urg);
       wait_for_it(1000);
       }
       drivemotors(left,right,100);
}

void fillhole(int reading){
int i;
      if((reading & 2) == 0){
          if(reading & 1){
                  rev(50,0,35,5);
                  drivemotors(50,0,100);
                  sleep(1.5);
            }
            else{
                  rev(0,50,35,5);
                  drivemotors(0,50,100);
                  sleep(1.5);
            }
       }
       rev(0,0,30,4);
       position();
       dump();
       flatten();
 }


void main(){
int priority[4]={100,75,300,250};/*Array for priority
(Obs,CDS,Bump,Hole);*/
int Right,Left,Urg,denom,i,mask;

init_robot();
while(1){
      if(bumped_something()){
/*                reverse_course();
            sleep(1.0);
            write("Bumped\n");
            put_char(13);*/
      }
      else{
            if(mask = foundhole()){/*add found hole here*/
                  fillhole(mask);
            }
            else{
                  Right = Left = Urg = 0;
                  object_avoid_behavior();
                  darkness_behavior();
/*This is a very cheesy */
                  if(urg[0]*priority[0]>urg[1]*priority[1]){
                   Right = motorr[0];
                   Left = motorl[0];
                   }
                   else{
                   Right = motorr[0];
                   Left = motorl[1];
                   }
                  if(urg[0]*priority[0]>priority[1]*urg[1]){
                     Urg = urg[0];
                     }
                  else{
                     Urg=urg[1];
                     }
                  drivemotors(Left,Right,Urg);
                sleep(1.0);
                if(Left < 0 && Right <0 ){
                   sleep(2.0);
                   }
                }}}}
```