IMDL Final Report

By

Tim McGraw

December 09, 1998

Table of Contents

Abstract

The robot, Heather, was designed to play a game of croquet. The design consisted of several phases:

- Integrated System: the processor and supporting electronics capable of running the game-playing algorithms and reading the necessary sensors.

- Mobile Platform: the mechanical platform capable of moving about the field of play, and striking the ball.

- Sensors: information gathered from the environment needs to include ball detection, wicket detection, and collision detection.

- Behaviors: the software written to command the platform based on internal state and sensor readings.

Executive Summary

Heather was designed to play croquet. Electronically, the system consists of a 68HC11 EVBU board with an ME11 expansion. The mechanical platform is a two-wheel drive system with a servo actuator mounted underneath to strike the ball.

Infrared was used for ball and wicket detection. An IR cannon is mounted on the platform and the intensity of reflected IR is analyzed to detect the ball. The wicket has an IR cannon mounted on top, and cross-view IR sensors on top of the platform are read to determine which direction to shoot the ball. A wheel-encoder system was designed to count rotations of the platform.

The game-playing behavior was developed in four steps: find ball, approach ball, find wicket, and strike ball. Additional processes run in the background to read the sensors and provide motor control.

Introduction

The goal of the robot is to play a game of croquet. The object of the game is to strike a ball through a target (called the wicket) in as few strokes as possible. This involves solving several problems:

- Possible long-range ball detection: What sensors to use? How should the data be analyzed? How should the sensors be physically arranged?

- Long-range wicket detection: Same as above.

- Ball actuation: What mechanism should be used to strike the ball?

- General mobile platform concerns: How to avoid obstacles and detect collisions?

The design of the system will be presented in the following report in several sections, each concentrating on one phase of the implementation: the electronic architecture of the system, the physical platform, the actuators used, the sensors used, and the behaviors programmed into the robot. The complete code will appear in the Appendix.

Integrated System

A complete functional block diagram of Heather's integrated system appears in Figure 1. The three main components are the 68HC11 EVBU board, the ME11 expansion board, and a group of expanded input and output ports. Inputs to each subsystem are shown on the left, and outputs on the right.

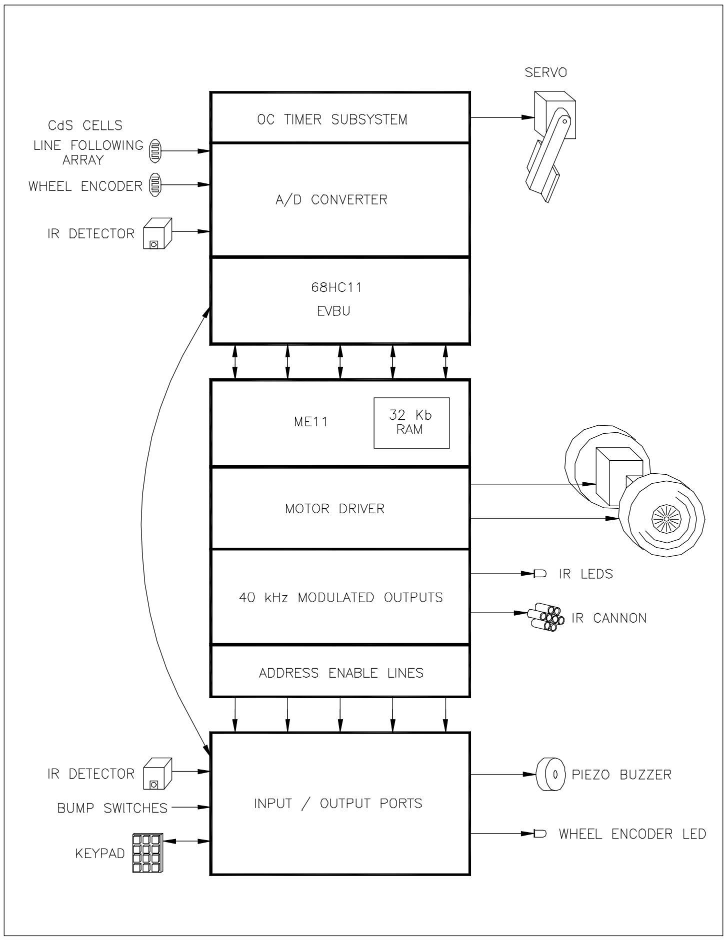


Figure1: Integrated System.

All but one of the connections to the 6811 are analog inputs. These consist of CdS cells and analog IR sensors. The line following array came about as an attempt to match the left and right motors. This feature never functioned as desired, and will not be further discussed in the body of this report, except for the line-following and CdS cell self-calibration code in the appendix. The lone output of the 6811 is used as the control signal for the servo actuator.

The 6811 is mated to the ME11 board. This board provides ample RAM, motor drivers, 40kHz modulated outputs, and additional address-select lines which greatly simplify expanded memory-mapped I/O.

The expanded inputs provide ports for a digital IR sensor, several bump switches. The outputs drive a piezo buzzer, and the wheel encoder LED. Inputs and outputs are required to do the raster scan of the keypad.

This integrated system meets the locomotive, processing/memory, and I/O needs of the robot with room for future expansion.

Mobile Platform

The mobile platform design chosen for Heather is shown in Figure 2.

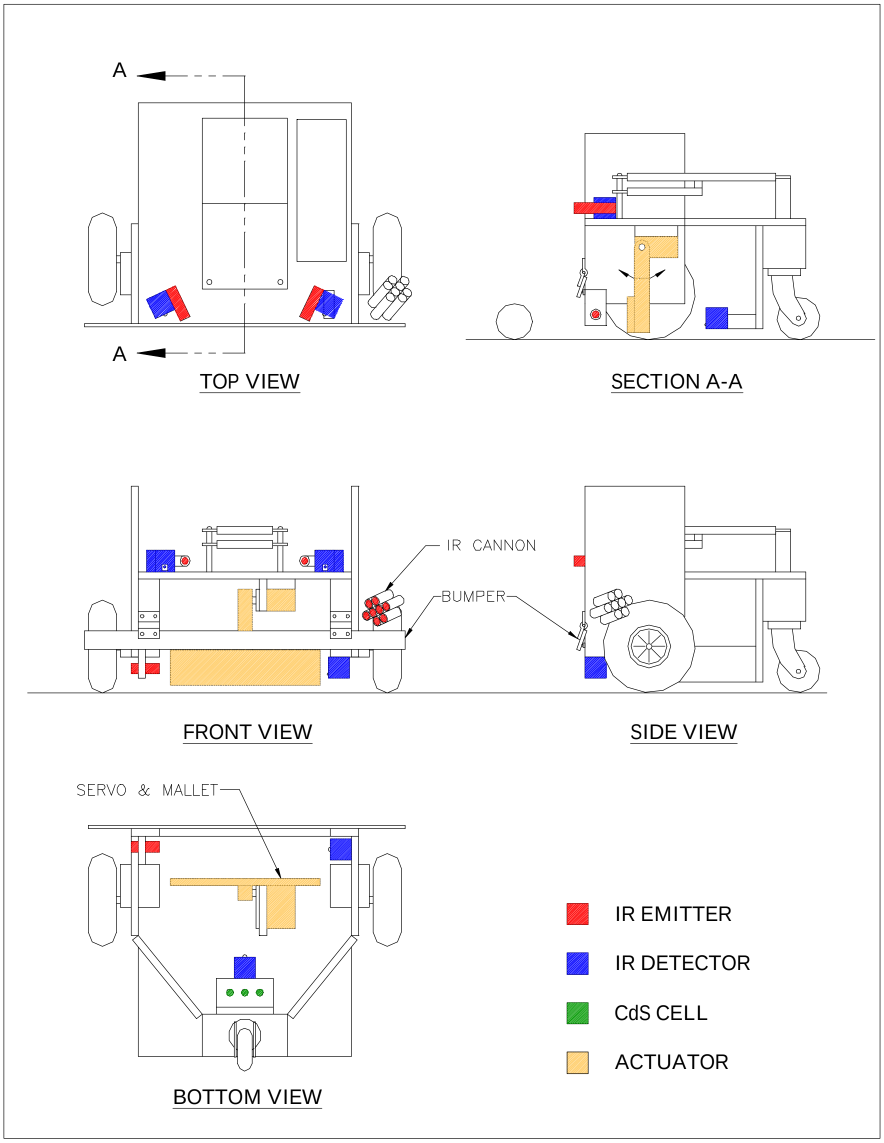


Figure 2: Heather Mobile Platform

In Section A-A, the ball is shown in front of the platform to show the relation between sensor placement and ball size. At the rear of the platform is an IR sensor at ball-level. This detector will be used to located the ball as it is illuminated by the IR cannon. In this position the detector is shielded from ambient light. Another IR detector, with a matching emitter, is placed at the front of the platform. This detects when the platform has reached the ball, and if the ball is escaping from the platform while the beacon is being searched for. When the ball breaks the beam, it is correctly placed for striking by the actuator. The bumper is placed so that the ball will just barely pass underneath.

A pair of detectors on top of the platform are arranged in cross-view fashion for beacon detection. Not shown in the figure is the wheel encoder, which is located behind the right-hand wheel.

The material for the body of the platform is ¼" oak plywood, and is nearly too heavy for the servos to support. This should be taken into account in any future work.

Actuation

The only type of actuator used in the implementation of the robot are servo motors. Two have been hacked to act as gearhead DC motors. The third is the actuator used to strike the ball and is mounted beneath the platform. This is connected to full battery voltage for power, and the 6811 output compare subsystem for control.

The selected servos are Futaba S-148's (38 in*oz at stall). This is adequate for locomotion, but is not fast enough for ball actuation over carpet. A high-speed servo, or some intermediate mechanism will be required in the future for longer range.

Sensors

Only three types of sensors were required to achieve the goals of Heather. These are IR detectors (analog and digital), CdS cells, and bump switches.

- The bump switches are used for collision detection, and don't contribute to game play.
- The IR detectors are used to detect IR emitted from the beacon, and reflected from the ball.
- The CdS cells are used for line-following, and one is integrated into a wheel-encoding device.

Although the wheel encoder is not fundamental to my robot's purpose, it can solve several problems:

1. Detect motor stall. Small obstacles in the path of the robot can pass under the bumper and jam the motors, resulting in high motor currents. This is detrimental to the motor, and to battery life.
2. Allow motor commands to use "real-world" units. The rate of drive motor rotation depends on battery voltage and the surface being driven on. Although the wheel encoder cannot account for slippage, it can account for battery

voltage, and surface incline. This is more desirable than a time-based motor command.

I propose to solve these problems with a low cost, and easily mounted CdS cell system.

The CdS cell is simply a variable resistor that is controlled by light. The greater the intensity of incident light, the lower the resistance. The only circuitry required to use the CdS cell is a voltage divider (between 5V and ground) and an A/D converter tapped to the center of the divider.

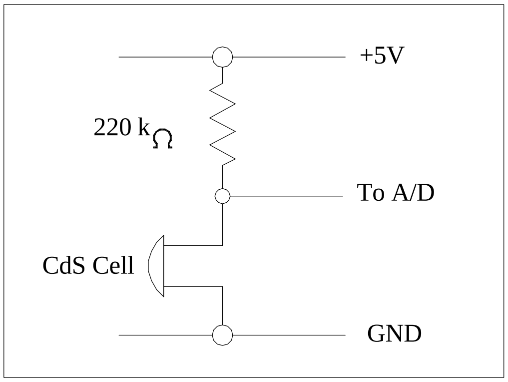


Figure 3: The Voltage Divider Circuit

One foreseeable problem associated with this application of the CdS cell is that the cell's voltage is not only a function of the intensity of light, but also of time. The resistance across the leads of the device may not change as quickly as the light intensity. This limits the number, and width of the stripes used to encode the wheel.

The effect of ambient light level may limit the effectiveness of the cell. To minimize this effect, the device was mounted in a block of plywood thick enough to partially shade the cell as shown in Figure 4.

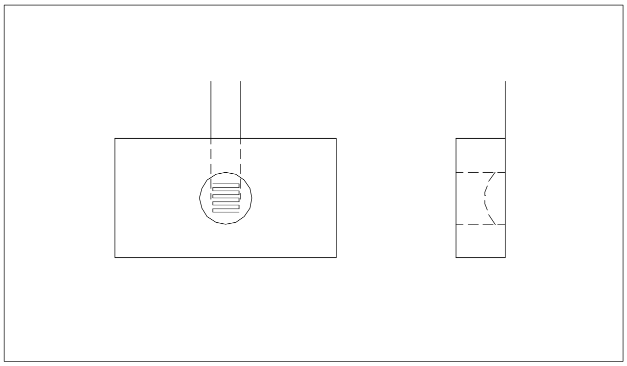


Figure 4: Mounted CdS Cell

Behaviors

The highest level behavior is, of course, game-playing. This is broken into four phases as shown in Figure 5 below.

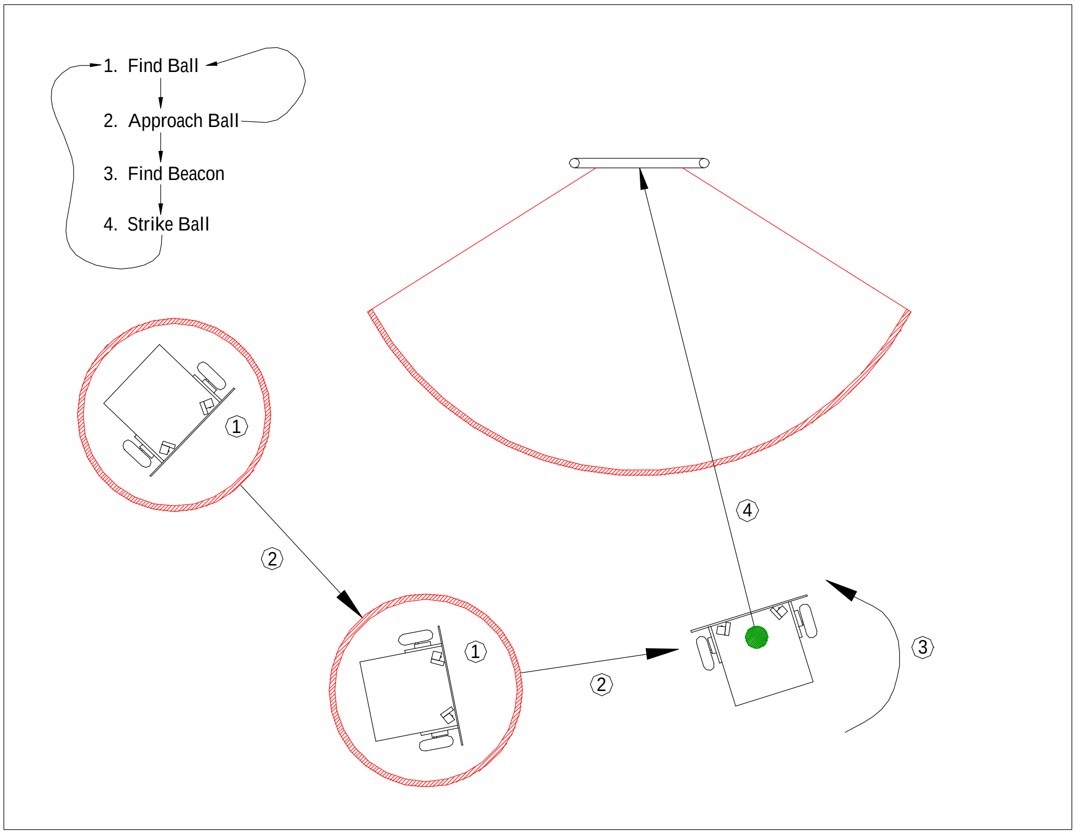


Figure 5: Game playing Behavior.

The ball finding algorithm (detailed below) and approach ball behaviors are repeated until the BALL_FOUND flag is set. This flag is set in the approach ball behavior when the ball detecting beam is broken. If no ball is found, this behavior times out after 3 seconds. The beacon finding (detailed below) and ball striking behaviors are executed sequentially, then the ball finding phase is entered again.

The ball finding behavior consists of several concurrent processes as shown below in Figure 6. The primary behavior consists of two phases.

- Find the maximum reflected IR in one revolution of the platform. This requires a concurrent process that reads the wheel encoder to determine when one rotation has been achieved.

- Continue rotating until an IR reading close to the maximum has been found. To ensure that this process eventually ends, that maximum is decremented after every revolution. To differentiate between the ball and the beacon, when a maximum is found the platform stops and take another reading after 500 ms.

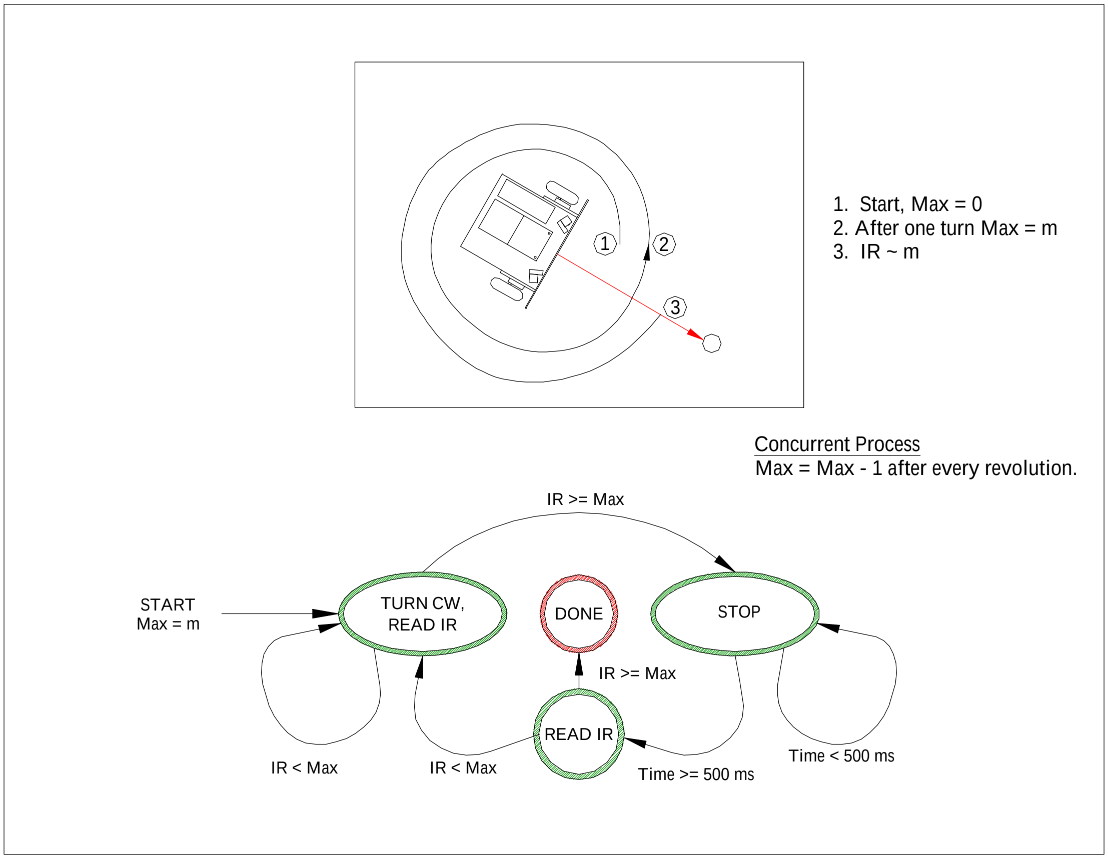


Figure 6: The Ball Finding Behavior

The beacon finding algorithm is summarized in Figure 7. This behavior also involves several concurrent processes.

- Rotate the platform until a greater IR reading is found opposite the direction of rotation. When this occurs, decrement and reverse the speed of rotation.
- Decrement the speed every 5 seconds, regardless of IR readings.
- If the ball detection beam is broken, scoot forward for 200 ms.
- Check exit condition (speed < 5.0).

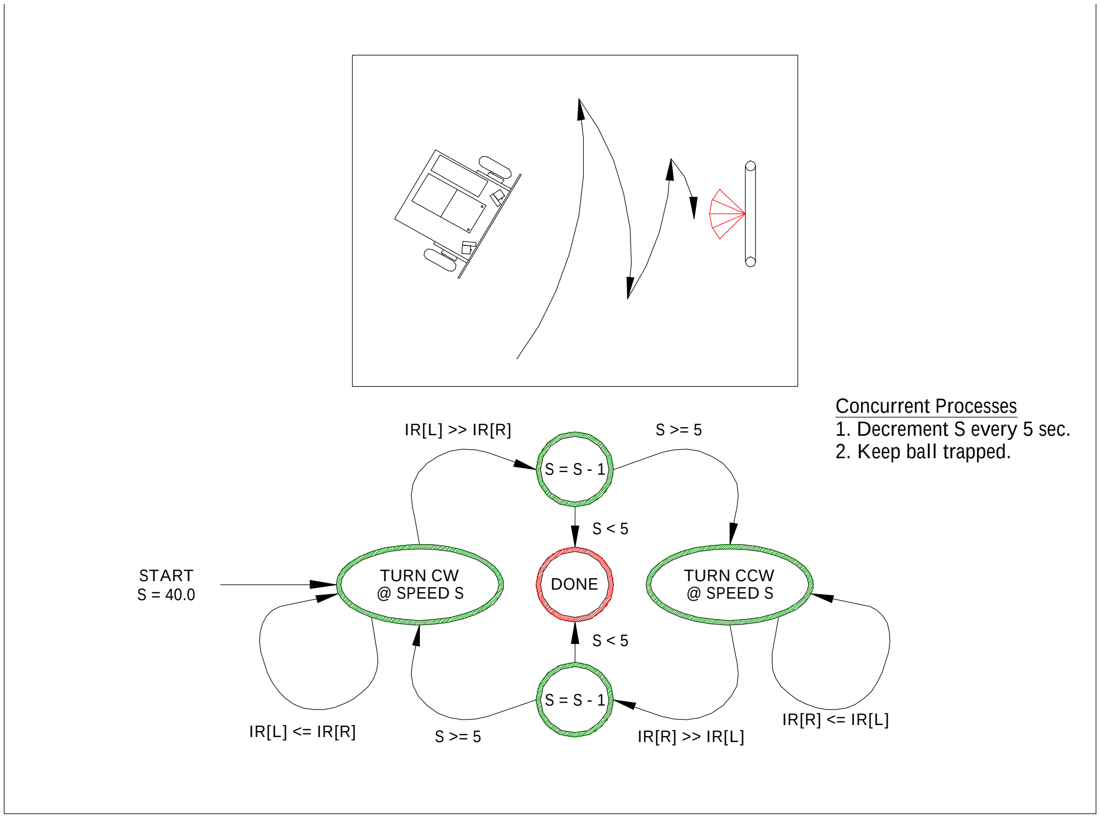


Figure 7: Ball Detection Behavior.

Experimental Methods and Results

A number of experiments were required during the design of Heather's wheel encoder. Although the resolution that "real" wheel encoders are capable of was not required for this application, I wanted to optimize the simple design discussed in the Sensors section of this report.

For the first experiment a short test program was written to turn on the motors at 50% power and take 100 readings from the analog port. This program was run under normal room lighting, and again in darkness. The results are shown in Figure 8.

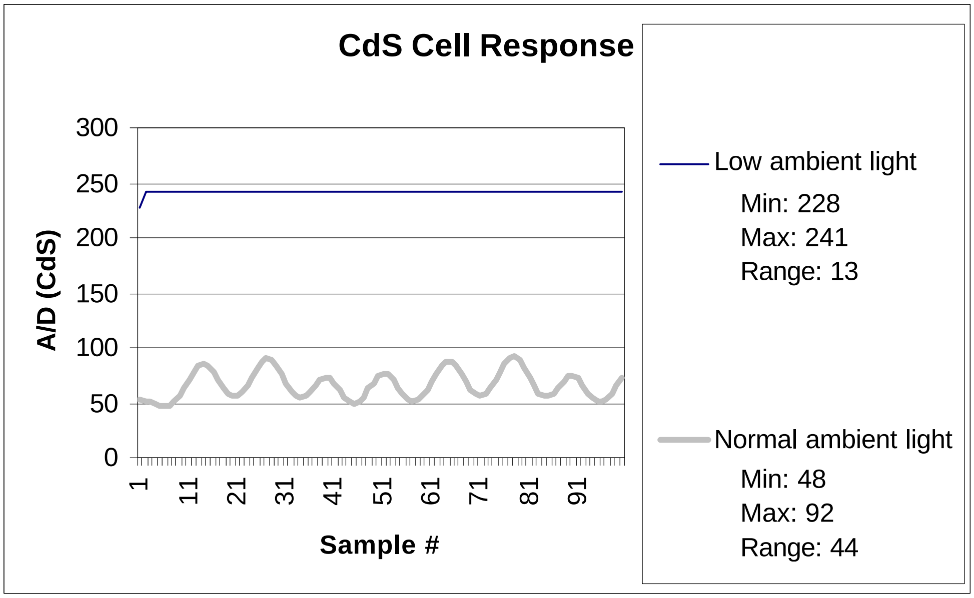


Figure 8: Experiment 1 Results

The result of experiment one show very poor response in low light conditions. The design of the device was changed at this point. An LED was integrated into the package to provide a constant source of light. This was driven by a digital output port. The test program was run again under normal and low light levels. The results are shown in Figure 9.

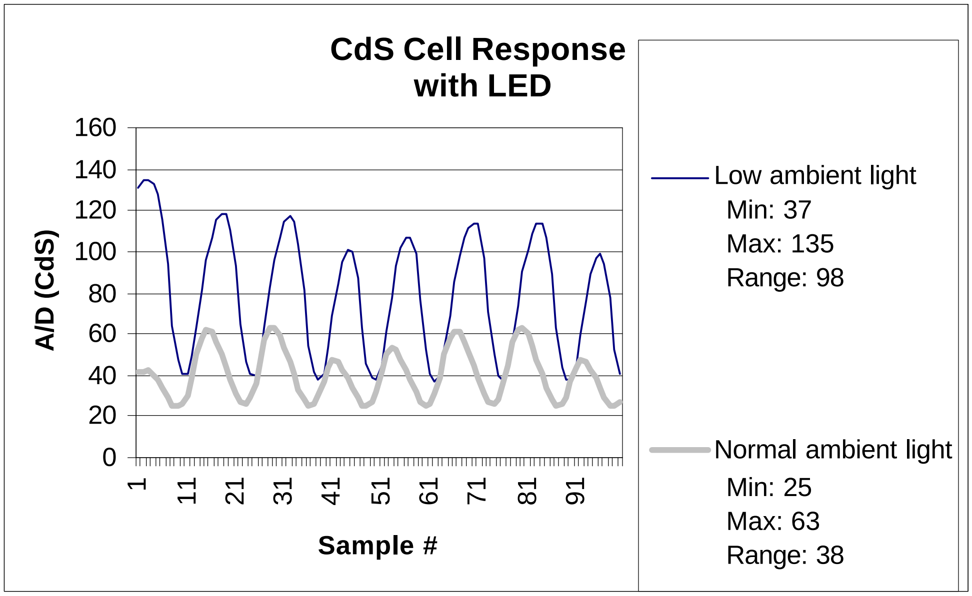


Figure 9: Experiment 2 Results

This experiment showed better sensor response in darkness than in normal room lighting. At worst, the device is capable of about a 5 bit range.

The next experiment was designed to optimize the width and number of stripes used to encode the wheel. The test program was run again with the three wheel configurations shown in Figure 10. The narrow stripes are 1/8" wide, and the wide stripes are ¼" wide.

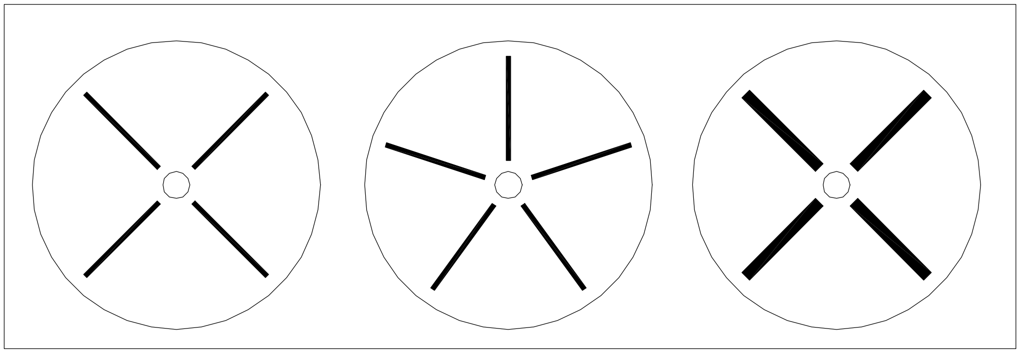
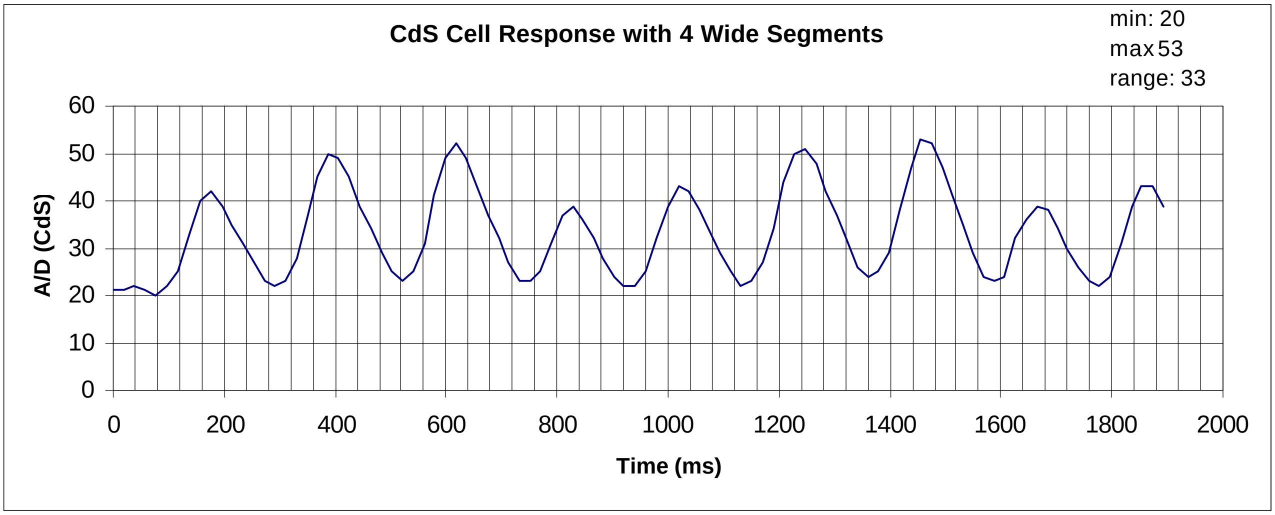
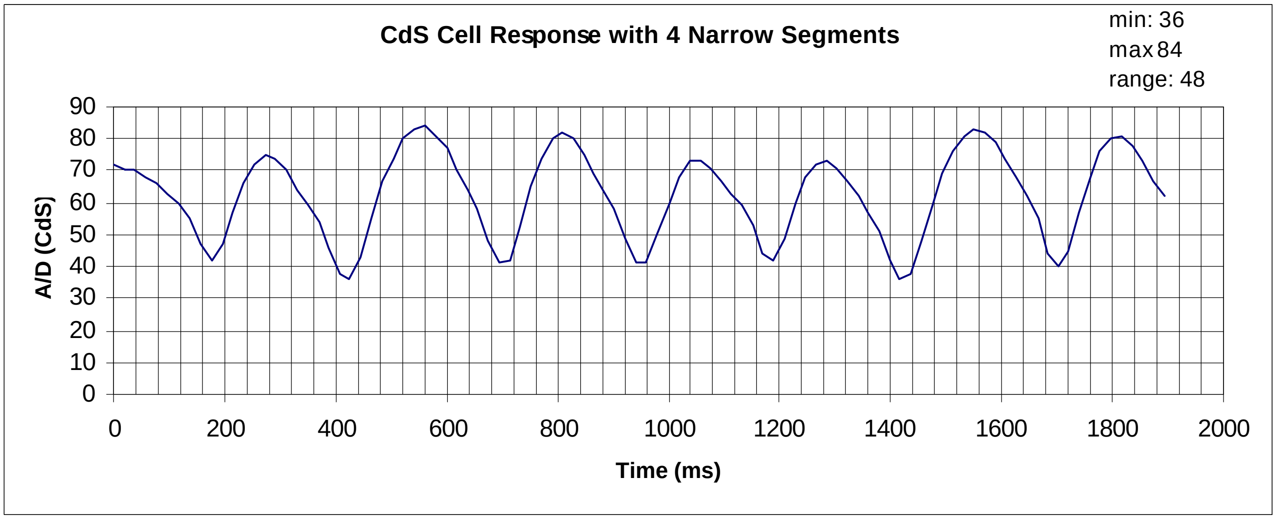
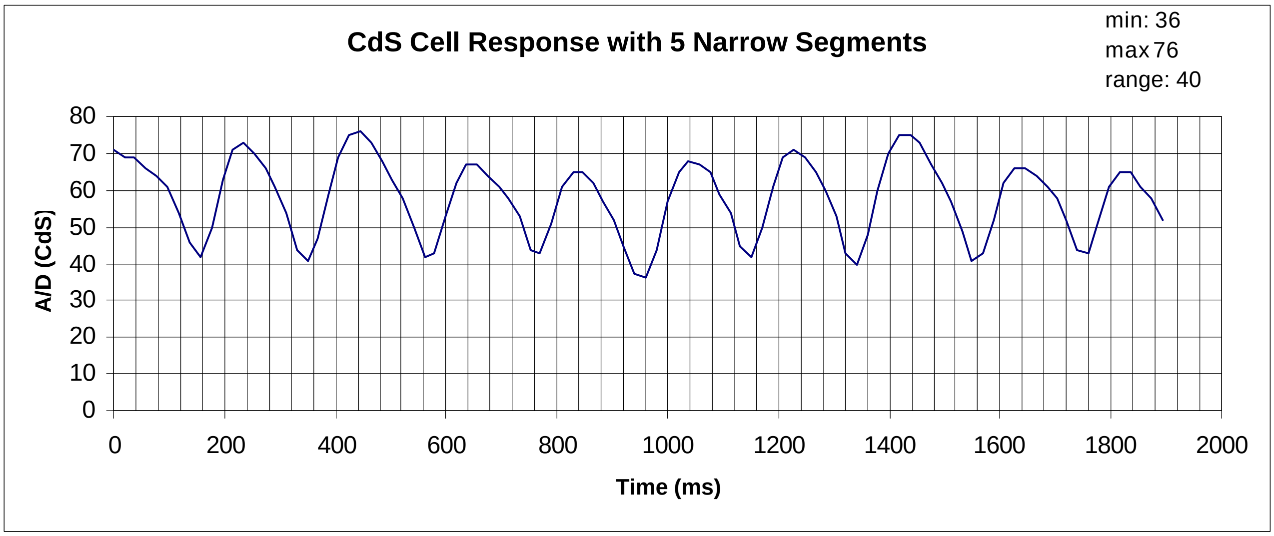


Figure 10: Stripe Configurations for Experiment 3

The results of the three trials are shown below.







Figures 11a, 11b, 11c: Experiment 3 Results

The greatest range was achieved with four narrow stripes(6b), and this was the configuration selected for the final design. Also, it was experimentally determined that 15 peaks correspond to one platform revolution.

Conclusion

I consider the Heather project to be a success overall, but there are several improvements to be implemented in future work.

- Ball detection is the least reliable of Heather's behaviors. This is partially due to the limitations of IR in general, but the range may be increased by using a cross-view pair of IR cannons.

- The use of two different frequencies of IR for ball detection and wicket detection would greatly simplify and speed up ball detection.

- The simple servo actuator has limited range, but this range corresponds well to the sensor range. As sensor range improves a stronger actuator will be feasible.

- The weight of the platform is near the limit that can be supported by cheap, bearingless servos.

Appendices

Use the following code under IC with lib_rw11.c.

demo.lis:

```
servo.icb
servo.c
cdtest2.c
keypad.c
wheel.c
```

cdtest2.c: line-following array calibration and line-following routine

```
int cdb1, cdb2, cdb3;
int cdw1, cdw2, cdw3;
int m1,m2,m3;


float fabs(float n)
{
 if(n>0.0) return n;
 else return -1.0*n;

}



void cd(int light)
{
int cd1,cd2,cd3;
int n=1;
cd1=analog(4);
cd2=analog(5);
cd3=analog(6);
while(n<50)
      {
      cd1=cd1*n+analog(4);
      cd2=cd2*n+analog(5);
      cd3=cd3*n+analog(6);


      n++;

      cd1 /= n;
```

```
      cd2 /= n;
      cd3 /= n;
      }
if(light){ cdw1=cd1; cdw2=cd2;cdw3=cd3;}
else {cdb1=cd1; cdb2=cd2; cdb3=cd3;}
start_process( buzz(50) );

}

void slopes()
{
m1=10*(cdb1-cdw1);
m2=10*(cdb2-cdw2);
m3=10*(cdb3-cdw3);
}

int ncd(int i){
int temp;
if(i==1)        {
      temp=A4;
      return ( ((temp-cdb1)*m2)/m1 + cdb2);

      }

if(i==3)        {
      temp=A6;
      return ( ((temp-cdb3)*m2)/m3 + cdb2);

      }
else return 0;


}

void line_follow(){
int a,b,c;
float s;
int t=1;
s=-20.0;
slopes();

while(1)
      {

      a = ncd(1);
      b = A5;
      c = ncd(3);

      if( (a >= (b+t)) && (a >= (c+t)) )
                  {
                  MOTOR1=0.0; MOTOR0=s;
                  defer();
                  }

      else if( (c >= (b+t)) && (c >= (a+t)) )
                  {
                  MOTOR1=s; MOTOR0=0.0;
```

```
                  defer();
                  }

      else  {MOTOR0=s; MOTOR1=s; defer();}




      }
}
```

<u>keypad.c: keypad scanning routine</u>

```
int keypad(void)
{

int inval=0;
/* scan col 1 */

poke(0x4000,1);
if((inval=( peek(0x5000) & 0b00001111) )!= 0)
      {
       if(inval==1) return 3;
       if(inval==2) return 6;
       if(inval==4) return 9;
       if(inval==8) return 12;
      }

poke(0x4000,2);
if((inval= (peek(0x5000) & 0b00001111) ) != 0)
      {
       if(inval==1) return 2;
       if(inval==2) return 5;
       if(inval==4) return 8;
       if(inval==8) return 11;
      }
poke(0x4000,4);
if((inval= (peek(0x5000) & 0b00001111) ) != 0)
      {
       if(inval==1) return 1;
       if(inval==2) return 4;
       if(inval==4) return 7;
       if(inval==8) return 10;
      }
return 0;




}

void beeper()
{
 poke(0x5000,128);
 poke(0x5000,0);
```

```
}
```

<u>wheel.c : all other behavior routines.</u>

```
int CLICKS;
int BUMPED;
int TRAPPED;
int BALL_FOUND;
float MOTOR0, MOTOR1;
int IN_PORT_4000, IN_PORT_5000;
int OUT_PORT_4000, OUT_PORT_5000, OUT_PORT_7000;
int KEY;
int BEAM;
int A1, A2, A3, A4, A5, A6, A7;
float T1, T2;


int IR_CANNON=1;
int LEFT_IR = 32;
int RIGHT_IR = 128;
int L_AND_R_IR = 160;
int BEAM_IR = 8;
int SMOOTH=0;


int RS_PID, ARB_PID, WE_PID, CB_PID;
int RK_PID, BEACON_PID, BALL_PID, STRIKE_PID;
int APPROACH_PID, TRAP_PID, PLAY_PID, LINE_PID;
int OBS_PID;



int abs(int n)
{
 if (n>=0) return n;
 else return -n;

}


int approx_eq(int a, int b, int tol)

{
 if( abs(a-b) <= tol) return 1;
 else return 0;


}


void one_turn(float speed)
{
CLICKS=0;
MOTOR0=speed; MOTOR1=-speed;
```

```
while(CLICKS<20) defer();
MOTOR0=0.0; MOTOR1=0.0;
CLICKS=0;

}

void play()
{


 while(1)
      {
      BALL_FOUND=0;
      TRAPPED=1;

      while(BALL_FOUND==0)
            {
            find_ball();
            approach_ball();
            }

      find_beacon2();
      strike_ball();
      }


 }




void start()
{

 MOTOR0=0.0; MOTOR1=0.0;
 OUT_PORT_4000=0; OUT_PORT_5000=0; OUT_PORT_7000 = BEAM_IR;
 BUMPED=0; TRAPPED=1; BALL_FOUND=0;
 BEACON_PID=0;
 BALL_PID=0;
 STRIKE_PID=0;
 APPROACH_PID=0;
 TRAP_PID=0;
 LINE_PID=0;
 OBS_PID=0;
 RS_PID = start_process( read_state() );
 ARB_PID = start_process( arbitrate()  );
 WE_PID = start_process( wheel_encode() );
 CB_PID = start_process( check_bumper() );
 RK_PID = start_process( read_keypad() );

}

void read_keypad()
{
      while(1)
      {
      KEY=keypad();
```

```
if(KEY==1)
      {if (BALL_PID==0)
            BALL_PID=start_process( find_ball() );
      }

if(KEY==2)
      {if (APPROACH_PID==0)
            APPROACH_PID=start_process( approach_ball() );
      }

if(KEY==3)
      {if (BEACON_PID==0)
            {
            BEACON_PID=start_process( find_beacon2() );
            }
      }

if(KEY==4)
      {if (STRIKE_PID==0)
            {

            STRIKE_PID=start_process( strike_ball() );
            }
      }

if(KEY==5)
      {if (PLAY_PID==0)
            PLAY_PID=start_process( play() );
      }
if(KEY==6)
      {if (OBS_PID==0)
            OBS_PID=start_process( obs_avoid() );
      }

if(KEY==7)
      {
      cd(0);
      }
if(KEY==8)
      {
      cd(1) ;
      }
if(KEY==9)
      {if (LINE_PID==0)
            LINE_PID=start_process( line_follow() );
      }
if(KEY==10)
      {
      if(SMOOTH) SMOOTH=0;
      else SMOOTH=1;

      buzz(100);

      }
if(KEY==11)
      {if(BALL_PID)
       {kill_process(BALL_PID); BALL_PID=0;}
```

```
             if (APPROACH_PID)
             {kill_process(APPROACH_PID); APPROACH_PID=0;  }
             if(BEACON_PID)
             {kill_process(BEACON_PID); BEACON_PID=0; }
             if(STRIKE_PID)
             {kill_process(STRIKE_PID); STRIKE_PID=0;}
             if(TRAP_PID)
             {kill_process(TRAP_PID); TRAP_PID=0;}
             if(PLAY_PID)
             {kill_process(PLAY_PID); PLAY_PID=0;}
             if(LINE_PID)
             {kill_process(LINE_PID); LINE_PID=0;}
            if(OBS_PID)
             {kill_process(OBS_PID); OBS_PID=0;}
             MOTOR0=0.0; MOTOR1=0.0;
             TRAPPED=1;
            }

      wait(500);
      }

}
void read_state()
{
 while(1)
      {
      A1=analog(1);
      A2=analog(2);
      A3=analog(3);
      A4=analog(4);
      A5=analog(5);
      A6=analog(6);
      A7=analog(7);
      IN_PORT_4000 = peek(0x4000);
      IN_PORT_5000 = peek(0x5000);
      BEAM= !(IN_PORT_5000 & 0b10000000);
      defer();
       }
}


void arbitrate()
{

 int lastm0, lastm1;
 float sm0, sm1;
 int f, f1;
 f=88;

 f1=100-f;
 lastm0=0; lastm1=0;
 while(1)
 {
 poke(0x4000,OUT_PORT_4000);
 poke(0x5000,OUT_PORT_5000);
 poke(0x7000,OUT_PORT_7000);
```

```
      if(BUMPED)
      {motor(0,-100.0); motor(1,-25.0);}

      else if(SMOOTH)
      {
          sm0=(float)((f*lastm0+f1*(int)MOTOR0)/100);
          sm1=(float)((f*lastm1+f1*(int)MOTOR1)/100);
          motor(0,sm0); motor(1,sm1);
      }

      else if (!TRAPPED)
      {motor(0,20.0); motor(1,20.0);}

      else
      {motor(0,MOTOR0); motor(1,MOTOR1);}

 lastm0=(int)sm0; lastm1=(int)sm1;
 defer();
 }

}


void wait(int ms)
{
 long time;
 time=mseconds() + (long)ms;
 while( time > mseconds() ) defer();

}

void wheel_encode()
{
 int lastcd, thiscd, dcd, lastdcd;
 CLICKS=0;
 OUT_PORT_5000 = OUT_PORT_5000 | 1;
 lastcd=A7;
 defer();
 thiscd=A7;
 dcd=thiscd-lastcd;
  while(1)
  {
  lastdcd=dcd;
  lastcd=thiscd;
  thiscd=A7;
  dcd=thiscd-lastcd;

  if( (lastdcd<=0) && (dcd>=0) && (MOTOR0 > 0.0) )

       CLICKS=CLICKS+1;


  wait(40);
  }
}

void check_bumper()
```

```
{
BUMPED=0;
while(1)
      {
      if (IN_PORT_4000 != 0)
            {BUMPED = 1;
            wait(2000);
            BUMPED=0;
            }
      defer();
      }
}

void trap()
{
TRAPPED=1;
while(1)
      {if(!BEAM)
            {TRAPPED=0;
            wait(200);
            TRAPPED=1;}
      defer();
      }
}



void buzz(int dur)
{
      OUT_PORT_5000 |= 0b10000000;
      wait(dur);
      OUT_PORT_5000 &= 0b01111111;

}



void find_beacon2()
{

      int avg;
      int min;
      int mean;
      int range;
      int max;
      int done;
      int best;
      float mot;
      float time;
      float timeout;
      time = seconds() + 30.0;
      timeout=seconds()+5.0;
      mot=40.0;
      done=0;
      max=0;
      min=255;
      best=0;
```

```
TRAP_PID= start_process( trap() );

MOTOR0=mot; MOTOR1=-mot;
OUT_PORT_7000 = BEAM_IR;
defer();

while(!done)
{
if(A1>max) max=A1;
if(A1<min) min=A1;

if(A2>max) max=A2;
if(A2<min) min=A2;

avg=(A1+A2)/2;
range=max-min;
mean=(min+max)/2;

if(mot < 10.0)          done=1;
if(KEY==11)             done=1;
if(seconds() >= time)   best=1;
if(seconds() >= (time+20.0))  done=1;
if(seconds() >= timeout)
      {
       mot-=2.0;
       timeout=seconds()+5.0;
      }

if( approx_eq(avg,max,5) && (best==1))
      done=1;

if( A2 >= (A1+2))
      {if(MOTOR0>0.0) mot=mot-2.5;
      MOTOR0=-mot; MOTOR1=mot;
      }
else if( (A2 > A1) && (A1 > mean) )
      {if(MOTOR0>0.0) mot=mot-.5;
      MOTOR0=-mot; MOTOR1=mot;
      }


if( A1 >= (A2+2))
      {if(MOTOR0<0.0) mot=mot-2.5;
      MOTOR0=mot; MOTOR1=-mot;
      }

else if( (A1 > A2) && (A2 > mean) )
      {if(MOTOR0<0.0) mot=mot-.5;
      MOTOR0=mot; MOTOR1=-mot;
      }

}

MOTOR0 = 0.0; MOTOR1 = 0.0;
kill_process(TRAP_PID);
TRAPPED=1;
```

```
      TRAP_PID=0;
      start_process( buzz(50) );
      BEACON_PID=0;


}


int pulse()
{
int reflect;
/*hog_processor();*/
poke(0x7000,161);
sleep(.03);
A1=analog(1);
A2=analog(2);
A3=analog(3);
poke(0x7000,OUT_PORT_7000);
reflect=A3;
return reflect;
}


void obs_avoid()
{
 OUT_PORT_7000 = L_AND_R_IR;
 MOTOR0=100.0; MOTOR1=100.0;
 defer();
 while(1)
      {


       if( A1>(A2+1) && A2>88)
            {
            MOTOR1=-100.0; MOTOR0=100.0; defer();
            }

       else if( A2>(A1+1) && A2>88)
            {
            MOTOR1=100.0; MOTOR0=-100.0; defer();
            }

       else if(A2<90 && A1<90)
            {
            MOTOR0=100.0; MOTOR1=100.0;defer();
            }

       else if(A2>105 && A1>105)
            {
            MOTOR0=-50.0; MOTOR1=-50.0; defer();
            }

       else
            {
            MOTOR0=50.0; MOTOR1=50.0; defer();
            }
      }
```

```
}

void find_ball()
{
 int thisir, maxir;
 int lastdcd;
 int newmax;
 int done;
 int minir;
 int medianir;
 int r_top_ir, l_top_ir;
 float time;

 done=0;
 newmax=0;
 maxir=0;
 minir=255;

  MOTOR0=30.0; MOTOR1=-30.0;
  defer();
  CLICKS=0;

  while(CLICKS<17)
      {
      thisir=pulse();
      wait(10);
      if (thisir>maxir) maxir=thisir;
      if (thisir<minir) minir=thisir;
      }
  start_process( buzz(200) );

  medianir=(maxir+minir)/2;
  MOTOR0=25.0; MOTOR1=-25.0;
  CLICKS=0;
  OUT_PORT_7000 |= L_AND_R_IR;
  defer();

  if( approx_eq(medianir, maxir, 1) ) done = 1;
  if( approx_eq(medianir, minir, 1) ) done = 1;

  while(!done)
      {
      MOTOR0=20.0; MOTOR1=-20.0;
      if(KEY==12) done=1;
      if((CLICKS/15) >= 1)
            {
            maxir -= 1;
            CLICKS=0;
            }

      thisir = pulse();
      defer();

      r_top_ir=A2; l_top_ir=A1;

      if ( thisir >= (maxir-2) )
            {
```

```
            MOTOR0=-12.0; MOTOR1=12.0;
            wait(500);

            start_process( buzz(50) );

            newmax=pulse();
            defer();

            if(A2 < r_top_ir) r_top_ir = A2;
            if(A1 < l_top_ir) l_top_ir = A1;

            if( (newmax >= ( thisir - 8) )) done=1;
            if( (newmax >= ( maxir - 8 ) )) done=1;
            if( (newmax >= ( medianir-2) )) done=1;

            if( (r_top_ir > 100) && (l_top_ir > 100) )
                  done = 0;
            if(!done)
                  {
                  MOTOR0=25.0; MOTOR1=-25.0;
                  wait(80);
                  }
            }
  }
  start_process( buzz(50) );
  MOTOR0 = 0.0; MOTOR1 = 0.0;
  BALL_PID = 0;

}

void strike_ball()
{
 float time;

 time=seconds()+3.0;
 OUT_PORT_7000 |= BEAM_IR;
 MOTOR0=-25.0; MOTOR1=-25.0;
 defer();
 while( BEAM && (seconds() <= time) ) {}


      if(BEAM) {MOTOR0=0.0; MOTOR1=0.0;}
      else
      {
      servo_deg(80.0);
      MOTOR0=100.0; MOTOR1=100.0;
      servo_on();
      wait(1500);
      servo_deg(25.0);
      MOTOR0=0.0; MOTOR1=0.0;
      wait(1500);
      servo_off();
      poke(0x1016,255);
      poke(0x1017,255);
      }
      start_process( buzz(50) );
      STRIKE_PID=0;
```

```
}


void approach_ball()
{

float time ;
time = seconds()+5.0;
BALL_FOUND=0;
OUT_PORT_7000 |= BEAM_IR;
wait(100);
MOTOR0=50.0;
MOTOR1=50.0;

while( (!BALL_FOUND) && (seconds() <= time) )
      {
      BALL_FOUND=peek(0x5000)&&128 ;
      }

OUT_PORT_7000 &= 0b11110111;
wait(150);
MOTOR0=0.0;
MOTOR1=0.0;

start_process( buzz(50) );
APPROACH_PID=0;
}

 int flash(int which)
 {
 int max;
 int a;
 float time;

 max=0;
 time=seconds();
 while(seconds() <= (time+2.0))
      {
      a=analog(which);
      if(a>max) max=a;

      }

       return max;
       }
```