JAWS

The Autonomous Ball Collecting Robot

BY

Kurnia Wonoatmojo

EEL 5666 Intelligent Machine Design Laboratory

Summer 1998

Prof. A. A Arroyo

Prof. M. Schwartz

Table of Contents

ABSTRACT

Originally, I designed this robot to collect golf or tennis balls. However, later I found out that two of my classmates are making the same robot. So, I changed my design a little bit by making it collects a particular color of balls and avoiding balls of other colors. In other words, I want to make this robot imitate a living creature (an animal). An animal would eat its favorite food and maybe avoids other things, JAWS perform the same behavior based on the color of the balls.

EXECUTIVE SUMMARY

For my robot's body, I decided to build it on a square platform. It uses three wheels: two wheels will be driven by motor and one is just an ordinary castor wheel. It will have a scoop made of plastic in front and a collection bin on top of the body. It will also have two arms to raise the scoop; one of the arms will be powered by a servo. To detect the ball, I will use a color sensor that I design myself with the help of my TA, Scott Jantz, out of regular CDS cells with a color filter in front. JAWS will also have IR sensors, bump sensors, and limit sensors for the arm. For the main processor, I am using one 68HC11 board with the ME11 expansion.

INTRODUCTION

JAWS will be a ball collecting robot which will detect the color of the balls. It has four distinct behaviors. One is obstacle avoidance, which is done by using two IR emitters and sensors placed on top of the scoop. The second behavior is bump detection. This behavior is achieved by placing four roller switches at the corner of the robot's body. Third is ball collecting, it will detect one particular color of ball and scoop it up to its collection bin. Forth is ball avoidance, it will avoid a particular color of ball or balls.

This robot will explore the possibility of color detection, which I believe has not been done by many students before. Everything else that I put or design for this robot is pretty much has been done before. JAWS' bumper design is also my original design; it detects collision from the front and side. The only drawback is that it does not detect collision in the middle of the robot's body or if the robot is hit in an angle

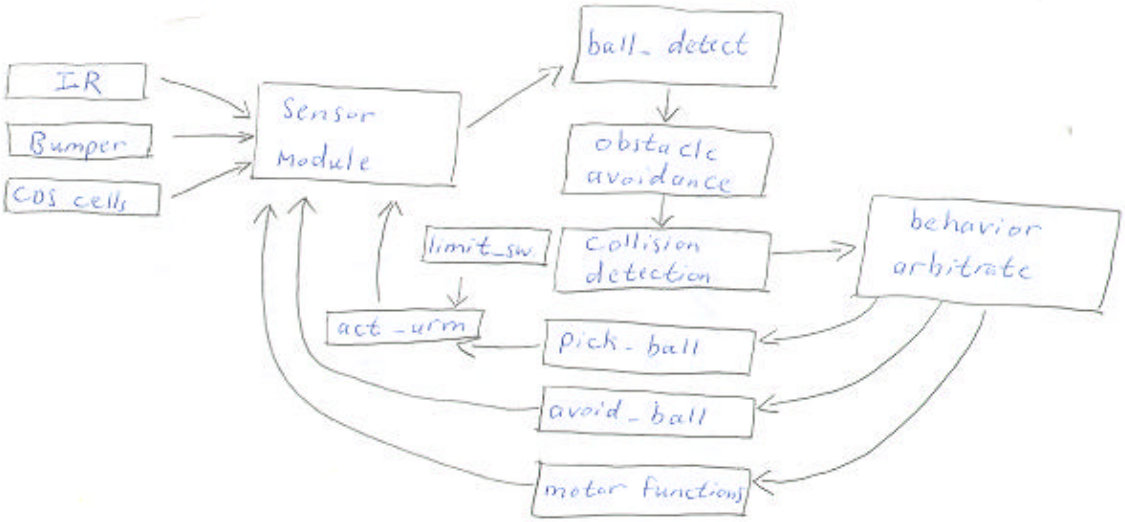
INTEGRATED SYSTEM

The system that I have on my robot are one 68HC11 processor, two IR sensors, four roller switches as bumper, one roller switch as limit switch, and four CDS color sensors. All the sensors, except the limit switch, in the software domain are controlled by one sensor module which will take the value of each of the sensors and put it in the appropriate parameter.

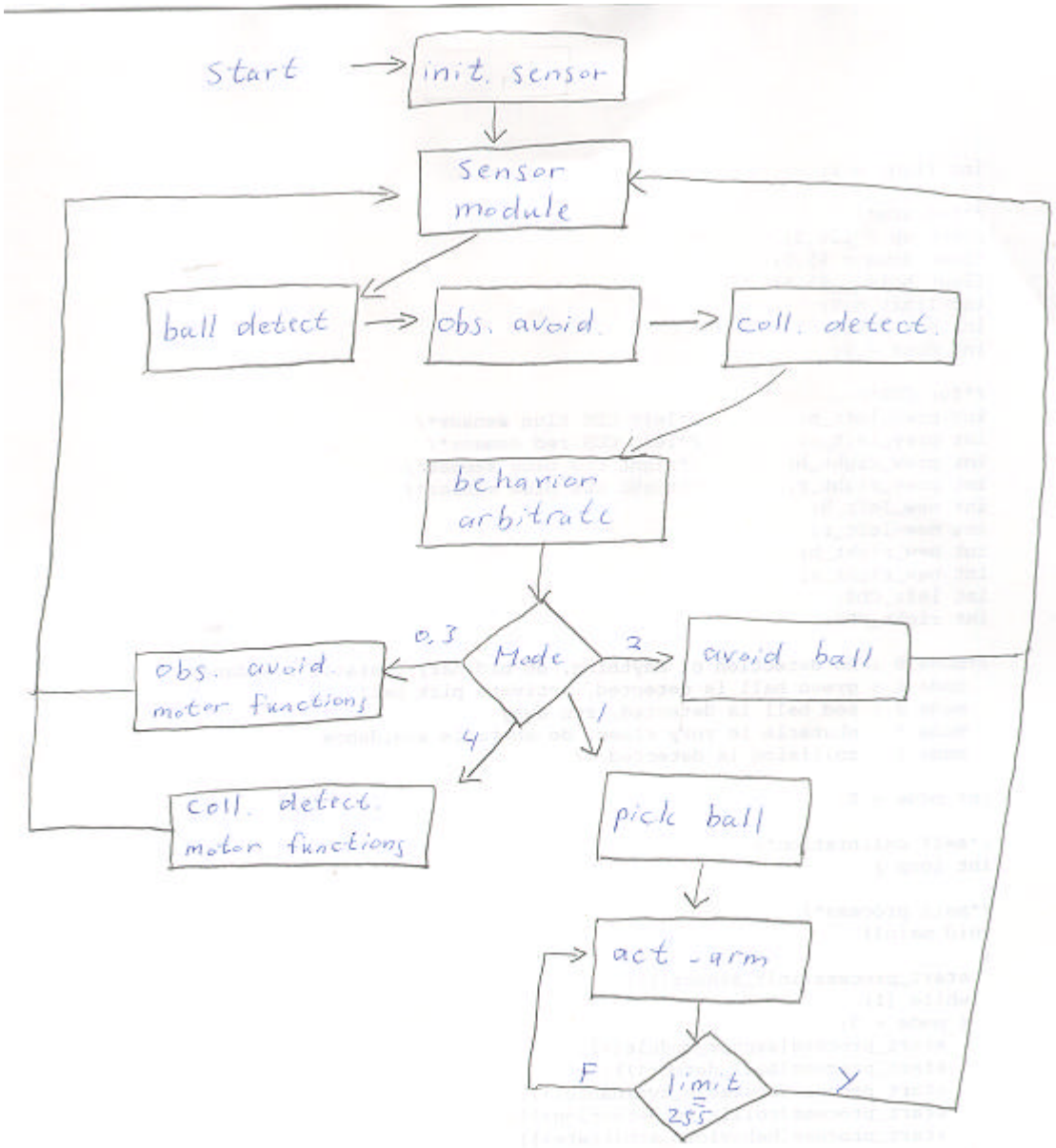
I design the IR sensors and the CDS color sensors to perform self-calibration the first time the robot runs. JAWS will spin around in circle and taking data as to be compared later.

After all sensors value has been read, an arbitrator is called to perform the appropriate task according to the reading of each parameter. If the arbitrator is executed really fast, it will give a sense of multitasking, even though it is really not a multitasking environment. All the arbitrator does is call the corresponding function and pass out the right parameter. This functions are the one that actually control the motors or servos; therefore maneuvering the robot. The flowchart diagram of the control is included on the next page.

The system block diagram:



FLOWCHART DIAGRAM



MOBILE PLATFORM

The platform for JAWS is a regular 8x11 inch birch plywood. I borrow the bottom design of TJ (Talrik Junior) for my motor and wheel compartment. I figure the design is simple and also separable from the main body. The ability to separate the wheel compartment with the body is useful for debugging. For the third wheel, I use a regular castor wheel. These three wheels form a triangle and are very stable. The first problem that I encounter with this design is when I changed movement from moving forward to backward, the castor wheel is trapped in a sideways position. Therefore it is inhibiting the robot's mobility. I found an easy way to debug this by using a rubber band and make the castor wheel stiff enough to not let it slip when moving backward but also allow it to turn.

My scoop is actually the bottom of a regular plastic garbage can. I tried to make the thickness of the scoop as thin as possible by using the garbage can to avoid pushing the ball around. However, the first time I tried to scoop up some balls, the scoop is still pushing the balls around. I found a way to trap the ball by adding a teeth or gripper in front of the scoop. With this new design, I can scoop up the balls successfully.

To raise the scoop, I design two arms in shape of the letter S. I use a servo to raise the scoop up or down. The arms have to be long enough to raise the scoop over the 68HC11 board that I place in front of the robot. I found that the 45-ounce servo is more than sufficient to raise my scoop.

ACTUATION

I use two fully hacked servos to move the wheels. Fully hacked meaning taking out the potentiometer and the logic circuit of the servo, leaving only the motor. These 45 ounce motors area cheap and strong enough to move my robot. To control these motors, I use the motor driver from the ME11 expansion board. ME11 has two motor drivers and two servo controllers. To drive the motors, all the ME11 does is toggle the +5 V pin. Unlike servos, which have three control lines, motors only need the power and ground wires.

The servo that I use for my arm is the same as the one I hack for my motor. I hacked the potentiometer so that it does not know the position it is at now. The reason for this hack is because I want to be able push the scoop as close as possible to the ground. I found that if the scoop is not level with ground, it has problem to scoop up the balls. To help me know if I have grounded the scoop, I add a limit switch at the arm. To know how far to raise it, I use a delay routine and perform several experiment to know how long the delay should be.

To control the motors and servo, I use a programming language called IC (Interactive C). The command to control the motor is: motor (0 or 1, speed). It has a speed range from –100.0 to 100.0 with the negative sign meaning going backward. For the servo, IC has a library routine called servo.icb and servo.c, which has to be downloaded from the library directory in IC. The control command are servo_on(), servo_deg(), and servo_off(). Servo_on() turns the servo on, servo_deg() moves the servo to the appropriate degree, and servo_off() turns off the servo.

The biggest problem in my project comes from the actuation. I did not find out until late in my design progress that the servo and motor command in IC uses the same pin (TOC 1). The motor command requires that TOC 1 is set to 0, while the servo command change the value of TOC 1. Running the servo and motor at the same time messes up both the servo and motor. I lost control of the servo's direction and the motor's speed and direction.

SENSORS

I have four types of sensors mounted on JAWS. They are IR sensors, color sensors, bump sensors, and limit switch. With the IR, I can detect obstacle in front of my robot. The color sensors are used to detect if I have a ball in front of my robot and the color of the ball. The bump sensors are used for collision detection. Finally, the limit switch is used for my scoop movement.

IR Sensors

Many people have done something with IR sensors, so I will be brief with my report on IR sensors. I conduct some experiment with my sensors and come up with the following data:

Distance (feet)	IR left (fed to port E 0)	IR right (fed to port E 1)
Very far	88	86
2	89	87
1 1/2	91	89
1	92	90
1/2	96	94
0.1	113	113

Table 1

The reading is mostly linear from 2 foot to 1 foot, but becomes non-linear between 1 foot to 1 inch. To implement some sort of Fuzzy Logic, I divide the area in front of my servo to three ranges: close, medium, and far. If the reading falls in the far region, my robot will go straight. If it falls within the medium region, it will turn left/right depending on where the obstacle is. If it falls inside the close region, my robot will back off.

Bump sensor

I use four roller switches connected as the figure 1 (on the next page). With this type of hack, I was able to save the analog port input since it only uses one pin. To detect

where the collision occurs, I connect each roller switch to a different resistor value. Whenever one is close, the reading from the Port E will be different because of the voltage divider circuit. My TA, Scott Jantz, shows this circuit to me.

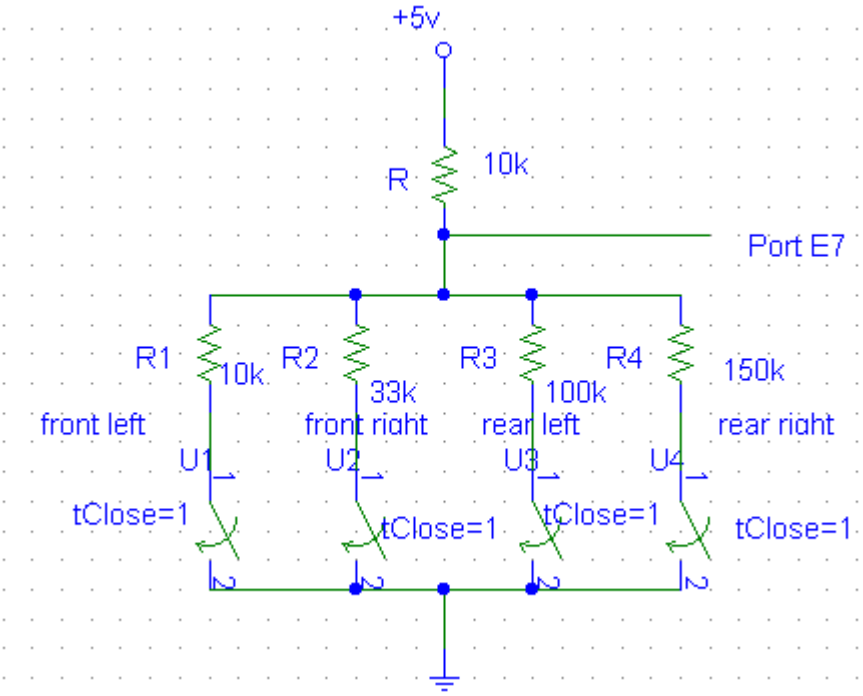
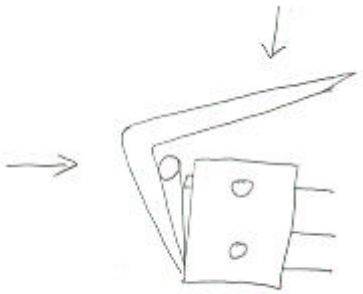


Fig. 1

I design the bumper to have the shape of the letter L. I glued the short end along the roller and stick the long end to the front/end of my robot. This bumper design will detect collision from the front/back and side. Below I include a picture of my bumper design.



The reading from port E is given on the next page:

Front left	Front right	Rear left	Rear right	Port E reading
√				128
	√			196
√	√			111
		√		233
			√	240
		√	√	219-220

√ = Activated

Table 2.

My algorithm for the bump detection is keep checking if one of the above readings is detected on Port E7. If one of the readings is detected, a mode is selected and a particular motor speed is pass on to the arbitrator. If it is not detected, then the robot moves on to the next function.

Color sensors

Each of my CDS sensors is set up as figure 2a and 2b below. In front of them, I put a color filter, which is basically a stack of color transparent sheets.

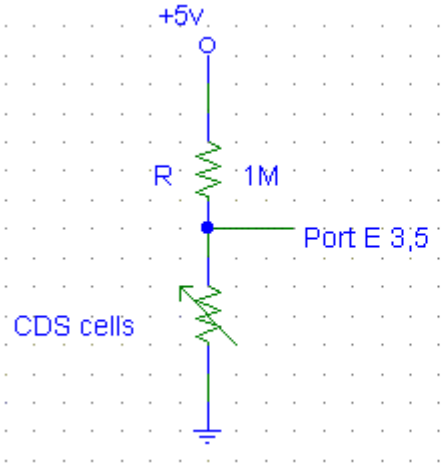


Figure 2a

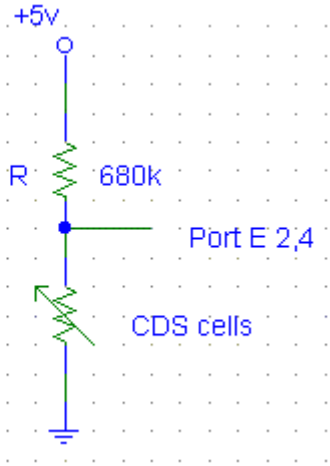


Figure 2b

I come up with the 1 MΩ and 680 KΩ resistors by experiment. I check the resistance range of the CDS cells. I found that the minimum range for the blue CDS cell is 700 KΩ and for the red CDS cell is 200 KΩ. The maximum range for the blue CDS

cell is > than 20 MΩ and for the red CDS cell is 1.8-2 MΩ. My experimental result from my CDS cells are included below. They are the readings from the analog port.

Original reading of each CDS cells.

red	blue	red	Blue
63-72	40-45	170-186	216-218
LEFT		RIGHT	

Table 3a

Color reading from the red CDS cells

Color of balls	Left (from Port E2)	Right (from Port E4)
White	± 2	± 4
Red	+ 7-10	+ 20-22
Yellow	+ 8-11	+ 22-27
Black	+ >20	+ >20
Blue	+ 5-12	+18-24
Green	+ 8-12	+ 24-31

Table 3b

Color reading from the blue CDS cells

Color of balls	Left(Port E3)	Right(Port E5)
White	- 8-15	- 15-25
Red	+ 1	+ 4-7
Yellow	+ 1-3	+ 6-9
Black	+ > 10	+ > 10
Blue	+1-5	+ 4-9
Green	+ 4-6	+ 2-5

Table 3c

+ = Reading from analog port increases

± = Reading from analog port can increase or decrease

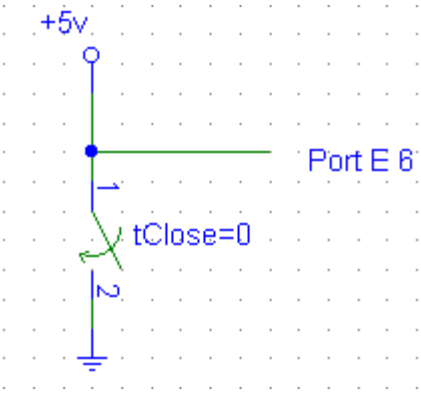
- = Reading from analog port decreases

From the table above, I could not get a good differentiation between colors. Their values fall to the same or similar range. I can only distinguish them to three separate groups: white, black, and others (red, yellow, blue, and green). I tried changing the resistor, but that just shift the numbers up or down. The other problem that I encounter is that the increase or decrease of each color is not the same everyday. This is caused by the sensitivity of the CDS cells. They are so sensitive that a slight change in lighting will change my whole data. Prof. Schwartz suggested putting a light bulb that will shine on the balls to get a more consistent reading. I incorporated that suggestion to my design and am able to get a more consistent increase or decrease each time. The data that fills up the three tables is the one that I get about 75% of the time.

After many algorithm tests, I found that my color sensor can detect white color the best. Originally, I tried to get the reading from both color sensor and subtract them. This is the value that I hope will distinguish the colors. However, this value is still to close to one another. The second is trying to see if the reading falls to one particular range. This algorithm does not work because each reading from a different day is different, so the range for any color is different each day. The third algorithm that I found to be the most successful is checking if I get a drop from my blue CDS sensors and a steady reading from my red CDS sensors (I defined $\pm$ 3 as steady). This algorithm will detect white ball most of the time. My success rate is probably 75% and is the highest of all my other algorithms. To help my ball detection, I incorporated the two IR sensors to my algorithm. The IR sensors cannot detect the color but they can detect when I have a ball in front of the robot. They will give me an increase reading of 2-3.

Limit Switch

For my limit switch, I am using another roller switch that I glued to the side of my arm. I measure the distance so that the switch closes when the scoop is level with ground. This switch is connected to my analog port (pin E6). It makes a straight connection from power to ground; so that when it closes, the reading will be 255. My connection diagram is drawn on the next page.



This is my only sensor that is not read by my sensor_module. It is read by my act_arm function when it tries to return the arm to be level to ground. The algorithm simple since it just keep lowering the arm until a value of 255 is detected from pin E6.

BEHAVIORS

As stated before, JAWS has four distinct behaviors. They are obstacle avoidance, collision detection, ball collecting, and ball avoidance. There are five major functions that keep running all the time. First is the sensor_module. This function has the task to read all the analog port and put the data in the appropriate variable. Then I have the obstacle_avoidance, collision_detection, and ball_detect functions that will run right after the sensor_module. These functions will pass on parameters to the arbitrator and change the mode of operation. Finally, I have my arbitrator, which will carry out all the necessary command to react to the mode and parameters from my function module.

To make sure that I am not running two or more behaviors at the same time, I design my arbitrator to use priority in deciding which behavior to perform. The behavior that has the highest priority is collision_detection. When a collision is detected, my robot will execute the collision_detection algorithm no matter what the other sensor might read. Second is my obstacle_avoidance. Third is my ball detection.

My biggest problem in implementing the behavior is timing. Since IC does not have multitasking and does not have interrupts, each sensor is read consecutively. This means that my robot will not be able to react to its surroundings right away. There is a

delay to read all the sensors and for the arbitrator to make a decision. Although the delay is not that bad, but it still make the robot less responsive to its environment.

EXPERIMENTAL RESULTS

Overall, my robot performs just as expected. I am still having problem to run both servo and motor at the same time. My obstacle_avoidance and collision_detection works perfectly. My ball detection and avoidance is not that great. I am still missing some balls or detecting a ball when there is nothing. Later I found out that putting a diode allows me have the motor and diode running at the same time. My TA, Scott Jantz, pointed it out to me. However, a new problem arises, I can only run them both together one time. The second time I run them together, sometimes my motor turns on or changes speed when it is not supposed to. The same goes to the servo, sometimes it raises/lowers the scoop faster sometimes slower. Sometimes, my servo raises the scoop but never lowers it down.

My body and scoop perform quite well. The body is strong enough to hold everything together without losing balance or dragging anything. My scoop is strong enough to lift a golf ball and scoop it to my collection bin. My bumper design is also detecting collision from the front and side as expected. The only problem is that it is not strong enough and has fallen off many times. This is because I only use hot glue to glue it to the roller switches.

CONCLUSION

I have built an autonomous mobile robot that I proposed to build earlier in the semester. I realized that it is not perfect and still need a lot of improvement, especially the color sensor. I can not totally eliminate the motor-servo problem. I can only run them once together. After that I have to reset the board and do everything over again to avoid the motor-servo unwanted interaction.

For future works, I would try a different sensor for my color sensor besides CDS cells. CDS cells are great to detect light or shadow, but not colors. They are too sensitive to light and very unstable.

One personal note is try to stay away from super glue. I use super glue to hold my robot's body together. Super glue is very strong and it holds my robots body well, but I am having huge problem when I have to take things apart. I have to fix my servo gear one time, but I super glued it to my robot already. It is hard and painful to separate it from my robot. I would recommend using screws or nuts to hold things together.

DOCUMENTATION

Servo.icb
Servo.c
Librw_11.c

CODE

```
/*Variables*/

/*for collision detection*/
int coll = 0;

/*for obstacle avoidance*/
int l_middle = 95;
int l_far ;
int r_middle = 94;
int r_far ;
int up_close = 98;
int time =0;

/*for color sensor*/
```

```
int left_CDS;
int right_CDS;

/*for IR*/
int left_eye;
int right_eye;
int left_far;
int right_far;
int my;

/*for arm*/
  float up = 120.0;
  float down = 85.0;
  float hold = 91.5;
  int limit = 0;
  int pos = 0;
  int done = 0;

int prev_left;
int prev_right;
int new_left;
int new_right;
int left =0;
int right=1;
int mode;

int loop;
int pid;

void main()
{
  init_sensor();
  start_process(sensor_module());
  start_process(behavior_arbitrate());
}

void init_sensor()
{
 init_IR();
 init_CDS();
}

void init_IR()
{
  poke(0x7000,0xff);
  wait(50);
  l_far = analog(0);
  r_far = analog(1);
}


void init_CDS()
{
  loop = 200;
  prev_left = analog(3);
  prev_right = analog(5);
  while (loop > 0)
```

```
   {
    new_left = analog(3);
    new_right = analog(5);
    prev_left = (prev_left + new_left) / 2;
    prev_right = (prev_right + new_right) / 2;
    loop = loop-1;
   }
  motor(left,-100.0);
  motor(right,100.0);
  wait(100);
  motor(left,0.0);
  motor(right,0.0);
}

void sensor_module()
{
   while(1)
    {
       left_eye = analog(0);
       right_eye = analog(1);
       left_CDS = analog(3);
       right_CDS = analog(5);
       coll = analog(7);
       wait(100);
    }
}

void behavior_arbitrate()
{
  while(1)
  {
    l_middle = 95;
    r_middle = 94;
    up_close = 100;
    left =0;
    right=1;
    up = 120.0;
    down = 85.0;
    hold = 91.5;
    mode = 0;
    if (time ==0)
     {
      ball_detect();
      time = 1000;
     }
    obstacle_avoidance();
    collision_detection();
    if (mode == 4)
    {
     motor(left, bump_left_speed);
     motor(right, bump_right_speed);
     wait(1000);
    }
    if ((mode == 0) || (mode == 3))
    { /* no other sensor is detecting anything run the lowest priority
behaviour
       or obstacle is very close avoid*/
```

```
     motor(left, IR_left_speed);
     motor(right, IR_right_speed);
    }
   time = time -1;
 }
}


void ball_detect()
{
 motor(0,0.0);
 motor(1,0.0);
 sleep(2.0);
 init_CDS();
 my = 5000;
 while (my > 0)
  {
   if  ( ((prev_left - left_CDS) > 4) || ((prev_right - right_CDS) > 9)
)

       pick_ball();



   if  ( ((left_CDS - prev_left) > 2) || ((right_CDS - prev_right) > 2)
)
       run_away();
   my = my -1;
  }
}

float IR_left_speed, IR_right_speed;
int turn = 0;
void obstacle_avoidance()
{

        if ( (left_eye < (l_far+4)) && (right_eye < (r_far+4)) )
         { IR_left_speed = 100.0; IR_right_speed = 100.0; mode=0;}
/*no obstacle go straight*/
        else
         {
           if( (left_eye > up_close)  &&  (right_eye > up_close))
          {                                                    /* obstacle
back off*/
             mode = 3;
             if (turn == 0)
            {IR_left_speed = -100.0; IR_right_speed = -50.0; turn = 1;}
             else
              {IR_left_speed = -50.0; IR_right_speed = -100.0; turn
=0;}
           }
           if ((left_eye > l_middle) && (right_eye < r_middle) )
               {IR_left_speed = 80.0; IR_right_speed = -40.0;}
           if ((left_eye < l_middle) && (right_eye > r_middle) )
               {IR_left_speed = -40.0; IR_right_speed = 80.0;}
       }
}
```

```
float bump_left_speed = 0.0;
float bump_right_speed = 0.0;

void collision_detection()
{
      if (coll < 254)                           /*no collision is
detected*/
       {
        if ((coll < 130) && (coll > 126))            /*collision in
left/front side*/
         { bump_left_speed = -20.0; bump_right_speed = -100.0; mode =
4;}
        if ((coll < 198) && (coll > 194))            /*collision in
right/front side*/
         { bump_left_speed = -100.0; bump_right_speed = -20.0; mode =
4;}
        if ((coll < 113) && (coll > 109))            /*collision
straight in front*/
       { bump_left_speed = -100.0; bump_right_speed = -100.0; mode =
4;}
        if ((coll < 236) && (coll > 230))       /*collision in
left/rear side*/
         { bump_left_speed = 100.0; bump_right_speed = 60.0; mode = 4;}
        if ((coll < 242) && (coll > 238))            /*collision in
right/back side*/
         { bump_left_speed = 60.0; bump_right_speed = 100.0; mode = 4;}
        if ((coll < 221) && (coll > 218))            /*collision in
the rear*/
       { bump_left_speed = 100.0; bump_right_speed = 100.0; mode = 4;}
       }
      else mode =0;
}


void pick_ball()
{
  motor(left, 100.0);
  motor(right, 100.0);
  wait(1000);
  act_arm();
  wait(2000);
  init_CDS();
  wait(100);
}

void act_arm()
{
  servo_on();
  up = 120.0;
  servo_deg(up);
  wait(600);
  limit=0;
  motor(0,0.0);
  motor(1,0.0);
  servo_deg(hold);
```

```
  my = 4000;
  while (my > 0)
   { my = my-1;}
  down = 85.0;
  servo_deg(down);
  sleep(0.3);
  while (limit < 253)
    {limit = analog(6);}
  servo_off();
  poke(0x1016,0x00);
  poke(0x1017,0x00);
}

void run_away()
{
  motor(left, 100.0);
  motor(right, -100.0);
  my = 4000;
  while (my > 0)
   {
    my = my -1;
   }
  motor(left, 0.0);
  motor(right, 0.0);
}
```