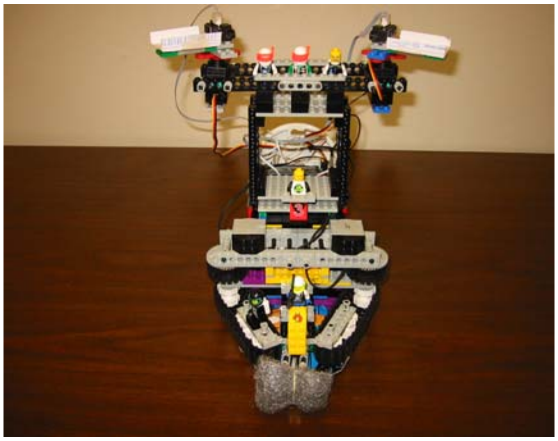
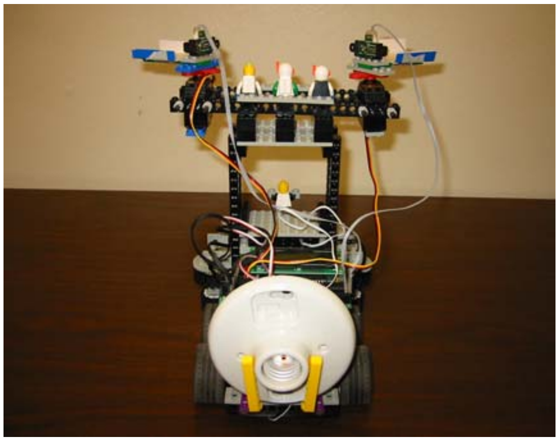
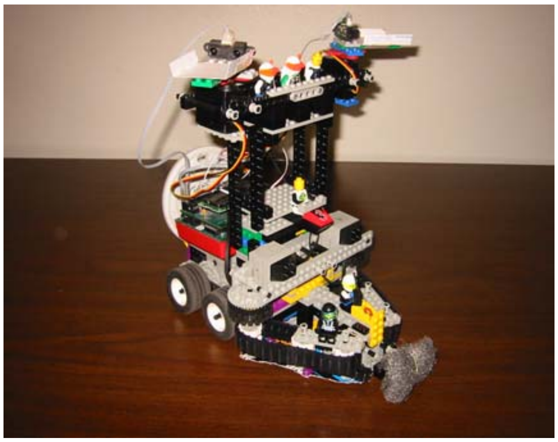
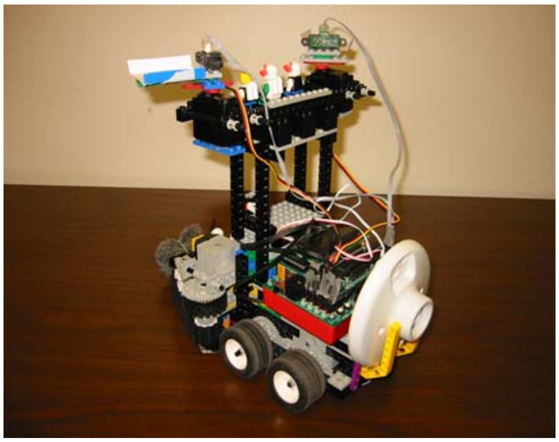



GROUP 7
PROJECT 2

Robert Moe
Charlie Sun
Mark Woehrer
John Zumwalt

Section 1 - Hardware Design

1.1 Introduction

The basic idea behind the robot's design was to allow for a brute-force approach. Rather than sensing small rocks and other nuisance obstacles (walls, light cords, etc.), the robot hardware is designed to get us around any of these hazards. However, this is not to say that it is blind to everything. On the contrary, we tried to increase the accuracy when detecting the buckets by concentrating most of sensing at a vertical level that would only be occupied by buckets. Only minimal detection is done at the ground level to escape from head-on collisions.

1.2 Sensing Hardware Components

- ET Range Sensors
- Light Sensors
- Switch (Bump) Sensor

1.3 Sensing Hardware

Range Sensors – The ET range sensors are mounted on a platform high above the robot. This restricts the set of objects that the sensors will see. Since the highest obstacles in the room are the buckets, mounting the sensors at this level allows isolation of sensor readings from the ET's to avoiding buckets.

The guards underneath the sensor are light-shields. They were added to at least lessen the amount of interference that the range sensors were subjected to at close ranges to the target light-bulbs. It was found during testing that the ET's would get strange readings when close to the lights and these seemed to help.

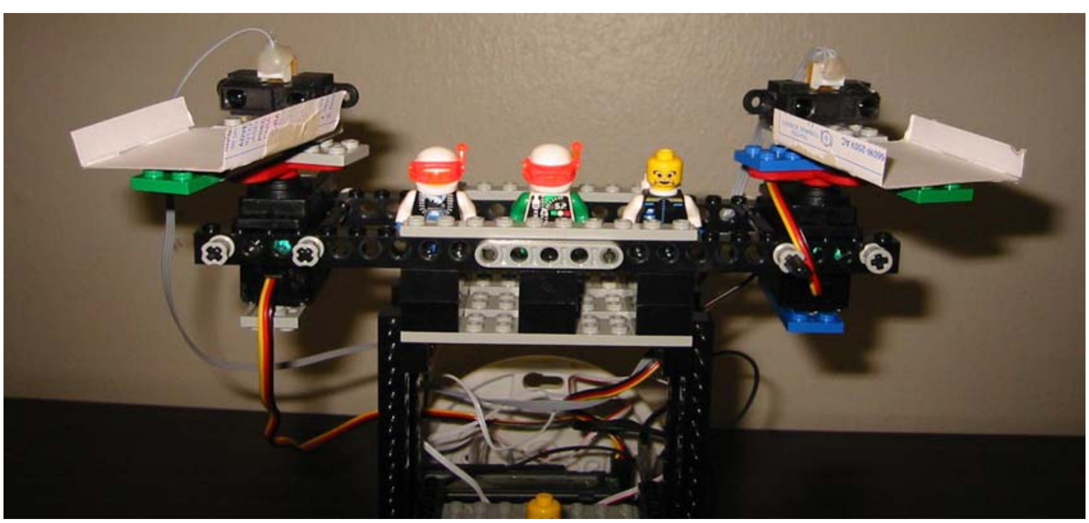


Figure 1.1 – Range Sensors (top platform), cardboard shields to block out ambient light, mounted on servos to provide sweeping ability

Light Sensors – Two light sensors are housed near the center of the robot. They are separated from each other and shielded on the top and bottom. This positioning is necessary to provide a difference in readings.

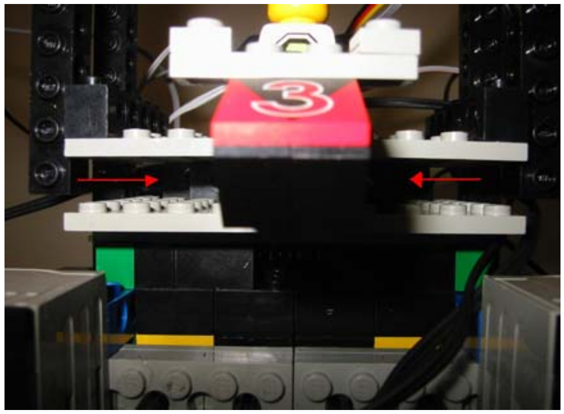


Figure 1.2 – Light Sensors

Bump Sensor – The bump sensor mechanism is housed inside the duster assembly. It is comprised of an extension arm which activates the bump sensor. At the end of the arm is a flat LEGO piece that provides a mounting point for steel wool.

1.4 Hardware Systems

The "Duster" – This is perhaps the main hardware feature of the robot. It consists of tank tracks mounted in the shape of a wedge designed to shed rocks to the sides of the robot (Figure 1.3). This assembly is mounted on the front of the robot and is powered by two separate LEGO motors. The tracks are driven at the same ratio as the drive-motors (5:1) to ensure that the tracks work in concert with the actual drive-motors. In addition, wheels are attached to the drive-shafts of the tracks (Figure 1.4). These were added in an effort to provide some physical guidance when stuck or hugged up next to the edge of the arena.

Figure 1.3 – The "Duster" (front), arrows point out the tracks.

Figure 1.4 – The "Duster" (right and left), motors are mounted vertically connecting to the large 40 tooth gears. Below the gears are the guide wheels.

Weight Holder – The weight holder was a late addition to the robot. It was found to be necessary to lift the bulky front end off of the floor. Initial runs were plagued by the problem of clearance and the weight helped a little. In addition, the extra weight allowed the robot to push rocks because of the increased traction.

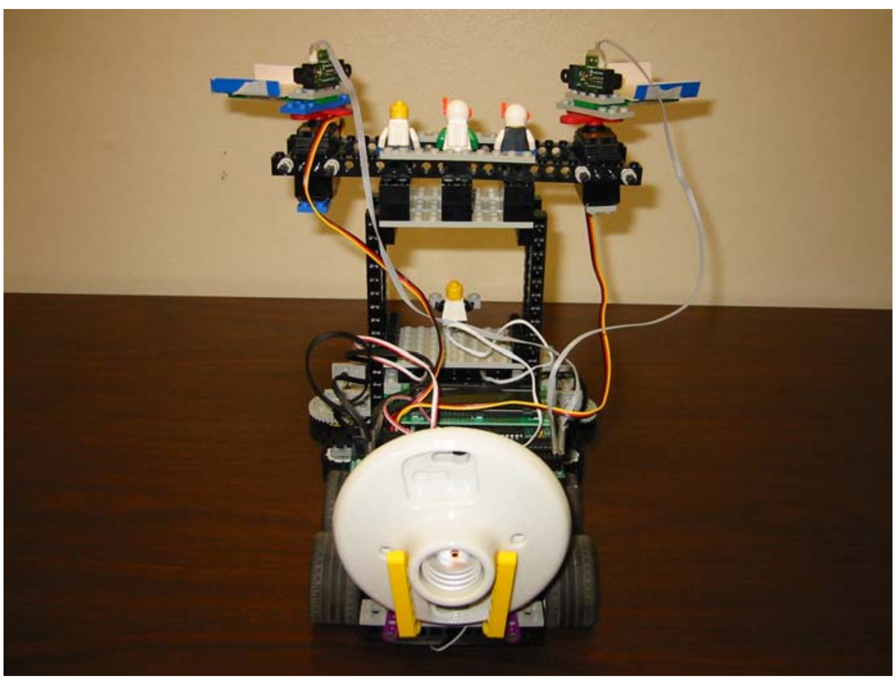


Figure 1.5 – Weight and Weight Holder (back), stolen light housing

Drive Gears – The robot is propelled by a 4-wheel-drive system. Each side of the robot is independently powered by a LEGO motor. The aim was to provide a high-torque, low-speed drive system. This is accomplished by using a 5:1 gear ratio with an 8-tooth gear on the motor directly powering a 40 tooth gear.

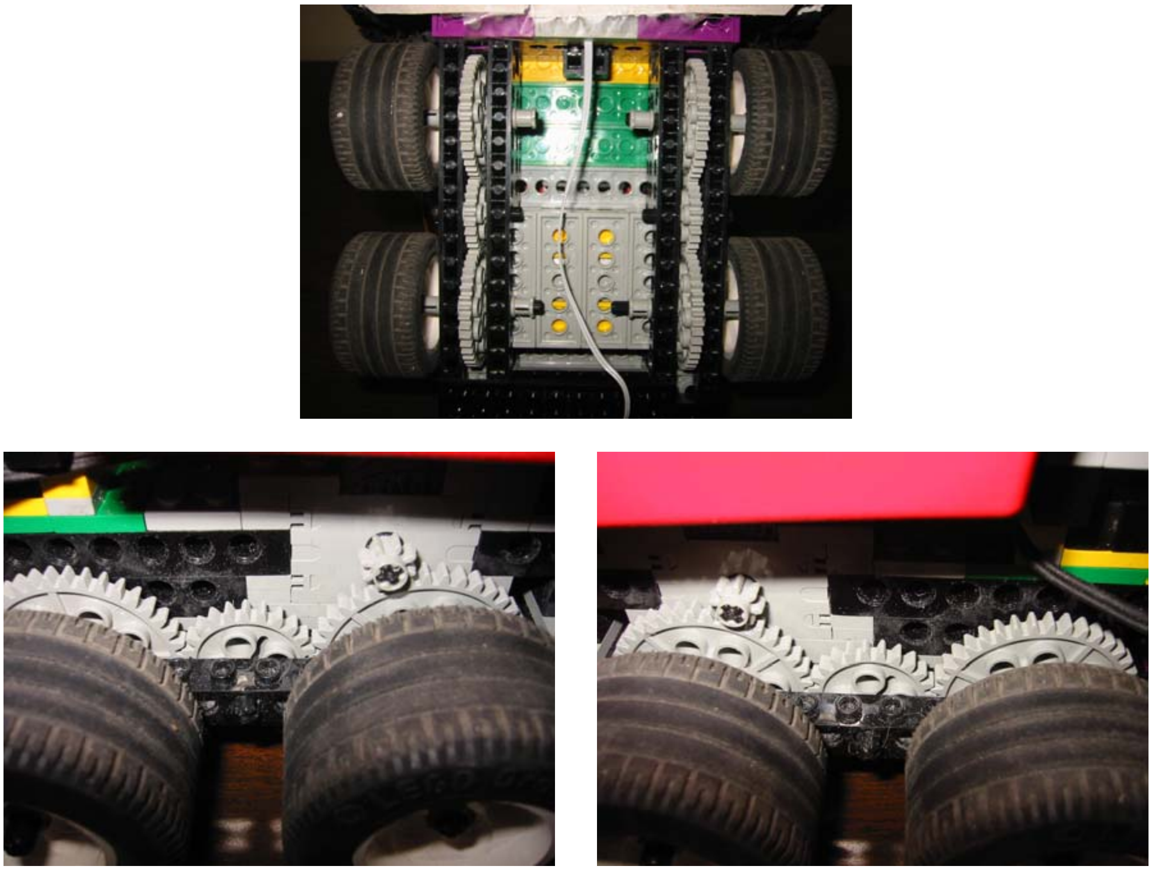
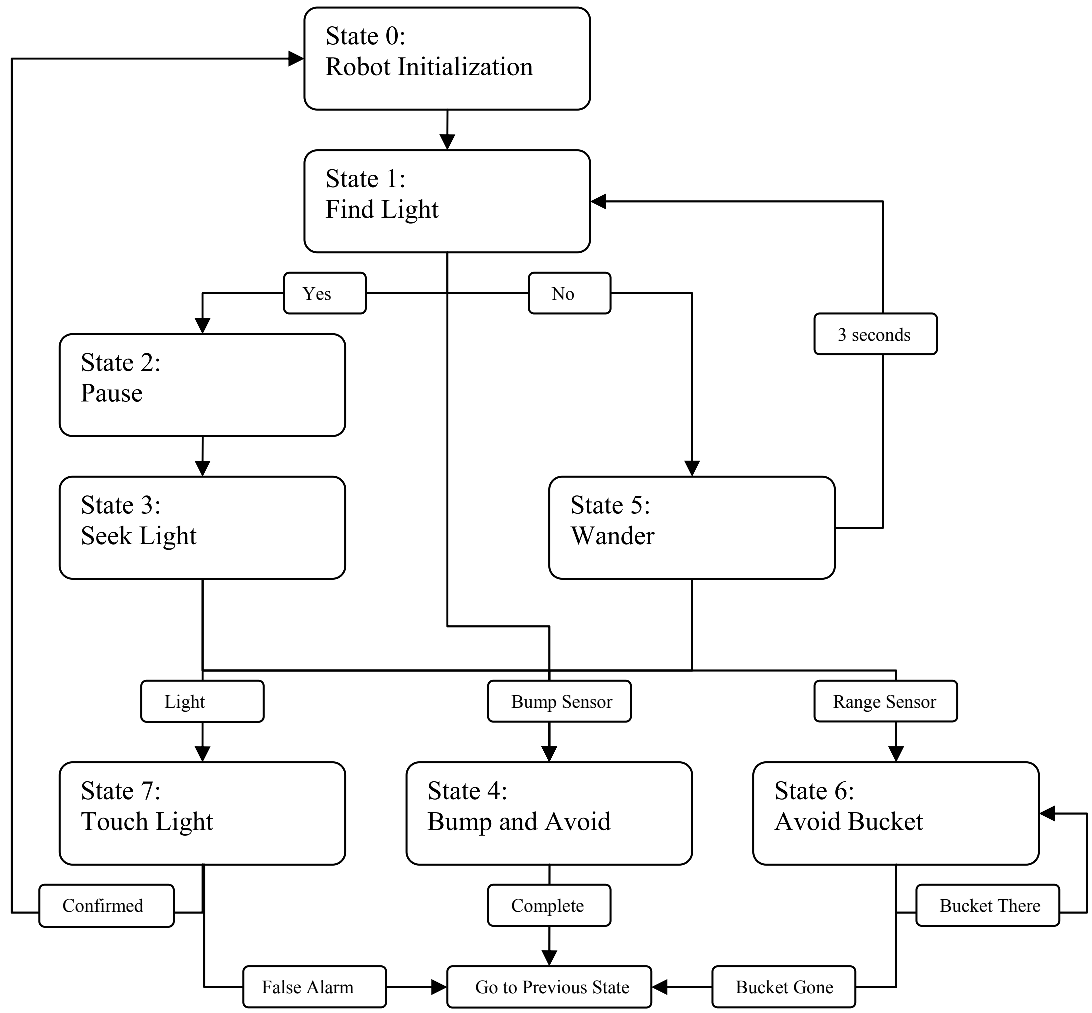


Figure 1.6 – Robot drive train (bottom, right, & left)

Section 2 – Software Design

2.1 Introduction

Our team decided to use parallel state machines instead of parallel processes in Interactive C. The basic idea is that you can have multiple state machines operating in parallel, each advancing one state every 10ms. In our implementation a main state machine acted as the sequencer/controller logic for the other state machines, i.e. it picked which lower-level state machines to advance in a particular main state, and was responsible for making sure the state machines were reset to their initial state if needed.



2.2 Implementation

Each state machine is responsible for keeping track of its own state using a global variable which can be set by the main state machine. Each state machine can suppress or inhibit the other state machines using global variables. In this way each state machine is implemented as a regular function call with the a return value of 0 if the state machine is still running, 1 if complete, 2 if and error has occurred. It is important to note that a state machine does not always need to have a return value, i.e. the return type is void. An example is the move_servos() state machine which simply moves the range finder sensors back-and-forth and does not return a value.

Implementing a msleep() function with state machines.
Since the main loop was updated 100 times/second is is not possible to use the msleep() and sleep() functions without disrupting the other state machines. For example if you wanted to backup for 250ms you could not just insert msleep(250L) in your code. Instead you need to define a new state in your state machine, and define a global countdown variable (prefixed with the state machine name for uniqueness). Then you would set the countdown variable in the state you are in (in this case 25 ticks) then move

to the sleep state, wait in the sleep state until the countdown variable reaches zero, and then move to the next state. Note that all transitions between states take place on 10ms boundaries.

Since each state machine can be inhibited in the main state machine the motor() function calls can be used in each state machine as if it is the only state machine running. We decided early on that we would use with the four-wheel drive system, an so we did not use an abstract motor schema. This was a benefit for us be cause it allowed us to use more a more optimal control behavior.

2.3 Additional State Machines.

State Machines:

- move_servos()
- seek()
- find_light()
- backup_and_turn()
- avoid_bucket()
- touch_light()

2.3.1 Seek function.

The seek() state machine is responsible for the photo-taxis behavior of the robot. It uses a simple proportional feedback controller with a gain sequence. The basic idea is that you look at the difference in the intensities from the left and right light sensors, and force that difference to zero (using feedback). The gain determines how quickly the robot responds to a difference. If the gain is too high for the given inertial of the robot (which in our case was quite large) the robot will oscillate (which is not desired) when the robot is not close to the light source. Our add-hoc solution to this problem was to use a gain sequence which started the gain at a low value and monotonically increased as the intensity increased. The exact values were determined experimentally.
The seek() state machine includes a timeout variable which resets the gain to a minimum value in case the robot gets close to the light but then is forced to move away from the light by a bucket.

2.3.1 find_light function.

The find_light() state machine is responsible for finding the brightest light around the robot. The basic idea is that the robot spins for a fixed amount of time to ensure that the robot completes at least one full revolution. During this initial spin the robot is looking for the maximum intensity. The robot then spins a second time and stops when it finds the maximum intensity from the first spin. This simple operation is further enhanced by stopping the initial spin (and skipping the second spin) if it finds an intensity above some threshold. This behavior is useful because we do not want the robot to continue the search if the light is nearby.

2.4 Helper functions

- find_bump()
- find_bucket()
- found_bucket()
- bump_light_routine()

Section 3 – Testing

The testing process was quite revealing. We tried to test the robot at all phases of development. Initially we tried a multiple process approach similar to the *Mobile Robots: Inspiration to Implementation* code. However, we found that the multi-process approach was somewhat unpredictable on the Handyboard. The robot would see things that simply were not there and would behave erratically. Therefore, we completely changed our approach to the code. Moving to a state machine approach seemed to provide more stability to the execution of the code, however, it was significantly more hindering to read and understand.

Section 4 – Conclusions

In conclusion, our design was marginally successful, having a high score of only 1 point. The robot was able to score two or three points but would loose them by getting stuck or by hitting a bucket. It seems that our design was simply too large for the arena. We would get stuck in corners and could not squeeze through narrow gaps between buckets and found no way to avoid this without completely redesigning the robot. Moreover, on our final run, we attempted to change too much, resulting in a range-sensing bug that destroyed our run.

Finally, it seems that our choice to re-write the code shortened our timeframe. If our initial coding attempts had been more successful, we would have had more time to work out the detail specific bugs in the new code.

Appendix A – Source Code

```
/*****************************************************************
*   Name:         Group 7
*            Robert Moe
*            John Zumwalt
*            Mark Woehrer
*            Celi Sun
*   Class:               CS 4970.1 Intro to Intelligent Robots
*   Instructor:  Dr. Hougen
*   Date:                03/31/03
*   FileName:  main_test.ic
*****************************************************************

#define LEFT_LIGHT 2
#define RIGHT_LIGHT 3
#define RIGHT_MOTOR 0
#define LEFT_MOTOR 1
#define LEFT_TRACK 3
#define RIGHT_TRACK 2
#define FRONT_BUMP 7

#define LEFT_RANGE 16
#define RIGHT_RANGE 17
#define RANGE_THRESH 65
#define DETECT_RIGHT 2
#define DETECT_LEFT 3
#define DETECT_NONE 0

#define DEAD_ZONE 25

#define LEFT_SERVO_CENTER 2300
#define RIGHT_SERVO_CENTER 2800
#define SERVO_DELTA 200
#define SERVO_TIME 10
#define SEEK_TIMEOUT 1500

#define left_servo servo0
#define right_servo servo2


/*****************************************************************
* Function: move_servos(void)
* Inputs: none
* Global Variables:
*   int move_servos_state       state indicator of the servo state
*   int move_servos_count       countdown of the servo swing delta
*   int kill_servo_swing        stop the servos from swinging
```

```
*********************************************************************/
int move_servos_state;
int move_servos_count;
int kill_servo_swing = 0;

void move_servos (void){
    if(move_servos_state==0) {             // Initialize Servo
        move_servos_count=SERVO_TIME;
        move_servos_state=1;

    }
    else
      if(move_servos_state==1){            // Swing L&R Servo FWD
          if(move_servos_count >=0 ){
              move_servos_count--;
          }else{
              move_servos_count=SERVO_TIME;
              move_servos_state=2;

                left_servo=LEFT_SERVO_CENTER+SERVO_DELTA;
                right_servo=RIGHT_SERVO_CENTER-SERVO_DELTA;

          }
      }
      else
        if(move_servos_state==2){          // Swing L&R Servo BACK
          if(move_servos_count >=0 ){
              move_servos_count--;
          }else{
               move_servos_count=SERVO_TIME;
                move_servos_state=1;
                left_servo=LEFT_SERVO_CENTER-SERVO_DELTA;
                right_servo=RIGHT_SERVO_CENTER+SERVO_DELTA;
          }

        }
        else                               // Should never get here
        {
            beep();
        }

}

/*********************************************************************
* Function: seek(void)
* Inputs: none
* Global Variables:
*   int seekfun
```

```
*   int seek_state
*   int seek_timeout
*   float seek_u_old
*   float seek_e_old
*   float seek_timeout
**********************************************************************/
float seek_u_old;
float seek_e_old;
float seek_K=0.0;
int seekrun;
int seek_state;
int seek_timeout;

int seek (void)
{
   float seek_u;
   float seek_e;
   int left_out,right_out;
   int sum;

   seek_e_old=seek_e;
   seek_e=(float)analog(LEFT_LIGHT)-(float)analog(RIGHT_LIGHT);

   sum=analog(LEFT_LIGHT)+analog(RIGHT_LIGHT);

   {
      {
         //Graduallyl increase the seek_K Gain as the robot moves
         //  closer to the light to better increase accuracy.
         if (seek_state == 0){
            seek_K=0.5;
            seek_state=1;
         }
         else
           if (seek_state == 1){
              if(sum <=16){
                 seek_K=8.0;
                 seek_state=2;
              }
           }
           else
             if (seek_state == 2){
                if(sum <=12){
                   seek_K=20.0;
                   seek_state=3;
                }
             }
             else
```

```
            if (seek_state == 3){
              seek_K=70.0;
              seek_state=4;
              seek_timeout=SEEK_TIMEOUT;

            }
            else
             if (seek_state == 4){
                if(seek_timeout >= 0){
                   seek_timeout--;

                }else{
                   return 1;
                }
             }
            else
             if (seek_state == 5){

             }
              return 0;

      }
   }

   // Seek function for motor output.
   seek_u_old=seek_u;
   seek_u = seek_u_old + seek_K * (seek_e - seek_e_old);

   left_out =(int)(100.0+seek_u);
   right_out=(int)(100.0-seek_u);

   motor(LEFT_MOTOR, left_out);
   motor(RIGHT_MOTOR,right_out);
}


/*****************************************************************
* Function: find_light(void)
* Inputs: none
* Global Variables:
*   int find_light_state
*   int find_light_max_count
*   int find_light_max
*   int find_light_min
*   int find_light_status
*   int find_direction
*****************************************************************/
int find_light_state;
```

```
int find_light_max_count;
int find_light_max,find_light_min;
int find_light_status=0;
int find_direction=0;

int find_light (void){
   int my_sum;

   if(find_light_state==0){

      find_light_status=0;
      find_light_max_count=500;
      find_light_max=0;
      find_light_min=10000;

      if(find_direction){
         kill_servo_swing = 1;
         motor(LEFT_MOTOR, -100);
         motor(RIGHT_MOTOR,+100);
         motor(LEFT_TRACK, -100);
         find_direction=1;
      }else{
         kill_servo_swing = 1;
         motor(LEFT_MOTOR, +100);
         motor(RIGHT_MOTOR,-100);
         motor(RIGHT_TRACK, -100);
         find_direction=1;
      }

      find_light_state=1;
   }else
    if(find_light_state==1){
       if(find_light_max_count>0){
          find_light_max_count--;

          my_sum=analog(LEFT_LIGHT)+analog(RIGHT_LIGHT);

          if(my_sum <= 20) {
             //beep();beep();beep();beep();beep();beep();
             motor(RIGHT_MOTOR, 0);
             motor(LEFT_MOTOR, 0);
             motor(RIGHT_TRACK, 100);
             motor(LEFT_TRACK, 100);
             kill_servo_swing = 0;
             return 1; }


          if(my_sum>find_light_max) find_light_max=my_sum;
```

```
        if(my_sum<find_light_min) find_light_min=my_sum;
      }else{
        //printf("sum:%d min:%d\n", my_sum,find_light_min);
        find_light_state=2;
      }
  }
  else
   if(find_light_state==2){
      //motor(LEFT_MOTOR, 0);
      //motor(RIGHT_MOTOR,0);
      find_light_max_count=500;
      find_light_state=3;
      beep();
   }
   else
    if(find_light_state==3){

        if(find_light_max_count>0){
          find_light_max_count--;

          my_sum=analog(LEFT_LIGHT)+analog(RIGHT_LIGHT);
          if(my_sum<=find_light_min+1){ //!!!
            find_light_state=4;
            if(my_sum <= 20) { //!!!mkw change me
              motor(RIGHT_TRACK, 100);
              motor(LEFT_TRACK, 100);
              kill_servo_swing = 0;
              find_light_status=1;
            }
            else {
              motor(RIGHT_TRACK, 100);
              motor(LEFT_TRACK, 100);
              kill_servo_swing = 0;
              find_light_status=2;
            }
            motor(LEFT_MOTOR, 0);
            motor(RIGHT_MOTOR,0);
          }
        }
        else {
          find_light_state = 4;
          find_light_status = 2;
        }
    }
    else
     if(find_light_state==4){
        motor(LEFT_MOTOR, +100);
        motor(RIGHT_MOTOR,-100);
```

```
        find_light_state=5;
      }
      else
       if(find_light_state==5){
          motor(LEFT_MOTOR, 0);
          motor(RIGHT_MOTOR,0);
          return find_light_status;
       }

       //return status;

}

/*******************************************************************
* Function: find_bump(void)
* Inputs: none
* Global Variables:
*   none
*******************************************************************/

int find_bump() {
   return digital(7);
}

/*******************************************************************
* Function: find_bucket(void)
* Inputs: none
* Global Variables:
*   none
*******************************************************************/
int find_bucket() {
   int lrange = analog(LEFT_RANGE);
   int rrange = analog(RIGHT_RANGE);
   if(lrange >= RANGE_THRESH) {
      //printf("\nlrange=%d", lrange);
      return DETECT_LEFT;
   }
   if(rrange >= RANGE_THRESH) {
      //printf("\nrrange=%d", rrange);
      return DETECT_RIGHT;
   }
   //c0cac0la
   //printf("\n");
   return DETECT_NONE;
}

/*******************************************************************
* Function: backup_and_turn(void)
```

```
* Inputs: none
* Global Variables:
*   int avoid_state
*   int backstate
*   int direction
*   int turncount
*   int forwardcount
******************************************************************/
int avoid_state;
int backcount;
int direction;
int turncount;
int forwardcount;

int backup_and_turn() {

    if(avoid_state==0){
        backcount = 25;
        forwardcount = 80;
        direction = analog(RIGHT_LIGHT)- analog(LEFT_LIGHT);
        motor(RIGHT_MOTOR, -100);
        motor(LEFT_MOTOR, -100);
        avoid_state=1;
    }
    else if(avoid_state==1){
         if(backcount == 0) {
             avoid_state = 2;
             turncount = 10;
         } else {
             backcount--;
         }
     }
     else if(avoid_state==2){
          if(turncount == 0) {
              avoid_state = 3;
          }
          else  {
              if(direction >= 0) {
                  motor(LEFT_MOTOR, -100);
                  motor(RIGHT_MOTOR, 100);
              } else {
                  motor(LEFT_MOTOR, 100);
                  motor(RIGHT_MOTOR, -100);
              }
              turncount--;
          }
      }
      else if(avoid_state==3) {
```

```
            if(forwardcount == 0) {
                return 1;
            }
            else {
                if(!find_bump()) {
                    motor(RIGHT_MOTOR, 100);
                    motor(LEFT_MOTOR, 100);
                }
                else {
                    avoid_state = 0;
                }
                forwardcount--;
            }
        }
        return 0;
}

/*********************************************************************
* Function: avoid_bucket(void)
* Inputs: none
* Global Variables:
*   int ab_state
*   int ab_direction
*   int ab_turncount
*   int ab_pausecount
*   int ab_drivecount
*********************************************************************/
int ab_state;
int ab_direction;
int ab_turncount;
int ab_pausecount;
int ab_drivecount;

int avoid_bucket() {

    if(ab_state==0){
        ab_pausecount = 50;
        motor(RIGHT_MOTOR, 0);
        motor(LEFT_MOTOR, 0);
        ab_state=1;
    }
    else if(ab_state==1){
          if(ab_pausecount == 0) {
              ab_state = 2;
              ab_turncount = 25;
          } else {
              ab_pausecount--;
```

```
        }
    }
    else if(ab_state==2){
        if(ab_turncount == 0) {
            ab_drivecount = 60;
            ab_state = 3;
        }
        else  {
            if(ab_direction == DETECT_LEFT) {
                motor(LEFT_MOTOR, 100);
                motor(RIGHT_MOTOR, -100);
            } else if(ab_direction == DETECT_RIGHT) {
                motor(RIGHT_MOTOR, 100);
                motor(LEFT_MOTOR, -100);
            }
            ab_turncount--;
        }
    }
    else if(ab_state==3) {

        if(ab_drivecount == 0) {
            return 1;
        }
        else {
            if(find_bucket() != DETECT_NONE) {
                ab_state = 0;
                ab_direction = find_bucket();
                return 0;
            }
            if(!find_bump()) {
                motor(RIGHT_MOTOR, 100);
                motor(LEFT_MOTOR, 100);
            }
            else {
                motor(RIGHT_MOTOR, -100);
                motor(LEFT_MOTOR, -100);
            }

            ab_drivecount--;
        }
    }
    return 0;
}


/*****************************************************************
* Function: found_bucket(int frange)
* Inputs: frange
```

```
* Global Variables:
*   none
*********************************************************************/
void found_bucket(int frange) {
   ab_state=0;
   ab_direction = frange;
}

/*********************************************************************
* Function: touch_light(void)
* Inputs: none
* Global Variables:
*   int touch_light_state
*   int touch_light_timeout
*********************************************************************/
int touch_light_state;
int touch_light_timeout;

int touch_light() {

   if((seek_state ==
4)&&(analog(LEFT_LIGHT)+analog(RIGHT_LIGHT))>=LIGHT_THRESHOLD) {
      return 1;
   }
      return 0;

}

/*********************************************************************
* Function: main(void)
* Inputs: none
* Global Variables:
*   none
*********************************************************************/
int LIGHT_THRESHOLD;
void main()
{
   long time_old;
   int main_state;
   int prev_state;
   int countdown;
   int found_range;
   int skip_bump;
   int print_count;
   int servoRcount;
   int servoLcount;
```

```
while(!start_button()){
   LIGHT_THRESHOLD = 2 * knob();
   printf("%d\n", 2* knob());
}

printf("Press START\n");
while(!start_button());

printf("GO\n");

beep();

motor(RIGHT_TRACK,100);
motor(LEFT_TRACK,100);

init_expbd_servos(1); //center

left_servo=LEFT_SERVO_CENTER;
right_servo=RIGHT_SERVO_CENTER;

time_old=mseconds();


main_state=0;
move_servos_state=0;
while(!stop_button()){
   if(!kill_servo_swing) {
      move_servos();
   }
   else {
      left_servo = 1000;
      right_servo = 4000;
   }

   if(main_state == 0){ // INIT
      find_light_state=0;
      main_state=1;
      seek_state = 0;
      //main_state=3;
      skip_bump=0;
      kill_servo_swing = 1;

   }
   else
     if(main_state == 1){ // FIND LIGHT
        int tmp;
        found_range = find_bucket();
```

```
    if(found_range != DETECT_NONE) {
        found_bucket(found_range);
        main_state = 6;
        prev_state = 0;
    }
    tmp=find_light();
    if(tmp==1){
        main_state=2;
        countdown=100;
        beep();
    }
    else if(tmp == 2) {
        countdown = 300;
        main_state=5;
        motor(RIGHT_MOTOR, 100);
        motor(LEFT_MOTOR, 100);
    }
}
else
  if(main_state == 2){ // PAUSE
      if(countdown == 0){
          main_state=3;
          seek_state=0;
          touch_light_state = 0;
          touch_light_timeout = 10;
          motor(RIGHT_MOTOR, 100);
          motor(LEFT_MOTOR, 100);
      }else{
          countdown--;
      }

  }
  else
    if(main_state == 3){ // SEEK
        skip_bump = 0;
        found_range = find_bucket();
        if(found_range != DETECT_NONE) {
            found_bucket(found_range);
            main_state = 6;
            prev_state = 3;
        }
        if(touch_light()) {
            main_state = 0;
            skip_bump = 1;
            //beep();beep();beep();beep();
            //motor(RIGHT_MOTOR, -100);
            //motor(LEFT_MOTOR, -100);
            //msleep(50l);
```

```
        //motor(RIGHT_MOTOR, 0);
        //motor(LEFT_MOTOR, 0);
        seek_state = 0;
        find_light_state=0;
    }
    if(find_bump() && !skip_bump) {
        main_state=4;
        prev_state=3;
        avoid_state = 0;
    }
    skip_bump = 0;

    if(seek()) {
        main_state=0;
    }
    //seek();
}
else if(main_state == 4) {
      found_range = find_bucket();
      if(found_range != DETECT_NONE) {
          found_bucket(found_range);
          main_state = 6;
          prev_state = 3;
      }
      else if(backup_and_turn()) {
          main_state = prev_state;
      }
 }
  else if(main_state == 5) {
        kill_servo_swing = 0;
        if(countdown == 0) {
            main_state = 0;
        }
        else {
            found_range = find_bucket();
            if(found_range != DETECT_NONE) {
                found_bucket(found_range);
                main_state = 6;
                prev_state = 5;
            }
            if(find_bump()) {
                prev_state = 5;
                main_state = 4;
                avoid_state = 0;
                beep();
            }
            countdown--;
        }
```

```
            }
            else if(main_state == 6) {
                kill_servo_swing = 0;
                if(avoid_bucket()) {
                    motor(RIGHT_MOTOR, 100);
                    motor(LEFT_MOTOR, 100);
                    found_range = DETECT_NONE;
                    main_state = prev_state;
                }
             }
             else if(main_state == 7) {
                 motor(RIGHT_MOTOR, 100);
                 motor(LEFT_MOTOR, 100);
                 main_state = 0;
              }




             //printf("\nState=%d", main_state);

        if(print_count<10){
           print_count++;

        }else{
           printf("\nState=%d", main_state);
           print_count=0;
        }


        while((mseconds()-time_old) < 10L){}
        time_old=mseconds();
    }

    init_expbd_servos(0);

    motor(LEFT_MOTOR, 0);
    motor(RIGHT_MOTOR,0);
    motor(2,0);
    motor(3,0);

}
```