```
/* Project - 2
   TEAM 8
   Date: 31st March, 2003

   The aim of this project is to make the robot follow the light and switch
   off the
   light by touching it, while avoiding rocks and escaping big obstacles
   (buckets)
   in the arena.
*/

#define L_MOTOR 1 // left motor
#define R_MOTOR 3 // right motor
#define LEFT_BUMPER 7  // left bumper
#define RIGHT_BUMPER 13 // right bumper
#define MIDDLE_LEFT_BUMPER 15  // middle left bumper
#define MIDDLE_RIGHT_BUMPER 11  // middle right bumper
#define LEFT_RANGE_SENSOR 16   // left range finder
#define RIGHT_RANGE_SENSOR 19    // right range finder

// various powers given to motors
#define POWER 100
#define REVERSEPOWER -10
#define STOP_POWER -20

// various command variables used
#define TRUE 1
#define FALSE 0
#define GO_STRAIGHT 1
#define AVOID_LEFT_COLLISION 2
#define AVOID_RIGHT_COLLISION 3
#define AVOID_STRAIGHT_COLLISION 4
#define LEFT_EVASIVE_ACTION 5
#define RIGHT_EVASIVE_ACTION 6
#define STRAIGHT_EVASIVE_ACTION 7
#define NULL 8
#define MIN_VALUE=45;

int wander_flag;
int wander_command;
int avoid_obstacles_flag;
int avoid_obstacles_command;
int motor_input;
int escape_obstacles_flag;
int escape_obstacles_command;
int left_bumped, right_bumped, middle_left_bumped, middle_right_bumped;


//for follow_light
#define LIGHT_FRONT_RIGHT  5
#define LIGHT_FRONT_LEFT   4
#define LIGHT_BACK_LEFT    2
#define LIGHT_BACK_RIGHT   3
#define BACK_STRONG_SENSORS  0
#define FRONT_STRONG_SENSORS 1
#define NO_LIGHT     200


int light_front_R;
int light_front_L;
int light_back_L;
int light_back_R;
int no_light;

#define FOLLOW_LIGHT_RIGHT       10
#define FOLLOW_LIGHT_LEFT        11
#define FOLLOW_LIGHT_SHARP_LEFT  12
```

```
#define FOLLOW_LIGHT_SHARP_RIGHT 13

#define RESOLVE_RIGHT 14

void main()
{
    // test_follow();
    while(!start_button());
    start_process(wander());
    start_process(follow_light());
    start_process(avoid_obstacles());
    start_process(escape_obstacles());
    start_process(arbitrate());

}

// wander behavior
void wander()
{
    while(1)
      {
        wander_flag=TRUE;
        wander_command = GO_STRAIGHT;

        defer();
    }
}


// obstacle avoidance behavior
void avoid_obstacles()
{
    start_process(checkbumpers());
    while(1)
      {
        if (left_bumped)  // left obstacle detected
          {
            avoid_obstacles_flag=TRUE;
            avoid_obstacles_command = AVOID_LEFT_COLLISION;
            sleep(0.75);
        }
        else if(middle_left_bumped || middle_right_bumped)  // hit in the
        center
            {
              avoid_obstacles_flag=TRUE;
              avoid_obstacles_flag=AVOID_STRAIGHT_COLLISION;
              sleep(0.75);
          }
          else if(right_bumped)         // left obstacle detected
              {
                avoid_obstacles_flag=TRUE;
                avoid_obstacles_command=AVOID_RIGHT_COLLISION;
                sleep(0.75);
            }
            else   // cool. nothing happened
              avoid_obstacles_flag=FALSE;
        defer();
    }
}


// getting readings from the four bumpers
void checkbumpers()
{
    left_bumped  =digital(LEFT_BUMPER);
    right_bumped =digital(RIGHT_BUMPER);
    middle_left_bumped=digital(MIDDLE_LEFT_BUMPER);
```

```
    middle_right_bumped = digital(MIDDLE_RIGHT_BUMPER);
}


// arbitration module, subsuming and fusion takes place
void arbitrate(){
    avoid_obstacles_flag=FALSE;
    follow_light_flag=TRUE;     //default is TRUE, should be!!! as it is the
    lowest behavior
    escape_obstacles_flag=FALSE;
    previous_command = FOLLOW_LIGHT_LEFT;
    while(1){
        if(escape_obstacles_flag == TRUE)  // escape behavior is active
          {
            printf("\n ESCAPE...");
            motor_input == escape_obstacles_command;
            execute_motor_commands();
        }
        else if(avoid_obstacles_flag == TRUE)   // avoid behavior is active
            {
              printf("\n AVOID...");
              motor_input = avoid_obstacles_command;
              execute_motor_commands();
          }
          else if (follow_light_flag==TRUE){   // follow light behavior is
          active
                printf("\n LIGHT...");
                motor_input=follow_light_command;
                execute_motor_commands();
            }
            else if (follow_light_flag==TRUE){   // wander behavior is active
                  printf("\n WANDER...");
                  motor_input=wander_command;
                  execute_motor_commands();
              }

              else if (wander_flag==TRUE){   // wander behavior is active
                    printf("\n WANDER...");
                    motor_input=wander_command;
                    execute_motor_commands();
                }

                defer();
    } //while
}


// calls the relevant functions for the active behaviors, which eventually
give
// powers to motors to steer the robot
void execute_motor_commands()
{
    if(motor_input == LEFT_EVASIVE_ACTION)
      left_evasive_action(0);
    else if(motor_input == RIGHT_EVASIVE_ACTION)
        right_evasive_action(0);
      else if (motor_input == STRAIGHT_EVASIVE_ACTION)
          straight_evasive_action();
        else if (motor_input == AVOID_LEFT_COLLISION){
              avoid_left_collision();
          }else if(motor_input == AVOID_STRAIGHT_COLLISION)
              avoid_straight_collision();
            else if(motor_input == AVOID_RIGHT_COLLISION){
                  avoid_right_collision();
              }else if(motor_input == FOLLOW_LIGHT_RIGHT){
                    go_mild_right();
                    sleep(.15);
```

```
                }else if(motor_input == FOLLOW_LIGHT_LEFT){
                      go_mild_left();

                  }else if(motor_input == FOLLOW_LIGHT_SHARP_LEFT){
                        go_mild_left();
                        sleep(1.0);
                        go_straight();
                        sleep(0.25);
                    }else if(motor_input == FOLLOW_LIGHT_SHARP_RIGHT){
                          go_mild_right();
                          sleep(1.0);

                      }else if(motor_input == GO_STRAIGHT) {
                            go_straight();
                            sleep(0.25);
                        }

}

// escape behavior
int left_range_sensor, right_range_sensor, left_obstacle, right_obstacle;
int iamdone=1;
void escape_obstacles()
{
    while(1)
      {
        if(iamdone == 1)
          {

            left_obstacle = 0;
            right_obstacle = 0;
            check_range_sensors();
            if (left_range_sensor > MIN_VALUE)   // bucket on left side
              {
                escape_obstacles_flag=TRUE;
                left_obstacle = 1;
            }
            else if (right_range_sensor > MIN_VALUE) // bucket on right side
                {
                  escape_obstacles_flag = TRUE;
                  right_obstacle = 1;
              }
              if (left_obstacle == 1){
                iamdone=0;
                motor_input = LEFT_EVASIVE_ACTION;
            }
            else if (right_obstacle == 1)
                {
                  iamdone = 0;
                  motor_input = RIGHT_EVASIVE_ACTION;
              }
              else if(left_obstacle == 1 && right_obstacle == 1) // bucket on
              both sides or exactly ahead
                  {
                    iamdone = 0;
                    motor_input = STRAIGHT_EVASIVE_ACTION;
                }
                else escape_obstacles_flag=FALSE;    // cool nothing happened
        }
    }
}


// reads the range finders readings
void check_range_sensors()
{
    int port;
```

```
    port = LEFT_RANGE_SENSOR;
    left_range_sensor = get_average_reading(port);
    port = RIGHT_RANGE_SENSOR;
    right_range_sensor = get_average_reading(port);
}

// gives the average of ten continuous readings of the range finders
int get_average_reading(int port)
{
    int sum=0;
    int i=0;

    while(i<15){
        sum+=analog(port);
        msleep(20L);
        i++;
    }
    sum=(int)sum/i;

    return sum;
}

// taking evasive action when there is obstacle on the left
void left_evasive_action(int option)
{
    stop_wheels();
    sleep(0.5);
    go_back();
    sleep(0.75);
    while(1)
      {
        beep();
        stop_wheels();
        check_range_sensors();
        if (left_range_sensor < MIN_VALUE && right_range_sensor < MIN_VALUE)
          break;
        else
          {
            go_back();
            sleep(0.5);
        }
        if(option == 0)
          {
            go_more_left();
            sleep(1.0);
        }
        else if(option == 1)
            {
              go_more_left();
              sleep(1.0);
          }

    }

    go_more_left();
    sleep(0.25);

    iamdone = 1;
}

// taking evasive action when there is obstacle on the right
void right_evasive_action(int option)
{
    stop_wheels();
    sleep(0.5);
    go_back();
    sleep(0.75);
```

```
    while(1)
      {
        beep();
        stop_wheels();
        check_range_sensors();
        if (right_range_sensor < MIN_VALUE && left_range_sensor < MIN_VALUE)
          break;
        else
          {
            go_back();
            sleep(0.5);
        }

        if(option == 0)
          {
            go_more_right();
            sleep(1.0);
        }
        else if(option == 1)
            {
              go_more_right();
              sleep(1.0);
          }
    }
    go_more_right();
    sleep(0.5);
    iamdone = 1;
}

// taking evasive action when there is obstacle on both sides or
// exactly in the front
void straight_evasive_action()
{
    int randomnum;
    beep();
    stop_wheels();
    randomnum = random(2);
    if(randomnum == 0) {
        left_evasive_action(1);
    }
    else if(randomnum == 1) {
          right_evasive_action(1);
      }
      iamdone = 1;
}

void right_loop(){

    motor(L_MOTOR,-50);
    motor(R_MOTOR,-50);
    sleep(0.5);
    motor(L_MOTOR, 50);
    //motor(R_MOTOR, );
    sleep(1.0);
    motor(L_MOTOR,50);
    motor(R_MOTOR,50);
    sleep(1.0);

}

void avoid_left_collision()
{

    stop_wheels();
    go_back();
    sleep(0.25);
    go_mild_right();
```

```
    sleep(0.5);
}

void avoid_straight_collision()
{
    int randomnum;
    stop_wheels();
    go_back();
    sleep(0.25);
    randomnum = random(2);
    if (random == 0) {
        go_more_left();
    } else if(random == 1) {
          go_more_right();
      }

      sleep(0.5);
}

void avoid_right_collision()
{
    stop_wheels();
    go_back();
    sleep(0.25);
    go_mild_left();
    sleep(0.25);
}

void go_straight()
{

    motor(L_MOTOR, POWER-20);
    motor(R_MOTOR, POWER-20);
}

void go_back()
{
    motor(L_MOTOR, -1 * POWER-40);
    motor(R_MOTOR, -1 * POWER-40);
}

/* reverses motors briefly for a quick stop */
void stop_wheels()
{
    motor(L_MOTOR,STOP_POWER);
    motor(R_MOTOR,STOP_POWER);
    sleep(0.25);
    ao();
}

void go_mild_left()
{
    motor(L_MOTOR,REVERSEPOWER*1);
    motor(R_MOTOR,POWER-40);
}

void go_mild_right()
{
    motor(L_MOTOR,POWER-40);
    motor(R_MOTOR,REVERSEPOWER*1);
}

void go_more_right()
{
    motor(L_MOTOR,POWER);
    motor(R_MOTOR,REVERSEPOWER*1);
}
```

```
void go_more_left()
{
    motor(R_MOTOR,POWER);
    motor(L_MOTOR,REVERSEPOWER*1);
}

/*****************************************************************************
***
//for follow_light
*****************************************************************************
***/

int follow_light_flag;
int follow_light_command;

void follow_light(){

    int strong_sensor;

    follow_light_flag=TRUE;
    follow_light_command=GO_STRAIGHT;
    while(1){

        read_light_sensors();

        //if no light is detected do this
        if((light_front_L >= NO_LIGHT)&&(light_front_R >=NO_LIGHT)&&

                          (light_back_L>=NO_LIGHT)&&(light_back_R>=NO_LIGHT))
                          {
            follow_light_command = GO_STRAIGHT;
            follow_light_flag=TRUE;
        }
        else{

            strong_sensor=strong_sensor_is();
            //printf("L:%d, R:%d, BL:%d, BR:%d",light_front_L,light_front_
             // R,light_back_L,light_back_R);
            //printf("s: %d",strong_sensor);
            //printf("$: ,%d\n", follow_light_command);
            // sleep(0.5);


            if(strong_sensor==FRONT_STRONG_SENSORS){
                //forward
                follow_light_command=GO_STRAIGHT;
                follow_light_flag=TRUE;

            }else if(strong_sensor==BACK_STRONG_SENSORS){
                  follow_light_command=FOLLOW_LIGHT_LEFT;
                  follow_light_flag=TRUE;

              }else if (strong_sensor==LIGHT_FRONT_LEFT){
                    follow_light_command =FOLLOW_LIGHT_LEFT;
                    follow_light_flag=TRUE;

                }else if (strong_sensor==LIGHT_FRONT_RIGHT){
                      follow_light_command =FOLLOW_LIGHT_RIGHT;
                      follow_light_flag=TRUE;

                  }else if (strong_sensor==LIGHT_BACK_RIGHT){
                        follow_light_command =FOLLOW_LIGHT_SHARP_RIGHT;
                        //follow_light_command =FOLLOW_LIGHT_RIGHT;
                        follow_light_flag=TRUE;

                    }else if (strong_sensor==LIGHT_BACK_LEFT){
```

```
                            follow_light_command =FOLLOW_LIGHT_SHARP_LEFT;
                            // follow_light_command =FOLLOW_LIGHT_LEFT;
                            follow_light_flag=TRUE;

                        }
        }//first else
        //sleep(0.4);
        defer();
    }//while

}//follow_light

/*THis function calculates what sensor has the strongest readings
if it does not find find one returns LIGHT_NO_STRONG_SENSOR havue
*/

int strong_sensor_is(){

    int delta_front;
    int delta_back;
    int tolerance_front;
    int tolerance_back;
    int front_strong_sensor;    //= LIGHT_LEFT (if left), =LIGHT_RIGHT (if
    right)
    int back_strong_sensor;     //= LIGHT_BACK_LEFT (if back left),
    =LIGHT_BACK_RIGHT (if back right
    int strong_sensor;
    //all tollerance levels are set to 17%
    float error = 0.17;  //0.17

    strong_sensor=LIGHT_FRONT_LEFT; //default but should never happen!!!
    delta_front=light_front_L-light_front_R;
    delta_back =light_back_L -light_back_R;
    tolerance_front= (int)((float)(light_front_L + light_front_R)*0.5*error);
    tolerance_back = (int)((float)(light_back_L  + light_back_R)*0.5*error);

    //is back or front light is stronger?
    if( (light_front_L+light_front_R) <= (light_back_L+light_back_R) ){

        //which one of front sensors gets more light?
        if(abs(delta_front) > tolerance_front){
            if(delta_front>0){strong_sensor=LIGHT_FRONT_RIGHT;}
            else {strong_sensor=LIGHT_FRONT_LEFT;}

        }//the sensors get almost the same light.
        else{strong_sensor=FRONT_STRONG_SENSORS;}

        //back sensors get more light
    }else{
        //which one of back sensors gets more light?
        if(abs(delta_back) > tolerance_back){
            //back right is getting more light
            if(delta_back>0){strong_sensor=LIGHT_BACK_RIGHT;}
            //back left is getting more light
            else {strong_sensor=LIGHT_BACK_LEFT;}

        }//end if
        else{strong_sensor=BACK_STRONG_SENSORS;}
    }//last else

    return strong_sensor;
}//strong sensor readings


int abs(int num){
    if(num <0)
      return -num;
```

```
    else
      return num;
}//abs


void read_light_sensors(){

    light_front_R = analog(LIGHT_FRONT_RIGHT);
    light_front_L = analog(LIGHT_FRONT_LEFT);
    light_back_L  = analog(LIGHT_BACK_LEFT);
    light_back_R  = analog(LIGHT_BACK_RIGHT);
}//end of read_light_sensors


//resolve

int resolve_command;
int resolve_flag;

void resolve(){

    int current_sum=0;
    int previous_sum=0;
    int moved_enough=0;
    float sample_rate = 15.0;  //seconds
    float error2=0.12;  //this is the %


    current_sum=light_front_R + light_front_L;


    while(1){


        sleep(20.0);
        previous_sum=current_sum;
        current_sum =light_front_R + light_front_L;

        moved_enough = (int)((float)(abs(current_sum - previous_sum))*error2);

        //we are stuck!, do something

        if(abs(current_sum-previous_sum)<= moved_enough){
            beep();beep();beep();
            resolve_flag=TRUE;
            resolve_command = RESOLVE_RIGHT;
            sleep(2.0);
        }//end if
        else
          resolve_flag = FALSE;

        printf("D: %d, E: %d",abs(current_sum-previous_sum),moved_enough);
        printf(" cond: %d\n",resolve_flag);
    }//while



}//resolve

void test_follow(){
    int sensor;
    while(1){
        read_light_sensors();

        sensor=strong_sensor_is();
        printf("L:%d R:%d BL:%d BR:%d",light_front_L,light_front_R,
        light_back_L,light_back_R);
```

```
        //printf("follow_l: ,%d", follow_light_command);
        printf(" s: %d\n",sensor);
        sleep(2.0);
    }
}
```