

**TEAM 5**  
**Project #2**  
**Final Report**

**Ayesha Ahsan**  
**Steven Layton**  
**Ethan Martin**  
**Marty Thompson**

## Section 1: Hardware Documentation

### Introduction

The construction of the robot is one of the primary tasks for this project. The sub-team for the construction, called the H-team, consists of Ethan Martin and Steven Layton. Ethan is the Senior Member of the H-Team because he was on the H-Team for Project One. Construction was an ongoing process throughout the entire development of the robot. As issues arose with the performance of the robot, corrections in the initial, conceptual design of the robot needed to be made. These corrections may be due to flaws in the design, or by shortcomings of the various sensors used. These changes included the addition and removal of some hardware components. This paper will describe the final design of the robot as it was demonstrated, and some of the design changes the robot evolved through.

The robot consists of three main parts, all of which will be explained in detail:

- Transmission
- Chassis
- Sensory Equipment

### Transmission (Figures A and B)

The design of the transmission for our robot was taken from *The Definitive Guide to LEGO Mindstorms: Second Edition* by Dave Baum. Our robot uses a dual differential adder-subtractor transmission to navigate its way around the course. The transmission consists of a primary and secondary motor, each of which is responsible for a different part of navigation. The primary motor, located in the rear of the robot, is the main drive motor. This main motor powers the robot in a straight line either forwards or backwards. The secondary motor is the turning motor. Sending power to this motor turns the two drive wheels in opposite directions, allowing the robot to turn in place. In addition, sending power to both the primary and secondary motor at the same time can allow the robot to move in a variety of arcs and curves, depending upon the ration of power sent to the motors. The transmission design consists of 17 gears and two differentials. This allows the drive wheels to turn in the same direction when powered by the primary motor, and in different directions when powered by the secondary motor.

### Chassis (Figures C and D)

The chassis of the robot was constructed to be both as small as possible (as far as its footprint is concerned) and very sturdy. There is the possibility that the robot will come into contact with either buckets or rocks placed throughout the course, so our robot is constructed in a manner so that it will not fall apart if it strikes an object. The actual footprint of our robot is very small, not much bigger than the adder-subtractor transmission enclosed within. There are two 1-inch drive wheels with rubber tires in the back, and two 2.25-inch wheels in the front wrapped in electrical tape to reduce friction. The wheels in the front are meant to be dragged around when the robot is turning, so the less friction the better. The main structure of our robot goes upward from its base. The Handyboard, placed directly above the primary motor located at the rear of the robot, is

attached on both sides to the chassis by LEGOS pegs placed in the holes drilled into its sides. A servo, elevated by LEGOS, is placed in front of the Handyboard and on top of the secondary motor. This servo has a LEGO beam placed on top of it, and is responsible for moving the light sensors. Straddling the servo is a large rig designed to hold the range finders. This rig is hot-glued to the chassis for structural stability. Located in the extreme rear of the robot, behind the primary motor, are two yellow LEGO pieces that curve upward designed to hold a weight. This weight is necessary to keep weight on the drive wheels of the robot. Due to the floor being extremely dusty, without the weight, the robot is unable to turn because the drive wheels constantly slip. A large piece of steel wool is attached to the very front part of the chassis so that upon contact with the lamp, our robot will complete the circuit and turn the lamp off.

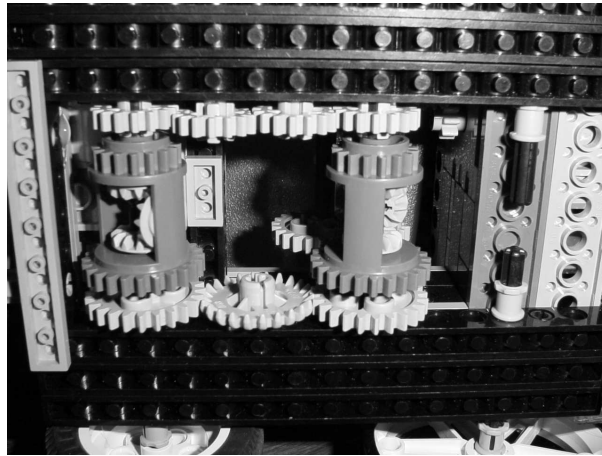
### **Sensory Equipment (Figures C and D)**

Three different types of sensors are used on our robot: range finders, light sensors, and bump sensors. The range finders are placed on a rig that straddles the servo, and is located towards the front of the robot. The rig elevates the range finders so that they are able to detect the large buckets, but are high enough to not detect the lamps and rocks. One range finder is placed on each side of the rig, so that both the left and right side of the robot has coverage. Two light sensors, one facing forwards and one facing backwards, are necessary to ensure that our robot finds the brightest source of light no matter where it is located. The light sensors are placed on a beam glued to a servo so that they may be swept in a circular motion and detect where the greatest source of light is located. To ensure the light sensors detect light only directly in front of them, black paper cones were constructed to give the light sensors “tunnel-vision”. In order for our robot to detect coming into contact with a rock or a boundary wall, two bump sensors are placed in front, one next to each wheel. A LEGO axle is dangled in front of each bump sensor so that they are able to detect contact with any location on the front of the robot.

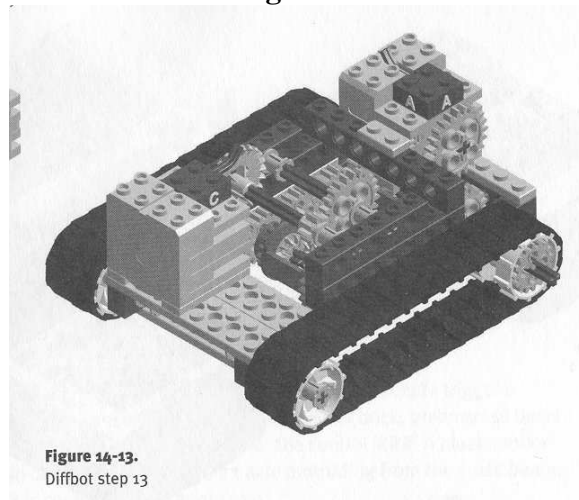
### **Problems We Encountered**

The C-Team encountered several problems which lead to various changes in the design. We had originally planned to place range finders on a servo that would constantly sweep back and forth and detect upcoming obstacles. Due to the limited performance of the range finders (we later discovered one was actually broken), the C-Team decided that static range finders would work much more consistently. During the second demonstration, one of the bump sensors fell off and started dragging the ground, confusing the robot very badly. We then came up with the current design of our bump sensors. Originally in our design, there was no rock placed in the back for weighing down the wheels. However, upon testing, it was immediately obvious that without more weight in the back over the drive wheels, our robot could barely turn.

# Figures

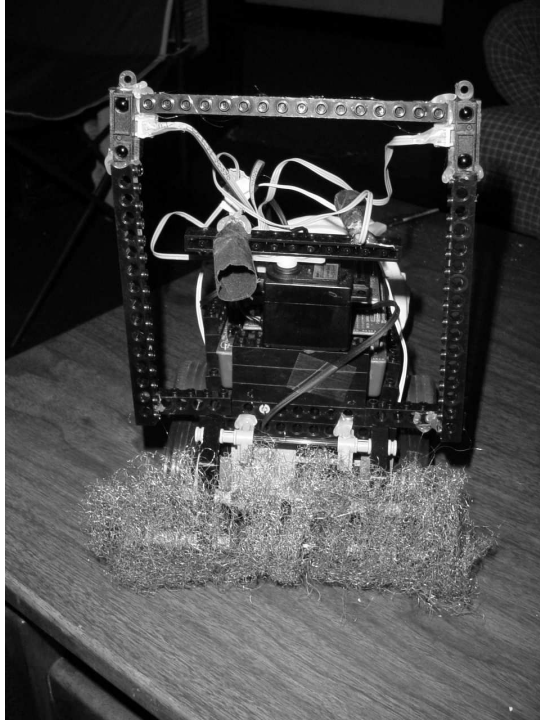


**Figure A**

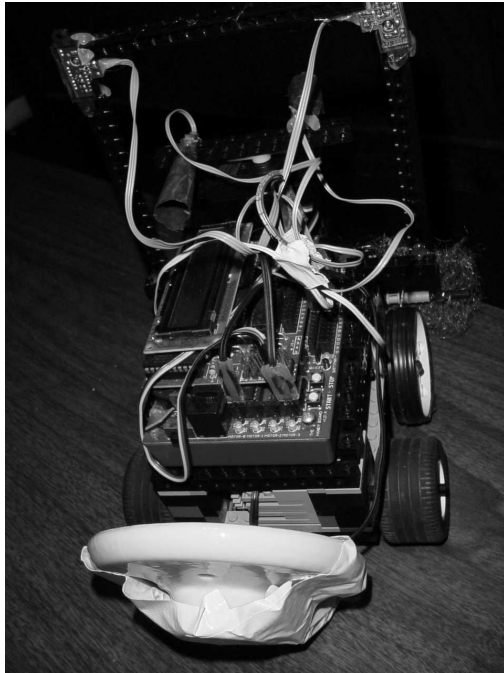


**Figure 14-13.**  
Diffbot step 13

**Figure B**



**Figure C**



**Figure D**

## Section 2: Code Documentation

### Introduction

The software design for the robot was one of the primary tasks for this project. The sub-team (C-team) for the code side of the project consisted of Ayesha Ahsan, who was the Senior member, and Marty Thompson, the Junior member. The initial prototype for the design of the software was to have a process for each set of sensors, but we quickly realized that the Handyboard could not handle three or more processes effectively. To resolve this, only two processes were used (excluding the main method). These processes allowed the robot to take environmental readings using light sensors and two bump sensors, and use the data obtained to make appropriate decisions that would lead the robot accomplishing its task (finding the next light while avoiding tall obstacles).

The processes used to ensure the success of the robot and will be noted here and described in detail in the following section:

- Find light caller process
- Avoid process

The architecture used from this project is most similar to the one proposed by Brooks. The avoid process implements the cruise and avoid behaviors and the light process implements the seek light behavior. The role of these behaviors is more clearly illustrated in the following sections.

### Find Light Caller Process

The find light caller process is a light seeking behavior that finds the strongest light source in the environment and turns the robot in the appropriate direction so that the robot is facing that light source. This behavior is the highest in the architecture subsuming over the avoid and cruise behavior. The process subsumes over the avoid process by killing the avoid process and restarting it after execution. Listed below is the algorithm implemented in the light seeking behavior.

- **Stop avoid process** – The avoid process was stopped so the find light process could have full control of the motors. When the avoid process was not stopped, the find light process could not effectively turn towards the strongest light source.
- **Measure ambient light and find strongest light source** – The measure of ambient light was determined by summing the readings of light from the front (or back, both were used) and dividing by the number of readings taken. This gave the robot a way to measure the distance from the light source. A high ambience implied the distance between the robot and the light source was large (and vice versa with a small ambience). This ambient measure of distance was used in calculating the tolerance, which was used in the turning algorithm explained below.

- **Turn in direction of light source** – After the best light source was found and the tolerance was computed, the robot turned in the appropriate direction until the light intensity was found to be within the tolerance of the strongest light source. To avoid an infinite turn, a counter was used to determine if the robot had turned too long. When the counter reached a specific value, the robot would turn in the opposite direction for a longer period of time with a higher tolerance. The increase of the tolerance on every successive turn ensured that the robot would head in the general direction of the light source.
- **Re-start avoid process** – Restarting the avoid process enabled the robot to travel to the next light source while avoiding tall obstacles and rocks.

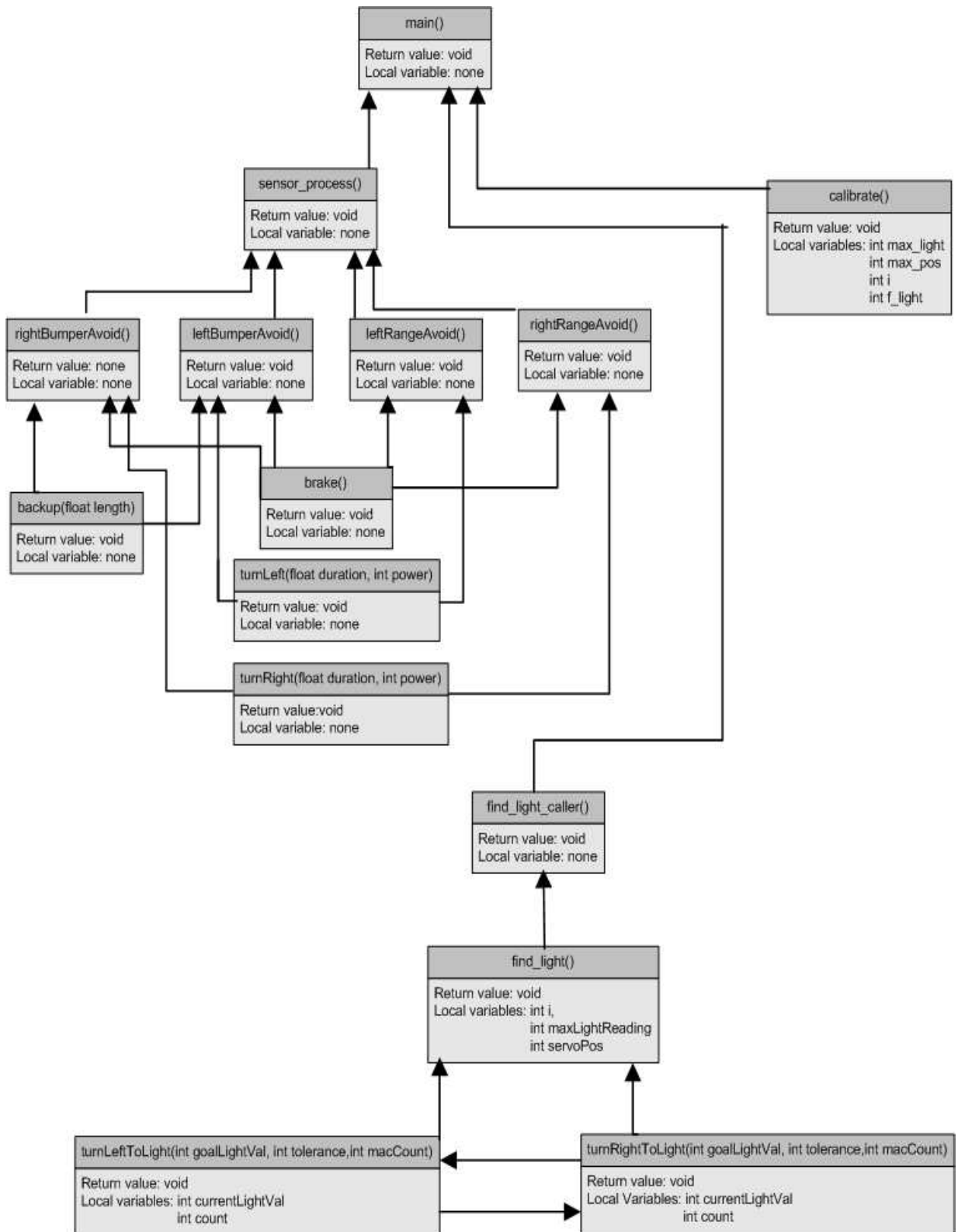
**N.B.** The sleeping time of the seek light behavior was determined in the testing phase of the project. We found that smaller sleep times would allow the robot to correct itself more often, and with a larger sleep time the robot would accomplish its goal in a lesser timeframe. Taking into account the relatively small size of the arena, the sleep time was set to ten seconds.

### **Avoid Process**

The avoid process is a combination of two behaviors: the cruise behavior and the avoid behavior. Although the cruise behavior is not explicitly expressed in the software design, it is implied in the avoid process. The cruise behavior was subsumed by the avoid behavior, and the avoid behavior subsumed by the seek light behavior. Listed below is the algorithm implemented in the avoid process.

- **Check left and right bump sensors** – The left and right bump sensors were checked and if either was triggered, a call was made to the associated bumper avoid function. If the left bumper was triggered, the robot would brake (thus subsuming over the cruise behavior), backup for approximately one second, turn right for one second, and resume the cruise behavior. The only difference between the right and left bumper avoid functions is that right bumper avoid would turn left instead of turning right.
- **Check left and right range finders** - The left and right range finders were checked and if either had a value above the minimum value allowed (implying that the object was too close), the robot would access the appropriate range avoid function. The avoid range function is as follows: brake (subsume over the cruise behavior), turn left or right until the object is no longer within the minimum distance, and continue cruising. The left range finder caused a right turn and the right range finder a left turn.

A functionality diagram and source code is attached.





```
//team 5
//project 2
//Dr. Hougen
```

```
#define PRIMARY_MOTOR 0
#define PRIMARY_SPEED 120
#define SECONDARY_MOTOR 3
#define FRONT_LIGHT_PORT 2
#define REAR_LIGHT_PORT 5
#define LEFT_RANGE_PORT 16
#define RIGHT_RANGE_PORT 19
#define RIGHT BUMPER 7
#define LEFT BUMPER 8
#define OFFSET 700
#define INC 200
#define MIN_DIST 70
```

```
//flag for bumpers
int leftBumperHit, rightBumperHit;
```

```
int leftRangeHit, rightRangeHit;
int leftRangeValue, rightRangeValue;
int light_id, avoid_id;
int MIDDLE_SERVO_POS, RIGHT_SERVO_POS, LEFT_SERVO_POS;
int RIGHT_EXTREME, LEFT_EXTREME;
```

```
//sets the middles servo pos to strongest light source. then determines the far right
//and far left servo positions
```

```
void calibrate()
```

```
{
    int max_light = 100;
    int max_pos = 0;
    int i;
    int f_light;
```

```
servo3 = 3000;
sleep(0.5);
for (i = 3000; i >= 1000; i -= 100)
{
    servo3 = i;
```

```
    f_light = analog(FRONT_LIGHT_PORT);
    if (f_light < max_light)
```

```

    {
        max_light = f_light;
        max_pos = i;
    }
    sleep(0.2);
}

MIDDLE_SERVO_POS = max_pos;
RIGHT_SERVO_POS = MIDDLE_SERVO_POS + OFFSET;
LEFT_SERVO_POS = MIDDLE_SERVO_POS - OFFSET;
RIGHT_EXTREME = RIGHT_SERVO_POS + 300;
LEFT_EXTREME = LEFT_SERVO_POS - 300;
printf("mid: %d", MIDDLE_SERVO_POS);

servo3 = MIDDLE_SERVO_POS;
}

//backs up robot for specific time
void backup(float length)
{
    ao();
    sleep(0.5);
    motor(PRIMARY_MOTOR, -60);
    sleep(length);
    ao();
}

//stops the robot in place
void brake()
{
    ao();
    motor(PRIMARY_MOTOR, -30);
    sleep(0.2);
    ao();
}

//turns the robot right with a certain power and length of time
void turnRight(float duration, int power)
{
    printf("turning right\n");
    off(PRIMARY_MOTOR);
    motor(SECONDARY_MOTOR, -1 * power);
    sleep(duration);
    ao();
}

```

```

//turns the left right with a certain power and length of time
void turnLeft(float duration, int power)
{
    printf("turning left\n");
    off(PRIMARY_MOTOR);
    motor(SECONDARY_MOTOR, power);
    sleep(duration);
    ao();
}

//if left bumper is hit, turn right
void leftBumperAvoid()
{
    brake();
    backup(0.9);
    turnRight(1.0, 60);
    motor(PRIMARY_MOTOR, PRIMARY_SPEED); //resume cruise
}

//if right bumper is hit, turn left
void rightBumperAvoid()
{
    brake();
    backup(0.9);
    turnLeft(1.0, 60);
    motor(PRIMARY_MOTOR, PRIMARY_SPEED); //resume cruise
}

//if left range finder sees something, turn until the object is no longer in front
void leftRangeAvoid()
{
    brake();

    while(leftRangeValue >= MIN_DIST)
    {
        turnRight(0.2, 60);
        leftRangeValue = analog(LEFT_RANGE_PORT);
    }
    turnRight(0.3, 60); //turn just a little more to be safe

    motor(PRIMARY_MOTOR, PRIMARY_SPEED); //resume cruise
}

//if left range finder sees something, turn until the object is no longer in front
void rightRangeAvoid()
{

```

```

brake();

while(rightRangeValue > MIN_DIST)
{
    turnLeft(0.2, 60);
    rightRangeValue = analog(RIGHT_RANGE_PORT);
}
turnLeft(0.3, 60); //turn just a little more to be safe

motor(PRIMARY_MOTOR, PRIMARY_SPEED); //resume cruise
}

//cruise and avoid behaviors are implemented by this process.
//check bump sensors and range finders and make appropriate decision
void avoid_process()
{
    while(1)
    {
        leftRangeValue = analog(LEFT_RANGE_PORT);
        rightRangeValue = analog(RIGHT_RANGE_PORT);

        printf("r_range=%d l_range=%d\n", rightRangeValue, leftRangeValue);

        if (digital(LEFT_BUMPER))
            leftBumperAvoid();
        if (digital(RIGHT_BUMPER))
            rightBumperAvoid();

        if (leftRangeValue >= MIN_DIST)
            leftRangeAvoid();
        if (rightRangeValue >= MIN_DIST)
            rightRangeAvoid();

        sleep(0.05);
    }
}

//turns the robot right, towards the strongest light source
void turnRightToLight(int goalLightVal, int tolerance, int maxCount)
{
    int currentLightVal, count;

    currentLightVal = analog(FRONT_LIGHT_PORT);
    ao();
}

```

```

count = 0;

//turn until maxcount is reached or light is found within certain tolerance
while(currentLightVal > (goalLightVal + tolerance))
{
    printf("light %d tol = %d", currentLightVal, tolerance);
    motor(SECONDARY_MOTOR, -60);
    currentLightVal = analog(FRONT_LIGHT_PORT);
    sleep(0.25);
    count++;

    //if best light not found yet, increase the tolerance and turn time, turn in opposite
direction
    if (count == maxCount)
    {
        count = 0;
        tolerance = tolerance + 2;
        ao();
        sleep(0.25);
        turnLeftToLight(goalLightVal, tolerance, count * 2);
        break;
    }
}
ao();
}

//turns the robot right, towards the strongest light source
void turnLeftToLight(int goalLightVal, int tolerance, int maxCount)
{
    int currentLightVal, count;

    currentLightVal = analog(FRONT_LIGHT_PORT);
    ao();
    count = 0;

    //turn until maxcount is reached or light is found within certain tolerance
    while (currentLightVal > (goalLightVal + tolerance))
    {
        printf("light %d tol %d", currentLightVal, tolerance);
        motor(SECONDARY_MOTOR, 60);
        currentLightVal = analog(FRONT_LIGHT_PORT);
        sleep(0.25);
        count++;

        //if best light not found yet, increase the tolerance and turn time, turn in opposite
direction

```

```

    if (count == maxCount)
    {
        count = 0;
        tolerance = tolerance + 2;
        ao();
        sleep(0.25);
        turnRightToLight(goalLightVal, tolerance, count * 2);
        break;
    }
}
ao();
}

```

//this function is used by the find\_light\_caller process. it finds the strongest light source  
//and turns in that direction. it also measures the ambient light in both directions

```

float f_ambience, r_ambience;
int f_lightReading, r_lightReading;

```

```

void find_light()

```

```

{
    int i, maxLightReading, servoPos;

```

```

    //beep();

```

```

    brake();

```

```

    maxLightReading = 500; //max light reading is a low number, so set to high right now

```

```

    servo3 = RIGHT_EXTREME;

```

```

    sleep(0.4);

```

```

    r_ambience = f_ambience = 0.0;

```

```

    for (i = RIGHT_EXTREME; i >= LEFT_EXTREME; i -= INC)

```

```

    {

```

```

        servo3 = i;

```

```

        f_lightReading = analog(FRONT_LIGHT_PORT);

```

```

        r_lightReading = analog(REAR_LIGHT_PORT);

```

```

        f_ambience += (float)f_lightReading;

```

```

        r_ambience += (float)r_lightReading;

```

```

        printf("f=%d, r=%d, ml=%d, p=%d\n", f_lightReading, r_lightReading,
maxLightReading, servoPos);

```

```

    if ((f_lightReading < maxLightReading) && (f_lightReading < r_lightReading))
    {
        maxLightReading = f_lightReading;
        servoPos = i;
    }
    else if ((r_lightReading < maxLightReading) && (r_lightReading <
f_lightReading))
    {
        maxLightReading = r_lightReading;
        servoPos = i * -1; //negative means that the light is behind it
    }
    sleep(0.2);
}
servo3 = MIDDLE_SERVO_POS;
sleep(0.5);

//ambience determined by the sum divided by the number of servo clicks
f_ambience = (f_ambience / (((float)RIGHT_EXTREME - (float)LEFT_EXTREME) /
100.0));
r_ambience = (r_ambience / (((float)RIGHT_EXTREME - (float)LEFT_EXTREME) /
100.0));

printf("f= %f , r= %f", f_ambience, r_ambience);
sleep(1.5);

//light is on the front right side, turn right
if (servoPos >= MIDDLE_SERVO_POS)
{
    printf("turning right %d\n", servoPos);
    //sleep(1.0);
    turnRightToLight(maxLightReading, (int)(f_ambience / 15.0), 20);
    beep();
    ao();
    sleep(0.6);
}

//light is on the rear right side, turn right
else if ((servoPos) <= -LEFT_EXTREME && (servoPos >= -
MIDDLE_SERVO_POS))
{
    printf("turning right %d\n", servoPos);
    //sleep(1.0);
    turnRightToLight(maxLightReading, (int)(r_ambience / 15.0), 20);
    beep();
    ao();
}

```

```

        sleep(0.6);
    }

    //light is on the front left side, turn left
    else if ((servoPos <= MIDDLE_SERVO_POS) && (servoPos >=
LEFT_EXTREME))
    {
        printf("turning left %d\n", servoPos);
        //sleep(1.0);
        turnLeftToLight(maxLightReading, (int)(f_ambience / 15.0), 20);
        beep();
        ao();
        sleep(0.6);
    }
    //light is on the rear left side, turn left
    else
    {
        printf("turning left %d\n", servoPos);
        //sleep(1.0);
        turnLeftToLight(maxLightReading, (int)(r_ambience / 15.0), 20);
        beep();
        ao();
        sleep(0.6);
    }
}

//seek light behavior, subsumes over cruise and avoid. uses the find_light function
//to find the greatest light source and turn towards that source
void find_light_caller_process()
{
    while(1)
    {
        ao();
        beep();
        kill_process(avoid_id);
        find_light();
        motor(PRIMARY_MOTOR, PRIMARY_SPEED);
        avoid_id = start_process(avoid_process());
        sleep(10.0);
    }
}

void main()
{
    while (!start_button())
    {

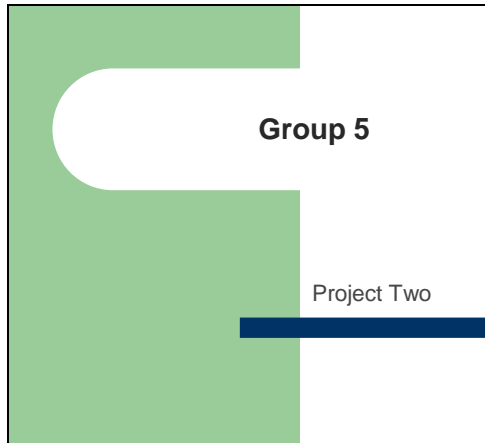
```



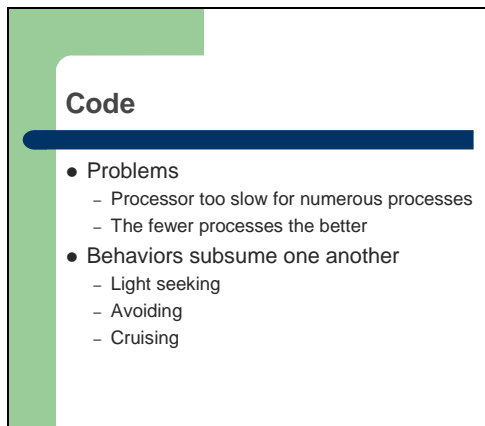
```
}  
init_expbd_servos(1);  
  
calibrate();  
  
while (!start_button())  
{  
}  
  
//start both processes  
avoid_id = start_process(avoid_process());  
light_id = start_process(find_light_caller_process());  
  
}
```

## Section 3: Presentation

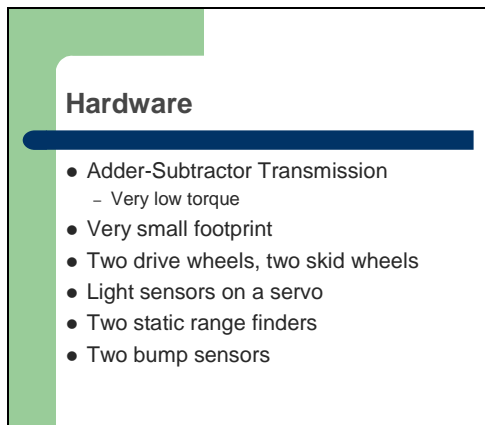
Slide 1



Slide 2



Slide 3



Slide 4

### Transmission

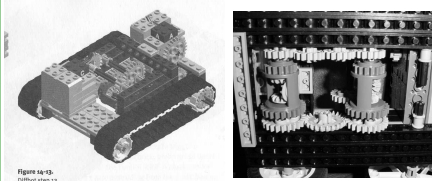


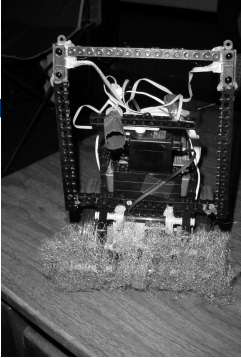
Figure 14-19  
Dribble step 13

The image shows a LEGO Technic transmission assembly. On the left is a perspective view of a completed assembly with a black rubber band. On the right is a close-up of the internal gear mechanism, showing a central gear connected to several other gears on a common shaft.

Slide 5

### Sensors

- Could not get range finders to work correctly on a servo, so a rig was made to hold them
- Went through 3 different bump-sensor designs....they kept falling off upon contact
- Cones around light sensors to make them unidirectional




The image shows a custom-built sensor rig mounted on a robot. It features a metal frame with a servo motor and a range finder sensor. The rig is positioned to detect obstacles in front of the robot.

Slide 6

### What Didn't Work

- Bump sensors kept falling off
- Weight in back kept tipping robot
- Couldn't drive over certain obstacles



PLAY  
0:07:30  
SP

The image shows a person standing behind a table with a robot on it. The robot is a small, dark-colored robot with a camera or sensor on top. The person is wearing a dark jacket and pants. The table is covered with various objects, including a bucket and some small pieces. A digital display in the background shows "PLAY 0:07:30 SP".

## Slide 7

### What did work

- Excellent light seeking
- Only once hit a bucket
- Finally drove over cord
- Hit two lights, a third after tin

