**COMP 4500, Mobile Robotics I**

**Spring 2018, Prof. Yanco**

**Daniel MacMillan**

**Dangnhi Ngo**

**Lab 3: Braitenberg Vehicles, Emergence, Meta-Sensing and Randomness**

**Lab Report**

**Part I: Individual Questions**

**Daniel’s Answers:**

1) Our lab 2 robot was a Braitenberg Vehicle in the sense that it had two sensors that controlled two motors that would steer the robot in reaction to some stimulus – in this case the color of the ground beneath the sensor.

2) Braitenberg's method can be used to describe complex robot behaviors because even with just two motors and two sensors, behavior can be changed according to how one programs the robot to react to the sensor readings. One can program the robot to slow down after

Also, different sensors can be added on to the robot, as well as different types of outside stimulus that can trigger different types of sensors. There is also a way to

**Dangnhi’s Answers:**

1/ Our robot from lab 2 – line-following robot is a Braitenberg’s Vehicle which was equipped with two top-hat sensors and two motors. The connection is not too complex. When two IR sensors on the two sides of the robot detect the color of the white background, it means that the robot is on the black line and the sensors will control the motor to make the robot go straight and follow the black line. When one of two sensors detect the value of the black color, the motor will be controlled to turn the robot into that direction. For example, the left sensor which detects the black line would steer the robot to the left. This is the operation mechanism of a Braitenberg’s Vehicle.

2/ In my perspective, the Braitenberg’s method can explain the complex behaviors of the robot. There are four different types of robot behaviors described in Coward, Aggressive, Love and Explore. With the “Coward” Behavior, the robot is equipped with each sensor to the motor on the same side and the influence is positive. If the source of light is in the front, the robot may hit the source unless it is redirected from its source. In addition, if the source of light is to one side, for example, in the left, the sensor in the left would be more excited than the one in the right, which controls the motor in the left to run faster. It results in the running away the light of the robot. However, with the “Aggressive” Behavior, each sensor to the motor on the opposite side, and it will make the robot run toward the source of light (e.g: The sensor in the left controls the motor in the right to work harder, which makes the robot turn to the left in the direction where the light is). With the “Love” Behavior, the robot is equipped with each sensor to the motor on the same side as “Coward” Behavior; nonetheless, the influence is switched from positive to negative. With this mechanism, the robot comes to rest facing the source. In the opposite, the robot will run away the source light with crossed connections in “Explore” Behavior.

By selecting different sensors, various connections in straight or crossed, the sign of influence, the complicated robot behaviors can be achieved in a simple control mechanism of Braitenberg’s Vehicles.

**Part II: Lab**

***A. Light sensors***

1) Light sensor readings fall below 2000 when a flashlight is shown nearby from a standing value of 3000 with just the lights from the room.

2) The light sensor readings vary by a factor of 200 when shone under the same light. They both prefer the light source to be straight ahead of them, and not above or to the side of them.

3) The minimum readings we can obtain is under 200, the maximum reading we can obtain is just under 4000, the typical readings in a lit room is around 3500.

4) One of our light sensors is slightly stronger than the other, its readings vary from the other by a factor of 200.

/\* Daniel MacMillan and Dangnhi Ngo

\*

\* Lab 3 - Part A

\* Print out the sensor readings from the analog ports.

\*/

#include <kipr/botball.h>

int main()

{

while(!c\_button\_clicked())

{

printf("%d %d\n", analog(0), analog(1));

}

return 0;

}

***B. Shielding light sensors***

For both sensors, the readings when the light is pointed directly down the tubes is under 200. When the sensor turns away at 10 degrees, the value goes up slightly up to 1000. When the sensor turns away from the light at 20 degrees, the sensor readings go back up to 3000, and completely lose any indication that the light is there at all beyond that angle.

***C. Normalizing light readings to motor commands***

/\* Daniel MacMillan and Dangnhi Ngo

\*

\* Lab 3 - Part C

\* normalize light readings to motor commands

\*/

#include <kipr/botball.h>

#define MAX\_VEL 25

/\* Normalize: take light reading from individual sensors and turn it into

\* the proper motor speed.

\*/

int normalize(int light)

{

int MAX\_LIGHT = 100;

int MIN\_LIGHT = 4000;

int output = ((light - MIN\_LIGHT) \* 100) / (MAX\_LIGHT - MIN\_LIGHT);

// correct the values if they somehow go overboard

if(output < 0)

return 0;

if(output > 100)

return 100;

return output;

}

***D. Light-seeking***

The light seeking algorithm we have written works to try and turn the robot, but next time we should change the hardware of the robot to make it physically turn the robot around.

**-------Light-seeing Braitenberg vehicle------**

/\* Daniel MacMillan and Dangnhi Ngo

\*

\* Lab 3 - Part D1

\* normalize light readings to motor commands and make it follow light

\*/

#include <kipr/botball.h>

#define MAX\_VEL 25

/\* Normalize: take light reading from individual sensors and turn it into

\* the proper motor speed.

\*/

int normalize(int light)

{

int MAX\_LIGHT = 100;

int MIN\_LIGHT = 4000;

int output = ((light - MIN\_LIGHT) \* 100) / (MAX\_LIGHT - MIN\_LIGHT);

// correct the values if they somehow go overboard

if(output < 0)

return 0;

if(output > 100)

return 100;

return output;

}

int main()

{

while(!c\_button\_clicked())

{

// print the new motor values

int normal\_left = normalize(analog(0));

int normal\_right = normalize(analog(1));

printf("Left: %d | Right: %d\n", normal\_left, normal\_right);

// move the motors based on light values

// if we want to avoid light, make motor values negative, or swap normal\_left/right

motor(0, normal\_right);

motor(1, normal\_left);

}

ao();

return 0;

}

**-------Avoid sources of light------**

The way our robot avoids light is by swerving to the right when the right sensor detects light, and swerves left when the left sensor detects light. All that we did to accomplish this was swapping normalized values going to the motors from the avoid program.

/\* Daniel MacMillan and Dangnhi Ngo

\*

\* Lab 3 - Part D2

\* normalize light readings to motor commands and make it avoid light

\*/

#include <kipr/botball.h>

#define MAX\_VEL 25

/\* Normalize: take light reading from individual sensors and turn it into

\* the proper motor speed.

\*/

int normalize(int light)

{

int MAX\_LIGHT = 100;

int MIN\_LIGHT = 4000;

int output = ((light - MIN\_LIGHT) \* 100) / (MAX\_LIGHT - MIN\_LIGHT);

// correct the values if they somehow go overboard

if(output < 0)

return 0;

if(output > 100)

return 100;

return output;

}

int main()

{

while(!c\_button\_clicked())

{

// print the new motor values

int normal\_left = normalize(analog(0));

int normal\_right = normalize(analog(1));

printf("Left: %d | Right: %d\n", normal\_left, normal\_right);

// move the motors based on light values

// if we want to avoid light, make motor values negative, or swap normal\_left/right

motor(0, normal\_left);

motor(1, normal\_right);

}

ao();

return 0;

}

***E. Light and touch sensitivity***

The bumper code does the trick when trying to avoid obstacles because without it, the robot could keep moving toward light when it bumps into the light. The light sensor has no way of detecting how close a light source actually is, so this is a welcome addition.

/\* Daniel MacMillan and Dangnhi Ngo

\*

\* Lab 3 - Part E

\* add bumper code from lab 1 so that the robot avoids obstacles

\*/

#include <kipr/botball.h>

#define MAX\_VEL 25

/\* Normalize: take light reading from individual sensors and turn it into

\* the proper motor speed.

\*/

int normalize(int light)

{

int MAX\_LIGHT = 100;

int MIN\_LIGHT = 4000;

int output = ((light - MIN\_LIGHT) \* 100) / (MAX\_LIGHT - MIN\_LIGHT);

// correct the values if they somehow go overboard

if(output < 0)

return 0;

if(output > 100)

return 100;

return output;

}

/\* turn\_around: procedure from lab 2 to be called when the bumper is hit.

\*/

void turn\_around()

{

// first go backward

motor(0,-MAX\_VEL\*2);

motor(1,-MAX\_VEL\*2);

msleep(700);

// then swivel around to go a different direction

motor(0,-MAX\_VEL\*2);

motor(1, MAX\_VEL\*2);

msleep(2000);

return;

}

int main()

{

while(!c\_button\_clicked())

{

// print the new motor values

int normal\_left = normalize(analog(0));

int normal\_right = normalize(analog(1));

printf("Left: %d | Right: %d\n", normal\_left, normal\_right);

// move the motors based on light values

// if we want to avoid light, make motor values negative

motor(0, normal\_right);

motor(1, normal\_left);

// Hit detection

if(digital(2) || digital(3))

{

turn\_around();

}

msleep(20);

}

ao();

return 0;

}