Embodied Cognition Workshop: Braitenberg Vehicles

João Frazão, Danbee Kim, Gonçalo Lopes

October 30, 2013

Abstract

The goals of this workshop are: (1) To give people a chance to play with robots, and (2) implement the first four Braitenberg Vehicles.

1 Braitenberg Vehicles

Vehicle 1: Alive

Vehicle 1, the simplest vehicle. The speed of the motor (rectangular box at the tail end) is controlled by a sensor (half circle on a stalk, at the front end). Motion is always forward, in the direction of the arrow, except for perturbations.

Components: Sensor and motor.

Principle: The more there is of the quality (e.g., heat) to which the sensor is tuned, the faster the motor goes.

Description: alive, restless, doesnt "ulike" heat.

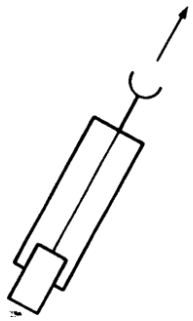

Figure 1: Vehicle 1

Vehicle 2a: Cowardly

Components: 2 sensors, 2 motors, each sensor connected to the motor on the same side ("uncrossed").

Principle: The more there is of the quality to which the sensor is tuned, the faster the motors go ("excitatory").

Description: dislikes source to which the sensor is tuned; occasionally "attacks" it.

Vehicle 2b: Aggressive

Components: 2 sensors, 2 motors, each sensor connected to the motor on the opposite side ("crossed")

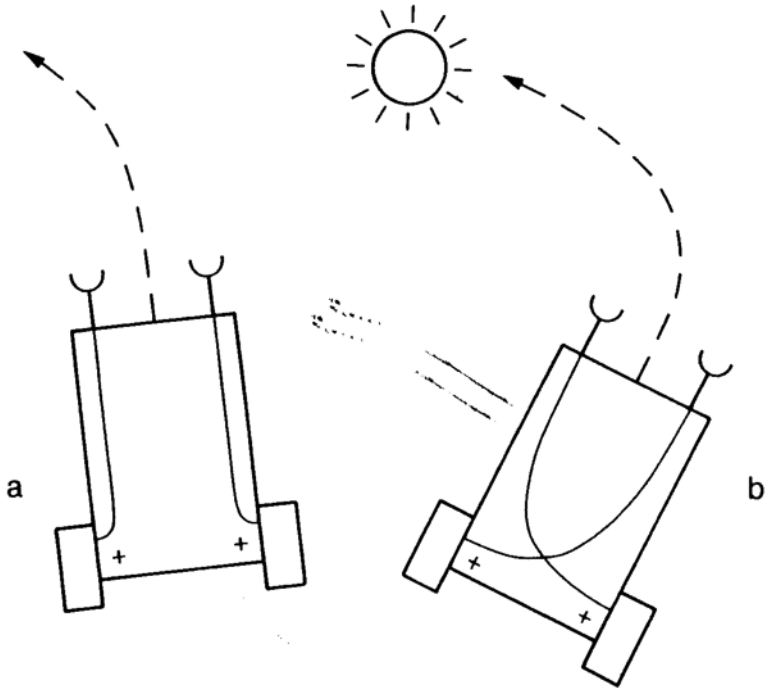

Figure 2: Vehicles 2a and 2b are in the vicinity of a source (circle with rays emanating from it). Vehicle 2b orients toward the source, 2a away from it.

Principle: The more there is of the quality to which the sensor is tuned, the faster the motors go ("excitatory").

Description: dislikes source to which the sensor is tuned; "attacks" it.

Vehicle 3a: Loving/Quietly Adoring

Components: 2 sensors, 2 motors, each sensor connected to the motor on the same side ("uncrossed")

Principle: The more there is of the quality to which the sensor is tuned, the slower the motors go ("inhibitory").

Description: loves the source, wants to be near it, comes to rest facing it.

Vehicle 3b: Loving/Exploring

Components: 2 sensors, 2 motors, each sensor connected to the motor on the opposite side ("crossed").

Principle: The more there is of the quality to which the sensor is tuned, the slower the motors go ("inhibitory").

Description: likes the source, but easily attracted away.

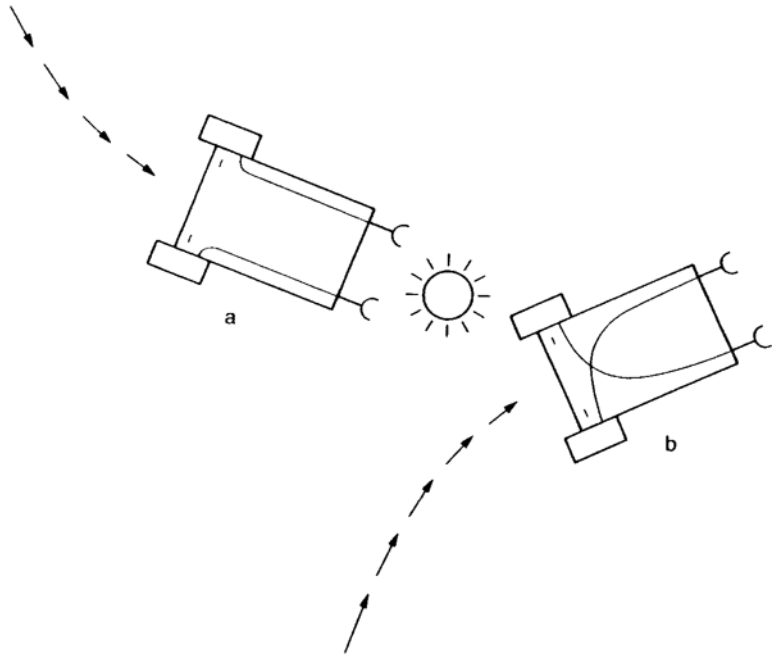

Figure 3: Vehicle 3, with inhibitory influence of the sensors on the motors.

Vehicle 3c: Knowing, Valuing

Components: 4 sensors, 4 motors, each tuned to different properties of the environment.

Principle: one each of the four types so far:

uncrossed/excitatory: tuned to temperature crossed/excitatory: tuned to light uncrossed/inhibitory: tuned to organic material crossed/inhibitory: tuned to oxygen level Description: Cowardly toward areas of high temperature;

Aggressive toward light sources;

Loves organic material; leaves and seeks new source if environment is depleted;

Restlessly seeks best source of oxygen.

Vehicle 3c appears to know a great deal.

Vehicle 4a: Displaying Instincts, Specialization

Components: sensors and motors

Principle: connections both excitatory and inhibitory but non-monotonic

Description: does everything 3c vehicles do, but with much less predictability.

Vehicle 4b: Making Decisions

Components: sensors and motors, and threshold devices

Principle: connections both excitatory and inhibitory but non-monotonic

Description: does everything 3c vehicles do, but with much less predictability, and appears to ponder over its "decisions"; appears to will.

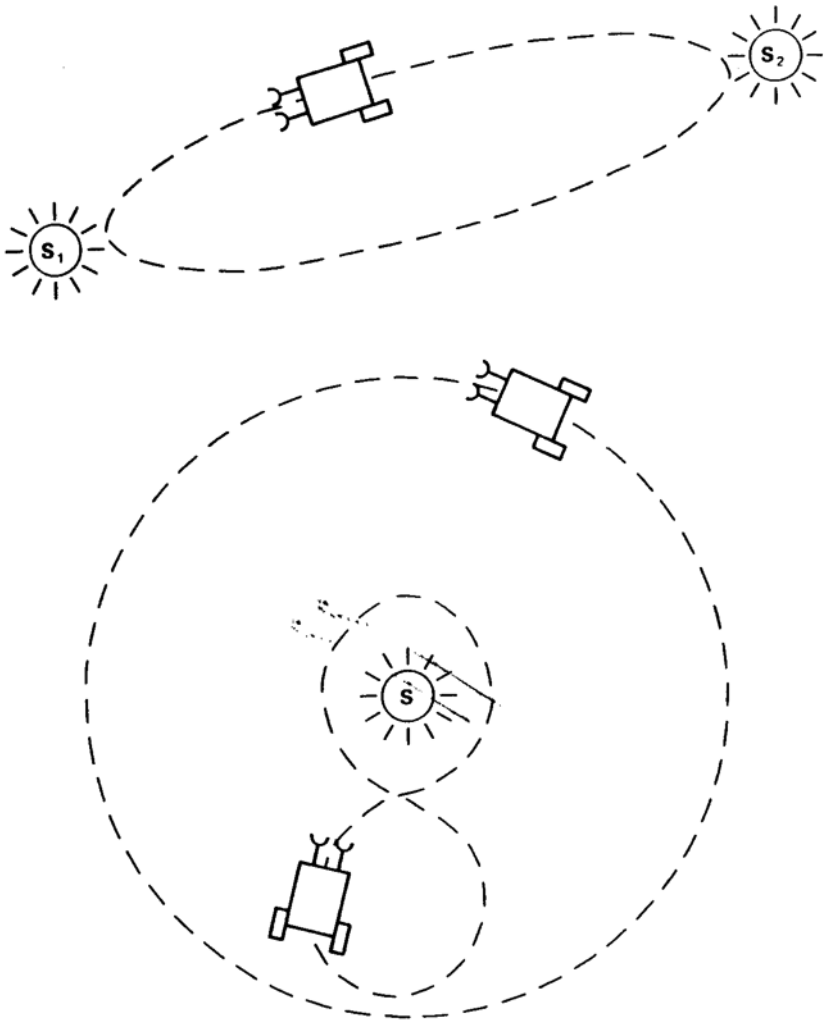

Figure 4: Trajectories of vehicles of brand 4a around or between sources

2 Robot Control Code

You will find the following code useful:

BraitenbergVehicle.c

```
// ---------------------- HARDWARE MAPPING CONSTANTS ----------------------//

// ------------ Excitatory/Inhibitory Pins ---------------- //
int LeftMotorExcitatoryPin = 1;
int RightMotorExcitatoryPin = 2;
int LeftMotorInhibitoryPin = 3;
int RightMotorInhibitoryPin = 4;

// ------------ Motor  Pins ---------------- //
int MotorA0Pin = 6;
int MotorA1Pin = 11;
int MotorB0Pin = 3;
int MotorB1Pin = 5;

// ------------ Bumper  Pins ---------------- //
int LeftBumperPin = 8;
int RightBumperPin = 7;

// ---------------------- CALIBRATION CONSTANTS ---------------------- //
// -- VARY THE MAXIMUM VALUES TO GET THE ROBOT TO GO STRAIGHT -- //
int LeftMaxSpeed = 255;
int RightMaxSpeed = 255;
int LeftMinSpeed = 0;
int RightMinSpeed = 0;

// ------------------------ SENSOR VARIABLES ------------------------ //

// ------------ Excitation/Inhibition ---------------- //
int leftMotorInhibition;
int rightMotorInhibition;
int leftMotorExcitation;
int rightMotorExcitation;

// ---------------- Bumper State ---------------- //
int leftBumperState;
```

```
int rightBumperState;



// ------------------------- MOTOR VARIABLES ------------------------- //

// ---------------- Motor Drive ---------------- //
int leftMotorDrive;
int rightMotorDrive;



// ------------------------- INITIALIZATION ------------------------- //
void setup()
{
  Serial.begin(9600);

  //declare motor pins as outputs
  pinMode(MotorA0Pin, OUTPUT);
  pinMode(MotorA1Pin, OUTPUT);
  pinMode(MotorB0Pin, OUTPUT);
  pinMode(MotorB1Pin, OUTPUT);

  // declare frontal bumpers as inputs
  pinMode (LeftBumperPin, INPUT);
  pinMode (RightBumperPin, INPUT);

  // activate internal resistance of bumper pins
  digitalWrite(LeftBumperPin, HIGH);
  digitalWrite(RightBumperPin, HIGH);

  delay(1000);
}

// ------------------------- CONTROL LOOP ------------------------- //
void loop()
{
  // Sample excitation/inhibition levels
```

```
  leftMotorExcitation = analogRead (LeftMotorExcitatoryPin);
  rightMotorExcitation = analogRead (RightMotorExcitatoryPin);
  leftMotorInhibition = analogRead (LeftMotorInhibitoryPin);
  rightMotorInhibition = analogRead (RightMotorInhibitoryPin);

  // Sample bumper state
  leftBumperState = digitalRead(LeftBumperPin);
  rightBumperState = digitalRead(RightBumperPin);

  // Map excitation levels to motor drive
  leftMotorDrive = map (leftMotorExcitation, 0, 1023, LeftMinSpeed, LeftMaxSpeed);
  rightMotorDrive = map (rightMotorExcitation, 0, 1023, RightMinSpeed, RightMaxSpeed);

  // Suppress motor drive according to current inhibition levels
  leftMotorDrive -= map (leftMotorInhibition, 0, 1023, LeftMinSpeed, LeftMaxSpeed);
  rightMotorDrive -= map (rightMotorInhibition, 0, 1023, RightMinSpeed, RightMaxSpeed);

  DriveMotors(leftMotorDrive, rightMotorDrive);
  delay(10);
}
```

DriveMotors.c

```
// updates the drive on the left and right motors
void DriveMotors(int leftDrive, int rightDrive)
{
 if (leftDrive > 0)
 {
  analogWrite(MotorA0Pin, leftDrive);
  digitalWrite(MotorA1Pin, LOW);
 }

 if (leftDrive < 0)
 {
  digitalWrite(MotorA0Pin, LOW);
  analogWrite(MotorA1Pin, map(leftDrive, 0, -255, 0, 255));
```

```
 }

 if (leftDrive == 0)
 {
  digitalWrite(MotorA0Pin, LOW);
  digitalWrite(MotorA1Pin, LOW);
 }

 if (rightDrive > 0 )
 {
  analogWrite(MotorB0Pin, rightDrive);
  digitalWrite(MotorB1Pin, LOW);
 }

 if (rightDrive < 0 )
 {
  digitalWrite(MotorB0Pin, LOW);
  analogWrite(MotorB1Pin, map(rightDrive, 0, -255, 0, 255));
 }

 if (rightDrive == 0)
 {
  digitalWrite(MotorB0Pin, LOW);
  digitalWrite(MotorB1Pin, LOW);
 }
}
```

EscapeReflex.c

```
// activates the escape reflex, given a bumper pin number
void EscapeReflex(int bumperPin)
{
  // drive back for some amount of time
  DriveMotors(-LeftMaxSpeed, -RightMaxSpeed);
  delay(500);
```

```
  // if the left bumper was activated, turn to the right
  if (bumperPin == LeftBumperPin)
  {
    DriveMotors(-LeftMaxSpeed, RightMaxSpeed);
  }

  // if the right bumper was activated, turn to the left
  if (bumperPin == RightBumperPin)
  {
    DriveMotors(LeftMaxSpeed, -RightMaxSpeed);
  }

  // wait for turn to finish
  delay(250);
}
```