ME 586: Biology-Inspired Robotics

University of Washington, Fall 2018, Prof. Sawyer B. Fuller

Problem Set 1

The goals of this problem set are to

- Demonstrate and promote good software development practices for simulation code. In particular, you will learn how to write and simulate a dynamical system that evolves according to $\dot{\boldsymbol{q}} = \boldsymbol{f}(\boldsymbol{q}, \boldsymbol{u})$, where $\boldsymbol{q}$, $\boldsymbol{f}$, and $\boldsymbol{u}$ may be vector-valued variables.
- Explore dynamics and control simulations in the context of a system that is difficult to control using classic approaches

The skeleton code provided here is written in MATLAB (see below for how to download and a basic tutorial). You may also write software in another programming language, e.g. Python, but that will require re-implementing the rest of the code that is provided with this problem set. You are encouraged to work together to work out solutions, but you must submit your own work.

Please read Braitenberg1984, chapters 1–4, available under the `papers` section of the course website. You will be simulating the behavior of some of the vehicles he describes. For our case, imagine your vehicle is moving on flat ground, such as a tabletop (Figure 1).

1. You will first simulate vehicle 1, ALIVE, which follows a straight line. Let's suppose its sensor is responding in proportion to brightness of the light as detected by the sensor (rather than heat as suggested by the book). To implement this in simulation, we introduce the notion of a state vector, which comprises are all of the parameters of the robot that evolve in time. For this robot, you will use a state consisting of its position and orientation, $\boldsymbol{q} = [x, y, \theta]^T$. Vehicle 1's wheels are connected in such a way that they spin with a speed that is proportional to the sensor reading. If the robot's speed is given by v, then we can write that

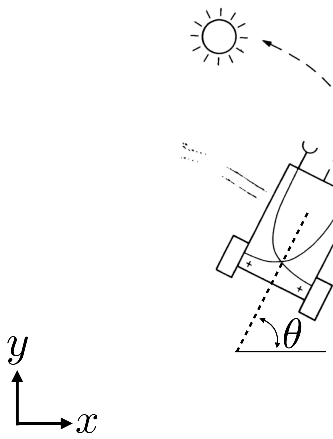

Figure 1: Braitenberg vehicle.

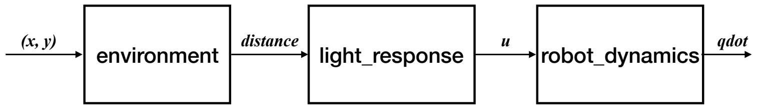

Figure 2: Flow of information. Note that the robot only receives readings from its sensors and does not know, for example, its position.

the state evolves according to

$$\dot{\boldsymbol{q}} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ 0 \end{bmatrix}. \tag{1}$$

This is a specific example of the general dynamical system formulation, $\dot{\boldsymbol{q}} = \boldsymbol{f}(\boldsymbol{q}, \boldsymbol{u})$, where $\boldsymbol{u}$ is some input to the system. Here, the input is the wheel velocity v. Note that this vehicle is rolling on the ground so it moves in the direction it is facing, and will not intercept the light source unless it is facing directly at it.

To simulate such a system on a digital computer, it is necessary to approximate its behavior by calculating how it changes over small increments of time ΔT. The simplest approximation and assume that $\boldsymbol{f}$ and $\boldsymbol{u}$ are constant during the interval. With this approximation, the state $\boldsymbol{q}$ changes by $\Delta T\dot{\boldsymbol{q}}$ after each time increment. How $\boldsymbol{q}$ evolves with time can be written as an iterative computation in which, for each time index k (where k is an integer), the state is updated by

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k + \Delta T\dot{\boldsymbol{q}}_k,$$

where $\boldsymbol{q}_0$, the initial condition, must be specified.

Skeleton code to simulate a robot performing these dynamics is provided online in `braitenberg_1.m` (note that this software uses language features only available after release 2016b. If you have an older version, you are encouraged to upgrade or use the remote server). In addition to performing the iteration simulation, the software includes a number of functions that compute various aspects of the simulation (Figure 2).

The function `robot_dynamics` in that file computes $\dot{\boldsymbol{q}}$ (Equation 1). Please update it to include these dynamics. Next, update the function `simulate_environment` so that it correctly computes the distance from the light source to the robot (you may assume the robot's sensor is exactly at its $(x,\, y)$ position for this problem). Here, you will simulate a robot whose velocity acts in inverse proportion to the distance to the light source. Submit your updated software code and an image of the resulting motion of the vehicle.

2. Next, you will implement vehicles 2a and 2b, COWARD and AGGRESSIVE. For these robots, you must compute distances to the light source for two sensors placed on either side of the robot. Note that your code must calculate where the light sensor is by computing where the vehicle is and how it is oriented (by using its state). You have some flexibility to decide where on the vehicle the light sensors are (front/middle/back) - but they should be on both sides somewhere. Skeleton code is in `braitenberg_2.m`; you must incorporate your solution to the functions `robot_dynamics`, `simulate_environment`, and `light_response`. Note that your new dynamics must also incorporate the ability of the robot to rotate ($\dot{\theta} \neq 0$) because of a difference in velocity of your two wheels. Knowing the robot's width and wheel speed, you can compute its rotation rate.

As you write your code, it is sometimes useful to test your functions with known inputs to perform a sanity check. One way to do that is to put the function into a separate `.m` file and run it from the MATLAB command prompt with inputs you specify.

Submit your software code and plots showing light following and light avoiding maneuvers (2a and 2b).

3. Next, you will implement vehicle 4. For these robots, you will need to implement an additional feature in your vehicle 2 code in the `light_response` function. Your new function will produce a non-monotonic wheel velocity response to an input light reading intensity I, where I is inversely proportional to the distance to the light source. One possible function is a Gaussian function of I that varies according to

$$u = e^{-\frac{(I-\mu)^2}{\sigma^2}},$$

where μ and σ are the center and width of the Gaussian function, respectively. By adjusting μ and σ, see if you can get this robot to perform some sort of interesting motion around the light source. Note: you may discover that his robot's behavior can be very tricky to reason about! You are also invited to explore other functions, but this is not required. Submit your code and a representative figure.

Notes on MATLAB

MATLAB is a programming language designed to make numerical computation as simple as possible. It is available through the Mechanical Engineering Department's remote desktop server (more information at `https://www.me.washington.edu/computing.html`) or for purchase for a fee at U. Washington's U-Ware site. The following tutorial will get you up to speed or serve as a refresher: `http://faculty.washington.edu/minster/bio_inspired_robotics/files/matlab_tutorial.pdf`

This problem set uses two additional aspects of MATLAB that are not covered in the tutorial:

Structures These are convenient for combining number of different values (such as the parameters of our vehicle) into a single variable. They are different from arrays because each field has a name. To create a structure, simply assign to one of its fields:

```
mystruct.field1 = 22;
mystruct.asdf = [1, 2, 3];
```

You can use it in an expression like any other variable:

```
>> mystruct.field1 * 3

ans =
66
```

Functions These allow you to encapsulate functionality, making them good programming practice. They are defined as follows:

```
function return_val = myfunction(argument1, argument2)
   blah = 22;

   % return_val must be set somewhere inside the function:
   return_val = blah * argument1 * argument2;
end
```

There can be any number of arguments in your definition.

Calling a function you write is like calling any function:

```
>> myval = myfunction(1, 5)

ans =
110
```

You can get more help on these and other MATLAB features by using the "help" command:

```
>> help function
```