# ME 586: Biology-Inspired Robotics

University of Washington, Winter 2020, Prof. Sawyer B. Fuller

Problem Set 1

Goals: familiarize you with the basic elements of python and good software design, install necessary software, learn how to simulate a dynamic system of the form $\dot{\boldsymbol{q}} = \boldsymbol{f}(\boldsymbol{q}, \boldsymbol{u})$, and to linearize it for purposes of control. Your submission will be in the form of python Jupyter notebook(s).

Reading:

- Feedback Systems 2nd Ed. section 6.4 of Chapter 6: Linearization.
- Braitenberg1984, chapters 1–4, available under the `papers` section of the course website.
- The site https://docs.scipy.org/doc/numpy/user/numpy-for-matlab-users.html#table-of-rough-matlab-numpy-equivalents comparing MATLAB to numpy may be useful.
- This on python tutorial may be useful https://www.programiz.com/python-programming/tutorial.

**0. Python and Jupyter notebook installation for Windows or Mac (∼600 MB download).**

1. Install miniconda for python 3.7 by downloading and running the installer here `https://docs.conda.io/en/latest/miniconda.html` (64-bit version recommended; 54 MB download)
2. Install the Python Jupyter notebook by running the following command in a terminal (on Windows, run the program "Anaconda prompt." On a Macintosh, open the Terminal app.)
   ```
   conda install jupyter
   ```
   (150 MB download)
3. Install numerical and control systems packages by running the command
   ```
   conda install -c conda-forge control
   ```
   and
   ```
   conda install -c conda-forge slycot
   ```
   (350 MB download)
4. Run the command `jupyter notebook`. A tab will open up in your browser. Navigate the file listing to where you want to create a new notebook and click "new"... "Python 3 notebook" near the top right of the page.

**I. Python and simulation.**

1. Lists and Numpy arrays vs. MATLAB.

   When Python was first created in the 1990's, it came with an array type called a "list." Lists can hold any type of variable, e.g. `[5, 32.3, 'asdf']` is an acceptable list. Later, a numerical array system called Numpy was added to Python. Numpy arrays can only hold one type of variable, usually numbers, but unlike lists, they allow for math operations to be performed on the array as a whole. Numpy arrays are comparable to MATLAB matrices; lists are like MATLAB cell arrays.

   To compare lists and Numpy arrays, write two functions using the Python `def` command:

   - `add_two_list(input_list)` that adds 2 to every element in a list of numbers, and

- `add_two_array(input_array)` that does the same for a Numpy array.

Example: `add_two_list([1, 3, 5])` should return `[3, 5, 7]`. (Hint: the Numpy version should be much shorter).

Another difference with MATLAB is that it is possible for Numpy arrays to be one-dimensional. In MATLAB, the transpose of a one-dimensional row-vector is a column vector, but this is is not the case for 1D Numpy array. Provide at least one means to convert a one-dimensional Numpy array `A` into a two-dimensional column vector. (Note: `A.T`—taking the transpose of a 1D row array—does not work!)

2. Consider the following dynamical system:

$$\begin{aligned} \dot{q}_1 &= a q_1 - b q_1 q_2 \\ \dot{q}_2 &= b q_1 q_2 - c q_2 \end{aligned}$$

where $q_1, q_2 \geq 0$ and $a, b, c > 0$ are positive constants. Find all equilibrium points and assess their stability: are they stable, marginally stable, or unstable?

Next, choose arbitrary $a, b, c$ values and simulate the response of this system for a few different initial conditions. 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 $\Delta T$. The simplest approximation and assume that $\boldsymbol{f}$ is constant during the interval. With this approximation, the state $\boldsymbol{q}$ changes by $\Delta T \dot{\boldsymbol{q}}$ after each time increment. Therefore the way $\boldsymbol{q}$ evolves with time can be written as an iterative computation in which, at each time instant $t = k\Delta T$ (where $k$ is an integer), the state is updated by

$$\boldsymbol{q}_{t+\Delta T} = \boldsymbol{q}_t + \Delta T \dot{\boldsymbol{q}}_t = \boldsymbol{q}_t + \Delta T \boldsymbol{f}(\boldsymbol{q})$$

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

Write a simulation of this system using an Jupyter notebook and use Matplotlib to plot its response in 2D to a few different initial conditions $\boldsymbol{q}_0$. In the lecture slides is an example of a simulation you may want to use as a starting point, including an example for how to write your function $\boldsymbol{f}$.

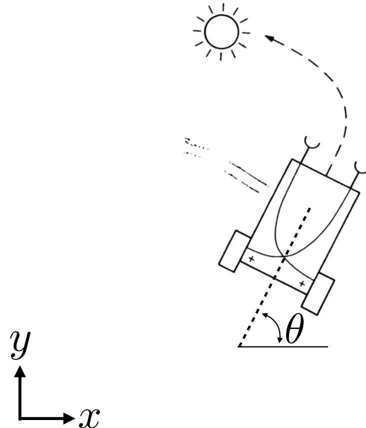


Figure 1: Braitenberg vehicle.

**II. Simulating a Braitenburg vehicle with input.** Figure 1 above shows a "Braitenberg vehicle". To simulate this robot, you will use a state consisting of position and orientation, $\boldsymbol{q} = [x,\, y,\, \theta]^T$. The vehicle'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 the state evolves according to

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

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

Here, the system input is $\boldsymbol{u} = [v, \dot{\theta}]^T$. Skeleton code to simulate a robot performing these dynamics is provided in the `software_examples` section of the course web page in `me586_braitenberg.ipynb.` In addition to performing the iteration simulation, the software includes a number of functions that compute various aspects of the simulation (Figure 2). In this exercise, you will implement vehicles 2a and 2b, COWARD and AGGRESSIVE, as well as vehicle 4.

1. The function `robot_dynamics` in that file computes $\dot{\boldsymbol{q}}$ (Equation 1). Please update it to include these dynamics (filling in missing parts between the `###`'s). Next, update the function `environment` so that it correctly computes the distance from the light source to the robot's sensors. 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 sensors are 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 (e.g. front or middle), but they must be on the sides somewhere to get the desired behavior. Lastly, you must implement `light_response`, which causes the speed of your robot's two wheel velocities to vary *in inverse proportion to the distance to the light source.* Note that your dynamics must 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.

2. Next, you will implement vehicle 4. For this robot, 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 $\mu$ and $\sigma$ are the center and width of the Gaussian function, respectively. By adjusting $\mu$ and $\sigma$, 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 a Jupyter notebook that includes plots showing the motion of vehicles 2a and 2b in response to the light source, as well the behavior of your vehicle 4 (you may add new cells that implement the new behavior as new functions so that it all fits on a single notebook, or submit multiple notebooks).