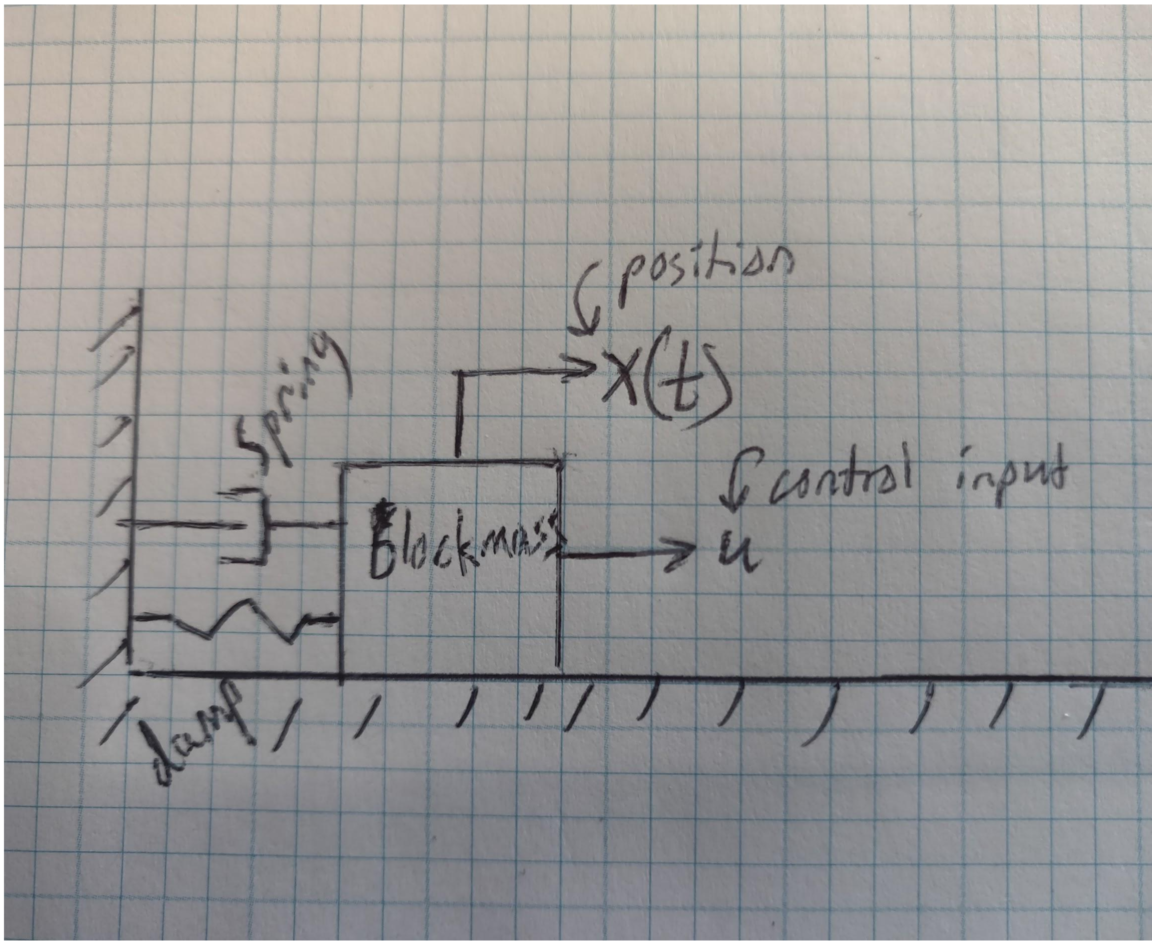
Documentation (tutorial) for Spring-Mass-Damper

- Overview
  - This is a script that takes the dynamics of a spring mass damper and graphs its position and velocity over time for a given initial condition. It also creates a controller for it using an assumed full state estimation, and graphs the mass' position and velocity as it's controlled to a desired value. The system itself is linear, I wanted to try and see if I could make a controller without having to deal with linearizing the system as a first step.
- Installation packages
  - Installing Python
    - https://realpython.com/installing-python/
  - Setting up virtual environment
    - https://docs.python.org/3/library/venv.html
  - Installing Numpy and Scipy
    - Once you've installed python and pip, run this command in the terminal to install numpy: (maybe use the 2nd one here?)
    - `pip install numpy`
- Math
  - System diagram:

- 

- Resources for the linear quadratic regulator. The files I used are listed in the Code section, these are pages that talk a little bit about what a linear quadratic regulator does.
  - https://en.wikipedia.org/wiki/Linear%E2%80%93quadratic_regulator
  - https://www.youtube.com/watch?v=1_UobILf3cc&list=PLMrJAkhIeNNR20Mz-VpzgfQs5zrYi085m&index=14&ab_channel=SteveBrunton
    - This channel was a cool resource, I followed their "Control Bootcamp" series to flesh this out in python.

- Code
  - https://github.com/DukeTresnor/forfun-integration
  - springMassDamp.py
    - You can control several different parameters within the springMassDamp.py file
    - blockmass, spring, and damp are all variables that change the physical properties of the system.
    - state describes your initial conditions for the spring mass damper with no controller (initial position, initial velocity).
    - mod_state describes the initial conditions of the system with its controller (initial position, initial velocity).

    - des_state describes your desired state (desired position, desired velocity).
    - Q and R are matrices used in the linear quadratic regulator, they influence how the cost function is weighed.
      - The elements of Q determine what factor of control is placed on each element of the system's state. In this spring-mass-damper system, Q is a diagonal 2x2 matrix, with its first value representing the controller's influence on the mass' position, and the last value representing the controller's influence on the mass' velocity.
      - R is a 1x1 matrix that represents how expensive your controller is in terms of energy expenditure. The larger the value, the larger the overall cost, and the harder it is to minimize that cost.
  - linearQuadraticRegulator.py
    - Code is from Mark Wilfried Mueller:
      - http://www.mwm.im/python-control-library-controlpy/
      - Github page with modified controlpy: https://github.com/markwmuller/controlpy
      - Tools for synthesizing controllers for LTI systems: https://github.com/markwmuller/controlpy/blob/master/controlpy/synthesis.py
        - You can clone this file and download it to use with springMassDamp.py. (figure out variable names to change if you use this provided file).
      - These are useful because it lets you use controlpy without having to install Slycot as in the Python Control Systems Library by Richard Murray:
        - https://pypi.org/project/control/0.6.6/
- What's still needed
  - Cleaner options for user input
  - Simulation
  - Accounting for non full state estimation
  - Introducing logic -- what to do if the system isn't controllable, etc.
  - Clearer variable names
  - Guidelines to translate from theory to irl.