ME231B/EE220C: Homework – Programming project
Due: 12:00 (midday) on Friday, 5 May – for 100 P.

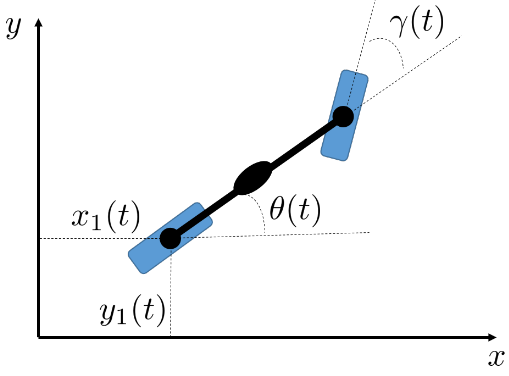
Instructions: You may collaborate with up to 2 other students to solve this assignment. You may use either *Python* or *Matlab* for your solution. Any code must be sufficiently well documented (comments) that it can be easily understood.
Submitting: See submission instructions.
Late submissions: you will lose 20 percentage points if your homework is late by less than eight hours. For each additional hour, you lose an additional 10 percentage points.
Academic conduct: Please see `http://sa.berkeley.edu/conduct/integrity/definition`. We have no patience for misconduct.
Last update: 2023-03-24

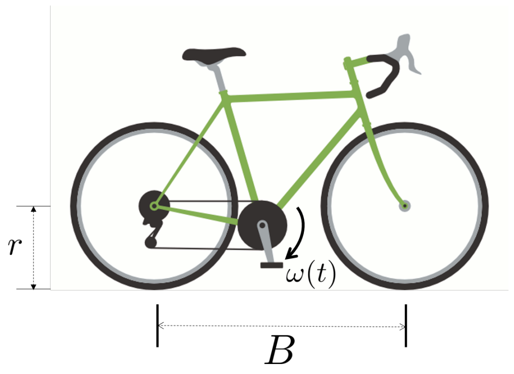
Introduction

This problem set requires you to implement a state estimator for a complex system. The task, though simplified, is similar to what you may experience "in the field". Specifically, your goal is to implement a state estimator to track the position and heading of a bicycle as it moves. You are free to implement any estimator structure / method that you like, and you will be scored partially on the efficacy of your estimator (specifically, how well you estimate the position of the bicycle over time). Your estimator must be able to run in a reasonable time, i.e. we will penalize you if your submission is deemed excessively computationally expensive (this is made precise below). We encourage you to experiment with different estimator strategies to compare efficacy.

The system

The system is illustrated below[1]





where $(x_1(t), y_1(t))$ is the position of the rear wheel, $\theta(t)$ is the heading, $\gamma(t)$ is the steering angle (relative to the bicycle frame), $\omega(t)$ is the pedaling speed, r is the wheel radius, and B is the wheelbase. The bicycle moves in a flat horizontal plane, and the y-axis indicates North, while the x-axis indicates East.
The bicycle dynamics are given by the below equations

$$\dot{x}_1(t) = v(t)\cos(\theta(t)) \tag{1}$$
$$\dot{y}_1(t) = v(t)\sin(\theta(t)) \tag{2}$$
$$\dot{\theta}(t) = \frac{v(t)}{B}\tan(\gamma(t)) \tag{3}$$

where $v(t)$ is the linear velocity of the bicycle. The bicycle is geared such that the angular velocity of the rear wheel is 5 times the pedaling speed.

[1]Picture of bicycle courtesy of https://www.ebicycles.com/bicycle-tools/frame-sizer/road-bike

Known signals

Position measurements $p(k)$ of the bicycle center (with units meters) are collected at discrete times t_k. An idealized model of these position measurements is given by

$$p(k) = \begin{bmatrix} x_1(t_k) + \frac{1}{2}B\cos(\theta(t_k)) \\ y_1(t_k) + \frac{1}{2}B\sin(\theta(t_k)) \end{bmatrix} \tag{4}$$

The measurements are available at a period of approximately 0.5 seconds, but they do not necessarily come at exactly this rate. The measurements are furthermore known to be unbiased.

The bicycle is furthermore instrumented so that the bicycle's steering angle $\gamma(t)$ is known at all times, and similarly the pedal speed $\omega(t)$ is known. You may assume that these quantities are piece-wise constant.

You also know that, for each provided data set, the cyclist starts near the origin (0,0) and is initially heading approximately North-East.

Uncertainty

The measurements are corrupted by uncertainty, due to electrical noise in the sensor, timing imprecision, and atmospheric disturbances that warp the path of the GPS signals. Furthermore, the bicycle's physical parameters are not known perfectly: as a function of manufacturing tolerances, and flex due to the bicycle rider's weight, the baseline B is uncertain (to within approximately $\pm 10\%$). Furthermore, due to wear, varying levels of inflation pressure, and different models of tires, the tire radius r is uncertain (to within $\pm 5\%$). Finally, the bicycle's wheels may slip (slightly), and the steering angle and pedal speed information may not be perfect.

The manufacturer's nominal values are given below:

$$r = 0.425\text{m} \tag{5}$$

$$B = 0.8\text{m} \tag{6}$$

Provided data

You are provided a set of comma separated text files, that contain data corresponding to different bicycle rides. Each file corresponds to a different cycle ride, with a different bicycle and different rider. A calibration data file is provided (identified as experiment 0), that corresponds to position measurements for a stationary bicycle – you may thus use this file, for example, to characterize the position measurements.

You do not need to understand the format of the files, as they are parsed by the provided code. However, for completeness we describe their format, and specifically the data contained in each column:

1. current time in seconds
2. current steering angle in radians
3. current pedal speed in radians per second
4. current measurment x (given as NaN if no measurement at the current time)
5. current measurment y (given as NaN if no measurement at the current time)
6. true position x in meters
7. true position y in meters
8. true angle θ in radians

The last three columns will be NaN (not a number) for all but the last time instant, and are provided so you can get some feeling for how well you do.

We provide you sufficient data sets that you can exhaustively characterize your estimator's performance. When we score you, we will use a dataset that was not originally provided to you.

Consider run 1 as "evaluation data", as discussed in the deliverables section, below.

Implementation

There are two possibilities for submitting your solutions: using either Python or Matlab. The complete set of code files provided to you are described below:

main This is the main file, that will run your estimator, record the outputs, and generate some figures. Feel free to modify this file for your development, but you will not submit this file. If you decide to modify it, keep in mind that you cannot modify the interface to the estimator – we will run your estimator with our own (original) main file. You can set which data file is loaded by modifying the value of 'experimentalRun'.

estInitialize *Modify & submit* This function is called before any data arrives, and its purpose is to initialize your estimator. You may use it, for example, to create an initial guess of the system's state at time 0. The function returns an object called 'internalState', which must be in the same format as the argument to the 'estRun' function.

estRun *Modify & submit* This function is run at every time instant, and has multiple arguments, including timing, the estimator internal state (in the same format as is returned by your estInitialize), information on the current steering angle and pedal speed, and the latest measurement. You may do whatever you like in this function, but it must be a reasonable computation (i.e. if your code takes more than 30 seconds total to execute on a modern i7-based laptop, we will penalize you).

Implementation notes

Python Your code will be run using Python 3.10. You may use *only* the libraries `numpy` and `scipy` in your solutions.

Matlab We will run your code *without any matlab toolboxes, except the standard controls toolbox.* It is your responsibility to ensure that you don't rely on matlab toolboxes (i.e., check which functions you're calling).

Code that doesn't run If we cannot execute your code, we will give the error only a cursory look. If the issue can be easily resolved by us, we will do so and apply a moderate penalty. If we cannot, we will send the code back to you to fix, and we'll apply a penalty to your submission. The magnitude of such penalties will be determined by us, and are non-negotiable.

Plagiarism Make sure that you understand what constitutes plagiarism. Adapting a solution you found online will be considered cheating here; we will be merciless on suspected dishonesty.

Deliverables

You must hand in two things at the end of the project: a brief report describing what you implemented, the design decisions needed, and a discussion of results, and the implemented estimator function. Note – you must submit *only one* solution; you may of course test multiple, but only submit one.

Scoring

We will execute your estimator on "standard data", and generate a numerical performance score (based on your estimation error, compared to the true position (known to us)). The final score is broken down as follows:

- Modelling 25%
- Design decisions and justification 25%
- Evaluation & discussion using evaluation data (run #1) 40%

- Estimator performance as evaluated by instructors, relative to your peers 10%

Besides being technically correct, your report must be clear and concise. We may apply a penalty based on "signal-to-noise ratio" if we perceive that your report is not clearly written, neat, and concise. A similar penalty may be applied to your code, if it is insufficiently clear. Code clarity means two things: 1) variable and function names are sensible, the flow is clear, and 2) comments provide context for the code. Comments that simply rewrite the code (e.g., 'x = 5; # here we set variable x to 5') are considered noise and may be penalized more than having no comments – instead comments should clarify intention or reasoning.

Submission

Submit your solution through Gradescope. If you are working in a team, submit only once for the team.

Report Submit your brief report to "Project 1 Report". After submission, add your group members on Gradescope by clicking "Group Members" on the bottom bar.

Code Upload "estInitialize" and "estRun" ONLY to "Project 1 Code". After submission, add your group members on Gradescope by clicking "Group Members" on the bottom bar. The code must be text files, either `*.py` or `*.m`. Only submit *one* implementation.

Typical scores

The instructor's estimator has the following final errors for the first few data sets:

Run #	x [m]	y [m]	θ [rad]
1	0.246	0.304	0.066
2	0.110	0.198	0.091
3	-0.139	0.594	0.210
4	0.235	-1.580	0.059
5	-0.147	-1.419	-0.001

Plagiarism

All submissions will be tested for plagiarism. If you are unsure of what constitutes academic misconduct, please see `http://sa.berkeley.edu/conduct/integrity/definition`. Dishonesty will not be tolerated.