Introduction

The final lab report tests your understanding of all EECS 16B Labs, with an emphasis on conceptual and analytical understanding. It also allows you to look at these labs from a bigger picture and reflect on your design process and choices. You may use your homeworks, pre-labs, labs, lab notes, presentation slides, and any other resources we provided throughout the semester to help you. **However, all of your answers and explanations must be in your own words; you are not allowed to directly copy from those resources.**

This final lab report is an abridged version of the lab report from last semester, so the style of this lab report is quite different than the midterm lab report. Due to a lack of standard checkoffs, **Section 4 (Classification) will not be graded**.

Requirements

Format

The report is to be done with your lab group using LaTeX or Google Docs/Microsoft Word. **At the top of the report, please include the names and emails of all your group members, as well as the group ID you use for checkoffs.**

Contributions

Under Section 7, please detail each group member's contributions to the lab report. **If we find a highly disproportionate amount of work distribution among the group, we will adjust grades accordingly to penalize non-contributors.** Please cite any sources outside of course materials, if used.

Submission

The final lab report is (tentatively) due on Friday, December 9. **Only one group member should submit the lab report to Gradescope and the rest of the group members should be added to the same submission.**

1 System ID

1. How did you choose the region to collect finer data on (data_fine.txt)? Why is it important to choose such a region to run least-squares regression on?

   Solution: We select a region where the data points from data_course.txt are fairly linear.
   It is important because we're modeling our data using least-squares regression, which best models linear data. The best model would apply if the relationship between velocity and PWM input u[i] is linear.

2. What do θ and β represent physically, not mathematically?

   Solution: θ - change in velocity for each unit increase in PWM input. Basically, the "sensitivity" of the motors. Not "slope." Always positive.
   β - constant velocity offset, can be understood as the input required to overcome static friction. Imperfections catch-all due to external factors from the surroundings or environment. "Constant offset in velocity" is also acceptable. Not "y-intercept or negative of y-intercept." Not "speed without any voltage input." Usually negative.

3. Why do we have separate θ and β values for the left and right wheels?

   Solution: Both motors are different, and may react differently to a change in input PWM (have different sensitivities). They also likely have different resistances that affect the "constant velocity offset" needed to appropriately model them or to overcome any external imperfections from the surroundings or environment like static friction.

4. Why do we set v^* to the midpoint of our overlapping wheel velocity range, instead of closer to the boundaries? What would happen if we operated our car at a velocity outside of the overlapping velocity range?

Solution: To maximize margin of error on both sides, so that both wheels can continue following our model, both when we're going straight and turning.
If we operate our car outside of this overlapping velocity range, at least one wheel will not be able to drive at that velocity. This means that the two wheels will travel at different velocities and the car will never be able to go straight. Also, it is likely that our linear model no longer holds, and the approximation gets worse. We expect that the further outside the operating range we get, the worse we get at predicting the actual velocity the car will run based on a given input PWM. This means our car will likely not drive as intended.

2 Controls Part 1

1. What are the open-loop model equations for our PWM input, $u[i]$?

   Solution:

$$\begin{aligned} u_L[i] &= \frac{v^* + \beta_L}{\theta_L} \\ u_R[i] &= \frac{v^* + \beta_R}{\theta_R} \end{aligned} \tag{1}$$

2. What is the purpose of the jolts? Why might the left and right jolts be different?

   Solution: In order to start our cars from rest, we must overcome static friction, and our PWM inputs may not be sufficient for this. In order to overcome friction, we provide each motor with a large PWM for a brief period of time/first couple of time steps.
   The two motors have different sensitivities to the PWM input (theta) and different velocity offsets (beta), so they require different inputs to overcome the static friction. This is also evidence of difference in motor parameters, motor efficiencies, or mass imbalance of the car.

3. Why does open-loop control fail? Why do we need to implement closed-loop control in order to have the car travel straight?

   Solution: In open-loop control, we set the PWM inputs once and let the car do its thing without correcting its behavior during its run. The car can't tell the difference between whether it's driving straight or turning. Note that the PWM inputs for open-loop are always constant and are not adjusted at all.
   Because of model mismatch cases (difference between actual and model theta/beta values) and disturbances/noise introduced by the environment, we need to implement closed-loop control to have the car travel straight.

4. What are the closed-loop model equations for our PWM input, $u[i]$? Explain the purpose of each term.

   Solution:

$$\begin{aligned} u_L[i] &= \frac{v^* - f_L\delta[i] + \beta_L}{\theta_L} \\ u_R[i] &= \frac{v^* + f_R\delta[i] + \beta_R}{\theta_R} \end{aligned} \tag{2}$$

   $u_L[i]$ and $u_R[i]$ - input PWM for each wheel, controls velocity of the wheels
   v^* - operating velocity point
   θ - change in velocity per unit increase in input PWM, "sensitivity" of the motors to change in PWM
   β - constant velocity offset, represents friction and other imperfections
   f_L and f_R - feedback control gain for each wheel, proportionally scales delta, defined to be non-negative
   $\delta[i]$ - state variable, representing difference in distances traveled between left wheel minus right wheel

5. When testing out different f-values in practice, how do you know if the system eigenvalue has gone from positive to negative based on the car's behavior?

   Solution: Eigenvalue moves to negative when we start seeing oscillations in the car's movement. This is because delta's polarity will flip every time step.

6. What effect does setting both f-values to 0 have on the car's control scheme? How is this different from non-zero f-values? Why are non-zero f-values necessary?

   Solution: Open-loop control if both f-values set to 0. It doesn't take feedback anymore, so non-zero f-values are needed in order to use feedback control to correct the car.

7. Why can't we use negative f-values for both wheels? If we wanted to use negative f-values for both wheels, how should we change our closed-loop model equations such that our car goes straight and corrects any errors in its trajectory?

   Solution: Negative f-values for both wheels would mean that we're reinforcing the disturbance (positive feedback). It also brings the magnitude of the eigenvalue over 1, which causes the system to become unstable.
   Method 1:

   (a) $u_L[i] = \dfrac{v^* + f_L\delta[i] + \beta_L}{\theta_L}$

   (b) $u_R[i] = \dfrac{v^* - f_R\delta[i] + \beta_R}{\theta_R}$

   (c) $\delta[i+1] = (1 + f_L + f_R)\delta[i]$

   Method 2: $\delta[i] = d_R[i] - d_L[i]$

8. What does a zero `delta_ss` value tell you about your car's trajectory? What about a non-zero `delta_ss` value? What kind of error is it supposed to correct when we add it to our control scheme? (**Hint**: Think about the difference between the trajectories for a zero versus a non-zero `delta_ss` value.)

   Solution: Zero `delta_ss`: Car goes straight in the same exact heading as when it started.
   Non-zero `delta_ss`: Car has turned and is now driving straight, but in a different direction/heading than it was initially placed in.
   Steady-state error correction: error caused by model mismatch

3 Controls Part 2

1. How did you change the closed-loop model equations to allow the car to turn? Write the equations below and explain how they change for turning left, turning right, and going straight.

   Solution: Turn by adding time-dependent `delta_reference` to delta, $\delta = \delta + \delta_{ref}$.
   Left: return positive `delta_reference`, so $\delta = \delta + \dfrac{lv^*i}{mr}$
   Right: return negative `delta_reference`, so $\delta = \delta - \dfrac{lv^*i}{mr}$
   Straight: return 0 for `delta_reference`, so $\delta = \delta + 0 = \delta$

2. Why do we divide v^* by $m = 5$ for the turning expressions?

   Solution: Because our control loop and data collection have different sampling frequencies and periods, and we want to add in our `delta_reference` in each time step of the control loop. Our data collection was sampled every half a second, while the controller was sampled every tenth of a second. Essentially, the control loop was sampled 5 times faster than the data collection, $m = \frac{F_c}{F_d} = 5$.
   To correct for this, we want to make our velocity (distance/timestep) 5 times smaller, so we use $\frac{v^*}{m}$ instead of v^*.

3. How is using `STRAIGHT_CORRECTION` different from `delta_ss` in Controls Part 1?

   Solution: `delta_ss`: steady-state error correction, delta as time step i approaches infinity and converges. Supposed to correct when the car sometimes turns to a different direction/heading and then drives straight. Delta must converge to a constant (zero or non-zero) value for the car to drive straight (wheels have equal velocities). `delta_ss` is added to delta in every time step.
   `straight_correction`: used when car (encoders) think car is driving straight (delta converges to constant (zero or non-zero) value) but in reality it's still turning slightly at a constant rate; supposed to instruct car to turn

slightly in opposite direction to correct for mechanical differences like axle mismatch/wobble, etc.
They are different in how the car behaves. Both `delta_ss` and `straight_correction` involve cases where our closed-loop controller sees that delta approaches a constant value (`delta_ss`), but the car's path/trajectory is different in the two cases (they are used to correct different things). This is because for `delta_ss`, the car is already traveling straight but for `straight_correction`, the car is still turning.

4 Classification

1. What are some characteristics of a good set of four words for classification? Provide at least two features.

   Solution: A good set of four words will have different envelopes – the shape of their speech vectors should be and look distinct. At least two features must be provided, including but not limited to the following: different number of syllabus, words or parts of words that can be enunciated differently / sound distinct, words with different endings, etc.

2. What are length, prelength, and threshold for our data processing? Include both the definitions and the values you chose.

   Solution: Length, prelength, and threshold are used for word alignment to clean the data set.
   Length: total number of samples/features for each word
   Prelength: number of samples/features before threshold is reached
   Threshold: Fraction of max signal, used to determine the start of the word within the recording
   Provided reasonable values for all 3 (default values were length 80, prelength 5, threshold 0.5)

3. Why do we process our data so that the words are aligned before we run SVD/PCA on it?

   Solution: Looking for features in the words, so we need the data to have a common shape first by aligning words when preprocessing recordings. Otherwise, the data is all over the place and has no common trend over time.

4. Why do we need to use SVD/PCA to represent our data set?

   Solution: SVD/PCA helps us choose the most important features out of our data set and performs a dimensionality reduction. Launchpad/Arduino has memory/space limitations, so it's easier to work with if we project our data onto 2-3 dimensions from length-dimensions.

5. Why do we use the V^T vectors for our lab instead of the vectors inside of the U matrix returned by SVD?

   Solution: The columns of V (rows of V^T) hold the principal components/features of the rows, while the columns of U hold principal components/features of the columns. Our words are stored in rows (we vertically stacked the horizontal recorded word vectors to make the data matrix), so we take the V^T row vectors since we're interested in the features of the words.

6. Why can we simply take the dot product when projecting our recorded data vector onto the principal component vectors?

   Solution: We care about scalar projections. The matrix of projected coefficients XP, where X is our demeaned data matrix and P is our basis of principal component vectors, all contain scalars. Since the principal components are orthonormal, they have unit norm and a dot product between the recorded data vector and the principal component vector is sufficient (we do not need to divide by norm of principal component vector since it is 1).
   But why are the principal components orthonormal? The U and V^T matrices returned by the SVD of our demeaned data matrix are orthogonal/orthonormal matrices, so their row and column vectors are all orthonormal and have unit norm.

7. If we keep increasing the number of PCA vectors, how does the increase in accuracy with each subsequent PCA vector change?

   Solution: This is like the Law of Diminishing Returns. As we increase the number of PCA vectors / principal components, the classification accuracy of our data set increases but slows down over time (increasing but at a decreasing rate).

We reach a point where adding the number of principal components is no longer significant, since the drawback of increasing memory on Launchpad/Arduino outweighs the small accuracy increase.

8. What is `EUCLIDEAN_THRESHOLD`? What is `LOUDNESS_THRESHOLD`?

   Solution: `EUCLIDEAN_THRESHOLD` sets the maximum distance that the recording can be from any centroid after projection and demeaning. If it's too far from the closest centroid, the recording is thrown away and not classified as an instruction.
   `LOUDNESS_THRESHOLD` sets the minimum amplitude/loudness of the recording before the recording is recognized as a word. Otherwise, it is discarded and treated as noise.

5 Integration/Final Demo

1. Briefly discuss what you learned throughout the S1XT33N car project and in the labs. What was your favorite part? Least favorite part? Please answer this question individually.

   Solution: Graded on effort.

2. What was the most difficult bug you encountered this semester? How did you resolve the bug? What did you learn from the debugging experience?

   Solution: Graded on effort.

6 Feedback

Please provide any feedback you have about 16B lab or anything we can do to better support you.

7 Collaborators and Sources

Please detail each group member's contributions to the lab report. Also, cite any sources you used that were not provided with the course materials.