Lab #6 Report: Path Planning

Team #1

Kyri Chen
Alex Ellison
Ashley Holton
Jordan Ren
Laura Rosado

6.141: RSS

April 21, 2022

1 Introduction (Kyri)

At the conclusion of the previous lab, our autonomous robot (Beatrice) was able to localize itself in a given map. However, Beatrice can hardly claim to be an autonomous robot if she cannot navigate her way throughout a map without human guidance. In this lab, we tackle the problem of finding and traversing a collision-free path within a known environment.

This problem was approached in two parts. First, given a start and end point, we wish to efficiently find a collision-free path from the start to the end. We accomplished this through A*, a search-based method that computes a path over a discrete representation of the map. Next, we implemented pure pursuit, a trajectory-tracking algorithm that leverages geometric identities to calculate the correct steering angle. Pure pursuit will allow Beatrice to follow the path found during an A* search. The combination of these two parts will finally give Beatrice the ability to plan paths and autonomously drive along them.

2 Technical Approach (Laura)

Each of the two main components in this lab has its own ROS node. The first is path planning, which publishes a trajectory represented as a series of points. The second is pure pursuit, which subscribes to the trajectory topic and publishes drive commands to the robot. This section will detail our design choices and methods while developing these two nodes.

2.1 Path Planning (Laura)

Before choosing a path-planning algorithm, we first considered the constraints of the problem:

- Path-planning is done using a static map of the Stata Center basement.
- The map is represented as an occupancy grid, where obstacles at a given point are represented as a probability from 0 to 100.
- Searches, and by extension paths, are two-dimensional.

It was also useful to define our own performance expectations, which was for paths to be found in under a minute.

With the problem well-defined, we chose A* for our search algorithm, the implementation of which will be described in the following sub-section. The search space is a fully defined grid-space, so A* will return an optimal path so long as the heuristic is admissible. We also chose perform the search in the pixel coordinates of the map image and only transform the final path into "real life" coordinates before publishing. There are two advantages to this choice: any neighboring points to a given point can be easily inferred and it is easy to index into the occupancy grid to check for potential obstacles.

2.1.1 A* Search (Ashley)

Once a goal pose has been broadcasted to the path planning node, a callback function in the node triggers a search for a path from the robot's current position to the goal.

The A* algorithm, also known as best-first search, considers potential paths with the lowest estimated total distance first. The algorithm estimates the total distance of a potential path by summing the distance taken to the path so far and a heuristic estimate of the distance from the current point to the goal. The algorithm explores the paths in order by taking them off of a queue sorted by lowest estimated total distance. The paths are built up by taking the last point of the path currently being considered and making a new path that adds each of the point's neighbors. By keeping track of what points have been expanded into paths and never expanding a point more than once, we can make sure not to duplicate work. The algorithm ends once the goal is reached or there are no more paths to examine on the queue.

Once a path is found, we convert the points in the path from the map coordinate system to the real-world coordinate system so that our paths will be deployable in real-world situations. This path is then published as a trajectory consisting of points, which the pure pursuit controller subscribes to.

2.2 Pure Pursuit (Kyri)

The purpose of a pure pursuit controller is to track a given reference trajectory closely and smoothly. Trajectory-tracking is achieved by selecting a reference point at a fixed lookahead distance, denoted ℓ_d, from the vehicle, and computing a steering angle δ such that the vehicle would hit the reference point if the steering were kept constant. The intuition of this algorithm comes from the idea that a human driver might constantly steer to hit a point on the road a short distance ahead of their vehicle. In our implementation, We first find the point on the trajectory nearest to the car, then find the lookahead point in the neighborhood of that segment, and finally calculate the steering-angle to hit the given lookahead point.

2.2.1 Nearest Segment (Alex)

The first step in developing our pure pursuit algorithm was to find the closest point on the trajectory to our car to limit the search space of our lookahead distance and ensure we are always moving forward. To find the closest point, one could imagine looping through all the points on some discretization of our trajectory and computing the distance to each point, storing these distances, then taking the minimum and returning that point. In practice, this method is very inefficient, so instead we calculate the closest point on each segment using projections, and then return the start point of the line segment that contained the closest point. To accomplish finding the overall closest point we looped through each of the segments specified by a start point p_1 and end point p_2 and computed the closest point on the segment to our current point c as follows:

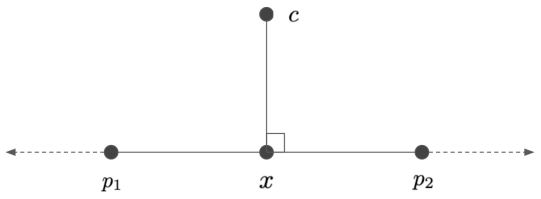

Figure 1: Geometric setup for finding the closest point on line segment to another non-colinear point.

1. The first step in calculating the closest point on a line segment to a non-colinear point is to notice that the point will lie on the line that goes through the point and the line segment and is orthogonal to the segment. You can see these relationships in Figure 1 which showcases an example setup of points p_1, p_2, c, and x which will be described below. With the information just discussed, we can start computing the closest point with

the following equations:

$$x = p_1 + t(p_2 - p_1) \tag{1}$$

$$(c - x) \cdot (p_2 - p_1) = 0 \tag{2}$$

In the above equations x is defined as a point lying on the line extending from the segment connecting p_1 and p_2 parameterized by t and we derive the second equation from the fact that the dot product of two orthogonal vectors is 0. Now substituting equation (1) into equation (2) and simplifying we are left with an equation for the parameter t to give the point on the line closest to c. The variable t can be thought of as how far along on the line the point lies.

$$t = \frac{(c - p_1) \cdot (p_2 - p_1)}{\|(p_2 - p_1)\|} \tag{3}$$

2. The next step is to clip the value we calculate for t to lie on the range $[0, 1]$ since we only care about points that lie on the line segments themselves. The intuition behind this is that our value of t can give us a point anywhere on the line extending from our segment connection p_1 and p_2, so in order to ensure our equation can only give points between p_1 and p_2 we can simply constrain our t values to $[0, 1]$. One can observe this by plugging in $t = 1$ and $t = 1$ into equation (1) or looking at Figure 2 to see the geometric setup of the math done so far. The green rectangle represents Beatrice (our robot), the red dot labeled $t = 0$ represents the first point on the segment (p_2) and $t = 1$ represents the second point (p_2).

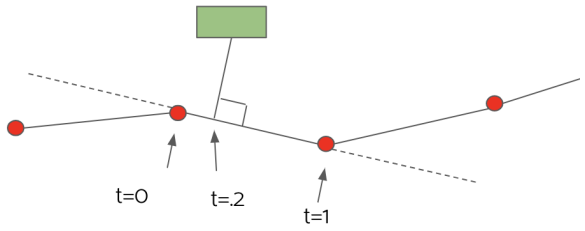

Figure 2: Visual representation of finding the closest point on trajectory with visualization of how the parameter t is used. The green rectangle represents our robot while each of the red dots represents a point on the trajectory.

3. Finally, to return the point itself we simply plug in our calculated t back into equation (1) to get x and return the starting point of that line segment!

After we store this for all the line segments, we found the point across all line segments that had minimum distance and return that as the closest point.

2.2.2 Lookahead Point (Jordan)

Since all potential lookahead points must lie on both the perimeter of a circle of radius ℓ_d centered at the car and the trajectory, the solution is to find an intersection between the circle and any of the line segements that compose the path. Because we know the closest point to the car on the trajectory, we can save a significant amount of time when finding the lookahead point by only searching through path segments beyond that closest point. In our implementation, we chose to search through the upcoming 6 path segments to balance completeness and computation speed.

For each line segment that begins at p_1 and ends at p_2, we express it as $v = p_2 - p_1$ and paramaterize it as $p_1 + tv$ for $t \in [0, 1]$. We denote the car's location as q, and all points will be represented as a vector from the origin to that point. A point x is on the circle if its distance from the car is equal to the radius, or if

$$|x - q| = \ell_d. \tag{4}$$

Since x lies on some point along v, we can substitute $x = p_1 + tv$ into the expression and attempt to solve for t. Squaring both sides and expanding, we obtain a quadratic equation in standard form with coefficients

$$a = v \cdot v \tag{5}$$

$$b = 2(v \cdot (p_1 - q)) \tag{6}$$

$$c = p_1 \cdot p_1 + q \cdot q - 2p_1 \cdot q - \ell_d^2. \tag{7}$$

Plugging these coefficients into the quadratic formula, we obtain two values of t such that

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}. \tag{8}$$

Since the line segments are only defined for $t \in [0, 1]$, we will only keep solutions that fall within that range. We will always have multiple solutions, so we choose the solution furthest along the path to keep the lookahead point as far forward as possible.

2.2.3 Steering Angle (Kyri)

Before we can calculate the steering angle, we first need to have a model of Beatrice's lateral movement. For our implementation, we use the standard kinematic bicycle model shown in Figure 3.

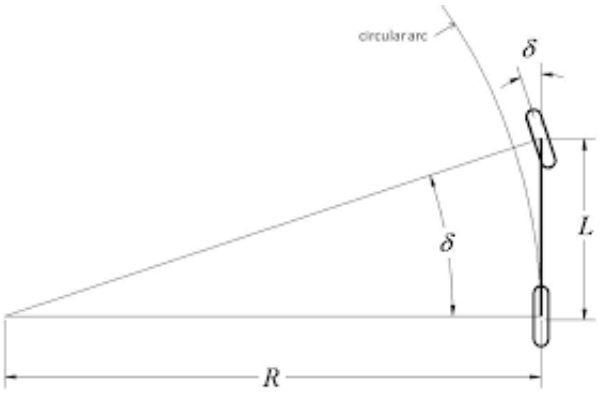

Figure 3: Standard kinematic bicycle model.

When a car with axle-length L steers at an angle δ, it will follow the arc of a circle with radius R. Therefore, if we know the arc we wish to drive along, we can find the proper steering angle through the trigonometric relation

$$\delta = \tan^{-1}\left(\frac{L}{R}\right). \tag{9}$$

With a kinematic model in place, we can now examine the geometry behind steering towards the reference point. As shown in Figure 4, the desired turn traces out an arc of radius 2α, where α is the angle between the car's current heading and the reference point. Using the isosceles triangle composed of the rear-axle, the reference point, and the center of rotation, we can see

$$R = \frac{\ell_d}{2\sin\alpha}. \tag{10}$$

Plugging this into our equation for δ, we arrive at our solution of

$$\delta = \tan^{-1}\left(\frac{2L\sin\alpha}{\ell_d}\right). \tag{11}$$

In our implementation of pure pursuit, we are immediately given most of the values in this equation. We can find L using a tape measure and ℓ_d is a given value that's constant throughout the drive. We can take advantage of the current pose of the robot and some vector mathematics to find α. If the current pose of the robot is (x, y, θ), then $a = (\cos\theta, \sin\theta)$ is a unit vector pointing in the heading of the car. A simple subtraction of the current pose from the goal pose gives the reference vector $b = (g_x - x, g_y - y)$. We use a vector identity to calculate

$$\alpha = \cos^{-1}\left(\frac{a \cdot b}{\|a\|\|b\|}\right). \tag{12}$$

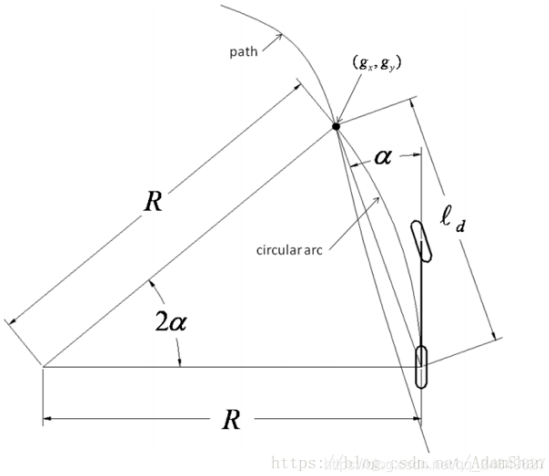

Figure 4: Illustration of pure pursuit.

This calculation always results in a positive angle, so we will only ever get a positive steering angle. When the reference vector b is to the right of the car's heading a, we wish to flip the sign of the steering angle to turn right. We note that the cross product $s = a \times b$ is negative only when the above condition is true, so we can derive our final steering angle formula as

$$\delta = s \cdot \tan^{-1}\left(\frac{2L \sin \alpha}{\ell_d}\right). \tag{13}$$

As we tested our pure pursuit in simulation, we noticed that a shorter lookahead distance would work very well when facing sharp turns but would cause the car to oscillate during straightaway sections. To maintain Beatrice's ability to navigate turns while reducing oscillations, we added a P-controller to the steering angle. Furthermore, we increased the lookahead distance when the angle between the current and upcoming line segments are less than $\frac{\pi}{6}$, which means the turn is very mild.

3 Experimental Evaluation (Laura)

Our path planning and pure pursuit nodes required different metrics to evaluate their performances. To evaluate our implementation of the A* algorithm, we compared its runtime with a simple breadth-first search. To evaluate our pure pursuit controller in simulation, we calculated the car's distance to the trajectory at each time step as a measurement of error. Finally, performance on the hardware was evaluated subjectively, as the ground truth is impossible

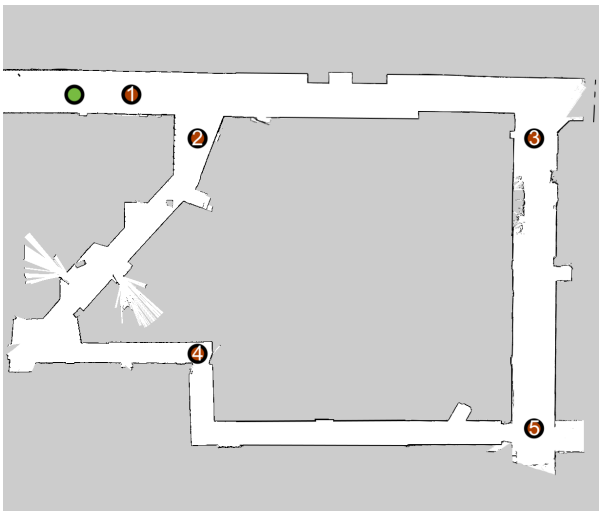

Figure 5: Map showing the approximate locations of start and end nodes used to evaluate path planning performance. Each set starts with the green dot and ends at one of the five numbered red dots.

to determine in real life.

In the process of developing our path planning and pure pursuit nodes, we also wrote a variety of unit tests to ensure individual functions were performing as intended. Since these unit tests were an intermediate evaluation, we will not discuss them here.

3.1 Path Planning Performance (Laura)

Tables 1 and 2 report the average time in seconds it took to find a path between two points on the Stata Basement map, taken over three attempts with no visualization programs running. The five scenarios are shown in Figure 5, chosen to test our search algorithm in varied scenarios.

Initially, neighboring points were defined as the four points horizontally and vertically adjacent to the current point. The resulting data is shown in Table 1. On average, BFS found a solution faster than A* in four of five scenarios. The only exception is in set 1, where the difference is in milliseconds. We attributed this disparity to the sorting function applied to our queue in every iteration of our A* search.

To speed up the path-finding process, we changed the definition of neighboring points to be the eight points adjacent to the current point. The data from this is reported in Table 2. This change is disadvantageous to BFS, since it increases

Table 1: Runtime comparison between BFS and A* algorithms for 5 sets of start and end points, only considering horizontally and vertically adjacent neighbors while constructing paths.

	Set 1	Set 2	Set 3	Set 4	Set 5
BFS	0.0341 s	9.13 s	5.33 s	8.93 s	19.5 s
A*	0.0244 s	11.9 s	23.1 s	13.0 s	24.0 s

Table 2: Runtime comparison between BFS and A* algorithms for 5 sets of start and end points, considering all adjacent neighbors (including diagonals) while constructing paths.

	Set 1	Set 2	Set 3	Set 4	Set 5
BFS	0.109 s	7.86 s	8.73 s	8.74 s	24.9 s
A*	0.0153 s	10.7 s	0.521 s	11.2 s	16.5 s

the number of paths added to the queue. However, BFS did not see a significant increase in time compared to the first iteration. On the other hand, A*'s performance improved significantly. Though neither has a clear advantage over the other,

There are further optimizations that could be implemented to improve the speed of our A* search, but as both versions accomplished our goal of finding a path in under a minute, this performance was sufficient.

3.2 Pure Pursuit Performance (Jordan)

We evaluated the performance of our pure pursuit controller by measuring the distance of our car's predicted position to the closest point of the path during the traversal of the trajectory. Over three runs of our pure pursuit on the path on Figure 7, we got mean squared errors of 0.0118m, 0.00631m, and 0.00393m (Figure 6). This meant that there was little deviation from the path as a whole while using pure pursuit. As our car approached tight corners in the path it was able to adjust its look ahead distance accordingly to never cut through any walls, while making its look ahead distance large again on straightaways to avoid oscillation over the path. On gradescope we received scores of 2.852 out of 3.0 for pure pursuit and 3.947 out of 4.0 for integration, where pure pursuit followed 95% and 96% of the path respectively. Overall we found the results of our pure pursuit satisfactory, with it being robust enough to handle multiple different speeds and turns.

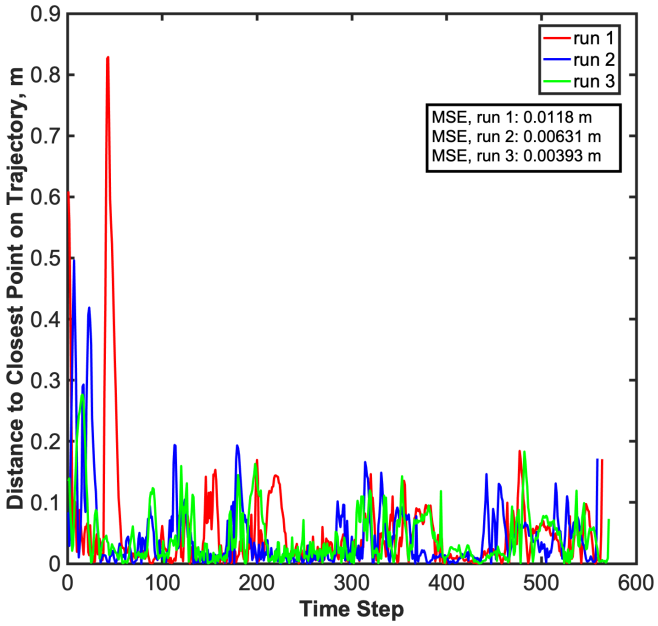

Figure 6: Car distance to closest trajectory point while following the path shown in Figure 5 over three trials.

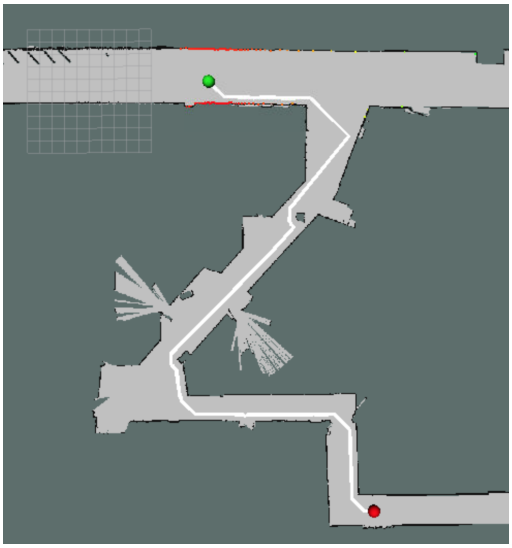

Figure 7: Approximate path used to evaluate pure pursuit performance in simulation.

3.3 Hardware Integration Performance (Jordan)

The goal of the hardware integration was to have our car Beatrice be able to traverse from any point on the stata basement map to another without any aid. This involved getting localization running on the car, then running both pure pursuit and path planning for the car to follow a given path. While in theory this should have been a simple integration step, our team ran into multiple issues while attempting the hardware integration.

In order for us to be able to use our path planning and pure pursuit we needed to have our localization up and running so the car has a point of reference to begin path planning. While we were able to get localization working on the car for the localization lab, when trying to get the map server running these past two weeks we encountered a bug that stopped it from publishing the map transform. This meant that our localization solution was unable to figure out where it was on the map, as it had no map to reference. To debug this we tried multiple methods such as running the map server in isolation, manually publishing the map transforms, and scanning topics, nodes, and trees to ensure transforms were correct. After debugging this for approximately a week from help with the TAs, we were eventually given a new car, Bob, from the TAs. Since the TA car also had a Velodyne lidar we simply had to clone our code onto the new car. After debugging some issues with the lidar on the TA car, we once again ran into the same map server issue. While discussing next steps with our assigned TA Valerie and Professor Dylan, we decided it would be best for us to simply focus on the final race to be able to complete it in time. However, once we are able to perform lane following for the final race, we will be able to evaluate our pure pursuit controller's performance on hardware.

4 Conclusion (Ashley)

In this lab, we were able to successfully create modules for our robot that can plan a path between two points and follow that path. Our first implementation of the A* algorithm for path-planning took longer to run than breadth-first-search in most of the cases we tested it, and while we managed to close the gap with some small changes there is still room for further optimization. We could also explore using a sampling-based search method such as RRT for path planning. Our pure pursuit implementation for following paths also performed well, but could be improved by scaling the look-ahead distance with the current speed.

Even though we were successful in creating paths and following them, we ran into major issues integrating the two modules with localization and getting our implementation to work on the physical robot. Even though our integration worked fine on our local machines, our performance in the auto-grader did not fare as well. We found out that the problem was the way we were interpreting the angle in the look-ahead portion of pure pursuit was incorrect. Once we

fixed that issue, our performance in the auto-grader simulations was very good. Unfortunately, hardware problems outside of our control led us to be unable to test our solution either on Beatrice or the car provided by the teaching assistants.

5 Lessons Learned

5.1 Alex

This week was the most challenging one yet for me. Technically my portion of the code was relatively straightforward, but challenged me to learn how to optimize code with numpy which was good. The real technical challenge, however, was integration. It was super difficult to figure out what pieces were causing issues when we put stuff together so I learned a valuable lesson in isolating parts for testing to just look at the connection between adjacent pieces instead of the whole system at once to find errors. I also learned that collaborating during the process of testing integration was extremely important since instead of trying to figure out exactly what the other team members code was doing myself, I could just ask quick pointed questions that did not inconvenience them much but saved me loads of time. Finally, communication wise I think we did well this week in executing a good briefing, but as always I think we could of communicated better timelines as a team in which we needed our individual pieces to be done to allow ample time for integration.

5.2 Ashley

In this lab, I learned more about the nuances of the A* search algorithm as well as ways to optimize a search algorithm. It was also another lesson in the importance of getting done the individual parts of a lab early to have enough time to debug integration, since we seem to keep running into the most problems during the integration phases of labs. In future work, it could help to sit down as a team at the beginning of the lab and plan out milestones for when we want certain parts of the work done by to leave enough time for experimentation and debugging.

5.3 Jordan

This lab was one of the more exciting labs that we have done, as we had to incorporate a lot of our past code base to get this working. From a technical standpoint I thought that the process of finding a lookahead point to drive towards was simple enough, however it became more complex as we had to use numpy to vectorize our functions. Furthermore, in order to have a smooth following of the trajectory we had to create a variable lookahead distance based on speed and upcoming turns, which turned out to be more difficult than I imagined. I think that in the future I could save a lot of time by discussing some of my ideas for a portion of the lab with a TA before implementing them, as getting a second opinion on how to approach a problem is always helpful.

From a organizational standpoint I think that we did a great job on the pure pursuit section of the lab at first as we were able to get our individual sections working well early into the lab. However, we should have started the integration testing a little bit earlier as we struggled to fix a small bug until right before the lab was due. For the final we will be sure to integrate as early as possible.

5.4 Kyri

Although this lab was relatively straightforward in the algorithms we were asked to implement, the details of implementation and integration were extremely difficult. Vectorizing numpy operations was once again a challenge while implementing pure pursuit, and there was often some confusion into how messages/values were being published. During integration, we ran into a ton of trouble with the Gradescope submission while seemingly having everything working locally. Although I think we did a good job debugging issues as a team and asking for assistance in office hours, we once again underestimated the challenges of integration and didn't have enough time to finish our code. In terms of technical lessons, I think I learned everything covered in this lab extremely well and I'm glad I got to explore the nuances of finding paths and following them.

5.5 Laura

This lab was the first one where I felt comfortable enough with a concept to take the lead on implementing A*; of course it was also the lab where my section presented the most bugs. I wrote unit tests and did a lot of runtime diagnostics to get to the root of the problem, but in the end reaching out to a TA for help would have resolved the slow speeds earlier. Throughout this class, I have been learning how to communicate precisely with my teammates and course staff in order to make the most efficient use of our time. This was especially useful for when we spent a lot of time in office hours for this lab due to all of our hardware issues. While there are no bad questions I learned there are badly worded ones. I'm proud of the tenacity our team showed for this lab when confronted with so many challenges out of our control, and I hope we can continue this trend in the final stretch.