Algorithmic Determination of Vehicle Steering Angle from Plant Location Data

Work by,

Noah King, Kevin McConnell — Simpson College Students

With mentorship and guidance from,

Josue Calderon, Controls Engineer — Ag Leader Technology

Apurba Kumar Das, Controls Engineer — Ag Leader Technology

Introduction

Ag Leader technology was started by an engineer thirty years ago hoping to make farmers' lives better. This led to the invention of the first commercially successful on-the-go yield monitor, or a piece of technology installed on a combine harvester that measures crop yield. Today, they strive to continue bringing precision technology to the market. Ag leader pushes to help farmers "plan, plant, apply and harvest more efficiently and accurately" (Ag Leader Technologies, 2024). As a corporation, Ag Leader's mission is to "provide a workplace that inspires motivated individuals to create innovative products" (Ag Leader Technologies, 2024). For this project, we have Scientist Josue Calderon and Apurba Das as our mentors, who are both controls system engineers for Ag Leader. Our team's project aims to

dynamically track and correctly join a series of points in space (points already provided). This challenge involves identifying points representing rows accurately, dealing with curves, and handling variations in the number of points per row. Because Ag Leader specializes in precision farming, our project directly supports its mission by improving precision and efficiency in agricultural operations. In the past, farming has been done by manual steering, and by GPS steering. While GPS steering is more precise than manual, it's still not the final stages of accuracy, plants are not planted perfectly, so they cannot be harvested perfectly. Being able to identify exactly where plants are and adjusting the vehicle for this will take farming to a higher level of efficiency. This project's applications can be used in any stage past crop planting.

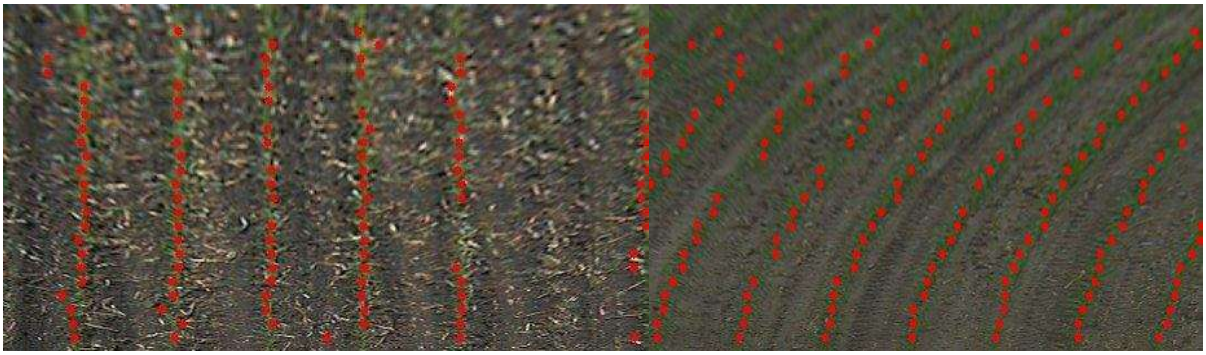
Our main focus is to classify these points into separate rows in order to determine the number of rows represented in a frame (see picture). While it is simple to look at a graph and classify which points go in which rows, a computer cannot judge by their "eye," so we must find a way to computationally explain why each point belongs in a row. We've attempted to split this into two separate problems at a smaller scale. 1: How do we mathematically determine where the rows are accurately in all scenarios, and 2: When our rows are accurately determined, how do we throw out bad data points, fit the individual rows, and mesh them to steer the vehicle with one input.

Our Mentor, Josue Calderon, has suggested several methods we could broadly look at this, this can be done visually with machine learning, or mathematically through algorithms. We have gone ahead towards the algorithmic direction. Part two is a compounding issue from part one. If part one is not accurate, part two will become harder. Thus, we've worked primarily on perfecting part one to ensure part two can go smoothly. Our data is provided as a set of x and y values for the

plants, for each frame of a 30 second video highlighting the different environments we can encounter. In an ideal scenario, the final product should be able to work for all environments, but currently they're being tackled individually with the hope that a final product can combine the best of both worlds. This final product would take the points in, classify them into rows, identify a fit for the rows, then find an average of those fits to finalize a steering direction from the center of the data.

Methods

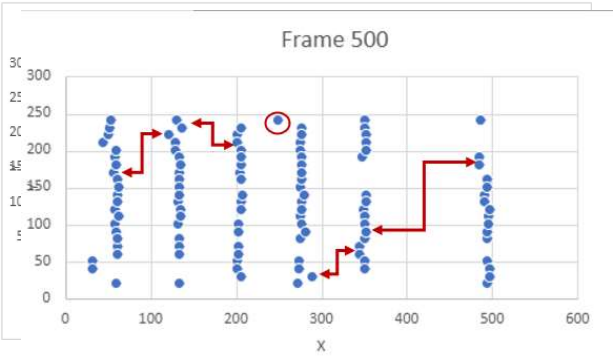
Our project was separated into stages by difficulty. We started with straight rows, then transferred our knowledge to the more difficult curves. An example image of these can be seen, although it should be noted curves have an incredible amount of variation in what they can do.



Straights are linear with near zero curvature, but they will still have some angle of slant. If they were completely vertical, then no steering would be needed. Curves often have some curvature, though some of the data in between can still appear like straights due to field geometry. When both row types intersect, headlands occur. Headlands are created when farmers do the outside of the field in a rectangular/circular motion, which lead to intersections in a near perpendicular fashion. This creates a field end, where you must recognize to stop steering based on the nonsensical imagery ahead, indicating you're entering a different section.

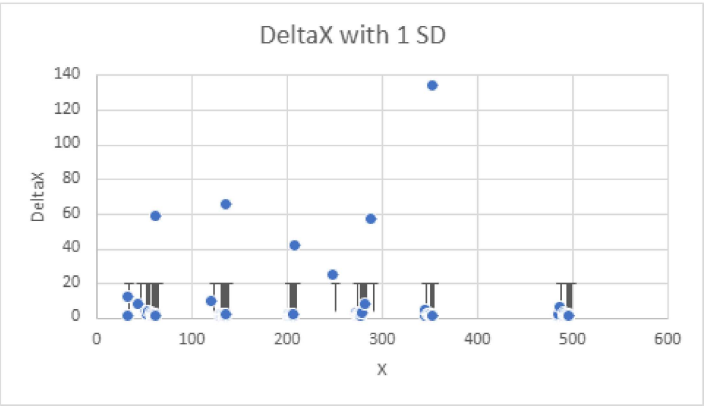
All row types were grouped into two separate parts: finding out how to group the data into rows mathematically, then cleaning the data, and finally combining each row's slope into a singular steering line.

To begin, we will examine frame 500 from our straight row data, start to finish. The original data can be seen in the "Frame 500" graph. First, we decided that y-values were irrelevant to grouping the points, as our data points are mostly stacked on top of each other.

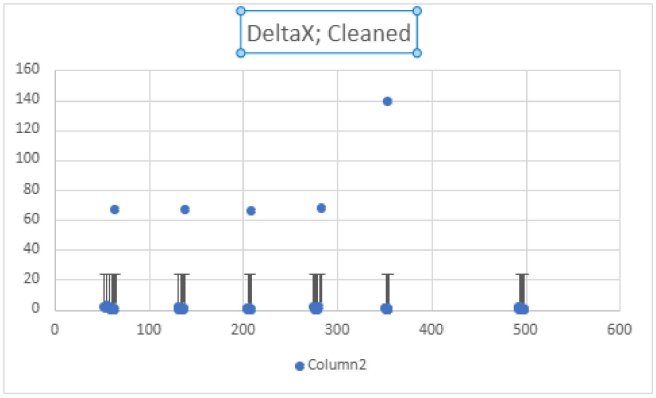


We noticed that there was a big jump in the x-values between rows, and that would likely be enough to accomplish the grouping.

We sorted all data points by x-values, and this new column was called our "DeltaX." The way this DeltaX (change in x-value) worked was as follows: {1,1,2,1,2,2,53,3,2,1,1}. From this, it could be told where the row ends, since there is such a large jump in value. This method is shown in the graph labeled "DeltaX with 1 SD", with the DeltaX acting as a y-value on the graph. Initially, we wanted to use a standard deviation from the trendline to figure out the cutoff point for outliers. For this frame, there are five rows, which means we should expect five large jumps: rows 1-2, 2-3, 3-4, 4-5, and 5-6. This standard deviation method was where we ran into our first issue. As you can see in the graph, there are actually more row jumps than expected for this frame. This happened in other frames as well, leading to

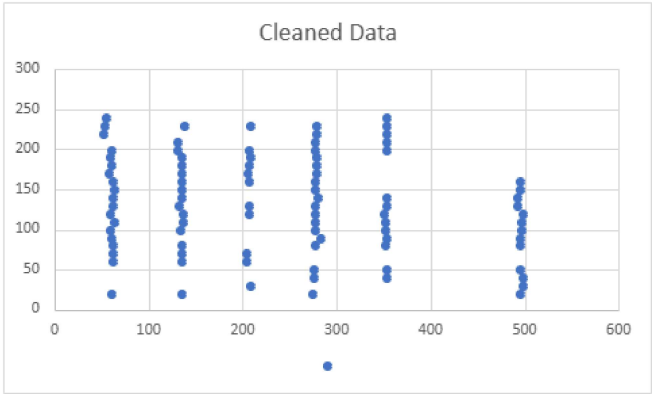


inaccurate grouping. This can be seen visually in the “frame 500” graph, where there are some points in-between two rows that cannot be easily grouped by eye or computer.



To accomplish this, we created an idea referred to as the Chemotherapy method, which uses the same standard deviation aspect. We knew from the beginning we could throw away a decent amount of data, since we only need two points to mathematically create a linear trendline. The Chemotherapy method utilizes this plot to throw away every value over one standard deviation. In this scenario, we will throw away six points. Some of those points are legitimate points, and some of them are false/imperfect points, like the ones between rows. We can throw away both point types because we know that there are way more good points than there are imperfections. After these points were deleted, this entire method was run again. In total, we ran Chemotherapy four times in the completed data, removing around 20 points on average. This removes all the outliers and leaves only the good data left. The cleaned data can be seen in the graph “DeltaX, cleaned.”

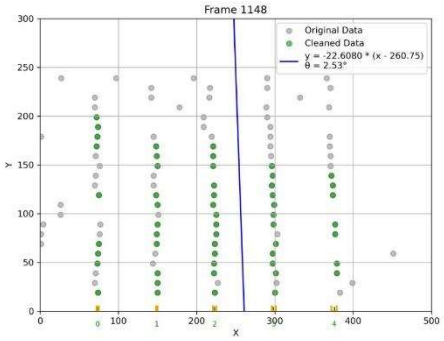
Six straight lines are present within the “Cleaned Data” graph, meaning Chemotherapy was a success. The cleaned DeltaX plot indicated the rows mathematically with four large outlier values (row jumps). With this, we’ve solved the first half of the problem.



Once we have successfully separated the data points into several rows, we need to combine these row slopes into one trendline for the vehicle to steer on. Originally, we took an

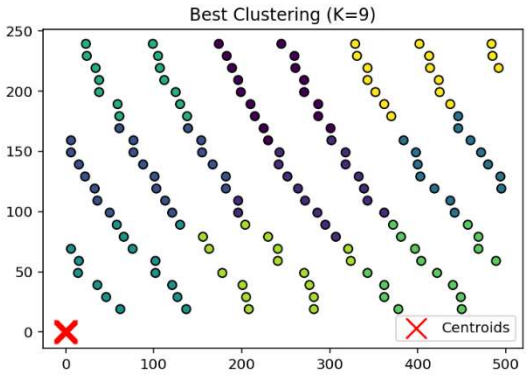
average slope of all the rows. For instance, if there are six rows, we add the slopes of each row and divide it by six. We also based the starting point for this steering row off the average of the six x-intercepts. However, we noticed there would be a problem if one of the rows had an infinite slope (straight vertical line) because we cannot mathematically divide by 0 to get the slope into our program. Moreover, if one of our slopes is computed as infinity, that would make the slope of the steering line infinity, making it inaccurate. Because of this, we decided to transpose the data, so our infinite slopes will change into a slope of 0, which is simple to compute and will not throw off the average. After this, we got a singular slope for our steering line. For placing the line on the frame, we took the average of all six x-intercepts.

To test accuracy, we decided to attempt another method where we found the slope of each line by averaging the start and end points. After this, we ran the same method to create the steering line; however, instead of averaging the x-intercepts, we found the midpoint between each set of x-intercepts and picked the closest to x=250. This method had a more accurate x-value (located relative to the middle of the frame), but the other method had a more accurate slope. Because of this, we combined the best sloping and positioning methods in the final result.

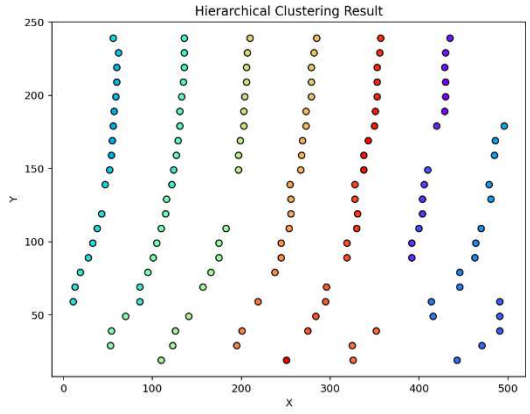


Here is an example final frame for the straight data. The gray points were deleted due to chemotherapy, and the green points are the ones utilized to format the trendlines. As seen, the blue guidance line is in the middle of two rows, as expected, with a directionality matching the slightly left nature of the frame.

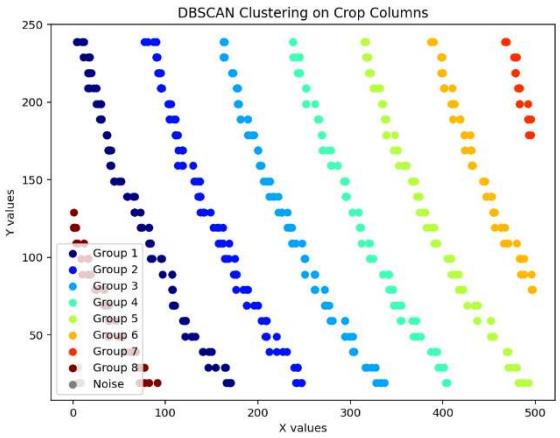
Once our straight rows have been successfully grouped and steered, we moved onto our curved row data. Since curves can become extreme at certain frames, the rows overlap vertically, meaning that we cannot use our DeltaX method from straight rows. Because of this, we decided to attempt three commonly known grouping algorithms: K-means clustering, DBSCAN, and Hierarchical clustering. K-means groups data based on centroids, which means it groups data radially (Pedregosa et al., 2011). Because of this, the algorithm does not work well on our linear data. Moreover, K-means needs a hard-coded row number, which would defeat the point of making our program automatic.



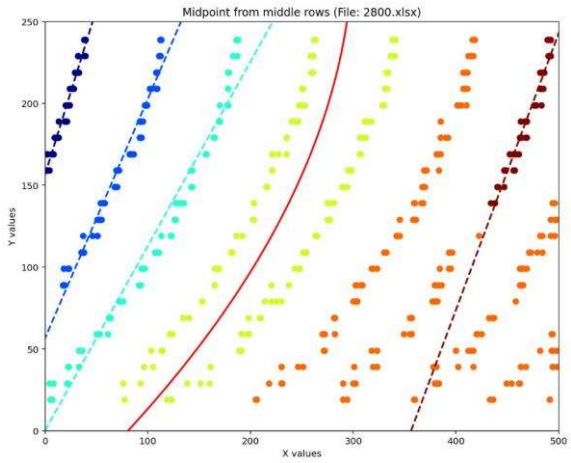
As for DBSCAN, which groups based on density, it worked well for most frames (Pedregosa et al., 2011). However, since the curved rows are placed so close to one another, it is often difficult for the algorithm to pick up these voids of density so the data can be sectioned off. We then tried our final algorithm of Hierarchical clustering (Virtanen et al., 2020). This algorithm sections data by creating a bracket that sections off as a large jump in data is detected. Because of this computation, the row number is not hard-coded, which is always positive. This algorithm worked very well on most of our data; however, the algorithm had trouble when the width between rows fluctuates within the frame. For example, if there is a greater distance between rows two and three at the top of the frame than the bottom, the bracket hierarchical clustering creates has trouble readjusting the current bracket width.



Because of all these issues, we realized we had to manipulate our data in some way (like chemotherapy for straights) so these commonly known algorithms could work. Since the vehicle moves at around thirty frames per second, we deemed it acceptable to overlap a frame with its previous three frames. This enhances the density of each row by combining more points into the cluster. Because of this, DBSCAN is now able to pick up on these drastic changes in density between rows, meaning we will stick to DBSCAN for the rest of this project. It is important to recognize that DBSCAN does not work perfectly, so we must take this into account when creating trendlines.



With rows clustered by our algorithms, we again move onto steering. There are two main differences here, the polynomials needed to fit, and the overlay of data meaning a lot more points to fit to a trendline. A consequence of our original data is that, due to the image recognition technique, all data is y=10 spaced evenly, so in our exact case, points lie on y= {2,12,22…242}. Following the discussion of how the trendline is made, it will be discussed how it is picked. With this overlayed data, we often have three or four x values on a given y= 10n line for a fit. For the neighboring fit, the same will be true. Thus, we utilized a midpoint method, where we plot the midpoints between the two at a given x value. For example, if we had line one with {110,113, 115} and line two with {191,193} at y=122, we'd plot the midpoint of both, ultimately getting a midpoint between 113 and 192 at (152.5,122). Then we go up by ten and repeat this process for the rest of the frame. This creates a new plot, and we take a trendline for this after all y-values have been accounted for. Given our increase in data, this method works very accurately.



In addition, this program must work even when the clustering doesn't. As seen in the figure, we have two clusters, yellow and orange, that do not work. To solve this, we applied a convexity test, which tests for ensuring the second derivative is a constant sign. In cases where it's not, it's often an inaccurate trend line, which happened for our yellow and orange trend lines. The convexity test has a small tolerance of around 20% but will still easily remove inaccurate trendlines like the ones created by the yellow and orange clusters. With the bad clusters removed, we only need two remaining good clusters to create a fit. We choose the two clusters with the most data remaining, and by data, that means the greatest distribution in y values. We wouldn't want to take a fit between the two most left lines (dark and light blue), as there are only a small number of y-values for the dark blue; in fact, there is such little data it looks like a straight. Because of this, we fall back to the light blue and teal fits to take a fit. We apply the midpoint method previously used on straights and then utilize another threshold to determine if it should be centered. This threshold calculates the average x-value of the fit over y=0 to y=250 and determines if it should shift based on if it's already close to 250. In this case, it's close to around 50, between the blue and teal lines, so it shifts. However, it does shift the trendline purely to x=250, which doesn't necessitate that it will follow our previous rule of not intersecting plants. Nevertheless, this is acceptable for these bad case scenarios, because it is still able to create some level of guidance.

Results

As this project comes to a close, it is important to recognize successes, unsolved issues, and futures steps. Firstly, we were able to transform all our methods from straight and curved data fully into python programs with the help of ChatGPT (OpenAI, 2024). This means that our project can be run at a very large level, and our methods no longer need to be done by hand in a frame-by-frame fashion. However, the straight and curved algorithms are currently separate programs, as both datasets use nearly opposite methods for grouping. Specifically, straights use the deletion of data via Chemotherapy and curves add more data via overlapping frames. We are certain that a combination of these programs is possible, as our DBSCAN methods work on our straight data. The main problem comes when we create our trendlines, as the straight data's y-values are not evenly spaced, meaning we can no longer create a new trendline by using the midpoint method. Furthermore, a polynomial fit will add unwanted curvature to straight rows.

Along with this, headlands were untouched, but this was expected due to the structure of this project with its time constraints. This project focused heavily on initially grouping data and finding slopes, then building upon that for the large variability within curved data. Because of this, headlands are a step that is expected to be completed once this project is passed onto a new group of students. Similarly, now that our original methods have been implemented into python, most of the project's future time can be spent brainstorming rather than iterating through frames by hand.

We would like to thank Pandas (McKinney, 2010) for creating our graphs and NumPy (Harris et al., 2020) for allowing our excel files to compute mathematical functions. Lastly, we want to express our gratitude to our mentors, scientists Josue Calderon and Apurba Das for their guidance, support, and encouragement throughout the duration of this project. Their insights and feedback were crucial in shaping the direction of this work. We are deeply appreciative of the time and effort

they dedicated to helping us succeed and gain experience in real-world mathematical applications.

References

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., … & Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

McKinney, W. (2010). Data structures for statistical computing in Python. *Proceedings of the 9th Python in Science Conference*, 51–56. https://doi.org/10.25080/Majora-92bf1922-00a

OpenAI. (2024). *ChatGPT* (Apr 25 version) [Large language model]. https://chat.openai.com/

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … & Duchesnay, É. (2011). *Scikit-learn: Machine learning in Python*. https://scikit-learn.org/stable/modules/clustering.html

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., … & van der Walt, S. J. (2020). *SciPy v1.0: Fundamental algorithms for scientific computing in Python*. https://docs.scipy.org/doc/scipy/reference/cluster.html