Self Steering Car Algorithm - Final Documentation

Joshua Falck, Joey Karlsson, Mohamad Khalil, Joel Mattsson
University of Gothenburg

DIT639 - Cyber Physical Systems and Systems of Systems

Christian Berger

30th May 2024

2. Project Organisation and Planning

The problem that we were given to solve in this course was to create an algorithm that could take in an OpenDLV input of a remote control car being driven, and create an output of a ground steering request, a number between the range of -0.3 and 0.3 that would represent how sheer the turn should be.

Milestone Task Completion Chart

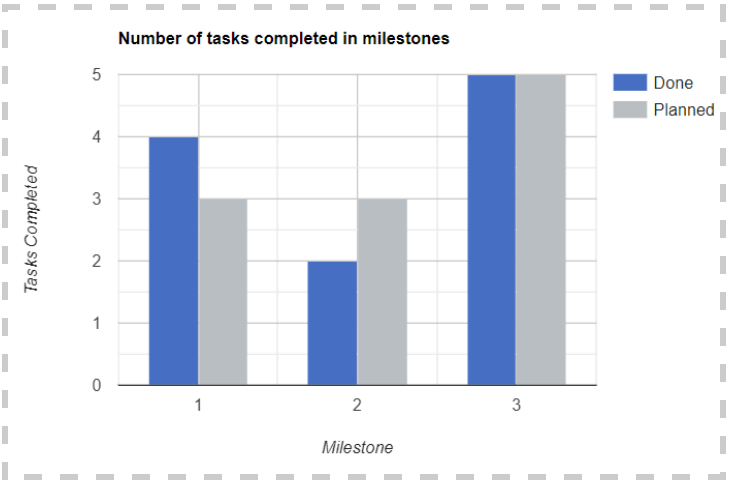


Fig 1. Chart showing the number of tasks planned and done in each milestone

The chart above shows the planned activities for each milestone and the completion rate of the planned activities. In terms of planning, we decided to split it up so that we had three issues to complete in milestones 1 and 2, and five in milestone 3. Once the project implementation had officially begun, we worked hard in milestone 1 and finished the planned tasks early, allowing us to complete a fourth issue that was taken in milestone 2. This meant that we could work a little slower during milestone 2, and in milestone 3, we managed to hit all the goals that we had planned out for ourselves.

Individual Contributions

The broad scope of this project entailed a large problem space, in which the group divided the work accordingly:

Joey: Algorithm implementation, OpenCV cone detection, and production pipeline

Josh: Hsv Detection & filtering, yolov8 detection, CSV output, and pipeline

Mohamad: Linear regression analysis, adjust steering angle formula, add testing debug window, determine the direction the car is heading in.

Joel: Trigonometric algorithm, yolov8 detection, support multiple architectures

3. Conceptual ideas of algorithmic aspects

Cone Detection

The first aspect of the algorithm involves detecting the cones on the sides of the road which is crucial for determining the correct steering angle. The first step is to filter the frame using predetermined HSV values corresponding to the colors of the cones (yellow/blue)

Then we crop out some of the regions that we are not interested in, e.g. the wires that are visible on the car and the top of the frame. After that, we perform generic morphology operations (dilations and erosion) to get rid of some more unwanted noise. Finally, we use OpenCV to detect the contours of the remaining objects in the frame, and after doing some basic checks such as constraining the objects to a certain width and shape (cone) we are left with what will most likely be a cone.

Linear Regression Algorithm[1]

After noticing that the car was providing periodic data, we decided to run statistical analyses using Jasp[2] to find potential correlations between the different values. It was apparent in our findings that the angular velocity as well as the infra-red readings had a significant effect on the value of the steering angle, this was clear to us after noticing a p-value lower than 0.05. As the velocity increased, the steering angle would increase. In addition, as the distance from the car to the cone increased, the steering angle also increased. These results pointed us in the right direction and allowed us to create the following formula:

Steering Angle = (abs(velocity) * 0.002) + (1 / (abs(distanceFromCone) * 10))

With this formula, we were able to go from 17% accuracy on the best video to 26% accuracy on the worst video and 32% accuracy on the best video.

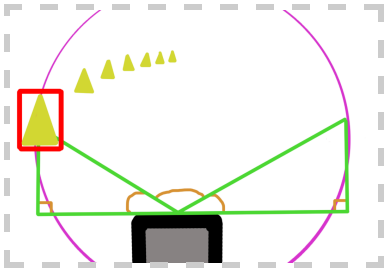
[1] https://www.youtube.com/watch?v=hGX3K6fhwp8
[2] https://jasp-stats.org/

The data is extracted on the go using the capabilities of the cluon library, the algorithm accepts the current angular velocity, and the infra-red readings as an input and produces the steering angle as an output.

The last part of the algorithm involves determining the direction in which the car is supposed to be traveling. In order to achieve this, the algorithm keeps track of which cone color was last seen on the left/right side of the road which allows it to compute the correct steering angle direction on every timestamp.

Trigonometric Algorithm

The group's initial algorithmic attempt was to use the positions of the cones to determine the diagonal and horizontal distance from the car's center. In turn, we could draw rectangles and resolve the associated angles[3] to ultimately decide how close a cone is, and hence, how intense the car had to steer. This would be done by using Pythagoras Theorem to calculate the length of the sides combined with tan inverse to resolve the angles. In the picture below, the car is driving in an intense curve, resulting in the blue cones being unnoticed:



In this case, the magnitude of the angle separating the yellow cone and the center of the car would define the degree to which the car would turn in the opposite direction. If the angle is bigger, it implies that the cone is further away, making the calculated ground steering value less significant. Conversely, a close cone would yield a small angle, but generate a steering value that reflects a strong curve. This is implemented by working with the angles within a range from 0 to 180, where 0 is equivalent to a sharp 90-degree turn to the left, 180 is a 90-degree turn to the right, and 90 is the central angle, indicating that the car is driving in a straight line[4]. In the implementation, we would either subtract

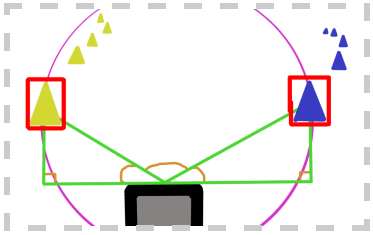
[3] https://en.wikipedia.org/wiki/Tan-1
[4] https://www.youtube.com/watch?v=V5ArB_GFGYQ

or add the calculated angle from the central angle, depending on whether a left or right cone was detected However, in case of slighter turns where cones at both sides are present, we needed to take into account for the degree of each angle and establish a mathematical algebraic relationship that gives precedence to the closest cone, yet still lets the cone in the back have an impact on the outputted steering angle. We expressed this relationship by taking the average of the absolute values of each angle from the central angle 90:

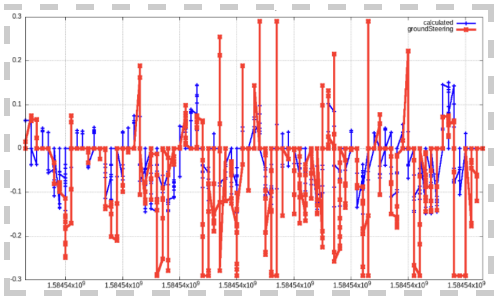
$$d1 = abs(90 - v1),\ d2 = abs(90 - v2)$$

$$result = (d1 + d2) / 2$$

This concept is exemplified in the figure below, where both cones are equally close to the center of the car, causing the two corresponding angles to "eliminate" each other such that the output angle becomes 90, which is then mapped to the final value of 0:



Once we reached the disappointing result of only achieving an accuracy of 12%, the graph was plotted such that the blue line is our output compared to the actual ground steering:

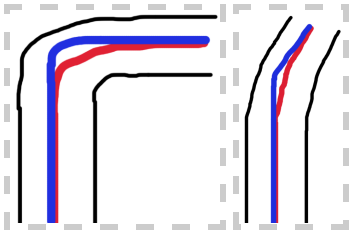


When we observed the blue line, it appeared as if it turned more on the easier curves but failed to match the magnitude of the expected intensity in the harder curves. Due to this, we developed the hypothesis that the algorithm worked as intended, in the sense that if it would be applied on the car it would stay within the road and avoid crashing. We reasoned that if the sum of the generated ground-steering values would be equal, meaning that both algorithms make the car turn the same number of degrees in the same video, then our algorithm must work as intended, despite the disappointing accuracy:

$$calculatedGroundSteeringSum * X = correctGroundSteeringSum$$

$$X = 0.96$$

Keep in mind that a human drives the car in the video. This would undoubtedly entail a higher inaccuracy, as opposed to running a simulation with the car on the implemented algorithm. For instance, if the human drives too deep in a curve, our algorithm encounters a case it never should encounter (because it would turn significantly earlier) with respect to the positions and angles to the detected cones. Below follows a visualization of how our algorithm differs from the expected value, where the red line represents the calculated value:



The question that remains now is: Now when we have mathematical proof that the algorithm is capable of steering the car on the cone road without crashing, why did it only reach an accuracy of 12%? First of all, as mentioned above the fact that the car is running on a human-driven video exposes it in scenarios it never would reach if our algorithm was applied, resulting in erroneous cone-angles (input) and thus inaccurate ground steering (output). The second contributing factor is concerned with the requirement of staying within 25% of the expected output. To put this in perspective, let's observe the numerical constraints in two concrete cases:

Current car angle: 90 degrees (groundSteering = 0)

- **Angular margins: 90 * 0.25 = 22.5**
- ***Maximal allowed angle: 90 + 22.5 = 112.5***
- ***Minimal allowed angle: 90 - 22.5 = 67.5***

Current car angle = 45 degrees (groundSteering = -0.15)

- **Angular margins = 45 * 0.25 = 11.25**
- ***Maximal allowed angle: 45 + 11.25 = 56.25***
- ***Minimal allowed angle:45 - 22.5 = 33.75***

The two cases above convey that the difference is significant yet not too distant for it to be dismissed when the algorithm operates with erroneous input (a fixed video without a real-time simulation of the applied algorithm's performance). In other words, when the two factors are coupled together, the 25% requirement doesn't make up for the fact that the input video is pre-recorded. Ultimately, we concluded that applying this trigonometric algorithm to the car would correctly drive it as in avoiding crashing into cones. We also acknowledged that this approach wouldn't yield a satisfactory result with respect to the set-out requirements of this course as we moved on to the next algorithmic attempt, linear regression.

4. LLM Discussion

General LLM Discussion

LLMs are here to stay whether the engineers like them or loathe them, the real endgame is to embrace them as it could potentially lead to faster progress in solving user stories/requirements as it's decently good at doing the grunt work, such as unit tests while being inexpensive compared to hiring a junior developer. By letting the engineers focus more on complex and creative intensive problem solving, it can potentially improve productivity and innovation. As technology improves by leaps and bounds, so too does the frontline move forward. It has been proven time and time again that technological advancement is not linear, but instead exponential, and this is only truer for the newest technologies. Although Large Language Models have a long way to go, applications such as ChatGPT have made strides in improvements, going from a simple text completer in 2022, to a full-blown, large-scale, corporate AI, already used in a variety of use cases in industry, as well as generally by over millions of regular customers. Many leading minds in software engineering believe that Large Language Models will revolutionize programming and software development, and for good reason since we can already see the way that it has affected how humans interact with machines to produce software products. We have already seen widespread use of language models in programming with the introduction of the Github copilot, and even some initiatives threatening to make programmers obsolete like Devin. All in all, there are many positive things to say about the use of AI in programming such as the ease of getting simple snippets of code for common problems, querying pre-existing codebases, and explaining key programming concepts, however, it is still clear to see that AI isn't at the stage to replace humans any time soon, with it missing many faculties necessary to make

fully-fledged programming such as logical reasoning and deduction, time and resource management, abstraction and specialization, and agency.

Similar to the Industrial Revolution, there was once a time when humans had responsibility for tedious work and had to devote their time, resources, and effort 8 hours a day in a factory to put together physical products with their hands. The problem is not only that it was costly for the owner to provide monetary resources for the employees, but that the employees' imperfect work was unmatched by the idea of robotic automated perfection. On the other hand, this was an opportunity for many individuals to provide resources for their families at the brink of survival. In today's world, this would be equivalent to the argument of: When allowing AI to replace previously human-oriented work, then what would happen to those losing their profession? Would we still be able to provide a job to everyone such that society is able to operate and survive? If AI completely takes over society, then what's left for us humans to provide for society? How would we make a living?

Well, history doesn't repeat itself but it rhymes. Although we can't be entirely sure what the future holds, we do know that, in retrospect, the Industrial Revolution liberated us from tedious work. This allowed us to direct focus and creativity on pursuits of higher importance, one of which was software development. Now, when looking back at the last decades and the significant technological progress humans have made, it's safe to say that if we were still tied to the tedious professions that were replaced in the Industrial Revolution, many engineers today wouldn't be able to spend their time on the important creative tasks of innovating that have the possibility of changing the course of the advancement of technology for the better.

Our own perspective on LLMs

In our project we did not incorporate LLM-generated code for our solution, the reason we attempted to use copilot was to help us achieve an algorithm that would generate hallucinated solutions that looked pretty to the naked eye but turned out to be broken and unusable. There could be multiple factors for this such as bad prompting, lack of context, bias in its training data, and more.

Although the LLM could not help us generate a solution, we did use GitHub Copilot as it was really good at explaining code that was used in understanding the cluon library and the given template. With that being said, it is crucial to fact-check the output that is provided by an LLM. This is due to the fact that LLMs can produce hallucinated material which leads to confusion among the developers as well as inefficient time utilisation. In addition, LLMs are usually unaware of newer syntax and technologies since they're trained on outdated data, this could cause some of the output to not work with the latest versions of necessary dependencies, this problem was made clear to us when we tried to generate a CMAKE file for a fairly recent version which was not compatible with the default version which the LLM considered.

5. Algorithmic Details References

1. Simple linear regression: https://www.youtube.com/watch?v=hGX3K6fhwp8 (Retrieved 2024-04-28)
2. Tan inverse: https://en.wikipedia.org/wiki/Tan-1 (Retrieved 2024-04-24)
3. Unit circle: https://www.youtube.com/watch?v=V5ArB_GFGYQ (Retrieved 2024-04-26)

6. Retrospective

What went well:

Writing tests for invariants such as the angle accuracy requirement before the implementation of the algorithm turned out to be a great decision. We were able to get instant feedback on the effects of our changes which ultimately allowed us to improve upon our best score at each sprint. In addition, the tests gave us insights into how well our current algorithm was performing. For example, if a change caused the accuracy to go down, we investigate the causes of the drop and try to pinpoint what exactly was causing the degradation.

Setting up our continuous integration and deployment pipeline was also a very important and useful decision. This allowed us to revert with ease to an older working version of the product in the case of crashes or degradations in performance. In addition, it gave us useful overviews of how the current state of the repo is performing (via automated tests, reports, etc..)

What didn't go well:

Initially, we started to experiment with different ways to detect cones besides the HSV which was the common approach. We used YOLOv8's machine-learning algorithm to detect the cones which went well until we came to the point of connecting it to the shared memory. We had successfully taught the program to detect each cone from an mp4 video but as mentioned connecting it to the shared memory turned out to be a bigger hurdle than anticipated. In the second presentation, Christian informed us that there was an existing implementation of YOLO that we did not know about before, but at that time the HSV detection had already been done and YOLOv8 continued to be archived.

Starting with the implementation without addressing scenarios that can affect the solution caused us to slow down. Certain aspects like assuming that blue cones are always to the left and not knowing if a negative angle corresponds to a clockwise or counter-clockwise turn.

7. Appendix of received peer reviews

Group 14

Covered Well

All Slides - Pictures, videos, and graphs are well made and very useful for a clear understanding of what they are explaining. This is especially true for the slides detailing the development of the steering algorithm.

Slides 2 to 12 - Topics were explained in detail and at length. Regarding the algorithm, slowly adding together its parts made it easier to comprehend and get a grasp on. This applies to the detailed explanation of the tools that were used to make the algorithm as well - for example the pythagorean theorem.

Further Clarity

Slide 13 to 15 - Although the usage of LLM was mentioned, it was done in a brief and general way, not explaining specifically where in the project these tools were used. Looking at the video we can assume that GitHub Copilot was used for the development, but we do not get a deeper explanation on the concrete use cases that this tool had.

General comment - It was hard to follow along due to the microphone cutting out, this was the case particularly for some of the slides. For these cases, we recommend adding additional clarification in the report to what was discussed so nothing goes missing.

Group 16:

2-3 aspects covered well:

- The fundamental functionality for color detection i.e HSV masking and usage of contours along with the process of cone identification (division of screen, detecting cones on left and right) was well covered and could be understood easily.
- The concept of calculation of angles is demonstrated in a well mannered way with an example to make it easier to grasp, making the entire thing easier to understand.
- The process of noise removal along with the step by step explanation for the same is explained clearly. Including images which show each process, i.e the transformation of the image after erosion and dilation for noise removal process, helped understand the process used by your group.

2-3 aspects require more clarity:

- It would be more clarifying if you elaborate more on the strategy used in the counterclockwise steering. The current explanation for the strategy seems to be briefly glancing over the strategy. For the report it would be a good addition to include more information on the strategy and also include the current usage of this strategy.

- Another point to mention in the report is some elaboration on what Random Forest Regression is. Since its usage was simply stated in the video, it would be a good idea to include more detail on this regression method and some evidence which shows the improvement.

8. Addressing of Peer Reviews:

Concrete LLM examples:

> The use of LLMs is addressed in more detail in the LLM section, we mention that while we didn't use LLMs to generate code for the solution, we did use GitHub Copilot for tasks such as understanding the general syntax of libraries such as cluon.

Include clarifications for bad audio slides:

> We tried to include clarifications for the slides that had bad audio (LLMs, algorithms). This is hard to address in the report however it will be taken into regard in future recordings.

Elaborate on counter-clockwise steering:

This aspect of the algorithm is discussed in more detail in the last part of the section titled *Linear Regression Algorithm*. In short, the algorithm always knows which color cone corresponds to the left or right side of the road.

Elaborate on the random forest regression:

We were a bit confused about your comment about random forest regression. It is nothing we have, maybe due to the poor audio quality you heard something else if we assume that you meant linear regression in general, that has been addressed in the section titled *Linear regression algorithm*.

9. Appendix of LLM usage

As mentioned in the LLM section, we did not use LLM to produce any code. However, we did use GitHub copilot to give further clarifications regarding the given template and its dependencies.

Concrete use cases:

- Cluon library
- Template-opencv.cpp