Pacer Drone

|  |  |  |
| --- | --- | --- |
| Evan Arthur *Colby College* Waterville, USA edarth21@colby.edu | Eli Decker *Colby College* Waterville, USA edecke20@colby.edu | Tate Ewing *Colby College* Waterville, USA ctewin21@colby.edu |

*Abstract*— An algorithm for visual lane detection on tracks to be used for pacing track runners without relying on GPS or other location tools. Using a Tello drone and the processing power of a laptop connected through Wi-Fi, the algorithm takes raw video from the drone and detects the lanes to be used for guidance around the track while pacing an athlete.

Keywords—drone, computer vision, track, pacing, lane detection

# Introduction

In Spring 2017, Nike-sponsored athletes attempted to break two hours in the marathon through the pursuit of marginal gains that were hoped to push the runners under the elusive 2-hour barrier. An important innovation made was the implementation of a pace car that emits laser lines which allows the runner to maintain perfect pace. No longer would the runners have to constantly check their watch to make sure they are at the correct pace, breaking up their stride and slowing them down in the process. Instead, the mental challenge of maintaining pace is eliminated, allowing the runner to simply follow a line on the ground. Our goal was to replicate this experience on a running track. Because it is impossible to use a pacer car on a track, we decided the most effective option was to create a drone capable of flying around a track at a constant speed while carrying a laser emitting a pace line on the track behind it. We chose to use the DJI Tello drone because of its affordable price and easy to use SDK for sending commands and receiving the video stream.

# Related Work

Because a track lane is defined by two painted lines, our work was inspired by the much more common task of car lane detection. A car lane detection system attempts to identify the two road lanes on either side of it in order to stay in between them. This problem is well documented and has been implemented in commercial vehicles available to the general public. Because no one to our knowledge has attempted to make a drone fly around a track, we were not able to find any directly related work. We looked specifically at Kemal’s curved lane detection system on Hackster.io designed for cars. He used a sliding window algorithm to detect the lane and subsequently give a curve radius that can be used to alter the car's path. Though designed for a different purpose, the problems are similar enough that his method was transferable from a car on a road to a drone on a track.

# Methods

## Our project can be divided into three main problems:

* Connecting to and sending directions to the drone.
* Taking the drones video stream and computing where the drone should go.
* Maintaining a constant pace throughout.

Because we chose to use a DJI Tello, connecting and sending directions to the drone was handled by connecting our computer to the Wi-Fi signal emitted by the drone. Combined with the SDK provided by DJI, we were able, in theory, to send commands to the drone and receive the drones video stream in almost real time.

The DJI Tello drone has a built in Wi-Fi connection, and the SDK allows the user to control the drone over Wi-Fi and receive the video stream from the drone’s front-facing camera. In theory, we should easily be able to access the video stream, process the video to determine controls for the drone, and send the instructions to the drone.

Our first step was to configure a lane detection algorithm to determine the radius of the track’s curve so we could control the drone. After receiving the video stream, we put each frame through a pipeline in order to identify the track lines and the accompanying curve radius in order to send the correct flight commands to the drone. Because our computer will potentially not be able to handle each frame and maintain real-time, we implemented a threaded camera that allows us to always be looking at the most recent frame:

* First, we resize and recolor the video.

A picture containing outdoor, grass, red, player

Description automatically generated A picture containing outdoor, grass, green, court

Description automatically generated

Fig. 1. Showing the original image on the left and the recolored image on the right.

* Then we un-distort the image based off of a collection of photos taken by the drone of a chess board so that we know how the camera distorts a known object.
* We apply a perspective warp so that what is directly in front of us is mapped so that it appears as if we are looking down on the road without perspective.

A picture containing clock

Description automatically generatedA picture containing clock

Description automatically generatedA picture containing black, clock, white

Description automatically generated

Fig. 2. Fine-tuning the values of a perspective warp to achieve a top-down perspective for better mapping. The bottom row shows the output of the optimum perspective warp.

* We apply a Sobel filter that turns all pixels that are below our threshold black and all ones above the threshold white, leaving us with an image that is all black instead of white lines. Obtaining these thresholds was a process of trying a variety of values and keeping the most effective ones that captured the whole line, but no extra noise.
* We implement a sliding window algorithm to identify the lines and fit a curve to mimic the line. (describe the sliding window)
* We calculate the curve radius based off of the curve found by the sliding window algorithm
* We translate the curve radius into a yaw value to fly the drone at and send it to the drone

As driverless cars have become a hot area for research, we were able to leverage simple lane-finding algorithmic techniques used in self-driving cars. Luckily, a track is even more of a closed system than a road so there are some built-in assumptions we could make when analyzing our video stream.

By flying at a similar height, the whole time, the relative size of the lane lines will remain fairly constant, which means we can alter the perspective of the camera to make an overhead view centered on the inside lane lines. Also, tracks always have painted lane lines for humans to easily see, which means we could do Sobel filtering on the video stream to help separate the lines from the track surface and turn it into black and white (binary) for easy processing. Before a Sobel filter, we used a recoloring

After filtering, we used a sliding window algorithm. Each window takes the average of where the white pixels are to come up with an approximation for where the line in the window must be (after Sobel filtering the line is in white, the track surface is in black). The size of the windows was fine-tuned to maximize detection of lanes when curving while balancing the tradeoff of minimizing interference such as number markings on the track. We determined the size of the windows by examining the minimum number of windows required to detect the lane pixels without getting distracted by noise. The stack of sliding windows then use their combined averages to approximate the radius of the curve based on the centering of white pixels in each window.

A screenshot of a cell phone

Description automatically generated

Fig. 3. The stages of processing the original raw video (furthest left), then applying Sobel filter with proper thresholding values, then the sliding windows algorithm. All units in pixels.

A screen shot of a social media post

Description automatically generated

Fig. 4. Fine-tuning of the threshold values showing not enough sensitivity and missing details of the lanes. Units in pixels.

A picture containing object, clock, black, meter

Description automatically generated

Fig. 5. Fine-tuning of the threshold values showing too much sensitivity and extraneous noise hampering the detection of lanes. Units in pixels.

Once we had the algorithm dialed in, it was time to address how to control the drone with the curve radius. Although the SDK advertises various methods and features that would perfectly suit the task, it was not as simple as anticipated. The communication between the drone and computer presented some frustrating challenges.

The DJI Tello accepts a variety of commands, which it receives over a UDP connection in JSON format. Writing code for the UDP connection and parameterized functions to easily test different values for controlling the drone would be a tedious process. Fortunately, we were able to leverage the djitellopy library (<https://github.com/damiafuentes/DJITelloPy>), which establishes the socket connection and formats the drone commands with convenient, object-oriented design.

For the most basic commands, the djitellopy worked without hiccup. We were able to command the drone to move forward, backward, side-to-side, and yaw. When attempting to use some of the more advanced methods, however, we encountered some issues. For example, listed in the Tello SDK Version 1.3.0 is the go\_x\_y\_z\_speed method, which commands the drone to fly in a curve, passing through the points (x1, y1, z1) and (x2, y2, z2) and a given speed in cm/s. This method would be especially useful for our task, because it is easy to calculate two points on a curve given the radius. Even after much experimentation the Tello drone responded with an error message every time.

Instead of using go\_x\_y\_z\_speed, we turned to send\_rc\_control, which allows us to control forward and backward speed, side-to-side speed, and yaw rate, allegedly in cm/s. We found that the actual speed of the drone was not consistent with the indicated cm/s unit. This inconsistency made it extremely difficult to control the drone based on a curve radius given in meters. After extensive trial and error, we determined a range of values for forward movement and yaw rate that fly the drone in an appropriate curve.

Given the periodic inconsistency of the algorithm’s detected curve radius and the speed/yaw units, the best solution is to make the drone fly in a curve when the radius is between 250 - 350 meters, and otherwise fly straight. When the algorithm accurately detects a curve, the radius is somewhere in this range. Most tracks have a similar curve radius, so it should work for most applications.

# Results

We were not able to measure how accurate the drone follows the lane on an actual track, per the technical limitations discussed in the methods, but we were able to gauge the accuracy of our lane detection algorithm--to some degree. Our algorithm calculates the curve radius of the lines detected, so to measure how accurate these calculations are we compared our results with the standard track curve radii. See below the graph of the algorithm's output.

A screenshot of a cell phone

Description automatically generated

Fig. 6. Graph showing output of detected curve radius from drone video over a distance of 400 meters (track is symmetrical, so the calculations were performed on a 200 meter segment and then doubled to complete the 400 meters).



Fig. 7. The detected curve radius bears little resemblance in shape to the standard curve radius, other than the straight parts of the track, and the scale of the measurements is very inaccurate.





Fig. 8 and Fig. 9. In the output of the lane detection algorithm, both the straightaways and the curved sections of the track, the curves appear to be tracked accurately. This leads us to believe that there is problem with the conversion from calculated pixels to meters.

##### Conclusion and Further Work

While some of the results were somewhat inconclusive, the research showed that this is a promising area of further exploration. One possible source of error may lie in not correcting the effects of the perspective warp on the final output curve radius. The algorithm output shows a clear difference in the straight and the curved sections of the track, however the

measurements in the curved section appear to be inaccurate, so there is room for further refinement there.

In many places in the world, network signals are unreliable or inaccurate and more stable computer vision techniques for autonomous drone movement provide a possible solution. The curve-mapping for drone use and computer vision techniques in this paper could be expanded upon to come up with more robust systems by utilizing neural networks for recognizing lane lines in varied lighting and weather conditions. Additionally, recognizing and utilizing other markers on the track such as start lines could be used for pace correction without requiring a GPS connection.

##### Acknowledgment

Thanks to Professor Ying Li for her guidance throughout the project.

##### References

Kemfic, “Curved lane detection,” HacksterIO, May 24, 2018, <https://www.hackster.io/kemfic/curved-lane-detection-34f771>

D. F. Escoté, “DJITello python library,” Github Repository, 2020,

<https://github.com/damiafuentes/DJITelloPy>