Classical Path Planning

Code Documentation

The Input

- **Car position**: Where the car is right now (X, Y coordinates and yaw)
- **Cone positions**: A list of cones with their locations and colors
    - Blue cones (color = 1) => LEFT side of track
    - Yellow cones (color = 0) => RIGHT side of track

The Output

- **A path**: A list of 25 points that the car should follow
- Each point is (X, Y) coordinates in meters

Part 1 (Handling 1-2 Cones per Side)

The concept is to stay in the middle of the blue and yellow cones

#Step-by-Step:

Step 1: Filter Cones (Which Ones Matter?)

We need to figure out which cones are relevant through the _filter_and_sort_cones function.

Leeh bna3mel kda?

-Too close cones: If a cone is right next to the car (< 0.5m), it's already behind us or we're passing it. Not needed for planning ahead.

-Behind us cones: We only care about cones ahead of the car. The math `forward_distance > -0.5` checks if the cone is generally in front.

-Sort by distance: Closest cones are most important - they tell us about the immediate track ahead.

Step 2: Separate by Color

Blue cones are ALWAYS on the LEFT and Yellow cones are ALWAYS on the right.

Step 3: Compute Waypoints

Now we create 2 "target points" (waypoints) that define where the path should go. The approach changes from case to case:

Case A: Both Sides Have Cones

Scenario 1: Two cones on each side (2 blue, 2 yellow)

The algorithm creates 2 waypoints, one in the midpoint of the closest pair and the other in the midpoint of the second pair.

Scenario 2: Two blue, one yellow (Asymmetric)

The algorithm first calculates the midpoint between B2 and Y1 (this is the cross-pairing step).
It has already calculated the first waypoint at the midpoint between B1 and Y1.

Then, it does the following:

1. Draws a vector from Y1 to B1 which gives the side-to-side direction of the track.
2. Rotates that vector 90 degrees clockwise, to get the forward direction of the track.
3. From the first waypoint (W1), it moves 2.5 meters along that forward direction to get a projected point.
4. Finally, it blends the cross-pair midpoint (between B2 and Y1) and the projected point from W1, using a 70:30 ratio (70% cross-pair, 30% projection), to compute the second waypoint (W2).

Why blend?

- **Cross-pairing alone**: Might pull the path toward the old yellow cone (creates weird angles)
- **Projection alone**: Might not account for the actual track direction shown by B2
- **Blending**: Gets the best of both - follows the cones but stays smooth

Case B: Only One Side Has Cones

Scenario: Only blue cones visible (no yellow cones)

The algorithm assumes the track is 5 meters wide (half of line equals 2.5m), then it finds the direction the blue cones are going and then offsets the path by 2.5m to the RIGHT of the blue boundary

Case C: No Cones at All

The algorithm drives straight ahead. ☺

Part 2 (Handling 3 Cones on One Side)

With the part 1 algorithm we'd only consider B1 and B2, but B3 would be completely ignored.

The Solution: Find the "Best Line" Through Cones

The algorithm finds the best line through the cones. (it's like trying to draw the best line you can through three non-colinear points, that is what the algorithm does in a nutshell)

The Math Behind It: PCA (Principal Component Analysis)

A. Find the Average Position

First, find the center point of all three cones (by finding the mean of the X coord. of the 3 cones and the Y coord. of the 3 cones)

B. Calculate Spread (Variance)

The algorithm measures how spread out the cones are in X, Y and how X & Y relate (called cov_xx, cov_yy, and cov_xy respectively)

Example, When:

- cov_xx = 1.5: Cones are spread out 1.5m² in X direction
- cov_yy = 0.167: Cones are barely spread out in Y direction
- cov_xy = 0.5: When X increases, Y also tends to increase

Note: The direction with MORE spread is the track direction, so in our example x has the most spread so the path is mostly going in the x direction (left/right)

C. Find The Best-Fit Line Angle

The algorithm uses a formula (referenced mn chatgpt) to find the **angle of the best line** through the cones.

D. Check the Line's Direction (Forward vs. Backward)

Problem: The angle could point forward OR backward.

The algorithm compares the angle of the fitted line to the angle from the **first cone to the last cone** (B1 to B3). If the angles are more than 90° apart, it means the fitted line is pointing backward, so the algorithm **flips it 180°** to ensure the path always goes **forward**. (if cones go left→right, first-to-last vector points right, so our fitted line should also point right)

E. Create Waypoints Along the Fitted Line

The algorithm projects each cone onto the fitted line and then offsets each point by 2.5 m towards the right (we assumed that the path is 5m wide), so the path stays on track away from the

cones to their left (when blue)

Why We Chose This Solution (Detailed Reasoning)

Reason 1: Simplest Solution

Simplest Solution I could think of for 3 cones is drawing a line connecting the 3 cones and then projecting it onto the track.

Reason 2: PCA is Robust to Noise

Cone detections can be slightly off due to sensor noise or perception (kolo mn perception 3mooman) errors. With PCA, small errors in one cone don't drastically affect the path, because the algorithm considers all three cones together, reducing the impact of any single mistake (each cone only has 1/3 of the impact)

Reason 3: Mathematical Soundness

PCA is a proven technique used across robotics, vision, and data analysis for identifying dominant directions in noisy data. It is quite popular and gives an optimal, stable direction estimate and runs efficiently enough for real-time use.

Code Limitations

Limit 1: The algorithm assumes the Track is a Straight Line

Real race car tracks curve, but the best-fit line is just a straight line approximation and we cant really draw a line on a curved road that well.

Limit 2: The algorithm uses a Fixed Lane Width (2.5m)

We do not know the width of the actual track, it could be more or less or it could be changing.

Limit 3: The algorithm assumes all cones are correct

If perception (rabena y5aleeh) sees a cone that is not really there or it could completely miss a cone, the algorithm wont care, it will treat all input cones equally.

Limit 4: Only 3 cones are detected at most

Limit 5: The path is not the MOST optimal

The algorithm always tries its best to stay in the middle, but the middle isn't necessarily the shortest path in real world racing.