Pure Pursuit

Contents

Algorithm

In this section we want to control the front wheel angle δ, such that the vehicle follows a given path. This is known as **lateral vehicle control**. In the pure pursuit method a target point (TP) on the desired path is identified, which is a **look-ahead distance** l_d away from the vehicle. The angle δ is chosen such that the vehicle will reach the target point according to the kinematic bicycle model. The look-ahead distance is a parameter, and is typically chosen to depend on the speed v via $l_d = K_{dd} v$, where the constant K_{dd} needs to be tuned. We can also enforce a minimal and maximal look-ahead distance, so as to avoid undesirable behavior at very high and very low speeds.

Let us draw the bicycle model and a given path we should follow. We also draw a circle of radius l_d around the center of the rear wheel. The intersection of this circle with the path is our target point TP. According to the kinematic bicycle model, the vehicle will move along the orange arc, which is determined by the front wheel angle δ. We want to choose δ, such that the orange vehicle trajectory will move to the target point.

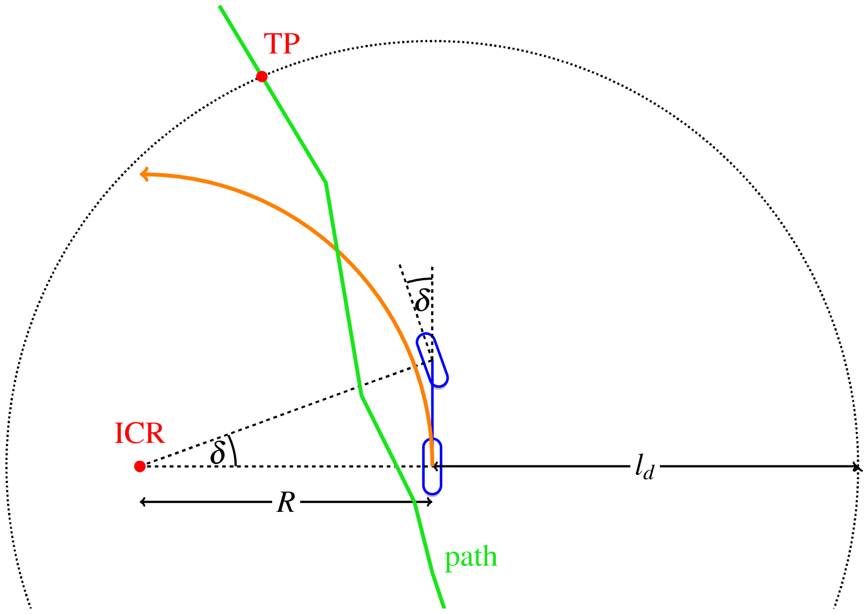
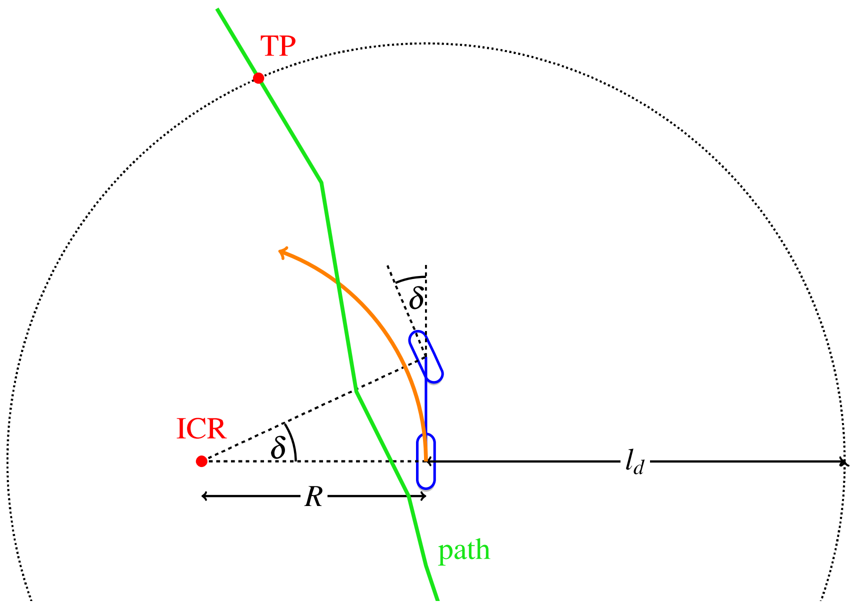


Fig. 27 Bicycle model should follow a path. With the current front wheel angle δ, it will not reach the target point TP.

Since the above drawing is generated programmatically (using tikz), we can change the value of δ until the vehicle trajectory goes through the target point:

δ = 25° δ = 20° δ = 15° δ = 11.3°



But there is a more elegant solution than just trying out a bunch of different δ values. We can actually compute the optimal δ based on the magenta triangle in the sketch below

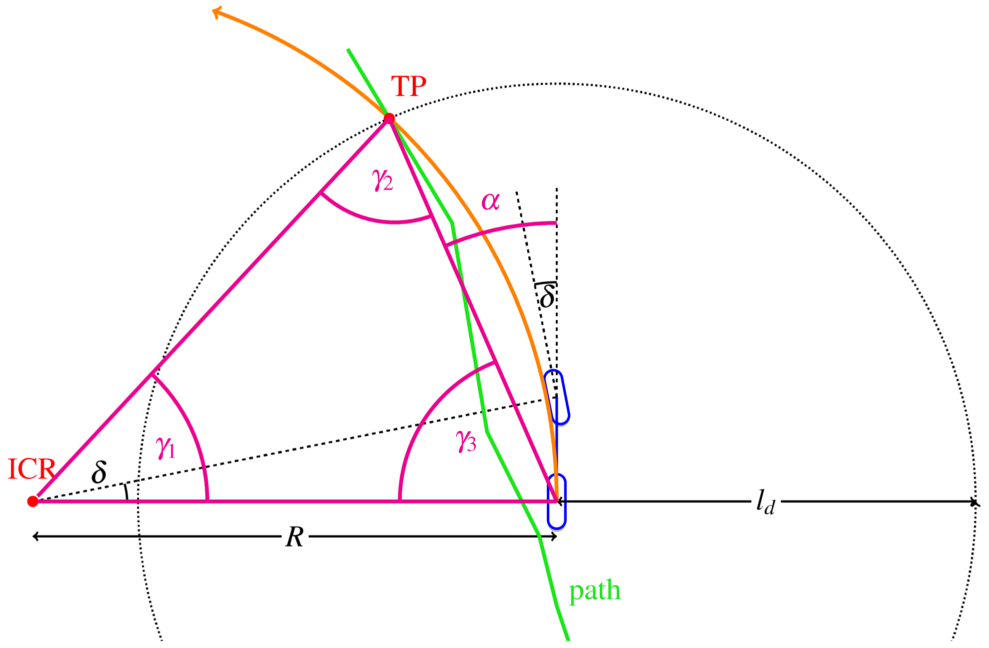


Fig. 28 The magenta triangle helps us to establish a formula for δ.

First, we note that the distance from the instantaneous center of rotation (ICR) to the target point (TP) is equal to R, since TP lies on the orange circle of radius R around ICR. Hence, the magenta triangle is isosceles and $\gamma_2 = \gamma_3$. From the figure we can see that $\gamma_3 + \alpha = 90^\circ$. Hence $\gamma_2 = \gamma_3 = 90^\circ - \alpha$. Since the sum of all angles in a triangle equals 180°, we have

$$180^\circ = \gamma_1 + \gamma_2 + \gamma_3 = \gamma_1 + (90^\circ - \alpha) + (90^\circ - \alpha)$$

which yields $\gamma_1 = 2\alpha$. According to the law of sines

$$\frac{l_d}{\sin(\gamma_1)} = \frac{R}{\sin(\gamma_2)}$$

Here, we used that the distance between the rear wheel and the target point TP is l_d. If we substitute $\gamma_1 = 2\alpha$ and $\gamma_2 = 90^\circ - \alpha$ into the above formula, we obtain

$$\frac{l_d}{\sin(2\alpha)} = \frac{R}{\sin(90^\circ - \alpha)}$$

Due to the trigonometric addition formulas, we have $\sin(90^\circ - \alpha) = \cos(\alpha)$ and $\sin(2\alpha) = \sin(\alpha + \alpha) = 2\sin(\alpha)\cos(\alpha)$. Hence, we can further simplify the formula above to find

$$\frac{l_d}{2\sin(\alpha)\cos(\alpha)} = \frac{R}{\cos(\alpha)}$$

which yields $R = l_d/(2\sin(\alpha))$. For the kinematic bicycle model we have previously derived a formula for the wheel angle δ as a function of R. It was $\delta = \arctan(L/R)$, where L is the wheel base, i.e., the distance between the wheels. Combining this with the newly found formula for R we finally obtain

$$\delta = \arctan\left(\frac{2L\sin(\alpha)}{l_d}\right) \tag{11}$$

This is the angle δ we need to pick to reach the target point! We can write down the pure pursuit algorithm now:

Pure pursuit algorithm

For each instant in time:

- Compute the look ahead distance l_d as `l_d = np.clip(K_dd * speed, min_ld, max_ld)`. The function `np.clip` is documented here. `K_dd`, `min_ld`, and `max_ld` are parameters that you can tune.
- Find the target point TP as the intersection of the desired path with a circle of radius l_d around the rear wheel.
- Using the target point coordinates `(x_tp,y_tp)`, determine α as `alpha=arctan2(y_tp,x_tp)`
- Use equation (11) to compute the pure pursuit front wheel angle δ
- **Act**: Turn your steering wheel to set the front wheel angle to δ

Exercise

In this exercise you will implement both pure pursuit and PID.

If you did not do the chapter on lane detection, you probably did not set up your python environment, and you did not download the exercise code. In this case, please visit the appendix to do this now.

To start working, open `code/tests/control/target_point.ipynb` and follow the instructions. Next, open `code/tests/control/control.ipynb` and follow the instructions. This exercise uses a simplistic vehicle simulator within the Jupyter Notebook to test your code. If you completed these exercises successfully, you **can** also run your controller in a Carla simulation:

- Start Carla by executing the file `CarlaUE4.exe` (Windows) or `CarlaUE4.sh` (Linux) in your Carla folder (If you did not download Carla yet, see the appendix).
- Execute `python -m code.tests.control.carla_sim --ex` from the parent directory of `code` and witness your control algorithm in action! If you omit the `--ex` flag, you will see the sample solution.
- By default, the center of the lane is queried from Carla's HD map and given as reference path to your controller. But, if you run `python -m code.tests.control.carla_sim --ex --ld` your `LaneDetector` will be used: The average of the left and right lane boundary, i.e., $(y_l(x) + y_r(x))/2$ will be given to your controller as the reference path. Note that there is a "TODO" item in `carla_sim.py` regarding the correct call to your `LaneDetector` constructor. You should work on this to make sure the simulation runs without error. Running the Carla simulation and your `LaneDetector` at the same time will eat up a lot of hardware resources. Hence, the simulation will probably run with only a few frames per second on your machine, unless it is very powerful.

By Mario Theers and Mankaran Singh