RE510 Experiment 5. Waypoint Following

20234345 이준

Introduction

In this experiment, we learned about vehivle model and controllers for following waypoints. With proper implementation of Stanely controller, I could acheieve cross-track error about 0.03 and speed error about 0.2.

Preliminaries

Model

In this assignment, we modeled our autonomous vehicle as bicycle. This system have 4 states; position along x-axis, position along y-axis, yaw angle, and linear velocity for heading direction. State-space represenation can be written as follows.

$$\begin{bmatrix} \dot{x} & \dot{y} & \dot{\phi} & \dot{v} \end{bmatrix}^T = \begin{bmatrix} v\cos\phi & v\sin\phi & \frac{v}{L}\tan\delta & a \end{bmatrix}^T$$

δ is steering angle and a is acceleration, and these are our control variables.

Controller

- PID controller is a type of controller that combines proportional, integral, and differential terms to manage errors. While PID control can be challenging for complex systems, it is widely used in industry for simpler systems because adjusting the PID gain values allows for easy control.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{d}{dt}e(t)$$

- Pure-Pursuit controller is a technique that geometrically computes and regulates the curvature needed by the rear wheel to move from the current position to the next target. Steering angle can be calucluated as below.

$$\delta(t) = \tan^{-1}(\frac{2L\sin(\alpha(t))}{l_d})$$

- Stanely controller manages vehicle control by accounting for both the heading angle error and the position error relative to the lane. The control point is shifted to the front wheel to ensure it follows the lane accurately. Steering anlge can be calculated as below.

$$\delta(t) = \theta_p(t) + \tan^{-1}(\frac{ke_{fa}(t)}{v_x(t)})$$

Implementation

`global2local`

The `global2local` function converts global coordinates to the ego vehicle's local coordinates by applying a transformation matrix that accounts for the vehicle's position (`ego_x`, `ego_y`) and orientation (`ego_yaw`). It combines the global coordinates into a matrix, applies the transformation, and returns the local x and y coordinates.

```
def global2local(ego_x, ego_y, ego_yaw, x_list, y_list):
    R = np.mat(
       [[math.cos(-ego_yaw),  -math.sin(-ego_yaw), -(math.cos(-ego_yaw)*ego_x-math.sin(-
ego_yaw)*ego_y)],
        [math.sin(-ego_yaw),   math.cos(-ego_yaw), -(math.sin(-ego_yaw)*ego_x+math.cos(-
ego_yaw)*ego_y)],
        [            0,                0,                                          1]])
    global_traj = np.vstack(
       [np.mat(x_list),
        np.mat(y_list),
        np.ones_like(x_list)])
    local_traj = np.matmul(R, global_traj)
    output_x_list = local_traj[0,:]
    output_y_list = local_traj[1,:]
    return output_x_list, output_y_list
```

`find_nearest_point`

The `find_nearest_point` function identifies the point closest to the ego vehicle's position. It calculates the distances from the ego vehicle to each point in `x_list` and `y_list`, finds the minimum distance, and returns this distance and the index of the nearest point.

```
def find_nearest_point(ego_x, ego_y, x_list, y_list):
    diff = np.mat([(x_list-ego_x),
                   (y_list-ego_y)])
    dist = np.linalg.norm(diff, axis=0)
    near_ind = np.argmin(dist)
    near_dist = dist[near_ind]
    return near_dist, near_ind
```

`calc_error`

The `calc_error` function uses the global2local function and the find_nearest_point function implemented above to obtain cross_track_error and heading_error for the current location and return it.

```
def calc_error(ego_x, ego_y, ego_yaw, x_list, y_list, wpt_look_ahead=0):
    local_x_list, local_y_list = global2local(ego_x, ego_y, ego_yaw, x_list, y_list)
    _, near_ind = find_nearest_point(ego_x, ego_y, x_list, y_list)
    lookahead_wpt_ind = (near_ind+wpt_look_ahead)%len(x_list)
    lookahead_wpt = (local_x_list.item(lookahead_wpt_ind),
local_y_list.item(lookahead_wpt_ind))
    next_lookahead_wpt_ind = (near_ind+wpt_look_ahead+1)%len(x_list)
    next_lookahead_wpt = (local_x_list.item(next_lookahead_wpt_ind),
local_y_list.item(next_lookahead_wpt_ind))
    error_yaw = math.atan2(next_lookahead_wpt[1]-lookahead_wpt[1], next_lookahead_wpt[0]-
lookahead_wpt[0])
    error_yaw = normalize_angle(error_yaw) # Normalize angle to [-pi, +pi]
    error_y   = lookahead_wpt[1]
    return error_y, error_yaw
```

`steer_control`

The `steer_control` function calculates the steer angle using `error_y` and `error_yaw`. This function implements Stanely method to minimize cross-track error both in straight and corner.

```
def steer_control(self, error_y, error_yaw):
    steer = error_yaw
    if self.ego_vx != 0:
        steer += math.atan2(self.STEER_P*error_y, self.ego_vx)
    steer = np.clip(steer, -self.MAX_STEER, self.MAX_STEER)
    steer = steer * self.MAX_STEER
    return steer
```

`speed_control`

The `speed_control` function uses simple PID controller to calculate desired throttle which related to linear acceleration of the ego vehicle. It uses velcoity error, its numerical derivation, and integration of error. PID coefficients are defined as member variable of `WaypointFollower` class.

```
def speed_control(self, error_v):
    throttle = error_v*self.THR_P + self.error_v_i/self.control_freq*self.THR_I -
(error_v-self.error_v_prev)*self.control_freq*self.THR_D
    self.error_v_i += error_v
    self.error_v_i = min(self.THR_IMAX, self.error_v_i)
    self.error_v_prev = error_v
    return throttle
```

Result

In this assignment, I implemented a controller for a simple autonomous vehicle designed to follow a set of waypoints. I chose the Stanley method for this task, a popular control technique known for its effectiveness in minimizing cross-track error. The controller was able to achieve a cross-track error of 0.034 and a speed error of 0.225, which are satisfactory results for this application. These performance metrics indicate that

the Stanley method is a robust choice for maintaining the vehicle's trajectory closely aligned with the desired path while keeping speed deviations minimal.

Discussion

Effects of the lookahead distance

The lookahead distance significantly impacted performance. A longer lookahead distance provided greater stability and smoother movements but increased error during cornering. Conversely, a shorter lookahead distance reduced cornering errors, enhancing accuracy but causing some jittering from sensitivity to small changes. Balancing the lookahead distance is crucial for optimizing both stability and accuracy in the control system. Since the target of this assignment was reducing cross-track error, I selected 1 as lookahead index, which has very short lookahead distance.

Strategies to minimize the cross-track error along road shape

- Straight Lines
    - Basically, cross-track error on straight line can be effectively reduced by I-control, since they designed to eliminate such steady-state error.
    - Larger lookahead index increased stability of steering contol, and eliminated jittering phenomenon, which leads cross-track error while driving straightly, occured when lookahead index was small.
- Corners
    - As I mentioned before, minimizing cross-track error can be done by reducing lookahead distance or lookahead index.
    - Braking while cornering was effective also, but it increases velocity error.

Effect of time step on simulation

Adjusting both the controller and system frequency of the simulator significantly improved the system's performance. The original control frequency was 100 Hz, and the system time step was 20Hz. By increasing both to 1000 Hz, the higher frequency updates allowed for more precise control, particularly noticeable in reducing overshooting during cornering. With the finer time step, the system and controller responded more accurately to changes, resulting in smoother and more stable maneuvers.