Traverse Points

Yuhao Zhang

April 15, 2018

1 Introduction

This project is a model of the car-like robot and a control algorithm to traverse several waypoints with specific coordinates and poses. The code is for Sunfounder's PiCar-V platform and no sensors are involved: the car receives no feedback from its motion and surroundings.

2 Kinematics and control law

Ackerman model can be used to describe the kinematics of a car-like robot:

$$\frac{dx}{dt} = v\cos\theta,$$

$$\frac{dy}{dt} = v\cos\theta,$$

$$\frac{d\theta}{dt} = \frac{v}{L}\tan\gamma,$$

where (x, y) is the position of the middle point of the back wheel axis of the robot in world reference frame. θ is the angle of pose of the robot. The steering wheel angle is γ and the velocity of the back wheel is v. L is the length of the vehicle or wheel base.

These equations can be re-written as:

$$\begin{pmatrix} \frac{dx}{dt} \\ \frac{d\omega}{dt} \\ \frac{d\theta}{dt} \end{pmatrix} = \begin{pmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} v \\ \omega \end{pmatrix}.$$

With the initial position-pose (x, y, θ) and the goal (x^*, y^*, θ^*), it is more convenient to write the equations in polar system via a transformation:

$$\rho = \sqrt{\Delta_x^2 + \Delta_y^2},$$

$$\alpha = \arctan \frac{\Delta_y}{\Delta_x} - \theta,$$
$$\beta = -\theta - \alpha + \theta^*.$$

The linear control law for $-\pi/2 < \alpha \leq \pi/2$, i.e. the waypoint is in front of the vehicle is:

$$v = k_\rho \rho,$$
$$\omega = k_\alpha \alpha + k_\beta \beta,$$

where k_ρ, k_α, k_β are arbitrary coefficients that satisfies $k_\rho > 0, k_\beta < 0, k_\alpha - k_\rho > 0$.

The control law for the cases where the waypoint is behind the vehicle is the same as above, but with transformed angles:

$$\alpha' = -\pi - \beta,$$
$$\beta' = -\pi - \alpha,$$

and $v' = -v$.

3 Pose estimation

Without the feedback from sensors, it is required to estimate the motion from the kinematics of the robot directly. Using the linear approximation of the kinematics equations for a short time period Δt one can obtain

$$\Delta(x, y, \theta) = (v \cos\theta \Delta t, v \sin\theta \Delta t, v/L \tan\gamma \Delta t).$$

Alternatively, the exact solution can be obtained by solving these differential equations directly:

$$\Delta(x, y, \theta) = (R_b \sin(K\Delta t), R_b(1 - \cos(K\Delta t)), K\Delta t),$$

where $R_b = L/\tan\gamma$ and $K = v/R_b$. However, this estimation is non-linear and makes the control law purposed unstable. Therefore the linear approximation will be used in the following sections.

4 Control algorithm implementation

The control law purposed in Sec.(2) and the pose estimation algorithm purposed in Sec.(3) have been implemented in `drive_to_points.ipynb`. Given the input `waypoints.txt`, the generated (one viable) trajectory and the estimated poses along it is plotted as Fig.(1). Yet this naive plan is actually not possible for the PiCar platform, for it ignores the speed limit

of the car. To amend this, two solutions are provided:1. round the off-limit speed to the limits, e.g. if $v > MAX_v$, $v = MAX_v$, but this approach causes the control law to be unstable. 2. use constant speed, and scale the steering wheel control law accordingly. The latter approach will be used in the rest of the paper.

The algorithm terminates as the estimated positions are at the vicinity of waypoints.

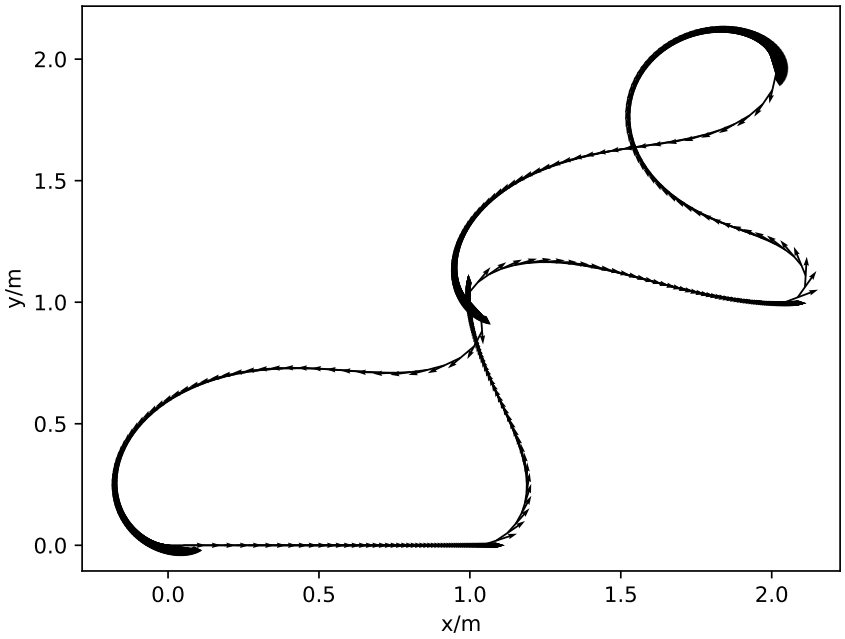

Figure 1: One generated trajectory.

5 Experiment and conclusion

The algorithm is run on Sunfounder's PiCar-V platform with input `waypoints.txt`. The designed trajectory is shown as Fig.(2)

The error for each points is listed as Tab.(1). The error-bar is given as the 95% confidence interval.

It can be seen that the waypoint $(1, 0, 0)$ is the easiest and the algorithm can finish it almost without deviation. (1,1,1.57) and (1,1,-.78) are the two hardest: the former one requires a sharp turn and has high requirement for the pose estimation algorithm. The latter one contains continuous turning and again is hard for pose estimation.

The algorithm is able to drive the vehicle back (although with huge deviation) and has a relatively small deviation in terms of pose (0.41).

Clearly due to the lack of feedback, the error accumulates and the robot deviates more at the later waypoints. The robot is also prune to other issues such as motor acceleration

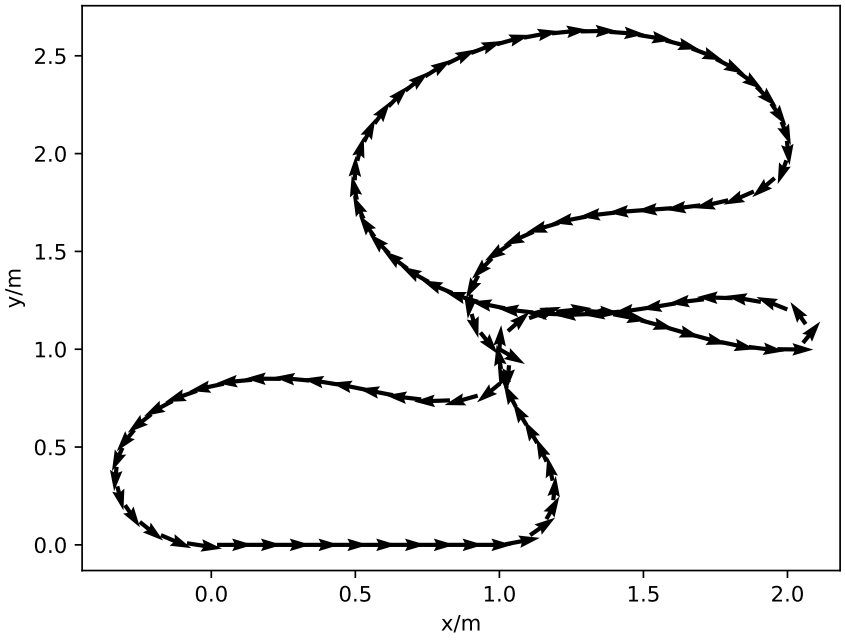

Figure 2: The trajectory and control plan actually used.

Table 1: Experiment results on the robot

Waypoints	Δ distance/m	Δ radian
(0,0,0)	/	/
(1,0,0)	0.04 ± 0.01	0.05 ± 0.02
(1,1,1.57)	0.67 ± 0.09	0.70 ± 0.12
(2,1,0)	0.86 ± 0.10	0.32 ± 0.04
(2,2,-1.57)	1.30 ± 0.06	1.98 ± 0.23
(1,1,-.78)	2.16 ± 0.44	1.59 ± 0.07
(0.0.0)	1.89 ± 0.52	0.41 ± 0.18

rate, tyre pressure and turning rate limit, power difference of the two motors, etc. The algorithm, especially the pose estimation part does not accommodate the sliding and the road condition. Additionally, the control is over the motor power instead of the actual speed of the vehicle, which is the major source of uncertainty and error of the model.