Instruction for Preliminary Project 3

Lookahead Pure Pursuit Control

1. Introduction

Imitation learning is a widely used learning method that has exhibits promising performance. Breifly there are two steps of implementing an imination learning framework. First expert demonstrations or datasets needs to be collected. Second, a model or an agent is trained upon the obtained dataset or demos. The goal of imitation learning is to exploit the expert demonstration behavior into our model or agent.

In this project, we will design a race car that can efficiently navigate a predefined racetrack. Waypoints are strategically posistioned along the track, and the race car will strive to reach each subgoal using a pure pursuit algorithm. To execute this algorithm, we will employ a PID controller to guide the race car towards the intended goal waypoint at each time step. This algorithm will be used to obtain expert demonstrations which will serve as valuable training data for our future algorithm.

2. PID control

2.1. Main theory

A PID controller(proportional-integral-derivative controller) is a control loop feedback mechanism widely used in industrial control systems and a variety of other applications requiring continuously modulated control. A PID controller continuously calculates and error value $e(t)$ as the difference between a desired setpoint and a measured process variable and applies a correction based on proportional, integral, and derivative terms (denoted P, I, and D respectively) which give their name to the controller. The PID control scheme is named after its three correcting terms, whose sum constitutes the manipulated variable (MV). The proportional, integral, and derivative terms are summed to calculate the output of the PID controller. Defining $u(t)$ as the controller output, the final form of the PID algorithm is

$$u(t) = K_\mathrm{p} e(t) + K_\mathrm{i} \int_0^t e(\tau)\,\mathrm{d}\tau + K_\mathrm{d} \frac{\mathrm{d}e(t)}{\mathrm{d}t},$$

where $e(t)$ is the error between the setpoint and the current possess variable and K_p, $K_{i,}$, K_d are the propotional gain, the integral gain and the derivative gain respectively.

The proportional term produces an output value that is proportional to the current error value. The proportional term mainly tries to adjust the error term $e(t)$ to zero. The proportional response can be adjusted by multiplying the error by a constant K_p called the proportional gain constant. A high proportional gain results in a large change in the output for a given change in the error, but the system can become unstable. In contrast, a small gain reults in a small output response to a large input error, and a less responsive or less sensitive controller. The contribution from the integral term is proportional to both the magnitude of the error and the duration of the error. The integral in a PID controller is the sum of the instantaneous error over time and gives the accumulated offset that should have been corrected previously. The accumulated error is then multiplied by the integral gain K_i, and added to the controller output. The integral term accelerates the movement of the process towards setpoint and eliminates the residual steady-state error that occurs with a pure proportional controller. However, since the integral term responds to accumulated errors from the past, it can cause the present value to overshoot the setpoint value. The derivative of the process error is calculated by determining the slope of the error over time and multiplying this rate of change by the derivative gain K_d. The magintude of the contribution of the derivative term to the overall control action is termed the derivative gain K_d. Derivative action predicts system behavior and thus improves settling time and stability of the system.

2.2 Discretization

In the PID controller, the integral term can be approximately discretized, with a sampling period Δt, as

$$\int_0^{t_k} e(\tau)d\tau = \sum_{i=1}^{k} e(t_i)\Delta t$$

and the derivative term can be discretized as

$$\frac{de(t_k)}{dt} = \frac{e(t_k) - e(t_{k-1})}{\Delta t}$$

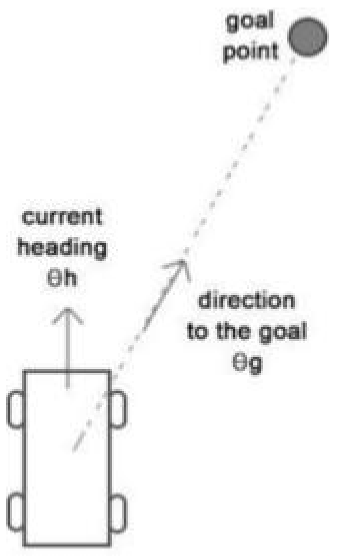
Therefore, the control value $u(t)$ can be approximated as

$$u(t) = K_p e(t) + K_i(t)\Delta t \sum_{k}^{t-1} e(k) + \frac{K_d}{\Delta t}(e(t) - e(t-1))$$

or in recursion relation form.

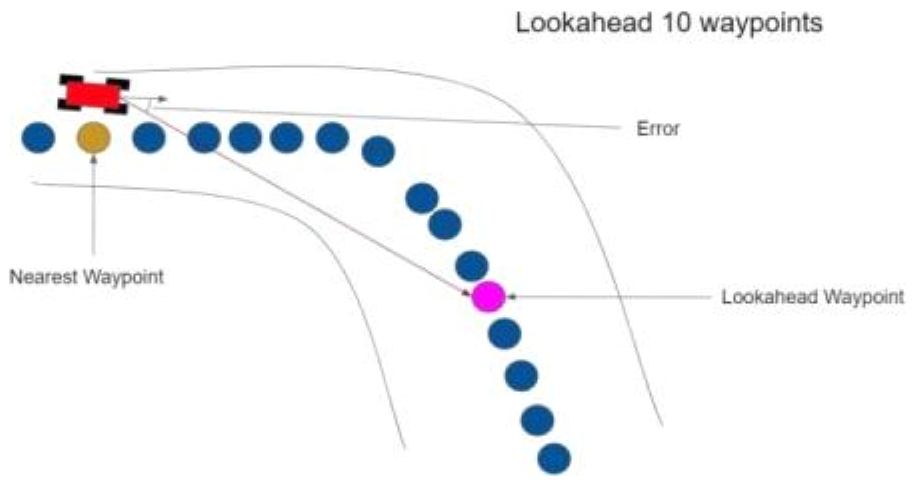
$$u(t) = u(t-1) + (K_p + \frac{K_d}{\Delta t})e(t) + (-K_p + K_i\Delta t - 2\frac{K_d}{\Delta t})e(t-1) + \frac{K_d}{\Delta t}e(t-2)$$

You should control the heading direction of the car to make it follow a given path. So, in this case, the variable to process is the heading direction of the car. To make the car reach the goal point, the car heading should face the goal point. You can calculate the direction to the goal point for your car in polar coordinate. This value becomes the setpoint which the controller should track. To summarize, the process value which to control is θ_h, and the set point can be θ_g, then $e(t) = \theta_g(t) - \theta_h(t)$.



3. Lookahead Pure pursuit

In this preproject, students are asked to use lookahead pure pursuit. In the original pure pursuit, the race car tries to reach a fixed waypoint, and when reached, the subgoal is changed to the next waypoint. In this project, this is modified a bit. The race car fined the nearest waypoint from its current position. Then, the race car tries to got to the 10th next waypoint from its nearest waypoint. Students will be asked to code towards this lookahead waypoint using a PID controller.



4. Assignment

For this preproject, you need to clone the github repository for updating the skeleton which we have pushed in our github repository. Fill in the TODO part inside the **pid_control.py** file ONLY and don`t modify other parts of the code, or else your code might fail. To run the pid_control node follow the command below. we have also updated the ReadME in our github repository.

4.1 Instructions

First clone the repository and install packages

```
git clone https://github.com/rllab-snu/Intelligent-Systems-RLLAB.git
cd Intelligent-Systems-RLLAB/Intelligent-Systems-2025-Pre/rccar_gym
pip install -e .
```

Then build the packages using colcon.

```
cd Intelligent-Systems-RLLAB/Intelligent-Systems-2025-Pre
colcon build —symlink-install

# if build fails try installing the dependencies first
rosdep update --rosdistro foxy
rosdep install -i --from-path src --rosdistro foxy –y
```

For running the node follow the command below

```
ros2 run rccar_bringup pid_control
# to save trajectories
ros2 run rccar_bringup pid_control --save
```

To evaluate your code and run rendering you need to publish /query topic manually.

```
ros2 topic pub --once /query message/msg/Query "{id: '0', team: 'RLLAB', map: 'map1', trial: 0,
exit: false}"
# you can use other maps we provide in the maps directory
```

TODOs

1) Find the nearest waypoint, calculate the error between the racecar heading and the direction vector between the lookahead waypoint and the racecar.
2) Determine the input steering using this error through the PID controller.
3) Use K_p, K_i, K_d to tune your PID controller. Do not change the other parameters predefined in the code.
4) Calculate the input velocity of the racecar appropriately in terms of input steering.
5) For future use, fill in the trajectory saving part in the skeleton code.

5. Submission

Please follow the instructions below to submit your project file. You should submit two contents.

1) Your code: your pid_control.py file
2) A short video or GIF file showing your RC car moving on any of the maps that we provided. The video or file name should be {map_name}.gif or {map_name}.mp4 (to save your video use ctrl shift alt r)
3) One .pkl file of your saved trajectory. The file name should be {map_name}.pkl

Upload these on eTL in a single compressed file named IS_{StudentNumber}_Pre_Project3.zip

Due to: 2025.10.15. 23:59 KST