Assignment 2: Control

The goal of this assignment is to make you acquainted with fundamental instances of controllers and their trade-offs. In this assignment you will implement three distinct controllers and analyze them with respect to each other. Each controller will be designed to track a predetermined trajectory, but as you will see they will face trade-offs in terms of capability, complexity and robustness. By the end of this assignment you will:

- become familiar with strengths and weaknesses of various feedback control strategies.
- become familiar with PID, pure pursuit, and model predictive control (MPC).
- be able to identify for which scenario each control strategy works best.
- have developed skills to analyze control strategies and iteratively improve them.
- gain an appreciation for the power of cost functions.

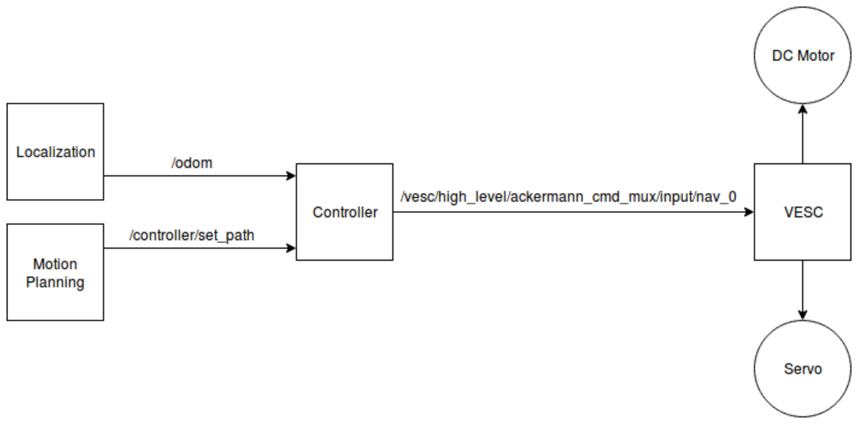
As you begin to familiarize yourself with the control laws you are implementing, we recommend consulting the following readings:

1. Fast line-following robots part 1: control, Andy Sloane, 2018
2. A Survey of Motion Planning and Control Techniques for Self-driving Urban Vehicles, Brian Paden, et. al., 2016
3. Implementation of the Pure Pursuit Path Tracking Algorithm, R. Craig Coulter, 1992
4. Modern Robotics: Mechanics, Planning, and Control, Kevin M. Lynch and Frank C. Park, 2017

1 Path Tracking

1.1 Overview

Path tracking algorithms have a rich history in mobile robotics. The controllers you will implement have been used in DARPA Grand Challenge and DARPA Urban challenge amoung many other contexts. In many land-based navigation scenarios, the robot's navigation task will be divided into two separate tasks: motion planning and control. The purpose of the motion planner is to generate a reference trajectory which the controller can follow. In our configuration, our controller will use the inferred pose generated by the localization module to generate controls which track a trajectory provided by a motion planning module. As such, your implementation will adhere to the following diagram:



Each reference trajectory ξ will be specified as a series of reference waypoints $\xi(\tau) = (x_{ref}, y_{ref}, \theta_{ref}, v_{ref})$ in the vehicle's world frame. Note that this does include reference velocities as well. For our purposes, the reference trajectory will be encoded as `XYHVPath.msg` ros message.

Familiarity with the following terms will help you with the design and implementation of your controller:

1.2 Reference Index (τ)

Every reference trajectory ξ will consist of a series of reference waypoints your controller can track. The "reference index" τ specifies which reference point is selected from the trajectory, and it is usually a function of the vehicle's pose p.

1.3 Cross Track Error (CTE, y_e)

Each controller will seek to minimize some form of error between the car's pose p and the reference pose p_{ref}. One way to compute this error is known as Cross Track Error (CTE). Cross Track Error y_e is the y component of the error vector e_p between the pose of the car p and the reference point $p_{ref} = (x_{ref}, y_{ref})$. In order to compute the CTE, you must rotate the error vector into the frame of the car, using the inverse rotation matrix R^T:

$$e_p = p - p_{ref}$$

$$e_p = \begin{bmatrix} x_e \\ y_e \end{bmatrix} = R^T(\theta_{ref}) \left(\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x_{ref} \\ y_{ref} \end{bmatrix} \right)$$

See Figure 13.20 in *Modern Robotics* for a useful visual explanation of the transformation.

2 PID Control

The proportional-integral-derivative (PID) controller is a feedback control mechanism that is simple to implement and widely used. In this part of the assignment you will implement the PID controller in `pid.py`, tune it, and evaluate its performance on a series of simple control tasks. PID control is usually defined as:

$$u(t) = K_\mathrm{p} e(t) + K_\mathrm{i} \int_0^t e(t')\, dt' + K_\mathrm{d} \frac{de(t)}{dt}$$

where K_p, K_i, and K_d, are all non-negative coefficients for the proportional, integral, and derivative terms of the error $e(t)$ (in our case this will be CTE). The weighted sum computes your control $u(t)$, which in the case of the racecar is going to be the steering angle $\delta(t)$. One slight modification we suggest you make, however, is to drop the integral term entirely. Thus you will actually be implementing **PD Control**.

Reference the `pid.py` file for getting started.

Once you a confident with the implementation of your controller, use the `runner_script.py` script to test your controller against a set of reference trajectories (right turn, left turn, and circle). You can vary the radii of the arcs of the turns and circle to test the robustness of your controller, as well as start with different initial poses to see how your controller responds. A principled approach to finding robust K$_p$ and K_d values is described in this article. You may find the included ipython notebook `Controller Plotting.ipynb` useful for generating plots that describe the performance of your controller. Use this as a starting point for analyzing your controller, and document your tuning process.

3 Pure Pursuit Control

Pure Pursuit is a classic feedback control mechanism that is frequently used on active mobile robots. As described in *Implementation of the Pure Pursuit Path Tracking Algorithm*: "Pure pursuit is a tracking algorithm that works by calculating the curvature that will move a vehicle from its current position to some goal position. The whole point of the algorithm is to choose a goal position that is some distance ahead of the vehicle on the path. The name pure pursuit comes from the analogy that we use to describe the method. We tend to think of the vehicle of chasing a point on the path some distance ahead of it - it is pursuing that moving point. That analogy is often used to compare this method to the way humans drive. We tend to look some distance in front of the car and head towards that spot. This lookahead distance changes as we drive to reflect the twist of the road and vision occlusions." We encourage you to reference this paper when implementing the controller. Some students may find this description also useful.

Reference the `purepursuit.py` file for getting started.

You will notice that pure pursuit relies on fewer parameters than the PD controller for its implementation. Experiment with different lookahead differences and document which distance yielded the most robust control across your test cases. We expect you to evaluate your controller across varying turn radii and speed regimes. Document your testing and provide a table which shows you exhaustively tested your controller across varying configurations.

How does pure pursuit compare against PD control? What tradeoffs would you consider when deciding between controllers.

4 Model Predictive Control

The model predictive controller is decidedly more complex than the previous two controllers. Give your team ample time for the implementation and testing of this controller.

Although the simple control laws above are adequate in many driving conditions, we have often found that taking additional factors into account (such as the slippage of the vehicle or nearby obstacles) is useful when tracking a path. Such details can be incorporated into a model of the vehicle or the environment, which is then used to solve a limited time horizon open-loop motion planning problem. Once the action has been determined, the system will execute it and immediately begin computing the next action given the most recent state estimate. Such "tight loop" control can allow the system to penalize collisions undesirable states, such as collisions, while rewarding paths that advance the pose of the car closer to the goal. Such "punishments" or "rewards" are encoded by the cost function.

Your model predictive controller will be implemented in `mpc.py`. In this file you will:

- Pre-compute a set of K control trajectories
- Roll out each control trajectory T steps forward from an initial pose specified by your localization module using the kinematic car model.
- Evaluate each rollout using a cost function that penalizes collisions and rewards states which are closer to the goal.

Writing an effective cost function will require some iteration. Document your process for determining your cost function using the plotting tool. To help with this, construct at least three different scenarios in which the car must avoid an obstacle. You can create new maps (by editing sandbox.png) which contain obstacles. What worked and what didn't? Are there any pathological cases which were difficult for the model predictive controller to solve?

5 (Extra Credit) Non-linear Control

TODO: Students will implement a Non-linear Lyapunov controller and do the following derivations.

6 Deliverables

Put the associated work in the directories and `tar` your repository.

```
$ tar czf lab2.tar.gz lab2
```

1. `lab2/videos/`: Videos of . . .

   (a) Your `subscriber.py` driving the car with messages from `publisher.py` (You can use Kazam to take a video of your screen.) `subscriber.mp4`
   (b) Your RACECAR performing a figure 8 in simulation (forward and backwards). `sim-figure8-forward.mp4, sim-figure8-backward.mp4`
   (c) Your RACECAR performing a figure 8 in real (forwards and backwards). A cell phone camera of the car will suffice. `real-figure8-forward.mp4, real-figure8-backward.mp4`

2. `lab2/bags/`: Bags:
   (a) A bag of your figure 8 in sim: `sim-figure8.bag`
   (b) A bag of your figure 8 in real: `real-figure8.bag`
3. `lab2/src/`: Code:
   (a) Your `subscriber.py`
   (b) Your `pose_markers.py`
   (c) Your `bag_follower.py`
   (d) Any extra credit source files.
4. `lab2/launch/`: Launch files:
   (a) Your `subscriber.launch`
   (b) Your `pose_markers.launch`
   (c) Your `bag_follower.launch`
   (d) Any extra credit launch files.
5. `lab2/writeup.txt`: Writeup (answer the following):
   (a) You should have collected two bags of the robot doing a figure-8, one in simulation, and one on the real robot. Play back both of these bags on the real robot. Explain any significant differences between the two performances.
   (b) Is the robot successfully able to navigate through Allen 022 when playing back the corresponding bag? Note any deficiencies in performance. Explain why these deficiencies might occur.
   (c) If you did any of the extra credit, describe which ones you did and shortly describe what you did.