John Alsterda

CS 221: AI

16 Nov 2018

Online Wheel Alignment via Reinforcement Learning - Progress Report

(re) Introduction

Vehicles undergo periodic manual calibration of their wheels to correct misalignment after striking curbs or potholes. A misaligned vehicle will 'pull' left or right and wear its tires quickly and unevenly [1]. For my CS 221 project, I plan to perform this tedious job automatically and online using Reinforcement Learning via Gradient Descent and/or Q-Learning strategies.

Model

At the heart of either strategy is a model which predicts the evolution of vehicle state over time:

$$x_+^{model} = f\left(x_0^{measured}, u, \Delta t\right)$$

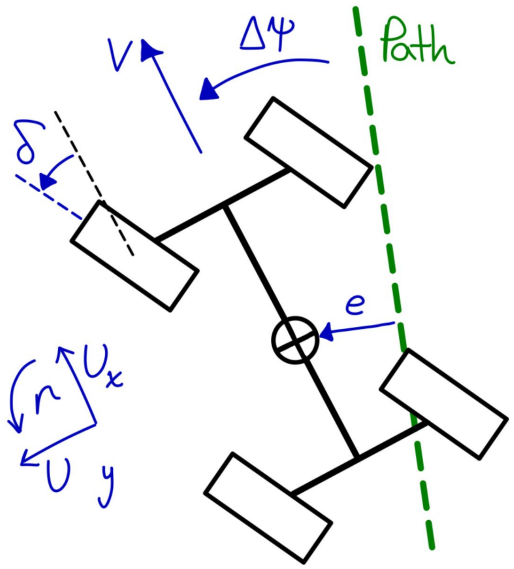
The model predicts that if control u is applied to the vehicle in state x_0 for time Δt, it will evolve to state x_+. This output is compared to a measured state x_+ to produce a model mismatch error Δx:

$$\Delta x = x_+^{model} - x_+^{measured}$$

This error is assumed to be caused by wheel misalignment. The actual vehicle's state evolution is perturbed by misalignment, whereas the naive model (initially) believe the wheels are installed correctly. Thus the model mismatch forms the basis for my cost function; the program aims to minimize this error by learning a correction angle for each wheel in the model.

The model, based on the states shown in the top-right diagram, is a linearized version of a high-fidelity 4-wheel vehicle model based on forces generated due to tire slip. Simulated results in this progress report substituted the high-fidelity nonlinear model to generate data, which I hope to replace later with experimental collected data. State e is the vehicle's lateral from the reference path, $\Delta\psi$ the heading error, U_y the lateral speed, and r the yaw rate. In linear algebraic form, the model becomes:

$$x_+ = \Delta t * \left(A\, x_0 + B\, u + C\right)$$



State vector $x = [\, e\ \Delta\psi\ U_y\ r\,]$ and input vector $u = [\, \delta_{FL}\ \delta_{FR}\ \delta_{RL}\ \delta_{RR}\,]$.

Gradient Descent

The algorithm I've implemented so far is Gradient Descent, which works iteratively minimize a cost function C via the update rule [2]:

$$\delta_{error} := \delta_{error} - \eta\, \nabla_{\delta} C$$

The first cost function I tried was the model mismatch squared, Δx^2. Taking its gradient and substituting into the update rule yielded:

$$\delta_{error} := \delta_{error} + 2\,\eta\,\Delta t\,[\, x_+^{meas} - (I + \Delta t\, A) x_+^{meas} - \Delta t\,(B\,u + C)\,]^T B$$

The Gradient Descent algorithm follows:

0. Initialize the vehicle state and desired path (and randomly assign misalignment to each wheel if simulating)
1. Calculate path following steering and throttle commands [3]
2. Deploy commands to the vehicle in experiment or simulation for Δt seconds. Collect x_+^{meas}
3. Preform Gradient Descent
4. Repeat

As the preliminary data show in the Results section, some progress was made toward learning the wheel misalignment. However, there is much room for improvement. In the coming weeks I hope to add an additional term to the cost function to hopefully increase performance. This term will either seek to minimize the longitudinal force required to maintain speed, or minimize the motor current required to hold steering angles. The core concept is to introduce a term which involves important dynamics ignored by the linear model. More to follow…

Q-Learning

In traditional Q-Learning, an optimal policy is generated by estimating the value of state-action pairs [2]. However, the simpler approach I plan to implement does not make use of actions. My states and rewards are:

- States: a correction angle **Δδ** for each wheel. $S \in R^4$, possibly discretized as shown below.

$\Delta\delta_{FL}$	…	-¼	0	+¼	-½	…
$\Delta\delta_{FR}$	…	-¼	0	+¼	-½	…
$\Delta\delta_{RL}$	…	-¼	0	+¼	-½	…
$\Delta\delta_{RR}$	…	-¼	0	+¼	-½	…

- Reward: Penalty based on $|\Delta x|$ or similar, motivating the algorithm to find the state which allows the vehicle to most closely follow the vehicle model f().

Instead of estimating the value of state-action pairs, I will simply find the best combination of states, one for each wheel. (Does this mean Value Iteration?) I will accomplish this by exploring the state-space with an Upper Confidence Tree (UCT) exploration / exploitation strategy [3]. In UCT, new states are chosen by maximizing anticipated reward and state-space coverage simultaneously.

My intended Q-Learning algorithm follows:

0. Initialize the vehicle state and desired path (and randomly assign misalignment to each wheel if simulating)
1. Calculate path following steering and throttle commands [3]
2. Choose a $\Delta\delta$ quartet to explore based on UCT; subtract it from the steer command.
3. Deploy commands to the vehicle in experiment or simulation for Δt seconds. Collect x_+^{meas}
4. Calculate reward and add it to current value of each $\Delta\delta$
5. Repeat

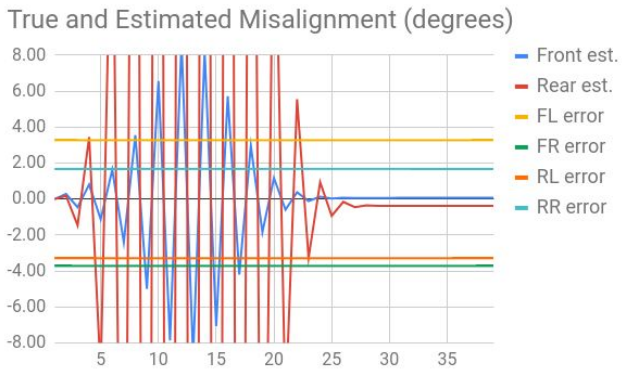
After many iterations each state will have an estimated value. At this point I can choose to estimate my misalignment by picking the four highest value states. Alternatively, I can use the values to regress a smooth value function and escape the confines of discrete states posed above. Interestingly, while Q-Learning is known as a model-free method, I'm still incorporating a model to calculate the reward.

Results

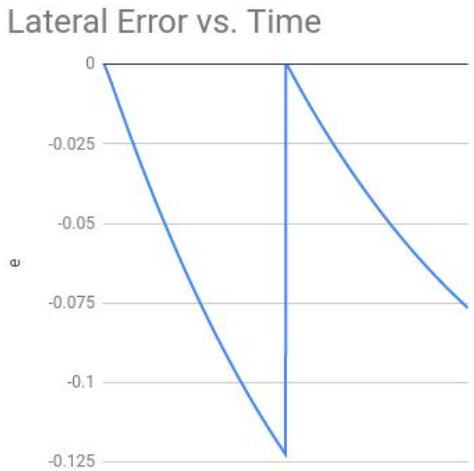
Before an experiment is warranted, my strategies must be shown to work in simulation. The plots on the following page demonstrate qualitatively how Gradient Descent performed in 4 typical trials. A more complete analysis will involve many more trials for a large N perspective. Please excuse the utterly basic state of my plots. They will be more descriptive and professional in my final paper - promise!

Experiment #1:

The plot below shows the true misalignment and estimated misalignment during the simulated experiment. Because the linear model treats the left and right of the vehicle equally, the front-right (FR) and front-left (FL) are lumped, as are the rear-right (RR) and rear-left (RL). Perhaps the learning rate is too great at the beginning; it decreases by one over the sqrt() of the iteration number. In this trial, the true misalignments for the right and left were opposite in sign, a bad case for a method which lumps the axle! However, the next plot shows that even a minor adjustment to dial out this error improved path tracking performance.



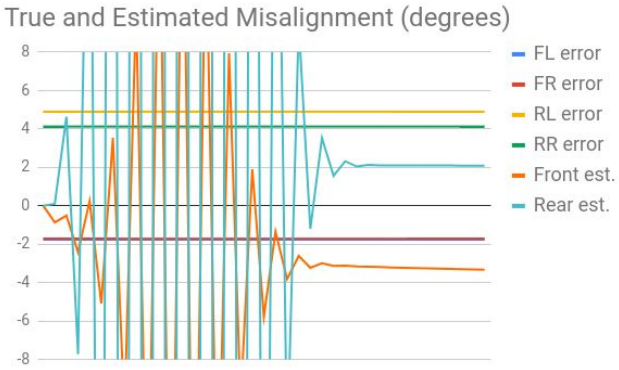
The next plot shows the lateral error during two path following runs. The first is the learning phase, where the misalignment estimate is not incorporated into control. Then the estimate is incorporated to see if path-following performance improves. Below we see that the vehicle began on the path with zero error, which accumulates to 125 cm. Next, even though the misalignment improve was very modest, the path tracking error deviates 50 cm less over the same period!



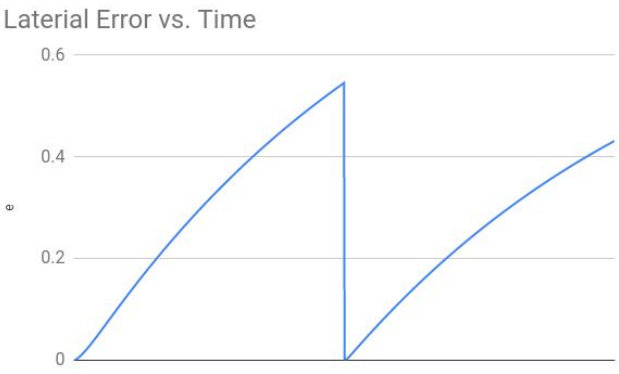
Next, the speed error is plotted. Here we can see speed quickly fall due to wheel scrubbing (friction due to misalignment). The integral controller is able to return to the desired speed by the end of the simulation. However, in contrast to the lateral error plots, no improvement was seen after applying the misalignment correction. Scrubbing was still significant!

Experiment #2:

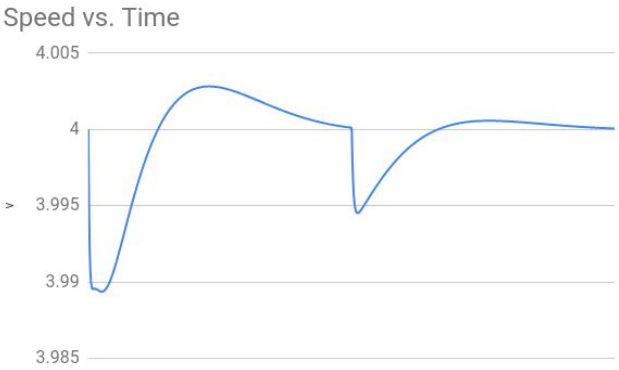
In the second experiment, gradient descent moved much more than in the first, and in the correct direction for both front and rear axles! This was possible because the FR and FL errors agreed in sign, as well as the RL and RR. This coincidence was merely luck, of course - the next edition of my algorithm must be able to handle the mixed misalignments we saw in experiment #1.



Lateral error is plotted again next. Interestingly, the improved misalignment estimation from this experiment did not result in quite as large an increase in path-tracking performance as in experiment #1. I believe it was because the front wheels (which steer to follow the path) are much more important than the rear. Gradient Descent overestimated the front misalignment in this trial.



Speed tracking, however, improved handsomely from the learning phase to the validation phase in this experiment. Once again we see the learning phase begin with a steep drop in speed as the misaligned wheels scrub speed off the car. But in the phase 2, after the estimated misalignment is applied, speed drops only half as much and recovers much more quickly.



Conclusion and Future Work

Once again, please forgive the awful shape these plots and analysis are in! I would be happy to sit down with you and talk about my progress and discuss what might be gleaned from these preliminary results. I'm excited to hear what other strategies might improve my work as we head to the end of the quarter. Running an algorithm in experiment (vice simulation) would be a major accomplishment, but is only possible if simulations are working first.

Thus far, gradient descent has proved to be a good first step towards showing that model mismatch can help identify wheel misalignment, and to make potentially correcting predictions. The basic linear model may have been easy mathematically to take the gradient, but its small angle assumptions may be removing helpful information. It will be especially necessary to get independent estimates for left and right wheels, at the least. Perhaps an automatic differentiation tool could allow me to use gradient descent on the full nonlinear model? Q-Learning could also use the nonlinear model, as no derivatives are required.

References

1. "Wheel alignment." Wikipedia. https://en.wikipedia.org/wiki/Wheel_alignment
2. "Gradient descent." Wikipedia. https://en.wikipedia.org/wiki/Gradient_descent
3. Kapania and Gerdes. "Path Tracking for Highly Dynamic AVs". American Control Conference. 2015. https://ddl.stanford.edu/sites/default/files/publications/2015_ACC_Kapania_Path_Tracking_ILC.pdf
4. "Q-Learning." Wikipedia. https://en.wikipedia.org/wiki/Q-learning
5. "Monte Carlo Tree Search." Wikipedia. https://en.wikipedia.org/wiki/Monte_Carlo_tree_search