Automobile Wheel Alignment via Machine Learning

John P. Alsterda[1]

Abstract— **This project presents a potentially novel strategy for correcting an automobile's wheel misalignment through machine learning. Wheel misalignment correction is an important periodic maintenance task which improves a vehicle's control performance and fuel economy while slowing tire degradation. The current state-of-the-art strategy requires a vehicle to be removed from operation and taken to a professional mechanic to make manual adjustments with expensive equipment. This project suggests a vehicle of the future should be aligned continually during normal operation, by comparing predicted vehicle states from a dynamics model to measured states based on differential-GPS. Assuming misalignment is primarily responsible for model mismatch, a correction angle can be learned and applied which eliminates difference between prediction and measurement. Gradient Descent and Value Estimation are implemented and compared, as competing strategies to learn a simulated vehicle's misalignment angles.**

I. INTRODUCTION

Vehicle wheels experience misalignment due to acute impact with potholes or curbs, or slowly over time as suspension and steering components age. When a wheel is misaligned, its physical angle with respect to the chassis is no longer consistent with the steering wheel angle and other wheels. The result is degraded control performance as the wheel angle cannot be precisely determined or commanded. The efficiency of the vehicle also suffers, as valuable energy is lost through friction between the misaligned tire and the road. This same scrubbing friction additionally contributes to uneven and rapid tire wear, shortening the tire's unusable life and may even lead to blowout [1]!

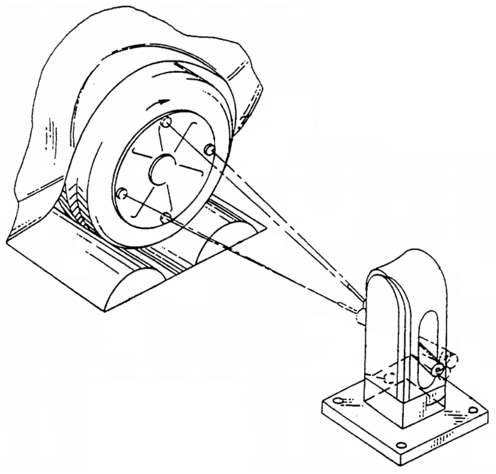

Fig. 1. LASER tracking apparatus invented by K. Spann *et al.* [2]

[1]Department of Mechanical Engineering, Stanford University, Stanford, CA 94305, USA. *alsterda@stanford.edu*

Fig. 2. X1, a 4-wheel independent steering research platoform

Wheel re-alignment is a regular maintenance task. Motorists are recommended to check alignment every 2 or 3 years, or whenever symptoms are observed such as uneven tire wear, steering wheel vibration, or if the vehicle pulls left or right. To measure misalignment, a state-of-the-art and near-universally applied procedure is to mount a "LASER Tracking Wheel Alignment Apparatus" to each wheel as shown in Fig. 1, clipped from the patent submitted by K.Spann *et al.* [2]. The measurement device rotates the wheel and records the deflection of the LASER beam, calculating toe and camber angles. A mechanic will then make fine mechanical adjustment to the wheel's suspension and steering members until a consistent zero deflection is reported.

There are several limitations to this method: First, the vehicle must be taken out of operation, temporarily decreasing the utility or profitability of the vehicle to its owner. Second, a vehicle will sometimes be driven a significant distance before misalignment is corrected, saddling its owner with the consequences of misalignment for longer than necessary. Finally, the LASER equipment and professional mechanic's time are expensive. An ideal solution would correct a wheel's alignment promptly and automatically without taking a vehicle off the road. It might even be accomplished without a driver's direction awareness.

One such invention has been accepted by the U.S. Patent Office, submitted by C. Kyrtsos [3] in 2001. This method leverages the concept that a vehicle's efficiency suffers with misaligned wheels. Over a period of time, the algorithm counts the number of revolutions turned by each wheel. These numbers are then compared to an expected number of revolutions for the distance travelled. This method is limited, however, because it cannot determine whether the left or right wheel is misaligned, nor can it determine whether the wheel is *toed-in* or *toed-out*. It claims to eventually accomplish its correction via a fuzzy-logic algorithm which learns the correct alignment via trial and error. C. Kyrtsos's patent also includes a second contribution relevant to this project: an invention for adjusting a wheel's alignment during normal vehicle operation. The design suggests a linear actuator should be installed in series with a wheel's steering tie-rod, which controls steering angle. By lengthening or shortening this tie-rod, alignment can be corrected online during operation.

The machine learning alignment program presented in this report also assumes a vehicle equipped with such an adjustment system. Currently, almost zero commercial vehicles are equipped to adjust alignment without manual wrenching. However, the Stanford Dynamic Design Lab's 'X1' experimental platform in Fig. 2 is fully capable. Instead of a linear actuator, each of its four wheels' steering angles are controlled independently by DC motors, shown in Fig. 3. By calibrating the zero-degree encoder point of each wheel's motor in software, alignment can be corrected in real-time, online. I anticipate that the next generation of electrified self-driving cars may include a similar steering system and allow online alignment.

A second, more recent invention uses differential GPS to measure misalignment. H. Bae and J. Ryu claim the patent for "Diagnosis of Wheel Alignment Using GPS" [4]. Their invention monitors the steering angle of the vehicle and its yaw rate, and reports misalignment when those measurements disagree. For example, a vehicle with a steering angle of zero should have a yaw rate which also approaches zero. This project takes note of their strategy and builds from it by also using dGPS and a mathematical vehicle model. However, this patent only claims to diagnose the existence of misalignment, and not to actually measure the wheels' misalignment angles.

This CS221 project endeavours to go beyond diagnosis to estimation of each misalignment angle, as well as to implement corrections in real time to ensure vehicles drive safely and efficiently. In Section II, a vehicle dynamics model is introduced to predict the evolution of the vehicle state according to its 4 steering angles, as well as throttle & brake commands. In Section III, a Gradient Descent method is developed to minimize a model mismatch cost function. It's followed by Value Estimation, a search method which also aims to minimize mismatch by exploring combinations of candidate misalignment angles. Section IV presents the experimental setup, data collected from each algorithm, and finally compares the two methods' performance. Section V concludes with next steps and lessons learned.

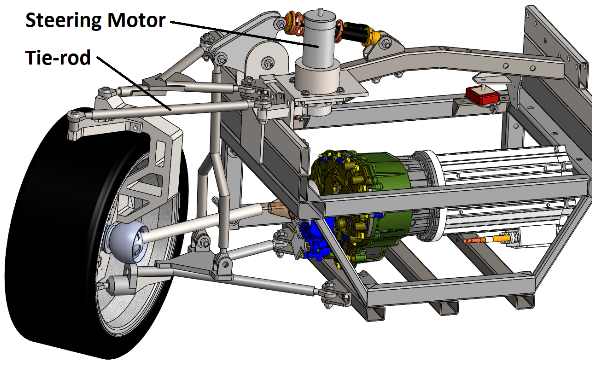

Fig. 3. Steering assembly for X1's rear-left wheel

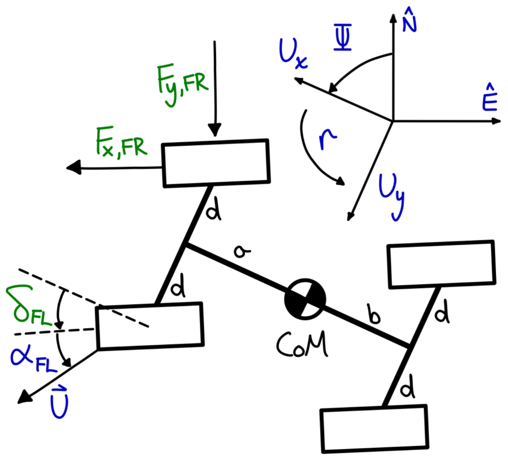

Fig. 4. States and inputs for a 4-wheel vehicle model

II. Vehicle Dynamics Model

The method proposed for estimating wheel misalignment depends crucially on a mathematical model which predicts how input commands affect the evolution of the vehicle state over time. Each iteration, an initial state x^0 is measured, followed by the next state x^+, dt seconds later. dt has been set to $10msec$ for this project. The vehicle state vector consists of three position and three velocity states: $x = [E, N, \Psi, U_x, U_y, r]$ as depicted above in Fig. 4. E, N is the vehicle's center of mass East, North position. Ψ is the vehicle's yaw angle, counterclockwise from due North. U_x and U_y are the vehicle's longitudinal and lateral speeds, respectively, and r is the yaw rate.

Additionally, state x^+_{model} is calculated from x^0 and the deployed input commands. Vehicle commands consist of steering angles $[\delta_{FL}, \delta_{FR}, \delta_{RL}, \delta_{RR}]$, where FL is front-left and similar for the vehicle's 3 other wheels. Throttle and brake commands are lumped into the longitudinal forces at each tire $[F_{x,FL}, F_{x,FR}, F_{x,RL}, F_{x,RR}]$. The differential equations which determine x^+_{model} are as follows:

$$\dot{E} = -U_x\, sin(\Psi) - U_y\, cos(\Psi) \tag{1}$$

$$\dot{N} = U_x\, cos(\Psi) - U_y\, sin(\Psi) \tag{2}$$

$$\dot{\Psi} = r \tag{3}$$

$$\dot{U_x} = r\, U_y + \frac{1}{m}\sum_i^4 F_{x,i}\, cos(\delta_i) - F_{y,i}\, sin(\delta_i) \tag{4}$$

$$\dot{U_y} = -r\, U_x + \frac{1}{m}\sum_i^4 F_{x,i}\, sin(\delta_i) - F_{y,i}\, cos(\delta_i) \tag{5}$$

$$\dot{r} = \frac{1}{I_z}\Big(d\,(-\vec{F}_{FL} + \vec{F}_{FR} - \vec{F}_{RL} + \vec{F}_{RR}) \cdot \hat{c}_x + a\,(\vec{F}_{FL} + \vec{F}_{FR}) \cdot \hat{c}_y - b\,(\vec{F}_{RL} + \vec{F}_{RR}) \cdot \hat{c}_y\Big) \tag{6a}$$

where

$$\begin{aligned} &\vec{F}_i \cdot \hat{c}_x = F_{x,i}\, cos(\delta) - F_{y,i}\, sin(\delta) \\ &and \\ &\vec{F}_i \cdot \hat{c}_y = F_{x,i}\, sin(\delta) + F_{y,i}\, cos(\delta) \end{aligned} \tag{6b}$$

In this model longitudinal forces $F_{x,i}$ are commanded directly, but lateral forces are a linear function of the tire slip angle α, which depends the steering angle and velocity states:

$$\begin{aligned} F_{y,FR|L} &= -C_F\ \alpha_{FR|L} \\ &= \quad C_F\Big(\delta_{FR|L} - tan^{-1}\Big(\frac{U_y + ar}{U_x \pm dr}\Big)\Big) \end{aligned} \tag{7a}$$

$$\begin{aligned} F_{y,RR|L} &= -C_R\ \alpha_{RR|L} \\ &= \quad C_R\Big(\delta_{RR|L} - tan^{-1}\Big(\frac{U_y - br}{U_x \pm dr}\Big)\Big) \end{aligned} \tag{7b}$$

Model mismatch is then defined as $(x^+ - x^+_{model})$, which is assumed to be primarily caused by wheel misalignment. In reality, many factors contribute to model mismatch, including road conditions, sensor noise, tire non-linearity, etc. In this simulation, model mismatch is caused only by wheel misalignment. Thus our machine learning algorithms will work to minimize model mismatch by finding each wheel's misalignment.

III. Learning Algorithms

The following subsections outline my work to implement Gradient Descent and Value Estimation to find wheel misalignment $\delta_{misalign}$. As both have the same high-level goal, I designed them to share a cost function $J(\delta)$ which they aim to minimize. The cost function must capture the desire to minimize model mismatch, but also be differentiable (both mathematically *and practically*) if taking the gradient. The cost function I chose was the 2-norm squared of model mismatch, with a forward Euler integration step:

$$\begin{aligned} J(\delta) &= |x^+ - x^+_{model}|^2_2 \\ &= |x^+ - (x^0 + dt\,\dot{x}_{model})|^2_2 \end{aligned} \tag{8}$$

The resulting high-level optimization to estimate misalignment is then:

$$\hat{\delta}_{misalign} = \underset{\delta_{misalign}}{\arg\min}\, J(\delta + \delta_{misalign}) \tag{9}$$

The Gradient Descent and Value Estimation methods developed herein represent two strategies to solve this optimization.

A. Gradient Descent

The first technique presented for minimizing model mismatch is gradient descent, an iterative strategy which attempts to find the minimum of a cost function by moving in the direction opposite the gradient. The update rule is defined:

$$\hat{\delta}_{misalign} := \hat{\delta}_{misalign} - \eta\,\vec{\nabla}_\delta\, J(\delta) \tag{9}$$

where η is the learning rate, tuned for convergence. I chose $1/\sqrt{iteration\#}$. The next step towards implementation is to calculate the gradient of the cost function. The first few chain rule steps are fairly innocuous, leading to:

$$\vec{\nabla}_\delta\, J(\delta) = -2\,dt\,\eta\,(x^+ - x^+_{model})^T \vec{\nabla}_\delta\, \dot{x}_{model} \tag{10}$$

However, taking the gradient of the nonlinear 4-wheel vehicle model presented in Section II is quite a challenge. As a first attempt, I chose to linearize this model to make its gradient simple. The linearized model takes the form below, with path-tracking lateral error e and heading error $\Delta\psi$:

$$\begin{aligned} \dot{x}_{linear} &= \big[\ \dot{e}\ \dot{\Delta\psi}\ \dot{U}_y\ \dot{r}\ \big]^T = Ax + Bu + C \\ &\longrightarrow \vec{\nabla}_\delta\, \dot{x}_{linear} = B \end{aligned} \tag{11}$$

Linearizing the model required substantial compromise, including small angle approximations and an assumption of constant speed, i.e. all longitudinal forces $F_x = 0$. Small angles eliminated the difference between right and left wheels; thus the linear algorithm only produced two unique misalignment estimates: one for the rear and another for the front. The constant speed assumption also removed any sense that the misaligned wheels would scrub speed off the car; I believe this lost information is important. The learning completed by this Gradient Descent was presented on my poster board, and I omit those results here for brevity. In summary, while path and speed-tracking performance did usually improve, learning was not impressive - especially in cases where left and right wheels were oppositely misaligned.

Next I decided to buckle down and calculate the gradient of the full nonlinear model. Its form is:

$$\vec{\nabla}_\delta\, \dot{x}_{model} = \begin{bmatrix} \partial_{\delta FL}\dot{N} & \partial_{\delta FR}\dot{N} & \partial_{\delta RL}\dot{N} & \partial_{\delta RR}\dot{N} \\ \partial_{\delta FL}\dot{E} & \partial_{\delta FR}\dot{E} & \partial_{\delta RL}\dot{E} & \partial_{\delta RR}\dot{E} \\ \partial_{\delta FL}\dot{\Psi} & \partial_{\delta FR}\dot{\Psi} & \partial_{\delta RL}\dot{\Psi} & \partial_{\delta RR}\dot{\Psi} \\ \partial_{\delta FL}\dot{U}_x & \partial_{\delta FR}\dot{U}_x & \partial_{\delta RL}\dot{U}_x & \partial_{\delta RR}\dot{U}_x \\ \partial_{\delta FL}\dot{U}_y & \partial_{\delta FR}\dot{U}_y & \partial_{\delta RL}\dot{U}_y & \partial_{\delta RR}\dot{U}_y \\ \partial_{\delta FL}\dot{r} & \partial_{\delta FR}\dot{r} & \partial_{\delta RL}\dot{r} & \partial_{\delta RR}\dot{r} \end{bmatrix} \tag{12}$$

Calculating each of these 24 partial derivatives (12 non-trivial) was as fun as it looks. The curious reader is encouraged to read my submitted code if they wish to see each term in detail. Results for nonlinear Gradient Descent are found in Section IV.

B. Value Estimation

The next strategy implemented was Value Estimation. In this framework, an algorithm seeks to find the state with the highest value. The value function was defined as $-J(\delta)$; therefore maximizing value was equivalent to this minimizing cost. The state is defined as a $(\hat{\delta}_{misalign,FL}, \hat{\delta}_{misalign,FR}, \hat{\delta}_{misalign,RL}, \hat{\delta}_{misalign,RR})$ tuple of angles. Hence finding the highest value state corresponds to finding the misalignment angle for each wheel.

Similar to Gradient Descent, I presented my first attempt for the poster session, and then improved it after receiving feedback. Next I briefly summarize the structure of that first attempt, and then describe how it was transformed into its final form. In my first attempt I discretized the state-space into a finite set of candidate angles which captured the range of possible misalignments, roughly $\pm 2°$. With 20 bins for each wheel, the state-space size was $20^4 = 160,000$. I deemed this state-space to be too large to search exhaustively online, and instead decided each wheel's state could be valued independently.

To estimate state values, I measured x^0 and x^+ each iteration $(dt = 10msec)$, and calculated x^+_{model} for several hundred random combinations of misalignment angles. I then updated each state's value to represent that average value calculated for that angle among all iterations:

$$value(angle_i) := \big(N * value(angle_i) - J(\delta_i)\big)/N \quad (13)$$

where i is the wheel, $angle$ is the random candidate angle, and N is the number of evaluations completed so far in that bin.

Finally, I calculated a best-fit curve to each wheel's values via least squares. The best-fit curve freed the algorithm to choose a result unconstrained by the discrete bins. As presented on my poster, this method worked better than linear Gradient Descent, but still not well enough to warrant an experiment beyond simulation on X1. I think the primary weakness was that evaluation at each candidate angle was poisoned by the random misalignment choice for each of the other 3 wheels. I hypothesized that averaging over a large number of iterations would cancel this effect out, but in practice they did not.

Onto the improved Value Estimation: To upgrade the algorithm, I wanted to harness the idea that my candidate-angle range could be narrowed and honed each iteration. There's no reason to keep evaluating angles which are clearly not correct! Moving forward, I would save only the highest value tuple each iteration, instead of all calculated values. To begin the next iteration, I re-discretized my state-space, narrowing the range by 1% from the last iteration. This range would also be centered around the highest value tuple from the last round, to make sure the window was centered around the highest value portion of the state-space. An illustration of re-windowing from iteration 1 to 2 is shown in Fig. 5

With the state-space now dynamic, and hopefully narrowing to the correct misalignment angles, I decided to

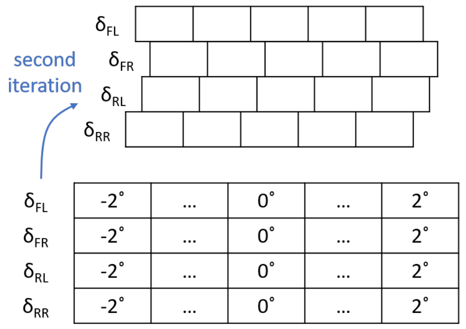

Fig. 5. Narrowed and re-centered discretization in each Value Est. iteration

increase the coarseness of the discretization such that each combination of candidate angles could be evaluated every iteration. Thus, each iteration would return a full tuple which maximized value. This process was repeated each iteration until convergence.

IV. Experimental Design and Results

To test and compare each algorithm, I conducted simulated experiments with a code archetecture designed to mimick an experiment on a real vehicle, namely X1. The psuedo code is as follows:

```
1) Setup: Initialize vehicle state, path, and four random
   misalignment angles.
2) Learning Loop:
   a) Simulate state forward,
      applying $\delta = \delta_{cmd} + \delta_{misalign}$
   b) Update $\hat{\delta}_{misalign}$ using
      Gradient Descent or Valiue Estimation
3) Validation Loop:
   a) Simulate foward,
      applying $\delta = \delta_{cmd} + \delta_{misalign}$ - $\hat{\delta}_{misalign}$
4) Compare $\hat{\delta}_{misalign}$ to $\delta_{misalign}$, and analyze model-
   mismatch performance.
```

To take this procedure from simulation to experiment on X1, take out "simulate forward" and replace with: Apply commands to X1 and record GPS sensor updates.

A. Gradient Descent Results

The following plots illustrate the learning phase and performance results for a typical Gradient Descent test, using the nonlinear model gradient. Given more time, I would like to record data for a large number of these tests and report statistics on its performance. Results do vary significantly among tests, especially based on the initial misalignment. I hope you the reader takes it on good faith that the individual test shown is nearly representative for the algorithm. I understand that results from one test do not constituate a fair or complete description of the algorithm's performance. However I am right up against the deadline and still making meaningful improvements to my code!

Fig. 6 shows how the estimate of the misalignment angles evolved over 100 iterations. The oscillations shown are an artifact of the underlying steering controller, which sweeps through steering angles to collect varied data for

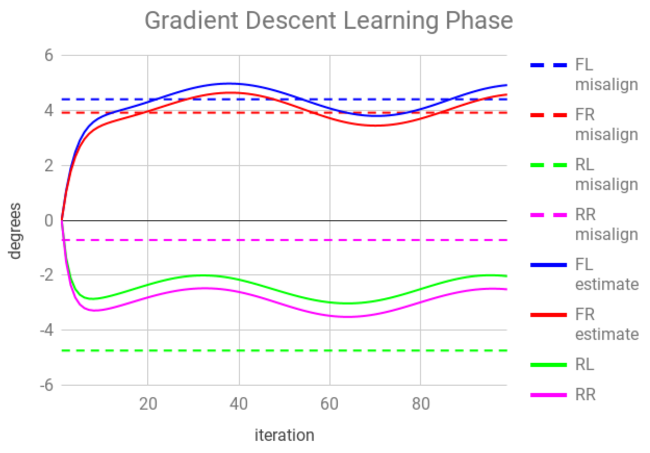

Fig. 6.

the algorithm to learn from. We can also see how tightly the estimates for the front wheels and the rear wheels are coupled. A common theme throughout this project has been difficulty in discerning effects betwen left and right; indeed the calculation posed by C. Kyrtsos's invention also cannot decide whether error exists on the left or right without trial and error. In this and most Gradient Descent tests, the front misalignments were found to be roughly an average of the FL and FR wheels' misalignments, and similar for the rear.

The algorithm was still able to make significant improvement, however, as illustrated by the performance metrics below. First we see in Fig. 7 that the vehicle initially strayed $12cm$ from the path before learning. After applying the

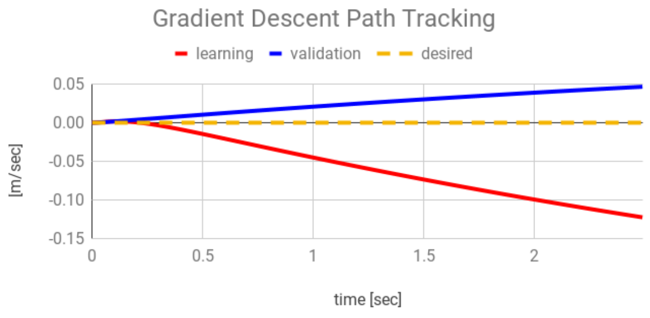

Fig. 7.

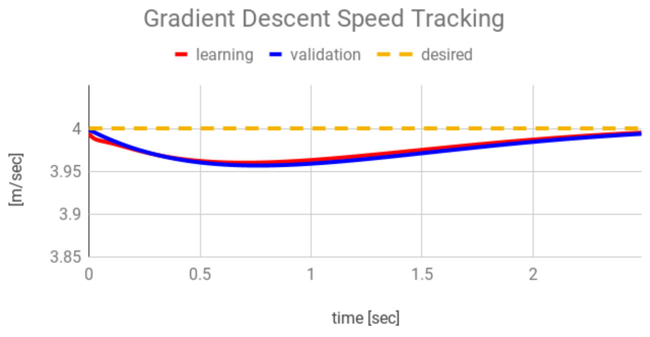

Fig. 8.

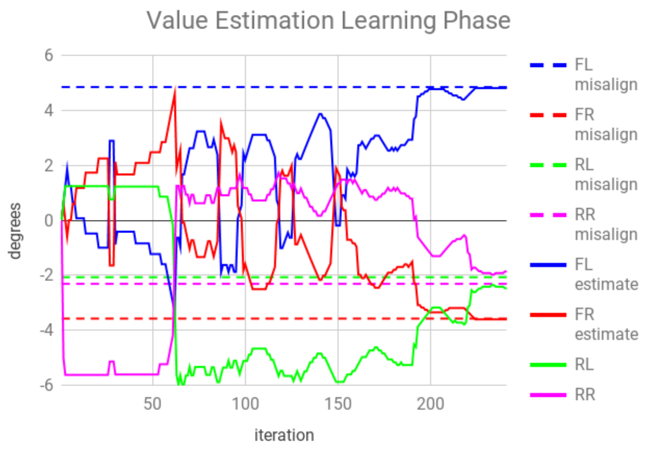

Fig. 9.

learned misalignments, the vehicle drifted in the opposite direction, but by only $5cm$. This overcorrection makes me wonder whether the algorithm would keep improving the correction if allowed to work once again after applying the first set of learned misalignments.

Next in Fig. 8 we see the speed of the vehicle following the path before and after learning. The initial loss of speed is due to misalignment friction. The proportional-integral speed conroller increases F_x over time to pull speed back up the desired rate. In this experiment basically zero improvement was made in speed tracking.

B. *Value Estimation Results*

Next we inspect an experiment from Value Estimation. As suggested by the noisy convergence above in Fig. 9, this algorithm is less consistent in finding the misalignment angles than Gradient Descent. While G.D. always makes finite but usually suboptimal performance, Value Estimation seems to hop wildly between choices and sometimes makes silly decisions. I believe this to be caused by the discretization; in experiments where I increase the number of bins, the performance is less noisy. However, the exponential increase in complexity quickly takes it from a potential real-time estimation to an offline program. Here we can also see tight coupling between left and right wheels. The FL and FR estimates often appear to mirror one another, as for the RL and RR.

Despite drama in the learning phase, the performance metrics are outstanding! Seeing this and similar results was a welcome surprise after many failures. In Fig. 10, the vehicle initially drifts right by $20cm$, but after applying the learned misalignment drive dead straight! In fact the maximum (simulated) drift in the blue validation curve is $.3\,mm$. In Fig. 11 we see another successful validation. The vehicle scrubs more than $10\,cm/sec$ of speed before correction, but loses less than $1\,mm/sec$ when the misalignment estimate was applied.

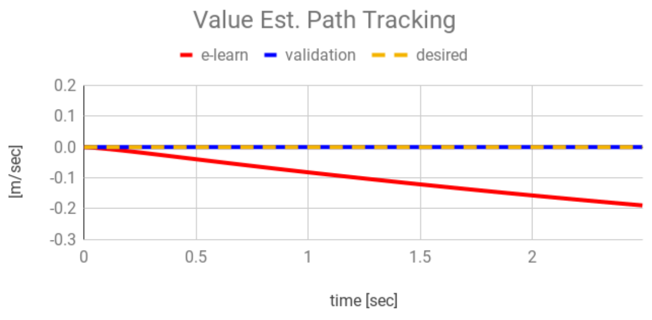

Fig. 10.

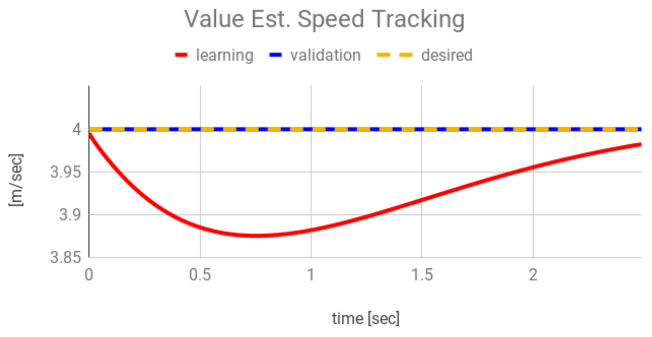

Fig. 11.

V. Conclusion

In conclusion, correcting wheel misalignment proved to be a challenging endeavour for my CS221 project. I attempted several iterations of Gradient Descent and Value Estimation, and clearly still have much progress to make before any strategy becomes ready for experiment or submitted to the U.S. Patent Office!

Two options for future development seem worthwhile. First, I'd like to build these algorithms into more of a reinforcement strategy, to learn from trial and error rather than simply learn from the misalignment's initial state. Second, I'd like to think about using steering motor current to influence my algorithm's learning. When a wheel is held out at an odd angle, X1's steering motors have to exert additional torque and current to maintain that angle. By measuring this current, I may be able to better learn which wheel is misaligned, and handle the left-right ambiguity issue.

The present and ongoing state of our code can found on GitHub at github.com/JohnHaTrick/autolign, in addition to the supplied code.zip. Thanks to the Alex and the CS221 staff for being very helpful. Sorry for all the typos - I'm right up against the buzzer!

Acknowledgent

I would like to thank several people who participated in this project: Masanori Ogihara, visiting scholar from Komatsu in Japan, who helped develop the vehicle model and simulator; Junwu Zhang, Mechanical Engineering MS student who developed his own MDP Q-Learning strategy to solve this problem for AA228/CS238; Jorge Cordero, Electrical Engineering MS Student who helped develop the data collection infrastructure.

References

[1] Bridgestone Tire Corp. "What You Need to Know About Tire Alignment." bridgestonetire.com/tread-and-trend/drivers-ed/tire-alignment
[2] K. Spann & P. Karez. "LASER Tracking Wheel Alignment Measurement Apparatus and Method." U.S. Patent No. 5,532,816. (1996).
[3] C. Kyrtsos. "Method and System for Detecting and Adjusting Wheel Misalignment of a Vehicle." U.S. Patent No. 6,275,753 B1. (2001).
[4] H. Bae & J. Ryu. "Diagnosis of Wheel Alignment Using GPS." U.S. Patent No. 8,706,347 B2. (2014).
[5] "Wikipedia: Gradient descent." wikipedia.org/wiki/Gradient_descent.
[6] "Wikipedia: Markov Decision Process." wikipedia.org/wiki/Markov_decision_process.