Checkpoint 1.2 Progress Report

Logan Zartman

Austin Atchley

1. Mathematical computation of our 1-D time optimal controller:

We use the following computation steps each control loop:

1. Time integration
    1. Compute straight-line distance traveled since last time step
    2. Compute velocity as distance / timestep size
    3. Update distance (total traveled) using explicit Euler integration
2. State prediction in 3 timeframes: last sensor read (S), predicted at current time (C), predicted at motor actuation (A)
    1. Let speed_S = norm(velocity)
    2. Let position_S = distance
    3. Compute speed_C and position_C using kinematics with last acceleration output and estimated sensor→control latency
    4. Compute speed_A and position_A using kinematics with estimated control→actuation latency
    5. Estimate stopping position using estimated state at actuation
3. Motion
    1. Check estimated stopping position against target position
    2. If stopping position >= target position then
        1. Compute required deceleration to stop at target
        2. Apply deceleration
    3. Else accelerate at max acceleration

2. Code organization:

Our code is largely organized the same way as checkpoint 1.1, but we did move some functionality to helper functions. Most of the new code is in the form of mathematical calculations that run each frame, called from `Navigation::Run()`. We perform the calculations described above according to the slides from class, but we did some measurement of our own to ensure that the latency values were optimal. We use the "Strategy 2" approach to latency compensation, meaning that we assume the car will continue to accelerate at the same rate as its last commanded acceleration. We predict the position and velocity using these values.

Instead of keeping track of odometer-measured speed, we collect the odometry position measurements and predict where the car will be during the control execution and motor actuation stages. We use these values to improve our accuracy on `time_to_stop` and to choose the correct acceleration value.

As latency compensation was the main objective on this checkpoint, that sums up most of the relevant work we have done so far.

3. Parameter tuning:

We performed basic parameter tuning for the VESC. We updated the speed to erpm gain based on a mismatch between encoder-reported distance and real distance. We calculated a scaling factor of 0.96 based on test runs, but found experimentally that a factor 0.975 produced more accurate distances.

We determined that error in travel distance was different when turning left than when turning right, so we decided to optimize for accurate straight-line motion. Of course, this extends well to curved motion, but the errors for each direction are slightly different.

The only tunable parameters in our 1D TOC computation are the latency values. Our calculations use two latency values: the latency between sensor data capture and control processing, and the latency between control and motor actuation. We chose to estimate that these values are the same, and used an initial guess of 0.1s. We refined this guess by increasing the latency until the car stopped short, and then decreasing as much as possible while maintaining a minimal error in the stop position. We tested first at a higher speed (2m/s max), and found that reducing the max speed did not significantly affect the error. We selected a final value of 0.085s for both sensor and actuation latency.

4. Challenges faced:

We had trouble deciding where and how to compensate for latency. We implemented an initial logical approach and debugged it to tune the values and where we should use each position/speed value. The most difficult aspect of this design was determining what kinds of latency exist and whether they could be measured. Rather than attempting to measure latency precisely, we made an initial guess and tuned it accordingly.

These challenges weren't major roadblocks though. We worked through them without being blocked for too long.

5. Contributions of each team member:

Austin - Initial approach and structure for latency compensation. Designated car driver. Refactoring.

Logan - Fine-tuning for latency compensation implementation and other parameters. Visualization and collection of data points.

State estimation code was pair-programmed with Logan committing.

6. GitHub Repo

https://github.com/austinatchley/F1-10-Autonomous-Driving

7. Video Demos

2 meters distance on pre-measured tape with 1m/s max velocity and 3m/s^2 max acceleration

https://drive.google.com/file/d/1-6I8ubXC_vUiwEjwKUGETv0ijHdRV1WZ/view?usp=sharing

2 meters distance with curvature of -1

https://drive.google.com/file/d/17qz0Eb2-Sf8woBZr_DeGcpqaZx4tWImE/view?usp=sharing

0.5 meters

https://drive.google.com/file/d/1vfNl40W3i84cjVTiD2gXZRp0lGstgpvQ/view?usp=sharing

Data for 2m straight-line travel at 1m/s:

Run	Encoder distance (m)
1	2.01172
2	2.00153
3	2.00435
4	2.00504

Average error: +0.00566m

8. Future improvements:

Sometimes the car lurches at the end because it hasn't quite reached the desired distance and has to accelerate again. This is due to incorrect estimates for latency values causing it to decelerate too aggressively.

In certain outlier runs, the car ends up passing over the mark slightly. It is unclear why this occurs. It is still within a margin of a few centimeters, but we would like to make stopping as smooth as possible.

Our code structure isn't exactly robust yet, but we will refactor to build abstraction when we need it. In the future checkpoints, we will improve this aspect of our project.