FEEDBACK CONTROL SYSTEMS FINAL REPORT CARAVAN CONTROL

TUCKER HAYDON AND CONNOR BRASHAR

Abstract. A fleet of self-driving trucks is simulated on a long, flat, straight piece of road. The fleet is brought into a 'caravan' formation within ten minutes via a discrete-time, finite horizon LQG controller. The trucks have limited sensing capabilities: GPS for the lead truck and range sensors between all following trucks. Finally, a hands-on museum exhibit mirroring the simulation and demonstrating autonomy and control concepts is proposed.

1. Scenario

You are an engineer in charge of designing and implementing an estimation and control system for a fleet of self-driving semi-trucks. Due to government regulation, the truck fleet can only operate in the self-driving mode when they are on long, straight stretches of highway between cities. In order to reduce costs and conserve fuel, the trucks drive very closely together at a pre-determined speed in 'caravan formation'. By driving very closely together, the trucks can draft off of one another, reduce drag, and save fuel by upwards of 21% [1].

For the system you are designing, only three trucks will be in the caravan. The caravan is equipped with the following sensor suite:

(1) The lead truck is equipped with a GPS receiver that measures its position at 1 Hz.
(2) The two following trucks are equipped with range sensors that measure the relative position between themselves and the truck in front of them at 10 Hz.

Each truck may be independently controlled and may be instructed to either accelerate or decelerate.

Date: December 6, 2018.

2. Nominal System Description

Define the system state.

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} \tag{1}$$

Define the system input.

$$\vec{u} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \tag{2}$$

Define the system dynamics.

$$\dot{\vec{x}} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{A} \vec{x} + \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{B} \vec{u} \tag{3}$$

Define the system observer equation.

$$\vec{y} = \begin{bmatrix} x_1 \\ \Delta x_{12} \\ \Delta x_{23} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{C} \vec{x} \tag{4}$$

2.1. **Observability.** Using the nominal system, determine whether or not the system is observable. The nominal system is linear and time-invariant, so the observability Grammian is simplified:

$$W_o = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix} \tag{5}$$

If the rank of $W_o = n$ then the system is observable. **The nominal system is observable**.

2.2. **Controllability.** Using the nominal system, determine whether or not the system is controllable. The nominal system is linear and time-invariant, so the controllability Grammian is simplified:

$$W_c = \begin{bmatrix} B & AB & A^2B & \dots & A^{n-1}N \end{bmatrix} \tag{6}$$

If the rank of $W_c = n$ then the system is controllable. **The nominal system is controllable**.

2.3. **Conclusion.** The fact that the nominal system is both observable and controllable indicates that it is well-designed and amenable to both estimation and control.

2.4. **Discrete-Time Dynamics.** Software implementations of continuous kinematic systems must first discretize the system dynamics. Many sensors and computers cannot produce nor consume continuous signals. As a result, system dynamics or measurements are represented as discrete systems.

Define the discrete system.

$$\begin{aligned} x_{k+1} &= A_d x_k + B_d u_k \\ y_k &= C x_k + D u_k \end{aligned}$$

For a discretized system, only the system dynamics matrices (A, B) change:

$$A_d = e^{A \cdot \Delta T}$$

$$B_d = \int_0^{\Delta T} e^{A\lambda} d\lambda B$$

where ΔT is the discrete time interval. For this project, ΔT was chosen to match the smallest measurement interval, $\Delta T = 0.1$.

These discrete-time dynamics are used in both the estimator and the controller.

3. Estimator

Estimators are designed to predict each state in a system, if that state information is not directly observable. Often, measurements are taken that contain indirect information about a state. For example, accelerometer measurements must be integrated twice to get a position measurement. Naturally, there is some noise and inaccuracy in this process, but provided the system is fully observable, a Kalman filter can be implemented on a linear system to get direct estimates of full state variables from measurement data.

Kalman filters are a simple form of estimator that works on linear systems of the form:

$$\begin{aligned}\dot{x} &= Ax + Bu + v\\ z &= Hx + w\end{aligned}$$

where v and w are respectively random process and measurement noise, and $H = C$. We note that the Kalman filter assumes no user input related to the measurement. Because user input is known and prescribed by the controller, we don't wish to include it in our states that we are estimating. As a result, our Kalman Filter will not try to estimate acceleration, only position and velocity. The Kalman filter model is sufficient for this caravan problem.

A Kalman filter operates in two steps.

3.1. **A Priori Measurements.** In the first step, the Kalman filter performs an *a priori* estimate, through which it produces an estimate of the state at the current discrete time step based only on the previous time step by forward-simulating the system dynamics:

$$\begin{aligned}\bar{x}_k &= A_d \hat{x}_{k-1} + B_d u_k\\ \bar{P}_k &= A_d \hat{P}_{k-1} A_d^T + Q_k\end{aligned}$$

Here, the P matrix is the covariance matrix of the system's states, while Q is a measure of the system's process noise. Discrete-time process noise is modeled as follows:

$$Q_k = \begin{bmatrix} \frac{T^5}{20} & \frac{T^4}{8} & \frac{T^3}{6} \\ \frac{T^4}{8} & \frac{T^3}{3} & \frac{T^2}{2} \\ \frac{T^3}{6} & \frac{T^2}{2} & T \end{bmatrix} \tilde{q}$$

Where T is the period between each measurement, here defined as 0.1s, and $\tilde{q}$ is the power spectral density (PSD) of noise for each component. For our system, we used relatively low process noise with a PSD of 1/10 for position, and 1/20 for velocity. We also note that the values of A_d and B_d are the discrete-time system dynamics, which are covered in section 2.4.

3.2. **A Posteriori Measurements.** Next, the Kalman filter takes new measurements into the system and incorporates them. The Kalman filter creates a model for measurements and compares it to the actual measurement data it receives, and weights it's *a priori* estimate with the new information to create a new, *a posteriori* measurement. The *a posteriori* measurement update equation:

$$\begin{aligned}
\bar{z}_k &= H\bar{x}_k \\
P_{zz} &= H\bar{P}H^T + R_k \\
W &= \bar{P}H^T P_{zz}^{-1} \\
\hat{x}_k &= \bar{x}_k + W(z_k - \bar{z}_k) \\
\hat{P}_k &= \bar{P} - WP_{zz}W^T
\end{aligned}$$

This *a posteriori* formula can take many forms, and the form above is known as the Joseph's form. The reason it is used in lieu of other forms is that the Joseph's form equation copes well with sparse transition matrices that may lead to low precision inversion of the P_{zz} term. As a result, the Joseph's form works best for this system, which has some very sparse matrices.

The final result, $\hat{x}$, is the most accurate version of the system states based on both a model of how these states grow, and measurements that relate to those states.

3.3. **Different Measurements at Different Times.** For our system, two measurements were taken: position of the first vehicle, and range between each vehicle in the caravan. These measurements arrived at different time (1 Hz & 10 Hz), but the Kalman filter described above assumes that both measurements arrive simultaneously, however this does not occur in our system. We wish to combine both measurements into the Kalman filter as soon as these data arrive.

There are two ways to accommodate measurements taken at different times in a Kalman filter:

(1) Set the covariance of missing measurements infinity
(2) Change the size of the matrix H_k at each step to correspond only with the present measurements.

This latter method was chosen for this Kalman filter. Changing H_k was not difficult, while there were numerical issues with an infinite covariance.

3.4. **Simulation.** The Kalman filter was run both with and without control input to ensure that it was working. In both cases, the filter's error between estimated state and true state was driven to zero within two minutes. A plot of the estimation error and variances of estimated state positions is given below.

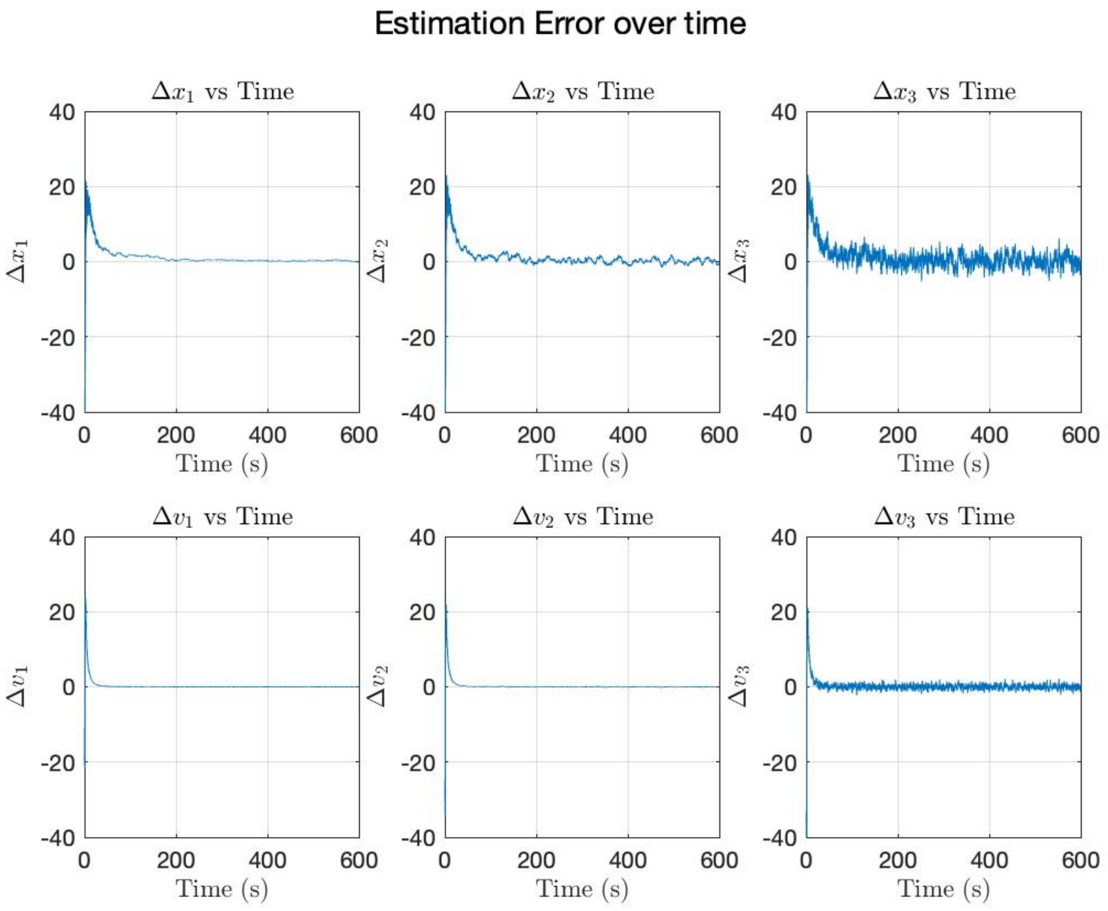

FIGURE 1. Estimation Error

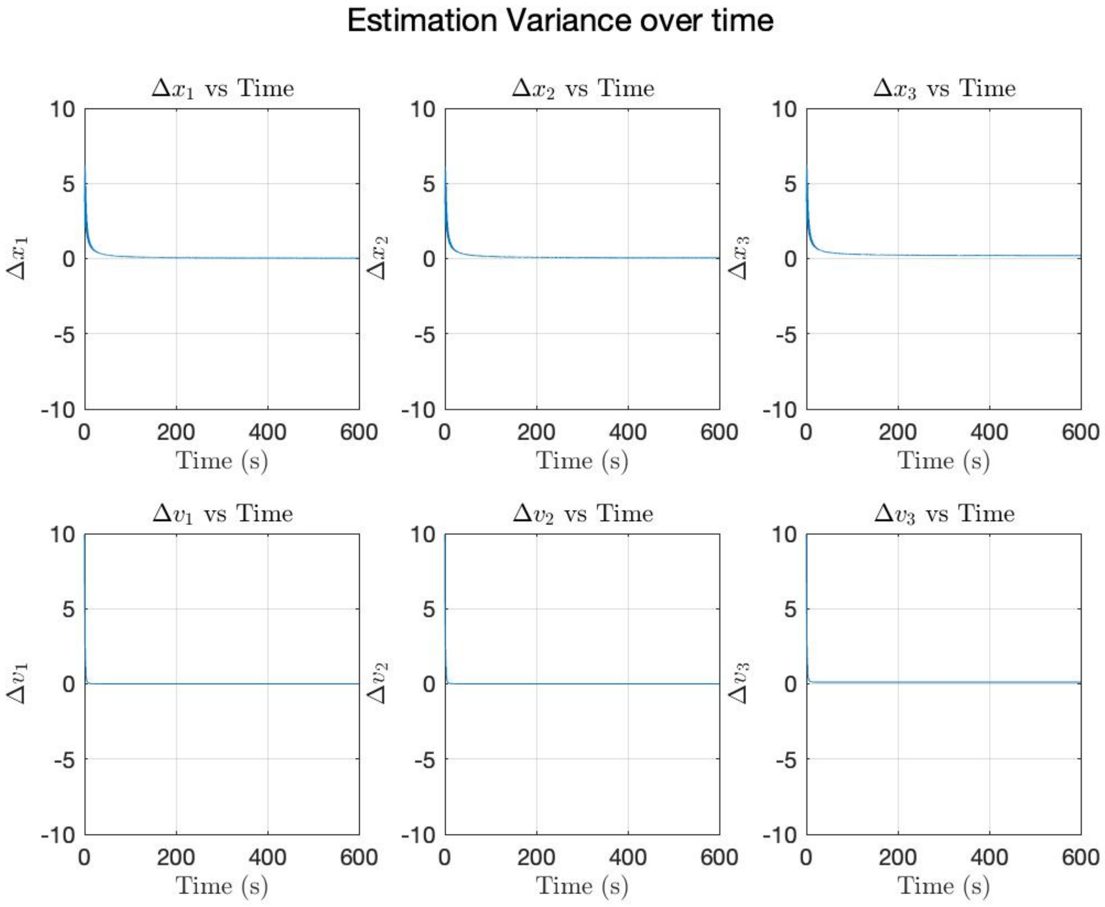

Figure 2. Estimated State Variances

As can be seen, the estimation error is driven to zero quickly, and remains relatively stable throughout the system. In a few rare instances, a noisy measurement of vehicle position bounced the error up for a second or two, but these errors were quickly accommodated by the Kalman filter.

Note the increasing variance of state error from trucks one to three. The Kalman filter's estimate decreased in accuracy along the extent. This makes sense, given that only the first vehicle has direct position measurements, and subsequent position estimates must be derived from the combination of that measurement with other additional noisy measurements. The noisy sensor measurements cascade down the caravan, decreasing the accuracy of the estimate along the way.

4. Control System Description

4.1. **Error State System.** Controllers drive systems to the origin. The nominal system, however, should not be driven to the origin. Instead an error-state system should be specified where the error is the deviation of the nominal system from a reference signal.

First, define the reference signal. From the problem statement, the goal is to maintain a specified distance between the trucks and to keep that at a constant velocity.

$$\vec{x_r} = \begin{bmatrix} x_{1_r} \\ \Delta x_{12_r} \\ \Delta x_{23_r} \\ v_{1_r} \\ v_{2_r} \\ v_{3_r} \end{bmatrix} \tag{7}$$

The error signal is the difference between the nominal and the reference states. Note that not all states have a corresponding reference signal.

$$\vec{x_e} = \begin{bmatrix} x_1 - x_2 - \Delta x_{12_r} \\ x_2 - x_3 - \Delta x_{23_r} \\ v_1 - v_{1_r} \end{bmatrix} \tag{8}$$

The error signal dynamics don't conform to the standard $\dot{x} = Ax + Bu$ form — there is an affine transform between the error dynamics and the nominal and reference signals. However, the standard form can be composed via the following trick: append the reference signal to the nominal state and define zero dynamics and full observability for the reference substate.

Stack the nominal and reference states into the *stacked state.*

$$\vec{x_s} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ v_1 \\ v_2 \\ v_3 \\ \Delta x_{12_r} \\ \Delta x_{23_r} \\ v_{1_r} \end{bmatrix} \tag{9}$$

Define the stacked state dynamics.

$$\dot{\vec{x}}_s = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ a_1 \\ a_2 \\ a_3 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{A_s} \vec{x}_s + \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{B_s} \vec{u} \tag{10}$$

Define the stacked state observer equation. Note that the reference states are specified and fully known so there is full state feedback for these states.

$$\vec{y} = \begin{bmatrix} x_1 \\ \Delta x_{12} \\ \Delta x_{23} \\ a_1 \\ a_2 \\ a_3 \\ \Delta x_{12_r} \\ \Delta x_{23_r} \\ v_{1_r} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}}_{C_s} \vec{x}_s + \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{D_s} \vec{u} \tag{11}$$

Now define the error state by linearly transforming the stacked state via a *similiarity transform.*

$$\vec{x}_e = \begin{bmatrix} x_1 \\ \Delta x_{12_e} \\ \Delta x_{23_e} \\ v_{1_e} \\ v_2 \\ v_3 \\ \Delta x_{12_r} \\ \Delta x_{23_r} \\ v_{1_r} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}}_{T} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ v_1 \\ v_2 \\ v_3 \\ \Delta x_{12_r} \\ \Delta x_{23_r} \\ v_{1_r} \end{bmatrix}$$

Now, leverage the similarity transform to define the error state dynamics.

$$\dot{\vec{x}}_e = TA_sT^{-1}\vec{x}_e + TB_s\vec{u}$$
$$y = CT^{-1}\vec{x}_e + D\vec{u}$$

Finally, clean up the equation by redefining the system.

$$\begin{aligned}\vec{x} &:= \vec{x}_e\\ A &:= TA_sT^{-1}\\ B &:= TB_s\\ C &:= C_sT^{-1}\\ D &:= D\end{aligned}$$

Now the system is in the standard form.

$$\begin{aligned}\dot{\vec{x}} &= A\vec{x} + B\vec{u}\\ y &= C\vec{x} + D\vec{u}\end{aligned}$$

4.2. **Discrete-Time, Finite Horizon LQR Controller.** A discrete-time, finite horizon LQR controller was chosen to bring the trucks into 'caravan' formation. Given a transient state weighting matrix, Q, a final state weighting matrix, Q_f, and an input weighting matrix, R, LQR controllers minimize the cost function:

$$J = \vec{x}_f^T Q_f \vec{x}_f + \sum_{k=0}^{N} \left[\vec{x}_k^T Q \vec{x}_k + \vec{u}_k^T R \vec{u}_k\right]$$

The control input at every time, u_k, is determined via dynamic programming. A proof of optimality is not provided in this report but is outlined in [2]. u_k is determined as follows:

(1) Set $P_N = Q_f$
(2) For $t = N, \ldots, 1$: $P_{t-1} = Q + A^T P_t A - A^T P_t B(R + B^T P_t B)^{-1} B^T P_t A$
(3) For $t = 0, \ldots, N-1$: $K_t = -(R + B^T P_{t+1} B)^{-1} B^T P_{t+1} A$
(4) For $t = 0, \ldots, N-1$: $u_t = K_t x_t$

At the beginning of the horizon, the entire LQR control trajectory is pre-computed and the respective gain matrixes cached. For the duration of the control period, the optimal control input is the product of the time-respective gain matrix and the current state.

4.3. **LQR Simulation.** The LQG controller was implemented in Matlab. The goal of the controller is to bring the trucks into the 'caravan' formation within 10 minutes. The trucks should follow each other at 30 meters/second with a separation of only 5 meters.

$$\vec{x}_0 = \begin{bmatrix} x_1\\ x_2\\ x_3\\ v_1\\ v_2\\ v_3 \end{bmatrix} = \begin{bmatrix} 200\\ 75\\ 0\\ 30\\ 27\\ 25 \end{bmatrix}$$

$$\vec{x}_r = \begin{bmatrix} \Delta x_{12} \\ \Delta x_{23} \\ v_1 \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \\ 30 \end{bmatrix}$$

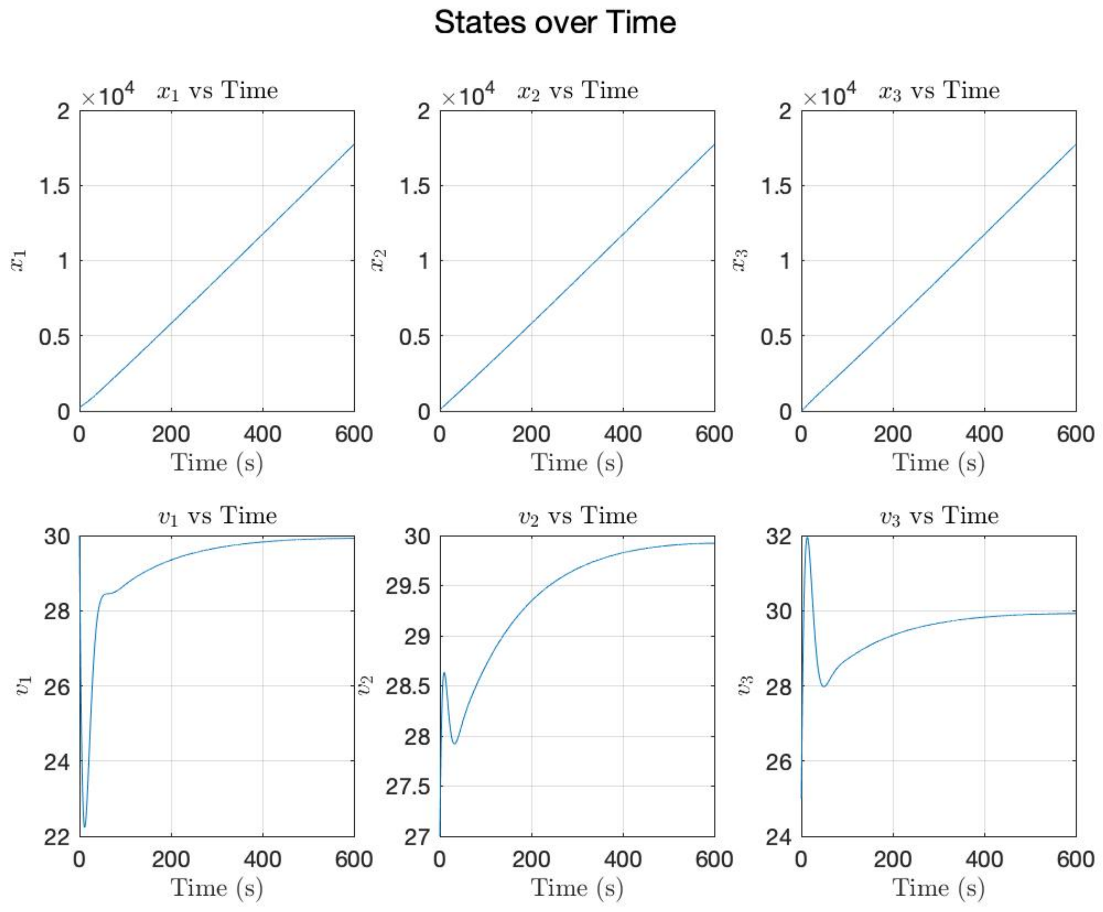

FIGURE 3. States Over Time with Full State Feedback

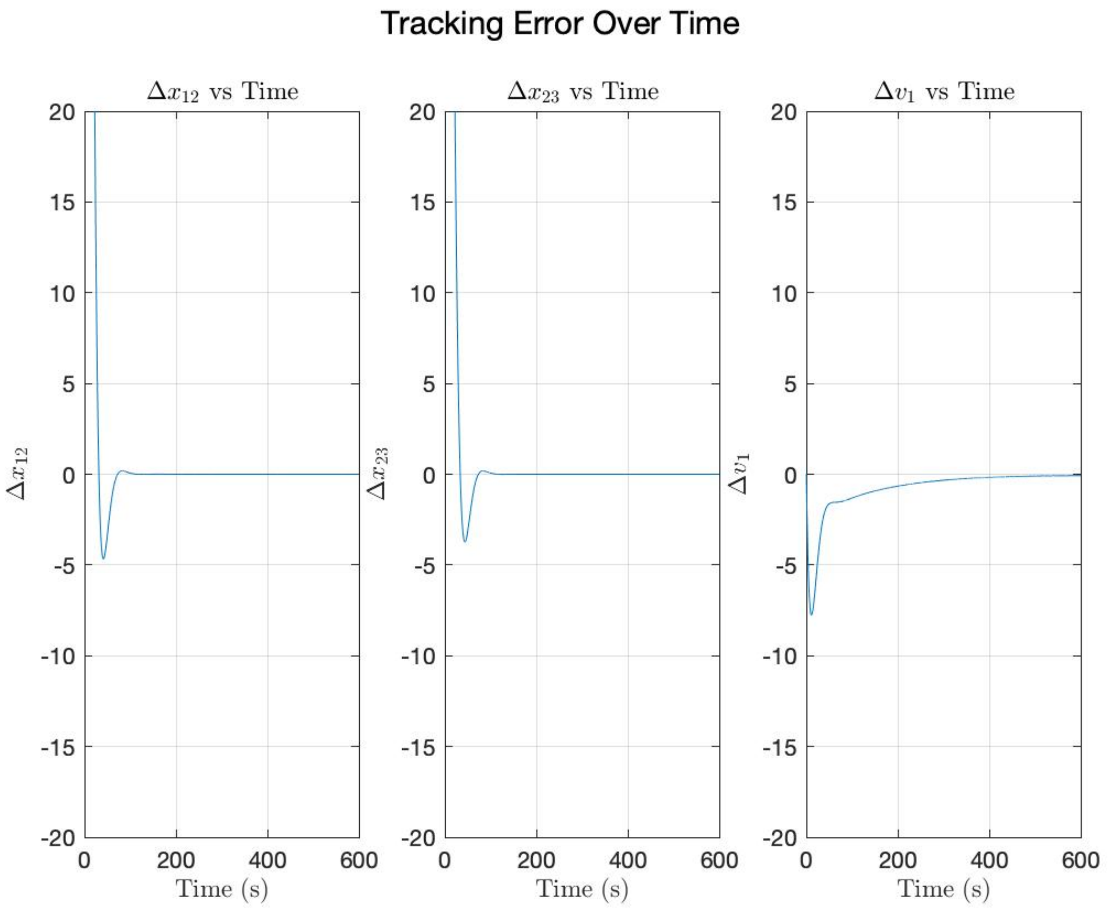

FIGURE 4. Tracking Error Over Time with Full State Feedback

The LQR controller is optimal for choices of Q_f, Q_k, and R_k, but it does not guarantee or follow any input or state constraints. This may cause problems for real-life systems. For example, in the simulation above, the first truck abruptly brakes and drops from 30 m/s to 22 m/s almost immediately. Not only is this physically impossible, but it would be very unsafe on a freeway! Moreover, from the error graph, one can see that the second truck nearly rams the first truck (error of -5m implies a crash). This LQR controller was hand-tuned to prevent that from happening, but there are no hard mathematical constraints preventing that from happening in a real system.

LQR controllers can admit state and input constraints, but the controllers become more complicated. For this project, no constraints were implemented.

4.4. **LQG Simulation.** Together, the Kalman Filter described in section 3 and the LQR controller described in section 4.2 form a Linear-Quadratic-Gaussian controller. The state estimate from the Kalman filter replaces the true state, but optimality is guaranteed via the Separation Principle.

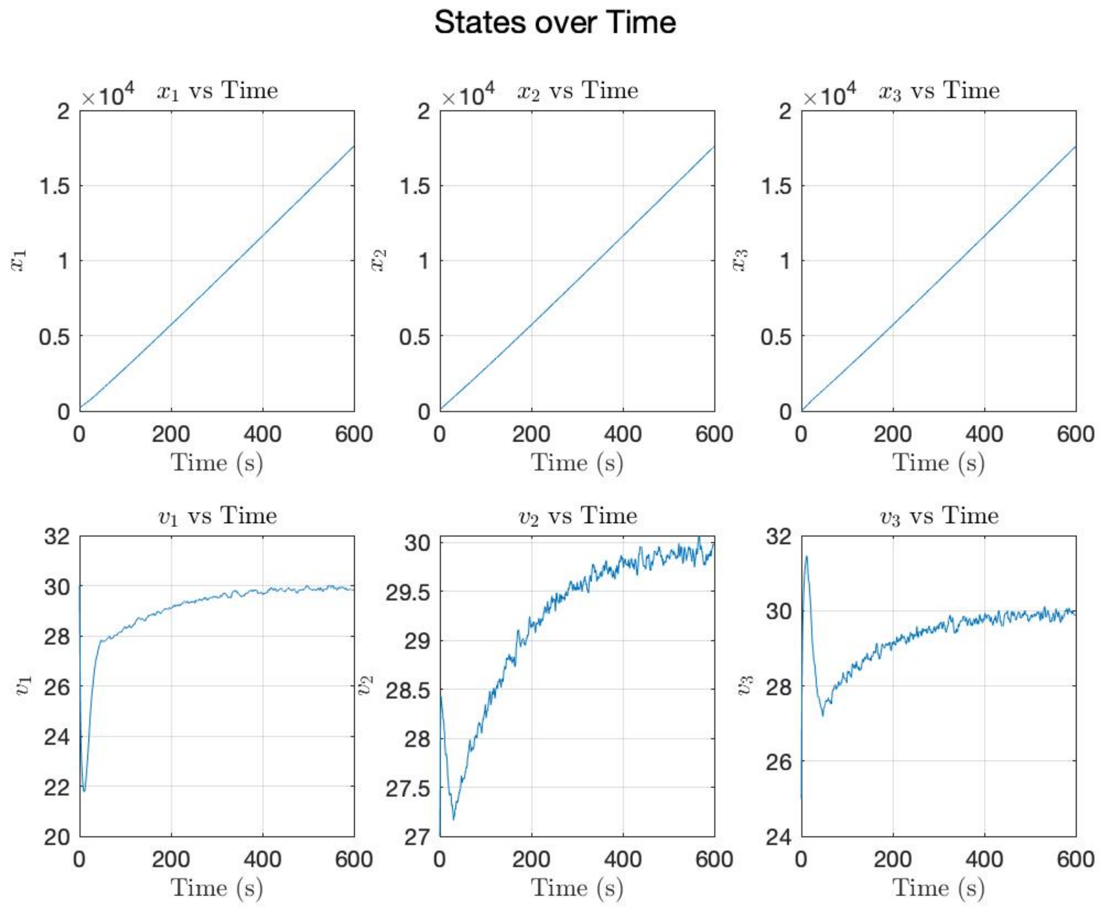

FIGURE 5. States Over Time with Estimated State

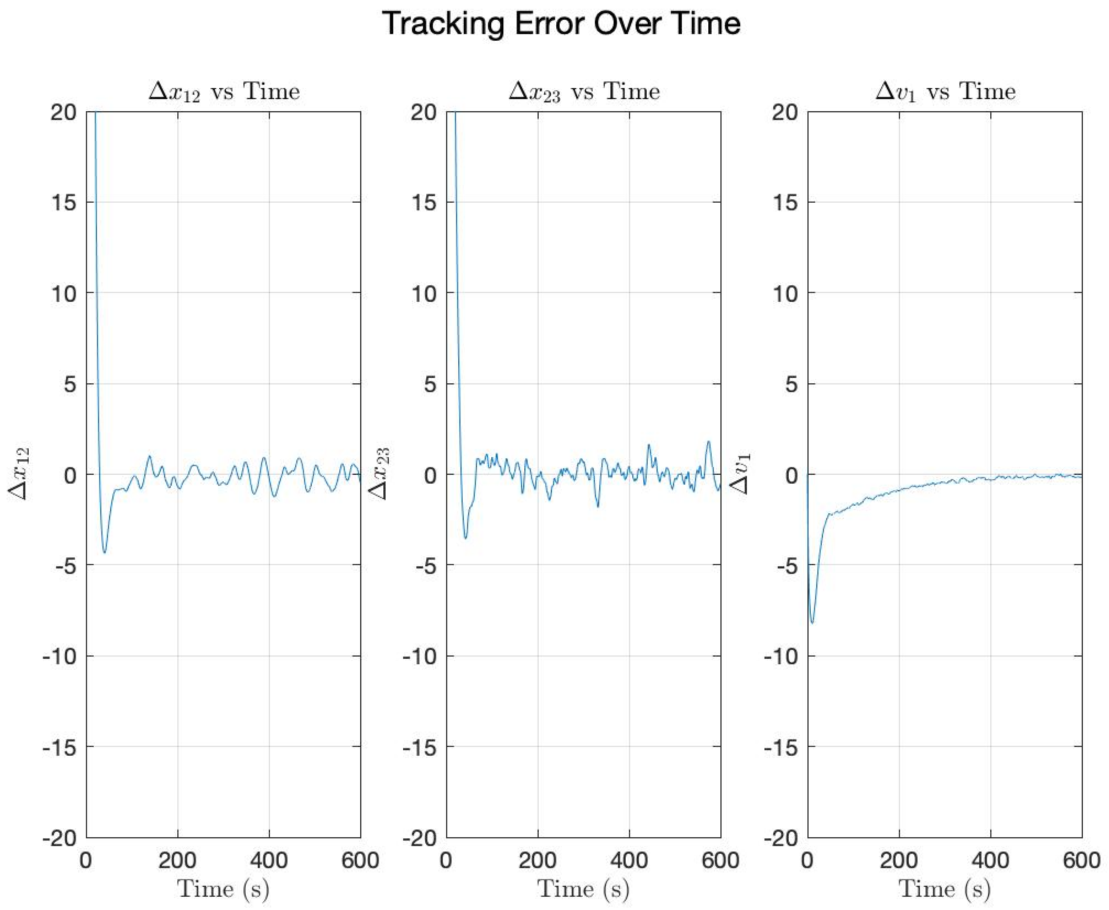

Figure 6. Tracking Error with Estimated State

Clearly the system is tracking and the trucks enter the caravan formation. However, the tracking performance is noticeably worse than the LQR controller. The LQR controller was designed assuming that full state would be known, but in fact, only a noise estimate is available so the performance is worse.

The LQG controller suffers from the same lack of state and input constraints as the LQR controller, but worse! The trucks nearly ram each other and have a rather high-frequency steady-state error. Few passengers would want to ride in these trucks!

For a safer autonomous truck system, both state and input constraints would have to be implemented. Moreover, the state of the system may be expanded to include acceleration/jerk estimates and constraints on these states may also be required to ensure passenger comfort. More sensors and rigorous testing/simulation may be required to reduce state uncertainty and increase robustness against sensor noise/outages.

5. Museum Exhibit

The following section describes a potential museum exhibit aimed at high-school and middle-school children.

Note that this section is an abridged version of the original project proposal. In discussion with Dr. Tanaka, we agreed that the primary portion of the project would be the design, implementation, and simulation of an LQG controller and that, to make grading easier, we would try to conform to the original project proposal.

5.1. **Description.** We propose a museum exhibit composed of toy trains on tracks where visitors attempt to tune an LQR controller to bring three independent locomotives into the caravan formation. On a 3 meter by 3 meter tables, toy train tracks are laid out in a wide circle [1]. Three battery-powered locomotives may be placed anywhere on the tracks. Visitors may turn 6 knobs corresponding to the 6 diagonal entries of the LQR weighting matrices to tune the system.

5.2. **Implementation.** Visitors may walk around all sides of the table. On one side of the table are six knobs that visitors may twist to tune the LQR controller. On the far end of the table opposite the knobs is a TV. The TV first shows a short 30-second video describing the goal of the exhibit, what a 'good' controller looks like, and what each of the 6 knobs do. Visitors are then free to twist the knobs (a digital value is displayed on the screen), and then press the 'go' button.

Behind the scenes, a Raspberry Pi B+ (RPB) controls the three locomotives. The locomotives have been 'hacked' and carry a Raspberry Pi Zero (RPZ) that modulates the supplied battery voltage to each of the locomotives. By modulating the battery voltage, the RPZ can directly control the speed of each of the locomotives.

A camera is suspended above the table and connected to the RPB. The RPB uses an image processing algorithm on the incoming camera data to track the location of each of the locomotives and generate the position and position difference measurements. The RPB fuses the camera measurements in a Kalman filter described in section 3 to generate position and velocity estimates.

After the 'go' button is pressed, the RPB reads in the current analog knob values and solves for optimal control input values. Once solved, the RPB streams control input commands to the RPZs via wifi. The RPZs convert the control input into motor voltages and drive the locomotives.

If the camera detects that two locomotives have touched (i.e. a crash has occurred), the trains are stopped, and the visitor is notified of the crash and given another chance.

[1] The hope is that a wide circle will approximately represent a long, straight, endless track

5.3. **Price Breakdown.**

Item	Total Price
2 Wooden Train Track	$100
3 Battery-Powered Locomotives	$150
1 Raspberry Pi B+	$40
1 Wifi Dongle	$15
1 Raspberry Pi Camera Module	$25
3 Raspberry Pi Zero W	$30
3 PWM Motor Shields	$120
1 LCD TV	$300
6 Knobs and 1 button	$50
Total Price	$830

6. Work Contribution

Both Connor Brashar and Tucker Haydon contributed equally to this project.

References

[1] C. Bonnet and H. Fritz, "Fuel consumption reduction in a platoon: Experimental results with two electronically coupled trucks at close spacing," tech. rep., SAE Technical Paper, 2000.

[2] "Linear quadratic regulator: Discrete-time finite horizon." `https://stanford.edu/class/ee363/lectures/dlqr.pdf`. Accessed: 2018-12-05.

The University of Texas at Austin

E-mail address: `thaydon@utexas.edu, connor.brashar@utexas.edu`