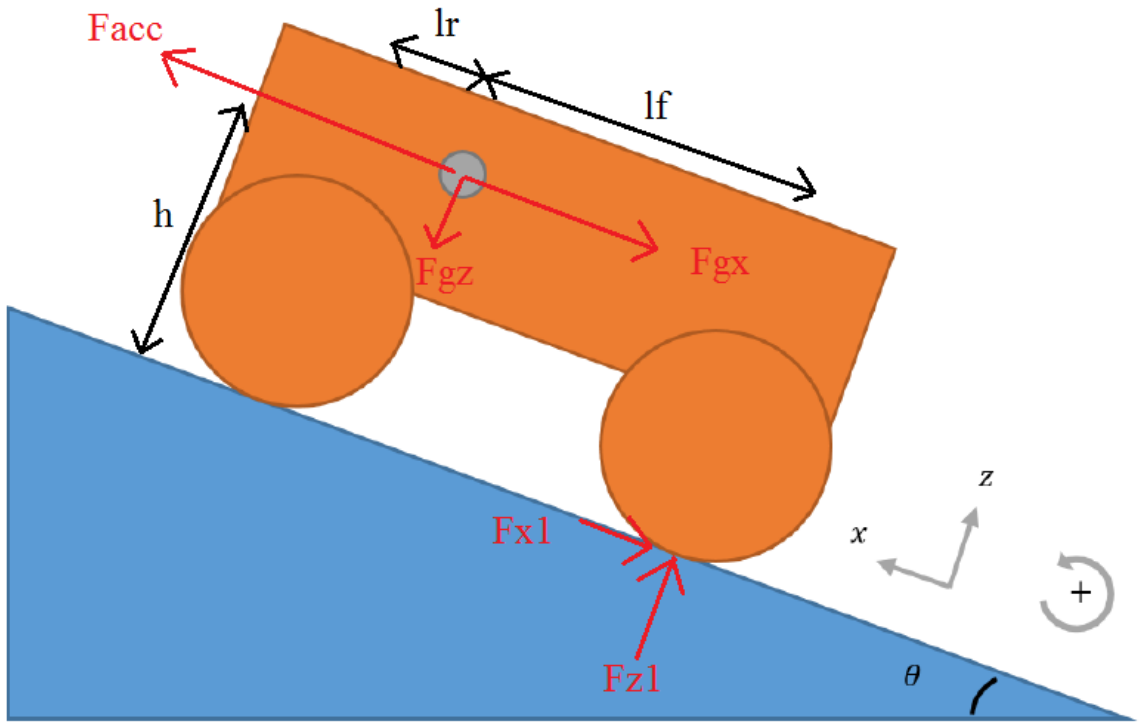
MATLAB code included in Appendix

1.1. Since rear wheel is leaving the ground, there are no forces on the rear wheel



1.2. Forces and moment equation:

$$\sum F_x = m\dot{v} \rightarrow -F_{x1} - F_{gx} = m\dot{v}$$
$$\sum F_z = 0 \rightarrow F_{z1} - F_{gz} = 0$$
$$\sum M = 0 \rightarrow F_{z1} l_f + hF_{x1} = 0$$

Using the F_z equation:

$$F_{z1} = F_{gz}$$

Using the M equation:

$$F_{x1} = -\frac{l_f}{h} F_{z1}$$

Combining the two equations above:

$$F_{x1} = \left(-\frac{l_f}{h}\right) F_{gz}$$

Plugging that into the F_x equation:

$$m\dot{v} = -F_{x1} - F_{gx}$$

$$= \left(\frac{l_f}{h}\right) F_{gz} - F_{gx}$$

Using trigonometry, we know:

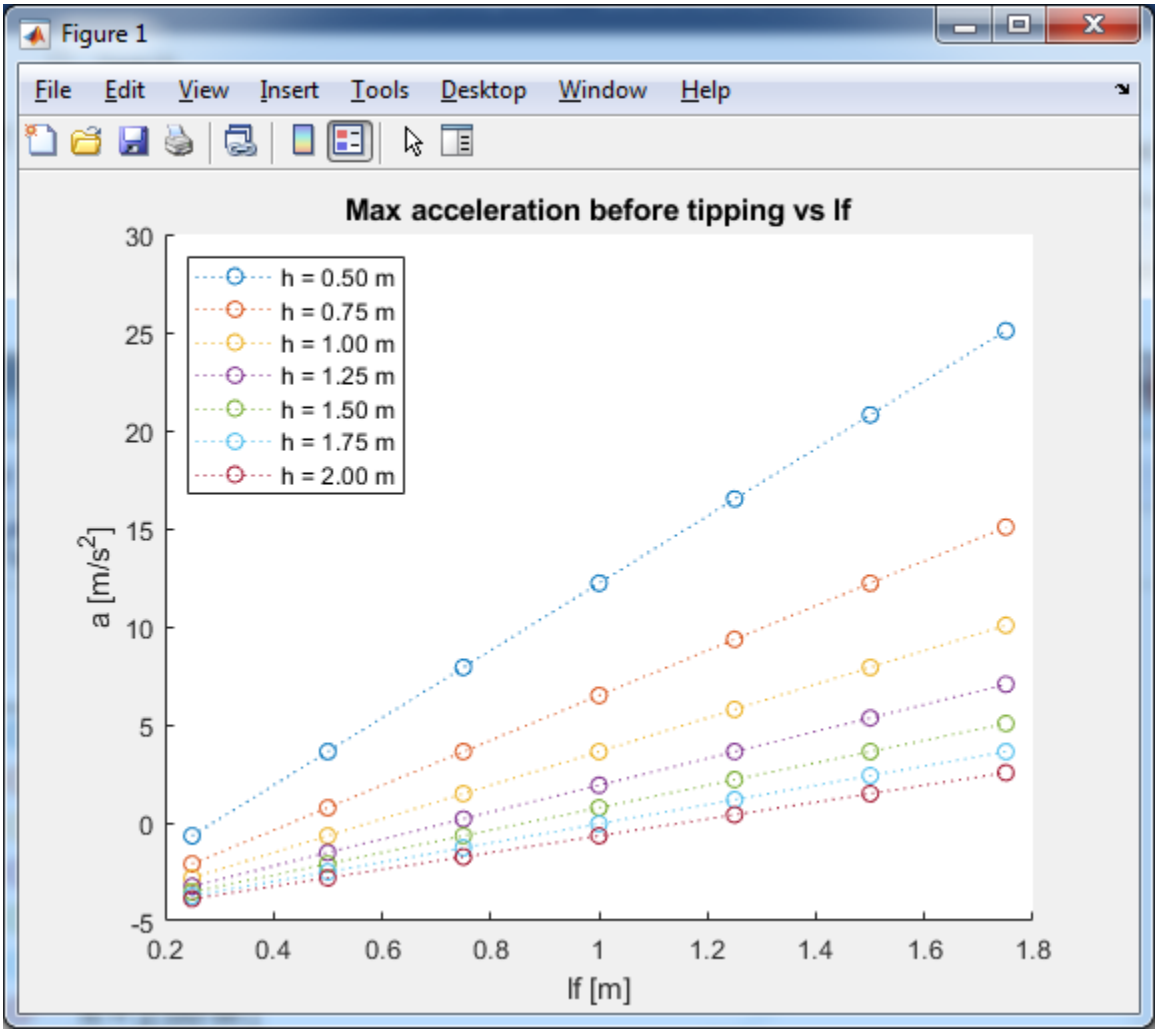
$$F_{gx} = mg \sin\theta \, , F_{gz} = mg \cos\theta$$

Thus, we get:

$$m\dot{v} = mg\left(\frac{l_f}{h}\cos\theta - \sin\theta\right)$$

$$\dot{v} = g\left(\frac{l_f}{h}\cos\theta - \sin\theta\right)$$

1.3.



1.4.

A. The lowest h (0.5 m) and the largest lf (1.75m) gave the greatest allowable deceleration before tipping.

B. This is what I expected. Intuitively, a lower center of gravity makes the vehicle harder to tip, so the lowest h makes sense. Since we're trying to keep the rear wheel from lifting, it makes sense the best way to prevent this is to put the entire vehicle's weight (its center of gravity) as close to the rear wheel as possible. This would make lr very small, and make lf very big.

C. It would depend on what I was designing the vehicle for, but probably not. A low h is good for stability, but it also makes the vehicle unusable in rough terrain; it would only work well for flat roads, where you don't have to worry about rocks or boulders lifting the car up. A large lf is good for this scenario, where we're trying to prevent the rear wheels from lifting, but it also makes lifting the front wheels very easy. For instance, if lf is large and the car is going uphill, there's a very high chance the vehicle will flip due to the front wheels lifting. For this reason, I probably wouldn't want to use the h and lf calculated here (unless I know the vehicle will ONLY go downhill, and will ONLY have to deal with flat surfaces; in that case these values are good)

2.1. Compaction resistance for rigid wheel:

$$R_{c(rw)} = \frac{\left(\frac{3F_z}{\sqrt{d_w}}\right)^{\frac{2n+2}{2n+1}}}{(3-n)^{\frac{2n+2}{2n+1}}(n+1)\left(k_c + bk_\varphi\right)^{\frac{1}{2n+1}}}$$

Plugging in $n = 0.5$:

$$R_{c(rw)} = \frac{\left(\frac{3F_z}{\sqrt{d_w}}\right)^{\frac{3}{2}}}{(2.5)^{\frac{3}{2}}(1.5)\left(k_c + bk_\varphi\right)^{\frac{1}{2}}} = F_z^{\frac{3}{2}} * \frac{\left(\frac{3}{\sqrt{d_w}}\right)^{\frac{3}{2}}}{(2.5)^{\frac{3}{2}}(1.5)\left(k_c + bk_\varphi\right)^{\frac{1}{2}}} = F_z^{\frac{3}{2}} * k$$

Note that k does not depend on F_z, so it will be the same for both the 4-wheeled and 6-wheeled vehicle. Therefore, when computing the ratio, k will cancel out.

For the 4-wheeled vehicle, $F_z = W/4$, while for the 6-wheeled vehicle, $F_z = W/6$. To get a ratio for total compaction resistance, we must multiply the per-wheel compaction resistance by 4 for the 4-wheeled vehicle, and by 6 for the 6-wheeled vehicle:

$$Ratio = \frac{4 * R_{c(rw,4)}}{6 * R_{c(rw,6)}} = \frac{4\left(\frac{W}{4}\right)^{\frac{3}{2}} * k}{6\left(\frac{W}{6}\right)^{\frac{3}{2}} * k} = \frac{2\left(\frac{1}{4}\right)^{\frac{3}{2}}}{3\left(\frac{1}{6}\right)^{\frac{3}{2}}} = 1.225$$

Therefore, the 4-wheeled vehicle has 1.225 times more total compaction resistance than the 6-wheeled vehicle. This means 6 wheels are better for minimizing compaction resistance, given both vehicles weigh the same.

2.2. We get the following ratio for soil thrust:

$$Ratio = \frac{4 * H_4}{6 * H_6} = \frac{4}{6} * \frac{cA * 1.2 + \frac{W}{4}\tan\varphi}{cA + \frac{W}{6}\tan\varphi} = \frac{cA * 4.8 + W\tan\varphi}{cA * 6 + W\tan\varphi} = \frac{cA * 4.8 + k}{cA * 6 + k}$$

$k = W\tan\varphi$, which is some constant that is the same for the numerator and denominator; for this reason, it will not change whether the ratio is greater than or smaller than unity (it will affect how close or far the ratio is from unity, but that's not important for this question). Focusing on the other terms, we see that $cA * 4.8 < cA * 6$, which means the ratio will be smaller than unity. Since the ratio is less than 1, this means the 4-wheeled vehicle has less soil thrust than the 6-wheeled vehicle. Therefore, the 6-wheeled vehicle is better for maximizing soil thrust.

2.3. For the 6-wheeled vehicle, with only compaction resistance, for a single tire:

$$DP = H - R_{c(rw,6)} = \left(cA + \frac{W}{6}\tan\varphi\right) * f(s) - \frac{\left(\frac{W}{2\sqrt{d_w}}\right)^{\frac{2n+2}{2n+1}}}{(3-n)^{\frac{2n+2}{2n+1}}(n+1)\left(k_c + bk_\varphi\right)^{\frac{1}{2n+1}}}$$

2.4. Soil thrust is given by the following equation:

$$H = cA + W\tan\varphi$$

Loose sand has low cohesion and high angle of internal friction (low c and high φ), while packed clay has high cohesion and low angle of internal friction (high c and low φ). This has the following implications:

- Vehicle weight: as weight increases, soil thrust will increase faster for loose sand than packed clay, as $\tan\varphi$ is higher for sand than for clay.
- Area: as area increases, soil thrust will increase faster for packed clay than for loose sand, as c is higher for clay than for sand.

Let's look at an example:

Dry sand: $c = 1kPa, \varphi = 45° \rightarrow 1000 * A + W * 1$

Lean clay: $c = 69kPa, \varphi = 20° \rightarrow 69000 * A + W * 0.364$

If area (A) is kept constant, increasing the vehicle weight by 1 Newton for dry sand will cause soil thrust to increase by 1 Newton. In lean clay, increasing the vehicle weight by 1 Newton will cause soil thrust to increase by 0.364 Newton. This means increase in soil thrust is 2.75 times greater in dry sand as opposed to lean clay, when the vehicle weight is increased by the same amount for both substances.

If weight (W) is kept constant, increasing the area by $1\ m^2$ for dry sand will cause soil thrust to increase by 1000 N. In lean clay, increasing the area by $1\ m^2$ will cause soil thrust to increase by 69000 N. This means increase in soil thrust is 69 times greater in lean clay as opposed to dry stand, when the area is increased by the same amount for both substances.

It should be noted that, generally, $c \gg \tan\varphi$, so soil thrust primarily depends on cA (barring extreme situations where A is tiny and W is huge). If we assume this holds true, then we can say soil with greater cohesion will result in greater soil thrust. Therefore, in the general case, operating in packed clay will result in greater maximum thrust than in loose sand, which also makes sense intuitively.

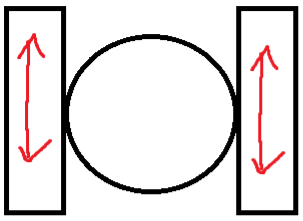
2.5. bulldozing resistance, obstacle resistance, and turning resistance are ignored (among other things).

2.6. W_{Gross}, or GVW, is gross vehicle weight in lbs. b is average tire section width in inches, d is average tire outside diameter in inches, m is total number of axels, n is average number of tires per axle, and δ is average hard-surface tire deflection in inches.

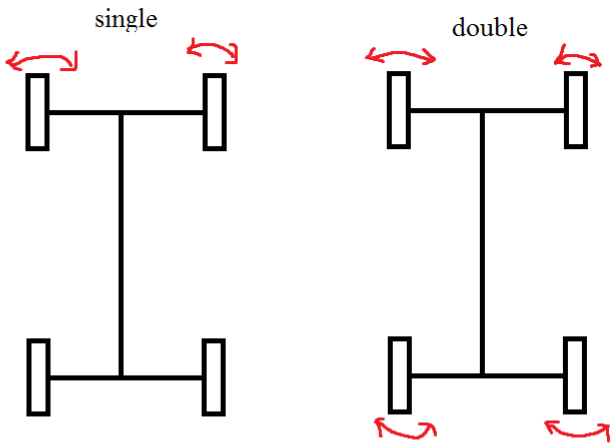
For two vehicle designs of the same weight, one with MMP = 40 and the other with MMP = 100, I would choose MMP = 40. On rigid terrain, such as pavement or a highway, MMP wouldn't matter very much, so either design would work fine. On soft terrain, however, the design with lower MMP would perform better. Lower MMP means less stress is applied to the terrain, making soil failure less likely to occur. Therefore, the design with MMP = 40 will have fewer issues on soft terrain than the design with MMP = 100, making it my preferred design.

3.1.

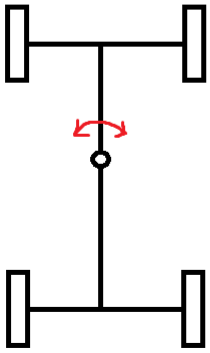
A



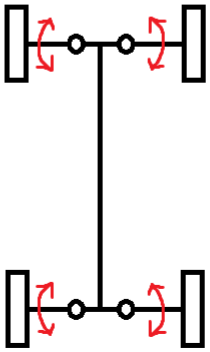
B



C



D



3.2.

Type	Pro	Con
Skid	Mechanically very simple	Requires high torque motors
Ackerman	Mechanically simple; very popular so a lot of reference material	Cannot turn in place, and minimum turn radius is large
Articulated body	May have very sharp turns, depending on how much the body can change	Requires serious mechanical consideration, as joint in chassis has huge implications for mechanics and form
Explicit	Has huge variety of options for different kinds of movements	Controls is very complex; many degrees of freedom to account for

3.3. There are several reasons why a high-speed, all-terrain wheeled vehicle may use a semi-active suspension:

- Adjust suspension for different terrain. Shifting the body of the vehicle higher may be helpful for avoiding/overcoming obstacles, while a body close to the ground will perform better on flat terrain

- Adjust suspension for different speeds. At low speeds, one set of spring stiffness and dampers may work best, while another set may be needed for higher speeds.

- Adjust suspension for different modes. A vehicle may have a race mode, where speed is all that matters, and the suspension should be set up to handle such loads. A vehicle may also have stealth mode, where the vehicle will try to move slowly and quietly. The suspension should then be adjusted to minimize movement (oscillation/shaking) and have a low center of gravity.

- A vehicle may also have a semi-active suspension because a passive suspension is insufficient due to lack of versatility, while an active suspension is too expensive or time-consuming to design and build.

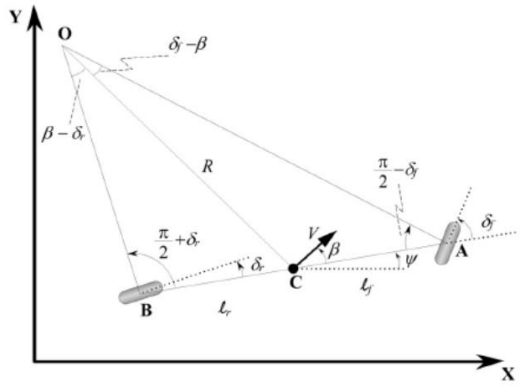
3.4. Assuming no slip, to move 2 ft/s with 20 in diameter wheels, the wheel must turn at 0.382 revolutions per second, or 22.92 RPM. At 100 lbs with 20 in diameter wheels, the motor must generate 112.98 Nm of torque.

My chosen motor and gearbox are:

- Leeson 48ZG55: Max 1800 RPM, max torque = 35 in-lbs = 3.95 Nm
- Dayton 4Z300: Max 29 RPM at output, 60:1 reducer

The gearbox can output up to 29 RPM, which is greater than the required 22.92 RPM, so this gearbox is adequate for our purposes. It has a 60:1 gear ratio, changing the required 112.98 Nm motor torque output to 1.883 Nm, and the 22.92 RPM motor speed output to 1375.2 RPM. Both the torque and RPM requirements are well within the specs of the motor. Therefore, this motor and gearbox combination are suitable for the robot design.

4.1. The general case diagram and equations are:



$$\dot{X} = V cos(\psi + \beta)$$

$$\dot{Y} = V sin(\psi + \beta)$$

$$\dot{\psi} = \frac{V cos\beta}{l_f + l_r}(tan\delta_f - tan\delta_r)$$

The equations and diagram used in Pepy are:

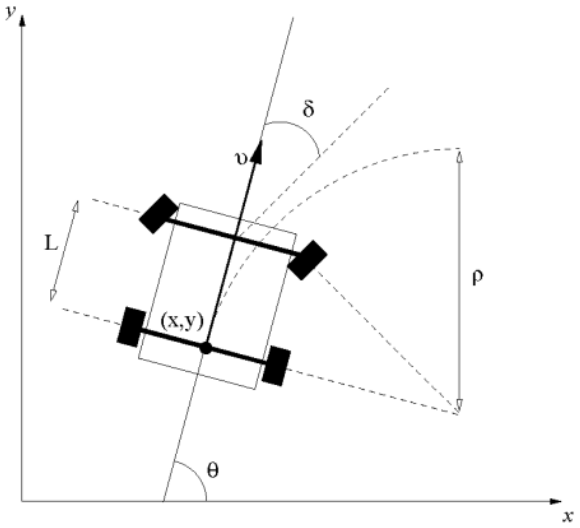


Figure 1.Kinematic model

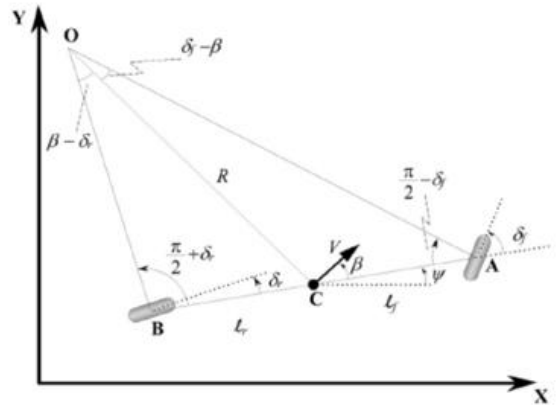
The general case shows V at angle β from the body, but the diagram used in Pepy shows that V is along the body; that is, $\beta = 0$. This allows us to simplify $\dot{x}$ and $\dot{y}$:

$$\dot{x} = V \cos \varphi$$
$$\dot{y} = V \sin \varphi$$

Since the rear wheel is not steerable, this forces $\delta_f = 0$, and with the previous discussion setting $\beta = 0$, we get the following equation:

$$\dot{\varphi} = \frac{V}{l_f + l_r} \tan \delta_f$$

4.2. The general case is below:



$$\dot{X} = V cos(\psi + \beta)$$

$$\dot{Y} = V sin(\psi + \beta)$$

$$\dot{\psi} = \frac{V cos\beta}{l_f + l_r}(tan\delta_f - tan\delta_r)$$

Point C is moving at velocity V with angle β above the body, which is angle φ above the x-axis. This gives us the following equations:

$$\dot{x} = V\cos(\varphi + \beta)$$
$$\dot{y} = V\sin(\varphi + \beta)$$

In the Kong paper, we are given the following equation:

$$\beta = \tan^{-1}\left(\frac{l_r}{l_f + l_r}\tan(\delta_f)\right) \qquad (1e)$$

We can solve for $\cos\beta$:

$$\tan\beta = \frac{\sin\beta}{\cos\beta} = \frac{l_r}{l_f + l_r}\tan\delta_f \rightarrow \cos\beta = \frac{\sin\beta}{\tan\delta_f} * \frac{l_f + l_r}{l_r}$$

Plugging in the equation for $\cos\beta$, and $\delta_r = 0$ (no steering in back wheel) into the provided equation for $\dot{\varphi}$ gives:

$$\dot{\varphi} = \frac{V}{l_f + l_r}\tan\delta_f * \cos\beta = \left(\frac{V}{l_f + l_r}\tan\delta_f\right)\left(\frac{\sin\beta}{\tan\delta_f} * \frac{l_f + l_r}{l_r}\right) = \frac{V\sin\beta}{l_r}$$

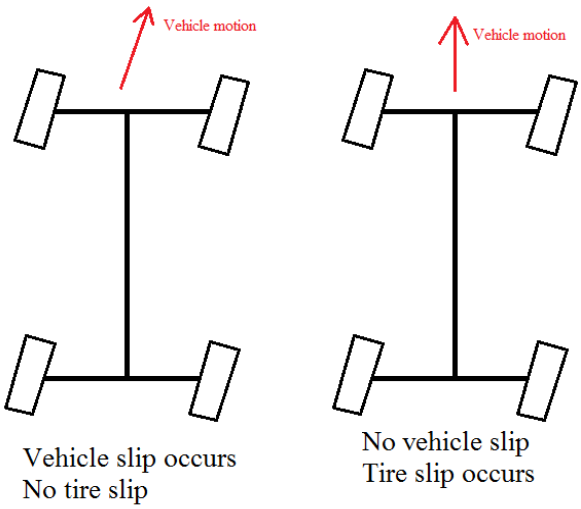
4.3.

A. Vehicle slip angle, or β, is not zero. The vehicle slip angle does not appear in the Pepy model equations because we are focusing on the rear-axel of the vehicle, rather than the center of gravity. The equation for the vehicle slip angle is:

$$\beta = tan^{-1}(\frac{l_r \tan\delta_f}{l_f + l_r})$$

B. Vehicle slip is the difference between the orientation of the vehicle and the direction of its movement. Tire slip is the difference between the orientation of the wheel and the direction of its

movement. The two are indeed related; how a wheel moves, and how the vehicle moves significantly impact each other. It is possible to vehicle slip without tire slip. If a vehicle has crab steering, the vehicle can move diagonally with respect to the body (vehicle slip) without any tire slip. It is also possible to have tire slip without vehicle slip. Using crab steering, the vehicle can move straight due to tire slip even if the wheels are oriented diagonally; this will result in no vehicle slip.



4.4.

A. We have an equation relating $\dot{\varphi}$ and δ_f:

$$\dot{\varphi} = \frac{V}{l_f + l_r} \tan \delta_f$$

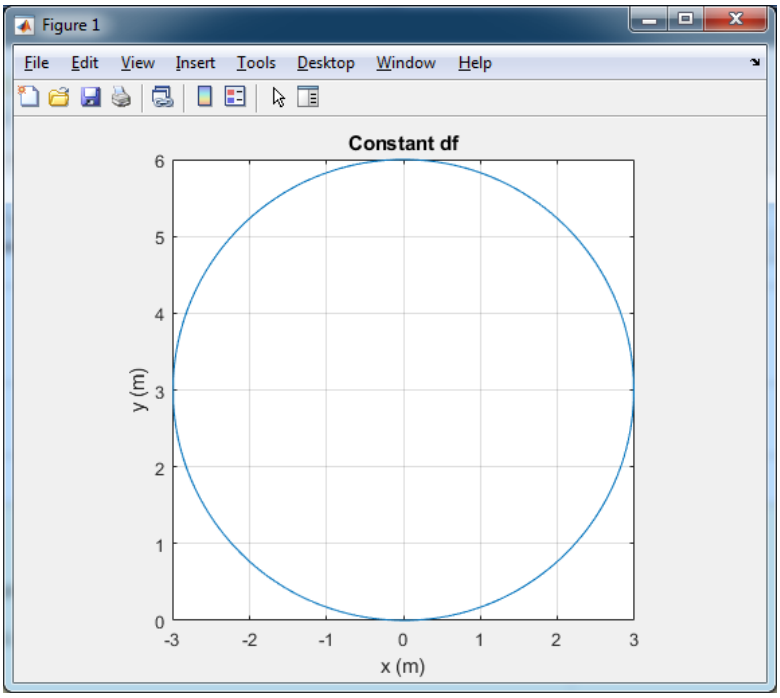
From the lecture notes we have an equation relating $\dot{\varphi}$ and R:

$$\dot{\varphi} = \frac{V}{R}$$

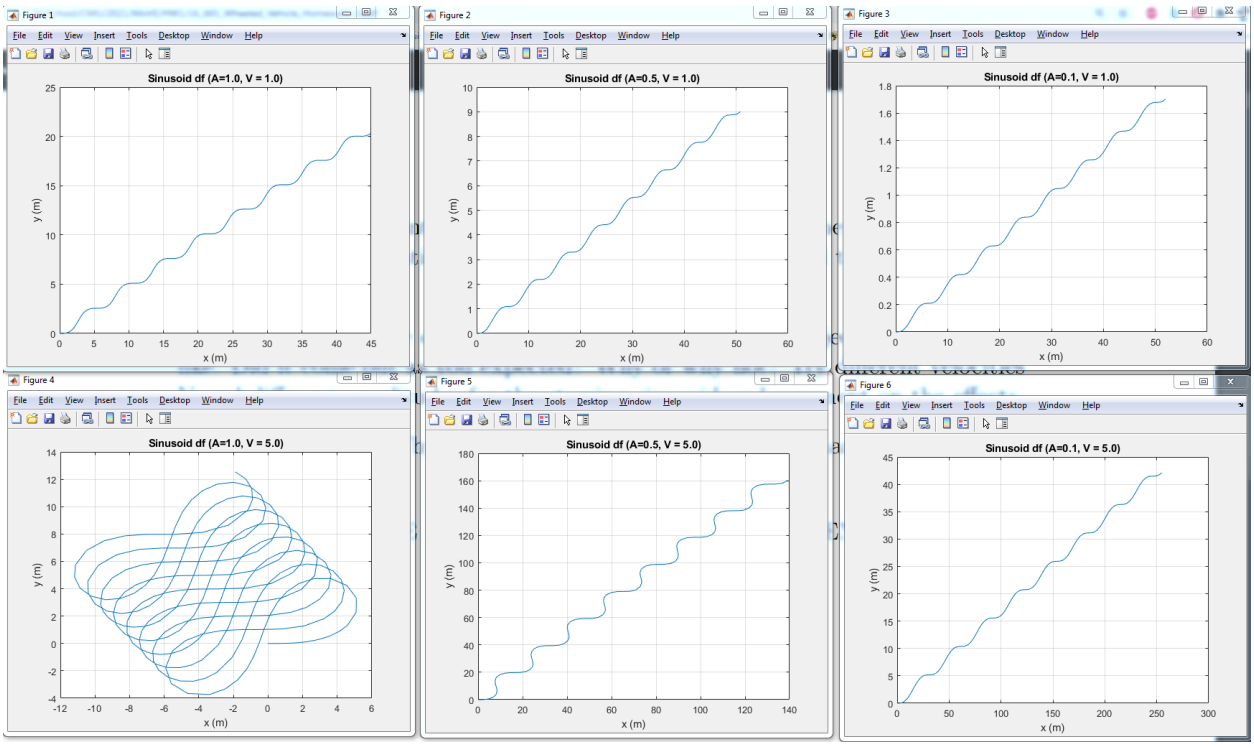
Combining the two equations gives us the following:

$$\frac{V}{l_f + l_r} \tan \delta_f = \frac{V}{R} \rightarrow R = \frac{l_f + l_r}{\tan \delta_f}$$

When δ_f is 45°, then radius is 3:



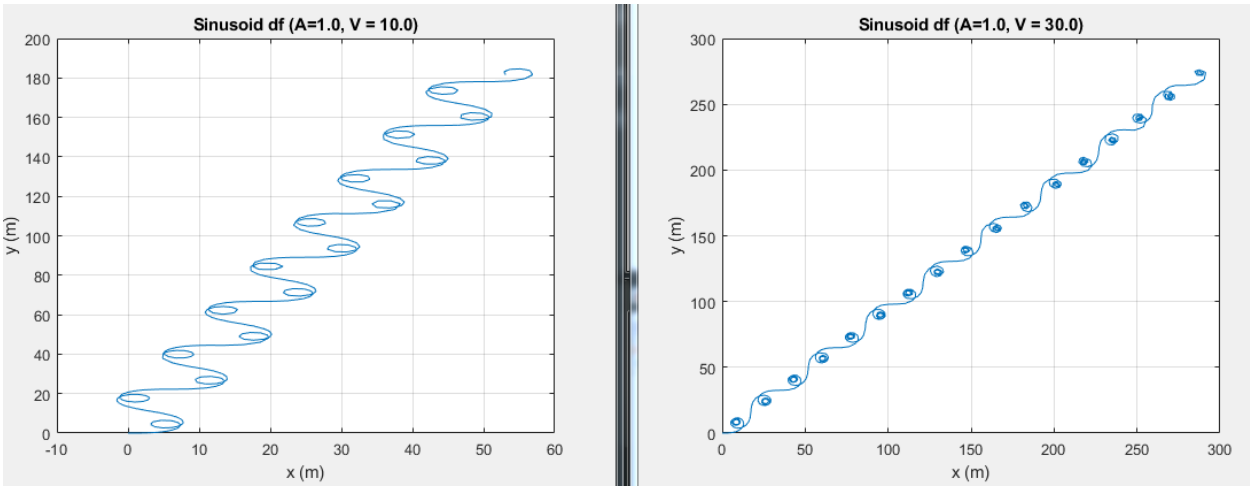
B. I suspect the path will be serpentine, or squiggly.



Above is the graphs for various amplitudes and velocities. My prediction is partially correct, but the results were ultimately different from what I expected. My prediction wasn't fully accurate as I did not fully realize the interplay between A and V.

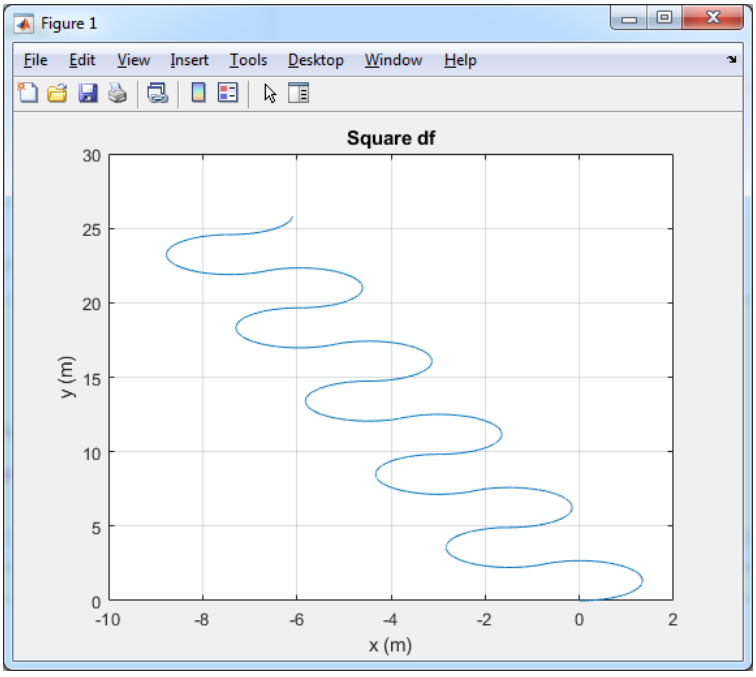
As A decreases, the path becomes less serpentine. This is due to the steering angle changing less as the amplitude of the sinusoid decreases. A smaller amplitude also has the effect of bringing the path closer to the x-axis; if A were zero, then the vehicle would travel on a straight line on the x-axis; as A increases, the vehicle deviates further from this path.

As V increases, the path becomes more serpentine. This is because larger velocity means the vehicle will move faster, amplifying the sinusoidal nature of the steering angle. What's very interesting is that large amplitudes and high velocities will produce strangely beautiful paths. $A = 1, V = 5$ produces figure-eight's, as shown above. Here are a couple more:



As velocity increases with high steering amplitude, the vehicle will move fast enough that it will do loops. That's why we saw the figure-eight earlier, and here we are seeing spirals.

C.



Here, we see the steering angle is set to one position for a while, then instantaneously changed to a different position. This results in the bicycle traveling one path for a while, then instantaneously switching to a different path, resulting in the serpentine path. This is very unrealistic, as this means the driver of the vehicle can move infinitely fast, and the vehicle can change its tires instantaneously. To handle this realistically in the model, we would need to limit the maximum derivative of the steering angle. Probably the best way to do this is to add inertia to the tires, so that the tires will gradually (not instantaneously) approach the desired steering angle. Alternatively, we could just cap the derivative of the steering angle, so the tires will move as fast as it can to the desired steering angle without breaking the laws of physics.

5.1. See MATLAB code for implementation of model. Useful snippets are included for convenience:

```
%% Constants
Vx = 30; % m/s
m = 1573; % kg
Iz = 2873; % kg m^2
lf = 1.1; % m
lr = 1.58; % m
Caf = 80000; % N/rad
Car = 80000; % N/rad
%% Matrices
% A
Ac1 = [0; 0; 0; 0];
Ac2 = [1; -2*(Caf+Car)/(m*Vx); 0; -2*(Caf*lf-Car*lr)/(Iz*Vx)];
Ac3 = [0; 2*(Caf+Car)/m; 0; 2*(Caf*lf-Car*lr)/Iz];
Ac4 = [0; 2*(-Caf*lf+Car*lr)/(m*Vx); 1; -2*(Caf*lf^2+Car*lr^2)/(Iz*Vx)];
A = [Ac1, Ac2, Ac3, Ac4];
% B1, B2
B1 = [0; 2*Caf/m; 0; 2*Caf*lf/Iz];
B2 = [0; -2*(Caf*lf-Car*lr)/(m*Vx)-Vx; 0; -2*(Caf*lf^2+Car*lr^2)/(Iz*Vx)];
```

```
function dxdt = prob5ODE(t, x, A, B1, B2, K, phid_t)
% K is 1x4 matrix for computing df using state feedback
% phid_t is a function of time for phid
%% ODE info
% x = [e1, e1', e2, e2']T
% x' = A*x + B1*df + B2*phid
%% Computation
%fprintf('t: %3.3f\n',t);
phid = phid_t(t);
dxdt = (A-B1*K)*x + B2*phid;
```

5.2. Using the following LQR controller:

```
% LQR
Q = [15, 0, 0, 0;
    0, 1, 0, 0;
    0, 0, 1, 0;
    0, 0, 0, 25];
R = 1;
K = lqr(A, B1, Q, R);
```

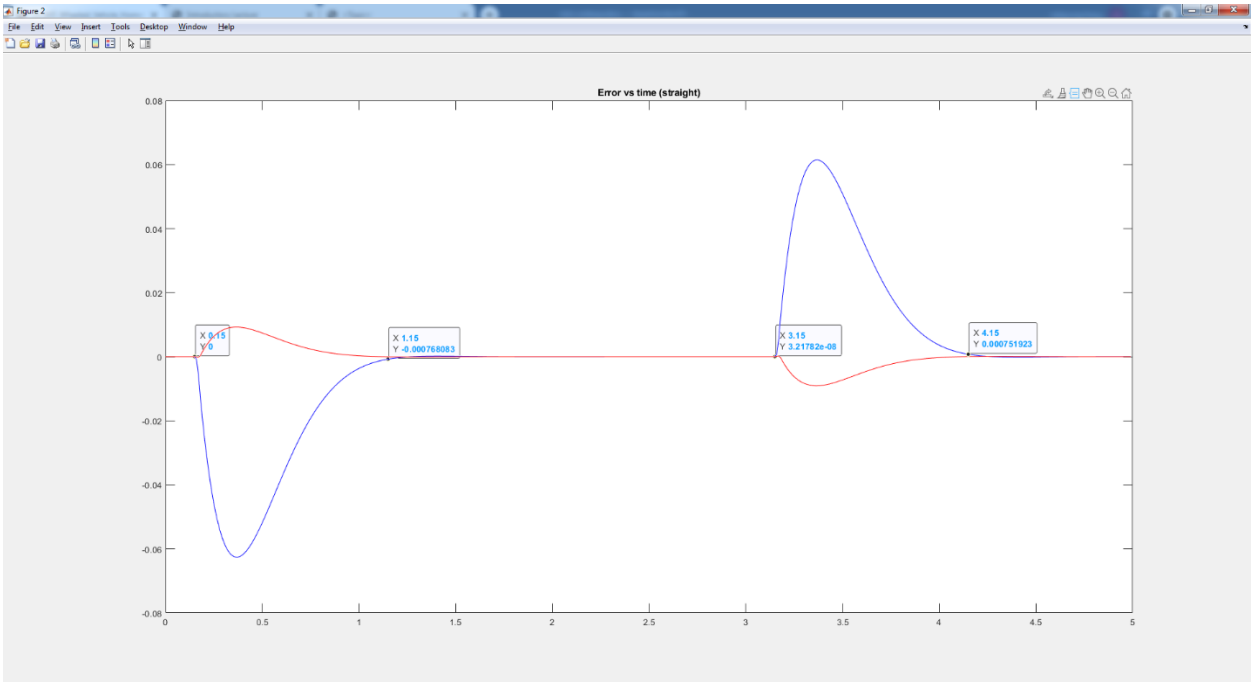
$\dot{\varphi_{des}}$ was computed by computing φ_{des} of the desired path, then taking its derivative:

```
for i = 2:(N-1)
    phid_des(i) = (phi_des(i+1)-phi_des(i-1))/(2*T);
end
```

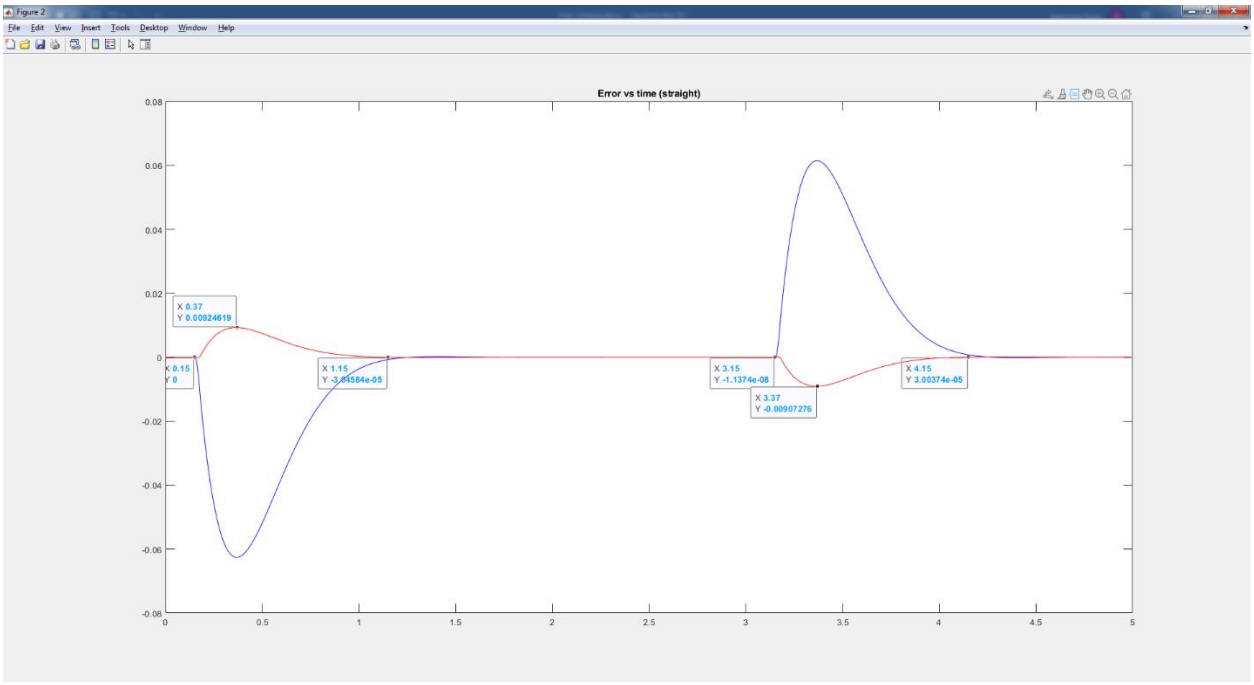
This was then input to the model, along with the K computed for the LQR controller. The output of the simulation was used to compute the actual position, as well as the error:

```
for i = 1:N
    x_act(i) = x_des(i)-x(i,1)*sin(phi_des(i)+x(i,3));
    y_act(i) = y_des(i)+x(i,1)*cos(phi_des(i)+x(i,3));
end
```

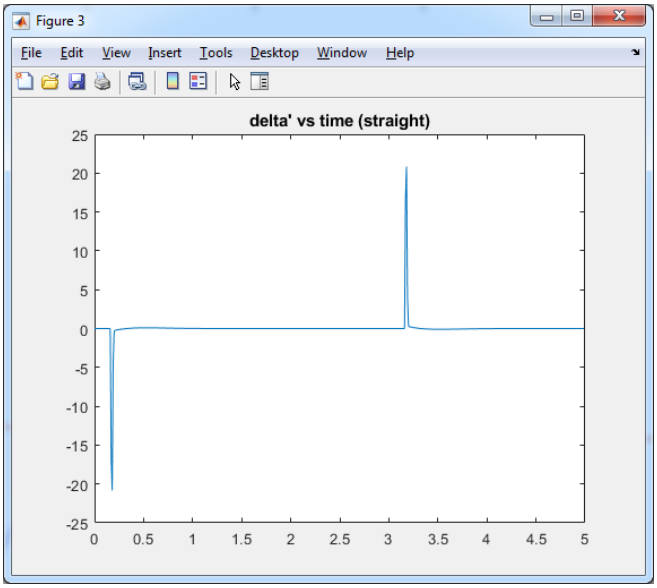
The following graphs shows e1 (blue) and e2 (red) vs time:



Here, we see that abs(e1) becomes less than 0.0008 m just one second after the disturbance, meeting the 0.002 m spec.

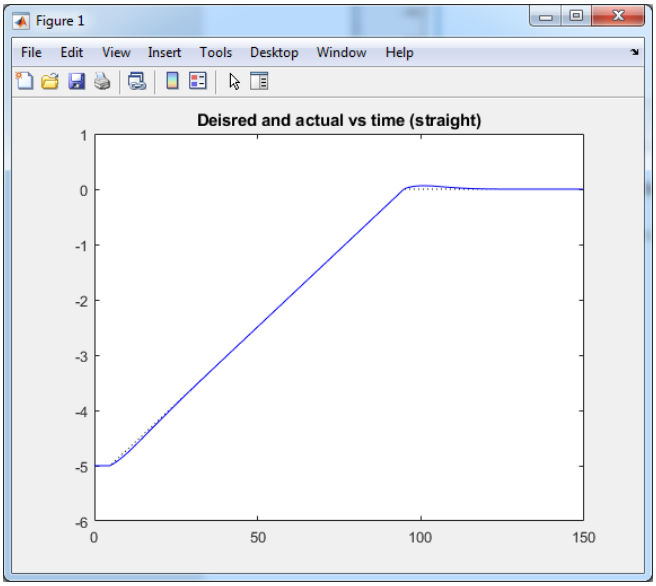


Here, we see that abs(e2) becomes less than $3.05 * 10^{-5}$ rad just one second after the disturbance, meeting the 0.0007 rad spec. Additionally, the maximum of abs(e2) is 0.0092 rad, meeting the 0.01 rad spec.

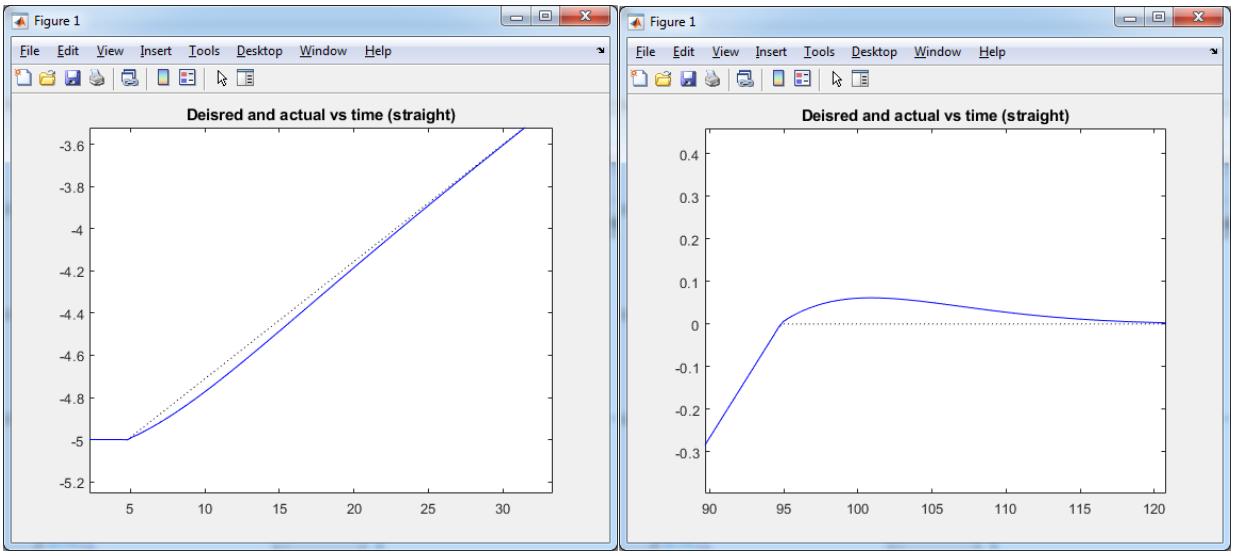


Above is the derivative of the steering input. It does not exceed 25 rad/sec.

5.3. Here is an overall plot of the desired and actual path:



The actual path is right on top of the desired path during the straight sections, and error occurs near transition points. Here is a zoom of each transition:



5.4. The following $\dot{\varphi_{des}}$ was constructed using V/R:

$$\dot{\varphi_{des}} = \begin{Bmatrix} 0, t < 1 \\ +\frac{V}{1000}, 1 \geq t < 5 \\ 0, 5 \geq t < 6 \\ -\frac{V}{500}, 6 \geq t < 12 \\ 0, t \geq 12 \end{Bmatrix}$$

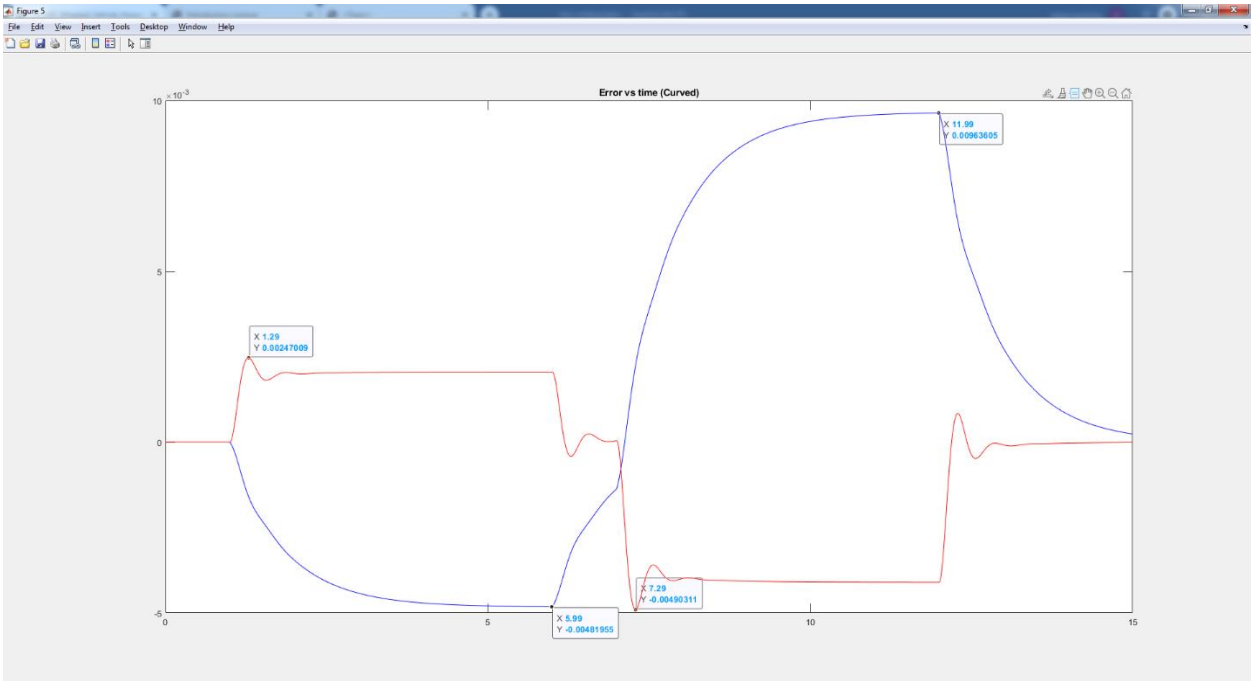
In code:

```
phid_des = 0:T:(T*(N-1));
for i = 1:100
    phid_des(i) = 0;
end
R = 1000;
for i = 101:600
    phid_des(i) = Vx/R;
end
for i = 601:700
    phid_des(i) = 0;
end
R = -500;
for i = 701:1200
    phid_des(i) = Vx/R;
end
phid_des(1201:end) = 0;
```

LQR was used to generate K:

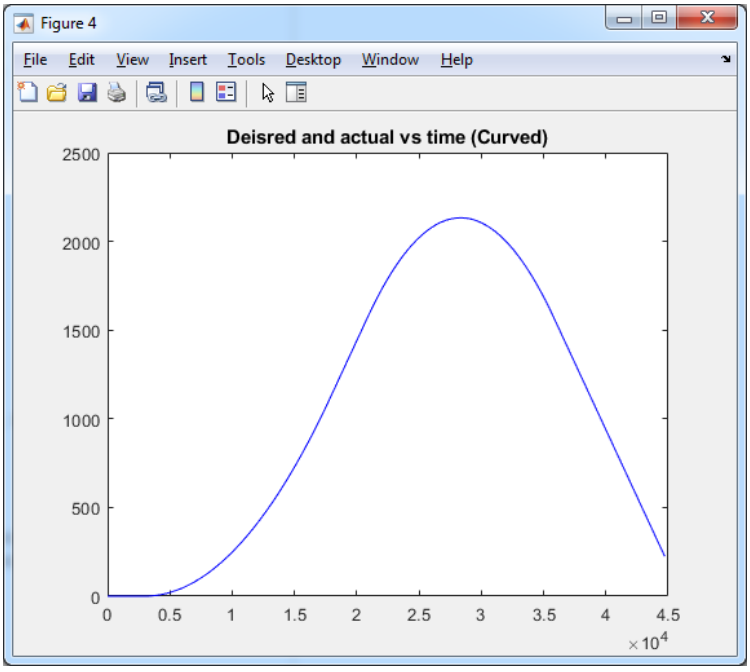
```
% Simulation Setup
Q = [15, 0, 0, 0;
    0, 10, 0, 0;
    0, 0, 15, 0;
    0, 0, 0, 3];
Q = Q;
R = 1;
K = lqr(A, B1, Q, R);
```

The simulation gave the following error:

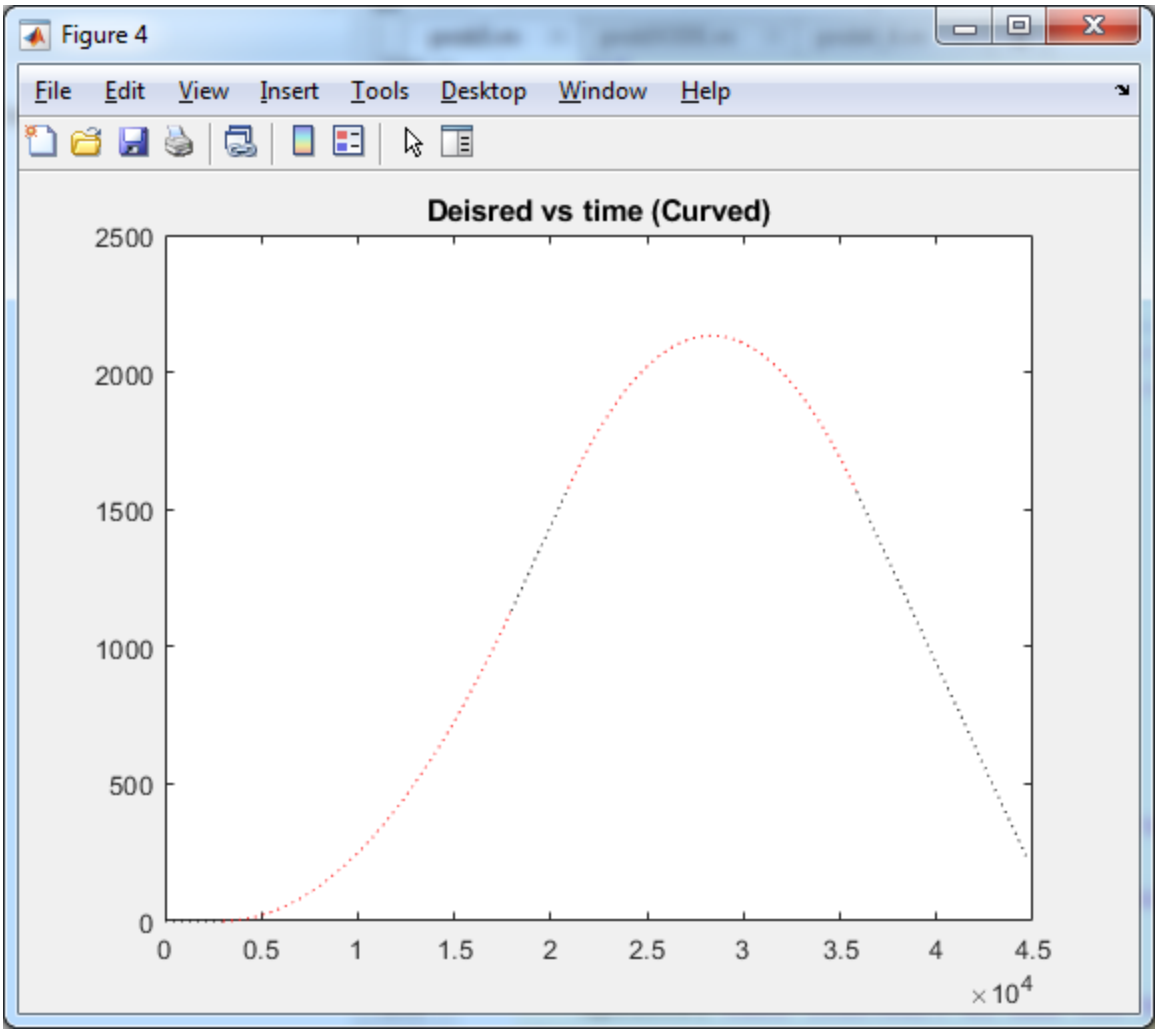


The graph above shows e1 (blue) and e2 (red) vs time. The maximum abs(e1) error is 0.0096 m, below the limit of 0.01 m. The maximum abs(e2) error is 0.0049 rad, below the limit of 0.01 rad. The performance of the vehicle meets specs.

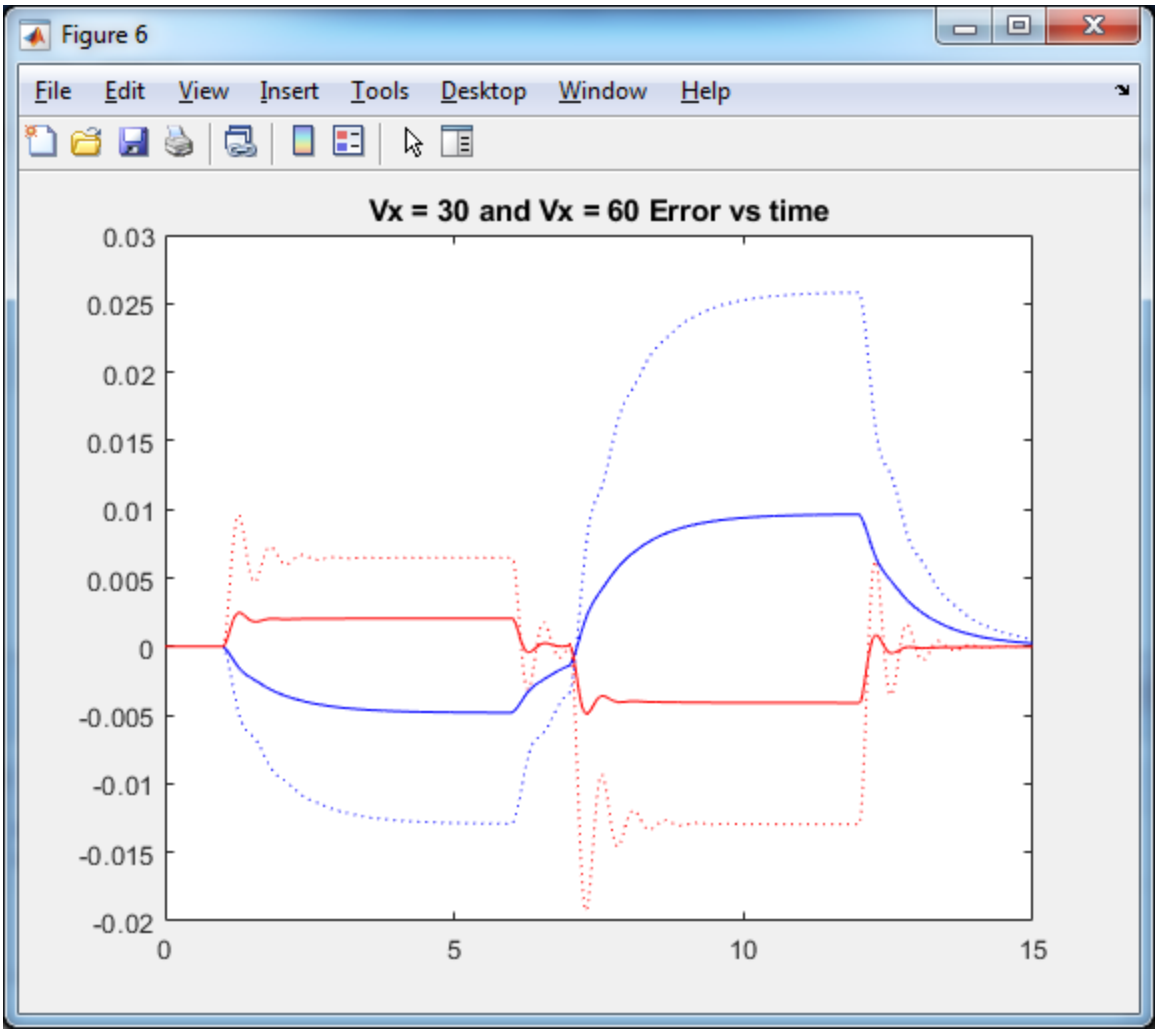
5.5 The following shows the desired and actual path:



The actual path overshadows the desired path, so here is a graph of just the desired path (straight sections in black, curved sections in red):



5.6. Error for Vx = 30 m/s and Vx = 60 m/s are shown below as a function of time:



The solid lines correspond to Vx = 30 m/s, and dashed lines correspond to Vx = 60 m/s. The blue lines correspond to e1, and the red lines correspond to e2.

The graph shows that as the speed of the vehicle increases, the errors increase significantly (LQR controller is identical for the two speeds). This shows that the performance of the controller degrades as the speed increases, mostly likely due to the controller having a harder and harder time reducing error as the timing requirements get harder (due to greater speeds).

Appendix: MATLAB Code

prob1_3.m

```
% Plot max acceleration as function of lf and h
% a = g * (lf/h * cos(theta)-sin(theta))
%% Constants
g = 9.91; % in m/s^2
m = 100; % in kg
theta = 30; % in deg
step = 0.25; % resolution for quantization
lf = 0.25:step:1.75; % in m
h = 0.5:step:2; % in m
%% Calculation
a = zeros(length(lf), length(h));
% parameterize h
for h_ind=1:length(h)
    cur_h = h(h_ind);
    for lf_ind=1:length(lf)
        cur_lf = lf(lf_ind);
        a(lf_ind,h_ind) = g*(cur_lf/cur_h * cosd(theta) - sind(theta));
    end
end
%% Plot
figure
hold on
for h_ind=1:length(h)
    plot(lf,a(:,h_ind),':o','DisplayName',sprintf('h = %2.2f m',h(h_ind)));
end
xlabel('lf [m]');
ylabel('a [m/s^2]');
lgd = legend('Location','northwest');
title('Max acceleration before tipping vs lf')
```

prob4_4.m

```
clear all; close all
%% Constants
lr = 1.5; % in meters
lf = 1.5; % in meters
V = 1; % in m/s
%% Initial condition
% x = [X, Y, PHI]'
xi = [0;0;0];
%% Time limit
tf = 52;
opts = odeset('MaxStep',0.01);
%% A: df is nonzero
df_t = @(t) 45*pi/180;
tspan=0:0.0001:51.78;
[t,x] = ode45(@(t, x) prob4_4ODE(t,x,lf,lr,V,df_t), tspan, xi,opts);
figure;
plot(x(:,1),x(:,2))
xlabel('x (m)'); ylabel('y (m)'); title('Constant df')
grid on
pbaspect([1,1,1])
%% B: df is sinusoid
for V = [1,5]
    for A = [1, 0.5, 0.1]
        df_t = @(t) A*sin(t);
        [t,x] = ode45(@(t, x) prob4_4ODE(t,x,lf,lr,V,df_t), [0, tf],
xi,opts);
        figure;
        plot(x(:,1),x(:,2))
        xlabel('x (m)'); ylabel('y (m)'); title(sprintf('Sinusoid df
(A=%2.1f, V = %2.1f)',A,V))
        grid on
    end
end
V = 1;
%% C: df is square
amp = 40;
freq = 1/5;
df_t = @(t) -(mod(floor(t*freq),2)*amp-amp/2);
[t,x] = ode45(@(t, x) prob4_4ODE(t,x,lf,lr,V,df_t), [0, tf], xi, opts);
figure;
plot(x(:,1),x(:,2))
xlabel('x (m)'); ylabel('y (m)'); title('Square df')
grid on
```

prob4_4ODE.m

```
function dxdt = prob4_4ODE(t,x,lf,lr,V,df_t)
% Note: df_t is function of time defining df
%% ODE info
% x = [X, Y, PHI]'
% X' = V*cos(PHI) = V*cos(x(3))
% Y' = V*sin(PHI) = V*sin(x(3))
% PHI' = (V/(lf+lr))*tan(df) = (V/(lf+lr))*tan(df)
df = df_t(t);
dxdt = [V*cos(x(3)); V*sin(x(3)); V/(lf+lr)*tan(df)];
```

prob5.m

```
clear all; close all;
%% Constants
Vx = 30; % m/s
m = 1573; % kg
Iz = 2873; % kg m^2
lf = 1.1; % m
lr = 1.58; % m
Caf = 80000; % N/rad
Car = 80000; % N/rad
%% Matrices
% A
Ac1 = [0; 0; 0; 0];
Ac2 = [1; -2*(Caf+Car)/(m*Vx); 0; -2*(Caf*lf-Car*lr)/(Iz*Vx)];
Ac3 = [0; 2*(Caf+Car)/m; 0; 2*(Caf*lf-Car*lr)/Iz];
Ac4 = [0; 2*(-Caf*lf+Car*lr)/(m*Vx); 1; -2*(Caf*lf^2+Car*lr^2)/(Iz*Vx)];
A = [Ac1, Ac2, Ac3, Ac4];
% B1, B2
B1 = [0; 2*Caf/m; 0; 2*Caf*lf/Iz];
B2 = [0; -2*(Caf*lf-Car*lr)/(m*Vx)-Vx; 0; -2*(Caf*lf^2+Car*lr^2)/(Iz*Vx)];
%% Straight path
% LQR
Q = [15, 0, 0, 0;
    0, 1, 0, 0;
    0, 0, 1, 0;
    0, 0, 0, 25];
R = 1;
K = lqr(A, B1, Q, R);

% Desired position computation
% On y=-5 from t=[0,16.66] -> (1:17)
% On diagonal from t = [16.66, 317.1226] -> (18:317)
% On y=0 from t=[317.1226, inf] -> (318, N)
N = 500;
T = 0.01; % simulation time step
theta = atan(5/90); % slope in rising portion
VT = Vx * T; % distance traveled in one time step
x_des = zeros(N,1);
y_des = zeros(N,1);
phi_des = zeros(N,1);
phid_des = zeros(N,1);
x_des(1) = 0;
y_des(1) = -5;
for i = 2:17
    x_des(i) = x_des(i-1)+VT;
    y_des(i) = -5;
end
for i = 18:317
    x_des(i) = x_des(i-1)+VT*cos(theta);
    y_des(i) = y_des(i-1)+VT*sin(theta);
    phi_des(i) = theta;
end
for i = 318:N
    x_des(i) = x_des(i-1)+VT;
end
```

```
% Phi_des'
for i = 2:(N-1)
    phid_des(i) = (phi_des(i+1)-phi_des(i-1))/(2*T);
end
% plot(phid_des)

% Run simulation
tspan = 0:T:(N-1)*T;
phid_t = @(t) phid_des(int64(t/T+1));
xi = [0;0;0;0];
opts = odeset('MaxStep',T);
[t, x] = ode45(@(t,x) prob5ODE(t, x, A, B1, B2, K, phid_t), tspan, xi, opts);

% Compute, display actual position and error
x_act = zeros(N,1);
y_act = zeros(N,1);
for i = 1:N
    x_act(i) = x_des(i)-x(i,1)*sin(phi_des(i)+x(i,3));
    y_act(i) = y_des(i)+x(i,1)*cos(phi_des(i)+x(i,3));
end
figure;
plot(x_des, y_des, 'k:', x_act, y_act, 'b');
title('Deisred and actual vs time (straight)');
figure;
plot(t,x(:,1),'b',t,x(:,3),'r');
title('Error vs time (straight)');

% Compute and display delta and delta'
delta = zeros(N,1);
deltad = zeros(N,1);
for i = 1:N
    delta(i) = -K*x(i,:)';
    if((i>1) && (i < N))
        deltad(i) = (delta(i+1)-delta(i-1))/(2*T);
    end
end
figure;
plot(t,deltad);
title('delta'' vs time (straight)');

% Pass / Fail
e2_max = 0.01;
e1_max_trans = 0.002;
e2_max_trans = 0.0007;
deltad_max = 25;
if(max(abs(deltad))>deltad_max)
    fprintf('FAIL (straight): deltad max too high (%3.3f > %3.3f)\n',
max(abs(deltad)),deltad_max)
else
    fprintf('PASS (straight): deltad within limits (%3.3f)\n',
max(abs(deltad)))
end
if(max(abs(x(:,3)))>e2_max)
    fprintf('FAIL (straight): e2 max too high (%3.3f > %3.3f)\n',
max(abs(x(:,3))),e2_max)
```

```
else
    fprintf('PASS (straight): e2 within limits (%3.3f)\n', max(abs(x(:,3))))
end
if(max(abs(x(17,1)-x(117,1)), abs(x(317,1)-x(417,1)))>e1_max_trans)
    fprintf('FAIL (straight): e1 transient too high (%3.3f > %3.3f)\n',
max(abs(x(17,1)-x(117,1)), abs(x(317,1)-x(417,1))), e1_max_trans)
else
    fprintf('PASS (straight): e1 transient within limits (%3.5f)\n',
max(abs(x(17,1)-x(117,1)), abs(x(317,1)-x(417,1))))
end
if(max(abs(x(17,3)-x(117,3)), abs(x(317,3)-x(417,3)))>e2_max_trans)
    fprintf('FAIL (straight): e2 transient too high (%3.3f > %3.3f)\n\n',
max(abs(x(17,3)-x(117,3)), abs(x(317,3)-x(417,3))), e2_max_trans)
else
    fprintf('PASS (straight): e2 transient within limits (%3.5f)\n\n',
max(abs(x(17,3)-x(117,3)), abs(x(317,3)-x(417,3))))
end
%% Curved path
% Phid_des = 0 for 1 sec, Vx/1000 for 5 sec, 0 for 1 sec, -Vx/500 for 5 sec
N = 1500;
phid_des = 0:T:(T*(N-1));
for i = 1:100
    phid_des(i) = 0;
end
R = 1000;
for i = 101:600
    phid_des(i) = Vx/R;
end
for i = 601:700
    phid_des(i) = 0;
end
R = -500;
for i = 701:1200
    phid_des(i) = Vx/R;
end
phid_des(1201:end) = 0;
% Phi_des is integral, or running sum, of phid_des
phi_des = 0:T:(T*(N-1));
for i = 2:length(phi_des)
    phi_des(i) = phi_des(i-1) + T*phid_des(i);
end
% Compute x_des and y_des using phi_des and Vx
x_des = 0:T:(T*(N-1));
y_des = 0:T:(T*(N-1));
for i = 2:length(x_des)
    x_des(i) = x_des(i-1) + Vx*cos(phi_des(i));
    y_des(i) = y_des(i-1) + Vx*sin(phi_des(i));
end
% Simulation Setup
Q = [15, 0, 0, 0;
    0, 10, 0, 0;
    0, 0, 15, 0;
    0, 0, 0, 3];
Q = Q;
R = 1;
K = lqr(A, B1, Q, R);
% Simulation
```

```
tspan = 0:T:(N-1)*T;
phid_t = @(t) phid_des(int64(t/T+1));
xi = [0;0;0;0];
opts = odeset('MaxStep',T);
[t, x] = ode45(@(t,x) prob5ODE(t, x, A, B1, B2, K, phid_t), tspan, xi, opts);
% Compute actual position
x_act = zeros(N,1);
y_act = zeros(N,1);
for i = 1:N
    x_act(i) = x_des(i)-x(i,1)*sin(phi_des(i)+x(i,3));
    y_act(i) = y_des(i)+x(i,1)*cos(phi_des(i)+x(i,3));
end
% Compute delta and deltad
delta = zeros(N,1);
deltad = zeros(N,1);
for i = 1:N
    delta(i) = -K*x(i,:)';
    if((i>1) && (i < N))
        deltad(i) = (delta(i+1)-delta(i-1))/(2*T);
    end
end
% Display results
figure;
plot(...
    x_des(1:100),y_des(1:100),'k:',...
    x_des(101:600),y_des(101:600),'r:',...
    x_des(601:700),y_des(601:700),'k:',...
    x_des(701:1200),y_des(701:1200),'r:',...
    x_des(1201:end),y_des(1201:end),'k:',...
    x_act, y_act, 'b')
title('Deisred and actual vs time (Curved)');
figure;
plot(t,x(:,1),'b',t,x(:,3),'r');
title(sprintf('Error vs time (Curved), V = %2.2f m/s', Vx));
% Pass / Fail
e1_max = 0.01;
e2_max = 0.01;
deltad_max = 25;
if(max(abs(deltad))>deltad_max)
    fprintf('FAIL (curved): deltad max too high (%3.3f > %3.3f)\n',
max(abs(deltad)),deltad_max)
else
    fprintf('PASS (curved): deltad within limits (%3.3f)\n',
max(abs(deltad)))
end
if(max(abs(x(:,1)))>e1_max)
    fprintf('FAIL (curved): e1 max too high (%3.3f > %3.3f)\n',
max(abs(x(:,1))),e1_max)
else
    fprintf('PASS (curved): e1 within limits (%3.5f)\n', max(abs(x(:,1))))
end

if(max(abs(x(:,3)))>e2_max)
    fprintf('FAIL (curved): e2 max too high (%3.3f > %3.3f)\n',
max(abs(x(:,3))),e2_max)
else
    fprintf('PASS (curved): e2 within limits (%3.5f)\n', max(abs(x(:,3))))
```

```
end
%% New velocity
Vx = 60; % m/s
% A
Ac1 = [0; 0; 0; 0];
Ac2 = [1; -2*(Caf+Car)/(m*Vx); 0; -2*(Caf*lf-Car*lr)/(Iz*Vx)];
Ac3 = [0; 2*(Caf+Car)/m; 0; 2*(Caf*lf-Car*lr)/Iz];
Ac4 = [0; 2*(-Caf*lf+Car*lr)/(m*Vx); 1; -2*(Caf*lf^2+Car*lr^2)/(Iz*Vx)];
A = [Ac1, Ac2, Ac3, Ac4];
% B1, B2
B1 = [0; 2*Caf/m; 0; 2*Caf*lf/Iz];
B2 = [0; -2*(Caf*lf-Car*lr)/(m*Vx)-Vx; 0; -2*(Caf*lf^2+Car*lr^2)/(Iz*Vx)];
tspan = 0:T:(N-1)*T;
phid_t = @(t) phid_des(int64(t/T+1));
xi = [0;0;0;0];
opts = odeset('MaxStep',T);
[t, x60] = ode45(@(t,x) prob5ODE(t, x, A, B1, B2, K, phid_t), tspan, xi,
opts);
figure;
plot(t,x(:,1),'b',t,x(:,3),'r',...
    t,x60(:,1),'b:',t,x60(:,3),'r:');
title('Vx = 30 and Vx = 60 Error vs time');
```

prob5ODE.m

```
function dxdt = prob5ODE(t, x, A, B1, B2, K, phid_t)
% K is 1x4 matrix for computing df using state feedback
% phid_t is a function of time for phid
%% ODE info
% x = [e1, e1', e2, e2']T
% x' = A*x + B1*df + B2*phid
%% Computation
%fprintf('t: %3.3f\n',t);
phid = phid_t(t);
dxdt = (A-B1*K)*x + B2*phid;
```