Jay Jiang

NetID: jjj75

Master of Engineering of Mechanical and Aerospace Engineering

MAE 6900 4 credits

I worked on the dynamic modeling and theory, bicycle controller designing and high level Matlab programming.

Dynamics and Control of a Self-stabilizing Bicycle

Jay Jiang

Master of Engineering Project

12/15/2014

Acknowledgement

My completion of the project cannot be accomplished without the support of my advisor, teammates and friends.

Firstly, I want to thank my advisor, Professor Ruina for pointing out the research direction whenever I am lost.

Secondly, I want express my special thanks to Mathew Sheen for generously devoting his own time helping me with the optimization code.

Thirdly, I want to thank Diego Olvera for sharing his knowledge on the subject.

Last but not least, I want to thank my project teammates, Elliott Grom, Ariam Espinal, Marvin Liu, Rany Megally, Shihao Wang, Ellen Yu for their continuous effort on the project.

Table of Contents

Abstract

The main goal of this project is to develop a self-stabilizing bicycle. In order to accomplish this design objective, appropriate dynamic model describing the motion of the bicycle has to be found first. This paper explores the differences and similarities between the Whipple model [1] and the point mass model. After the comparison, controllers are developed based on the point mass dynamic model. Multiple control approaches, such as PID, LQR and numerical optimization, are implemented, compared and validated against each other.

Introduction

Since the invention of the bicycle, how and why they balance themselves has remained a mystery. Instead of exploring this unsolved question, this paper mainly focuses on modeling the bicycle dynamics and bicycle control.

The developed model and designed controller will be implemented to a real bicycle in the Cornell University BioRobotics and Locomotion Lab for testing. In the near future, the designed self-stabilizing bicycle will be added a steer-by-wire system which turns the handle bar into a joystick. This feature, combined with the self-stabilizing capability, enables people do not know how to ride a bicycle enjoy the bicycle riding experience.

Dynamic Model

Whipple Model

Whipple model divides the bicycle into four parts: Rear body and frame, rear wheel, front handlebar and fork, and front wheel.

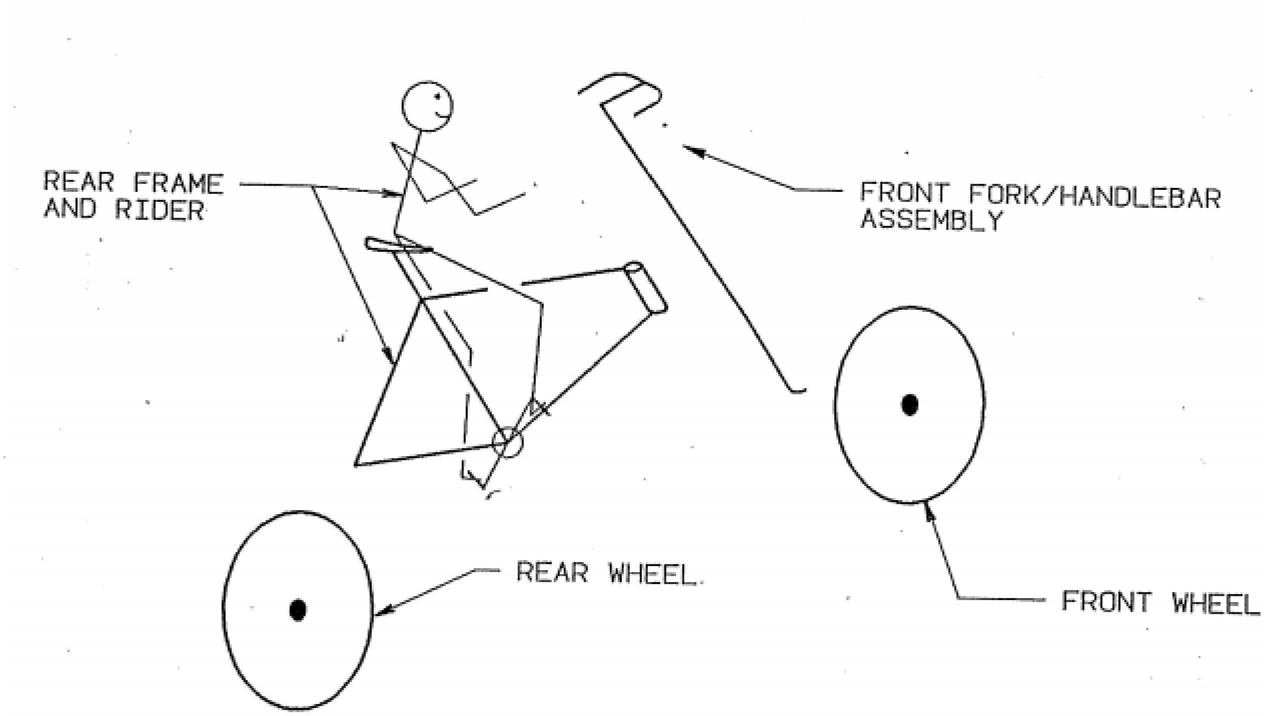

Figure 1. Bicycle model [5]

The equation of motion of the bicycle was linearized and simplified to:

$$M \begin{bmatrix} \ddot{\phi} \\ \ddot{\delta} \end{bmatrix} + vC_1 \begin{bmatrix} \dot{\phi} \\ \dot{\delta} \end{bmatrix} + (gK_0 + v^2 K_2) \begin{bmatrix} \phi \\ \delta \end{bmatrix} = \begin{bmatrix} T_\phi \\ T_\delta \end{bmatrix} \tag{1}$$

At any given speed, Equation (1) is rearranged into to the following form:

$$\begin{bmatrix} \ddot{\phi} \\ \ddot{\delta} \end{bmatrix} = M^{-1}\left(-C \begin{bmatrix} \dot{\phi} \\ \dot{\delta} \end{bmatrix} - K \begin{bmatrix} \phi \\ \delta \end{bmatrix} + \begin{bmatrix} T_\phi \\ T_\delta \end{bmatrix}\right) \tag{2}$$

Where Ø is the lean angle, and δ is the steering angle.

The motion of the bicycle is then described using mass matrix, M, damping matrix, C, stiffness matrix, K, and torque matrix, T. Refer to Appendix A for derivation of M, C and K.

During the early phase of the design, physical data and property of the actual bicycle in Cornell University BioRobotics and Locomotion Lab was not available, Matlab code was written to expedite matrix calculation once the data becomes available. To ensure the code was correctly written, parameters used in the code were the same as provided in source [1].

M =

80.8172 2.3194

2.3194 0.2978

K =

-794.1195 663.8749

-25.5013 16.0085

C =

0 101.5992

-2.5511 5.0562

The above matrix results were calculated directly from Matlab code and matched exactly with the data sheet provided in source [1]. Therefore, it can be trusted to calculate matrix for any bicycle properties. After actual data of the bicycle in the laboratory was provided by Elliott Grom, Matlab code was updated.

To further validate the numerical solution, natural motion of the uncontrolled bicycle was calculated in Matlab using ode45 function.

At forward velocity=5 m/s, the motion of the bicycle is shown in the following 2 figures.

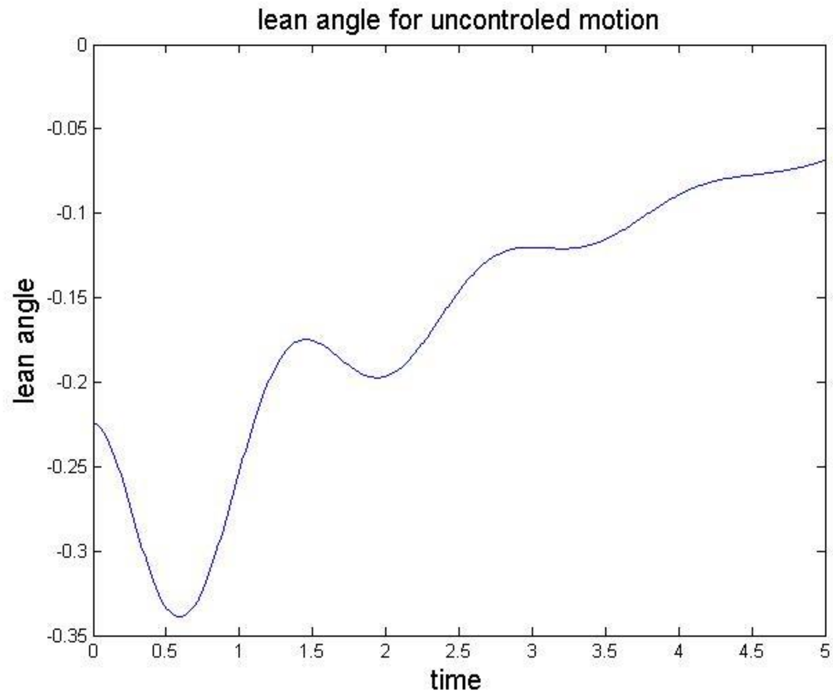

Figure 2: Lean angle vs time of uncontrolled motion for Whipple model

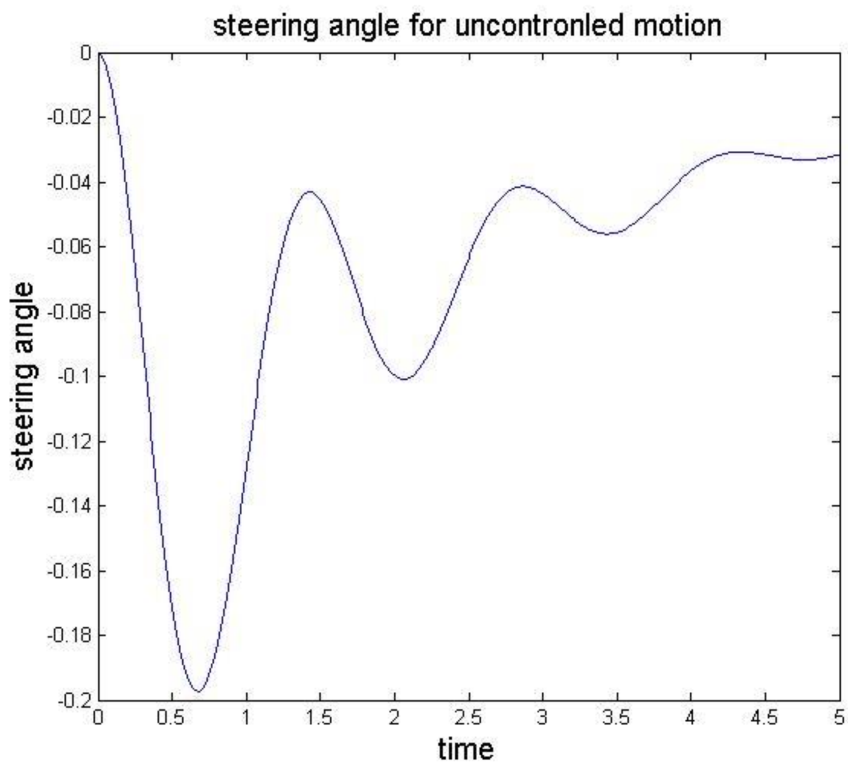

Figure 3: Steering angle vs time of uncontrolled motion for Whipple model

To compare this numerical solution with the one given in source [1], reproducing solution based on eigenvalues provided was attempted first. Because no eigenvector information was available, upon Professor Ruina's suggestion, Matlab function Iscurvefit and nlinfit were used to approximate the numerical solution. The approximated solution equations were converted to eigenvalues to compare with the provided data.

V (m/s)	Re	IM
1	3.5272	0.8077
2	2.6825	1.6806
3	1.7068	3.0791
4	0.4133	3.0791
5	-0.7753	4.4649
6	-1.5264	5.8768
7	-2.1385	7.1955
8	-2.6929	8.4609
9	-3.2167	9.6938
10	-3.7201	10.9061

Table 1: Eigenvalues from numerical solution.

Data in Table 1 matched closely (error smaller than 10^{-4}) with that given in the benchmark paper in Appendix B.

Point Mass Model

Whipple model depends on the torque matrix which is difficult to measure or control precisely in practice. By removing the second row of Equation (2), Whipple model is simplified to a mass model. The point mass model removes all the inertia and consider the whole bicycle is a point mass hinges about the contact line of the front and rear wheels. The control of the bicycle is then similar to balancing an inverted pendulum.

Different from the Whipple model, point mass model is never stable for uncontrolled motion.

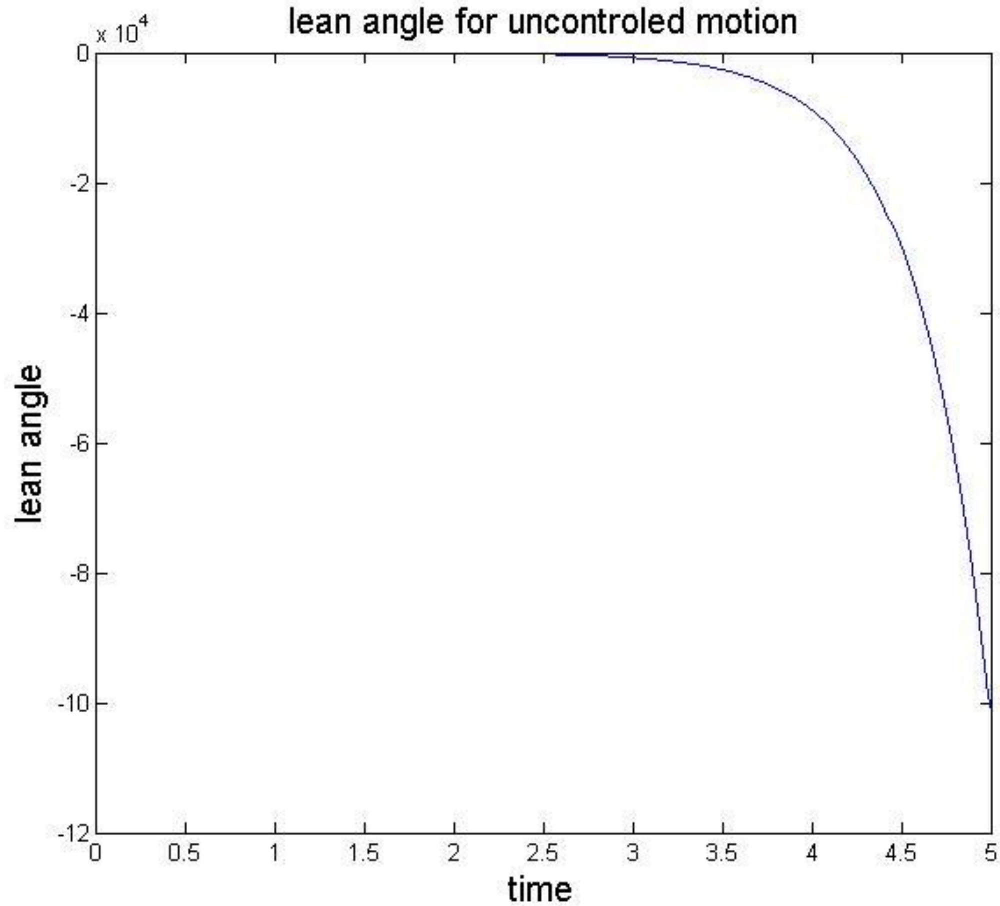

Figure 4: Lean angle vs time of uncontrolled motion for point mass model

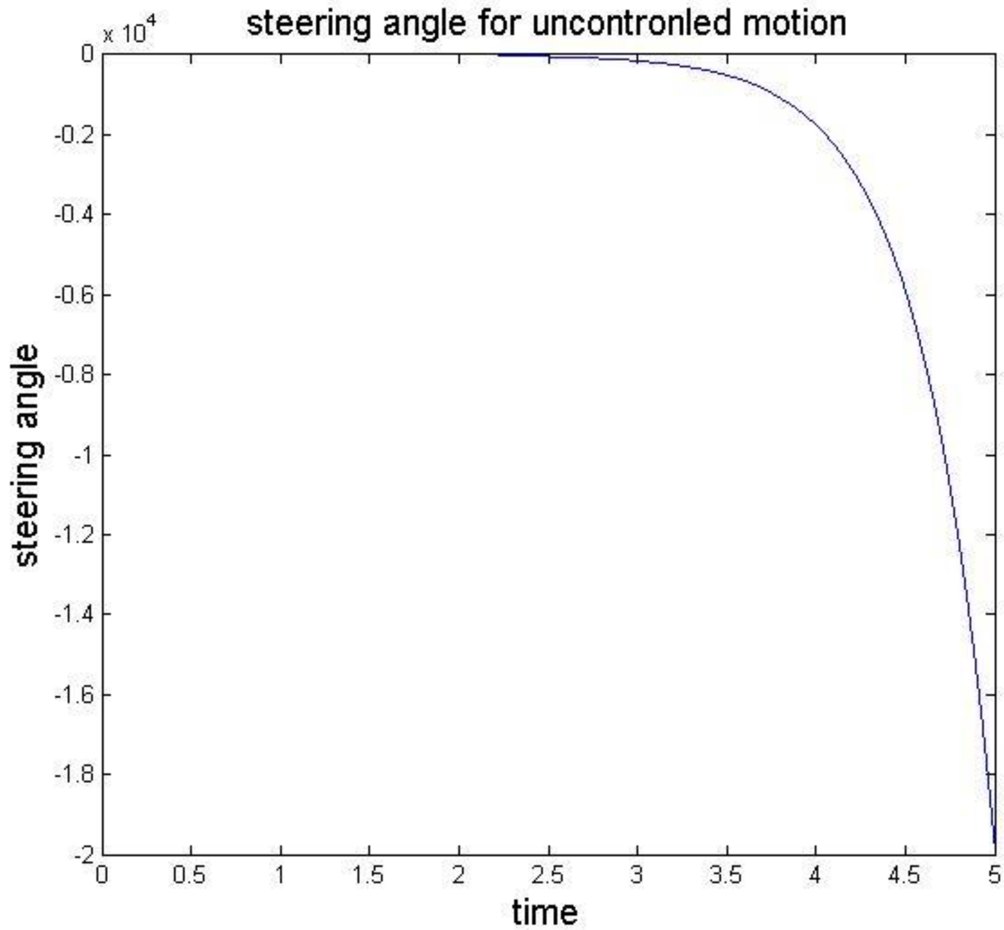

Figure 5: Steering angle vs time of uncontrolled motion for point mass model

As shown in Figure 4 and Figure 5, the point mass model is unstable. This matches with our prediction.

Controlling the Bicycle

Control

Since the point mass model is always unstable, in order to compare the two models, a controller need to be find for controlled motion. By assuming the motor can provide any steering angular velocity, $\dot{\delta}$ can be used both as a state and as a control variable. With this control notion in mind, the following matrix were constructed:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\delta} \\ \ddot{\phi} \end{bmatrix} = \mathbf{A} \begin{bmatrix} \phi \\ \delta \\ \dot{\phi} \end{bmatrix} + \mathbf{B} \left[\dot{\delta} \right] \tag{3}$$

For the Whipple model, A and B matrix were calculated using Equation (4) and (5).

$$\text{A\_whipple}= \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -\text{MiK}(1,1) & -\text{MiK}(1,2) & -\text{MiC}(1,1) & 0 \\ -\text{MiK}(2,1) & -\text{MiK}(2,2) & -\text{MiC}(2,1) & 0 \end{bmatrix} \tag{4}$$

$$\text{B\_whipple}= \begin{bmatrix} 0 \\ 1 \\ -\text{MiC}(1,2) \\ -\text{MiC}(2,2) \end{bmatrix} \tag{5}$$

Where Mi is the inverse of the mass matrix, M. MiK is product of Mi and K matrix, and numbers in the parenthesis is the index of the result matrix.

Similarly, A and B matrix were calculated for the point mass model as shown in Equation (6) and (7).

$$\mathbf{A}_{\text{pointmass}}= \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -\text{MiK}(1,1) & -\text{MiK}(1,2) & 0 \end{bmatrix} \tag{6}$$

$$\mathbf{B}_{\text{pointmass}}= \begin{bmatrix} 0 \\ 1 \\ -\text{MiC}(1,2) \end{bmatrix} \tag{7}$$

The controller design of the bicycle is greatly assisted by Jason Moore's research and papers [2]. The controller in use can bring the bicycle back to stable motion at any speed for any initial conditions. Proportional control is implemented so that steering angular velocity is dependent on lean angle, lean angular velocity and steering angle. The result of the controller are shown in the following figures.

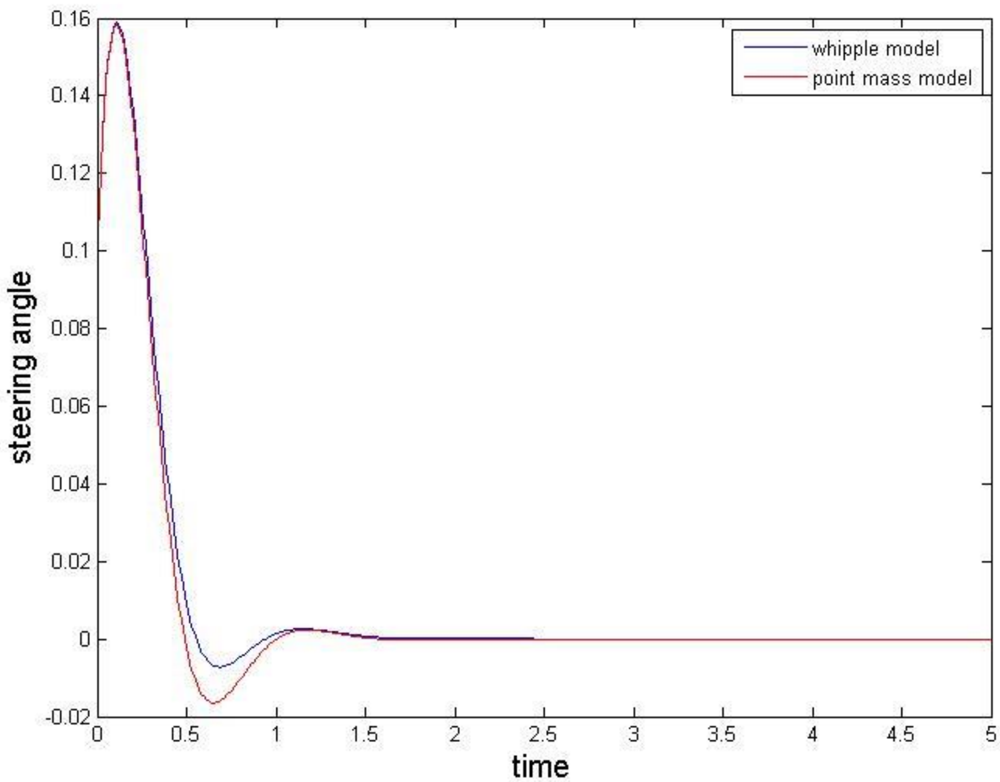

Figure 6: Steering angle for controlled motion

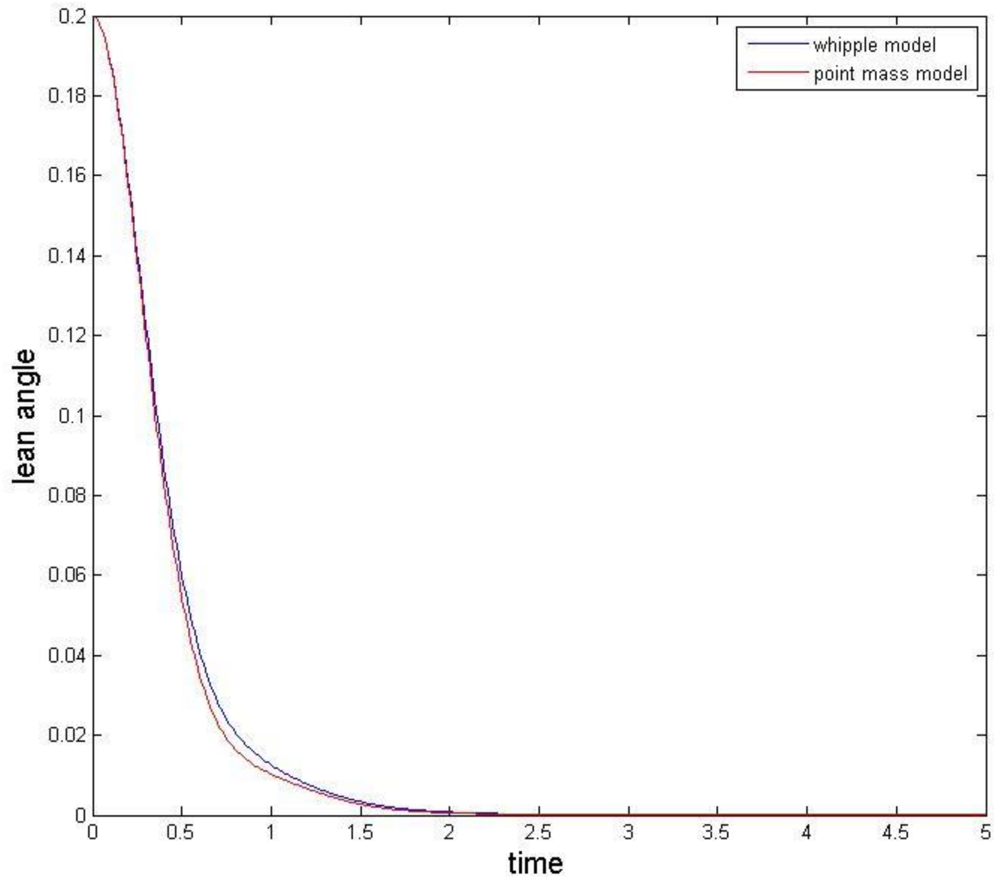

Figure 7: Lean angle for controlled motion

As shown in Figure 6 and Figure 7, the control of the bicycle based on two different models are successful and yield very similar result.

Controller Optimization

After a "good" controller was found, the controller optimization began. In theory, any observable and controllable system can be controlled given unlimited actuation power. The design of controller is often a tradeoff between the transient response and the control effort.

Based on the objective of the self-stabilizing bicycle and hardware specifications, the following requirements for the controller were created:

1. Less than 1 second 5% settling time.
2. Max steering velocity less than π rad/s (mechanical constraint).

PID controller

PID (Proportional-integral-derivative) controller is widely used in the industrial control field. Equation (8) describes the PID control theory.

$$\mathrm{u}(t) = \mathrm{MV}(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{d}{dt} e(t) \tag{8}$$

Where K_p, K_i and K_d are proportional, integral and derivative gains, respectively. By tuning these 3 open loop gains, error is eliminated and desired closed loop transient response is obtained.

Response	Rise Time	Overshoot	Settling Time	S-S Error
K_P	Decrease	Increase	NT	Decrease
K_I	Decrease	Increase	Increase	Eliminate
K_D	NT	Decrease	Decrease	NT

Table 2: PID control effect [3]

Table 2 shows the effects of transient response of increasing each gains independently.

For its simplicity and feasibility, PID control was first attempted to control the bicycle. After PID control was implemented in Matlab, it was found to have limitations on handling multiple input and output constraints.

Numerical Optimization

After the failed attempt with the PID control, “brutal force” numerical optimization was implemented. Matlab code, *fminsearch* and *fmincon*, were used to search for the optimal transient response and implement the constraint. The imbedded constraint function in *fmincon* has to be pre-defined and be run before the cost function. Due to this limitation and the fact that constraint in this problem has be updated every time ode45 is evaluated inside of the cost function, Matlab code was written manually to ensure the specifications were met. Upon Matthew Sheen’s suggestion, GPOPS II, a software specialized in trajectory optimization, was also used to assist in solving this problem.

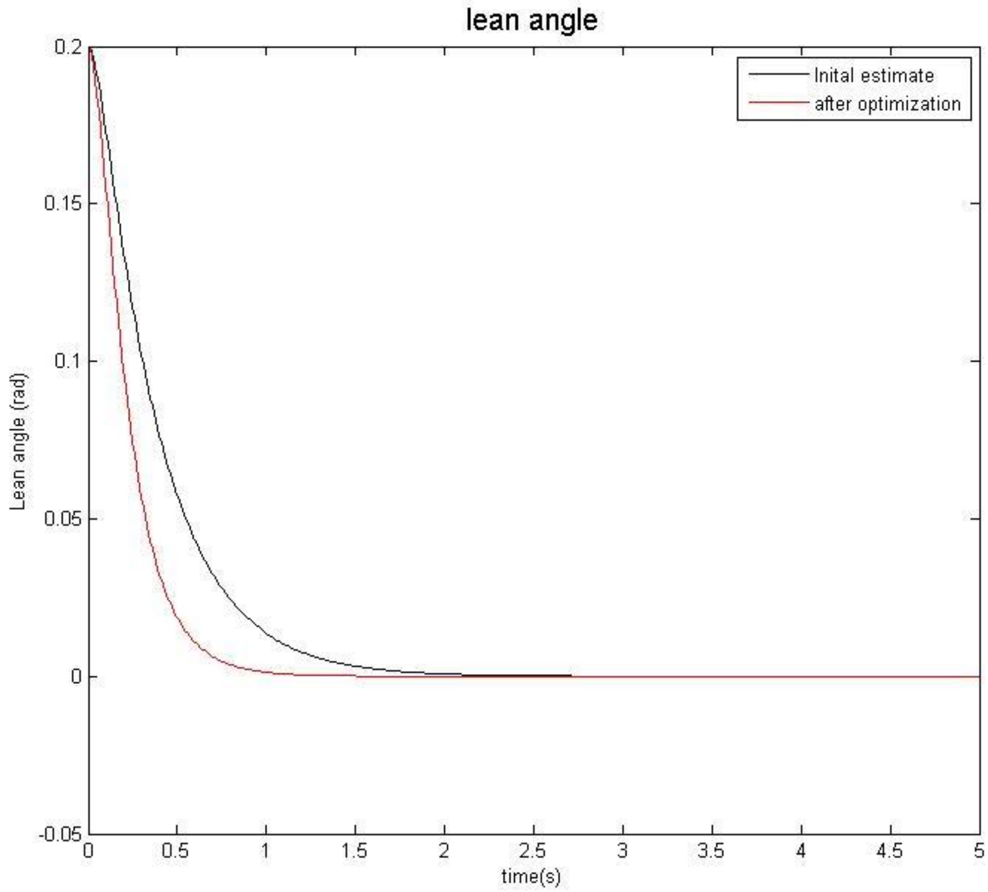

Figure 8: Transient response optimization result

As shown in Figure 8, the transient response of the lean angle recovery back to equilibrium is much better after the optimization code.

Validation of the numerical method

To further validate the developed numerical optimization method, LQR (Linear-quadratic regulator) was also used to control the system.

LQR performs well when the system dynamics are described by a set of linear differential equations and the cost is described by a quadratic function. The detailed theory behind the LQR design is well documented through various sources, and is therefore omitted in this paper.

$$g(x_t, u_t) = x_t^\top Q x_t + u_t^\top R u_t \tag{9}$$

Equation (9) describes the core equation in LQR design [4]. By changing the ratio between Q matrix and R matrix, different weights are applied to the control effort and transient response.

To use the Matlab LQR toolbox, the developed differential equations were converted into the state-space form.

For an unconstrained motion, the LQR and the numerical optimization yield very similar results as shown in Figure 9 and Figure 10. This similarity validates the numerical optimization method.

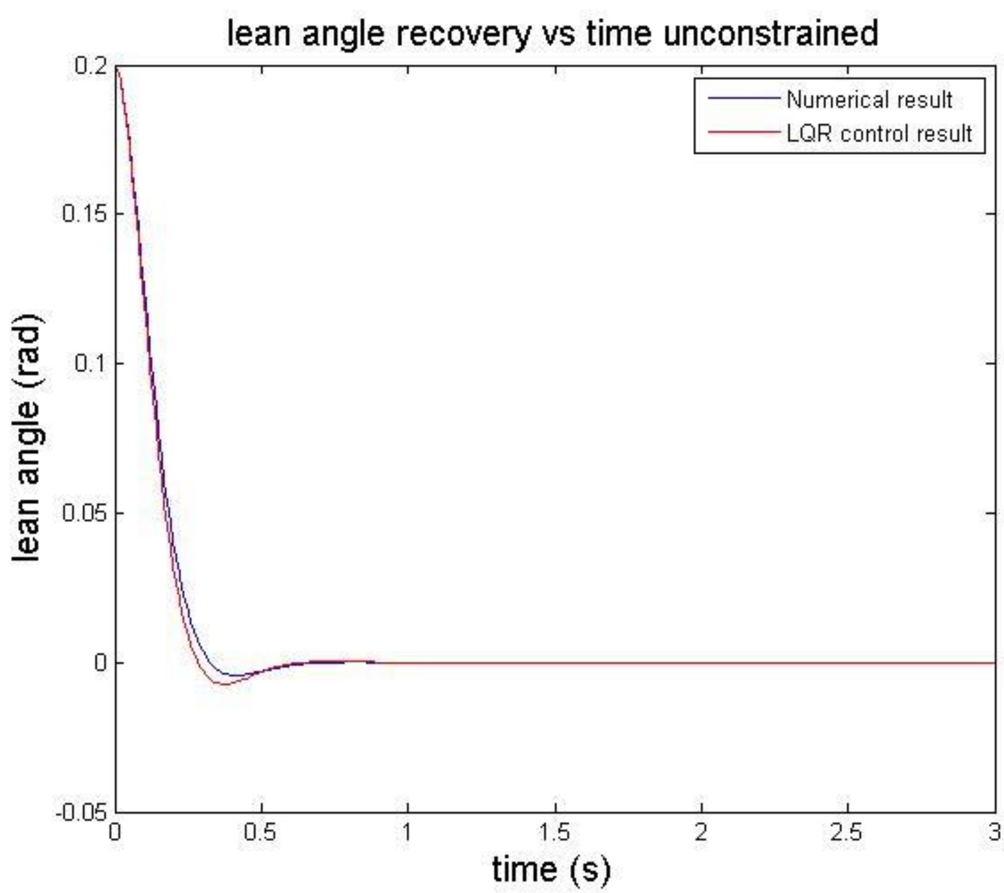

Figure 9: Comparison between Numerical and LQR results for lean angle

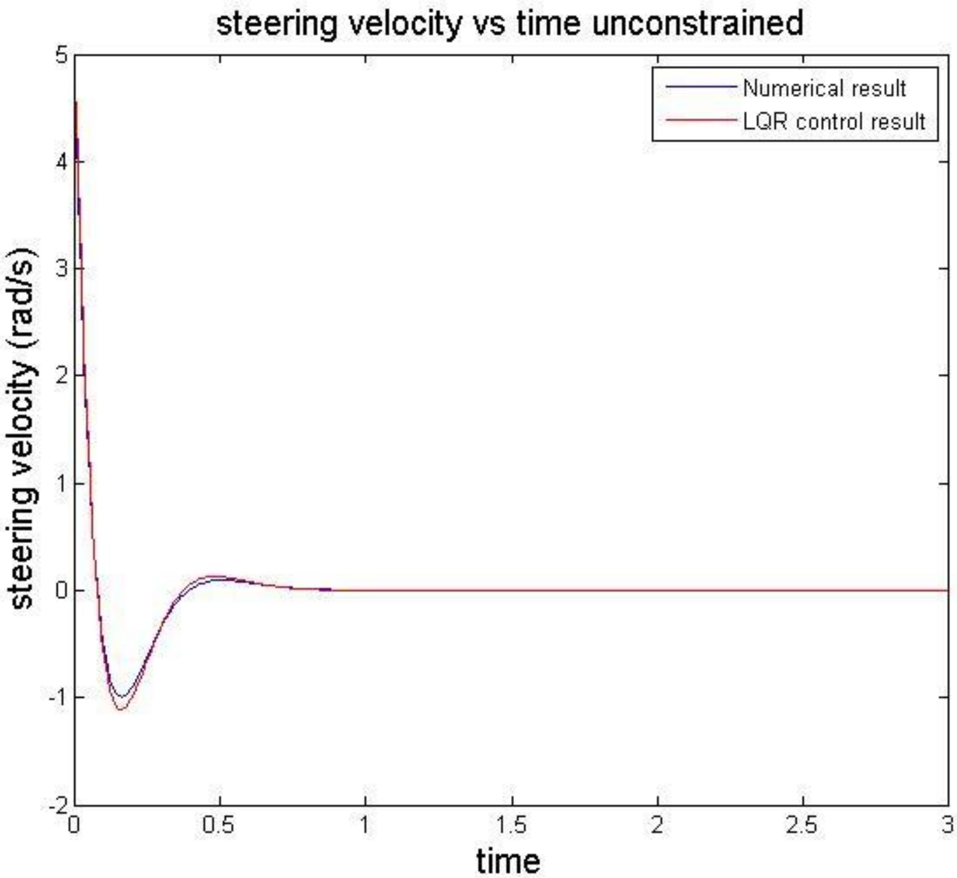

Figure10: Comparison between Numerical and LQR results for steering velocity

As shown in Figure 9 and 10, although the controller is able to bring the lean angle back to equilibrium in the shortest time possible, it exceeds the max angular velocity constraints.

After applying the mechanical constraint on the steering velocity to the numerical optimization method, the results are shown in Figure 11 and 12.

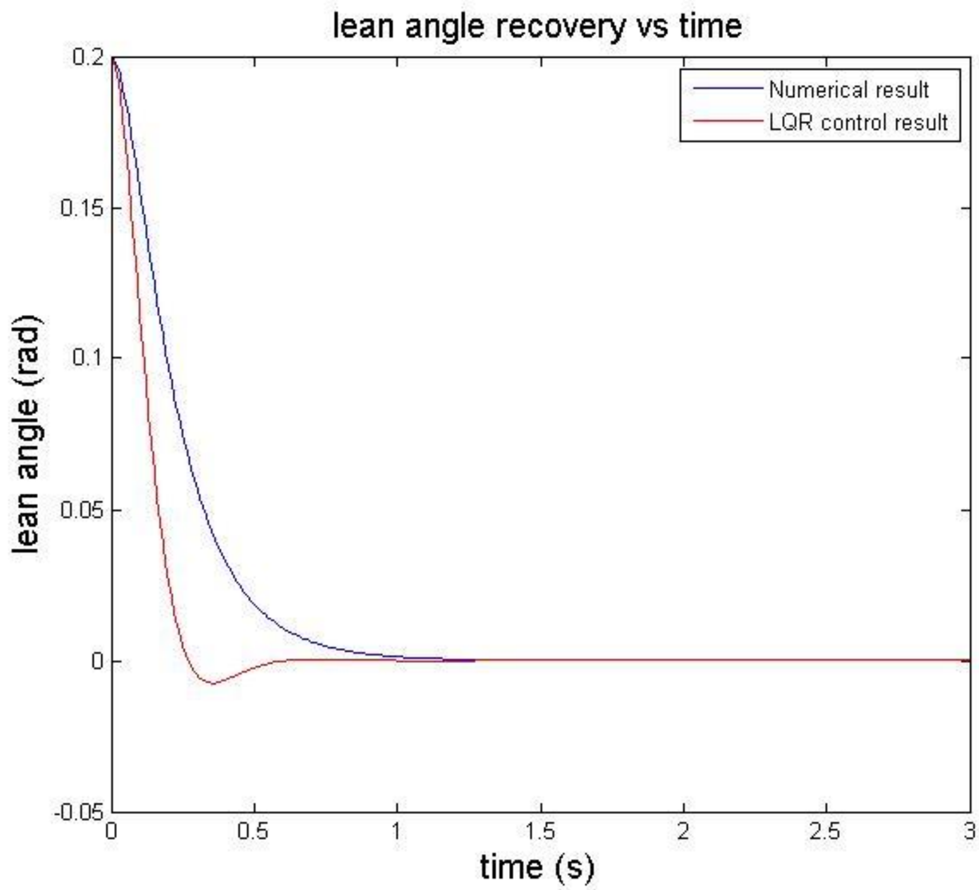

Figure11: Comparison between LQR and numerical results for lean angle with constraint

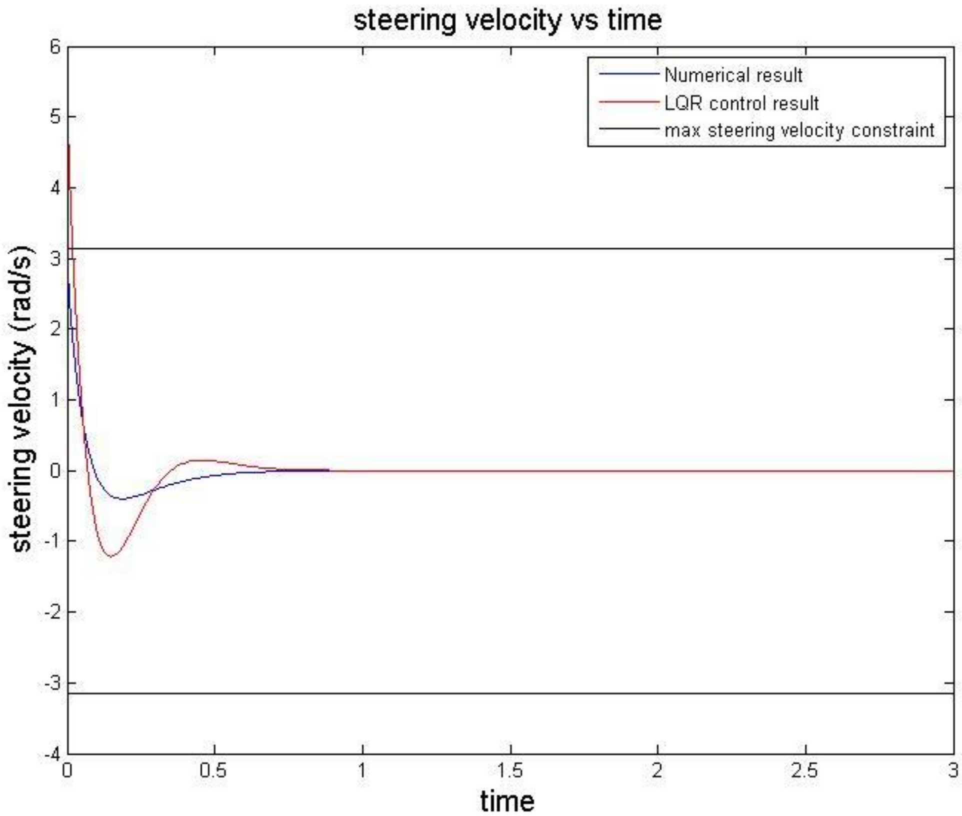

Figure12: Comparison between LQR and numerical results for steering velocity with constraint

As shown in Figure 11, the LQR control method performs better in stabilizing the lean angle; however, shown in Figure 12, it exceeds the max steering velocity constraint. Numerical optimization method, on the other hand, satisfies the lean angle settling time requirement and as well as the max steering velocity constraint. To check that this numerical method pushes the limit of the mechanical constraint, the max steering velocity for the numerical method was found to be 3.1381 rad/s, which was slightly smaller than the max velocity constraint, π rad/s, with an error of 0.11%.

As discussed earlier, Q and R matrix in the LQR design can be adjusted to change the weights on the input and the output. Values in Q and R matrix were manually changed until the steering velocity constraint was also satisfied.

The results are shown in Figure 13 and 14.

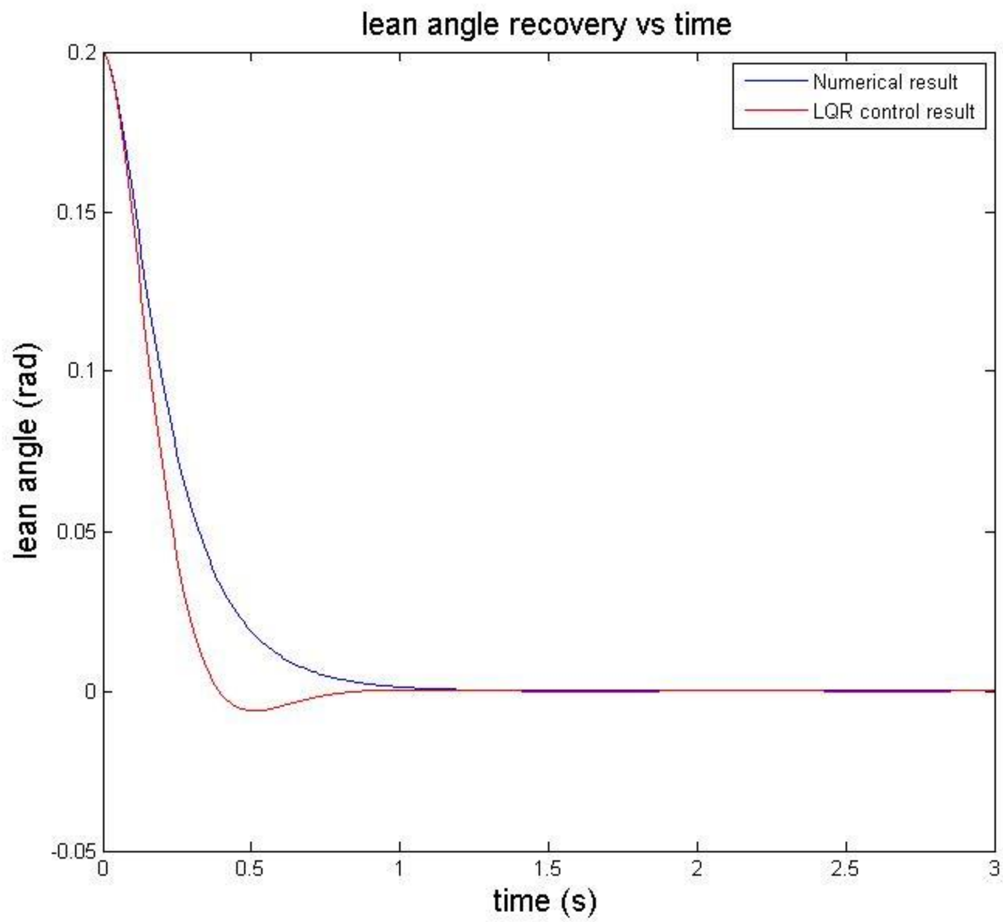

Figure13: Comparison between tuned LQR and numerical results for lean angle with constraint

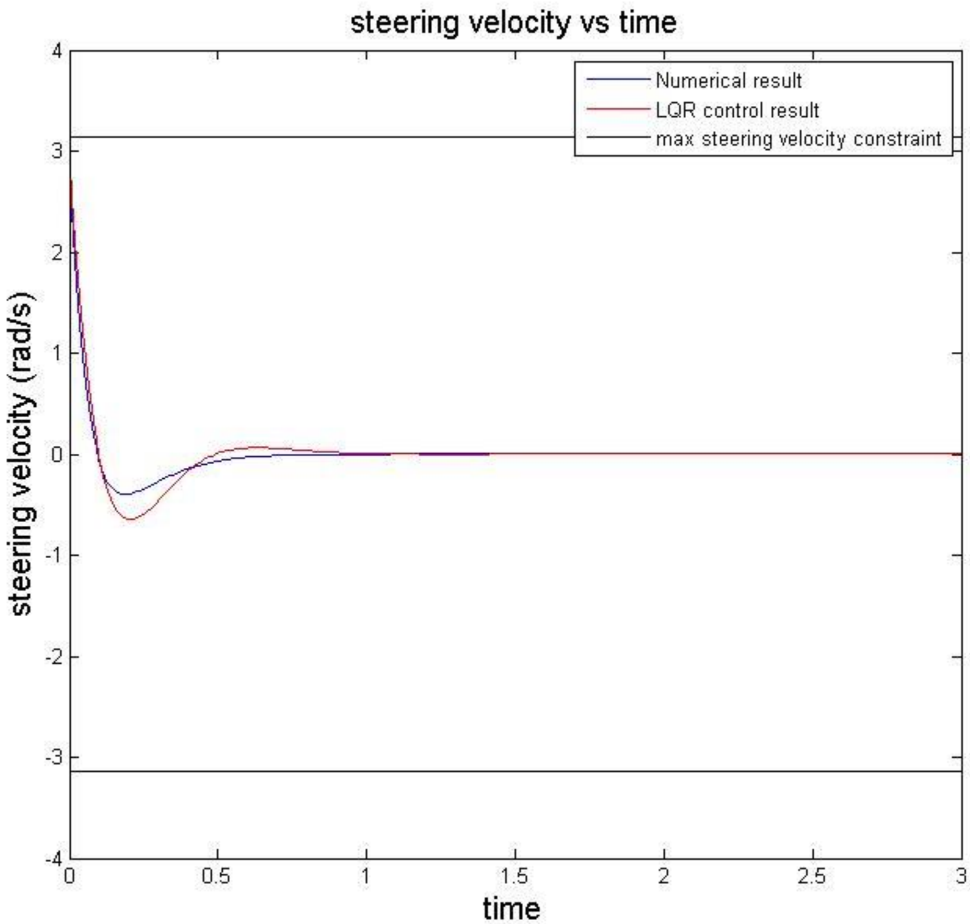

Figure14: Comparison between tuned LQR and numerical results for steering velocity with constraint

As shown in Figure 13 and 14, the LQR control results are similar to what accomplished by the numerical optimization program.

Advantages and Limitations

The developed numerical optimization method has the following advantages:

1. This numerical method does not require equations written in state-space form. ODE45 can be used inside of the cost function.
2. This numerical method does not require much brain power in tuning the gains in the correct direction for the correct amount. Instead, the code do the all these work automatically.
3. This numerical method can be easily modified to be applied to other control problems.

Although the developed numerical optimization method has several advantages, it also has its limitations.

1. This method is not robust enough. It is sensitive to the initial estimation of the controller. If the initial estimation is far from a reasonable controller, the code stuck into the local minimum and output faulty results.
2. This method requires long compiling time. Usually takes up to 7 min for the optimization code to finish depending on the initial estimation. This can be improved by replacing the current code with more elegantly written algorithm to reduce compiling time.

Animation and Demonstrations

After the controller was successfully designed, 3D animation was made in Matlab for demonstration.

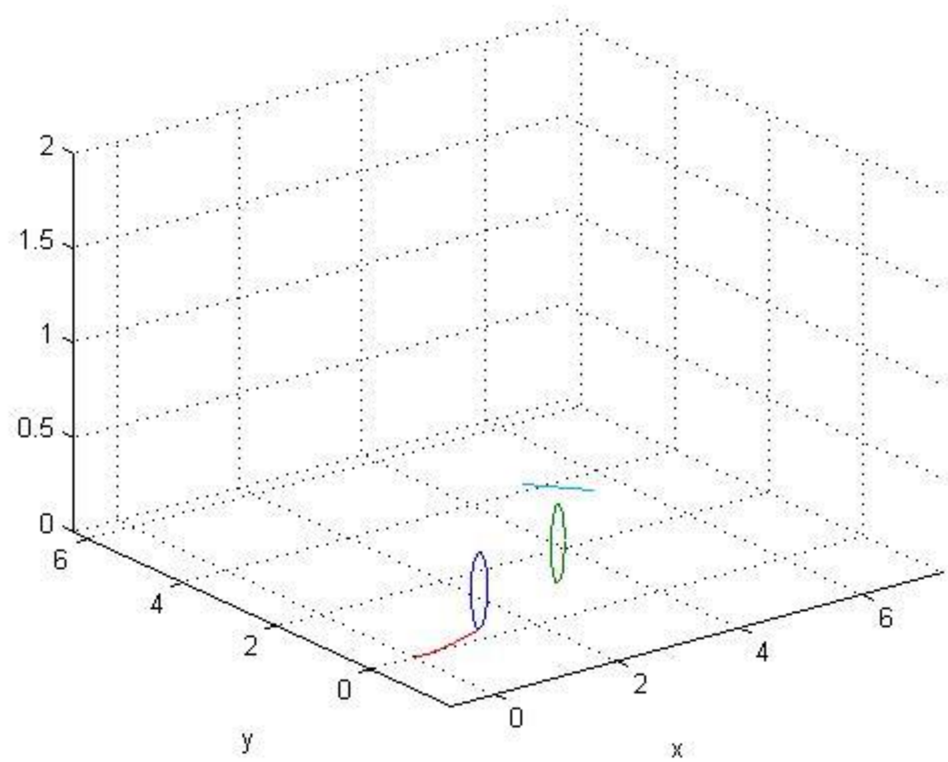

Figure15: 3D animation of the bicycle

This animation can be demonstrated by running animation.m in the attached Matlab files folder.

Conclusion and Future Work

This paper has presented the dynamic model and controller used to develop a self-stabilizing bicycle. The modeling and controller design of the bicycle is finished. The developed controller achieves the design objective based on numerical simulation. The control of the bicycle in this paper was based on a linearized model. To fully and more accurately describe the dynamics of the bicycle, non-linear model shall be investigated in the future.

Reference

[1] J. Meijaard, J.M. Papadopoulos, A. Ruina, and A. Schwab, Linearized dynamics equations for the balance and steer of a bicycle: a benchmark and review, Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science 463 (2007), pp. 1955{1982.

[2] Moore, Jason. "Human Control a Bicycle." (2014). Web. <http://moorepants.github.io/dissertation/control.html#ideal-control-models>.

[3] Jinghua Zhong, Mechanical Engineering, Purdue University (Spring 2006). "PID Controller Tuning: A Short Tutorial". Retrieved 2013-12-04.

[4] Anderson, Brian DO, and John B. Moore. *Optimal control: linear quadratic methods*. Courier Dover Publications, 2007.

[5] Hand, Richard Scott, 1963-. Comparisons And Stability Analysis of Linearized Equations of Motion for a Basic Bicycle Model. , 1988.

Appendix A: Derivation of Matrix [1]

$$
\begin{aligned}
m_{\mathrm{T}} &= m_{\mathrm{R}} + m_{\mathrm{B}} + m_{\mathrm{H}} + m_{\mathrm{F}}, && \text{(A 1)}\\
x_{\mathrm{T}} &= (x_{\mathrm{B}} m_{\mathrm{B}} + x_{\mathrm{H}} m_{\mathrm{H}} + w m_{\mathrm{F}})/m_{\mathrm{T}}, && \text{(A 2)}\\
z_{\mathrm{T}} &= (-r_{\mathrm{R}} m_{\mathrm{R}} + z_{\mathrm{B}} m_{\mathrm{B}} + z_{\mathrm{H}} m_{\mathrm{H}} - r_{\mathrm{F}} m_{\mathrm{F}})/m_{\mathrm{T}}. && \text{(A 3)}
\end{aligned}
$$

For the system as a whole, the relevant mass moments and products of inertia with respect to the rear contact point P along the global axes are

$$
\begin{aligned}
I_{\mathrm{T}xx} &= I_{\mathrm{R}xx} + I_{\mathrm{B}xx} + I_{\mathrm{H}xx} + I_{\mathrm{F}xx} + m_{\mathrm{R}} r_{\mathrm{R}}^2 + m_{\mathrm{B}} z_{\mathrm{B}}^2 + m_{\mathrm{H}} z_{\mathrm{H}}^2 + m_{\mathrm{F}} r_{\mathrm{F}}^2, && \text{(A 4)}\\
I_{\mathrm{T}xz} &= I_{\mathrm{B}xz} + I_{\mathrm{H}xz} - m_{\mathrm{B}} x_{\mathrm{B}} z_{\mathrm{B}} - m_{\mathrm{H}} x_{\mathrm{H}} z_{\mathrm{H}} + m_{\mathrm{F}} w r_{\mathrm{F}}. && \text{(A 5)}
\end{aligned}
$$

The dependent moments of inertia for the axisymmetric rear wheel and front wheel are

$$
I_{\mathrm{R}zz} = I_{\mathrm{R}xx}, \quad I_{\mathrm{F}zz} = I_{\mathrm{F}xx}. \qquad \text{(A 6)}
$$

Then the moment of inertia for the whole bicycle along the z-axis is

$$
I_{\mathrm{T}zz} = I_{\mathrm{R}zz} + I_{\mathrm{B}zz} + I_{\mathrm{H}zz} + I_{\mathrm{F}zz} + m_{\mathrm{B}} x_{\mathrm{B}}^2 + m_{\mathrm{H}} x_{\mathrm{H}}^2 + m_{\mathrm{F}} w^2. \qquad \text{(A 7)}
$$

The same properties are similarly defined for the front assembly A:

$$
\begin{aligned}
m_{\mathrm{A}} &= m_{\mathrm{H}} + m_{\mathrm{F}}, && \text{(A 8)}\\
x_{\mathrm{A}} = (x_{\mathrm{H}} m_{\mathrm{H}} + w m_{\mathrm{F}})/m_{\mathrm{A}}, &\qquad z_{\mathrm{A}} = (z_{\mathrm{H}} m_{\mathrm{H}} - r_{\mathrm{F}} m_{\mathrm{F}})/m_{\mathrm{A}}. && \text{(A 9)}
\end{aligned}
$$

The relevant mass moments and products of inertia for the front assembly with respect to the centre of mass of the front assembly along the global axes are

$$
\begin{aligned}
I_{\mathrm{A}xx} &= I_{\mathrm{H}xx} + I_{\mathrm{F}xx} + m_{\mathrm{H}}(z_{\mathrm{H}} - z_{\mathrm{A}})^2 + m_{\mathrm{F}}(r_{\mathrm{F}} + z_{\mathrm{A}})^2, && \text{(A 10)}\\
I_{\mathrm{A}xz} &= I_{\mathrm{H}xz} - m_{\mathrm{H}}(x_{\mathrm{H}} - x_{\mathrm{A}})(z_{\mathrm{H}} - z_{\mathrm{A}}) + m_{\mathrm{F}}(w - x_{\mathrm{A}})(r_{\mathrm{F}} + z_{\mathrm{A}}), && \text{(A 11)}\\
I_{\mathrm{A}zz} &= I_{\mathrm{H}zz} + I_{\mathrm{F}zz} + m_{\mathrm{H}}(x_{\mathrm{H}} - x_{\mathrm{A}})^2 + m_{\mathrm{F}}(w - x_{\mathrm{A}})^2. && \text{(A 12)}
\end{aligned}
$$

Let $\boldsymbol{\lambda} = (\sin\lambda, 0, \cos\lambda)^T$ be a unit vector pointing down along the steer axis where λ is the angle in the xz-plane between the downward steering axis and the $+z$ direction. The centre of mass of the front assembly is ahead of the steering axis by perpendicular distance

$$
u_{\mathrm{A}} = (x_{\mathrm{A}} - w - c)\cos\lambda - z_{\mathrm{A}}\sin\lambda. \qquad \text{(A 13)}
$$

$$
\begin{aligned}
I_{A\lambda\lambda} &= m_A u_A^2 + I_{Axx}\sin^2\lambda + 2I_{Axz}\sin\lambda\cos\lambda + I_{Azz}\cos^2\lambda, && \text{(A 14)}\\
I_{A\lambda x} &= -m_A u_A z_A + I_{Axx}\sin\lambda + I_{Axz}\cos\lambda, && \text{(A 15)}\\
I_{A\lambda z} &= m_A u_A x_A + I_{Axz}\sin\lambda + I_{Azz}\cos\lambda. && \text{(A 16)}
\end{aligned}
$$

The ratio of the mechanical trail (i.e., the perpendicular distance that the front wheel contact point is behind the steering axis) to the wheel base is

$$\mu = (c/w)\cos\lambda. \qquad \text{(A 17)}$$

The rear and front wheel angular momenta along the y-axis, divided by the forward speed, together with their sum form the gyrostatic coefficients:

$$S_R = I_{Ryy}/r_R, \quad S_F = I_{Fyy}/r_F, \quad S_T = S_R + S_F. \qquad \text{(A 18)}$$

We define a frequently appearing static moment term as

$$S_A = m_A u_A + \mu m_T x_T. \qquad \text{(A 19)}$$

The entries in the linearized equations of motion can now be formed. The mass moments of inertia

$$
\begin{aligned}
M_{\phi\phi} &= I_{Txx} \quad , \quad M_{\phi\delta} = I_{A\lambda x} + \mu I_{Txz},\\
M_{\delta\phi} &= M_{\phi\delta} \quad , \quad M_{\delta\delta} = I_{A\lambda\lambda} + 2\mu I_{A\lambda z} + \mu^2 I_{Tzz}, \qquad \text{(A 20)}
\end{aligned}
$$

are elements of the symmetric mass matrix $\mathbf{M} = \begin{bmatrix} M_{\phi\phi} & M_{\phi\delta} \\ M_{\delta\phi} & M_{\delta\delta} \end{bmatrix}$. (A 21)

The gravity-dependent stiffness terms (to be multiplied by g) are

$$
\begin{aligned}
K_{0\phi\phi} &= m_T z_T \quad , \quad K_{0\phi\delta} = -S_A,\\
K_{0\delta\phi} &= K_{0\phi\delta} \quad , \quad K_{0\delta\delta} = -S_A\sin\lambda, \qquad \text{(A 22)}
\end{aligned}
$$

which form the stiffness matrix $\mathbf{K_0} = \begin{bmatrix} K_{0\phi\phi} & K_{0\phi\delta} \\ K_{0\delta\phi} & K_{0\delta\delta} \end{bmatrix}$. (A 23)

The velocity-dependent stiffness terms (to be multiplied by v^2) are

$$
\begin{aligned}
K_{2\phi\phi} &= 0 \quad , \quad K_{2\phi\delta} = ((S_T - m_T z_T)/w)\cos\lambda,\\
K_{2\delta\phi} &= 0 \quad , \quad K_{2\delta\delta} = ((S_A + S_F\sin\lambda)/w)\cos\lambda, \qquad \text{(A 24)}
\end{aligned}
$$

which form the stiffness matrix $\mathbf{K_2} = \begin{bmatrix} K_{2\phi\phi} & K_{2\phi\delta} \\ K_{2\delta\phi} & K_{2\delta\delta} \end{bmatrix}$. (A 25)

In the equations we use $\mathbf{K} = g\mathbf{K_0} + v^2\mathbf{K_2}$. Finally the "damping" terms are

$$C_{1\phi\phi} = 0, \quad C_{1\phi\delta} = \mu S_T + S_F\cos\lambda + (I_{Txz}/w)\cos\lambda - \mu m_T z_T, \qquad \text{(A 26)}$$

$$C_{1\delta\phi} = -(\mu S_T + S_F\cos\lambda), \quad C_{1\delta\delta} = (I_{A\lambda z}/w)\cos\lambda + \mu(S_A + (I_{Tzz}/w)\cos\lambda),$$

which form $\mathbf{C_1} = \begin{bmatrix} C_{1\phi\phi} & C_{1\phi\delta} \\ C_{1\delta\phi} & C_{1\delta\delta} \end{bmatrix}$ where we use $\mathbf{C} = v\mathbf{C_1}$. (A 27)

Appendix B: Eigenvalues of the solution of the bicycle motion [1]

v [m/s]	$\mathrm{Re}(\lambda_{\text{weave}})$ [1/s]	$\mathrm{Im}(\lambda_{\text{weave}})$ [1/s]
0	–	–
1	3.526 961 709 900 70	0.807 740 275 199 30
2	2.682 345 175 127 45	1.680 662 965 906 75
3	1.706 756 056 639 75	2.315 824 473 843 25
4	0.413 253 315 211 25	3.079 108 186 032 06
5	−0.775 341 882 195 85	4.464 867 713 788 23
6	−1.526 444 865 841 42	5.876 730 605 987 09
7	−2.138 756 442 583 62	7.195 259 133 298 05
8	−2.693 486 835 810 97	8.460 379 713 969 31
9	−3.216 754 022 524 85	9.693 773 515 317 91
10	−3.720 168 404 372 87	10.906 811 394 762 87

Appendix C: Matlab Code

```
clear all; close all; clc;

load foranimation.mat
 %% animation
z=zeros(1,length(x))';
theta=0:0.2:2*pi;
ycircle=r_R*cos(theta);
zcircle=r_R*sin(theta)+r_R;
xcircle=zeros(1,length(ycircle));

rearwheel=[xcircle;ycircle;zcircle];

handlex=-.5:0.5;
handley=zeros(1,length(handlex));
handlez=zeros(1,length(handlex));
handlebar=[handlex;handley;handlez];
w=2;
height=0.5;
for ii=1:length(x)

yaww=yaw(ii);
leann=phi(ii);
steerr=delta(ii);

% plot3(frontwheel(1,:),frontwheel(2,:),frontwheel(3,:))
% grid on;
yaw_rotation=[cos(pi/2-yaww) sin(pi/2-yaww) 0;...
            -sin(pi/2-yaww)  cos(pi/2-yaww) 0;...
            0 0 1];
lean_rotation=[cos(leann) 0 -sin(leann);...
                   0      1       0;...
               sin(leann) 0 cos(leann)];
steer_rotation=[  cos(-steerr) sin(-steerr) 0;...
                    -sin(-steerr)  cos(-steerr) 0;...
                    0 0 1];

frontwheelnew=yaw_rotation* lean_rotation*steer_rotation*rearwheel;
rearwheelnew= yaw_rotation* lean_rotation*rearwheel;
handlebarnew=yaw_rotation* lean_rotation*steer_rotation*handlebar;

xrearwheelnew=rearwheelnew(1,:)+x(ii);
yrearwheelnew=rearwheelnew(2,:)+y(ii);
zrearwheelnew=rearwheelnew(3,:);

xfrontwheelnew=frontwheelnew(1,:)+x(ii)+w*cos(yaww);
yfrontwheelnew=frontwheelnew(2,:)+y(ii)+w*sin(yaww);
zfrontwheelnew=frontwheelnew(3,:);

xhandlenew=handlebarnew(1,:)+x(ii)+w*cos(yaww);
yhandlenew=handlebarnew(2,:)+y(ii)+w*sin(yaww);
zhandlenew=handlebarnew(3,:)+height;
```

```
figure(8)
plot3(xrearwheelnew,yrearwheelnew,zrearwheelnew,xfrontwheelnew,yfrontwh
eelnew,zfrontwheelnew,...
       x(1:ii),y(1:ii),z(1:ii),xhandlenew,yhandlenew,zhandlenew);

grid on;
axis([(x(ii))-2 (x(ii)+6) (y(ii))-2 (y(ii)+6) 0 (x(1)+2)])
xlabel('x')
ylabel('y')
zlabel('z')
pause(0.001)
end




function CONST=BikeConstant()
%% this data was obtained from the benchmark paper


w=.889;          %Wheelbase (m)
c=0.06;          %Trail (m)
tilt=pi/10;      %tile (rad)
g=9.81;          %gravity (m/(s^2))


v=4;


%% bicycle by component

%Rear B and frame
x_B=0.105;
z_B=-0.2962;        %Position of center of mass (m)

m_B=7.8245;          %mass (kg)


I_Bxx=0.2174;       %kg m^2
I_Byy=0.3588;        %kg m^2
I_Bzz=0.1531;       %kg m^2
I_Bxz=0.0371;       %kg m^2

I_B=[I_Bxx,0,I_Bxz;...
     0,I_Byy,0;...
     I_Bxz,0,I_Bzz];

 %Rear Wheel R

r_R=0.2032;          %radius (m)
m_R=1.4515;            %mass (kg)
```

```
I_Rxx=0.01583;    %Moment of inertia(kg m^2)
I_Ryy=0.0299;      %Moment of inertia(kg m^2)



%Front Handlebar and Fork Assembly H

x_H=0.78;
z_H=-0.453;       %Position of center of mass (m)

m_H=12.079;          %mass (kg)

I_Hxx=0.00692;      %kg m^2
I_Hyy=0.00056;      %kg m^2
I_Hzz=0.00654;      %kg m^2
I_Hxz=-0.00013;     %kg m^2

I_H=[I_Hxx,0,I_Hxz;...
     0,I_Hyy,0;...
     I_Hxz,0,I_Hzz];

%Front Wheel F


r_F=0.2032;        %radius (m)
m_F=7.022;           %mass (kg)
I_Fxx=0.07391;    %Moment of inertia(kg m^2)
I_Fyy=0.14478;      %Moment of inertia(kg m^2)

%% Whole Bike

m_T = m_R + m_B + m_H + m_F;            %Total mass

x_T = (m_B*x_B + m_H*x_H + m_F*w)/m_T;               %Total center of
mass
z_T = (-r_R*m_R + z_B*m_B + z_H*m_H -r_F*m_F)/m_T;   %(wrt contact
point P)



%% create a constant struct

CONST.w=w;          %Wheelbase (m)
CONST.c=c;          %Trail (m)
CONST.tilt=tilt;      %tile (rad)
CONST.g=g;          %gravity (m/(s^2))
CONST.v=v;             %forward velocity (m/s)


%Rear B and frame

CONST.x_B=x_B;
```

```
CONST.z_B=z_B;        %Position of center of mass (m)

CONST.m_B=m_B;          %mass (kg)

CONST.I_Bxx=I_Bxx;      %kg m^2
CONST.I_Byy=I_Byy;       %kg m^2
CONST.I_Bzz=I_Bzz;      %kg m^2
CONST.I_Bxz=I_Bxz;      %kg m^2

CONST.I_B=I_B;

%Front Handlebar and Fork Assembly H

CONST.x_H=x_H;
CONST.z_H=z_H;        %Position of center of mass (m)

CONST.m_H=m_H;           %mass (kg)

CONST.I_Hxx=I_Hxx;      %kg m^2
CONST.I_Hyy=I_Hyy;         %kg m^2
CONST.I_Hzz=I_Hzz;      %kg m^2
CONST.I_Hxz=I_Hxz;     %kg m^2

CONST.I_H=I_H;

%Rear Wheel R

CONST.r_R=r_R;          %radius (m)
CONST.m_R=m_R;            %mass (kg)
CONST.I_Rxx=I_Rxx;     %Moment of inertia(kg m^2)
CONST.I_Ryy=I_Ryy;       %Moment of inertia(kg m^2)

%Front Wheel F

CONST.r_F=r_F;         %radius (m)
CONST.m_F=m_F;            %mass (kg)
CONST.I_Fxx=I_Fxx;     %Moment of inertia(kg m^2)
CONST.I_Fyy=I_Fyy;       %Moment of inertia(kg m^2)

%Whole Bike

CONST.m_T = m_T;            %Total mass
CONST.x_T = x_T;               %Total center of mass
CONST.z_T = z_T;   %(wrt contact point P)


end

function target = BikeOpt( CtrlM, p )

max_steer_velocity=pi;
p.CtrlM=CtrlM;
%Run a simulation inside the optimization
```

```
[t,k] = ode45(@
bikeStateODE_pointmass_forced,p.tspan,p.ini,p.options,p);
lean_angle=k(:,1);
steer_angle=k(:,2);
lean_velocity=k(:,3);

deltadot=-CtrlM(1)*lean_angle-CtrlM(2)*steer_angle-
CtrlM(3)*lean_velocity;

if abs(deltadot)<max_steer_velocity

target=sum(abs(lean_angle));

else
   target=sum(abs(lean_angle))*10;
end

end


function [ derv ] = bikeStateODE( t,k,Matrix)

K=Matrix.K;
C=Matrix.C;
M=Matrix.M;
T=Matrix.T;

Mi=(Matrix.M)^-1;
% T=Matrix.T;
MiK=Mi*K;
MiC=Mi*C;

phi=k(1);
delta=k(2);
phidot=k(3);
% deltadot=k(4);
deltadot=k(4);


A=[0            0            1         0;...
   0            0            0         0;...
 -MiK(1,1)     -MiK(1,2)    -MiC(1,1)  0;...
 -MiK(2,1)     -MiK(2,2)    -MiC(2,1)  0];

B=[0;...
   1;...
   -MiC(1,2);...
   -MiC(2,2)];


% z=M^-1*(-C*[phidot deltadot]'-K*[phi delta]'+T);

kdot = A*k+B*deltadot;    %Z dot of Bicycle rotation and motor current
```

```
% phi_doubledot=z(1);
% delta_doubledot=z(2);


% derv(1)=phidot;
% derv(2)=deltadot;
% derv(3)=phi_doubledot;
% derv(4)=delta_doubledot;

derv=kdot;
end

function [ derv ] = bikeStateODE_forced( t,k,p)


K=p.Matrix.K;
C=p.Matrix.C;
M=p.Matrix.M;
T=p.Matrix.T;
CtrlM=p.CtrlM;
Mi=(M)^-1;
% T=Matrix.T;
MiK=Mi*K;
MiC=Mi*C;

phi=k(1);
delta=k(2);
phidot=k(3);
% deltadot=k(4);
deltadot=k(4);


A=[0            0            1        0;...
   0            0            0        0;...
 -MiK(1,1)     -MiK(1,2)    -MiC(1,1)  0;...
 -MiK(2,1)     -MiK(2,2)    -MiC(2,1)  0];

B=[0;...
   1;...
   -MiC(1,2);...
   -MiC(2,2)];

deltadot_control=-CtrlM*[phi;delta;phidot];

% z=M^-1*(-C*[phidot deltadot]'-K*[phi delta]'+T);

kdot = A*k+B*deltadot_control;    %Z dot of Bicycle rotation and motor
current

% phi_doubledot=z(1);
% delta_doubledot=z(2);
```

```
% derv(1)=phidot;
% derv(2)=deltadot;
% derv(3)=phi_doubledot;
% derv(4)=delta_doubledot;

derv=kdot;
end


 function [ derv ] = bikeStateODE_pointmass( t,k,Matrix,CtrlM)

K=Matrix.K;
C=Matrix.C;
Mi=(Matrix.M)^-1;
% T=Matrix.T;
MiK=Mi*K;
MiC=Mi*C;
% T_phi=0;

phi=k(1);
delta=k(2);
phidot=k(3);


A=[0                 0              1;...
   0                 0              0;...
   -MiK(1,1)     -MiK(1,2)          0];...

B=[0;...
   1;...
   -MiC(1,2)];...

delta_dot=0.48*phi;  % control,


kdot=A*k+B*delta_dot;     %Z dot of Bicycle rotation and motor current

derv=kdot;


end

function [ derv ] = bikeStateODE_pointmass_forced( t,k,p)

K=p.Matrix.K;
C=p.Matrix.C;
Mi=(p.Matrix.M)^-1;
% T=Matrix.T;
MiK=Mi*K;
MiC=Mi*C;

% T_phi=0;
```

```
CtrlM=p.CtrlM;
phi=k(1);
delta=k(2);
phidot=k(3);
yaw=k(4);

p.t=t;

A=[0                0              1;...
   0                0              0;...
   -MiK(1,1)      -MiK(1,2)        0];...

B=[0;...
   1;...
   -MiC(1,2)];...

v=p.v;
trail=p.trail;
wheelbase=p.wheelbase;
tilt=p.tilt;

deltadot_control=-CtrlM'*[phi;delta;phidot];


yawdot=(v*delta+trail*deltadot_control/wheelbase)*cos(tilt);
xdot=v*cos(yaw);
ydot=v*sin(yaw);

kdot=A*k(1:3,:)+B*deltadot_control;    %Z dot of Bicycle rotation and
motor current
kdot2=[yawdot;xdot;ydot];

derv=[kdot',kdot2']';


end

clear all; clc; close all;
M = .5;
m = 0.2;
b = 0.1;
I = 0.006;
g = 9.8;
l = 0.3;

p = I*(M+m)+M*m*l^2; %denominator for the A and B matrices

A = [0      1              0           0;
     0 -(I+m*l^2)*b/p  (m^2*g*l^2)/p   0;
     0      0              0           1;
     0 -(m*l*b)/p       m*g*l*(M+m)/p  0];
B = [     0;
     (I+m*l^2)/p;
          0;
```

```
        m*l/p];
C = [1 0 0 0;
     0 0 1 0];
D = [0;
     0];

states = {'x' 'x_dot' 'phi' 'phi_dot'};
inputs = {'u'};
outputs = {'x'; 'phi'};

sys_ss =
ss(A,B,C,D,'statename',states,'inputname',inputs,'outputname',outputs);
Q = C'*C;
%
Q(1,1) =10;
Q(3,3) = 1
R = 1;
K = lqr(A,B,Q,R)


Ac = [(A-B*K)];
Bc = [B];
Cc = [C];
Dc = [D];

states = {'x' 'x_dot' 'phi' 'phi_dot'};
inputs = {'r'};
outputs = {'x'; 'phi'};

sys_cl =
ss(Ac,Bc,Cc,Dc,'statename',states,'inputname',inputs,'outputname',outpu
ts);

t = 0:0.01:3;
r =0.2*ones(size(t));
x0=[0 0 0 0];
[y,t,x]=lsim(sys_cl,r,t,x0);
[AX,H1,H2] = plotyy(t,y(:,1),t,y(:,2),'plot');
set(get(AX(1),'Ylabel'),'String','cart position (m)')
set(get(AX(2),'Ylabel'),'String','pendulum angle (radians)')
title('Step Response with LQR Control')

clear all; close all; clc;


%% Loading Const
CONST=BikeConstants; % call bike constant function

w=CONST.w;          %Wheelbase (m)
c=CONST.c;          %Trail (m)
tilt=CONST.tilt;      %tile (rad)
g=CONST.g;          %gravity (m/(s^2))
v=CONST.v;             %forward velocity (m/s)
```

```
%Rear B and frame

x_B=CONST.x_B;
z_B=CONST.z_B;       %Position of center of mass (m)

m_B=CONST.m_B;         %mass (kg)

I_Bxx=CONST.I_Bxx;      %kg m^2
I_Byy=CONST.I_Byy;       %kg m^2
I_Bzz=CONST.I_Bzz;      %kg m^2
I_Bxz=CONST.I_Bxz;      %kg m^2

I_B=CONST.I_B;

%Front Handlebar and Fork Assembly H

x_H=CONST.x_H;
z_H=CONST.z_H;       %Position of center of mass (m)

m_H=CONST.m_H;          %mass (kg)

I_Hxx=CONST.I_Hxx;      %kg m^2
I_Hyy=CONST.I_Hyy;         %kg m^2
I_Hzz=CONST.I_Hzz;      %kg m^2
I_Hxz=CONST.I_Hxz;     %kg m^2

I_H=CONST.I_H;

%Rear Wheel R

r_R=CONST.r_R;         %radius (m)
m_R=CONST.m_R;           %mass (kg)
I_Rxx=CONST.I_Rxx;    %Moment of inertia(kg m^2)
I_Ryy=CONST.I_Ryy;      %Moment of inertia(kg m^2)


%Front Wheel F

r_F=CONST.r_F;        %radius (m)
m_F=CONST.m_F;           %mass (kg)
I_Fxx=CONST.I_Fxx;    %Moment of inertia(kg m^2)
I_Fyy=CONST.I_Fyy;      %Moment of inertia(kg m^2)

%% Find M C K matrix


%Whole Bicycle

m_T = m_R + m_B + m_H + m_F;            %Total mass

x_T = (m_B*x_B + m_H*x_H + m_F*w)/m_T;              %Total center of
mass
```

```
z_T = (-r_R*m_R + z_B*m_B + z_H*m_H -r_F*m_F)/m_T;    %(wrt contact
point P)

I_Txx = I_Rxx + I_Bxx + I_Hxx + I_Fxx ...
        + m_R*(r_R^2) + m_B*(z_B^2) + m_H*(z_H^2) + m_F*(r_F^2);
I_Txz = I_Bxz + I_Hxz ...
        - m_B*x_B*z_B - m_H*x_H*z_H + m_F*w*r_F;     %Mass moments and
                                                      %products of
inertia
                                                      %(wrt contact point
P)

I_Rzz=I_Rxx;
I_Fzz=I_Fxx;    %Axisymmetrix front and rear wheels

I_Tzz=I_Rzz+I_Bzz+I_Hzz+I_Fzz...                      %Moment of inertia
    +(m_B*(x_B^2))+(m_H*(x_H^2))+m_F*(w^2);           %about z axis


%Front Assembly

m_A=m_H+m_F;    %Mass of assembly

x_A=(x_H*m_H + w*m_F)/m_A;
z_A=(z_H*m_H - r_F*m_F)/m_A;    %Location of center of mass

I_Axx = I_Hxx + I_Fxx ...                                         %Moment
of inertia
        + m_H*((z_H - z_A)^2) + m_F*((r_F + z_A)^2);              %of
front assembly
I_Axz = I_Hxz - m_H*(x_H-x_A)*(z_H-z_A) + m_F*(w-x_A)*(r_F+z_A);%About
its
I_Azz = I_Hzz + I_Fzz + m_H*((x_H-x_A)^2) + m_F*((w-x_A)^2);    %center
of mass

u_A=(x_A - w - c)*cos(tilt) - z_A*sin(tilt);    %Perpend. dist. between
front
                                                %assembly and steering
axis

I_Att = m_A*(u_A^2) + I_Axx*(sin(tilt)^2) ...                 %Mom of
inertia
        + 2*I_Axz*sin(tilt)*cos(tilt) + I_Azz*(cos(tilt)^2);%abt. steer
axis
I_Atx = -m_A*u_A*z_A + I_Axx*sin(tilt) + I_Axz*cos(tilt);   %Prod. of
inertia
I_Atz = m_A*u_A*x_A + I_Axz*sin(tilt) + I_Azz*cos(tilt);    %rel to
crossed
                                                            %skew axis

mu=(c/w)*cos(tilt);     %ratio of mechanical trail to wheel base

S_R=I_Ryy/r_R;
S_F=I_Fyy/r_F;
```

```
S_T=S_R+S_F;          %Gyroscopic coefficients

S_A=m_A*u_A + mu*m_T*x_T;   %Static moment term



% M matrix (mass matrix)

M_rr = I_Txx;                                  %roll roll
M_rs = I_Atx + mu*I_Txz;                       %roll steer
M_sr = M_rs;                                   %steer roll
M_ss = I_Att + 2*mu*I_Atz + (mu^2)*I_Tzz;      %steer steer

M=[M_rr,M_rs;...
   M_sr,M_ss];

% K0 matrix (gravity dependent stiffness terms)

K0_rr = m_T*z_T;                               %roll roll
K0_rs = -S_A;                                  %roll steer
K0_sr = K0_rs;                                 %steer roll
K0_ss = -S_A*sin(tilt);                        %steer steer

K0=[K0_rr,K0_rs;...
    K0_sr,K0_ss];

% K2 matrix (velocity dependent stiffness terms)

K2_rr = 0;                                     %roll roll
K2_rs = (S_T-m_T*z_T)*cos(tilt)/w;             %roll steer
K2_sr = 0;                                     %steer roll
K2_ss = (S_A+S_F*sin(tilt))*cos(tilt)/w;       %steer steer

K2=[K2_rr,K2_rs;...
    K2_sr,K2_ss];

% C1 matrix (damping terms)

C1_rr=0;
C1_rs=mu*S_T+S_F*cos(tilt)+(I_Txz*cos(tilt)/w)-mu*m_T*z_T;
C1_sr=-(mu*S_T+S_F*cos(tilt));
C1_ss=(I_Atz*cos(tilt)/w)+mu*(S_A+(I_Tzz*cos(tilt)/w));

C1=[C1_rr,C1_rs;...
    C1_sr,C1_ss];

% K and C Matrices (made of K0, K2 and C1 matrices)

K=g*K0 + (v^2)*K2;

C=v*C1;
Matrix.K=K;
Matrix.C=C;
```

```
Matrix.M=M;
torquephi=0;
torquedelta=0;
T=[torquephi torquedelta]';

Matrix.T=T;

%%
CtrlM=[-23.23    5.85   -2.60]'; % initial estimate of control vector
clear k k0
k0(1)=0.2;  %initial lean angle phi
k0(2)=0; %initial steering angle delta
k0(3)=0; %initial lean velocity phi dot
k0(4)=0; % initial yaw
k0(5)=0;  % initial x
k0(6)=0;  % initial y

p.options =odeset('abstol',1e-6, 'reltol', 1e-6);
tt=0.1/10;
tspan=[0:tt:3];
p.tstep=tt;
p.Matrix=Matrix;
p.ini=k0;
p.v=v;
p.trail=0;
p.wheelbase=w;
p.tilt=0;
p.tspan=tspan;
p.CtrlM=CtrlM;
p.deltadot=0;
[t,k]=ode45(@bikeStateODE_pointmass_forced,tspan,k0,p.options,p);
lean_angle=k(:,1);
steer_angle=k(:,2);
lean_velocity=k(:,3);

deltadot_num=-CtrlM(1)*lean_angle-CtrlM(2)*steer_angle-
CtrlM(3)*lean_velocity;

Mi=(M)^-1;
% T=Matrix.T;
MiK=Mi*K;
MiC=Mi*C;

A_lqr=[0                0               1;...
   0                0              0;...
   -MiK(1,1)     -MiK(1,2)         0];...

B_lqr=[0;...
   1;...
   -MiC(1,2)];...


D_lqr=[0 1]'; % output deltadot set as constraint

C_lqr=[1 0 0; ...
```

```
        0 0 0];


states = {'phi' 'delta' 'phidot'};
inputs = {'deltadot'};
outputs = {'phi'; 'deltadot'};

sys_ss =
ss(A_lqr,B_lqr,C_lqr,D_lqr,'statename',states,'inputname',inputs,'outpu
tname',outputs);
%%
Q = C_lqr'*C_lqr;
%
Q(1,1) =880;
R = 2;

K_lqr = lqr(A_lqr,B_lqr,Q,R)

%
Ac = [(A_lqr-B_lqr*K_lqr)];
Bc = [B_lqr];
Cc = [C_lqr];
Dc = [D_lqr];

states = {'phi' 'delta' 'phidot'};
inputs = {'deltadot'};
outputs = {'phi'; 'deltadot'};


sys_cl =
ss(Ac,Bc,Cc,Dc,'statename',states,'inputname',inputs,'outputname',outpu
ts);
%%

deltadot =zeros(size(tspan));

k0_lqr=k0(1:3);
[y,t,x]=lsim(sys_cl,deltadot,tspan,k0_lqr);

lean_angle_lqr=x(:,1);
steer_angle_lqr=x(:,2);
lean_velocity_lqr=x(:,3);

delta_dot_lqr=-K_lqr(1)*lean_angle_lqr-K_lqr(2)*steer_angle_lqr-
K_lqr(3)*lean_velocity_lqr;

figure(1)
plot(t,lean_angle);hold on;

plot(tspan,lean_angle_lqr,'r')
title('lean angle recovery vs time unconstrained','fontsize',15)
legend('Numerical result','LQR control result')
xlabel('time (s)','fontsize',15);
ylabel('lean angle (rad)','fontsize',15);
```

```
figure(2)
plot(t,deltadot_num);hold on;

plot(tspan,delta_dot_lqr,'r');hold on;

xlabel('time','fontsize',15);
ylabel('steering velocity (rad/s)','fontsize',15);
legend('Numerical result','LQR control result');
title('steering velocity vs time unconstrained','fontsize',15)



clear all; close all; clc;


%% Loading Const
CONST=BikeConstants; % call bike constant function

w=CONST.w;          %Wheelbase (m)
c=CONST.c;          %Trail (m)
tilt=CONST.tilt;      %tile (rad)
g=CONST.g;          %gravity (m/(s^2))
v=CONST.v;              %forward velocity (m/s)


%Rear B and frame

x_B=CONST.x_B;
z_B=CONST.z_B;        %Position of center of mass (m)

m_B=CONST.m_B;          %mass (kg)

I_Bxx=CONST.I_Bxx;       %kg m^2
I_Byy=CONST.I_Byy;        %kg m^2
I_Bzz=CONST.I_Bzz;       %kg m^2
I_Bxz=CONST.I_Bxz;       %kg m^2

I_B=CONST.I_B;

%Front Handlebar and Fork Assembly H

x_H=CONST.x_H;
z_H=CONST.z_H;        %Position of center of mass (m)

m_H=CONST.m_H;           %mass (kg)

I_Hxx=CONST.I_Hxx;       %kg m^2
I_Hyy=CONST.I_Hyy;          %kg m^2
I_Hzz=CONST.I_Hzz;       %kg m^2
I_Hxz=CONST.I_Hxz;      %kg m^2

I_H=CONST.I_H;
```

```
%Rear Wheel R

r_R=CONST.r_R;          %radius (m)
m_R=CONST.m_R;            %mass (kg)
I_Rxx=CONST.I_Rxx;     %Moment of inertia(kg m^2)
I_Ryy=CONST.I_Ryy;       %Moment of inertia(kg m^2)


%Front Wheel F

r_F=CONST.r_F;         %radius (m)
m_F=CONST.m_F;            %mass (kg)
I_Fxx=CONST.I_Fxx;     %Moment of inertia(kg m^2)
I_Fyy=CONST.I_Fyy;       %Moment of inertia(kg m^2)

%% Find M C K matrix


%Whole Bicycle

m_T = m_R + m_B + m_H + m_F;            %Total mass

x_T = (m_B*x_B + m_H*x_H + m_F*w)/m_T;              %Total center of
mass
z_T = (-r_R*m_R + z_B*m_B + z_H*m_H -r_F*m_F)/m_T;   %(wrt contact
point P)

I_Txx = I_Rxx + I_Bxx + I_Hxx + I_Fxx ...
        + m_R*(r_R^2) + m_B*(z_B^2) + m_H*(z_H^2) + m_F*(r_F^2);
I_Txz = I_Bxz + I_Hxz ...
        - m_B*x_B*z_B - m_H*x_H*z_H + m_F*w*r_F;    %Mass moments and
                                                    %products of
inertia
                                                    %(wrt contact point
P)

I_Rzz=I_Rxx;
I_Fzz=I_Fxx;    %Axisymmetrix front and rear wheels

I_Tzz=I_Rzz+I_Bzz+I_Hzz+I_Fzz...                    %Moment of inertia
    +(m_B*(x_B^2))+(m_H*(x_H^2))+m_F*(w^2);         %about z axis


%Front Assembly

m_A=m_H+m_F;    %Mass of assembly

x_A=(x_H*m_H + w*m_F)/m_A;
z_A=(z_H*m_H - r_F*m_F)/m_A;    %Location of center of mass

I_Axx = I_Hxx + I_Fxx ...                                       %Moment
of inertia
```

```
        + m_H*((z_H - z_A)^2) + m_F*((r_F + z_A)^2);            %of
front assembly
I_Axz = I_Hxz - m_H*(x_H-x_A)*(z_H-z_A) + m_F*(w-x_A)*(r_F+z_A);%About
its
I_Azz = I_Hzz + I_Fzz + m_H*((x_H-x_A)^2) + m_F*((w-x_A)^2);    %center
of mass

u_A=(x_A - w - c)*cos(tilt) - z_A*sin(tilt);    %Perpend. dist. between
front
                                                %assembly and steering
axis

I_Att = m_A*(u_A^2) + I_Axx*(sin(tilt)^2) ...               %Mom of
inertia
        + 2*I_Axz*sin(tilt)*cos(tilt) + I_Azz*(cos(tilt)^2);%abt. steer
axis
I_Atx = -m_A*u_A*z_A + I_Axx*sin(tilt) + I_Axz*cos(tilt);   %Prod. of
inertia
I_Atz = m_A*u_A*x_A + I_Axz*sin(tilt) + I_Azz*cos(tilt);    %rel to
crossed
                                                            %skew axis

mu=(c/w)*cos(tilt);     %ratio of mechanical trail to wheel base

S_R=I_Ryy/r_R;
S_F=I_Fyy/r_F;
S_T=S_R+S_F;        %Gyroscopic coefficients

S_A=m_A*u_A + mu*m_T*x_T;   %Static moment term



% M matrix (mass matrix)

M_rr = I_Txx;                               %roll roll
M_rs = I_Atx + mu*I_Txz;                    %roll steer
M_sr = M_rs;                                %steer roll
M_ss = I_Att + 2*mu*I_Atz + (mu^2)*I_Tzz;   %steer steer

M=[M_rr,M_rs;...
   M_sr,M_ss];

% K0 matrix (gravity dependent stiffness terms)

K0_rr = m_T*z_T;                            %roll roll
K0_rs = -S_A;                               %roll steer
K0_sr = K0_rs;                              %steer roll
K0_ss = -S_A*sin(tilt);                     %steer steer

K0=[K0_rr,K0_rs;...
    K0_sr,K0_ss];

% K2 matrix (velocity dependent stiffness terms)
```

```
K2_rr = 0;                                %roll roll
K2_rs = (S_T-m_T*z_T)*cos(tilt)/w;        %roll steer
K2_sr = 0;                                %steer roll
K2_ss = (S_A+S_F*sin(tilt))*cos(tilt)/w;  %steer steer

K2=[K2_rr,K2_rs;...
    K2_sr,K2_ss];

% C1 matrix (damping terms)

C1_rr=0;
C1_rs=mu*S_T+S_F*cos(tilt)+(I_Txz*cos(tilt)/w)-mu*m_T*z_T;
C1_sr=-(mu*S_T+S_F*cos(tilt));
C1_ss=(I_Atz*cos(tilt)/w)+mu*(S_A+(I_Tzz*cos(tilt)/w));

C1=[C1_rr,C1_rs;...
    C1_sr,C1_ss];

% K and C Matrices (made of K0, K2 and C1 matrices)

K=g*K0 + (v^2)*K2;

C=v*C1;
Matrix.K=K;
Matrix.C=C;
Matrix.M=M;
torquephi=0;
torquedelta=0;
T=[torquephi torquedelta]';

Matrix.T=T;

%%
CtrlM=[-15.1456 5.0135 -2.9987]'; % initial estimate of control vector
clear k k0
k0(1)=0.2;  %initial lean angle phi
k0(2)=0; %initial steering angle delta
k0(3)=0; %initial lean velocity phi dot
k0(4)=0; % initial yaw
k0(5)=0;  % initial x
k0(6)=0;  % initial y

p.options =odeset('abstol',1e-6, 'reltol', 1e-6);
tt=0.1/10;
tspan=[0:tt:3];
p.tstep=tt;
p.Matrix=Matrix;
p.ini=k0;
p.v=v;
p.trail=0;
p.wheelbase=w;
p.tilt=0;
p.tspan=tspan;
p.CtrlM=CtrlM;
p.deltadot=0;
```

```
[t,k]=ode45(@bikeStateODE_pointmass_forced,tspan,k0,p.options,p);
lean_angle=k(:,1);
steer_angle=k(:,2);
lean_velocity=k(:,3);

deltadot_num=-CtrlM(1)*lean_angle-CtrlM(2)*steer_angle-
CtrlM(3)*lean_velocity;
maxdeltadot=max(abs(deltadot_num))
Mi=(M)^-1;
% T=Matrix.T;
MiK=Mi*K;
MiC=Mi*C;

A_lqr=[0                0              1;...
   0                0              0;...
   -MiK(1,1)     -MiK(1,2)         0];...

B_lqr=[0;...
   1;...
   -MiC(1,2)];...


D_lqr=[0 1]'; % output deltadot set as constraint

C_lqr=[1 0 0; ...
       0 0 0];


states = {'phi' 'delta' 'phidot'};
inputs = {'deltadot'};
outputs = {'phi'; 'deltadot'};

sys_ss =
ss(A_lqr,B_lqr,C_lqr,D_lqr,'statename',states,'inputname',inputs,'outpu
tname',outputs);
%%
Q = C_lqr'*C_lqr;
%
Q(1,1) =280;
% Q(1,1) =280;
R = 2;

K_lqr = lqr(A_lqr,B_lqr,Q,R)

%
Ac = [(A_lqr-B_lqr*K_lqr)];
Bc = [B_lqr];
Cc = [C_lqr];
Dc = [D_lqr];

states = {'phi' 'delta' 'phidot'};
inputs = {'deltadot'};
outputs = {'phi'; 'deltadot'};
```

```
sys_cl =
ss(Ac,Bc,Cc,Dc,'statename',states,'inputname',inputs,'outputname',outpu
ts);
%%

deltadot =zeros(size(tspan));

k0_lqr=k0(1:3);
[y,t,x]=lsim(sys_cl,deltadot,tspan,k0_lqr);

lean_angle_lqr=x(:,1);
steer_angle_lqr=x(:,2);
lean_velocity_lqr=x(:,3);

delta_dot_lqr=-K_lqr(1)*lean_angle_lqr-K_lqr(2)*steer_angle_lqr-
K_lqr(3)*lean_velocity_lqr;

figure(1)
plot(t,lean_angle);hold on;

plot(tspan,lean_angle_lqr,'r')
title('lean angle recovery vs time','fontsize',15)
legend('Numerical result','LQR control result')
xlabel('time (s)','fontsize',15);
ylabel('lean angle (rad)','fontsize',15);
figure(2)
plot(t,deltadot_num);hold on;

plot(tspan,delta_dot_lqr,'r');hold on;
constraint_neg=-pi*ones(length(t));
constraint_pos=pi*ones(length(t));
plot(tspan,constraint_pos,'k');
plot(tspan,constraint_neg,'k');
xlabel('time','fontsize',15);
ylabel('steering velocity (rad/s)','fontsize',15);
legend('Numerical result','LQR control result','max steering velocity
constraint');
title('steering velocity vs time','fontsize',15)



close all; clear all;clc;
load xx
% load yy

tspan=xx(1,:);
yyy=xx(2,:);
plot(tspan,yyy,'b')
hold on;
% yyy=0.01*exp(-0.2*tspan).*sin(3*tspan)+0.01*exp(-
0.2*tspan).*cos(3*tspan)
% plot(tspan,yyy);
```

```
x0=[10 2.6 1.68 10 0 0 ];
f = @(x,tspan) x(1)*exp(-x(2)*tspan).*sin(x(3)*tspan)+x(4)*exp(-
x(2)*tspan).*cos(x(3)*tspan)...
    +x(6)*exp(x(5)*tspan);
x = nlinfit(tspan,yyy,f,x0);
%
y=x(1)*exp(-x(2)*tspan).*sin(x(3)*tspan)+x(4)*exp(-
x(2)*tspan).*cos(x(3)*tspan);
plot(tspan,y,'r')
legend('orginal data','curvefit')

REpart=-x(2)
IMpart=-x(3)

clear all;
clc;
close all;

%% Loading Const
CONST=BikeConstants; % call bike constant function

w=CONST.w;          %Wheelbase (m)
c=CONST.c;          %Trail (m)
tilt=CONST.tilt;      %tile (rad)
g=CONST.g;          %gravity (m/(s^2))
v=CONST.v;             %forward velocity (m/s)


%Rear B and frame

x_B=CONST.x_B;
z_B=CONST.z_B;        %Position of center of mass (m)

m_B=CONST.m_B;          %mass (kg)

I_Bxx=CONST.I_Bxx;      %kg m^2
I_Byy=CONST.I_Byy;       %kg m^2
I_Bzz=CONST.I_Bzz;      %kg m^2
I_Bxz=CONST.I_Bxz;      %kg m^2

I_B=CONST.I_B;

%Front Handlebar and Fork Assembly H

x_H=CONST.x_H;
z_H=CONST.z_H;        %Position of center of mass (m)

m_H=CONST.m_H;           %mass (kg)

I_Hxx=CONST.I_Hxx;      %kg m^2
I_Hyy=CONST.I_Hyy;         %kg m^2
I_Hzz=CONST.I_Hzz;      %kg m^2
```

```
I_Hxz=CONST.I_Hxz;      %kg m^2

I_H=CONST.I_H;

%Rear Wheel R

r_R=CONST.r_R;          %radius (m)
m_R=CONST.m_R;            %mass (kg)
I_Rxx=CONST.I_Rxx;     %Moment of inertia(kg m^2)
I_Ryy=CONST.I_Ryy;       %Moment of inertia(kg m^2)


%Front Wheel F

r_F=CONST.r_F;         %radius (m)
m_F=CONST.m_F;            %mass (kg)
I_Fxx=CONST.I_Fxx;     %Moment of inertia(kg m^2)
I_Fyy=CONST.I_Fyy;       %Moment of inertia(kg m^2)

%% Find M C K matrix


%Whole Bicycle

m_T = m_R + m_B + m_H + m_F;            %Total mass

x_T = (m_B*x_B + m_H*x_H + m_F*w)/m_T;               %Total center of
mass
z_T = (-r_R*m_R + z_B*m_B + z_H*m_H -r_F*m_F)/m_T;   %(wrt contact
point P)

I_Txx = I_Rxx + I_Bxx + I_Hxx + I_Fxx ...
        + m_R*(r_R^2) + m_B*(z_B^2) + m_H*(z_H^2) + m_F*(r_F^2);
I_Txz = I_Bxz + I_Hxz ...
        - m_B*x_B*z_B - m_H*x_H*z_H + m_F*w*r_F;    %Mass moments and
                                                    %products of
inertia
                                                    %(wrt contact point
P)

I_Rzz=I_Rxx;
I_Fzz=I_Fxx;    %Axisymmetrix front and rear wheels

I_Tzz=I_Rzz+I_Bzz+I_Hzz+I_Fzz...                    %Moment of inertia
    +(m_B*(x_B^2))+(m_H*(x_H^2))+m_F*(w^2);         %about z axis


%Front Assembly

m_A=m_H+m_F;    %Mass of assembly

x_A=(x_H*m_H + w*m_F)/m_A;
z_A=(z_H*m_H - r_F*m_F)/m_A;    %Location of center of mass
```

```
I_Axx = I_Hxx + I_Fxx ...                                      %Moment
of inertia
        + m_H*((z_H - z_A)^2) + m_F*((r_F + z_A)^2);           %of
front assembly
I_Axz = I_Hxz - m_H*(x_H-x_A)*(z_H-z_A) + m_F*(w-x_A)*(r_F+z_A);%About
its
I_Azz = I_Hzz + I_Fzz + m_H*((x_H-x_A)^2) + m_F*((w-x_A)^2);    %center
of mass

u_A=(x_A - w - c)*cos(tilt) - z_A*sin(tilt);    %Perpend. dist. between
front
                                                %assembly and steering
axis

I_Att = m_A*(u_A^2) + I_Axx*(sin(tilt)^2) ...               %Mom of
inertia
        + 2*I_Axz*sin(tilt)*cos(tilt) + I_Azz*(cos(tilt)^2);%abt. steer
axis
I_Atx = -m_A*u_A*z_A + I_Axx*sin(tilt) + I_Axz*cos(tilt);   %Prod. of
inertia
I_Atz = m_A*u_A*x_A + I_Axz*sin(tilt) + I_Azz*cos(tilt);    %rel to
crossed
                                                            %skew axis

mu=(c/w)*cos(tilt);     %ratio of mechanical trail to wheel base

S_R=I_Ryy/r_R;
S_F=I_Fyy/r_F;
S_T=S_R+S_F;        %Gyroscopic coefficients

S_A=m_A*u_A + mu*m_T*x_T;   %Static moment term



% M matrix (mass matrix)

M_rr = I_Txx;                               %roll roll
M_rs = I_Atx + mu*I_Txz;                    %roll steer
M_sr = M_rs;                                %steer roll
M_ss = I_Att + 2*mu*I_Atz + (mu^2)*I_Tzz;   %steer steer

M=[M_rr,M_rs;...
   M_sr,M_ss];

% K0 matrix (gravity dependent stiffness terms)

K0_rr = m_T*z_T;                            %roll roll
K0_rs = -S_A;                               %roll steer
K0_sr = K0_rs;                              %steer roll
K0_ss = -S_A*sin(tilt);                     %steer steer

K0=[K0_rr,K0_rs;...
    K0_sr,K0_ss];
```

```matlab
% K2 matrix (velocity dependent stiffness terms)

K2_rr = 0;                                    %roll roll
K2_rs = (S_T-m_T*z_T)*cos(tilt)/w;            %roll steer
K2_sr = 0;                                    %steer roll
K2_ss = (S_A+S_F*sin(tilt))*cos(tilt)/w;      %steer steer

K2=[K2_rr,K2_rs;...
    K2_sr,K2_ss];

% C1 matrix (damping terms)

C1_rr=0;
C1_rs=mu*S_T+S_F*cos(tilt)+(I_Txz*cos(tilt)/w)-mu*m_T*z_T;
C1_sr=-(mu*S_T+S_F*cos(tilt));
C1_ss=(I_Atz*cos(tilt)/w)+mu*(S_A+(I_Tzz*cos(tilt)/w));

C1=[C1_rr,C1_rs;...
    C1_sr,C1_ss];

% K and C Matrices (made of K0, K2 and C1 matrices)

K=g*K0 + (v^2)*K2;

C=v*C1;
Matrix.K=K;
Matrix.C=C;
Matrix.M=M;
torquephi=0;
torquedelta=0;
T=[torquephi torquedelta]';

Matrix.T=T;

%% natural motion whipple model
%
% k0(1)=-pi/14;  %initial lean angle phi
% k0(2)=0;  %initial steering angle delta
% k0(3)=0; %initial lean velocity phi dot
% k0(4)=0;  %initial steering velocity delta dot
% tt=0.1/10;
% tspan=[0:tt:5];
% options =odeset('abstol',1e-9, 'reltol', 1e-9);
% [t,k]=ode45(@bikeStateODE,tspan,k0,options,Matrix);
%
% xx=[tspan; k(:,1)'];
% save xx;
% figure(1);
% plot(tspan,k(:,1));
% title('lean angle for uncontroled motion','fontsize', 15);
% xlabel('time','fontsize', 15)
% ylabel('lean angle','fontsize', 15)
% hold on;
% figure(2);
```

```
% plot(tspan,k(:,2));
% title('steering angle for uncontronled motion','fontsize', 15);
% xlabel('time','fontsize', 15)
% ylabel('steering angle','fontsize', 15)
% hold on;

%% point mass model natural motion

% clear k k0
% k0(1)=-pi/4;  %initial lean angle phi
% k0(2)=0; %initial steering angle delta
% k0(3)=0; %initial lean velocity phi dot
%
% tt=0.1/10;
% tspan=[0:tt:5];
% options =odeset('abstol',1e-9, 'reltol', 1e-9);
% [t,k]=ode45(@bikeStateODE_pointmass,tspan,k0,options,Matrix);
%
% xx=[tspan; k(:,1)'];
% save xx;
% figure(1);
% plot(tspan,k(:,1));
% title('lean angle for uncontroled motion','fontsize', 15);
% xlabel('time','fontsize', 15)
% ylabel('lean angle','fontsize', 15)

% figure(2);
% plot(tspan,k(:,2));
% title('steering angle for uncontronled motion','fontsize', 15);
% xlabel('time','fontsize', 15)
% ylabel('steering angle','fontsize', 15)
%

%% compare with values given in the benchmark paper
% v=4
% RE=-0.77534188219585;
% RE=0.41325331521125;
% IM=3.07910818603206;
% IM=4.46486771378823;
% a=2;
% b=-2*a*RE;
% c=(b^2-IM^2*(2*a)^2*-1)/4/a;
% pp.a=a;
% pp.b=b;
% pp.c=c;
%
% clear k0
% k0(1)=.01;  %initial lean angle phi
% k0(2)=0;  %initial lean velocity phi dot
% options =odeset('abstol',1e-9, 'reltol', 1e-9);
% [t,k]=ode45(@odetest,tspan,k0,options,pp);
% figure (1)
% hold on;
% plot(tspan,k(:,1),'k')
%
% legend('my model','professor result')
```

```
%
%
%  tfestimate(tspan,k(:,1))
% syms aa bb cc
% eqn='aa*D2y+bb*Dy+cc*y=0';
% cond='y(0)==0.01,Dy(0)==0';
% vpa(simplify(subs(dsolve(eqn,cond),[aa bb cc],[a b c])),4)

%% forced motion whipple model
% CtrlM=[-19 15.94  -12.8767]; % control matrix guess
%
% k0(1)=0.2;  %initial lean angle phi
% k0(2)=0.1;  %initial steering angle delta
% k0(3)=0; %initial lean velocity phi dot
% k0(4)=0;  %initial steering velocity delta dot
%
% tt=0.1/10;
% tspan=[0:tt:5];
% p.options =odeset('abstol',1e-9, 'reltol', 1e-9);
% p.Matrix=Matrix;
% p.ini=k0;
% p.tspan=tspan;
% p.CtrlM=CtrlM;
% [t,k]=ode45(@bikeStateODE_forced,tspan,k0,p.options,p);
%
% xx=[tspan; k(:,1)'];
% save xx;
% figure(1);
% plot(tspan,k(:,1));
% title('lean angle');
% hold on;
%
% figure(2);
% plot(tspan,k(:,2));
% title('steering angle');
% hold on;
%
% figure(3)
% plot(tspan,k(:,4));
% title('steering velocity');
% hold on;

%% forced motion point mass
CtrlM=[-15.1456 5.0135 -2.9987]'; % initial estimate of control vector
save Matrix
clear k k0
k0(1)=0.2;  %initial lean angle phi
k0(2)=0.5; %initial steering angle delta
k0(3)=0; %initial lean velocity phi dot
k0(4)=0; % initial yaw
k0(5)=0;  % initial x
k0(6)=0;  % initial y

p.options =odeset('abstol',1e-5, 'reltol', 1e-5);
tt=0.1/10;
tspan=[0:tt:3];
```

```
p.tstep=tt;
p.Matrix=Matrix;
p.ini=k0;
p.v=v;
p.trail=0;
p.wheelbase=w;
p.tilt=0;
p.tspan=tspan;
p.CtrlM=CtrlM;

[t,k]=ode45(@bikeStateODE_pointmass_forced,tspan,k0,p.options,p);
phi=k(:,1);
delta=k(:,2);
phidot=k(:,3);
yaw=k(:,4);
x=k(:,5);
y=k(:,6);

% deltadot=-CtrlM(1)*phi-CtrlM(2)*delta-CtrlM(3)*phidot;
% plot(t,deltadot)
%% plot bike motion
% xx=[tspan; k(:,1)'];
% save xx;
% figure(1);
% plot(tspan,k(:,1),'r');
% hold on;
% % legend('whipple model','point mass model')
% title('lean angle for controlled motion','fontsize', 15);
% xlabel('time','fontsize', 15)
% ylabel('lean angle','fontsize', 15)
%
% figure(2);
% plot(tspan,k(:,2),'r');
% % legend('whipple model','point mass model')
% title('steering angle for controlled motion','fontsize', 15);
% xlabel('time','fontsize', 15)
% ylabel('steering angle','fontsize', 15)
% hold on;
% figure(3);
% plot(tspan,k(:,3),'r');
% title('lean velocity for controlled motion','fontsize', 15);
% hold on;
%
%
% deltadot=-CtrlM(1)*phi-CtrlM(2)*delta-CtrlM(3)*phidot;
% figure(4);
% plot(tspan,deltadot,'r');
% title('steer velocity for controlled motion','fontsize', 15);
%
% figure(5)
% plot(tspan,x,'r');
% title('x displacement vs time','fontsize', 15);
% xlabel('time','fontsize', 15)
% ylabel('x','fontsize', 15)
%
% figure(6)
```

```
% plot(tspan,y,'r');
% title('y displacement vs time','fontsize', 15);
% xlabel('time','fontsize', 15)
% ylabel('y','fontsize', 15)
%
% figure(7)
% plot(tspan,yaw,'r');
% title('yaw vs time','fontsize', 15);
% xlabel('time','fontsize', 15)
% ylabel('yaw angle','fontsize', 15)
% %
% close all;



%% optimize controller
% clear k;
% problem.objective = @(CtrlM)BikeOpt(CtrlM,p);
% problem.x0 = CtrlM;
% problem.lb = CtrlM-5.1;
% % problem.lb = -Inf;
% % problem.ub = Inf;
% problem.ub = CtrlM+5.1;
% problem.solver = 'fmincon';
% problem.options=optimset('Algorithm','active-set');
%
% % problem.options.Display = 'notify-detailed';
%
% [sol,costval] = fmincon(problem)
%
% p.CtrlM=sol;
% [t,k]=ode45(@ bikeStateODE_pointmass_forced,tspan,k0,p.options,p);
% figure(1);
% plot(tspan,k(:,1),'k');
% % title('lean angle');
% figure(2);
% plot(tspan,k(:,2),'k');
% % title('steering angle');
% figure(3)
% plot(tspan,k(:,3),'k');
% % title('lean velocity');
% deltadot=-p.CtrlM(1)*k(:,1)-p.CtrlM(2)*k(:,2)-p.CtrlM(3)*k(:,3);
% figure(4)
% plot(tspan,deltadot)
% title('steering velocity');



%% Add Motor Dynamics

% Minv=M\eye(size(M));
%
% MinvK=M\K
%
% MinvC=M\C
```

```
%
% A=[[0,0,1,0];...
%    [0,0,0,0];...
%     -MinvK,-MinvC(:,1),zeros([2,1])];
%
% B=[0;1;-MinvC(:,2)];


%   -15.7080 optimal controller
%     4.7618
%    -1.9865
```