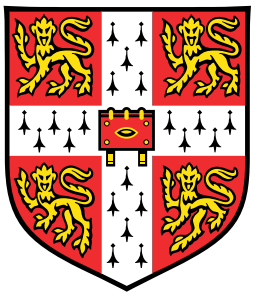
Self-Balancing Bicycle



Philip M. Salmony
Department of Engineering
University of Cambridge

Supervisor: Dr. Fulvio Forni

I hereby declare that, except where specifically indicated, the work submitted herein is my own original work.

Signed **Date**

Self-Balancing Bicycle

Philip Salmony, Wolfson College

Abstract

Contents

1 Introduction

This project involves the modelling, simulation, control system design, and implementation of a rider-less, self-balancing bicycle. Bicycles and their stability properties have been studied for well over a century, most notably by academics such as Whipple (1899, [1]), Papadopoulos (1980s onwards, [2]), and Astrom (2005, [3]). Most individuals know that a bicycle can be self-stable under certain conditions but it is still commonly unclear what primarily contributes to this self-stability. The full form of the equations of motion do not aid the intuitive understanding of bicycle stability and dynamics, as they are a set of coupled and highly non-linear differential equations. Thus, simplifying assumptions must be sought out.

In addition to simply understanding the causes of self-stability of a bicycle, there are further considerations to studying a bicycle in the frameworks of dynamics and control theory. For example, how does a human stabilise a bicycle and is it possible to replicate this behaviour with motors? What makes a bike more or less stable (for instance, changes in geometry, mass distribution, or wheel size - to name a few)? And would there be a benefit in having an *assisted* bicycle for certain people in certain situations?

This project therefore aims to try and answer - at least in part - some of these questions, with a primary focus on designing suitable control systems that achieve closed-loop stability over a large range of forward velocities.

Finally, the report details the implementation of the designed control laws for two rider-less bicycles. Firstly, a *Lego Mindstorms* prototype and secondly a full-scale, adult bicycle. Both bicycles are depicted in Figures 1 and 2 below. These bicycles however only have a drive actuator, to provide forward velocity, and an actuator placed at the handlebars to provide a *steer* torque. Without the additional component of being able to provide a *lean* torque, it is extremely difficult to guarantee stability at near-zero forward speed, as will be shown in this report. The bicycles being studied in this case can therefore be classed as *underactuated systems*.

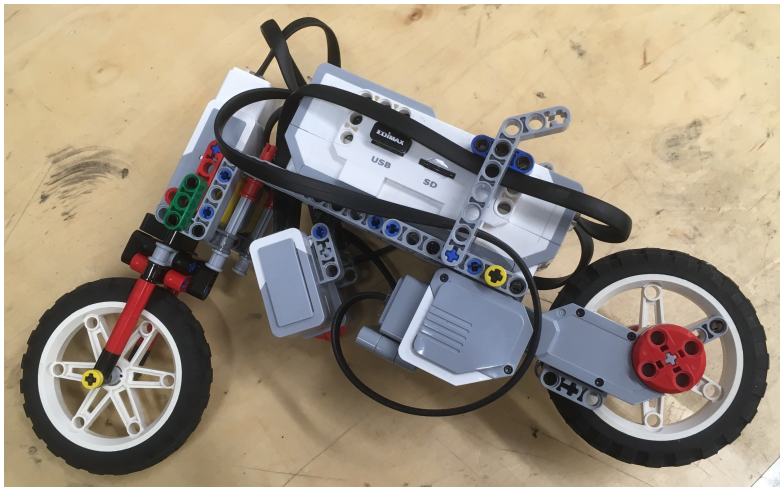


Figure 1: Lego Prototype Bicycle

Figure 2: Full-Scale Bicycle

2 Modelling

To be able to design suitable control laws in order to stabilise the bicycle, it is necessary to provide a mathematical description of the system. Additionally, developing a dynamical model of the system will give an insight into what factors influence a bicycle's behaviour in terms of its stability.

As previously mentioned, the equations of motion for a bicycle are a set of many, highly non-linear, and coupled differential equations. In this form, they do not aid an intuitive understanding of what makes a bicycle more or less stable and are inconvenient to use for control law design, which is based mainly on linear methods. For simulation however, these

non-linear equations of motion are important, as they capture the dynamics over the entire range of operating regimes.

This section aims to explore two commonly used linearised bicycle models of second and fourth order, which will be the basis for controller design in later sections.

2.1 Geometry and Definitions

The definitions of bicycle parameters, geometry, and coordinate system in this report follows the system given by Åström in [3], which is based on the *ISO 8855* standard [4]. A brief explanation of the bicycle parameters, as well as the top, rear, and side views of the bicycle are detailed in Table 1 and Figures 3 and 4.

Parameter	Symbol	Description
Center of Mass (z)	h	Distance of center of mass from ground.
Center of Mass (x)	a	Distance of center of mass from center of rear wheel.
Wheel Base	b	Distance between centers of both wheels.
Trail	c	Distance between intersection of steering axis with ground and point of contact with ground of front wheel.
Head Angle	λ	Angle between steering axis and ground.

Table 1: Bicycle Parameter Definitions

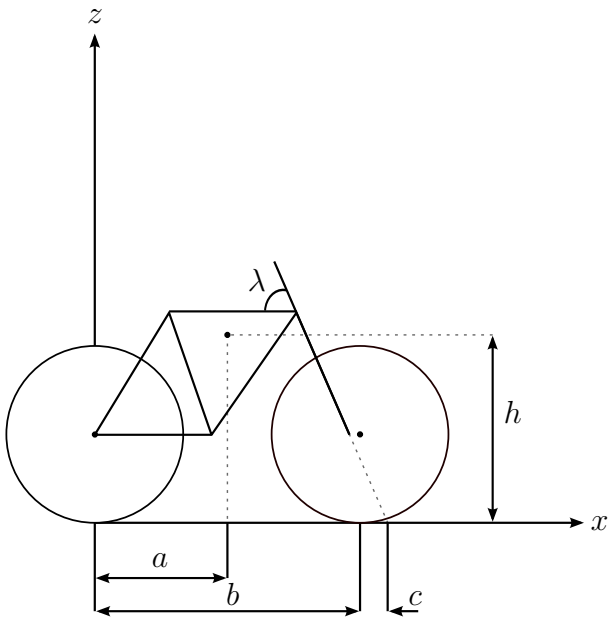
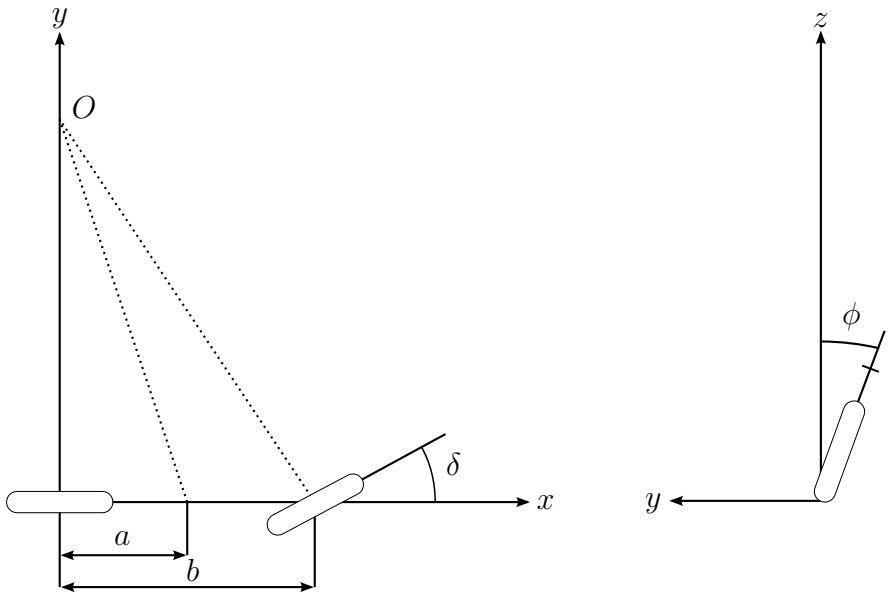


Figure 3: Side View



(a) Top View (b) Rear View

Figure 4: Geometric Definitions

2.2 Parameter Estimation

The parameters of the real-world bicycles needed to be estimated to be used in the models explored in this section.

The required lengths were obtained by simple measurement. However, parameters such as centers of mass, as well as moments of inertia were more difficult to estimate. To aid the estimation, a Python script was written that takes as an input the geometry and mass distribution of the bicycle, slices this geometry into a finite three-dimensional grid, and then computes the center of mass and moment of inertia terms via discrete approximations of the relevant integrals.

The estimated parameters for both bicycles can be found in Table 2 below.

Parameter	Symbol	Lego	Full-Scale
Mass	m	0.76kg	28kg
Center of Mass (x)	a	0.10m	0.26m
Center of Mass (z)	h	0.11m	0.55m
Wheel Base	b	0.23m	1.1m
Moment of Inertia (xz)	I_{xz}	$0.0084\text{kg}\,\text{m}^2$	$8.3\text{kg}\,\text{m}^2$
Moment of Inertia (x)	I_x	$0.0092\text{kg}\,\text{m}^2$	$3.9\text{kg}\,\text{m}^2$

Table 2: Lego Prototype and Full-Scale Bicycle Estimated Parameters

2.3 Second Order Model

The simplest model of a bicycle leads to a second order differential equation relating the steer angle δ to the lean angle ϕ. It does not take the dynamics of the front fork into account and assumes zero trail c, a 90° head angle λ, and a constant forward speed V.

Even though the model is limited due to its simplifying assumptions, it provides useful and intuitive insights into aspects of bicycle stability. In addition, it forms the basis for controller design in later sections.

2.3.1 Derivation

The derivation given in this section follows the method used by Åström in [3]. The first step in deriving the second order model is to consider a bicycle that is rotating about a fixed center O with angular velocity ω_z. Using the method of instantaneous centers, this can be calculated to be

$$\omega_z = \frac{V}{b}\tan(\delta).$$

Then, computing the angular momentum of the system:

$$\underline{L} = \mathbf{I}\cdot\underline{\omega} = \begin{bmatrix} I_x & 0 & I_{xz} \\ 0 & I_y & 0 \\ I_{xz} & 0 & I_z \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ 0 \\ \omega_z \end{bmatrix}$$

Extracting only the component about the x-axis and letting $D = -I_{xz}$ and $J = I_x$ gives

$$L_x = J\dot{\phi} - D\frac{V}{b}\tan(\delta).$$

Taking derivatives, whilst noting that a constant forward speed is assumed, leads to

$$\dot{L}_x = J\ddot{\phi} - \frac{D}{b}V\dot{\delta}\sec^2(\delta).$$

Now the torques acting on the system are required, which can be summed and used in conjunction with Newton's second law for moments,

$$\dot{L}_x = \sum_{i=1}^{N} \tau_i.$$

In this model, it is assumed that *two* torques are acting on the system. The first being due to gravity and a non-zero lean angle,

$$\tau_g = mgh\sin(\phi).$$

The second torque is due to circular motion and can be written as follows:

$$\begin{aligned} F_c &= m\omega_z^2 r = \frac{mV^2}{b}\tan(\delta) \\ \tau_c &= F_c h\cos(\phi) \\ &= \frac{mV^2 h}{b}\tan(\delta)\cos(\phi) \end{aligned}$$

Summing both torques results in the final moment balance:

$$J\ddot{\phi} - \frac{VD}{b}\dot{\delta}\sec^2(\delta) = mgh\sin(\phi) + \frac{mV^2h}{b}\tan(\delta)\cos(\phi) \tag{1}$$

This is the final form of the governing second-order, non-linear differential equation of motion and will be used for simulation. To make this model useful for controller design, it needs to be linearised about zero lean and zero steer angle by using small-angle approximations. Collecting terms, the linearised equation of motion then becomes,

$$J\ddot{\phi} - mgh\phi = \frac{VD}{b}\dot{\delta} + \frac{mV^2h}{b}\delta.$$

Now taking Laplace transforms, the transfer function for the second order bicycle model becomes:

$$G(s) = \frac{\bar{\phi}(s)}{\bar{\delta}(s)} = \frac{VD}{Jb}\frac{s + mhV/D}{s^2 - mgh/J}$$

Lastly, the moment of inertia terms can be approximated as $D \approx mah$ and $J \approx mh^2$, which

leads to the final version of the linearised, second order model,

$$G(s) = \frac{Va}{hb}\frac{s + V/a}{s^2 - g/h}. \tag{2}$$

This transfer function, relating the steering angle δ (in degrees) to the lean angle ϕ (in degrees), will be used as the basis for controller design and is *strongly* dependent on forward speed V. It is composed of a variable gain, variable left-half plane zero, and a fixed pair of symmetric real poles with one in the left-half plane and one in the right-half plane. Notice that imposing a steer angle in one direction, will cause the bicycle to lean in the *opposite* direction. The stability properties of this model can now be investigated.

2.3.2 Stability

Even though the model is substantially simplified, it provides a good insight into the stability properties of the bicycle. Four key aspects are as follows:

1. **Always unstable**: The poles are fixed and do not vary with forward speed. They are always symmetric about the imaginary axis, with one stable and one unstable pole.

2. **Speed-dependent gain**: The effectiveness of applying a steer angle input decreases as speed decreases. Therefore, controllability of the system decreases as well. Far larger inputs are needed at lower speeds, and smaller inputs are required at higher speeds.

3. **Speed-dependent zero**: The zero present in the system is moved further into the left-half plane as the forward speed increases. This in effect pulls the root-locus further into the left-half plane as well, making the system easier to stabilise.

4. **Center of mass**: The poles are located at $\pm\sqrt{g/h}$. Since g is fixed, the location of the center of mass above ground h is the only variable that changes the locations of the poles. Thus, the instability of the bicycle decreases as h increases, with the limiting case of marginal stability when $h \to \infty$. This is analogous to the stabilisation of a vertical beam: a longer beam is easier to stabilise.

Furthermore, a simple proportional controller of the form $\delta = -k_\delta \phi$ can stabilise the bicycle in theory if and only if the gain k_δ satisfies the following condition:

$$k_\delta > \frac{bg}{V^2} \tag{3}$$

Which shows that an ever-increasing gain is needed as the forward speed decreases. This, of course, is practically infeasible and shows that it is impossible to balance a stationary bicycle

via just controlling the handlebars. Interestingly, a longer wheel base (b) in turn requires the feedback gain to be larger as well.

2.3.3 Non-Minimum Phase Behaviour

As a brief aside, the model given above does not take fork parameters into account, such as trail c or head angle λ. Including these in the model leads to a zero that can move into the right-half plane, where the location is given by:

$$s = -\frac{V^2 h - acg}{Vah}$$

For a fixed geometry, the zero will enter the right-half plane depending on the forward speed ($V < \sqrt{acg/h}$). Thus, the system is not only unstable but also exhibits non-minimum phase behaviour, making the system extremely difficult to control.

However, since for the Lego prototype the trail c was found to be sufficiently small, it can be effectively ignored, as the model then approximates the simpler minimum phase, second order model derived previously. Furthermore, for the full-scale bicycle the speed is sufficiently large that the zero does not move into the right-half plane. Thus, the simpler second order model can be used for controller design for both bicycles as both do not exhibit non-minimum phase behaviour.

2.3.4 Transfer Functions

Using the estimated bicycle parameters for the Lego prototype and full-scale bicycle in conjunction with the second order model, the transfer functions in terms of forward speed can be given for both the Lego prototype and full-scale bicycle.

Lego Prototype

$$G_{Lego}(s) = 3.4 \cdot V \cdot \frac{s + 10 \cdot V}{s^2 - 74} \tag{4}$$

Full-Scale Bicycle

$$G_{FS}(s) = 0.44 \cdot V \cdot \frac{s + 3.8 \cdot V}{s^2 - 18} \tag{5}$$

2.4 Fourth Order Model

A natural evolution from the second order model is the fourth order model, which takes the dynamics of the front fork into account. This model can explain why the bicycle can keep itself self-stable under certain conditions. However, the control input is now the *torque* acting

on the handlebar as opposed to the steer angle. Thus, this model is not suited for control design for two reasons:

1. Driving the handlebar motor in torque is not feasible, since both motor current and voltage measurement, as well as knowledge of the internal motor parameters, is needed to estimate the torque.

2. If not most importantly, using a servo at the handlebars essentially eliminates the front fork dynamics by enforcing a steer angle.

Therefore, the second order model will be used for controller design. Nonetheless, this fourth order model is incredibly useful in describing the self-stable dynamics of a rider-less bicycle and will be briefly investigated here. A full derivation is given by Papadopoulos in [5]. The model takes the following standard form:

$$\mathbf{M} \cdot \ddot{\underline{q}} + \mathbf{C}_1 \cdot v \cdot \dot{\underline{q}} + (\mathbf{K}_0 + \mathbf{K}_2 \cdot v^2) \cdot \underline{q} = \begin{bmatrix} 0 \\ \tau_\delta \end{bmatrix}$$

Where $\mathbf{M}$, $\mathbf{C}$, and $\mathbf{K}$ are mass, damping, and stiffness matrices respectively. The vector $\underline{q}$ is composed of $\begin{bmatrix} \phi & \delta \end{bmatrix}^T$.

Alternatively, this can be converted to a four-state, state-space representation. For the unmodified, full-scale bicycle, this results in the following system matrices:

$$\dot{\underline{x}} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 17 & -0.76 - 1.9v^2 & -0.51v & -1.1v \\ 1.4 & 10 - 0.60v^2 & 5.1v & -1.3v \end{bmatrix} \underline{x} + \begin{bmatrix} 0 \\ 0 \\ -0.57 \\ 5.8 \end{bmatrix} \tau_\delta$$

Where the state vector is given by $\underline{x} = [\phi, \dot{\phi}, \delta, \dot{\delta}]^T$.

2.4.1 Stability

In comparison to the second order model given previously, the fourth order model does not give us a direct insight into factors that make the bicycle more or less stable. However, it does show the fact that when the front fork dynamics are taken into account, the bike will be self-stable across a certain range of forward speeds. In addition, it describes various modes apparent in the bicycle dynamics. This is illustrated in Figure 5 below, which shows the real parts of the eigenvalues of the system dynamics across a range of forward speeds for the full-scale bicycle.

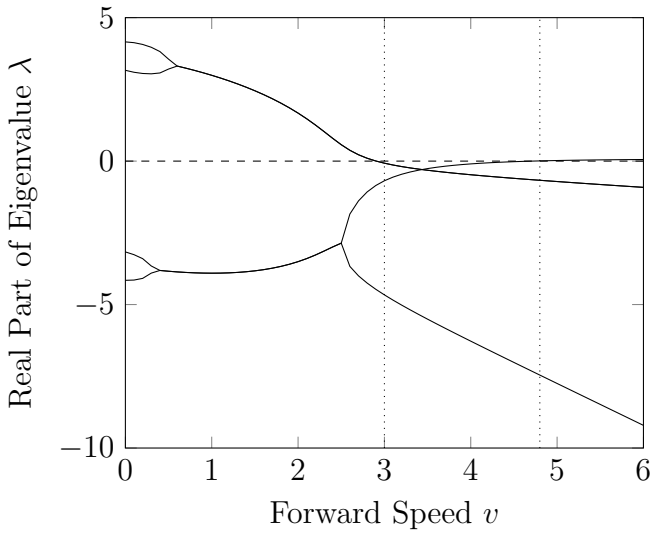


Figure 5: Pole Locations vs Forward Speed

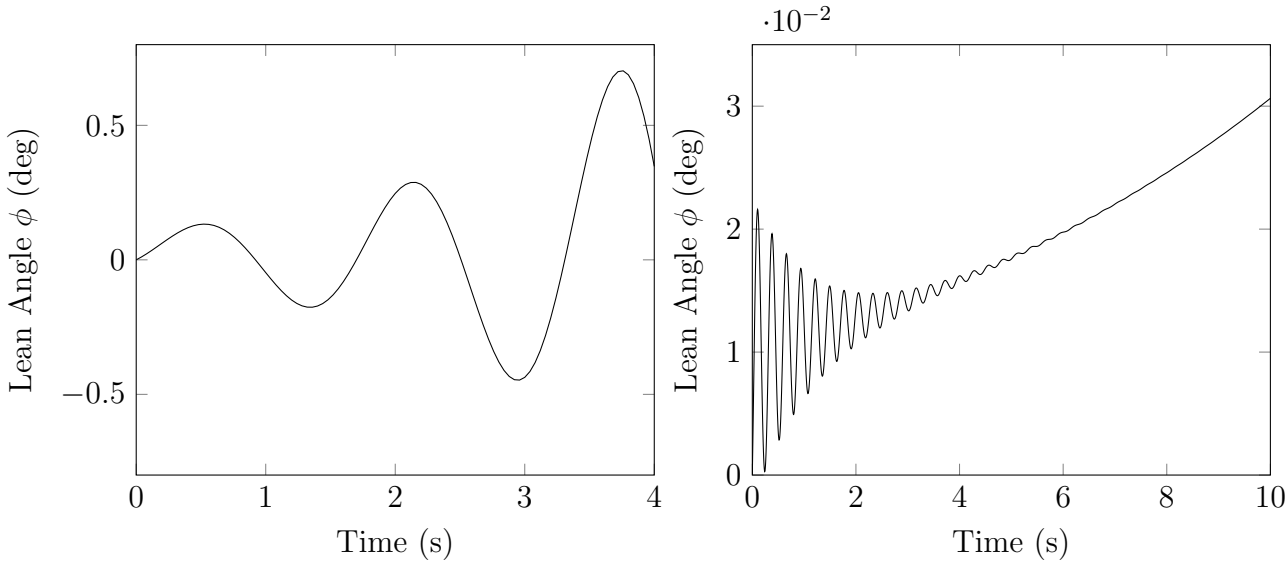
The plot can be divided into three sections:

1. **Unstable and weaving**: Up to approximately $3ms^{-1}$, the bike is unstable, as two of the four poles are in the right-half plane. The poles start off as four distinct, real poles and form into two complex-conjugate pairs. These complex poles give rise to an oscillatory, weaving motion, where the frequency of oscillation is dictated by the magnitude of the imaginary components of the poles.

2. **Self-stable**: From $3ms^{-1}$ to roughly $4.8ms^{-1}$, the bike is self-stable with all poles in the left-half plane. This stability is due to the fact that as the bicycle leans over the fork is now able to steer into the direction of the fall due to its geometry, thus *catching* the bike from from falling down further and restoring it to an upright position.

3. **Capsize**: Contrary to popular belief, increasing the speed does not improve stability indefinitely. After $4.8ms^{-1}$, a real pole again moves into the right-half plane. However, this pole is very slow and thus the system is easier to stabilise via feedback compared to slower speeds. This mode is commonly termed the *capsize* mode.

The weave and capsize modes can be excited by a lean torque disturbance when the bicycle is travelling at the relevant forward speed. This is shown in Figure 6 below for the full-scale bicycle model.



(a) Weaving ($v = 2.8\mathrm{m\,s^{-1}}$)

(b) Capsizing ($v = 10\mathrm{m\,s^{-1}}$)

Figure 6: Modes of Full-Scale Bicycle

The oscillations of the unstable weave mode dominate the response in a). On the other hand in b), the high frequency oscillations due to the stable complex-conjugate poles die out quickly with the very slow, non-oscillatory capsize mode dominating the response and finally causing the bicycle to fall over. Notice that even after ten seconds, the lean angle has only deviated by a couple hundredths of a degree. As stated before, this mode is therefore very easily stabilised.

2.5 Actuators

The modelling of control actuators is also of vital importance and needs to be included in the overall system representation. This is due to the fact that actuators cannot change their position instantaneously and therefore will introduce lag into the system, which may cause further instability.

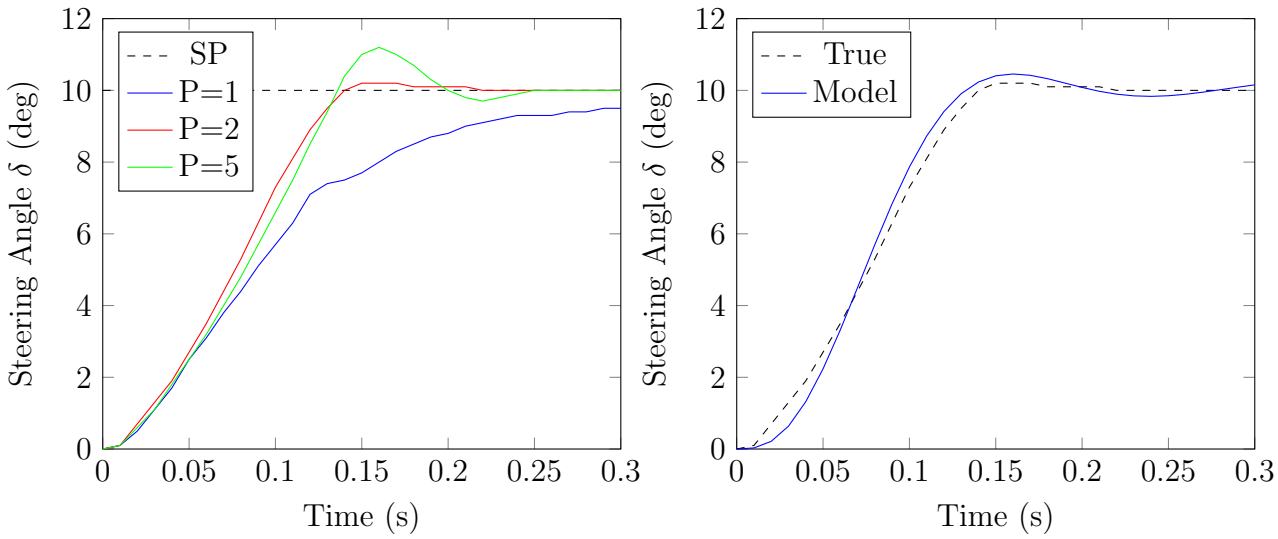
The actuator models were identified by applying a step input to the handlebar servos and recording the corresponding output response. It is important to mention that the step responses were under *loaded* conditions, meaning that the servos were mounted on the front forks, supporting the full weight of the bicycle.

An initial guess was made at the form of the transfer function, such as the model order and form. Then, the Matlab *System Identification Toolbox* was used to identify the locations of

the poles and zeros, as well as the magnitude of any possible delays.

2.5.1 Lego Prototype

Interfacing with the Lego handlebar motor can only be achieved by issuing a speed command, not an absolute position command. This meant that it was necessary to implement an additional controller to set the motor position. Luckily, an encoder was already fitted to the servo, meaning that positional measurement for feedback was available. After experimentation, it was decided to use a proportional-only controller, since it satisfied the requirements and was easy to implement. The step responses for a set of proportional gains can be seen in Figure 7 a) below. A proportional gain of 2 was chosen to give a critically damped response, with a fast rise time, and minimal steady-state error.



(a) True Servo Responses

(b) Model Verification (P=2)

Figure 7: Lego Prototype Servo Responses

The closed-loop, proportional-control motor system was given a step input of magnitude 100 degrees. The *System Identification Toolbox* was then used to fit transfer function parameters to the acquired step response data. By iteration, a third order model, consisting of a real pole and an under-damped pair of poles, of the following form was chosen:

$$G_a(s) = \frac{1}{(T_1 s + 1) \cdot (T_w^2 s^2 + 2\zeta T_w s + 1)} \tag{6}$$

Where $T_1 = 0.056$, $T_w = 0.030$, and $\zeta = 0.37$. The identification algorithm suggested a near 95% match between true data and simulated data from this third order model. A comparison

between the true and model step responses, visually confirming a good agreement, can be seen in Figure 7 b).

This servo model is then cascaded with the bicycle model to give an overall more accurate representation of the real-world system.

2.5.2 Full-Scale Bicycle

The same process can now be repeated for the full-scale bicycle. However, direct positional control of the motor was already implemented by the manufacturer and thus the step response data could be directly acquired.
Again, a third order model was fitted to the step response data, giving the following parameters: $T_1 = 0.07$, $T_w = 0.073$, and $\zeta = 0.48$. The comparison between the step response of the true system and the model can be seen in Figure 8 below, again indicating a near-ideal fit.

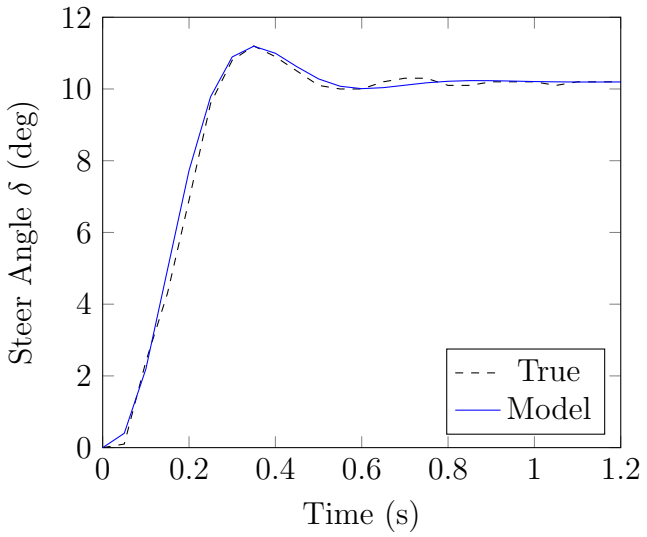


Figure 8: Full-Scale Bicycle Servo Step Response

2.6 Model Verification

Since the bicycle is an unstable system, accurate model verification without a stabilising controller is near impossible. Even with a stabilising controller in place, it is challenging to attain any kind of step or frequency response data. Therefore, model verification will be undertaken implicitly by analysing if the theoretically obtained control laws do indeed give stability when implemented on the real-world system.

3 Controller Design and Simulation

This section details the controller design process for the bicycle, where three controller types were investigated: *PID*, *LQR*, and H_∞ *loop-shaping*. Since the steering angle is the only control variable and the lean angle is the only measured variable, the controller design problem is a *single-input, single-output* (SISO) type.

For simulation, the second order non-linear model derived in the previous section will be used to test the suitability of the final controller designs. This is to determine if the controller will still manage to stabilise the bicycle, even if it is perturbed outside of the linear range the controller is designed for. Thus, all following plots are using the non-linear model.

The simulation and development environment was chosen to be Matlab and Simulink. Additionally, a bicycle simulator was developed from first principles using the game engine *Unity* and C#. This gives a full 3D representation of the bicycle dynamics and allows real-time simulation, where the user can impose disturbances, and change various parameters on the fly, such as controller gains, sensor noise, actuator limits, and so forth. The non-linear differential equation of the second order model was integrated using the fourth order *Runge-Kutta* method. A screenshot of the custom simulator is given in Figure 9 below.

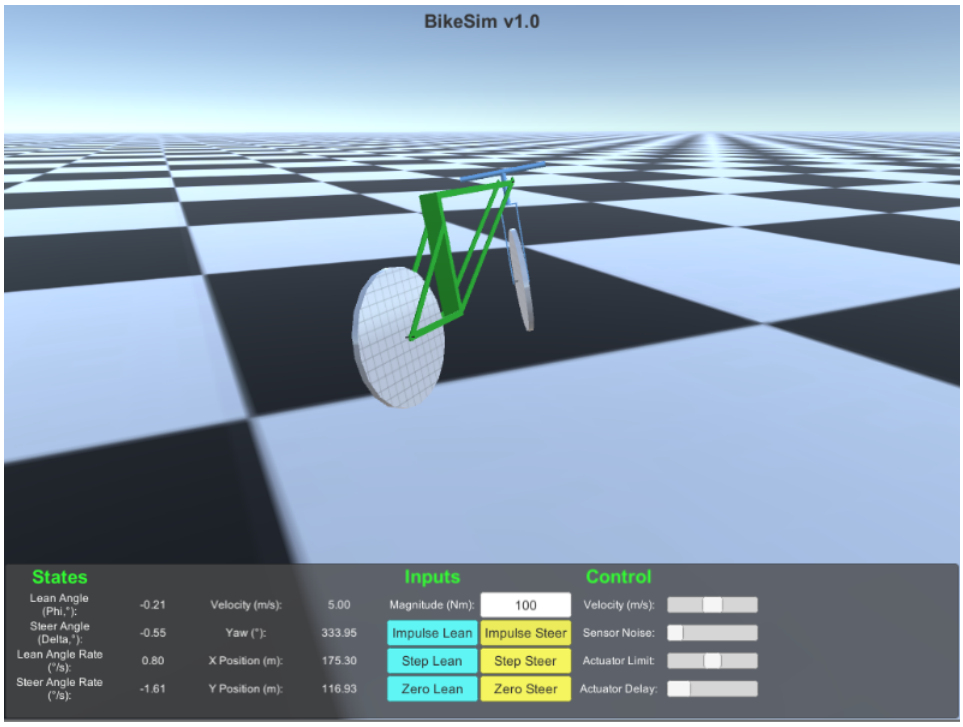


Figure 9: Custom Bicycle Simulator

For the sake of brevity, this section will only give detailed explanations of the controller design process for *one* of the two bicycles, namely the Lego prototype. The controller designs for the full-scale bicycle follow essentially the same logic, given that the plant and actuator models are of the same form, and thus the design steps need only be described once. However, the final controller designs and their performance in simulation will be given for both bicycles.

3.1 Performance Requirements

When designing a control system, it is important to be aware of any requirements and specifications that need to be met as to tailor the control law to those needs. In the case of the rider-less bicycle, the main goal is to ensure stability across a relatively broad range of forward speeds.

In addition, should stability be achieved, it is desirable to have a controller that is primarily focused on reducing the effects of disturbances acting on the bicycle, as opposed to tracking the reference since this will be predominantly set to zero. Disturbances could be as small as an uneven road or path that the bicycle is riding over, and as large as a strong gust of wind giving the bicycle a sideways push. As a further consideration, the amount of overshoot should be limited, so as not to excite any non-linearities by deviating too far from the linear operating point the controller is designed for.

For all controllers, the output is limited to ± 60 degrees to prevent extreme steering angles.

Furthermore, the cross-over frequency needs to be kept above the frequency of the right-half plane pole, as there the phase lead provided by the pole is needed to ensure an encirclement of the -1 point on the Nyquist diagram.

Overall however, the bicycle, as an under-actuated system with only a handlebar actuator, is an incredibly difficult system to control. This therefore limits the control performance one can expect and reduces the number of performance constraints that can be placed on it.

3.2 Operating Point

As the bicycle models are *strongly* dependent on forward speed, it needs be to ensured that that the controllers can cope well with changes in velocity. However, to simplify controller design and to reduce the amount of symbolic expressions, an operating point is selected (i.e. a specific forward speed) and controllers designed for that specific operating point. This

process can be repeated over a large range of forward speeds and then during the final implementation, the controllers can be gain-scheduled using gain look-up tables.

For the Lego prototype, the maximum achievable speed was found to be $V = 0.44ms^{-1}$, which is thus chosen as the operating point. An increase in speed would be desirable to enable better control performance, however this was limited by the motors available.

For the full-scale bicycle, the operating point was empirically selected to be $V = 3ms^{-1}$. This was chosen to be deliberately low for safety reasons, as not to damage the bicycle or any surrounding objects, should the bicycle become uncontrollable. Additionally, a control system that works for low speeds is harder to achieve, and therefore any increase in speed will make the control design easier.

3.3 PID

A first controller design using a Proportional-Integral-Derivative (PID) controller was investigated, for which the block diagram, including actuator $G_a(s)$ and plant dynamics $G(s)$, can be seen in Figure 10 below. PID controllers are used extensively in control engineering for their relative simplicity and overall good performance in a multitude of scenarios.

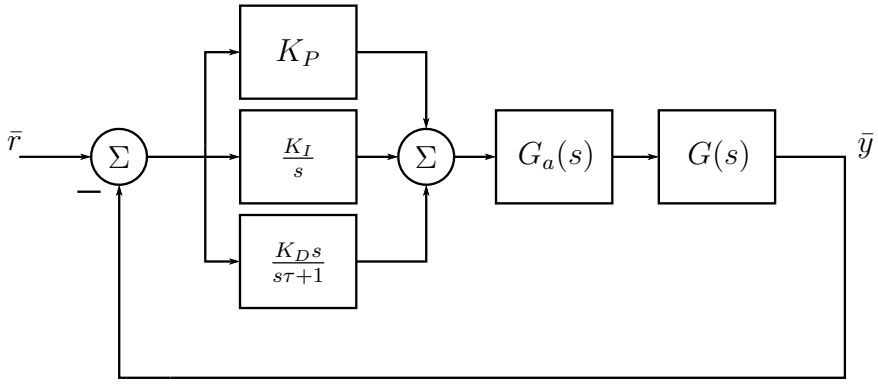


Figure 10: PID Controller Block Diagram

From the block diagram it can be seen that a band-limited differentiator is used, since a pure differentiator is not realisable in practice as the gain increases indefinitely as the frequency increases. Additionally, a pure differentiator amplifies high-frequency sensor noise in the system, which in turn greatly decreases control performance and actuator lifespan. Thus, a practical band-limited differentiator flattens off the gain at a cross-over frequency (in $rads^{-1}$) given by τ^{-1}. It can be seen as a pure differentiator followed by a first-order low-pass filter.

Lego Prototype

Firstly, closed-loop stability is to be achieved. From Equation 3, the bicycle can in theory be stabilised using a proportional-only controller with gain satisfying:

$$K_P > 11.4$$

However, this is without considering the effects of the actuator. Cascading the plant with the actuator gives the following root locus diagram:

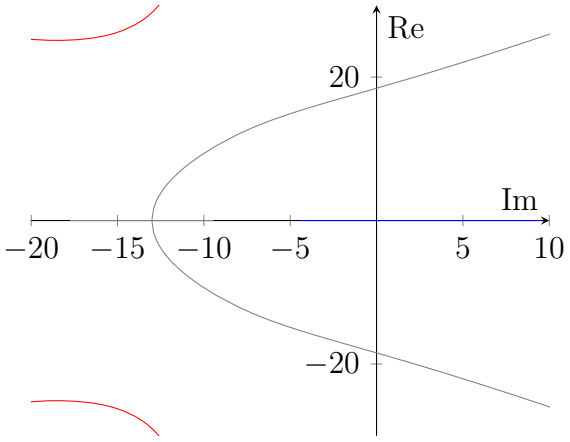


Figure 11: Root Locus for $K_P > 0$

As can be seen from the plot, there is now an additional upper bound on the gain, as the pair of complex conjugate poles move towards and into the right-half plane as K_P increases further. Thus, the closed-loop system is stable for a proportional gain in the range of:

$$11.4 < K_P < 14.3$$

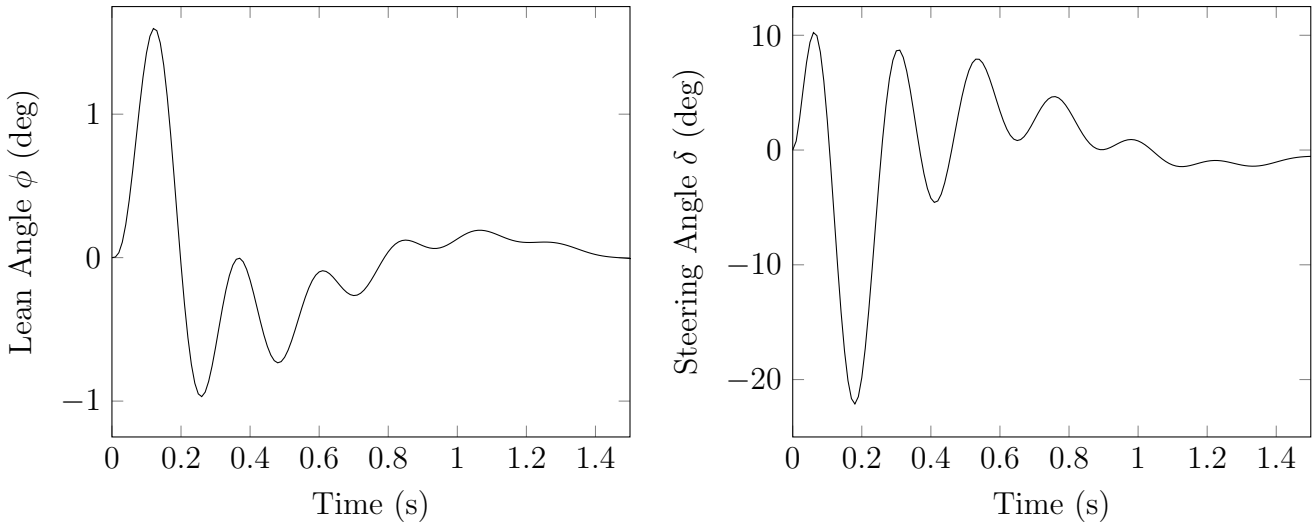
After choosing a proportional gain, both an integral gain and derivative gain can be chosen using the same root-locus approach by reforming the controller and plant into the respective Evans' form:

$$1 + K_I \frac{G_a(s)G(s)}{s(1 + (1 + K_P)G_a(s)G(s)} = 0$$
$$1 + K_D \frac{sG_a(s)G(s)}{1 + (1 + (K_P + K_I/s)G_a(s)G(s)} = 0$$

Integral gain is chosen to give zero steady-state error when the system is subjected to a constant disturbance, whereas derivative gain is then chosen to increase the damping of the system, thus reducing the oscillations.

Lastly, the time constant τ, which determines the cut-off frequency of the band-limited differentiator, is chosen by iteration so that stability and sufficient damping of the closed-loop system is maintained. However, τ must be kept large enough to suppress sensor noise from corrupting the response.

In this way the parameters were chosen to be: $K_P = 12$, $K_I = 38$, $K_D = 0.44$, and $\tau = 0.00062$. The response to an impulse disturbance of magnitude 10 at the input and the corresponding actuator effort can be seen in Figure 12 below.



(a) Lean Angle Response

(b) Actuator Effort

Figure 12: Lego PID Controller Response

As can be seen from the plots, the PID controller in simulation can recover from an impulse in approximately one second whilst keeping the controller effort relatively small. The response is rather oscillatory however due to the addition of the integrator.

3.4 Linear Quadratic Regulator

As a second controller type, optimal linear control was investigated by means of an infinite-horizon linear quadratic regulator (LQR), which aims to minimise the cost function for the state-space realisation $\dot{\underline{x}} = \mathbf{A}\underline{x} + \mathbf{B}\underline{u}, \underline{y} = \mathbf{C}\underline{x}$,

$$J = \int_0^\infty (\underline{x}^T \mathbf{Q} \underline{x} + \underline{u}^T \mathbf{R} \underline{u}) dt.$$

Where $\mathbf{Q}$ and $\mathbf{R}$ are positive semi-definite matrices, which respectively penalise having non-zero states and non-zero control inputs. The minimum cost is achieved by solving the *Control*

Algebraic Riccatti Equation (CARE):

$$\underline{0} = \mathbf{C}^T\mathbf{C} + \mathbf{X}\mathbf{A} + \mathbf{A}^T\mathbf{X} - \mathbf{X}\mathbf{B}\mathbf{B}^T\mathbf{X}$$

To give the stabilising feedback law,

$$\underline{u} = -\mathbf{B}^T\mathbf{X}\underline{x}.$$

The second-order bicycle model can be transformed into state-space form using controllable canonical form and results in the following system matrices:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ \frac{mgh}{J} & 0 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$
$$\mathbf{C} = \begin{bmatrix} \frac{mhV^2}{bJ} & \frac{VD}{bJ} \end{bmatrix}$$

As before, the control input u is the steer angle δ and the output y is the lean angle ϕ. For simplicity, the actuator model was *not* taken into account, as to keep the state dimension low.

Unfortunately, the states are not directly measurable. However, since the rank of the observability matrix $\mathcal{O}(A, C)$ is equal to the state dimension and the system is thus observable, an observer can be used to estimate the internal states of the system, given measured inputs and outputs. For simplicity, it was decided to use the *Luenberger* observer given by,

$$\dot{\underline{\hat{x}}} = \mathbf{A}\underline{\hat{x}} + \mathbf{B}\underline{u} + \mathbf{L}(\underline{y} - \underline{\hat{y}}).$$

Where $\underline{\hat{x}}$ denotes the state estimate. The rate of change of the estimation error $\dot{\underline{e}} = (\mathbf{A} - \mathbf{L}\mathbf{C})\underline{e}$ is determined by the eigenvalues of the matrix $\mathbf{A} - \mathbf{L}\mathbf{C}$. Thus, $\mathbf{L}$ must be appropriately chosen to ensure that the estimation error approaches zero quickly.

As pictured in Figure 13 below, the state estimates of the observer can then be used in conjunction with the state-feedback matrix $\mathbf{K}$ given by the LQR to form the closed-loop system.

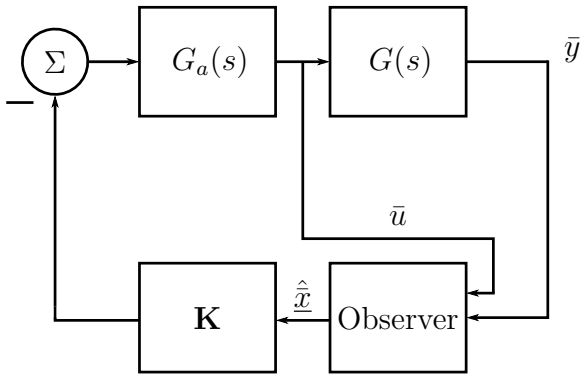


Figure 13: LQR Controller Block Diagram

The choice of **Q** and R, which for a single-input system is a scalar, in general is not trivial. However, some constraints can be placed on the magnitudes of both. For the bicycle, it is desirable to reduce actuator effort, which is achieved by increasing the magnitude of R. Rapid stabilisation in turn is brought into effect by increasing the elements of **Q**, thus heavily penalising a non-zero state.

Lego Protoype

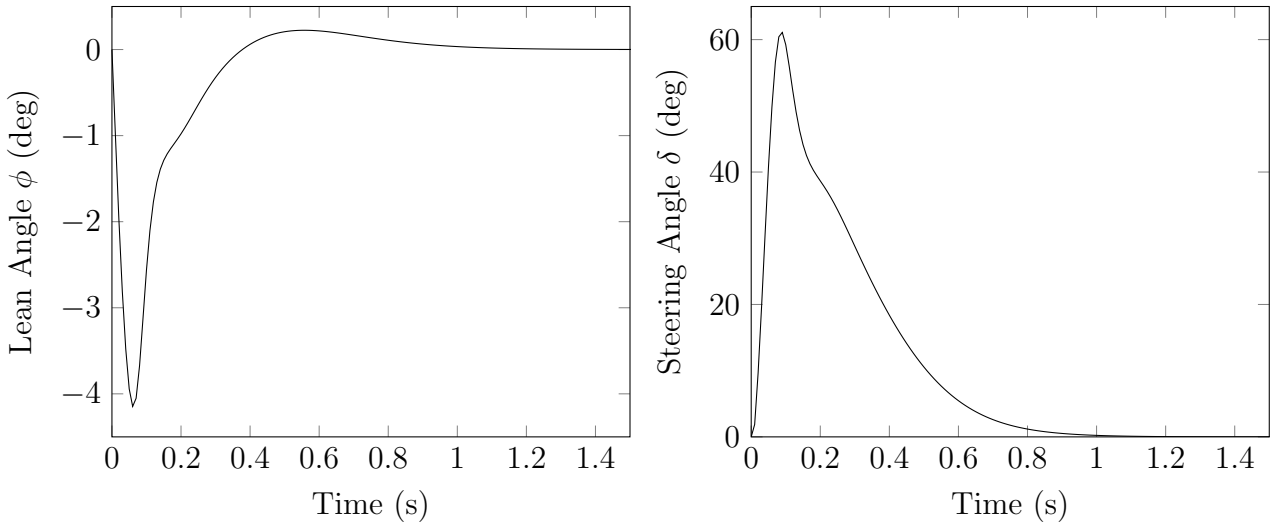
Values for **Q** and R for the Lego prototype model were chosen by iteration. The solution to the CARE and thus the resulting state-feedback gains were computed using the Matlab command `lqr` and resulted in the following gain matrix:

$$\mathbf{K} = \begin{bmatrix} 180 & 19 \end{bmatrix}$$

In addition, the observer gain matrix **L** was selected via pole placement and was iteratively found to give the best estimation results when the poles are located at $-2, -4$:

$$\mathbf{L} = \begin{bmatrix} 0.69 & 1.0 \end{bmatrix}^T$$

It was observed that changing the values of both **Q** and R over a broad range resulted in minimal changes in the controller gains. Thus, **Q** and R were simply set to the identity matrix and 1 respectively. The resulting response to an input disturbance is shown in Figure 14 below.



(a) Lean Angle Response

(b) Actuator Effort

Figure 14: Lego LQR Controller Response

3.5 H_∞ Loop-Shaping

The second order model of the bicycle is an extremely simplified formulation of the actual system dynamics and parameters of the bicycle are only approximately known. Furthermore, the plant changes significantly with forward speed. Thus, it is important to maximise the robustness of the closed-loop system to plant perturbations, so that stability is achieved for a broad range of uncertainties. H_∞ loop-shaping is concerned with robust stabilisation and thus an ideal candidate for this problem. The aim is to find a stabilising controller that maximises robustness, whilst trying to achieve a desired loop-shape of the return ratio. This section briefly covers aspects of the theory and then describes the prototype controller design process using this method.

For a general transfer function matrix $G(s)$, which is analytic in the right-half plane and satisfies $\sup_{Re(s)>0} |G(s)| < \infty$, the H_∞ norm of the system is defined as,

$$\|G(s)\|_\infty = \sup_\omega \bar{\sigma}(G(j\omega)).$$

Where $\bar{\sigma}$ denotes the maximum singular value. For a single-input, single-output system the problem reduces to finding the supremum of the magnitude for all ω.
Since the bicycle transfer function used for controller design has poles in the right-half plane, it is not in H_∞. Therefore, a normalised co-prime factorisation of the plant must be found,

so that

$$G_0(s) = \tilde{M}^{-1}\tilde{N}$$
$$\tilde{M}\tilde{M}^* + \tilde{N}\tilde{N}^* = I.$$

With $\tilde{M}$ and $\tilde{N}$ both in H_∞ and no common zeros in the extended right-half plane. Here, the factorisation is left-coprime. By then considering uncertainties in the coprime factorisations (Δ_M and Δ_N, both in H_∞), the uncertain plant can be modelled as:

$$G_\Delta = (\tilde{M} + \Delta_M)^{-1}(\tilde{N} + \Delta_N)$$

with

$$\|[\Delta_M, \Delta_N]\|_\infty \leq \epsilon.$$

Then, by the *Small Gain Theorem*, it can be shown that the closed-loop system is internally stable for all $\|[\Delta_M, \Delta_N]\|_\infty \leq \epsilon$ if and only if,

$$b(G, K) = \left\| \begin{bmatrix} K \\ I \end{bmatrix} (I - GK)^{-1} \begin{bmatrix} I & G \end{bmatrix} \right\|_\infty \geq \epsilon.$$

$b(G, K)$ is known as the stability margin. H_∞ loop-shaping aims to find a stabilising controller K that maximises $b(G_0W, K)$, where the weighting function $W(j\omega)$ shapes the return ratio as desired.

In practice, this is achieved using the Matlab *Robust Control Toolbox*, where functions are available for computing normalised coprime factorisations (`ncfmr`), as well as the optimal, stabilising controller using the *Glover-McFarlane* method (`ncfsyn`) proposed in [6].

Furthermore, it may be more convenient to use the ν-gap metric (denoted by $\delta_\nu(G_0, G_1)$) to check for robustness. In general, a gap between two plants G_0 and G_1 gives the smallest value of $\|[\Delta_M, \Delta_N]\|_\infty$ that perturbs G_0 into G_1. Thus, if the closed-loop system is stable for G_0 and K, then K will also stabilise G_1 if and only if,

$$\delta_\nu(G_0, G_1) < b(G_0, K).$$

Thus, after having computed a stabilising H_∞ optimal controller, the ν-gap metric can be used to determine the range of forward speeds, as well as the maximum uncertainties in system parameters, that can be tolerated in order to maintain closed-loop stability.

Table 3 below gives numerical values for the ν-gap between the nominal Lego prototype plant, again including the actuator model, and expected plant perturbations.

Parameter	Variation	ν-gap
Forward speed v	$\pm 30\%$	0.046,0.0615
Mass m	$\pm 5\%$	negligible
Center of mass h	$\pm 20\%$	0.057,0.047
Center of mass a	$\pm 20\%$	0.019,10.018
Wheel Base b	± 10	0.012,0.0098
Moment of Inertia D	$\pm 25\%$	0.022,0.022
Moment of Inertia J	$\pm 25\%$	0.073,0.057

Table 3: ν-gap Estimates for Expected Variations in Plant Parameters

As is evident from the table above, *single* variations in parameters are bounded by a maximum ν-gap of approximately 0.073. However, taking into account that there inevitably will be uncertainties in *all* parameters, the corresponding ν-gap between the nominal and true plant will be significantly higher. Thus, as a general rule of thumb, a stability margin $b(G_0, K)$ of at least 0.2 will be sought out to ensure robustness.

Lego Prototype

The frequency response of the nominal Lego Prototype model, including the actuator model, is shown in Figure 15 below.

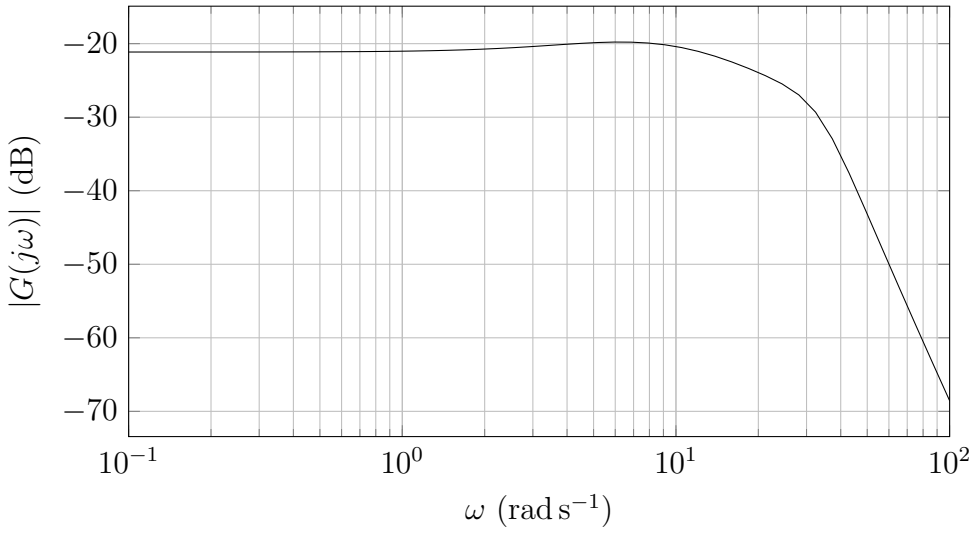


Figure 15: Bode Magnitude Plot for Lego Prototype Model

As previously mentioned, it is clear that a proportional gain of $K_P > 11.4$ is needed for stability at this nominal forward speed. Thus, as a first step, a proportional gain of $K_P = 12$ is included in the weighting function $W(j\omega)$. The plant can now be further *shaped* so that the closed-loop system has desirable properties, such as the typical requirements of reduced sensitivity at low frequencies (for good tracking and disturbance rejection) and reduced complementary sensitivity at high frequencies (to reduce the effects of sensor noise). Additionally, it is important to note that actuator limits must be taken into account, as not to make the demands on the system too large. This can be avoided by ensuring that the loop gains are not made arbitrarily large and that the target bandwidth of the system is bounded reasonably.

An appropriate weighting function, to give $b(G_0, K) \geq 0.2$, was then found by iteration and is comprised of three elements:

1. *Proportional Gain*: Keeps the cross-over frequency above the frequency of the right-half plane pole as to aid stability.

2. *Integrator*: Improves disturbance rejection and reference tracking by providing gain at low-frequencies.

3. *Zero*: Increases the bandwidth of the system and ensures the slope of the magnitude flattens off around the cross-over frequency. This helps improve the phase-margin, by keeping the phase away from $-180°$ at 0dB.

The overall weighting function is therefore,

$$W(s) = 12 \cdot \frac{s+1}{s}.$$

Then using Matlab to synthesize the controller resulted in a stability margin of $b(G_0, K) = 0.24$ with a sixth-order controller of the following form:

$$K(s) = W(s)K_\infty(s) = \left(12 \cdot \frac{s+1}{s}\right)\left(4 \cdot \frac{(s+0.17)(s+8.9)(s+18)(s^2+25s+1110)}{(s+1.1)(s+4.3)(s+48)(s^2+32s+2370)}\right)$$

The frequency response of the shaped loop, in comparison to the desired and nominal loop, is shown in Figure 16 below.

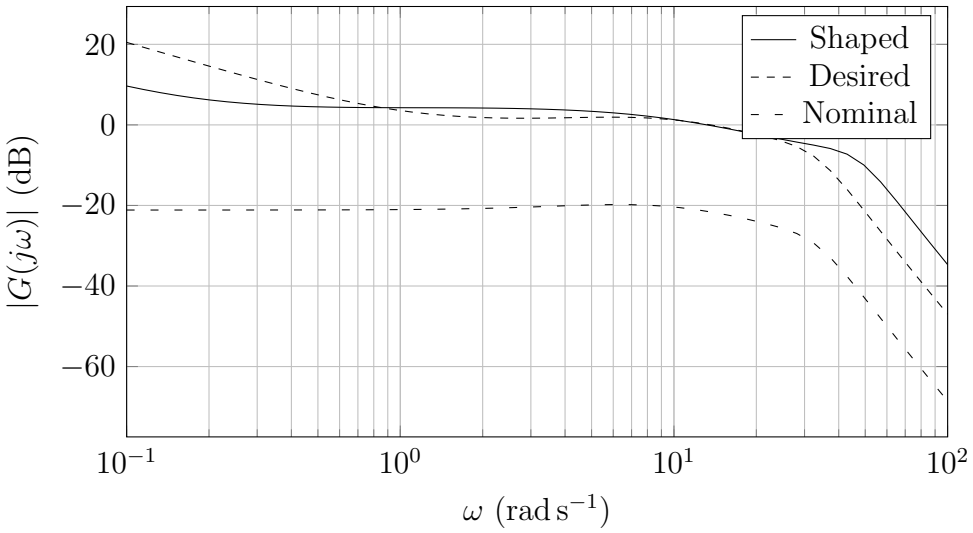
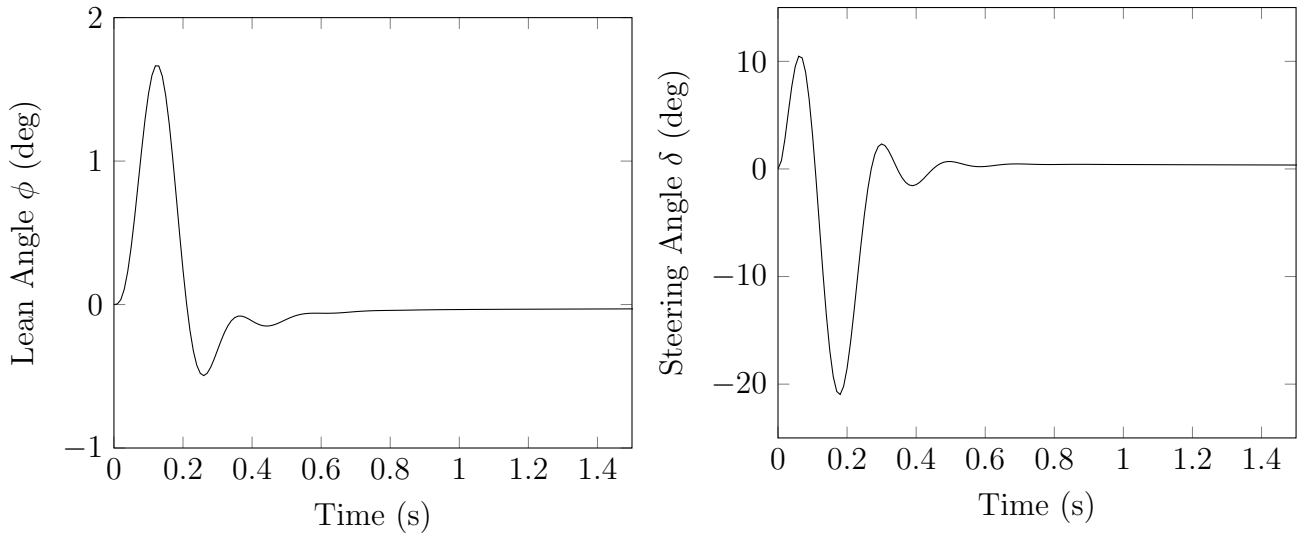


Figure 16: Bode Magnitude Plot for Shaped Lego Prototype

As can be seen from the plot, the final loop shape is not exactly the desired loop shape. In particular, the gain at low frequencies is reduced to maintain stability, however the cross-over frequency is approximately the same. The response to an input disturbance using the H_∞ loop-shaping controller can be seen in Figure 17 below.



(a) Lean Angle Response

(b) Actuator Effort

Figure 17: Lego H_∞ Controller Response

3.6 Non-Linear Methods

As opposed to having several linear controllers gain-scheduled for different operating points, want a single controller that works sufficiently well at *any* forward speed.

3.7 Comparison and Robustness

In simulation, out of all investigated controller designs, H_∞ loop-shaping provided the most desirable response. In particular, the recovery time due to a disturbance is small, the actuator effort reasonably bounded, due to the integrator there is zero steady-state error, and importantly the lean angle is kept small as to keep the system in the linear operating regime. Furthermore, the H_∞ controller synthesis also directly provides an estimate of the robustness of the closed-loop system, as stated previously.

On the other hand, PID control is the simplest to implement and tune *on-the-go*, whilst still providing satisfactory performance and the smallest controller effort. As a measurement of robustness, the phase margin of the nominal, PID-controlled system was found to be adequate at just under 30°.

Lastly, the simulated LQR controller performance was the worst out of all three. In particular, the lean angle deviated the most with an excessive steering angle imposed on the actuator to maintain stability. A measure of robustness for this controller type is also not directly available.

3.8 Implementation Details

All controller designs were performed in the continuous domain. However, as the controllers will eventually be implemented on a digital microprocessor, they need to be discretised. This is achieved by using the *Tustin* transformation, which maps the s to the z domain as follows:

$$s \to \frac{2}{T}\frac{z-1}{z+1}$$

The Tustin transform achieves the best frequency domain match between continuous and discrete systems. The choice of sampling time T is critical for two reasons:

1. A small value of T will reduce the effects of *frequency warping* induced by using the Tustin transform.

2. T must be chosen small enough so that the controller can react to the plant dynamics up to all relevant frequencies. As a general rule of thumb, the sampling time should be chosen so that $T \leq \frac{2\pi}{10\omega_c}$, where ω_c is the bandwidth of the plant.

However, the smallest possible sampling time is limited by the computational power of the processor.

Another implementation detail is the prevention of integrator wind-up. For instance, a large error signal sustained over a longer period of time can cause an integrating controller to produce a very large output, which will cause large amounts of overshoot, as well as instability. To counter-act this problem, all integrators will be limited using a dynamic clamping scheme proposed in [7].

As a last consideration, the controller output is limited to $\pm 60^\circ$ as to not cause excessive steering angles to be demanded of the actuator.

4 Implementation: Lego Prototype

After having completed the modelling and controller design for both bicycles, the final designs could then be tested on the real-world system. Firstly, the implementation using the Lego prototype bicycle was investigated, since it is quickly assembled, and easier to test in a confined space, compared to the full-scale bicycle.

4.1 Hardware

The Lego prototype was made up of standard *Lego Mindstorms EV3* parts. In particular, two large motors are used in parallel to propel the bicycle forwards at a maximum possible speed, whilst a smaller motor was placed at the handlebars to actuate the front fork assembly. Only a single gyroscopic sensor was available to provide a measurement of the rate of change of the lean angle. Lastly, the *Lego EV3 Brick* served as the central processing system, with a 300 MHz ARM processor, 16MB of Flash memory, and 64MB of RAM.

4.2 Software

The microprocessor was programmed using RobotC, a C-derivative specifically for Lego systems. The software structure is outlined in pseudo-code below.

Listing 1: Lego Software Pseudo-Code

```
initialiseSensorsAndMotors();
bias = calibrateGyroscope();
startDriveMotors(maxSpeed);
startTimer();

while (true) {
```

```
    if (currentSampleTime >= desiredSampleTime) {
        leandot = getSensorMeasurement() - bias;
        lean = estimateLeanAngle(leandot);
        ctrl = computeControllerOutput(0.0 - lean);
        pos = actuateHandlebarMotor(ctrl);
        logData(time,lean,ctrl,pos);
    }
}
```

The calibration routine records a fixed number of measurements and calculates their mean. This mean bias is then subtracted from each subsequent measurement in the control loop. The controller function computes the output of a difference equation obtained via discretisation, as mentioned in the previous section. Logging of data was achieved via Bluetooth using RobotC's inbuilt logging functionality. Finally, the desired sample time was chosen to be $T = 0.01s$.

4.2.1 Lean Angle Estimation

The Lego bicycle is only equipped with a gyroscopic sensor, meaning that only the rate of change of the lean angle is directly measurable. Direct integration of this measurement to give an estimate of the lean angle is not possible, since the measurements are corrupted by noise and the gyro has a time-varying bias. This noise and gyro bias will cause a drift in the estimated lean angle over time and thus deviate quickly from the true lean angle.

Therefore, it is necessary to estimate the lean angle at every timestep using a continuous-discrete *Extended Kalman Filter* (EKF). The EKF fuses the approximately-known, linearised, continuous system dynamics with discrete, noisy sensor readings to give an overall improved state estimate. Th full derivation of the EKF is given in [8]. The state vector and model dynamics are taken from the second-order model given in section 2.3. Note that this entire state estimation step would not be necessary if an accelerometer were available, which could be used as reference to correct for drift.

The continuous-discrete EKF can be divided into two steps. Firstly, the *prediction* step uses the system dynamics to estimate the state vector a timestep T in the future:

$$\underline{\hat{x}} = \underline{\hat{x}} + T \cdot \mathbf{A}$$
$$\mathbf{P} = \mathbf{P} + T \cdot (\mathbf{AP} + \mathbf{PA}^T + \mathbf{Q})$$

Where $\underline{\hat{x}}$ is the state estimate vector, $\mathbf{A}$ is the linearised state-transition matrix, $\mathbf{P}$ is the

error covariance matrix, and $\mathbf{Q}$ is the model noise matrix.

Following this, the *update* step fuses the linearised dynamics with the noisy sensor readings:

$$\mathbf{K} = \mathbf{P}\mathbf{C}^T(\mathbf{R} + \mathbf{C}\mathbf{P}\mathbf{C}^T)^{-1}$$
$$\mathbf{P} = (\mathbf{I} - \mathbf{K}\mathbf{C})\mathbf{P}$$
$$\underline{\hat{x}} = \underline{\hat{x}} + \mathbf{K}(\underline{y}_m - \mathbf{C}\underline{\hat{x}})$$

Where $\underline{y}_m$ is the measurement vector, $\mathbf{C}$ is the state output matrix, $\mathbf{R}$ is the measurement noise matrix, and $\mathbf{K}$ is the Kalman gain, which places a weighting on the measurements and internal model predictions.
$\mathbf{Q}$ is chosen depending on the model accuracy, whereas $\mathbf{R}$ is set to the noise covariance found in the sensor datasheet.

Lastly, it is important to note that an approximate model of the dynamics is used in conjunction with a noisy, biased sensor. Even though the resulting state estimation performance is much improved over using only the gyroscope readings, the inevitable inaccuracies will accumulate and degrade the lean angle estimate as time progresses. This ultimately has a direct impact on control performance and also only allows the final system to be tested for short periods of time before requiring a reset.

This is displayed in the simulated EKF state estimation in Figure 18 below, where it is evident that after an initially good estimation performance, after a brief amount of time the lean angle estimate starts to drift away from the true value.

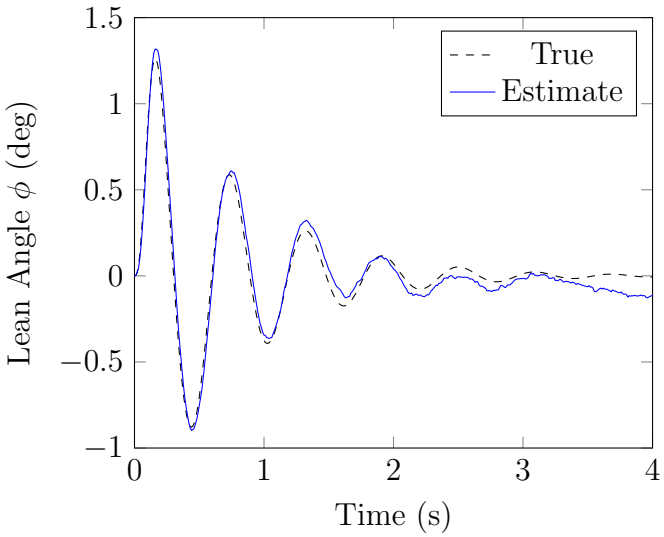


Figure 18: EKF Lean Angle Estimation

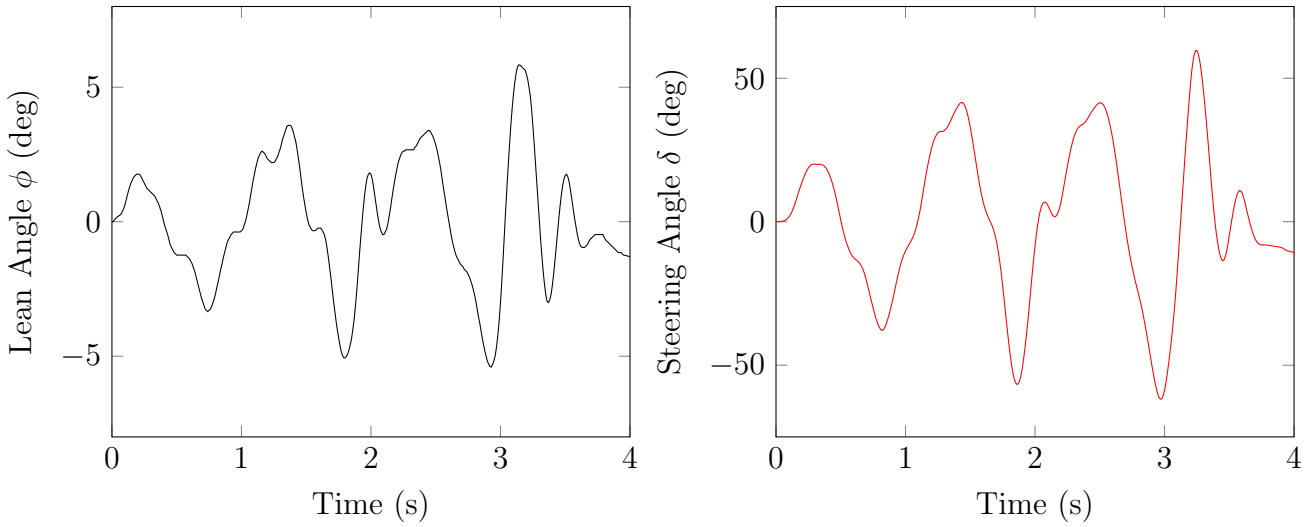
4.3 Results

4.3.1 PID

Proportional Only

A proportional-only controller was indeed able to stabilise the bicycle. As predicted by the second-order model and simulation, the bicycle could be stabilised at the given forward speed of $V = 0.44ms^{-1}$ by a gain greater than 11.4.

Figures 19 a) and b) show an excerpt of the response and actuator effort of the Lego prototype for a proportional gain of $K_P = 12$.



(a) Lean Angle Response

(b) Actuator Effort

Figure 19: P-Controller Response ($K_P = 12$)

The response for the simple proportional controller is certainly not ideal, even though stability is achieved. The magnitude of the lean angle is bounded by approximately 5°, which is an acceptable deviation from the desired setpoint of 0°. However, there is an oscillation present in the response, indicating that the system is on the boundary of stability. Knocking the bicycle gently, as to add an output disturbance, destabilises it. Thus the closed-loop system is not robust and sensitive to disturbances in its current state. This is to be expected however, since a proportional gain is used which places the closed-loop poles very close to the imaginary axis.

Furthermore, the *jittering* behaviour also causes the bicycle to move off-track easily and not follow a straight path. Lastly, the controller effort is larger than ideal.

Full PID Controller

Hello

Servo Model Verification

As a brief aside, the servo model is again verified by running a simulation in Matlab but using the real-world controller output data. As can be seen in Figure 20, the match is near perfect and confirms the actuator model identified in an earlier section of this report.

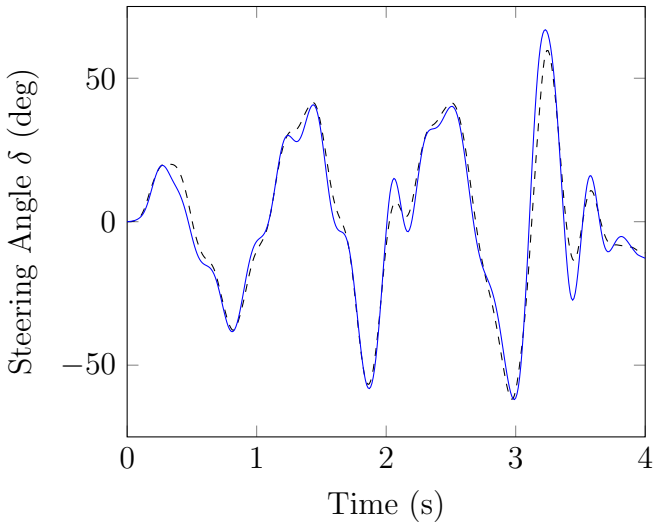


Figure 20: Lego Servo Model Verification

4.3.2 LQR

Unfortunately, the LQR controller was not able to stabilise the Lego prototype, even with varying feedback gains. It is assumed that the reason for this instability is that not only is the system relying on an approximate estimate of the lean angle, but it is now also relying on an additional state observer, which is using this degraded state estimate to again infer a further set of internal states. Thus, these combined errors will be fed back to the control law, which as seen from the previous controller type is already at its limits using a single state estimator.

4.3.3 H_∞ Loop-Shaping

4.3.4 Final Observations

hard to stabilise since h is small and time constants are short, not robust to disturbances. Have hardly any time to react. Additional limitation is maximum speed of lego model, which causes it to be very unstable. Stabilisation can be achieved but not a very good performance. Performance could probably be improved if we were to use a better angle estimation system, i.e. use an IMU. As soon as gyro estimate starts drifting we're in trouble (which turns out to be pretty quickly)...

5 Implementation: Full-Scale Bicycle

Having successfully stabilised prototype bicycle...

(a) Unmodified Form

(b) After Modification

Figure 21: Full-Scale Bicycle Before and After Modification

5.1 Hardware

Arduino (32-bit instead of 8-bit) as controller, why particular servo, servo mounting design, battery, voltage regulators, drive motor, motor controller disassembly, etc.

5.1.1 Bicycle Frame

5.1.2 Drive Motor

5.1.3 Handlebar Actuator

5.1.4 Microprocessor and Sensors

5.1.5 Miscellaneous

5.2 Software

5.2.1 Code Layout

Loops at different sample times: IMU, controller, telemetry, etc.

5.2.2 Lean Angle Estimation

Again, it was necessary to estimate the lean angle from raw sensor data. In the case of the full-scale bicycle however, an *inertial measurement unit* (IMU) was available, comprised of three gyroscopic sensors and three accelerometers, each aligned along an orthogonal axis. This meant that gyroscopic drift could now be directly accounted for without having to resort to the computationally expensive Kalman filter used for the Lego prototype.

At rest, an accelerometer outputs a signal directly proportional to the gravitational vector, this can then be used as a reference to correct for gyroscopic drift. However, when in accelerated motion, the accelerometer output will be *corrupted* by other accelerations terms, such as linear or Coriolis.

The problems present in both accelerometers and gyroscopic sensors are alleviated by the use of a complementary filter. Simply put, an accelerometer is useful over a long duration of time, while a gyroscopic sensors useful over a short period of time. It is therefore possible to low-pass filtere Combining both favourable aspects of these sensors leads to the use of a complementary filter.

Complementary filter, gyro HPF and acc LPF, more stable since we have accelerometer to correct for drift

5.2.3 Basestation and Telemetry

Figure 22: Screenshot of Bicycle Basestation

Packet handling, vary controller parameters on the fly, live data logging and display, safety start/stop, etc..

5.3 Results

6 Conclusions and Future Work

For the actual real-world implementation, controller design is actually a small part of the work. Need to worry about acquiring good estimates of states, writing all the peripheral software, drivers, sizing actuators, modelling the system, estimating parameters, etc, etc.
A controller that works in theory actually takes quite a lot of effort to work in practice.
Extremely difficult to stabilise this system, made even harder due to underactuation and difficulty in verifying model. Possibly could try adding flywheel to provide lean torque. Maybe better modelling.
Currently only using lean angle feedback, to keep on straight path would need some way of knowing heading - future work?

References

[1] Francis J. W. Whipple. *Stability of the Motion of a Bicycle*. The Quarterly Journal of Pure and Applied Mathematics, Vol XXX, 1899.

[2] A. L. Schwab, J. P. Meijaard, J. M. Papadopoulos. *A Multibody Dynamics Benchmark On The Equations of Motion of an Uncontrolled Bicycle*. 5th Nonlinear Dynamics Conferencec, 2005.

[3] Karl J. Åström, Richard E. Klein, Anders Lennartsson. *Bicycle Dynamics and Control: Adapted bicycles for education and research*. IEEE Control Systems, 10.1109/MCS.2005.1499389, September 2005.

[4] International Standard ISO 8855. *Road Vehicles - Vehicle Dynamics and Road-Holding Ability - Vocabulary*. Second Edition, December 2011.

[5] J. M. Papadopoulos, R. S. Hand, A. Ruina. *Bicycle and Motorcycle Balance and Steer Dynamics*. Draft, July 1990.

[6] Duncan McFarlane, Keith Glover. *A Loop Shaping Design Procedure Using H_∞ Synthesis*. IEEE Transactions on Automatic Control, Vol. 37 No. 6, June 1992.

[7] D. Wilson. *Teaching Your PI Controller to Behave*. Online: https://e2e.ti.com/blogs_/b/motordrivecontrol/archive/2013/04/13/teaching-your-pi-controller-to-behave-part-vii, April 2013.

[8] R. W. Beard, T. W. McLain. *Small Unmanned Aircraft: Theory and Practice*. Princeton University Press, 2012.

Appendix

A Risk Assessment