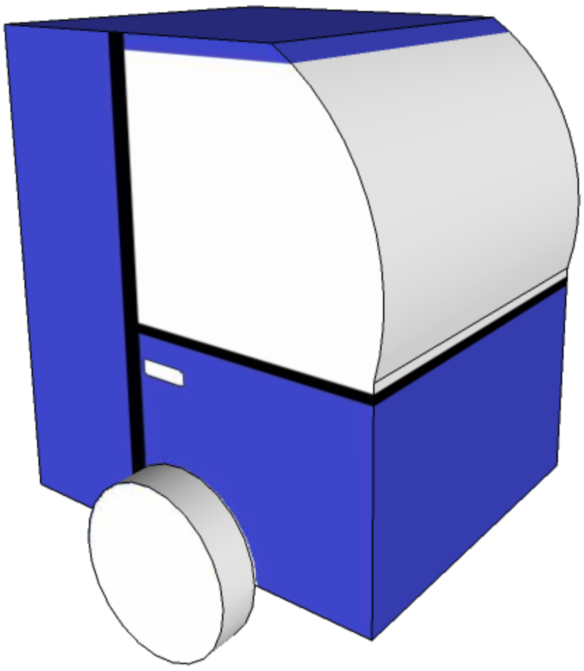
CHALMERS



One axle light weight truck

Dynamical modeling and implementation of a one axle vehicle

Master of Science Thesis by

ADNAN KLEMPIC
ANDREAS JANSSON

Department of Signals & Systems
Systems, Control & Mechatronics
Chalmers University of Technology
Gothenburg, Sweden 2012

Report Number EX022/2012

Abstract

The cities of tomorrow impose higher demands on flexibility and low pollution on vehicles distributing goods within city borders. Vehicles that have high maneuverability, light weight and zero emission are needed. In this master thesis a first assessment of the feasibility of a single axle truck concept is made.

A mathematical model based on Lagrangian mechanics is derived for this single axle truck concept. The model has five degrees of freedom including suspension. It is assumed that the modeled vehicle always has contact with the ground and no slip occurs. An LQG controller is used to balance and control the vehicle. Models of sensors and electrical motors are implemented in order to better simulate reality. It is possible to adjust all parameters and perform simulations of different driving scenarios due to the modular design of the model.

The LQG controller is able to control the vehicle sufficiently. As long as the wheels have full traction and the motors can deliver enough torque within a certain time, the truck can handle most normal situations. Simulations show that the vehicle can maneuver on an incline as well as hit a curb without falling over. Noisier sensor readings and longer sampling times impairs the performance of the vehicle and increases energy consumption due to higher torque fluctuations.

Acknowledgements

We would like to thank our three supervisors Stefan Pettersson (Viktoria Institute), Niklas Thulin (Volvo Technology) and Torsten Wik (Chalmers). A special thanks goes to the Viktoria Institute for letting us use their locales and resources.

Adnan Klempic and Andreas Jansson,
Gothenburg April 9, 2012

Contents

List of Figures

List of Tables

1

Introduction

The cities of tomorrow impose higher demands on flexibility and low pollution on vehicles distributing goods within city borders. Inner city areas are particularly vulnerable to emissions caused by fossil fueled vehicles. Emission induced health problems and congestion are becoming bigger and bigger problems for many cities to deal with.

To battle these problems, governments around the world are looking into the possibility of banning fossil fueled vehicles in city centers. Today, there are already bans against heavy trucks in cities and the future advances towards ever more restrictions against fossil fueled vehicles [1]. However, there is still a demand for distributing goods within city borders. To fill that demand new types of vehicles have to be constructed. Vehicles that have high maneuverability, light weight and zero emissions are needed. Such new types of vehicles could possibly be used in warehouses, logistics centers and depots.

1.1 Lightweight mobile truck

A one axle lightweight truck that operates on the principle of an inverted pendulum (see Figure 1.1) could be the answer to future city demands. To put it in a layman's words, itcan be seen as a bigger version of the known Segway™personal transporter. Since the truck will be electrically driven it will produce no emissions and it will have extremely good maneuverability since it has only one axis.

A mathematical model of the truck has to be derived in order to do simulations and to construct a controller. Simulations can then prove or disprove if the concept vehicle is plausible to apply in the real world. There are no limits on how advanced a mathematical model can be. Generally, a more advanced model gives a better mathematical representation of reality while a less advanced model is easier to work with and requires less computational effort to simulate. Among the variables that can be included or varied in the model is the number of degrees of freedom, air resistance, wheel resistance,

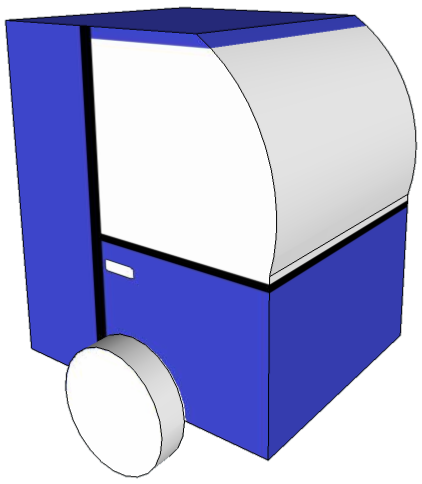


Figure 1.1: Sketch of the one axle truck

suspension etc.

1.2 Purpose

The purpose of this thesis can be divided into smaller sub goals that all together enable simulation of different driving cases and parameter variation. The sub goals are:

- Make a sufficiently good mathematical model over the truck.
- Design a controller that can balance the truck in upright position. It should also be able to move and turn the truck without it falling over and crashing.
- Add sensor noise and a motor model.
- Do a parameter study where noise level, weight, center of mass and sampling time are changed and the effects studied.
- Simulate what happens when the truck drives on a slope and when the truck hits a curb.
- Examine if it is a viable option to build and use in reality.

Note that these goals have to be fulfilled in order after each other since this model is only implemented in a simulation environment and not built in reality. For example, it is necessary to have a mathematical model before designing the controller. Also, it is required that the mathematical models and the controller are set up and designed before any simulations and calculations on the performance of the truck can be made.

1.3 Limitations

Avoiding a model with too much complexity means that certain assumptions have to be made. These assumptions leads to limitations in the model which cause some phenomena that occur in reality but not in the simulations. The most important limitations are:

- No slip between the wheels and the ground.
- The wheels are always in contact with the ground.
- Simplified models of:
  - Sensors.
  - Measurement noise.
  - Electrical motors.

It would be possible to include or improve these properties in the model. However, the focus of this thesis is on a parameter study to see if the concept is feasible and not on handling aspects. However, it would present an interesting topic for somebody else to research.

2

Theory

In this chapter some background theory is given so that the reader may better understand some of the elements in this report. It is assumed that the reader has some understanding of basic physical systems and notions such as Newton's laws of motion.

2.1 Mathematical modeling

A mathematical model of the truck has to be derived. This is important because it will enable simulations to be done on the truck. To make a sufficiently detailed model several approaches can be used.

In this report Langrange's equations is used to derive a model of the truck. Some of the basic theory behind the inverted pendulum, air resistance dynamics and a suspension model is presented in this section to improve the understanding of the implementation in Chapter 3.

2.1.1 Lagrangian mechanics

Lagrangian mechanics is a reformulation of classical mechanics and belongs to the branch of analytical mechanics. The equations of motion in Lagrangian mechanics are called Lagrange's equations and they are a mathematical approach for deriving equations of motion for mechanical systems [2].

When deriving the equations of motion for complex systems it is in many cases considerably simpler to use the Lagrange's equations compared to using the classical mechanics approach with Newton's laws. However, both approaches, Newton's Law and Lagrange's equations, are completely compatible with each other.

To derive the equations of motion with Lagrange's equations it is important to look at the existing energies in the system [3]. There are two types of energies for mechanical systems:

1. Kinetic energy
    - Translational energy
    - Rotational energy
2. Potential energy

The Lagrange function ($\mathcal{L}$) denotes the difference between the total kinetic energy and the total potential energy of the system

$$\mathcal{L} = T - V, \tag{2.1}$$

where T is the total kinetic energy and V is the total potential energy in the system.

Once the Lagrange function is derived it is possible to derive the equations of motion using Lagrange's equations

$$Q_j = \frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_j}\right) - \frac{\partial \mathcal{L}}{\partial q_j}, \tag{2.2}$$

where:

Q_j	-	Sum of applied forces projected on direction of generalised coordinates.
q_j	-	Generalised coordinates or displacement variables (x, y, z, θ, ϕ, etc.).
j	-	The number of generalized coordinate/displacement variable.
$\mathcal{L}$	-	Lagrange function.

If Q_j equals zero then there is no external influence on the system. However, in reality there are always non-conservative forces, such as friction or torque from a motor, that affects the system. In those cases simply let Q_j equal the external forces and a complete model of the dynamical system is derived.

An example of how Lagrange's equations can be used to derive the equations of motion are found in Section 2.1.2.

2.1.2 Inverted pendulum

A vehicle where the chassis is connected to one axle has the same dynamic as an inverted pendulum.

A pendulum that can rotate 360° has two equilibrium points. One point is at the bottom where the pendulum has the least potential energy. The second point is at the top where the pendulum will remain assuming that the string is stiff and not affected by any horizontal force at all. Seeing the pendulum from a control point of view, the lower point is a stable point and the upper point is unstable. It means in reality that if one wants the pendulum to remain in the upper point then the entire pendulum must have the ability to be moved and the movement has to be controlled.

One way of moving the pendulum could be by having a cart attached to the pendulum as Figure 2.1 illustrates. The cart should be moved forward or backwards depending on the direction in which the pendulum is falling. This is the way the classical controlled inverted pendulum problem is defined and it corresponds well with the mathematical model of the truck presented in this thesis. However, the truck has one very important difference. Instead of having a cart attached to the pendulum, a wheel axis is attached in its place. The difference will be that the axis will not only give a motion forward and backwards but also a torque at the point where it is attached to the stick of the pendulum.

The equation of motion for the cart controlled pendulum in Figure 2.1 can be derived using *Lagrange's equations*. A overview of the Lagrange mechanics is found in Section 2.1.1.

Let x represent the movement of the cart in the horizontal direction and α the angle the pendulum is leaning from its upright position (see Figure 2.1). Summing the translational and potential energy of the pendulum gives the Lagrangian:

$$\mathcal{L} = \frac{1}{2}Mv_1^2 + \frac{1}{2}mv_2^2 - mg\ell\cos(\alpha).$$

where:

$(1/2)Mv_1^2$	-	Kinetic energy of the cart.
$(1/2)mv_2^2$	-	Kinetic energy of the pendulum.
$mg\ell\cos(\alpha)$	-	Potential energy of the pendulum.
$v_1^2 = \dot{x}^2$	-	Velocity of the cart
$v_2^2 = \dot{x}^2 - 2\ell\dot{x}\dot{\alpha}\cos(\alpha) + \ell^2\dot{\alpha}^2$	-	Velocity of the pendulum
M	-	Mass of the cart.
m	-	Mass of the pendulum.
l	-	Length of the pendulum.

Rewriting the Lagrangian gives:

$$\mathcal{L} = \frac{1}{2}\left(M+m\right)\dot{x}^2 - m\ell\dot{x}\dot{\alpha}\cos\alpha + \frac{1}{2}m\ell^2\dot{\alpha}^2 - mg\ell\cos\alpha.$$

Using Lagranges equations below, the equations of motion can be derived:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}}\right) - \frac{\partial L}{\partial x} = F,$$
$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\alpha}}\right) - \frac{\partial L}{\partial \alpha} = 0,$$

where F is the non-conservative forces i the system. And x and α represent the two degrees of freedom of the cart. That means that the cart and pendulum cannot move or

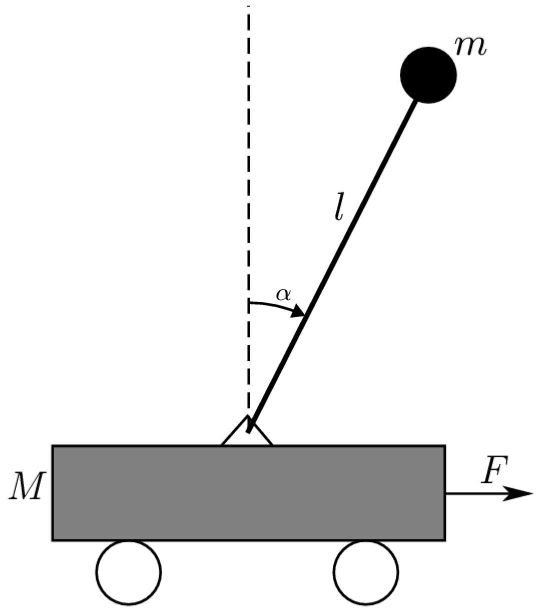


Figure 2.1: Inverted pendulum on a cart

rotate in any other direction.

Solving the derivatives of the equations of motion gives:

$$\begin{aligned}(M+m)\,\ddot{x} - m\ell\ddot{\alpha}\cos\alpha + m\ell\dot{\alpha}^2\sin\alpha &= F,\\ \ell\ddot{\alpha} - g\sin\alpha - \ddot{x}\cos\alpha &= 0.\end{aligned}$$

These equations can now be used in Simulink to describe the motion of the inverted pendulum. It is also possible to linearize the equation and then use the linearization to design a controller.

2.1.3 Suspension

Since the truck is intended to be driven by a human, it is important to equip it with a suspension system. This adds complexity to the dynamic model and there exists (to our knowledge) no documentation about an inverted pendulum model resting on two wheels with suspension. In this section, the dynamics of a simple suspension model (see Figure 2.2) that will be used in the complete model of the truck in Chapter 3 is presented [4].

The distance between the two points Z_1 and Z_2 can be expressed as $\Delta Z = Z_1 - Z_2$. The force in the suspension can then be described by

$$F_{spring} = -k\cdot\Delta Z + \Delta\dot{Z}\cdot b, \tag{2.3}$$

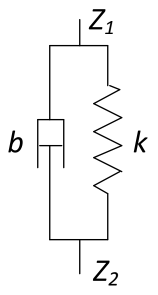


Figure 2.2: The suspension model used for the truck

where

k	-	The spring constant.
b	-	The damping constant.

2.1.4 Air resistance

An air resistance force affects the truck when it is moving forward or backwards. It is necessary to include the effect of that force in the mathematical model since it influences the dynamics of the truck. Obviously, the air resistance is dependent on how fast the truck is moving [4]. It is assumed that the drag is described by

$$F_{drag} = \frac{1}{2}\rho v^2 C_d A, \tag{2.4}$$

where

ρ	-	Density of the air ($\rho_{air} \approx 1{,}2$).
v	-	Speed of the object relative to the air.
C_d	-	Drag coefficient.
A	-	Area of the plane perpendicular to the direction of motion.

Some additional assumptions are made for the air resistance model when applying it to the final truck model in Chapter 3:

1. The frontal area of the truck is considered to be rectangular and of the same size as the back area.

2. When the truck is moving it will lean forward or backward. The vertical cross sectional area A will then slightly differ from when the truck is upright. In reality the area A should be a function of ψ (see Figure 3.1(a)). However, in this report, A is assumed to be constant.

2.1.5 Rolling resistance

When a vehicle is moving there is a rolling resistance caused by deformations of the tire as it rolls [4]. The rolling resistance increases with the rotational speed of the wheel. It is described as a torque working in the opposite direction of the wheel spin.

$$M_{rol.fr.} = k_{friction}\omega \tag{2.5}$$

where,

$M_{rol.fr.}$	-	Rolling resistance.
$k_{friction}$	-	Rolling resistance constant depending mostly on the tire and road type.
ω	-	Rotational speed of the wheel.

2.2 Controlling the truck

A controller has to be able to keep the truck upright while moving or remaining still. The cabin of the truck must under no circumstances hit the ground by falling over. This poses some challenges and there are numerous different controllers that can be used for the same purpose. Here, a Linear Quadratic Gaussian controller, commonly abbreviated as the LQG controller, will be used to stabilize the truck. The LQG controller consists of a Kalman filter combined with a Linear Quadratic Regulator (LQR) [5].

2.2.1 Linear Quadratic Regulator (LQR)

An LQR is an optimal control regulator which minimizes a cost function J, described by

$$J = \int_0^\infty \left(x^TQx + u^TRu + 2x^TNu\right) dt \tag{2.6}$$

Note: x is here a vector of all state variables.

The cost function is weighted with weighting factors (Q, R and N) which are tuned by the control engineer. These weighting factors decide how much the deviation from the reference signal from the different states of the system should influence the size of the control signal [6].

Consider an LTI [1] system described by

$$\dot{x} = Ax + Bu \tag{2.7}$$

The minimal solution of the cost function gives the state feedback law $u = -Lx$, where L is derived by solving S in the Riccati Equation discribed by

$$A^TS + SA - (SB + N)R^{-1}(B^TS + N^T) + Q = 0. \tag{2.8}$$

[1]Linear time-invariant

The optimal gain matrix L is then calculated by using S in equation described by

$$L = R^{-1}(B^T S + N^T). \tag{2.9}$$

In Matlab the optimal gain matrix L can easily be obtained by using the command $[L,S,e] = lqr(SYS,Q,R,N)$. Note that Q, R and N are tuning parameters and they have to be tweaked by a control engineer until the desired performance is obtained. This is usually done in a simulation environment where the parameters are tuned until the performance specifications are satisfied.

Integral action

One of the drawbacks with the standard LQR is that there is no integral action. Practically, this means that if the controlled system is subjected to a constant force there will always be a constant error where the output deviates from the reference signal. In order to remove this error it is possible to include integral action in the LQR. This is simply done by adding an extra state of the tracking error $r - y$, where r is the reference signal and y is the measured feed back signal. This will add one extra state per signal y having integral action to the model, and the tuning matrices Q, R and N have to be adjusted accordingly. In the complete model of the truck in Chapter 3, integral action is needed since the truck must be able to stand still on slopes.

2.2.2 Kalman filter

Often it is not possible to measure all states of a system and therefore some states have to be estimated. In reality there is also always noise affecting the system, creating new problems that have to be handled. The Kalman filter tries to solve both problems by filtering away the process and measurement noise while also estimating the unmeasured states[7]. It is estimating the states by using the known inputs u and the measurement y to generate the state estimates $\hat{x}$ and $\hat{y}$ [5].

Assume that a system is described by

$$\dot{x} = Ax + Bu + Gw \tag{2.10}$$

$$y = Cx + Du + Hw + v \tag{2.11}$$

where,

A, B, C, D	-	System state matrices
G	-	Process noise gain matrix relating the process noise to the state variables
H	-	A gain matrix relating the disturbances directly to the measurements
w	-	White noise disturbance vector (process noise)
v	-	White noise disturbance vector (measurement noise)

The estimate error covariance, where $\hat{x}(t)$ is the state estimator, is described by

$$P = \lim_{t\to\infty} E(\{x(t) - \hat{x}(t)\}\{x(t) - \hat{x}(t)\}^T) \tag{2.12}$$

The state estimator $\hat{x}$ is derived from

$$\dot{\hat{x}} = A\hat{x} + Bu + K(y - C\hat{x} - Du) \tag{2.13}$$

where the Kalman gain K is obtain by solving

$$K = PC^T(CPC^T + R)^{-1}, \tag{2.14}$$

where P is obtained by solving the differential Riccati equation described by

$$\dot{P} = -PC^TR^{-1}CP + AP + PA^T + Q, \tag{2.15}$$

where R is the covariance matrix.

The Kalman gain can be tuned for improved performance. With a high gain, the filter follows the observations more closely. With a low gain, the filter follows the model predictions more closely. Note that the model prediction are based on a linearized model of the true system. Hence, the model prediction will only be as good as the linearized model. Using a Kalman filter produces estimates that are closer to the true unknown values than those that would be based on a single measurement alone or the model predictions alone.

2.2.3 Linear Quadratic Gaussian (LQG) Controller

The separation principle states that, under some assumptions, it is possible to design an optimal feedback controller by designing an optimal observer that feeds into an optimal controller for the system [8]. Practically it means that it is possible to separate the problem of designing an optimal feedback controller into two smaller problems. Figure 2.3 illustrates how an LQG controller is connected to a plant.

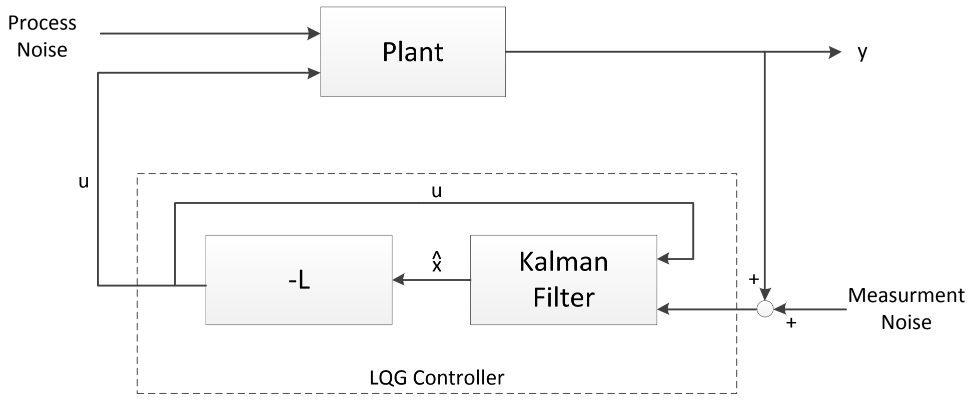


Figure 2.3: LQG control of a system

3

Implementation

3.1 System description

The vehicle is an inverted pendulum attached on an axle that is connected to two wheels. There is a suspension system located between the upper body and the wheel axle (see Figure 3.1(a)). The suspension is modeled as a linear spring in parallel connection with a damper. Equations that describe the dynamics can be found in Section 2.1.3.

The system is considered to have three mass centers of gravity located at left wheel, right wheel and upper body. They are given the coordinates (x_l, y_l, z_l), (x_r, y_r, z_r) and (x_b, y_b, z_b) (see Figure 3.1).

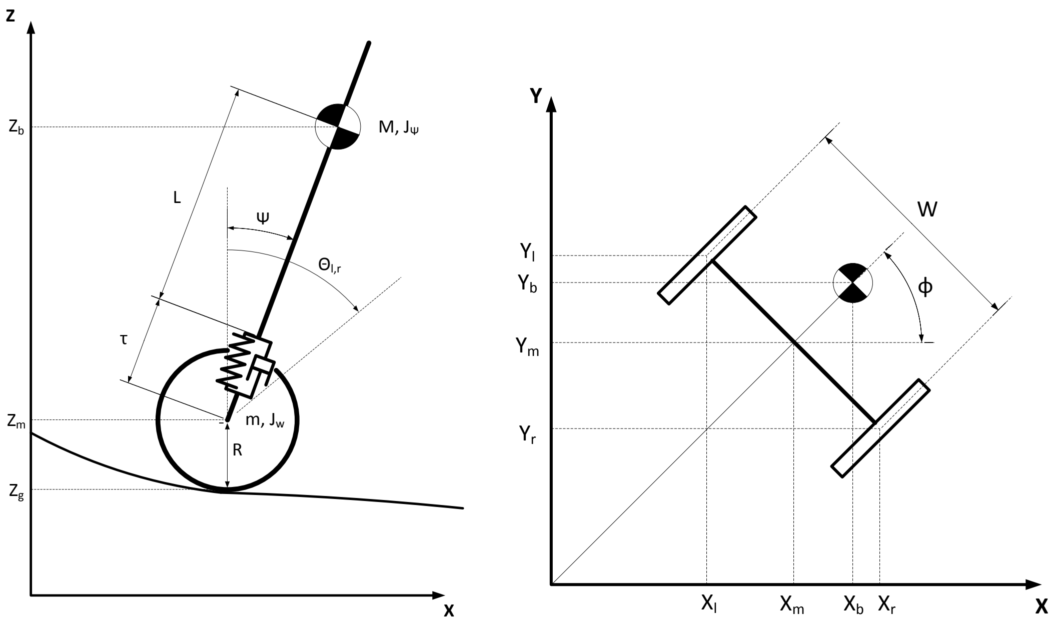
3.2 Lagrange Modeling

Lagrange's method has been used to create a mathematical model of the system (see Section 2.1.1). Formulating Lagrange's equations of motions requires a few steps that are described in this section. [9]

3.2.1 New coordinate system

The first step is to minimize the number of coordinates used to describe the system. This new set of coordinates is called the *generalized coordinates*. All of the other coordinates for the different mass centers have to be related to these new coordinates.

Since the vehicle is considered as a three mass system, three coordinates for each mass center needs to be described by the generalized coordinate system. One approach to do this is to start from the middle point of the vehicle and describe each mass center relative to this point. To do so requires that the position and orientation of the middle point of the vehicle (x_m,y_m,z_m) is known. The position of the wheels can be described from this point by knowing the width of the vehicle.



(a) Side view (b) Plane view

Figure 3.1: Vehicle overview

Five generalized coordinates were selected (see Table 3.2). θ, Φ and z_g are used to calculate the position and orientation of the middle point. Pitch angle ψ and the state of the suspension τ are used to describe the position of the upper body mass center (x_b,y_b,z_b).

The coordinates of the three mass centers are expressed relative to the middle point of the vehicle by

$$[x_l, y_l, z_l] = \left[x_m - \frac{W}{2} sin(\phi), y_m + \frac{W}{2}\cos(\phi), z_m\right], \tag{3.1}$$

$$[x_r, y_r, z_r] = \left[x_m + \frac{W}{2} sin(\phi), y_m - \frac{W}{2}\cos(\phi), z_m\right], \tag{3.2}$$

$$[x_b, y_b, z_b] = \Big[x_m + (\tau + L)\sin(\psi)\cos(\phi), \\ y_m + (\tau + L)\sin(\psi)\sin(\phi), z_m + (\tau + L)\cos(\psi)\Big], \tag{3.3}$$

where the position of the middle point is

$$[x_m, y_m, z_m] = \left[\int \dot{x}_m dt, \int \dot{y}_m dt, z_g + R\right]. \tag{3.4}$$

Table 3.1: System parameters

Terminology	Description	Value	Unit
g	Gravity	9,82	m/s^2
m	Mass of left and right wheel	15	kg
R	Wheel Radius	0,4	m
J_w	Wheel inertia moment	1,2	kgm^2
n	gear ratio	1	
M	Body weight	870	kg
W	Body width	1,6	m
D	Body depth	1,4	m
H	Body height	2	m
L	Distance between wheel axle		
	and center of gravity	0,67	m
J_ψ	Body pitch inertia moment	128,9	kgm^2
J_ϕ	Body jaw inertia moment	327,7	kgm^2
J_m	Motor inertia moment	$1 \cdot 10^{-5}$	kgm^2
A	Front area of the body	3,2	m^2
k	Linear spring constant	100000	N/m
b	Linear damping constant	10000	Ns/m

Table 3.2: Generalized coordinates

θ	-	Average angle of the two wheels, θ_l and θ_r
ψ	-	Pitch angle of the body
ϕ	-	Yaw angle of the body
z_g	-	The height position on the wheel axle
τ	-	Extension of the suspension

3.2.2 Lagrangian formulation

Formulating the Lagrangian equations requires a closer look at the existing energies in the system (see Section 2.1.1).

The translational kinetic energy is

$$T_{tr} = \frac{1}{2}m(\dot{x}_l^2 + \dot{y}_l^2 + \dot{z}_l^2) + \frac{1}{2}m(\dot{x}_r^2 + \dot{y}_r^2 + \dot{z}_r^2) + \frac{1}{2}M(\dot{x}_b^2 + \dot{y}_b^2 + \dot{z}_b^2). \quad (3.5)$$

The rotational kinetic energy is

$$T_{rot} = \frac{1}{2}J_w\dot{\theta}_l^2 + \frac{1}{2}J_w\dot{\theta}_r^2 + \frac{1}{2}J_\psi\dot{\psi}^2 + \frac{1}{2}J_\phi\dot{\phi}^2 + \frac{1}{2}n^2J_m(\dot{\theta}_l - \dot{\psi})^2 + \frac{1}{2}n^2J_m(\dot{\theta}_r - \dot{\psi})^2. \tag{3.6}$$

The potential energy is

$$U = mgz_l + mgz_r + Mgz_b + \frac{1}{2}k\tau^2. \tag{3.7}$$

3.2.3 Inserting the coordinates into the Lagrangian formulation

Taking the derivative of the coordinates in Equations (3.1)-(3.3) yields

$$[\dot{x}_l, \dot{y}_l, \dot{z}_l] = \left[\dot{x}_m - \frac{W}{2}\dot{\phi}\cos(\phi), \dot{y}_m - \frac{W}{2}\dot{\phi}\sin(\phi), \dot{z}_m\right], \tag{3.8}$$

$$[\dot{x}_r, \dot{y}_r, \dot{z}_r] = \left[\dot{x}_m + \frac{W}{2}\dot{\phi}\cos(\phi), \dot{y}_m + \frac{W}{2}\dot{\phi}\sin(\phi), \dot{z}_m\right], \tag{3.9}$$

$$\begin{aligned}[\dot{x}_b, \dot{y}_b, \dot{z}_b] = \Big[&\dot{x}_m + (\tau + L)\big(\dot{\psi}\cos(\psi)\cos(\phi) - \dot{\phi}\sin(\phi)\sin(\psi)\big) + \\ &\dot{\tau}\sin(\psi)\cos(\phi), \dot{y}_m + (\tau + L)\big(\dot{\psi}\cos(\psi)cos(\phi) - \\ &\dot{\phi}\sin(\phi)\sin(\psi)\big) + \dot{\tau}\sin(\psi)\cos(\phi), \\ &\dot{z}_m + \dot{\tau}\cos(\psi) - (\tau + L)\dot{\psi}\sin(\psi)\Big],\end{aligned} \tag{3.10}$$

where

$$[\dot{x}_m, \dot{y}_m, \dot{z}_m] = \left[R\dot{\theta}\cos(\phi), R\dot{\theta}\sin(\phi), \dot{z}_g\right]. \tag{3.11}$$

An expression for how the rotations of the left and right wheel affect the vehicle is needed. This motion has to be expressed in the generalized coordinate system. The average wheel angle and the jaw angle can be determined by

$$\theta = \frac{1}{2}(\theta_r + \theta_l), \tag{3.12}$$

$$\phi = \frac{R}{W}(\theta_r - \theta_l), \tag{3.13}$$

which can be rewritten as

$$\theta_r = \theta + \frac{W\phi}{2R}, \tag{3.14}$$

$$\theta_l = \theta - \frac{W\phi}{2R}. \tag{3.15}$$

The derivative is then

$$\left[\dot{\theta}_r, \dot{\theta}_l\right] = \left[\dot{\theta} + \frac{W\dot{\phi}}{2R}, \dot{\theta} - \frac{W\dot{\phi}}{2R}\right]. \tag{3.16}$$

The Lagrangian is derived by inserting Equation (3.16) in Equation (3.6) which gives the final formulation

$$\mathcal{L} = T_{tr} + T_{rot} - U \tag{3.17}$$

3.2.4 Equations of motion

The last step is to set up and solve Lagrange's equations of motion. The equations to be solved are

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{\theta}}\right) - \frac{\partial \mathcal{L}}{\partial \theta} = F_\theta, \tag{3.18}$$

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{\psi}}\right) - \frac{\partial \mathcal{L}}{\partial \psi} = F_\psi, \tag{3.19}$$

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{\phi}}\right) - \frac{\partial \mathcal{L}}{\partial \phi} = F_\phi, \tag{3.20}$$

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{\tau}}\right) - \frac{\partial \mathcal{L}}{\partial \tau} = F_\tau. \tag{3.21}$$

Solving these equations by hand is not very hard but very time consuming. To save time and to minimize the risk of making errors a Matlab script was used to derive the equations (see Appendix B.1). Solving the equations yields four equations, where each equation is describing the motion in the specified coordinate. Note that the script is also generating a fifth equation which corresponds the the motion at Z_g. It is necessary to include Z_g in the calculations to get the dynamics of Z_g in the other equations. But since the motion of Z_g will be controlled by the environment the equation itself is unused.

3.2.5 Implementation in Simulink

To be able to implement the dynamics in Simulink some modifications to the derived Lagrange's equations of motion are needed.

Solving Equations (3.18), (3.19), (3.20) and (3.21) yields

$$\ddot{\theta} = \frac{A1 - \ddot{\tau} \cdot A2 + \ddot{\psi} \cdot A3}{A4}, \tag{3.22}$$

$$\ddot{\psi} = \frac{B1 + \ddot{Z}_g \cdot B2 + \ddot{\theta} \cdot B3}{B4}, \tag{3.23}$$

$$\ddot{\phi} = C1, \tag{3.24}$$

$$\ddot{\tau} = \frac{D1 - \ddot{Z}_g \cdot D2 - \ddot{\theta} \cdot D3}{D4}, \tag{3.25}$$

where the expressions $A1$, $A2$, $A3$, $A4$, $B1$, $B2$, $B3$, $B4$, $C1$, $D1$, $D2$, $D3$ and $D4$ are dependent on $\dot{\psi}$, ψ, $\dot{\phi}$, ϕ, $\dot{\tau}$, τ, $\dot{Z}_g$ and Z_g. Note that Equations (3.22), (3.24) and (3.25) are interdependent while Equation (3.24) is not dependent on any of the other equations. The interdependency causes algebraic loops when implemented in Simulink. By doing multiple algebraic operations the interdependency between $\ddot{\theta}$, $\ddot{\psi}$ and $\ddot{\tau}$ can be broken.

The final result is four equations describing the motions of the vehicle. The complete equations can be found in Appendix A. Figure 3.2 illustrates how these equations were implemented in Simulink. Each one of the function blocks in Figure 3.2 contains an algebraic expression that calculates the equation of motion for a specific generalized coordinate.

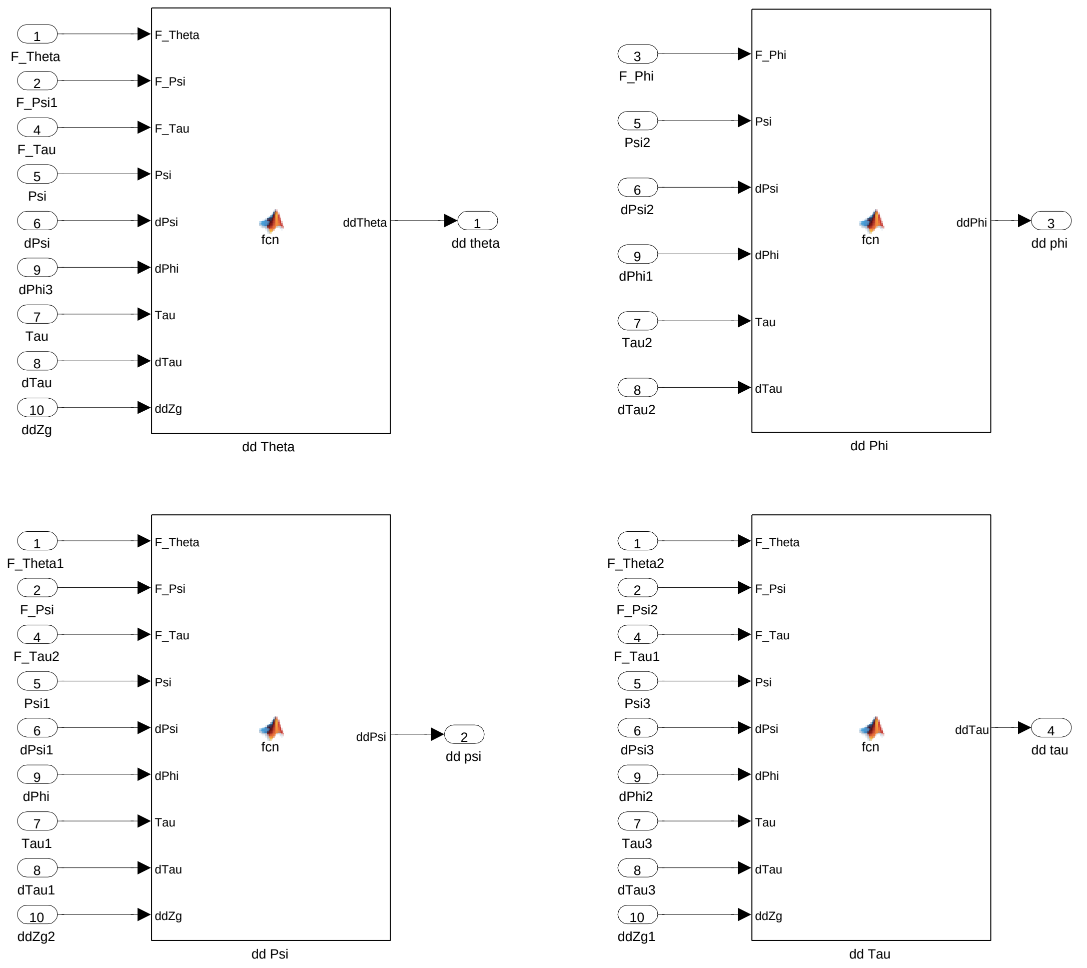


Figure 3.2: Function blocks in Simulink that calculates the equations of motion

3.2.6 External and non-conservative forces

External and non-conservative forces in the system are not included in the Lagrangian formulation and have to be modeled and implemented separately. External forces in the system can for example be torque from the motors and forces due to road inclination. Non-conservative forces are forces that do not transfer into potential or kinetic energy. Any friction-type like air resistance and frictions in the wheels are non-conservative forces [10]. Forces that are being taken into account are described in Table 3.3 (notation can be found in Table 3.1).

Table 3.3: External and non-conservative forces

External Forces	
Torque from the motors	$nM_{l,r}$
Turning force	$\frac{W}{2R}(F_r - F_l)$
Forces due to inclination	F_ξ
Non-conservative Forces	
Friction between wheel and motor	$f_m(\dot{\theta}_{l,r} - \dot{\psi})$
Friction in the wheel	$f_w\dot{\theta}_{l,r}$
Drag resistance	$\frac{1}{2}\rho AC\dot{x}_b^2$
Damping force in the damper	$-b\dot{\tau}$

The external and non-conservative forces are implemented as

$$\Big[F_\theta, F_\psi, F_\phi, F_\tau\Big] = \Big[F_l + F_r - F_\xi, F_\psi, \frac{W}{2R}(F_r - F_l), F_\tau\Big], \tag{3.26}$$

where

$$F_l = nM_l - f_m(\dot{\theta}_l - \dot{\psi}) - f_w\dot{\theta}_l, \tag{3.27}$$

$$F_r = nM_r - f_m(\dot{\theta}_r - \dot{\psi}) - f_w\dot{\theta}_r, \tag{3.28}$$

$$F_\xi = (M_w + M_b)(g + \ddot{Z}_g)sin(\xi)R, \tag{3.29}$$

$$F_\psi = -n(M_l + M_r) - f_m(\dot{\psi} - \dot{\theta}_l) - f_m(\dot{\psi} - \dot{\theta}_r) - F_{drag}, \tag{3.30}$$

$$F_{drag} = \frac{1}{2}\rho AC\dot{x}_b{}^2, \tag{3.31}$$

$$F_\tau = -b\dot{\tau}. \tag{3.32}$$

Inserting Equation (3.27)-(3.32) in (3.26) yields

$$F_\theta = n(M_l + M_r) + 2f_m\dot{\psi} - 2\dot{\theta}(f_m + f_w), \tag{3.33}$$

$$\begin{aligned} F_\psi = & - n(M_l + M_r) + 2f_m\dot{\theta} - 2f_m\dot{\psi} - \frac{H}{4}\rho AC... \\ & \big(\dot{x}_m + (\tau + L)(\dot{\psi}\cos(\psi)\cos(\phi)... \\ & - \dot{\phi}\sin(\phi)\sin(\psi)) + \dot{\tau}\sin(\psi)\cos(\phi)\big)^2, \end{aligned} \tag{3.34}$$

$$F_\phi = \frac{Wn}{2R}(M_r - M_l) - \frac{W^2}{2R^2}(f_m + f_w)\dot{\phi}, \tag{3.35}$$

$$F_\tau = - b\dot{\tau}. \tag{3.36}$$

3.3 Electric motor model

The motor dynamics are approximated by a first order transfer function with a time constant $motor\_delay = 20ms$. The time constant corresponds to the amount of time it takes for the motor's step response to reach 63% of its final value. The input torques are the desired torques while the output torques are the actual torques the motors are producing (see Figure 3.3).

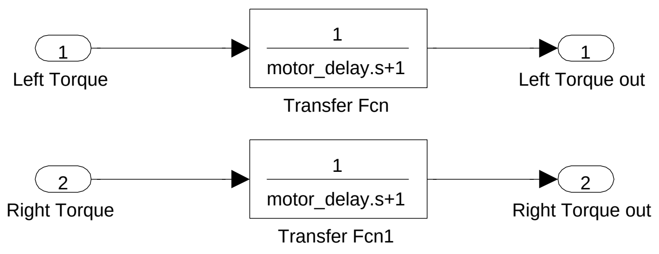


Figure 3.3: Simulink model of the motors

3.3.1 Energy consumption of the motors

The instant power output from the motors is calculated by

$$P(t) = \tau(t) \cdot \omega(t) \tag{3.37}$$

where

$\tau(t)$	=	Instantaneous torque for time t,
$\omega(t)$	=	Instantaneous angular speed for time t,
$P(t)$	=	Instantaneous power for time t.

When integrating instant power output over time, the total amount of energy consumed is obtained

$$W_{total} = \int_0^t P(t)\,\mathrm{d}t \tag{3.38}$$

This principle of calculating energy has been implemented in Simulink with the block diagram in Figure 3.4.

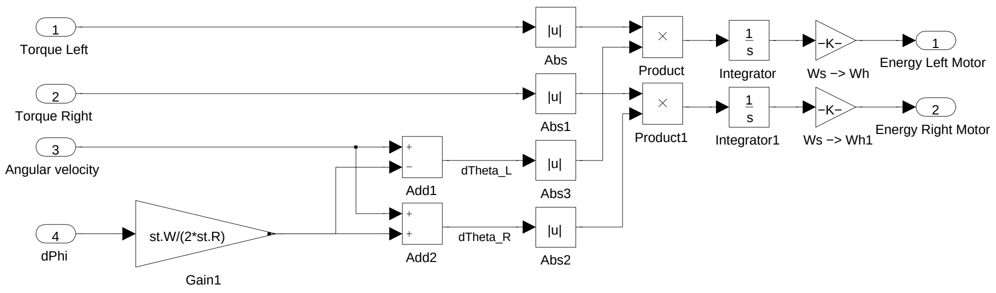


Figure 3.4: Motor energy calculation implemented in Simulink

3.4 Sensor models

Sensor models are implemented to better simulate real life conditions for the vehicle. In order to do that, some questions need to be answered:

- What states need to be measured and which sensors are needed to do that?
- What kind of noise do the sensors add?
- Which sensor resolution is required to achieve a good control and behavior of the vehicle?

3.4.1 Which states need to be measured?

This section describes which states that should be measured in order to control the vehicle. Note that some states could be estimated by the Kalman filter instead of measured. However, since there exists sensors that can accurately measure these states it is preferable to use them instead.

Pitch angle and pitch angle rate

The pitch angle ψ has to be measured in order to balance the vehicle in an upright position. There are different ways to measure ψ but the most common one is to use an accelerometer in combination with a gyroscope. This fusion of sensors is called an *Inertial sensor* and it will be able to measure both ψ and $\dot{\psi}$ with good resolution and small error.

Yaw angle rate

Since the vehicle controller should also be able to handle turning, the jaw angle rate $\dot{\phi}$ has to be measured. This can be measured with a simple gyroscope which is included in the Inertial sensor.

Speed

$\dot{\theta}$ has to be measured to enable control of the movement of the truck. The speed can easily be measured by using tachometers on the wheels. Since the tachometers are connected to the vehicle body, the position of the body has to be known to calculate the rotation of the wheels. Else, it would be impossible to say if it is the wheels that are rotating or if it is the body. Note that this will cause the errors from ψ measurement to also affect the speed measurement calculations.

Measured states

The four states that are being measured are (ψ, $\dot{\psi}$, $\dot{\phi}$ and $\dot{\theta}$). Measuring these four states enables balancing, speed and steering control of the vehicle.

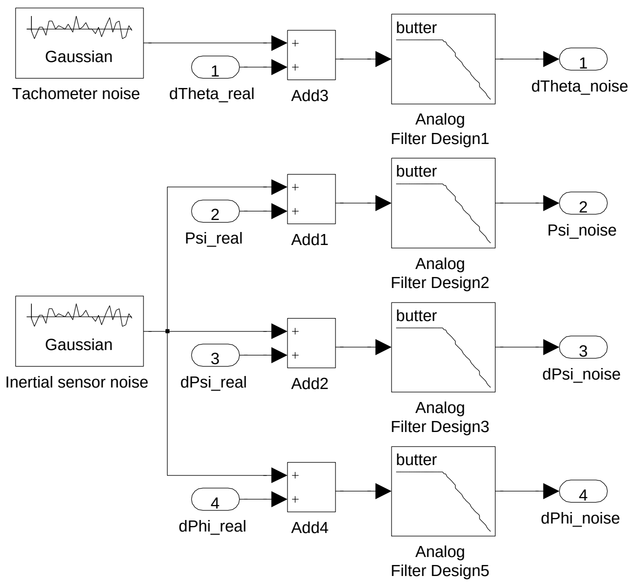


Figure 3.5: Overview of the sensor models

3.4.2 Noise modeling

Every state except the speed ($\dot{\theta}$) is measured by an inertial sensor. The inertial sensor will add noise to the measured states ψ, $\dot{\psi}$ and $\dot{\phi}$. There will also be quantization errors due to sensor resolution.

The noise level for the tachometer will be very small. But since the noise from the Inertial sensor also affects the speed measurement calculation, there will be a lot of noise from those readings as well. However, this correlation between the noises will not be accounted for in the design of the Kalman filter in Section 3.5.3. A Gaussian distributed noise with the same variance is therefore added to all the states in the sensor model. The variance chosen is 0.003, which is a common value for many of the Inertial sensor manufacturers.

3.4.3 Implementation

The sensor models are simply implemented in the Simulink model by creating a block ”Sensors” where the noise of each sensor is added to the measured state (see Figure 3.5). Anti-aliasing filter are also added to avoid aliasing effects when the measured signals are sampled.

3.5 Controller Design

The main task for the control system is to balance the vehicle in upright position and follow the reference signals. The Controller has two input reference signals. The first one is the requested speed of the vehicle $\dot{\theta}$, and the second one is the requested turning speed $\dot{\phi}$. The controller selected to perform these tasks is a linear LQG-controller. An LQG controller is basically an LQR controller linked with a Kalman filter (see Section 2.2.3).

3.5.1 Linearisation

The equations of motion are linearized around the upright operating point of the vehicle by applying $\theta \to 0$,$\psi \to 0$, $\phi \to 0$, $Z_g \to 0$ and $\tau \to 0$. Since linearizing by hand is a tedious task and it is also very easy to make mistakes, a Matlab script was developed to do that (see Appendix B.3). The resulting linearized system is present in a state space formulation. It is described by

$$\dot{x} = Ax + Bu, \tag{3.39}$$

$$y = Cx, \tag{3.40}$$

where x and u are considered to be states and inputs,

$$x = [\theta,\, \dot{\theta},\, \psi,\, \dot{\psi},\, \phi,\, \dot{\phi}]^T,\ u = [T_l,\, T_r], \tag{3.41}$$

and the system matrices are

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & a_{22} & a_{23} & a_{24} & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & a_{42} & a_{43} & a_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \tag{3.42}$$

$$B = \begin{bmatrix} 0 & 0 \\ b_2 & b_2 \\ 0 & 0 \\ b_4 & b_4 \\ 0 & 0 \\ -b_6 & b_6 \end{bmatrix}, \quad C = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.43}$$

where

$$a_{22} = \frac{-(2f_m + 2f_w)}{(E - \frac{(2J_m n^2 B)}{A} + \frac{(LMRB)}{A})}, \tag{3.44}$$

$$a_{23} = \frac{(MRg - \frac{(L^2M^2Rg)}{A} + \frac{(2J_m LMgn^2)}{A})}{(E - \frac{(2J_m n^2 B)}{A} + \frac{(LMRB)}{A})}, \tag{3.45}$$

$$a_{24} = \frac{2f_m + \frac{4J_m f_m n^2}{A} - \frac{2LMRf_m}{A}}{E - \frac{2J_m n^2 B}{A} + \frac{LMRB}{A}}, \tag{3.46}$$

$$a_{42} = -\frac{4(2J_m n^2 - LMR)(f_m + f_w)}{C + D}, \tag{3.47}$$

$$a_{43} = \frac{4Mg(LmR^2 + J_m Rn^2 + J_m Ln^2 + J_w L)}{C + D}, \tag{3.48}$$

$$a_{44} = \frac{4f_m(2J_w + MR^2 + 4J_m n^2 + 2R^2 m - LMR)}{C+D}, \tag{3.49}$$

$$b2 = \frac{n - \frac{2J_m n^3}{A} + \frac{LMRn}{A}}{E - \frac{2J_m n^2 B}{A} + \frac{LMRB}{A}}, \tag{3.50}$$

$$b4 = -\frac{4J_w n + 2MR^2 n + 4R^2 mn + 2LMRn}{C+D}, \tag{3.51}$$

$$b6 = \frac{Wn}{2R(J_\Phi + \frac{W^2 m}{2} + \frac{J_w W^2}{2R^2} + \frac{J_m W^2 n^2}{2R^2})}, \tag{3.52}$$

and A,B, C, D, E are given by

$$A = ML^2 + 2J_m n^2 + J_\Psi), \tag{3.53}$$

$$B = 2J_m n^2 - LMR, \tag{3.54}$$

$$C = n^2(4J_\Psi J_m + 8J_m J_w + 4J_m L^2 M + 4J_m MR^2 + 8J_m R^2 m + 8J_m LMR), \tag{3.55}$$

$$D = 4J_\Psi J_w + 4J_w L^2 M + 2J_\Psi MR^2 + 4J_\Psi R^2 m + 4L^2 MR^2 m, \tag{3.56}$$

$$E = 2J_w + MR^2 + 2J_m n^2 + 2R^2 m. \tag{3.57}$$

Poles and Zeros

By studying the poles of the linearized system the basic dynamics of the system can be observed. It is known that the inverted pendulum have two equilibrium points. One point is stable, which corresponds to when the pendulum is in a downright position. The other point is unstable and corresponds to when the pendulum is in an upright position. In Figure 3.6 the unstable and the stable poles for the reference model (see Table 3.1) can be observed. Note that the system also contains four other poles which are related to the other coordinates in the system.

The numerical values of the poles are −1.1991, −0.3526, 1.4930 and 0.

3.5.2 Discretization

It is assumed that the LQG-controller is implemented on a microprocessor (in reality). Therefore the entire system needs to be discretized. In Matlab this is easily done by running the command $SYSD = c2d(SYS,TS)$, where SYS is the continuous time representation of the system and TS is the sampling period. The sampling time is selected to $1ms$.

3.5.3 Kalman Filter

The Kalman filter is a part of the LQG-controller which filters noise and estimates unknown states (see Section 2.2.2).

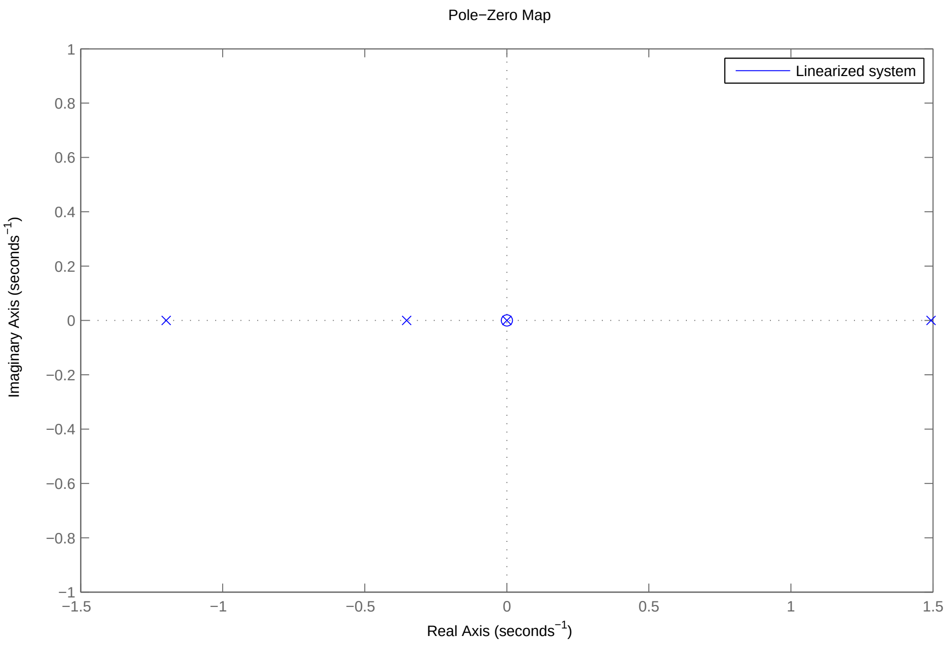


Figure 3.6: Poles and zeros of the linearized system

Observability

One necessary requirement for the Kalman filter to work is that the system is observable. This means that unmeasured states in the system need to be able to be estimated by an observer. To determine if the system is observable the rank of the *observability matrix* is calculated

$$\mathcal{O} = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix} \tag{3.58}$$

The rank of the observability matrix is calculated to 4, which is not full rank and therefore implies that the system is not fully observable. This means that some states cannot be determined through the sensor outputs and their value will be unknown for the controller.

By studying the properties of the system it can be seen that the non-observable states are θ and ϕ. Since these states are not necessary for controlling the vehicle, they can simply be discarded. Removing θ and ϕ results in a new state space representation

$$\dot{x} = A_{min}x + B_{min}u, \tag{3.59}$$

$$y = C_{min}x, \tag{3.60}$$

where the state vector x and the input vector u are

$$x = [\dot{\theta},\, \psi,\, \dot{\psi}, \dot{\phi}]^T,\ u = [M_l,\, M_r]. \tag{3.61}$$

The new system matrices are then

$$A_{min} = \begin{bmatrix} a_{22} & a_{23} & a_{24} & 0 \\ 0 & 0 & 1 & 0 \\ a_{42} & a_{43} & a_{44} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad B_{min} = \begin{bmatrix} b_2 & b_2 \\ 0 & 0 \\ b_4 & b_4 \\ -b_6 & b_6 \end{bmatrix} \tag{3.62}$$

$$C_{min} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.63}$$

Calculation of the Kalman gain

The Kalman gain K is calculated in Matlab by using the command $[KEST, K, P] = kalman(SYSD, R1, R2, R12)$. $SYSD$ is the discrete representation of the minimal linearized system and $R1$, $R2$ and $R12$ are design parameters (see Section 2.2.2).

The filter is tuned by adjusting $R1$, $R2$ and $R12$. $R12$ is the covariance between measurement noise and process noise. It is assumed that the measurement noise and process noise are independent and, thus, $R12 = 0$. $R2$ is the matrix that describes the variance of the measurement noise. Since the measurement noise is equal for all outputs and the noise is assumed to be independent from the other measurements (see Section 3.4.2), $R2$ set to a diagonal matrix with a value of 0.001. The parameters of $R1$ can now be tuned until the result is satisfying. Increasing the values in $R1$ means relying more on the measurements and less on the model. Decreasing $R1$ means that the estimate will depend more on the model and less on the measurements.

It is desirable to have a Kalman filter that removes as much noise as possible without being too slow to adapt to changes. That means that a compromise has to be found between noise reduction and system adaptation. After trial and error the tuning matrices were set to

$$R1 = \begin{bmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0.5 \end{bmatrix} \tag{3.64}$$

$$R2 = \begin{bmatrix} 0.001 & 0 & 0 & 0 \\ 0 & 0.001 & 0 & 0 \\ 0 & 0 & 0.001 & 0 \\ 0 & 0 & 0 & 0.001 \end{bmatrix} \tag{3.65}$$

$$R12 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{3.66}$$

Figure 3.7 illustrates how the Kalman filter was implemented in Simulink.

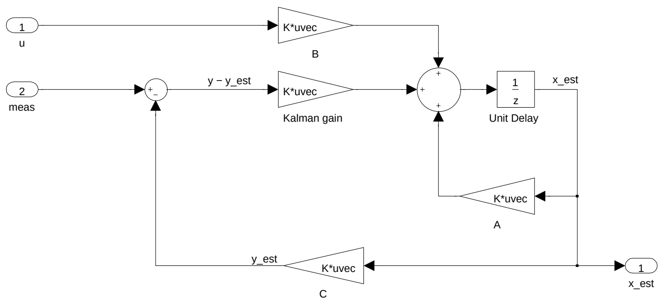


Figure 3.7: Implementation of the Kalman filter in Simulink

3.5.4 Controller

The Kalman filter feeds into a LQR controller with integral action (see Section 2.2.1). The integral action is necessary for good reference tracking when running or standing still on a slope. The optimal gain matrix is calculated in Matlab by using the command *[K;S;e] = lqi(SYSD;Q;R;N)*, where $SYSD$ is the discretized and linearized minimal realization (see Equations 3.62 and 3.63) and Q, R and N are the design parameters. To simplify N is set to 0. R is set to a 2x2 unit matrix since the two control signals are scaled equally. Q is the weighting matrix for each state in the state vector $[\dot{\theta}, \psi, \dot{\psi}, \dot{\phi}, \theta]$. The selected matrices are

$$Q = \begin{bmatrix} 2e4 & 0 & 0 & 0 & 0 \\ 0 & 1e7 & 0 & 0 & 0 \\ 0 & 0 & 1e-7 & 0 & 0 \\ 0 & 0 & 0 & 1e4 & 0 \\ 0 & 0 & 0 & 0 & 1e2 \end{bmatrix}, \qquad R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \qquad N = 0. \tag{3.67}$$

The optimal feedback gain is calculated to

$$K = 10^3 \cdot \begin{bmatrix} -0.1073 & -2.9260 & -0.6633 & -0.0707 & 0.0070 \\ -0.1073 & -2.9260 & -0.6633 & 0.0707 & 0.0070 \end{bmatrix} \tag{3.68}$$

Figure 3.8 illustrates how the controller is implemented in Matlab.

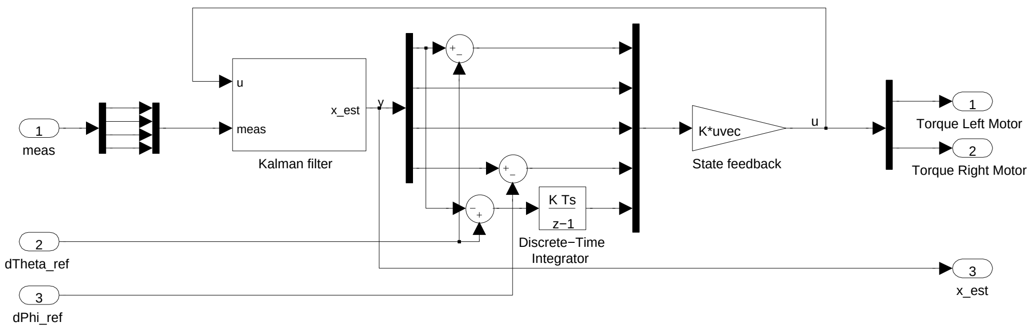


Figure 3.8: Implementation of the LQG controller in Simulink

3.6 Visualization

A visualization model is created to make it easier to observe and actually see how the vehicle behaves in different scenarios. The visualization will also give a good understanding for the dynamics in the vehicle without having any underlying knowledge about the system. It is therefore suitable to use as a presentation model for people that are not involved in this project.

3.6.1 Building the model

A VRML model (Virtual Reality Modeling Language) is built in the editor V-Realm Builder in order to represent the truck. The visualization is created within the Matlab *Simulink 3D Animation* toolbox [11]. The vehicle is only consisting of three objects. Two cylinders are used as wheels and one box represents the chassis (see Figure 3.9).

3.6.2 Implementation in Simulink

The visualization is implemented in Simulink as a separate model (see the ”3D Visualisation” block in Figure 3.11). Inputs to the block are the signals that are necessary to calculate position and rotation of the objects in the VRML model. Figure 3.10 illustrates how this was implemented in Simulink.

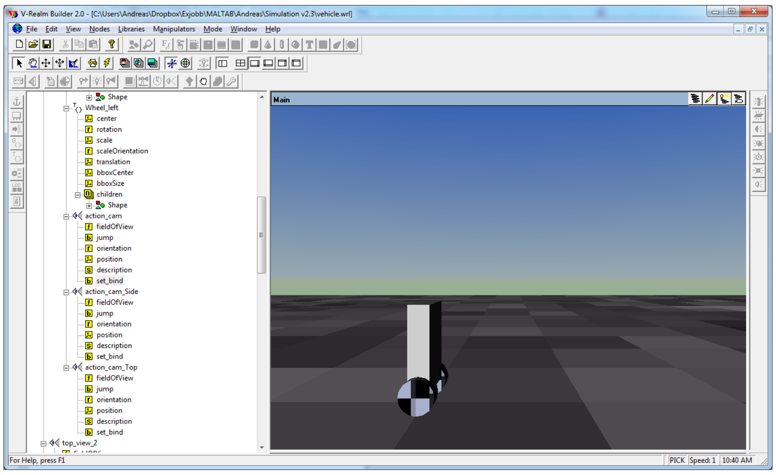


Figure 3.9: VRML model of the truck in V-Realm

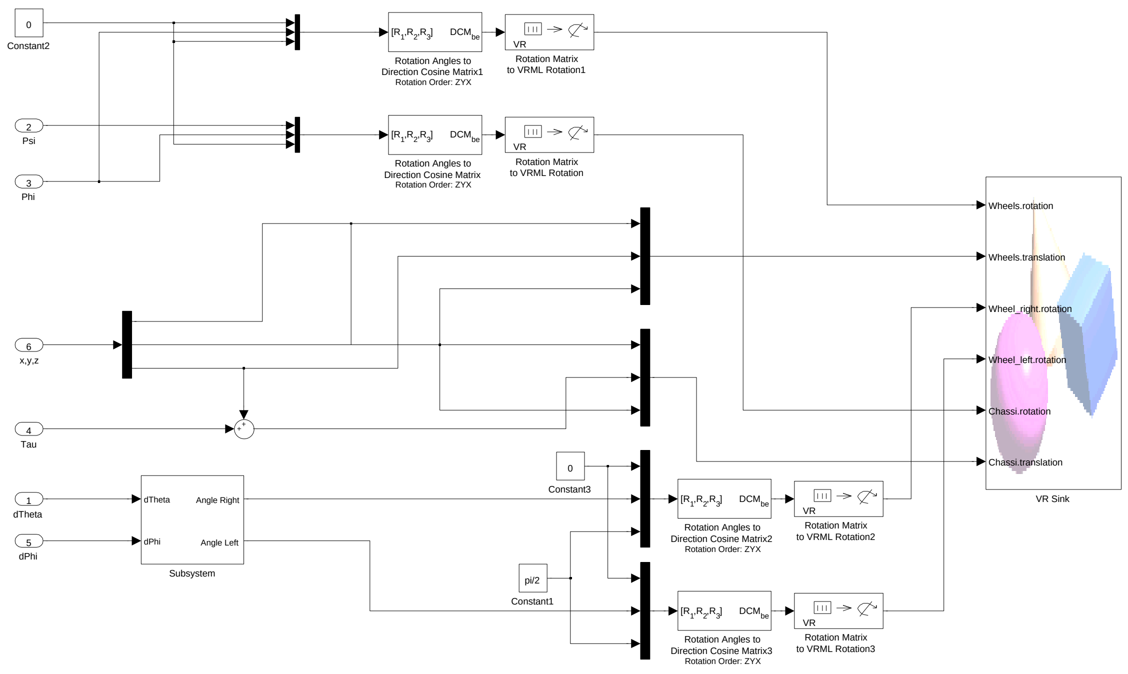


Figure 3.10: Visualization blocks

3.7 System Summary

Every part of the system is implemented in a Simulink model as a separate block (see Figure 3.11). This modularity makes it simple to make adjustments to or to replace any block. It also gives a good overview of all the parts included in the model. To tie all parts together a data bus is created. A data bus is a good way to keep track of all the signals included in the system and makes it easier to work with the model.

All subsystems in the model are operating in continuous time except the LQG-controller block (see Figure 3.11). Therefore A/D and D/A converters are included in the model to handle the time difference between these two systems.

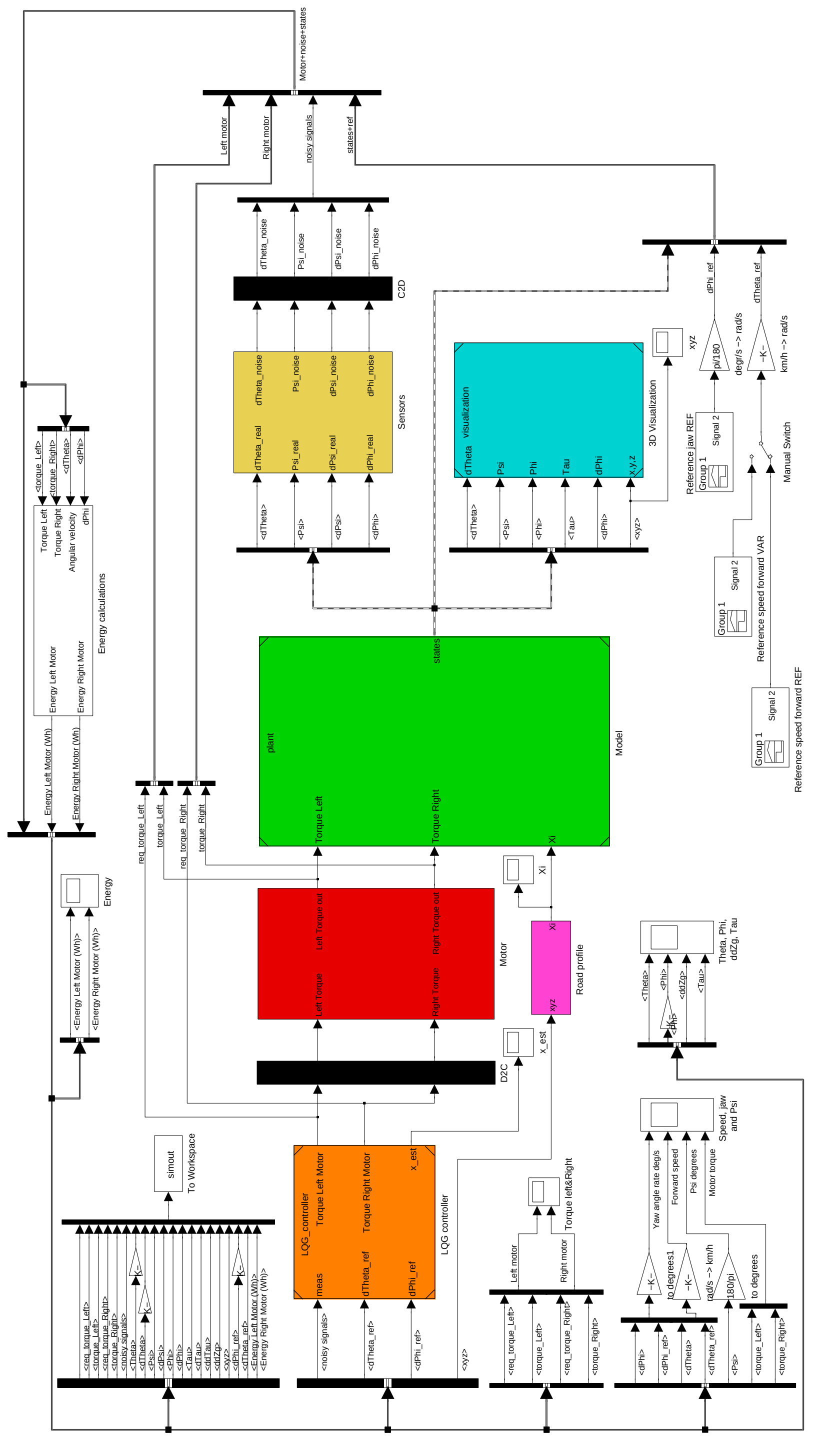


Figure 3.11: Overview of the Simulink model

4

Results

A parameter study was made by varying noise levels, weight, center of mass, sampling time and motor time constant of the truck model in Simulink. Simulations were done with the truck driving with different speeds, on slopes and over curbs. All results from those simulations are presented under different sections in this chapter. Every time a parameter was changed a new linearized model was derived with a script in Matlab. This new linearized model was then used in the LQG controller to stabilize and control the vehicle. Unless otherwise specified every simulation was done this way.

4.1 Torque requirement

A number of simulations have been carried out to show how much torque that is required to perform several actions.

4.1.1 Moving forward at constant speeds

These simulations show how much torque is required to maintain different speeds. Due to air drag and frictions in the wheels, higher torque and body angle is needed to maintain higher speeds (see Table 4.1).

4.1.2 Turning at different rates

The truck is standing still and turning left until it reaches either 90° or 180° (see Figure 4.1). There is no horizontal movement except for the small corrections back and forward to balance the truck. Whenever the truck turns, one motor will give a torque in the opposite direction of the other motor. For example if the truck turns right while standing still, the left motor will give a positive torque while the right motor torque will be negative. Note that the sensor noise is disabled in these simulations.

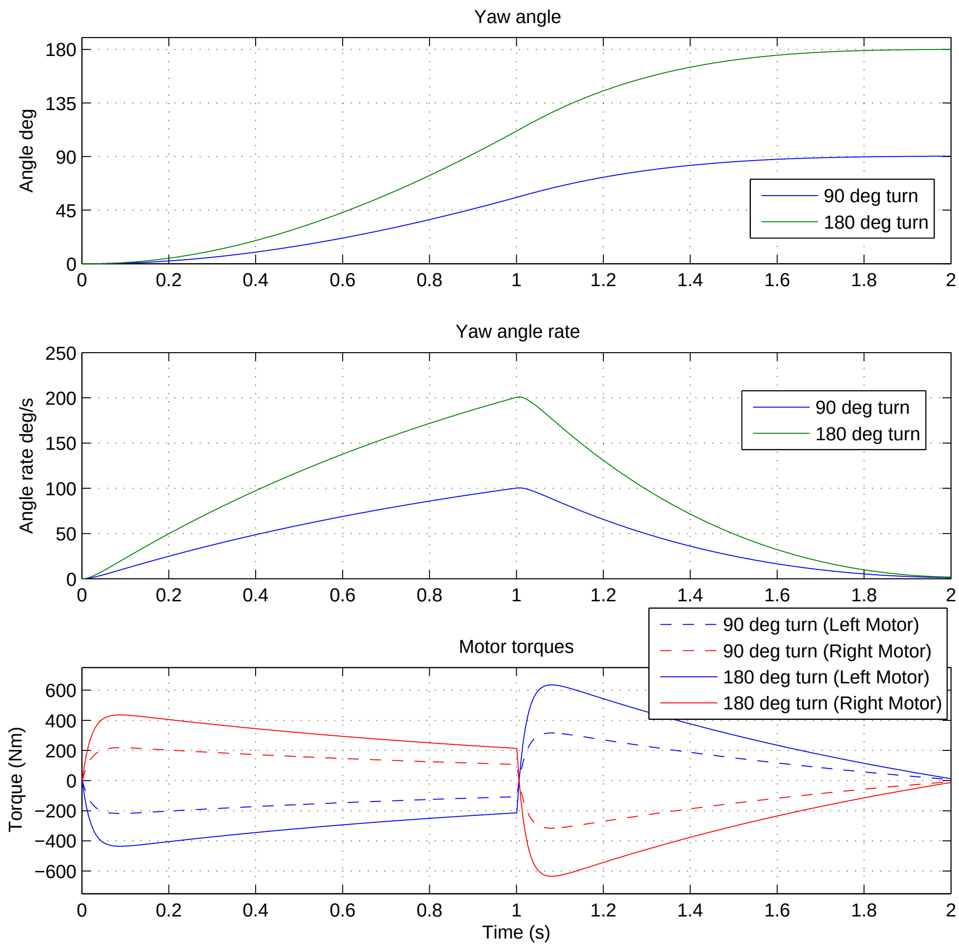


Figure 4.1: Turning 90° and 180° while standing still.

Table 4.1: Torque requirement during constant speeds

Speed (km/h)	**Total torque from both motors** (Nm)	**Body angle** $(degrees)$
10	14,4	0,24°
20	31,0	0,76°
30	47,0	1,5°
40	63,8	2,5°
50	81,2	3,8°
60	99,4	5,3°

4.1.3 Standing still on a slope

In many real life scenarios a vehicle will have to stop on a slope. A four wheeled vehicle can simply brake to avoid rolling backwards while the single axle truck has to continuously produce torque to balance upright while standing still. If one would try only to brake, the one axle truck would simply fall over. The results in Table 4.2 shows that steeper slopes require more torque for the vehicle to stand still. Sensor noise is disabled in these simulations.

Table 4.2: Torque requirement when standing still on a slope

Slope grade	**Torque required** (Nm)
3°(3,33%)	92
5°(5,56%)	153
7°(7,78%)	214
9°(10%)	275
12°(13,3%)	365

4.2 Disturbances

Process disturbances will be simulated by driving in a slope and over a curb. Measurement disturbances will be observed by varying the sensor noise in the sensor models.

4.2.1 Effects from process disturbances

Driving over a curb

A common scenario that may occur in real life is that the vehicle has to drive over a curb. To examine how the vehicle will behave in such a scenario simulations were done by running the vehicle onto an edge at three different speeds (10 km/h, 20 km/h and

30 km/h). The edge has a height of 10 cm (see Figure 4.2). It was implemented as a small slope with a big gradient but small traveling distance.

The simulations show that the controller has no problem to maintain stability of the vehicle. However, hitting the curb will give rapid change in the pitch angle of the body. This rapid change of Ψ will be measured by the sensors and the controller will compensate for this by quickly requesting a high torque from the motors. This is seen as high torque peaks in the second graph in Figure 4.2.

The third graph in Figure 4.2 shows the behavior of the suspension system. It can be seen how the suspension is compressed when the vehicle hits the edge and how it is extended when driving off the edge. The dotted line in Graph 3 is not in scale but just implemented as a guide line to see when the vehicle hits the curb.

Slope

This simulation was performed to observe how the vehicle would handle driving up a slope as well as stand still on a slope. The vehicle is accelerated to a speed of 25 km/h and after 6 seconds it hits a slope with a gradient of 9° (10%). The vehicle then stops on the slope before it start accelerating again (see Figure 4.3).

The simulation shows that the vehicle has no problem to maintain stability while driving on the slope. It is also shown how the vehicle gets a constant torque applied to the wheels and the body gets a constant angle while standing still on the slope (earlier discussed in Section 4.1.3).

4.2.2 Effects from measurement disturbances

In this simulation gaussian white noise with different variances are added to the measured signals as explained in Section 3.4. Even with the highest noise variance (0,3) the truck will perform similarly as with almost no noise at all (0,0003) (see Figure 4.4). For the variance values (0,03), (0,003) and (0,0003) there are almost no difference in the performance.

Torque fluctuation with noisy signals

The performance of the truck might still be good with higher sensor noise. However, as Figure 4.5 illustrates, the torque required to balance and move the vehicle fluctuates more with noisier measurements.

These simulations were done with the truck standing still and balancing upright. There is no movement of the truck besides the small corrections back and forward in order to balance. Each simulation took 30 s from start to finish.

Table 4.3 shows that there is a linear relation between sensor noise and the amount of energy required to balance which is natural since the system it self is linearized. More energy is required because the motor torques will fluctuate much more while basically doing the same job (standing upright).

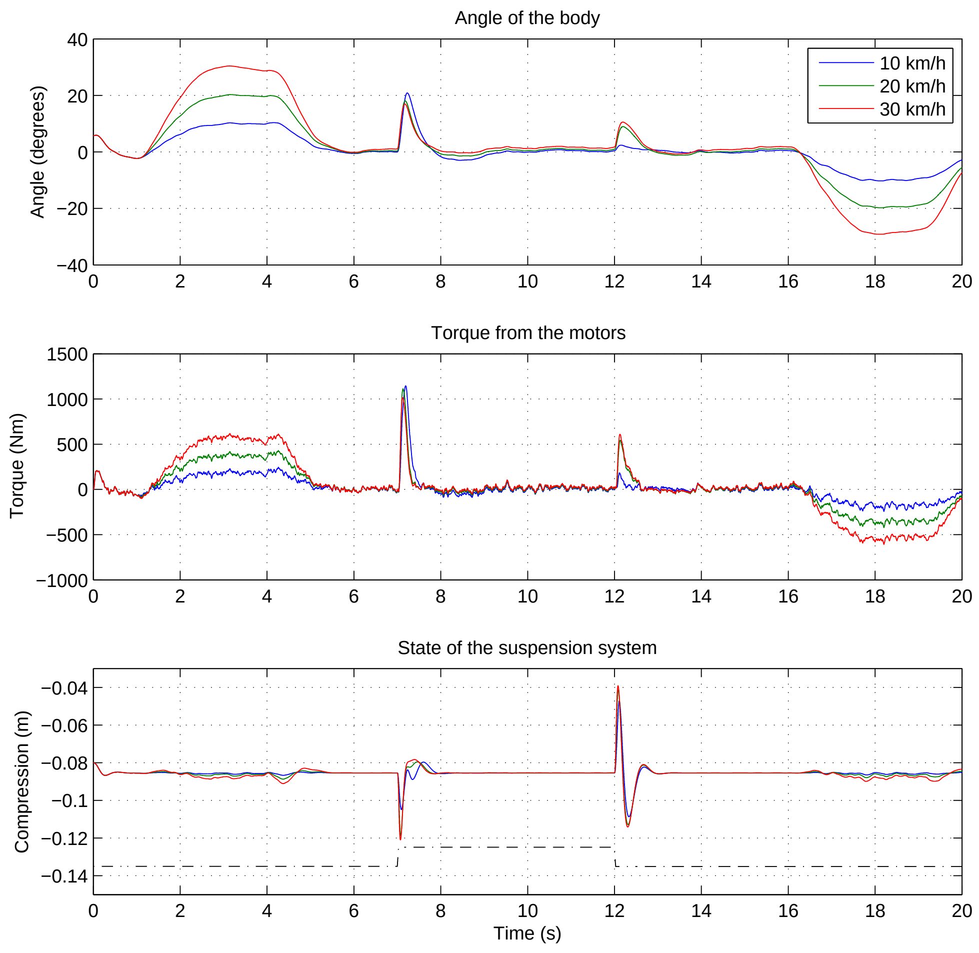


Figure 4.2: Running over a curb

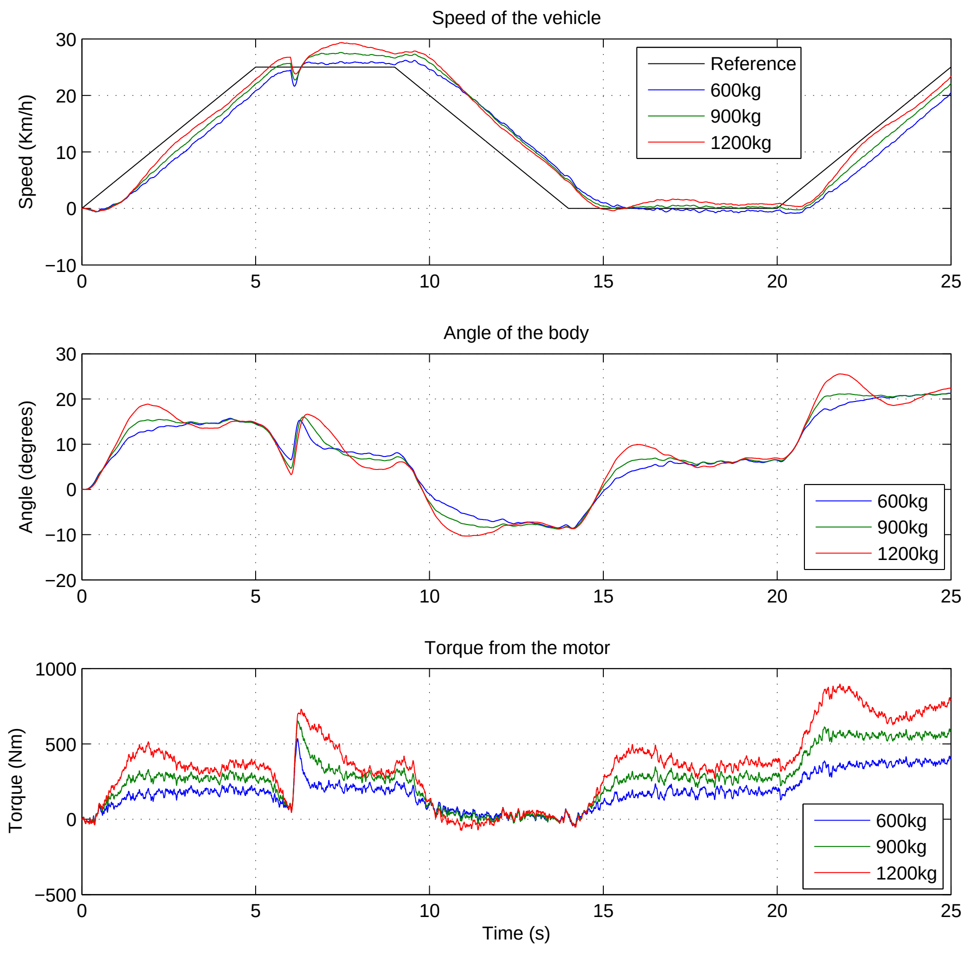


Figure 4.3: Running on a slope

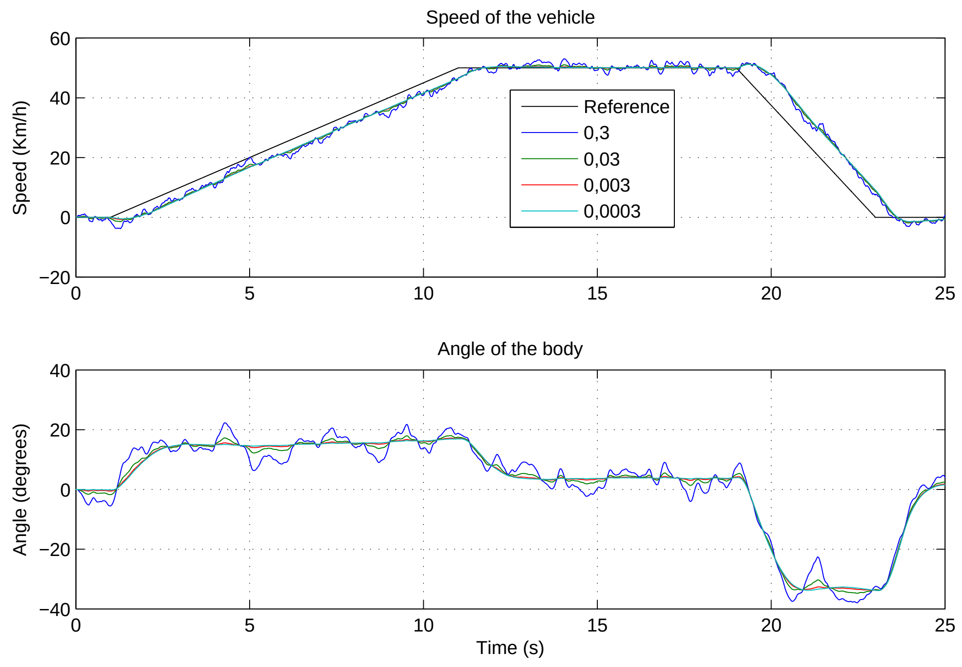


Figure 4.4: Influence of three different noise variance values for the measurement sensors.

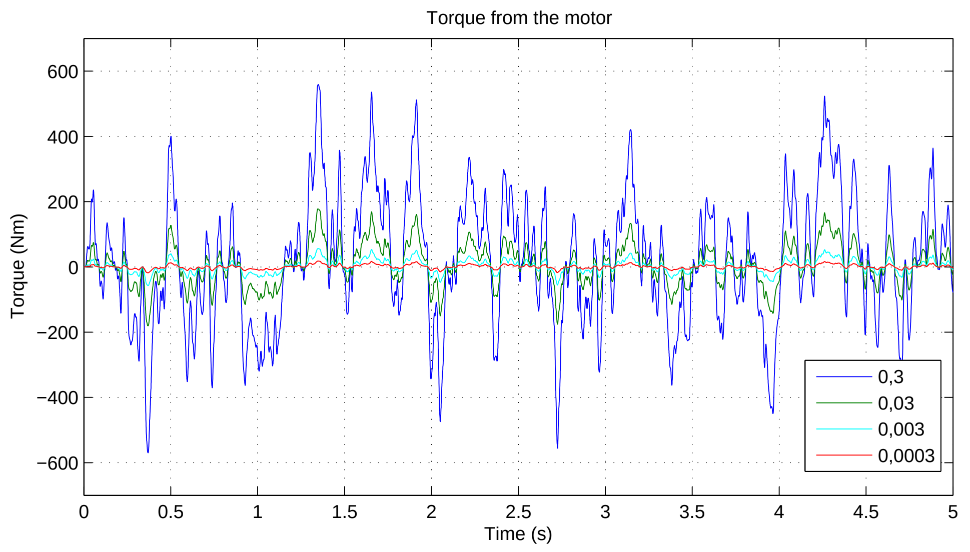


Figure 4.5: Sensor noise influencing torque output from the motor.

Table 4.3: Energy requirement for different sensor noise to balance the truck upright while standing still.

Sensor variance	**Energy consumed** (Wh)
0,3	0,982
0,03	0,0982
0,003	0,00983
0,0003	0,000983

4.3 Vary the mass of the vehicle

Since the vehicle is supposed to be used as a distribution vehicle, the weight may change during operation. A related question is if the control system can handle a varying weight or if the control algorithm has to be updated with a new linearized model (with different parameters) during operation. In this simulation the weight of the vehicle will be changed but the controller will remain the same. The controller used in these simulations will be tuned for 900kg and the mass will be deviating with $\pm 150kg$ and $\pm 300kg$.

The simulations shows that the behavior is affected by a deviating weight (see Figure 4.6). Decreasing the weight will give a slower system response to changes at the reference signal. However, the vehicle will remain stable and the body angle will deviate less than in the reference model.

Increasing the weight will give a bigger response to changes in the reference signal. This leads to poorer performance, with larger fluctuations of the body angle which results in overshoots and instability (see Figure 4.6).

4.4 Vary the distance to the center of mass of the vehicle

Changing the center of mass vertically means that the dynamics of the vehicle also changes. It is possible to vary the distance to the center of mass by changing parameter L in Table 3.1. Figure 4.7 illustrates how the distance is changed.

4.4.1 Accelerating and then driving up a slope

At time $t = 15$ the truck starts to climb a 9% incline after accelerating to a speed of 50 km/h. Figure 4.8 shows that with a lower center of mass a higher angle of body is needed in order to accelerate and maintain speed. In the case of the smallest distance $0{,}4 \cdot ref$, the truck could not even reach 50 km/h. Furthermore, when the truck hits the slope at $t = 15$ the trucks with lower center of mass can tilt the chassis faster in order to absorb the hit. Hence, there is a clear relation between the height of the center of mass and the performance of the truck.

Figure 4.9 shows that the poles of systems with lower center of mass tend to move further to the right and to the left. The pole on the right side of the map is the unstable

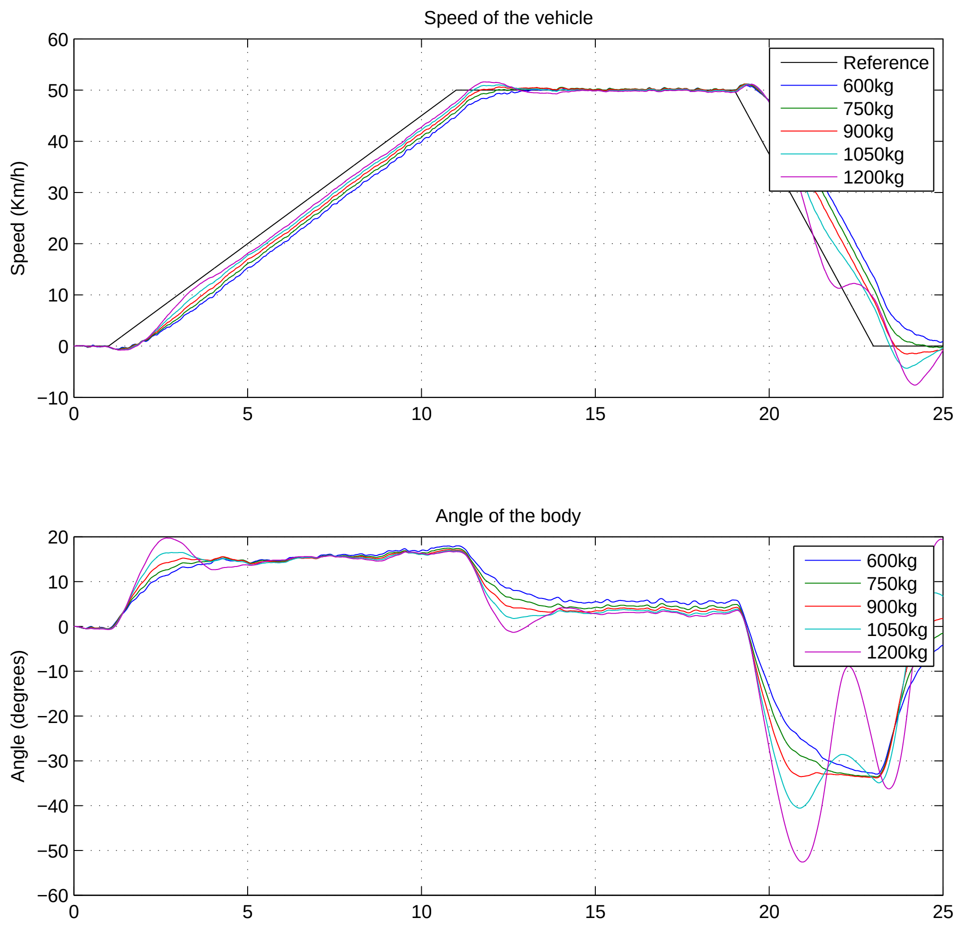


Figure 4.6: Varying the mass

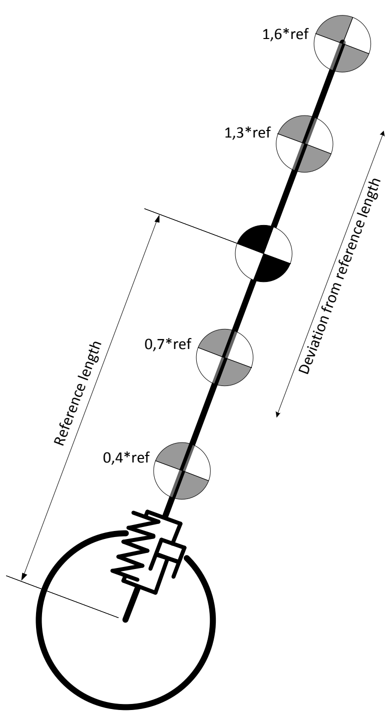


Figure 4.7: Deviating the position of center of mass.

pole (upright position) that has to be stabilized with the controller. The left side pole is the stable pole (downright position). When these two poles move closer to ∞ and $-\infty$ it means that the system becomes faster. In reality it means that when going from acceleration to retardation the truck with lower center of mass will tilt faster to the other side. The downside is that it will be harder to control and requires more body angle leaning for movement

4.4.2 How weight affects the center of mass

The distance to the center of gravity is also varying with how much the suspension is compressed on the truck. For example, if the truck is loaded with a weight then the suspension will lower the center of gravity by a couple of centimeters. Also if the road is bumpy then the suspension will vary. However, having the same controller for small deviation of the position of center of gravity gives the same performance and it is not necessary to change controller (derive a new linearized model in the LQG controller). Note that this only applies for small deviations of at most a few decimeters. Any larger deviation makes the system unstable and a new linearized model is needed in the controller.

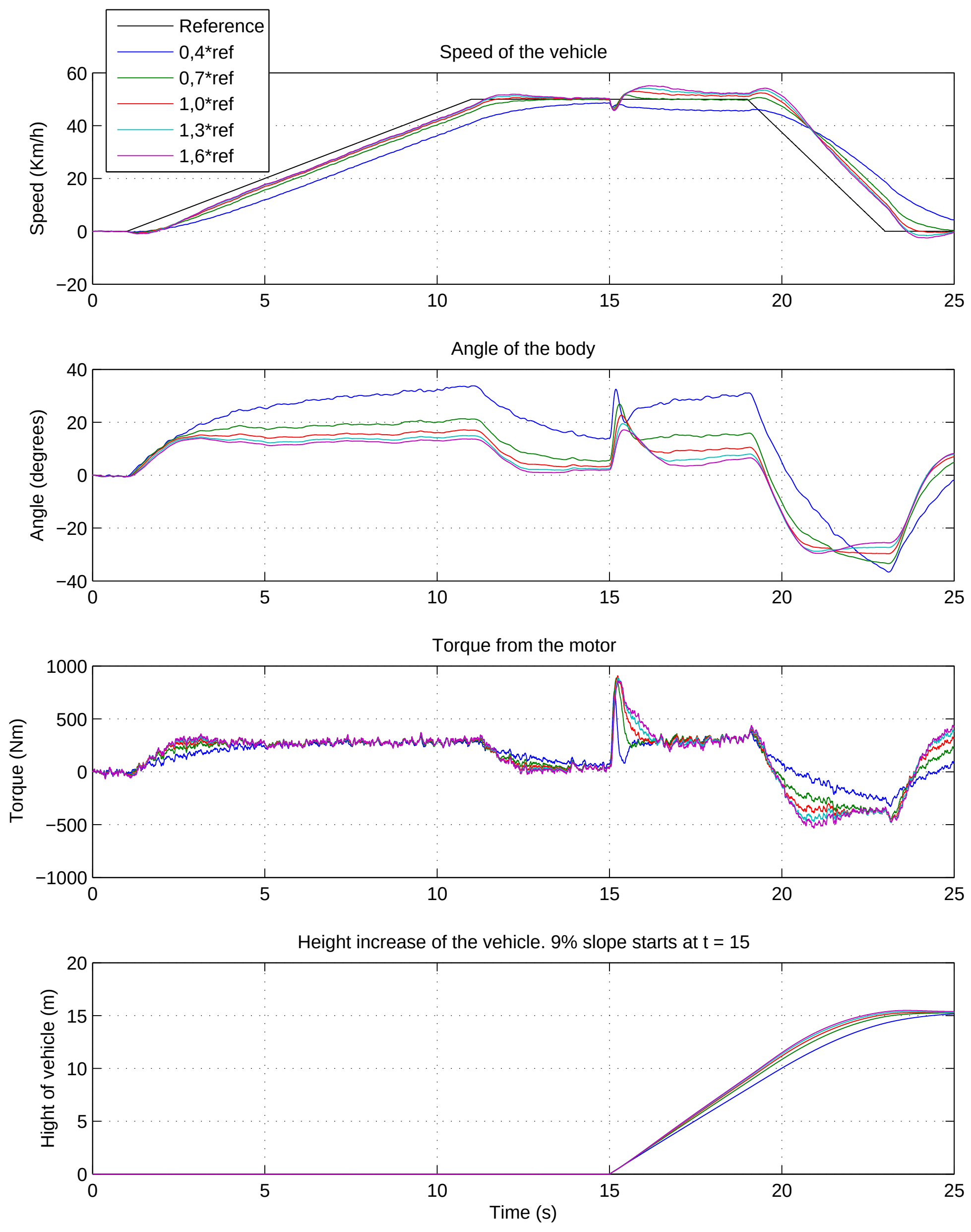


Figure 4.8: Influence of three different noise variance values for the measurement sensors.

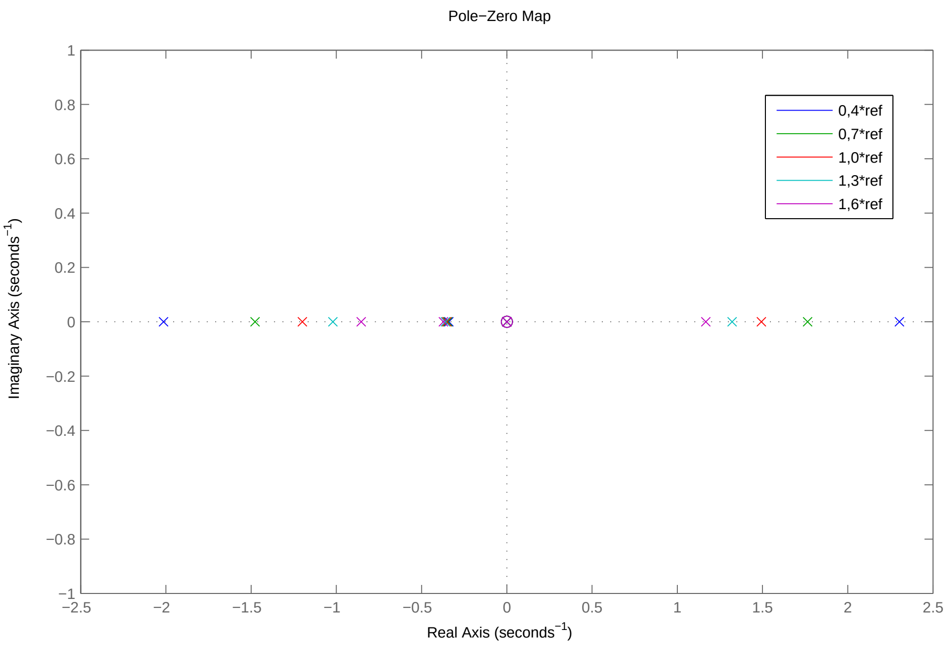


Figure 4.9: Sensor noise influencing torque output from the motor for four different sensor variance values

4.5 Influence of controller update frequency

Several simulations were made to investigate the effects of different update frequencies of the controller. In this simulation the reference vehicle tries to follow a specific track while the sampling rate of the controller and the sensors (T_{ctrl}) were varied. Note that the linearized model in the controller stays the same, it is only the update frequency that is varied.

Update rates simulated are T_{ctrl} = 1, 10, 20, 30, 40 and 50 ms and the results are shown in Figure 4.10. Figure 4.11 is a zoomed version of the same plot.

The figures show that the update frequency of the controller and the sampling time of the sensors have a huge impact on the system. Already at an update rate of 10 ms will result in worse performance in terms of balancing. An update rate of 40 ms or above results in increasing fluctuations which cause the system to fail.

4.6 Motor delay

In this simulation the effects of the time it takes before a requested torque is applied to the wheels are investigated. As described in Section 3.3, the torque response is approximated with a first order transfer function. By varying the time constant of the transfer function, the response time of the motor can be changed. Response times that

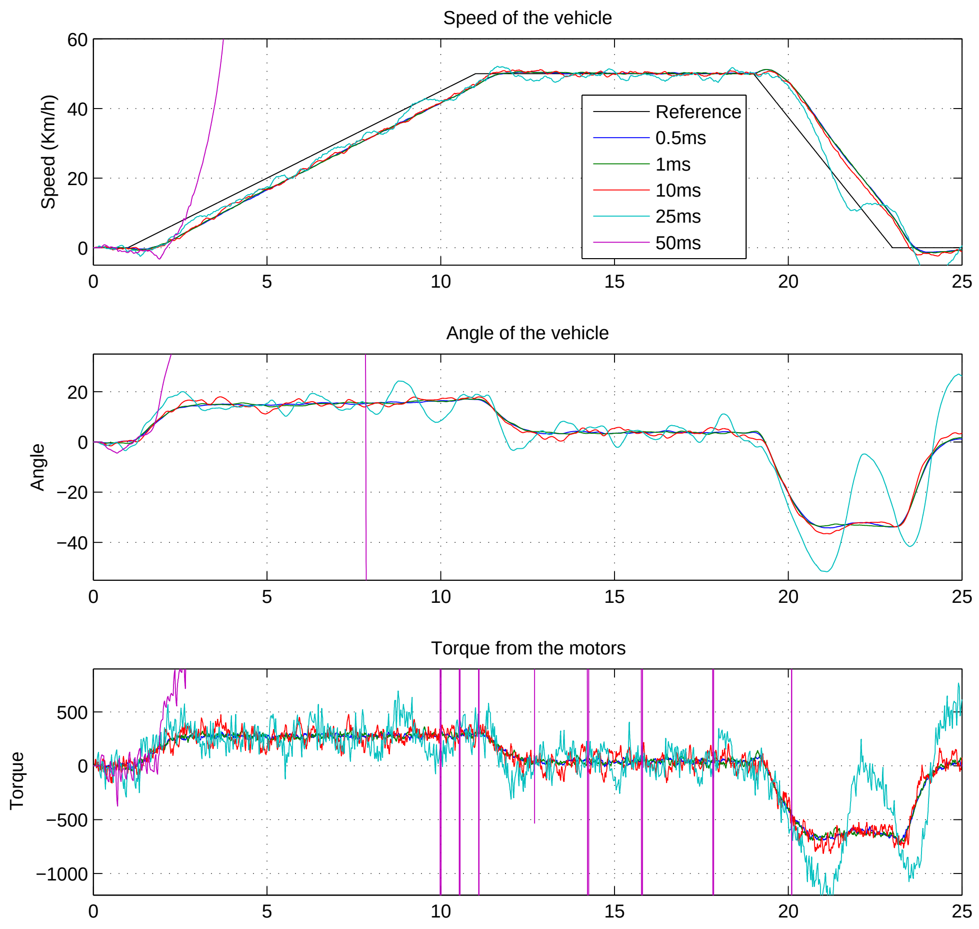


Figure 4.10: Influence of controller update frequency

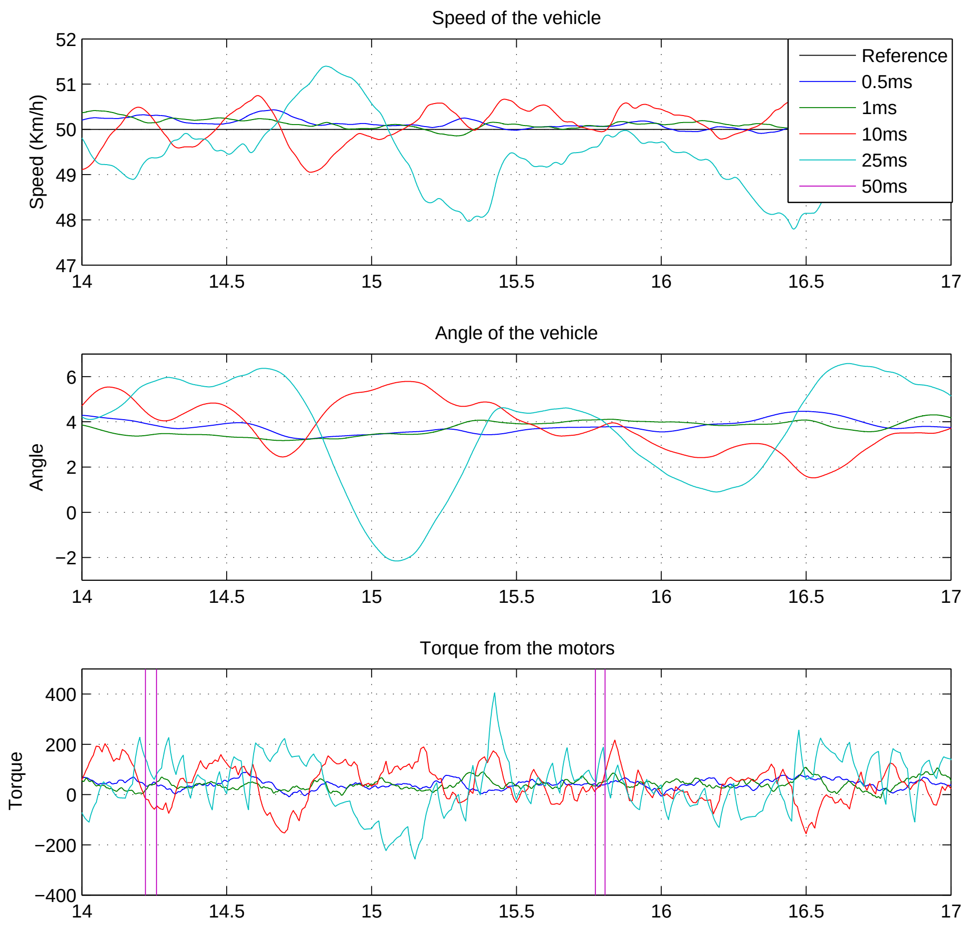


Figure 4.11: Influence of controller update frequency

have been simulated are T_{motor} = 10, 20, 50, 100 and 125 tms.

The simulations show that the time constant of the motor dynamics is very important to the system performance. In Figure 4.12 it can be seen that a time constant of 10, 20 and 50 ms will give a similar performance, while 100 ms and especially 125 ms will cause instability of the vehicle.

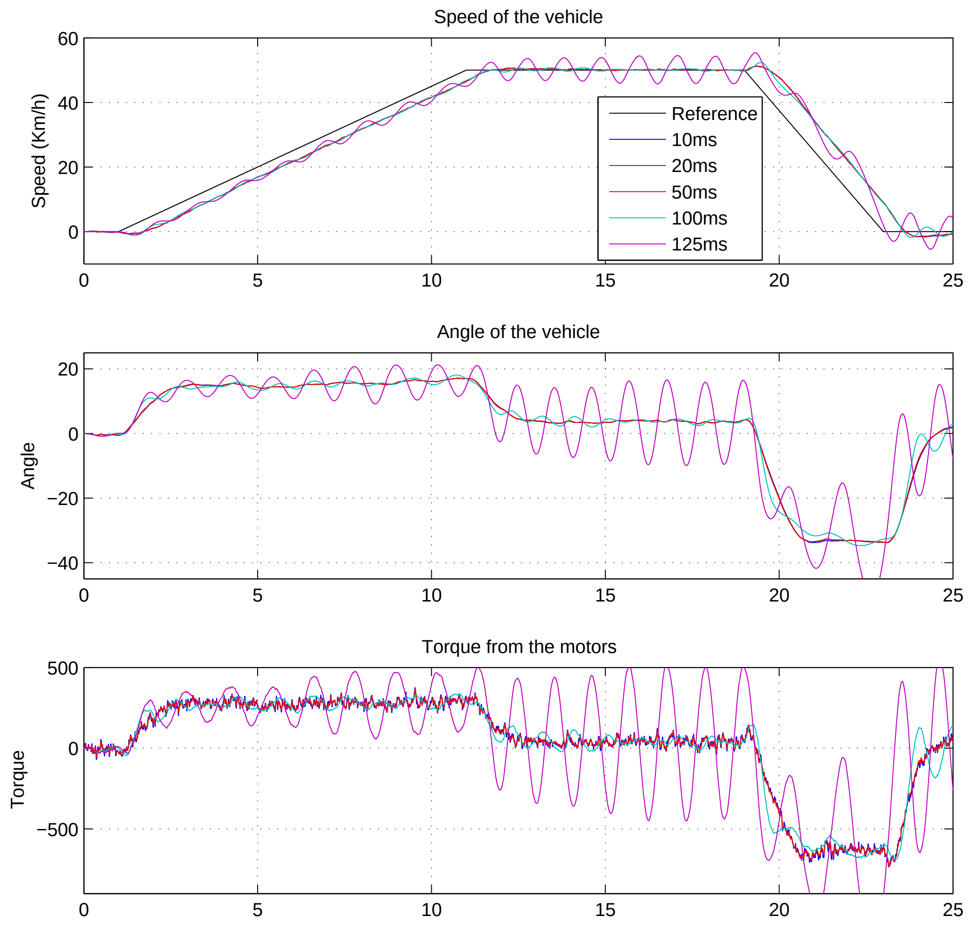


Figure 4.12: Motor delay

5

Discussion and Conclusion

Simulations in Chapter 4 show that as long as the wheels have traction and enough torque the single axle concept is working well in terms of controllability and robustness. However, large noise and long system delays impair the performance and might even make the truck fall over. It is also shown that small deviations of the center of mass have a very limited affect to the performance of the system. Therefore, the small deviations caused by the suspension system can be neglected from a control point of view. Practically that means that it is not necessary to include the dynamics of τ in the linearized model used when designing the LQG-controller.

5.1 Achieved goals

The goals listed can be found in Section 1.2. A model of the truck was successfully implemented. The controller was able to balance the truck in upright position and also move it around. Sensor and motor models were implemented and they performed as expected. However, the motor model was quite simplified. The parameter study showed that the system could handle common scenarios such as driving on a slope and over a curb, as long as is has enough torque. Since the model does not include slip it is hard to answer the question if it is a viable option to use in reality. Outside a controlled environment the road traction may vary a lot which may lead to problems. For example, if it starts to snow or rain the traction may become poor with serious consequences when driving the vehicle. The truck can definitely be built in reality since all critical components are available today (motor, sensors, battery). The cost of building the truck has not been considered.

5.2 Safety

The safety regarding the usage of this type of vehicle has not been regarded at all. Everything is assumed to work at all times and no simulations were done to show what happens when the truck crashes. However, it is an important topic to discuss and think about before building a vehicle of this type.

5.2.1 Hardware failure

A vehicle of this type is particularly sensitive to hardware failures since it is not in a stable position when operating. Thus, if a sensor stops working the vehicle might fall over. It should also be investigated if any kind of emergency brake system could be implemented. One solution could be to have support wheels that are lowered down and brake the vehicle in case of an emergency.

5.3 Further improvements

To achieve a better and more accurate model of the vehicle some improvements need to be done.

5.3.1 Include slip

Since traction is critical to the behavior of the vehicle it is important that slip is included in the model. With a model of the slip, improvements can be done in order to control and avoid slip. Since the slip is highly related to the torque applied by the motors, it will limit the maneuverability of the vehicle and make it harder to control.

5.3.2 Improve and add new hardware models

The hardware models that are implemented in the model are based on simplifications and approximations. Since delays in the propulsion system is critical to the stability of the vehicle more advanced motor model would be preferable. The new motor models should also include delays in gearboxes etc.

Bibliography

[1] Brussels wants no oil-fuelled cars in cities by 2050, [ONLINE: 25 10 2011.] [CITED: 25 10 2011.] www.euractiv.com/transport/brussels-wants-oil-fuelled-cars-cities-2050-news-503404 (March 2011).

[2] Lagrangian mechanics, [ONLINE: 08 11 2011.] [CITED: 08 11 2011.] http://en.wikipedia.org/wiki/Lagrangian_mechanics (November 2011).

[3] M. Levi, The mathematical mechanic: using physical reasoning to solve problems, Princeton University Press, 2009.

[4] BOSCH, Automotive Handbook, 5th Edition, Robert Bosch GmbH, 2000.

[5] W. S. Levine, The Control Handbook, 1st Edition, CRC-Press, 1996.

[6] Linear-quadratic regulator, [ONLINE: 02 02 2012.] [CITED: 02 02 2012.] http://en.wikipedia.org/wiki/Linear-quadratic_regulator (February 2012).

[7] L. L. T. Glad, Control Theory - Multivariable and nonlinear methods, Taylor and Francis, 2000.

[8] B. W. K. J. Åström, Computer Controlled Systems, 3rd Edition, Prentice Hall, 1997.

[9] Y. Yamamoto, Nxtway-gs model-based design - control of self-balancing two-wheeled robot built with lego mindstorms nxt (May 2009).

[10] Conservative forces, [ONLINE: 03 28 2012.] [CITED: 03 28 2012.] http://en.wikipedia.org/wiki/Conservative_force (March 2012).

[11] 3d simulation toolbox, [ONLINE: 03 18 2012.] [CITED: 03 18 2012.] http://www.mathworks.se/products/3d-animation/ (March 2012).

[12] J. Brody, P. Yager, R. Goldstein, R. Austin, Biotechnology at low Reynolds numbers, Biophysical Journal 71 (6) (1996) 3430–3441.
URL http://linkinghub.elsevier.com/retrieve/pii/S0006349596795383

A

Equations of motion

The equations of motion for the generalized coordinates.

Θ - coordinate:
$\ddot{\Theta} = (F_\Theta - Rsin(\Psi)(F_\tau - \tau k + LM\dot{\Psi}^2 + M\tau\dot{\Psi}^2 - M\ddot{Z}_g cos(\Psi) - Mgcos(\Psi) + LM\dot{\Phi}^2 sin(\Psi)^2 +$
$M\tau\dot{\Phi}^2 sin(\Psi)^2) + (2J_m n^2(F_\Psi + LM\ddot{Z}_g sin(\Psi) + LMgsin(\Psi) + M\tau\ddot{Z}_g sin(\Psi) + M\tau gsin(\Psi) +$
$(L^2M\dot{\Phi}^2 sin(2\Psi))/2 + (M\tau^2\dot{\Phi}^2 sin(2\Psi))/2 - 2LM\dot{\Psi}\dot{\tau} - 2M\tau\dot{\Psi}\dot{\tau} + LM\tau\dot{\Phi}^2 sin(2\Psi)))/(ML^2 +$
$2ML\tau + M\tau^2 + 2J_m n^2 + J_\Psi) - 2MR\dot{\Psi}\dot{\tau}cos(\Psi) + LMR\dot{\Psi}^2 sin(\Psi) + MR\tau\dot{\Psi}^2 sin(\Psi) -$
$(LMRcos(\Psi)(F_\Psi + LM\ddot{Z}_g sin(\Psi) + LMgsin(\Psi) + M\tau\ddot{Z}_g sin(\Psi) + M\tau gsin(\Psi) + (L^2M\dot{\Phi}^2 sin(2\Psi))/2 +$
$(M\tau^2\dot{\Phi}^2 sin(2\Psi))/2 - 2LM\dot{\Psi}\dot{\tau} - 2M\tau\dot{\Psi}\dot{\tau} + LM\tau\dot{\Phi}^2 sin(2\Psi)))/(ML^2 + 2ML\tau + M\tau^2 +$
$2J_m n^2 + J_\Psi) - (MR\tau cos(\Psi)(F_\Psi + LM\ddot{Z}_g sin(\Psi) + LMgsin(\Psi) + M\tau\ddot{Z}_g sin(\Psi) + M\tau gsin(\Psi) +$
$(L^2M\dot{\Phi}^2 sin(2\Psi))/2 + (M\tau^2\dot{\Phi}^2 sin(2\Psi))/2 - 2LM\dot{\Psi}\dot{\tau} - 2M\tau\dot{\Psi}\dot{\tau} + LM\tau\dot{\Phi}^2 sin(2\Psi)))/(ML^2 +$
$2ML\tau + M\tau^2 + 2J_m n^2 + J_\Psi))/(2J_w + MR^2 + 2J_m n^2 + 2R^2 m - MR^2 sin(\Psi)^2 + (2J_m n^2(-2J_m n^2 +$
$MR\tau cos(\Psi) + LMRcos(\Psi)))/(ML^2 + 2ML\tau + M\tau^2 + 2J_m n^2 + J_\Psi) - (LMRcos(\Psi)(-2J_m n^2 +$
$MR\tau cos(\Psi) + LMRcos(\Psi)))/(ML^2 + 2ML\tau + M\tau^2 + 2J_m n^2 + J_\Psi) - (MR\tau cos(\Psi)(-2J_m n^2 +$
$MR\tau cos(\Psi) + LMRcos(\Psi)))/(ML^2 + 2ML\tau + M\tau^2 + 2J_m n^2 + J_\Psi))$

Φ - coordinate:
$\ddot{\Phi} = (F_\Phi - (M\dot{\Phi}\dot{\Psi}sin(2\Psi)L^2 + M\dot{\Phi}\dot{\Psi}sin(2\Psi)\tau^2 + 2M\dot{\Phi}\dot{\tau}Lsin(\Psi)^2 + 2M\dot{\Phi}\dot{\tau}\tau sin(\Psi)^2 +$
$2M\dot{\Phi}\dot{\Psi}sin(2\Psi)L\tau))/(J\Phi + (W^2 m)/2 + (J_w W^2)/(2R^2) + L^2 Msin(\Psi)^2 + M\tau^2 sin(\Psi)^2 +$
$2LM\tau sin(\Psi)^2 + (J_m W^2 n^2)/(2R^2))$

Ψ - coordinate:
$\ddot{\Psi} = ((4F_\Psi J_m + 4F_\Theta J_m - 4F_\tau J_m Rsin(\Psi) - 8J_m LM\dot{\Psi}\dot{\tau} - 8J_m M\tau\dot{\Psi}\dot{\tau} + 4J_m LM\ddot{Z}_g sin(\Psi) +$
$4J_m LMgsin(\Psi) + 4J_m M\tau\ddot{Z}_g sin(\Psi) + 4J_m M\tau gsin(\Psi) + 4J_m R\tau ksin(\Psi) + 2J_m L^2 M\dot{\Phi}^2 sin(2\Psi) +$
$2J_m M\tau^2\dot{\Phi}^2 sin(2\Psi) + 2J_m MR\ddot{Z}_g sin(2\Psi) + 2J_m MRgsin(2\Psi) - 3J_m LMR\dot{\Phi}^2 sin(\Psi) -$
$3J_m MR\tau\dot{\Phi}^2 sin(\Psi) + J_m LMR\dot{\Phi}^2 sin(3\Psi) + 4J_m LM\tau\dot{\Phi}^2 sin(2\Psi) + J_m MR\tau\dot{\Phi}^2 sin(3\Psi) -$
$8J_m MR\dot{\Psi}\dot{\tau}cos(\Psi))n^2 + 4F_\Psi J_w + F_\Psi MR^2 + 4F_\Psi R^2 m + F_\Psi MR^2 cos(2\Psi) - 8J_w LM\dot{\Psi}\dot{\tau} +$
$F_\tau LMR^2 sin(2\Psi) - 8J_w M\tau\dot{\Psi}\dot{\tau} + F_\tau MR^2\tau sin(2\Psi) - 2F_\Theta LMRcos(\Psi) - 2F_\Theta MR\tau cos(\Psi) +$

$L^2M^2R^2\dot{\Phi}^2sin(2\Psi)+M^2R^2\tau^2\dot{\Phi}^2sin(2\Psi)+4J_wLM\ddot{Z}_gsin(\Psi)+4J_wLMgsin(\Psi)+4J_wM\tau\ddot{Z}_gsin(\Psi)+$
$4J_wM\tau gsin(\Psi)+2J_wL^2M\dot{\Phi}^2sin(2\Psi)+2J_wM\tau^2\dot{\Phi}^2sin(2\Psi)-MR^2\tau^2ksin(2\Psi)+4LMR^2\ddot{Z}_gmsin(\Psi)+$
$4LMR^2gmsin(\Psi) + 2LM^2R^2\tau\dot{\Phi}^2sin(2\Psi) + 4MR^2\tau\ddot{Z}_gmsin(\Psi) + 4MR^2\tau gmsin(\Psi) +$
$2L^2MR^2\dot{\Phi}^2msin(2\Psi)+2MR^2\tau^2\dot{\Phi}^2msin(2\Psi)+4J_wLM\tau\dot{\Phi}^2sin(2\Psi)-LMR^2\tau ksin(2\Psi)-$
$8LMR^2\dot{\Psi}\dot{\tau}m - 8MR^2\tau\dot{\Psi}\dot{\tau}m + 4LMR^2\tau\dot{\Phi}^2msin(2\Psi))/((4J_\Psi J_m + 8J_mJ_w + 4J_mL^2M +$
$2J_mMR^2 + 4J_mM\tau^2 + 8J_mR^2m + 8J_mLM\tau + 2J_mMR^2cos(2\Psi) + 8J_mLMRcos(\Psi) +$
$8J_mMR\tau cos(\Psi))n^2 + 4J_\Psi J_w + 4J_wL^2M + J_\Psi MR^2 + 4J_wM\tau^2 + 4J_\Psi R^2m + 4L^2MR^2m +$
$4MR^2\tau^2m + 8J_wLM\tau + J_\Psi MR^2cos(2\Psi) + 8LMR^2\tau m)$

τ - coordinate:
$\ddot{\tau} = 2\ddot{Z}_gsin(\Psi/2)^2 - g - \ddot{Z}_g + 2gsin(\Psi/2)^2 + F_\tau/M + L\dot{\Psi}^2 + \tau\dot{\Psi}^2 + L\dot{\Phi}^2sin(\Psi)^2 +$
$\tau\dot{\Phi}^2sin(\Psi)^2-(\tau k)/M-(Rsin(\Psi)(F_\Theta-Rsin(\Psi)(F_\tau-\tau k+LM\dot{\Psi}^2+M\tau\dot{\Psi}^2-M\ddot{Z}_gcos(\Psi)-$
$Mgcos(\Psi)+LM\dot{\Phi}^2sin(\Psi)^2+M\tau\dot{\Phi}^2sin(\Psi)^2)+(2J_mn^2(F_\Psi+LM\ddot{Z}_gsin(\Psi)+LMgsin(\Psi)+$
$M\tau\ddot{Z}_gsin(\Psi) + M\tau gsin(\Psi) + (L^2M\dot{\Phi}^2sin(2\Psi))/2 + (M\tau^2\dot{\Phi}^2sin(2\Psi))/2 - 2LM\dot{\Psi}\dot{\tau} -$
$2M\tau\dot{\Psi}\dot{\tau} + LM\tau\dot{\Phi}^2sin(2\Psi)))/(ML^2 + 2ML\tau + M\tau^2 + 2J_mn^2 + J_\Psi) - 2MR\dot{\Psi}\dot{\tau}cos(\Psi) +$
$LMR\dot{\Psi}^2sin(\Psi) + MR\tau\dot{\Psi}^2sin(\Psi) - (LMRcos(\Psi)(F_\Psi + LM\ddot{Z}_gsin(\Psi) + LMgsin(\Psi) +$
$M\tau\ddot{Z}_gsin(\Psi) + M\tau gsin(\Psi) + (L^2M\dot{\Phi}^2sin(2\Psi))/2 + (M\tau^2\dot{\Phi}^2sin(2\Psi))/2 - 2LM\dot{\Psi}\dot{\tau} -$
$2M\tau\dot{\Psi}\dot{\tau} + LM\tau\dot{\Phi}^2sin(2\Psi)))/(ML^2 + 2ML\tau + M\tau^2 + 2J_mn^2 + J_\Psi) - (MR\tau cos(\Psi)(F_\Psi +$
$LM\ddot{Z}_gsin(\Psi)+LMgsin(\Psi)+M\tau\ddot{Z}_gsin(\Psi)+M\tau gsin(\Psi)+(L^2M\dot{\Phi}^2sin(2\Psi))/2+(M\tau^2\dot{\Phi}^2sin(2\Psi))/2-$
$2LM\dot{\Psi}\dot{\tau} - 2M\tau\dot{\Psi}\dot{\tau} + LM\tau\dot{\Phi}^2sin(2\Psi)))/(ML^2 + 2ML\tau + M\tau^2 + 2J_mn^2 + J_\Psi)))/(2J_w +$
$MR^2+2J_mn^2+2R^2m-MR^2sin(\Psi)^2+(2J_mn^2(-2J_mn^2+MR\tau cos(\Psi)+LMRcos(\Psi)))/(ML^2+$
$2ML\tau+M\tau^2+2Jmn^2+J_\Psi)-(LMRcos(\Psi)(-2J_mn^2+MR\tau cos(\Psi)+LMRcos(\Psi)))/(ML^2+$
$2ML\tau+M\tau^2+2J_mn^2+J_\Psi)-(MR\tau cos(\Psi)(-2J_mn^2+MR\tau cos(\Psi)+LMRcos(\Psi)))/(ML^2+$
$2ML\tau + M\tau^2 + 2J_mn^2 + J_\Psi))$

B

Matlab scripts

The Matlab script that have been used for modeling and simulations.

B.1 Lagrange equation

Lagrange modeling

Matlab script that uses the function in B.1 to calculate the equations of motion.

```
%% Symbols
% Create symbols
clear;
clc;
syms Theta_r Theta_l dTheta_r dTheta_l;
syms g m R Jw M W D H L JPsi JPhi Jm Rm Kb Kt n fm fw k;

syms Xm Ym Zm dXm dYm dZm;
syms Xl Yl Zl dXl dYl dZl;
syms Xr Yr Zr dXr dYr dZr;
syms Xb Yb Zb dXb dYb dZb;

syms Tau dTau ddTau;
syms Zg dZg ddZg;

syms Theta dTheta ddTheta;
syms Phi dPhi ddPhi;
syms Psi dPsi ddPsi;

syms F_Theta F_Phi F_Psi F_Tau;

% Describe the mass centres in the new coordinate system
Theta_r = (W*Phi)/(2*R) + Theta;
```

```
Theta_l = Theta - (W*Phi)/(2*R);

Zm = R + Zg;

Zl = Zm;    %Z position of the left wheel
Zr = Zm;    %Zl and Zr are equal since we dont have any tilt
Zb = Zm + (L+Tau)*cos(Psi);    %Z position of the upper body mass

%Describe the differential equations
%Wheels
dTheta_r = ((W*dPhi)/(2*R)) + dTheta;
dTheta_l = dTheta - (W*dPhi/(2*R));
%Middle point
dXm = R*dTheta*cos(Phi);
dYm = R*dTheta*sin(Phi);
dZm = dZg;
%Left wheel
dXl = dXm - W/2 * dPhi*cos(Phi);
dYl = dYm - W/2 * dPhi*sin(Phi);
dZl = dZm;
%Right wheel
dXr = dXm + W/2 * dPhi*cos(Phi);
dYr = dYm + W/2 * dPhi*sin(Phi);
dZr = dZm;
%Upper body
dXb = dXm + (L+Tau)*(dPsi*cos(Psi)*cos(Phi)-dPhi*sin(Phi)*sin(Psi)) + ...
    dTau*sin(Psi)*cos(Phi);
dYb = dYm + (L+Tau)*(dPsi*cos(Psi)*sin(Phi)+dPhi*cos(Phi)*sin(Psi)) + ...
    dTau*sin(Psi)*sin(Phi);
dZb = dZm - (L+Tau)*dPsi*sin(Psi) + dTau*cos(Psi);

%%%%%%%% The Lagrangian equations %%%%%%%%
% T1 - Translational kinetic energy
T1 = (1/2)*(m*(dXl^2+dYl^2+dZl^2) + m*(dXr^2+dYr^2+dZr^2) + ...
    M*(dXb^2+dYb^2+dZb^2));
% T2 - rotational kinetic energy
T2 = (1/2)*(Jw*dTheta_l^2 + Jw*dTheta_r^2 + JPsi*dPsi^2 ...
    + JPhi*dPhi^2 + (n^2)*Jm*(dTheta_l-dPsi)^2 + (n^2)*Jm*(dTheta_r - ...
        dPsi)^2);
% Potential energy
U = m*g*Zl + m*g*Zr + M*g*Zb + 1/2*k*Tau^2;

%Calculate the Lagrangian
Lag = T1 + T2 - U;

%Use Lagrange.m to calculate the equations of motion
eqs=Lagrange(Lag,[Theta,dTheta,ddTheta, Psi,dPsi,ddPsi, ...
    Phi,dPhi,ddPhi, Tau,dTau,ddTau, Zg,dZg,ddZg])
%Rearange the algebraic equations
var = {'ddTheta' 'ddPsi' 'ddPhi' 'ddTau' 'ddZg'};
Fvar = {'F_Theta' 'F_Psi' 'F_Phi' 'F_Tau' 'F_Zg'};
temp = [];
answ = {};
```

```
for i= 1:length(var)
    eq = collect(eqs(i),char(var(i)));
    eq_cof = coeffs(eq,char(var(i)));
    answ(i) = {[char(var(i)), ' = (' char(Fvar(i)), ' − ( ' ...
        char(eq_cof(1)), ' )) / (' char(eq_cof(2)),');']};
    char(answ(i))
end
%%%%%% Solving the algebraic loops %%%%%%
eq1 = F_Theta − eqs(1);
eq2 = F_Psi − eqs(2);
eq3 = F_Tau − eqs(4);

% EQUATION 1
% substitute all ddTau in first equation from the third (eqs(4))
eq3_cof_ddTau = coeffs(eq3,'ddTau');
eq1 = subs(eq1, ddTau, (−eq3_cof_ddTau(1)/eq3_cof_ddTau(2)));
% substitute all ddPsi in first equation from the second equation
eq2_cof_ddPsi = coeffs(eq2,'ddPsi');
eq1 = subs(eq1, ddPsi, (−eq2_cof_ddPsi(1)/eq2_cof_ddPsi(2)));
% collect ddTheta terms and rest
eq1_cof = coeffs(eq1,'ddTheta');
ddTheta_lonely = −eq1_cof(1)/eq1_cof(2);
% write result
['ddTheta = ' char(−eq1_cof(1)) ' / ',  char(eq1_cof(2)) ';']

% EQUATION 2
% substitute all ddTheta in second equation with ddTheta_lonely
eq2 = subs(eq2, ddTheta, ddTheta_lonely);
% collect ddPsi terms and rest
eq2_simple = simple(eq2);
eq2_cof = coeffs(eq2_simple,'ddPsi');
ddPsi_lonely = −eq2_cof(1)/eq2_cof(2);
% write result
['ddPsi = ' char(−eq2_cof(1)) ' / ',  char(eq2_cof(2)) ';']

% EQUATION 3
% substitute all ddTheta in third equation with ddTheta_lonely
eq3 = subs(eq3, ddTheta, ddTheta_lonely);
% collect ddTau terms and rest
eq3_cof = coeffs(eq3,'ddTau');
ddTau_lonely = −eq3_cof(1)/eq3_cof(2);
% write result
['ddTau = ' char(−eq3_cof(1)) ' / ',  char(eq3_cof(2)) ';']
```

The Lagrange function file

This function solves the equations of motion. It can be found on Mathworks file exchange

```
% Lagrange is a function that calculate equations of motion (Lagrange's
% equations) d/dt(dL/d(dq))− dL/dq=0. It Uses  the Lagrangian that is ...
    a function that summarizes the
```

```
% dynamics of the system.  Symbolic Math Toolbox is required.
%
% Equations=Lagrange(Lag,V)
%
% Lag = Lagrange of the system (symbolic).
% V   = System Variables (symbolic) [q1 dq1 ddq1 q2 dq2 ddq2... qn dqn
% ddqn].
% Equations   = [1 X DOF] (Degrees of freedom of the system).
%
%
% *******Examples*********
%     *Falling mass*
%
% syms x dx ddx t m      %Define the symbolic variables.
% L=0.5*m*dx^2 + m*g*x;  %Define the Lagragian.
% Equations=Lagrange(L,[x dx ddx]) %Calculate the equations
%
% returns   m*ddx-g*m
%
% *Pendulum on a movable support*
%
% syms x dx ddx theta dtheta ddtheta t m M   %Define the symbolic ...
  variables.
%
% L=0.5*(M+m)+dx^2+ m*dx*l*dtheta*cos(theta)+ ...
% 0.5*m*l^2*dtheta^2+m*g*l*cos(theta)        %Define the Lagragian.
% Equations=Lagrange(L,[theta,dtheta,ddtheta,x,dx,ddx]) %Calculate the
% equations
%
% returns   [  m*l*(ddx*cos(theta)+l*ddtheta+g*sin(theta)),
%              2*ddx+m*l*ddtheta*cos(theta)-m*l*dtheta^2*sin(theta)]

function [M]=Lagrange(Lag,V)
syms t;
Var=length(V)/3;
Vt=V;
    for cont0=1:1:Var
        Vt(cont0*3-2)=strcat('f',num2str(cont0),'(t)');
        Vt(cont0*3-1)=diff(Vt((cont0*3)-2),t);
        Vt(cont0*3)=diff(Vt((cont0*3)-2),t,2);
    end
    for cont0=1:1:Var
        L1=simple(diff(Lag,V(cont0*3-1)));
        L2=simple(diff(Lag,V(cont0*3-2)));
        Dposx=L1;

        for cont=1:1:Var*3
             Dposx=subs(Dposx,V(cont),Vt(cont));
        end
        L1=diff(Dposx,t);

        for cont=Var*3:-1:1         %
             L1=subs(L1,Vt(cont),V(cont));
```

```
            end
            L1F=L1−L2;
            L1F=simple(expand(L1F));
            L1F=collect(L1F,Vt(cont0*3));%*****************
            M(cont0)=L1F;
        end
    end
```

B.2 Parameters

Parameters of the reference vehicle used in the simulations and calculations.

```
% Parameters
g = 9.82;                         % acceleration of gravity
m = 15;                           % wheel weight [kg]
R = 0.4;                          % wheel radius [m]
Jw = m * R^2 / 2;                 % wheel inertia moment [kgm^2]
M = 870;                          % body weight [kg]
W = 1.6;                          % body width [m]
D = 1.4;                          % body depth [m]
H = 2;                            % body height [m]
L = 1.0*(H/3);                    % distance to center of mass from the ...
    wheel axle [m]
JPsi = M * L^2 / 3;               % body pitch inertia moment [kgm^2]
JPhi = M * (W^2 + D^2) / 12;      % body yaw inertia moment [kgm^2]
fm = 0.2;                         % friction coefficient between chassis ...
    & motor
fw = 1;                           % friction coefficient between wheel & ...
    floor
n = 1;                            % gear ratio
Jm = 1E−5;                        % motor inertia moment [kgm^2]
C = 0.7;                          % drag coefficient
p = 1.2;                          % density of air
A = H*W;                          % area of the body
k = 100000;                       % spring Constant for suspension
b = 10000;                        % damping constant for suspension

% struct to import constants in embedded matlab functions in simulink ...
    model
st = struct('g', g, 'm', m, 'R', R, 'Jw', Jw, 'M', M, 'W', W, 'D', D, ...
    'H', H, ...
    'L', L, 'JPsi', JPsi, 'JPhi', JPhi, 'Jm', Jm, 'fm', fm, 'fw', fw, ...
        'n', n, ...
    'C', C, 'p', p, 'A', A, 'k', k, 'b', b);

save('parameters_real.mat', 'st')
```

B.3 Linearization

Matlab sccript used for lineariaztion

```
clc
clear
parameters
syms Theta_r Theta_l dTheta_r dTheta_l;
syms g m R Jw M W D H L JPsi JPhi Jm n fm fw k b p A C;
syms M_l M_r;

syms Xm Ym Zm dXm dYm dZm;
syms Xl Yl Zl dXl dYl dZl;
syms Xr Yr Zr dXr dYr dZr;
syms Xb Yb Zb dXb dYb dZb;

syms Tau dTau ddTau;
syms Zg dZg ddZg;

syms Theta dTheta ddTheta;
syms Phi dPhi ddPhi;
syms Psi dPsi ddPsi;

syms F_Theta F_Phi F_Psi F_Tau Fdrag;

% Load equaitions (without loops)
load ddPhi_ddTau_ddTheta_ddPsi

% Drag model
Fdrag = ((H*p*A*C)/4)*(R*dTheta*cos(Phi)+ ...
    (Tau+L)*(dPsi*cos(Psi)*cos(Phi)- ...
    dPhi*sin(Phi)*sin(Psi))+dTau*sin(Psi)*cos(Phi))^2;

% Substitute the F_:s to other variables
ddTheta = subs(ddTheta, [F_Theta,F_Psi,F_Phi,F_Tau], ...
    [n*(M_l+M_r)+2*fm*dPsi-2*dTheta*(fm+fw),-n*(M_l+M_r)+2*fm*dPsi-Fdrag ...
        ...
    ,(W*n*(M_r-M_l))/(2*R)-(W^2*(fm+fw)*dPhi^2)/(2*R^2), -b*dTau]);
ddPsi = subs(ddPsi, [F_Theta,F_Psi,F_Phi,F_Tau], ...
    [n*(M_l+M_r)+2*fm*dPsi-2*dTheta*(fm+fw),-n*(M_l+M_r)+2*fm*dPsi-Fdrag ...
        ...
    ,(W*n*(M_r-M_l))/(2*R)-(W^2*(fm+fw)*dPhi^2)/(2*R^2), -b*dTau]);
ddPhi = subs(ddPhi, [F_Theta,F_Psi,F_Phi,F_Tau], ...
    [n*(M_l+M_r)+2*fm*dPsi-2*dTheta*(fm+fw),-n*(M_l+M_r)+2*fm*dPsi-Fdrag ...
        ...
    ,(W*n*(M_r-M_l))/(2*R)-(W^2*(fm+fw)*dPhi^2)/(2*R^2), -b*dTau]);
ddTau = subs(ddTau, [F_Theta,F_Psi,F_Phi,F_Tau], ...
    [n*(M_l+M_r)+2*fm*dPsi-2*dTheta*(fm+fw),-n*(M_l+M_r)+2*fm*dPsi-Fdrag ...
        ...
    ,(W*n*(M_r-M_l))/(2*R)-(W^2*(fm+fw)*dPhi^2)/(2*R^2), -b*dTau]);

sys = [ddTheta, ddPsi, ddPhi];
```

```

% Find A matrix
v = [Theta, dTheta, Psi, dPsi, Phi, dPhi];
A_sys = jacobian(sys,v);
A_sys = subs(A_sys,[Theta, dTheta, Psi, dPsi, Phi, dPhi, Tau, dTau, ...
    ddZg],[0, 0, 0, 0, 0, 0, 0, 0, 0]);

% Find B matrix
v = [M_l, M_r];
B_sys = jacobian(sys,v);
B_sys = subs(B_sys,[Theta, dTheta, Psi, dPsi, Phi, dPhi, Tau, dTau, ...
    ddZg],[0, 0, 0, 0, 0, 0, 0, 0, 0]);

% replace with parameters
load parameters_real

B_sys = subs(B_sys, ...
    [g,m,R,Jw,M,W,D,H,L,JPsi,JPhi,Jm,fm,fw,n,C,p,A,k,b], ...
    [st.g,st.m,st.R,st.Jw,st.M,st.W,st.D,st.H,st.L,st.JPsi,st.JPhi,st.Jm,st.fm,st.fw,st.n,s
A_sys = subs(A_sys, ...
    [g,m,R,Jw,M,W,D,H,L,JPsi,JPhi,Jm,fm,fw,n,C,p,A,k,b], ...
    [st.g,st.m,st.R,st.Jw,st.M,st.W,st.D,st.H,st.L,st.JPsi,st.JPhi,st.Jm,st.fm,st.fw,st.n,s

% System with Theta, Psi, Phi
A_sys1 = [0 1 0 0 0 0; A_sys(1,:); 0 0 0 1 0 0; A_sys(2,:); 0 0 0 0 0 ...
    1; A_sys(3,:)];
B_sys1 = [0 0; B_sys(1,:); 0 0; B_sys(2,:); 0 0; B_sys(3,:)];
C_sys1 = [0 1 0 0 0 0;
          0 0 1 0 0 0;
          0 0 0 1 0 0;
          0 0 0 0 0 1];

sys = ss(A_sys1, B_sys1, C_sys1, 0);


save('linearized_system','sys');
clear
```

B.4 init simulation

This file initiate the simulation and it calculates the Kalman gain and the LQR feedback gain.

```
% Initial conditions for integrator blocks in simulink model
Tau_init = -0.08;
Phi_init = 0;
Psi_init = 0.0;
Theta_init = 0;

% Sensor variance
```

```
tachom_var = 0.003;
inertialsensor_var = 0.003;

% Motor parameters
motor_delay = 0.020;
motor_saturation = 1000*100;

% Sensor filters
sensor_passband_freq = 100*2*pi;

% Sensor sampling time
Ts_sensor = 0.001;

% Sampling time in controller block
Ts_controller = 0.001;

% Sampling time for the motor
Ts = 6e-5;

% Load parameters
load parameters_real
% Load states bus from the plant
load bus_from_plant
% Load linearized system
load linearized_system;

%%%% LQG controller without integration factor %%%%
% Remove the states we dont care about (dTheta, Psi, dPsi and dPhi left)
check_poles = sys;
sys = minreal(sys);
% Discretisize the system
sys_d = c2d(sys, Ts_sensor, 'zoh');

% Set LQR parameters
%       dTheta    Psi     dPsi     dPhi  dTheta_integral
Q = [    2e4      0        0        0        0;
         0        1e7      0        0        0;
         0        0        1e-7     0        0;
         0        0        0        1e4      0;
         0        0        0        0        1e2];

R = [1     0;
     0     1];

sys_for_lqi = ss(sys.A, sys.B, [1 0 0 0], 0);
sys_for_lqi_d = c2d(sys_for_lqi, Ts_sensor, 'zoh');

[K_lqr,S_lqr,e_lqr] = lqi(sys_for_lqi_d, Q, R);

% Matrices describing the noise
% R1 = E[ww'],   R2 =[vv'],   R12 = E[wv']
% w - process disturbances
% v - measurement noise (on y)
```

```

R1 = 0.5* [1       0     0     0;
           0       1     0     0;
           0       0     1     0;
           0       0     0     1]; %QN

R2 = [0.0001      0         0          0;
        0         0.0001    0          0;
        0         0         0.0001     0;
        0         0         0          0.0001]; %RN

R12 = []; %NN

% Rewrite the system to this form
% SYS=SS(A,[B G],C,[D H])
%sys_kalman = ss(sys.A, [sys.B eye(6)], sys.C, [sys.D zeros(4,6)]);
sys_kalman = ss(sys.A, [sys.B eye(4)], sys.C, [sys.D zeros(4,4)]);
sys_kalman_d = c2d(sys_kalman, Ts_sensor, 'zoh');

[kalmf,KalmanGain,P] = kalman(sys_kalman_d, R1, R2, R12);

% This system will be used in the simulink model of the Kalman filter
sys = sys_d;
```