TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Electrodynamic magnetic levitation

Citation for published version (APA):
Boeij, de, J. (2003). *Electrodynamic magnetic levitation: NASA's vision of the future*. (DCT rapporten; Vol. 2003.055). Technische Universiteit Eindhoven.

Document status and date:
Published: 01/01/2003

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 17. Nov. 2023

Electrodynamic Magnetic Levitation
NASA's vision of the future

DCT 2003.55

Jeroen de Boeij (s461064)
July 4, 2003

Preface

To reduce the cost of a space flight, NASA is developing new methods to launch a spacecraft. The new spacecraft will not have rocket boosters or external tanks. A new rocket engine is under development, which is efficient and powerful enough to launch the spacecraft into space without additional boosters. One of the disadvantages of this new type of engine is that it needs a certain speed to ignite. Therefore a launch assist is necessary to boost the spacecraft to this initial speed. The idea is to put the spacecraft on a carrier, which will function as a launch platform and accelerate the carrier to approximately 225 meters per second. The high speed and acceleration requirements complicate the design.

Magnetic levitation (Maglev) is a technique that can match the requirements for the launch assist [1]. The carrier is lifted of the track and propelled by magnetic forces. This eliminates friction and wear which makes it very suitable for high-speed applications. There are two types of magnetic levitation.

1. Electrodynamic Maglev (EDS Maglev) uses repulsive forces induced by the motion of a magnet relative to a fixed coil. When the levitation circuits are passive, EDS requires an external propulsion mechanism, usually a linear motor, to induce the magnetic fields required to achieve levitation. The main advantages are its low lift to drag ratio and the large gaps allowed between the carrier (sled) and the track. When permanent magnets are used on the sled there is no need for power on the sled. The technique does have disadvantages. The lift forces are induced by the motion of the sled and therefore an initial velocity is needed to lift the sled. This implies that an additional suspension is necessary to support the carrier at take off. The dynamics of the magnetic suspension are stable but have very poor damping and become instable at higher speeds.

2. The other type is Electromagnetic Maglev (EMS Maglev), which uses electromagnetic attractive forces produced by the pass of a current through a coil wrapped around an iron core. Large electromagnets on the sled provide levitation, so stand-still levitation is possible. EMS however has one major disadvantage. The suspension only operates within very tight tolerances: the gap between the track and the carrier must be accurate within a few millimeters. This makes the technique extremely expensive and not suitable for a couple of kilometers long launch track.

EDS Maglev is the only technique capable of matching the launch assist requirements, but before this technique can be used a better insight of its dynamics is needed, since only feedback control can provide the reliability required. In this report a three DOF mathematical model is proposed for Electrodynamic maglev system with a passive sled. The model is based on a test track developed by NASA and Foster-Miller [1]. Simulations of the model are used to study the suspension dynamics. Measurements done on the track are used to verify the model.

To improve the dynamics of the magnetic suspension the possibility is studied to replace the passive suspension by an active suspension. The mathematical model is used to develop a controller for the active suspension and the controller is tested in simulations.

Contents

1 Introduction **5**
- 1.1 Introduction to the Test-Track . . . 5
  - 1.1.1 Sled . . . 5
  - 1.1.2 Track . . . 5
- 1.2 Technical Background . . . 6

2 3-DOF Model of EDS Maglev **8**
- 2.1 Model Assumptions . . . 8
- 2.2 Interaction of one permanent magnet with two figure-8 levitation coils . . . 8
- 2.3 Full Scale 3-DOF Model of Sled Dynamics . . . 10
  - 2.3.1 Interaction of Six Magnets with Ten Coils . . . 10
  - 2.3.2 Dynamics of the Sled . . . 11
- 2.4 State-Space Notation and Parameter Estimation . . . 14
- 2.5 Model Implementation . . . 16

3 Model Simulation Results **17**
- 3.1 Sled Dynamics . . . 17
- 3.2 Coil Currents . . . 17

4 Model Validation **22**
- 4.1 Experimental Setup . . . 22
  - 4.1.1 Limitations of the Experimental Setup . . . 23
- 4.2 Coil Current Measurements . . . 23
- 4.3 Validation of the State-Space Model . . . 24

5 Feedback Control of 3-DOF Model **34**
- 5.1 Input Mapping . . . 34
- 5.2 Linear Control . . . 35
- 5.3 Sliding Mode Control . . . 36
- 5.4 Controller Implementation . . . 37

6 Controller Simulation Results **38**

7 Conclusions and Future Research **43**
- 7.1 Conclusions . . . 43
- 7.2 Future Research for Model Validation . . . 43
- 7.3 Future Research for Feedback Control . . . 43

A Flux and Voltage equations of 3-DOF Model **45**

B C-code 3-DOF EDS Maglev Model **48**

C M-File for parameter estimation **61**

D C-code for Feedback Control Simulation **68**
D.1 C-code Sliding Mode Controller 68
D.1.1 C-code for Matrix Inversion 79
D.2 C-code 3-DOF Model for Feedback Control 80

Chapter 1

Introduction

1.1 Introduction to the Test-Track

Before the mathematical model is developed, the geometry and working principles of the track and the sled are explained.

1.1.1 Sled

The main parts of the sled are the two arrays of 6 permanent magnets. The propulsion coils of the track are located between the two arrays of magnets. Also the sled is equipped with guidance wheels. These wheel can make contact with the guidance rail to limit the levitation to a certain lower and upper boundary, and also to restrict sideways motion. The flight computer and data acquisition equipment for real-time measurements are located on top of the sled.

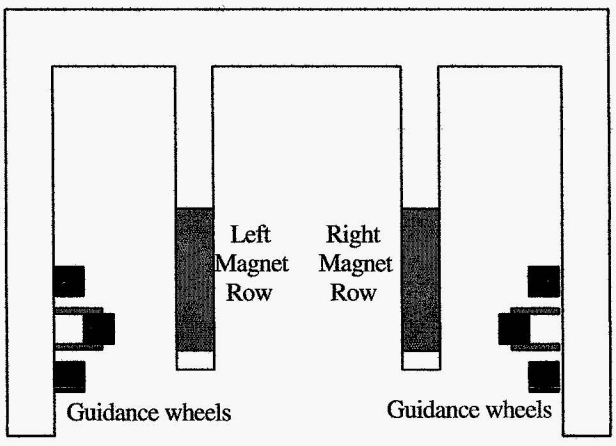

Figure 1.1: Sled Layout

1.1.2 Track

The track has two sections, a propulsion section and a braking section. Both sections have three parts. The outer parts contain an array of figure eight levitation coils. The middle part of the propulsion section is an array of propulsion coils. The middle part of the braking section is a metal rail, which functions as an eddy current brake.

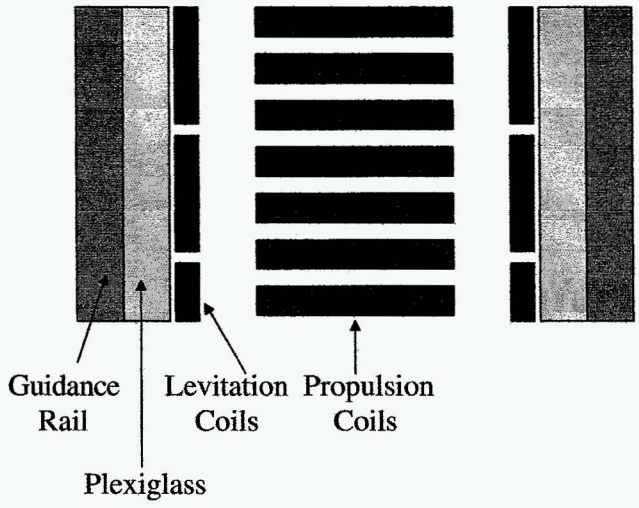

Figure 1.2: Track Layout

The current through the propulsion coils is triggered by the position of the sled along the guideway. Optical sensors on the track detect the position of the sled and these sensors activate the current driver boards and switches. For the 3-DOF levitation model, the propulsion is not relevant. It is assumed that the propulsion generates a propulsion force $F(t)$, which results in an arbitrary profiles for acceleration $a(t)$, velocity $v(t)$ and position $x(t)$ along the guideway. More detailed information about the propulsion section and its hardware is found in [1].
On both sides of the track, an array of figure-8 shaped coils is mounted on the track. When the magnets of the sled pass a coil, a current is generated by the difference in change of flux between the top and the bottom coil. The induced current produces a force in the magnet given by Lorentz's Law. Three different forces are generated, but only the vertical or levitation force is considered. Drag forces are not relevant since the propulsion force is not part of the model.

1.2 Technical Background

To give an insight in previous research in the field of EDS Maglev the technical background of this research project is discussed in this section.

Since the introduction of the concept of figure-8 shaped levitation coils for Maglev applications, extensive research has been done in this field. Several models have been developed to describe the interaction between the levitation coils and the magnets (either permanent or super conductive) on the sled. He, Rote and Coffey developed models as in [2] and [5] using dynamic circuit theory. This theory uses matrix equations to describe the behavior of the currents and voltages in the system. The parameters in the matrices depend on time and space so the models are highly accurate but have the disadvantage that the computing is time consuming and a large set of parameters has to be estimated. The models proposed by Davey in [3] and [4] use a more simple approach. Not the complete system is analyzed, but only the interaction of either a limited number of coils or a limited number of magnets. This analysis results in frequency-domain expressions, which can be used to calculate time-averaged characteristics of EDS maglev and are very useful for design purposes but are not suited for real-time applications.
In addition, none of these models links the electrodynamic forces to vehicle dynamics. The maglev principle as studied by NASA [6] has a separate propulsion section (LSM) and uses null-flux coils for the suspension only. Therefore, an integrated analysis of the suspension and the propulsion as in [2] and [5] is not necessary for this type of maglev applications. In this report, a model of

the maglev test track from NASA [1] is proposed that is simple and accurate enough for real-time applications.

Several attempts have been made to improve the dynamics of the EDS maglev suspension. For instance, the Japanese have made several attempts to stabilize the dynamics of their EDS maglev train by adding additional damping coils to the system [12]. This increased the damping but not enough to provide a stable ride. The Americans had similar stability problems with the Holloman Rocket System and studied the possibility to add damping by using damping plates [9]. Again, there was some improvement but not enough to meet the operational criteria. The next step was to use active damper coils. The Japanese studied this [10] and again some improvement was made. Brunelli, Casadei, Serra and Tani developed a similar active damping control [11]. Nevertheless, the dynamics of the EDS maglev suspension are still very poor. The first attempt to actively control EDS maglev by feedback was made by Thompson and Thornton [8]. They developed a rotational test rig and designed a simple controller that improved the dynamics of the suspension. This approach was limited in the sense that only the vertical dynamics (1 DOF) was considered. In this report, a controller is developed to control the 3-DOF behavior of an EDS maglev vehicle using a state-space model of the sled.

Chapter 2

3-DOF Model of EDS Maglev

To develop a model for EDS Maglev, first some assumptions are made to simplify the model equations. Then the interaction between two figure-8 levitation coils and one magnet is discussed. Finally, the basic interaction of one magnet with two coils are used to develop the full scale 3-DOF model of the sled dynamics.

2.1 Model Assumptions

The following assumptions have been made:

1. The magnetic field of the permanent magnet is uniform over its surface. This in practice means that a non-uniform field can be approximated by an average time-invariant effective field strength.
2. The mutual inductance of the figure-8 coils can be neglected. This assumption is supported by finite element simulations of the magnetic field in adjacent coils in a null-flux system as described in [1] , as well by our own observations: when exciting a figure-8 coil with a magnet moving in the vertical direction, zero currents were observed in the neighboring coils.
3. The flux through the coil depends only on the x- and z-position of the magnet with respect to the coil.
4. The magnet stays always between the upper and lower edges of each figure-8 coil.

2.2 Interaction of one permanent magnet with two figure-8 levitation coils

To illustrate the model the equations will be derived for one magnet interacting with two coils. Figure 2.1 shows the geometric interaction of one magnet with two levitation coils. Based on the previous assumptions, the following equations describe for the flux through each coil,

$$Coil_A : \Phi_A = Bx\left(\frac{h}{2}+\delta\right) - Bx\left(\frac{h}{2}-\delta\right) = 2Bx\delta \tag{2.1}$$

$$Coil_B : \Phi_B = B(b-x)\left(\frac{h}{2}+\delta\right) - B(b-x)\left(\frac{h}{2}-\delta\right) = 2B(b-x)\delta \tag{2.2}$$

where x is the horizontal displacement of the magnet with respect to the left edge of coil A, B is the magnetic field strength of the permanent magnet (assumed constant) and δ is the vertical displacement of the centerline of the magnet with respect to the null-flux axis. In this particular

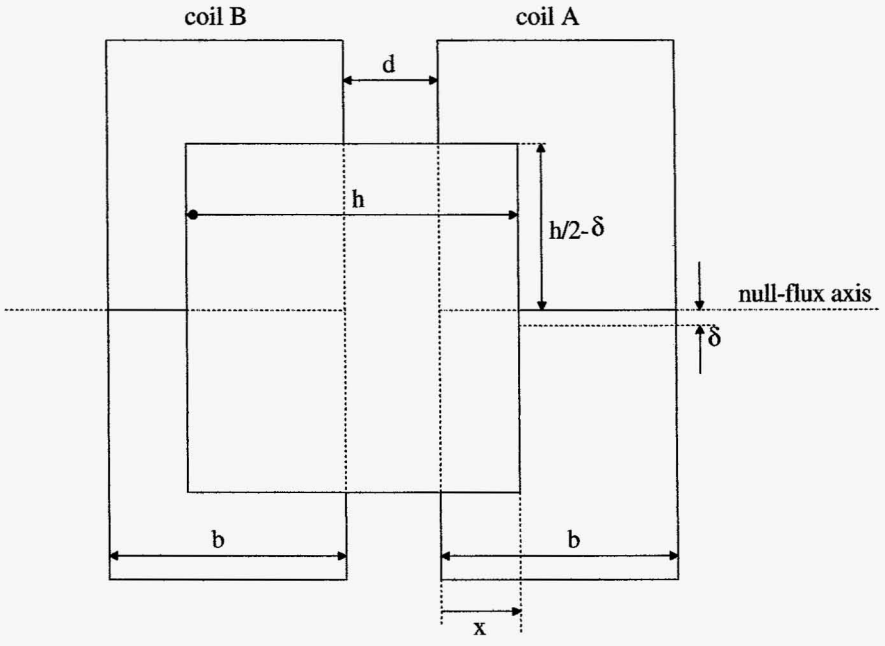

Figure 2.1: One permanent magnet with two successive figure-8 levitation coils

system where $h = b + d$, the maximum value of x is $b + d = h$ and after that a new coil will be influenced by the magnet (becoming coil A), the former coil A becomes coil B and x must be reset to zero.

The equations do not stay the same for all values of x. For $b \leq x \leq h$, the flux equations for coil A and coil B change to:

$$Coil_A : \Phi_A = 2Bb\delta \tag{2.3}$$

$$Coil_B : \Phi_B = 0 \tag{2.4}$$

In systems with different geometry it should be rather straightforward to develop a set of similar expressions. In a similar way equations can be derived for the flux through multiple levitation coils interacting with multiple magnets.
These flux equations are used to calculate the induced voltages in the coils using the following equation:

$$V_{coil} = -\frac{\partial \Phi_{coil}}{\partial t} \tag{2.5}$$

For the example above the equations become:

$$\begin{aligned} Coil\,A: \quad & V_a = -2Bx\dot{\delta} - 2B\delta\dot{x} && 0 \leq x \leq b \\ & V_a = -2Bb\dot{\delta} && b \leq x \leq h \\ Coil\,B: \quad & V_b = -2Bb\dot{\delta} + 2Bx\dot{\delta} + 2B\delta\dot{x} && 0 \leq x \leq b \\ & V_b = 0 && b \leq x \leq h \end{aligned} \tag{2.6}$$

Using the standard equations for electric circuits the currents passing through the levitation coils can be calculated.

$$\frac{\partial i}{\partial t} = \frac{1}{L}(V - Ri) \tag{2.7}$$

After the current in each coil has been calculated, it is possible to calculate the lifting force acting on the magnet. This can be done by using the Lorentz force equations or by a formula that can be used in this particular case.

$$F_{magnet} = i\frac{\partial \Phi_m}{\partial \delta} \tag{2.8}$$

The force on one magnet depends only on the flux generated by that particular magnet. The magnet in this example is lifted by the following force:

$$\begin{aligned} F &= i_a 2Bx + i_b 2B(b-x) \quad 0 \leq x \leq b \\ F &= i_a 2Bb \quad b \leq x \leq h \end{aligned} \tag{2.9}$$

These are the basics of the model. The complete model of the sled and track interaction considers a total of twenty figure-8 levitation coils interacting with the 12 magnets on the sled.

2.3 Full Scale 3-DOF Model of Sled Dynamics

The full scale EDS Maglev model describes the sled's motion in 3 DOF: levitation height (δ), pitch and roll. The vertical forces acting on each magnet are used to calculate the total lift force and the torques on pitch and roll. Drag forces are not considered since the horizontal motion is not part of the model, and guidance or centering forces are not included since the yaw motion of the sled is restricted by roller supports. Some disturbances and non-linearities have been incorporated in the model.

2.3.1 Interaction of Six Magnets with Ten Coils

In this section the equations for the full scale model are discussed. The principles used are the same as those presented in the former section. In figure 2.2 the layout of one side of the sled is displayed. Each side of the sled has 6 permanent magnets interacting witch 10 levitation coils. Also all geometric features of the magnets and coils used in the model are displayed in figure 2.2.

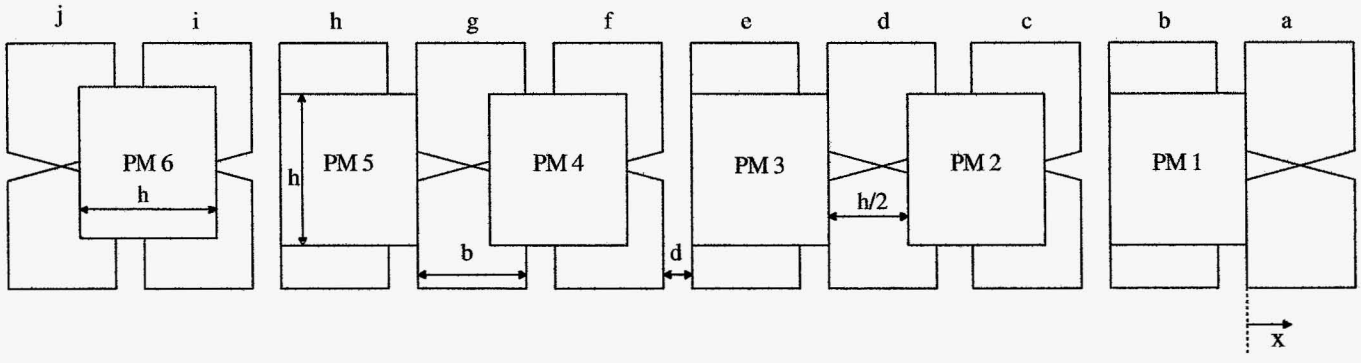

Figure 2.2: Interaction of six permanent magnets with ten figure-8 levitation coils

To derive the flux equations for the coils, the problem has to be split in four different sections, depending on the location of the sled along the guideway.

1. $0 \leq x \leq \frac{1}{2}h - d$
2. $\frac{1}{2}h - d \leq x \leq \frac{1}{2}h$

3. $\frac{1}{2}h \leq x \leq b$

4. $b \leq x \leq h$

When x reaches the next coil ($x = h$), x is reset to 0. The coils are updated, so the new coil becomes coil a and a becomes b, b becomes c, etc. Using the principles of one magnet and two coils, flux equations for each coil are derived. Flux equations are later used to derive the voltage equations for each coil and the force equations for each magnet, using equations 2.5, 2.7 and 2.8. The flux and voltage equations are listed in appendix A. The force equations are listed below, since these are used to describe the sled's 3-DOF dynamics.

1. First set of force equations: $0 \leq x \leq \frac{1}{2}h - d$ (2.10)

$$
\begin{aligned}
F_1 &= i_a 2B_1 x + i_b 2B_1(b - x) \\
F_2 &= i_c 2B_2(\tfrac{1}{2}h + x) + i_d 2B_2(\tfrac{1}{2} - d - x) \\
F_3 &= i_d 2B_3 x + i_e 2B_3(b - x) \\
F_4 &= i_f 2B_4(\tfrac{1}{2}h + x) + i_g 2B_4(\tfrac{1}{2}h - d - x) \\
F_5 &= i_g 2B_5 x + i_h 2B_5(b - x) \\
F_6 &= i_i 2B_6(\tfrac{1}{2}h + x) + i_j 2B_6(\tfrac{1}{2}h - d - x)
\end{aligned}
$$

2. Second set of force equations: $\frac{1}{2}h - d \leq x \leq \frac{1}{2}h$ (2.11)

$$
\begin{aligned}
F_1 &= i_a 2B_1 x + i_b 2B_1(b - x) \\
F_2 &= i_c 2B_2 b \\
F_3 &= i_d 2B_3 x + i_e 2B_3(b - x) \\
F_4 &= i_f 2B_4 b \\
F_5 &= i_g 2B_5 x + i_h 2B_5(b - x) \\
F_6 &= i_i 2B_6 b
\end{aligned}
$$

3. Third set of force equations: $\frac{1}{2}h \leq x \leq b$ (2.12)

$$
\begin{aligned}
F_1 &= i_a 2B_1 x + i_b 2B_1(b - x) \\
F_2 &= i_b 2B_2(x - \tfrac{1}{2}h) + i_c 2B_2(b - x + \tfrac{1}{2}h) \\
F_3 &= i_d 2B_3 x + i_e 2B_3(b - x) \\
F_4 &= i_e 2B_4(x - \tfrac{1}{2}h) + i_f 2B_4(b - x + \tfrac{1}{2}h) \\
F_5 &= i_g 2B_5 x + i_h 2B_5(b - x) \\
F_6 &= i_h 2B_6(x - \tfrac{1}{2}h) + i_i 2B_6(b - x + \tfrac{1}{2}h)
\end{aligned}
$$

4. Fourth set of force equations: $b \leq x \leq h$ (2.13)

$$
\begin{aligned}
F_1 &= i_a 2B_1 b \\
F_2 &= i_b 2B_2(x - \tfrac{1}{2}h) + i_c 2B_2(b - x + \tfrac{1}{2}h) \\
F_3 &= i_d 2B_3 b \\
F_4 &= i_e 2B_4(x - \tfrac{1}{2}h) + i_f 2B_4(b - x + \tfrac{1}{2}h) \\
F_5 &= i_g 2B_5 b \\
F_6 &= i_h 2B_6(x - \tfrac{1}{2}h) + i_i 2B_6(b - x + \tfrac{1}{2}h)
\end{aligned}
$$

2.3.2 Dynamics of the Sled

Now that all the electrodynamic forces acting on the sled's magnets are known, the geometry of the sled is used to derive the dynamic equations for each DOF. In figure 2.3 and 2.4 the basic geometry of the sled is displayed and the pitch and roll angels are defined. The distances L4, L5, L6 and D2 have negative sign.

This geometry links the position and velocity of the magnets to levitation height δ, pitch p and roll r.

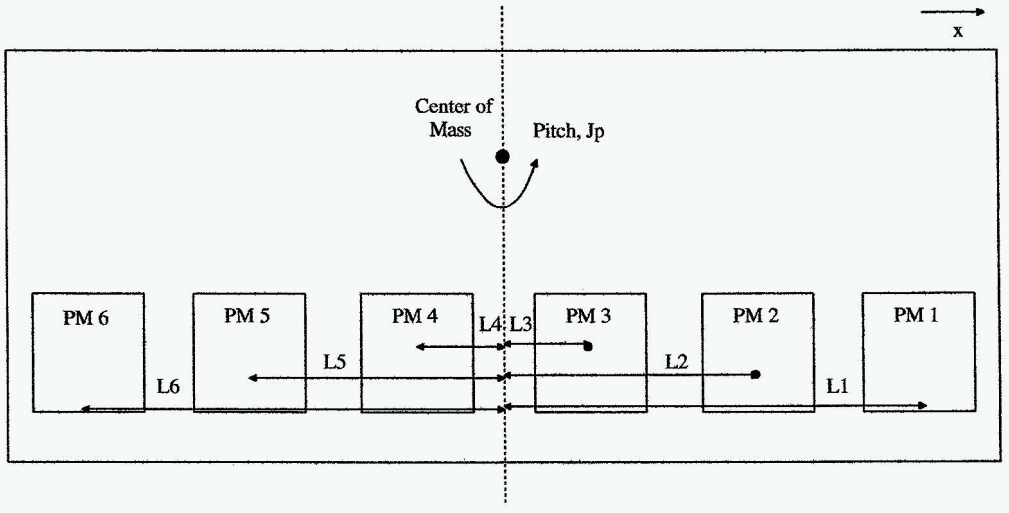

Figure 2.3: Sled geometry and pitch angle

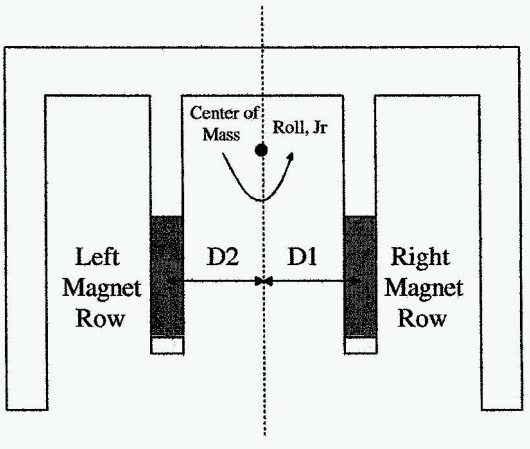

Figure 2.4: Sled geometry and roll angle

1. Equations for right row magnets: (2.14)

$$\begin{aligned}
\delta_1 &= L_1\sin(p) + D_1\sin(r) + z & \dot{\delta}_1 &= L_1\dot{p}\cos(p) + D_1\dot{r}\cos(r) + \dot{z}\\
\delta_2 &= L_2\sin(p) + D_1\sin(r) + z & \dot{\delta}_2 &= L_2\dot{p}\cos(p) + D_1\dot{r}\cos(r) + \dot{z}\\
\delta_3 &= L_3\sin(p) + D_1\sin(r) + z & \dot{\delta}_3 &= L_3\dot{p}\cos(p) + D_1\dot{r}\cos(r) + \dot{z}\\
\delta_4 &= L_4\sin(p) + D_1\sin(r) + z & \dot{\delta}_4 &= L_4\dot{p}\cos(p) + D_1\dot{r}\cos(r) + \dot{z}\\
\delta_5 &= L_5\sin(p) + D_1\sin(r) + z & \dot{\delta}_5 &= L_5\dot{p}\cos(p) + D_1\dot{r}\cos(r) + \dot{z}\\
\delta_6 &= L_6\sin(p) + D_1\sin(r) + z & \dot{\delta}_6 &= L_6\dot{p}\cos(p) + D_1\dot{r}\cos(r) + \dot{z}
\end{aligned}$$

2. Equations for left row magnets: (2.15)

$$\begin{aligned}
\delta_1 &= L_1\sin(p) + D_2\sin(r) + z & \dot{\delta}_1 &= L_1\dot{p}\cos(p) + D_2\dot{r}\cos(r) + \dot{z}\\
\delta_2 &= L_2\sin(p) + D_2\sin(r) + z & \dot{\delta}_2 &= L_2\dot{p}\cos(p) + D_2\dot{r}\cos(r) + \dot{z}\\
\delta_3 &= L_3\sin(p) + D_2\sin(r) + z & \dot{\delta}_3 &= L_3\dot{p}\cos(p) + D_2\dot{r}\cos(r) + \dot{z}\\
\delta_4 &= L_4\sin(p) + D_2\sin(r) + z & \dot{\delta}_4 &= L_4\dot{p}\cos(p) + D_2\dot{r}\cos(r) + \dot{z}\\
\delta_5 &= L_5\sin(p) + D_2\sin(r) + z & \dot{\delta}_5 &= L_5\dot{p}\cos(p) + D_2\dot{r}\cos(r) + \dot{z}\\
\delta_6 &= L_6\sin(p) + D_2\sin(r) + z & \dot{\delta}_6 &= L_6\dot{p}\cos(p) + D_2\dot{r}\cos(r) + \dot{z}
\end{aligned}$$

Using the force equations and the number of windings on each coil N, the dynamic equations can be derived.

Total force on right magnet row:

$$F_{right} = \sum_{i=1}^{6} NF_{i,right} \tag{2.16}$$

Total force on left magnet row:

$$F_{left} = \sum_{i=1}^{6} NF_{i,left} \tag{2.17}$$

Total force on sled:

$$F_{total} = F_{left} + F_{right} \tag{2.18}$$

Height dynamics:

$$\ddot{z} = \frac{F_{total}}{M} - g \tag{2.19}$$

Torque on pitch axis and pitch dynamics:

$$\begin{aligned}
T_p \;=\; & Ncos(p)((F_{l,1} + F_{r,1})L_1 + (F_{l,2} + F_{r,2})L_2 + (F_{l,3} + F_{r,3})L_3 \\
& +(F_{l,4} + F_{r,4})L_4 + (F_{l,5} + F_{r,5})L_5 + (F_{l,6} + F_{r,6})L_6)
\end{aligned} \tag{2.20}$$

$$\ddot{p} = \frac{T_p}{J_p} \tag{2.21}$$

Torque on roll axis and roll dynamics:

$$T_r = (F_{right}D_1 + F_{left}D_2)\cos(r) \tag{2.22}$$

$$\ddot{r} = \frac{T_r}{J_r} \tag{2.23}$$

To include the effect of uneven weight distribution, a constant disturbance torque is added to the pitch and roll equations. The moment of inertia in pitch and roll was estimated using the geometry and the material properties. Furthermore, some of the track non-linearities such as the rail guidance (which bounds the levitation span) have been incorporated in the model. Also, the first part of the track is not equipped with levitation coils; and a disturbance torque on pitch due to aerodynamic drag forces has been also included.

$$T_{drag} = \frac{1}{2}\rho_{air}C_dAv^2d \tag{2.24}$$

In this equation ρ_{air} is the density of air, C_d the drag coefficient (which value is 2 for a flat plate), A is the area of the front of the vehicle, v the speed of the vehicle and d is the arm of the aerodynamic force.

2.4 State-Space Notation and Parameter Estimation

To identify the system using standard linear techniques a state-space notation must be developed. The parameters to be estimated are the field strength of the magnets (B) and the moments of inertia for pitch (J_p) and roll (J_r). If the model is accurate enough to describe the dynamics of the sled it must be possible to fit measured output data by estimating these parameters. The geometry of the sled, the geometry of the coils with respect to the magnets and the mass of the sled are assumed constant.

To use standard linear system identification techniques, all non-linearities must be transformed to inputs of the system. Therefore, all the currents are assumed to be inputs of the system. By doing so, neither the coil dynamics nor the corresponding non-linear terms are states of the system. The only equations necessary to describe the dynamics of the sled are the force equations for each magnet and the dynamic equations that link the magnet forces to the sled dynamics. For feedback control, it is sufficient to be able to describe the relationship between the currents in the coils and the dynamics of the sled. The coil dynamics can be neglected because current amplifiers are used for feedback control.

The state-space model is defined as:

$$\begin{aligned} \vec{x}(t) &= [z(t) \quad \dot{z}(t) \quad p(t) \quad \dot{p}(t) \quad r(t) \quad \dot{r}(t)]^T \\ \vec{u}(t) &= [i_{a,right}(t) \quad \dots \quad i_{j,right}(t) \quad i_{a,left}(t) \quad \dots \quad i_{j,left}(t)]^T \\ \dot{\vec{x}}(t) &= A\vec{x}(t) + B(t)\vec{u}(t) \\ y(t) &= C\vec{x}(t) + D\vec{u}(t) \end{aligned} \tag{2.25}$$

The forces acting on the system are linear functions of the currents in the levitation coils. When the currents become inputs of the system, the B-matrix of the state-space model contains all non-linear terms, since the forces generated by the currents depend on the x-position of the sled and the torques depend on the pitch and roll angles. The A-matrix of the state-space model only contains the integrant terms for each DOF. The C-matrix and D-matrix are straightforward. The outputs are the three DOF and there is no direct feed through so D is a matrix containing zeros.

The B-matrix is a 6x20 matrix. The rows 1, 3 and 5 are all zeros. The other terms are non-linear terms linking the coil currents to the acceleration for each DOF. The terms do not only contain non-linear expressions but also constant factors, which can be estimated when all the non-linearities are treated as inputs. So, for the estimation not the current itself is used as an input, but the current and its associated geometric non-linearity. The constant factors in the B-matrix are expressions of the field strength of the magnet that interacts with the coil divided by an inertia term. To illustrate the problem the terms linking coil C on the right side of the vehicle with the acceleration in each DOF are discussed.

$$\begin{aligned} B_{23} &= \frac{2B_2(\frac{1}{2}h + x)N}{M} \\ B_{43} &= \frac{2B_2(\frac{1}{2}h + x)N\cos(p)}{J_p} \\ B_{63} &= \frac{2B_2(\frac{1}{2}h + x)N\cos(r)}{J_r} \end{aligned} \tag{2.26}$$

The non-linear geometric term can be shifted to the input just as the number of coil windings and the factor of 2. This leads to the following expressions, which can be used for linear estimation.

$$\frac{B_2}{M}u_{r,c,z} \qquad (2.27)$$
$$\frac{B_2L_2}{J_p}u_{r,c,p}$$
$$\frac{B_2D_1}{J_r}u_{r,c,r}$$

It is necessary to define 60 inputs instead of 20 to take the non-linear terms for each DOF into account. For the example the inputs become:

$$\begin{aligned} u_{r,c,z} &= 2(\frac{1}{2}h + x)Ni_{r,c} \qquad (2.28)\\ u_{r,c,p} &= 2(\frac{1}{2}h + x)NL_2\cos(p)i_{r,c}\\ u_{r,c,r} &= 2(\frac{1}{2}h + x)ND_1\cos(r)i_{r,c} \end{aligned}$$

Some of the coils interact with two magnets. Then the corresponding term in the B-matrix depends on two expressions with different geometric non-linearities. In this case it is not possible to estimate the field strength and only the inertia term can be estimated. With this strategy, it is possible to estimate all the magnetic field strengths of all 12 permanent magnets and it is also possible to estimate the moments of inertia in pitch and roll. The mass of the vehicle, the number of coil windings and the geometric dimensions are assumed as exactly known.

The state-space model does not incorporate gravity. To add gravity as an acceleration in the estimation model, an additional input is created. Additional inputs are created to estimate the disturbances on pitch and roll due to unbalances in the vehicle. Finally, an extra input is added to estimate the aerodynamic disturbance on pitch.

Table 2.1: Estimated Parameters

Levitation		Pitch		Roll		Miscellaneous	
Left row	Right row	Left row	Right row	Left row	Right row		
$\frac{B_1}{M}$	$\frac{B_1}{M}$	$\frac{B_1L_1}{J_p}$	$\frac{B_1L_1}{J_p}$	$\frac{B_1D_2}{J_r}$	$\frac{B_1D_1}{J_r}$	dT_p	Pitch unbalance disturbance
$\frac{B_2}{M}$	$\frac{B_2}{M}$	$\frac{B_2L_2}{J_p}$	$\frac{B_2L_2}{J_p}$	$\frac{B_2D_2}{J_r}$	$\frac{B_2D_1}{J_r}$	dT_r	Roll unbalance disturbance
$\frac{B_3}{M}$	$\frac{B_3}{M}$	$\frac{B_3L_3}{J_p}$	$\frac{B_3L_3}{J_p}$	$\frac{B_3D_2}{J_r}$	$\frac{B_3D_1}{J_r}$	C_dAd	Pitch aerodynamic disturbace
$\frac{B_4}{M}$	$\frac{B_4}{M}$	$\frac{B_4L_4}{J_p}$	$\frac{B_4L_4}{J_p}$	$\frac{B_4D_2}{J_r}$	$\frac{B_4D_1}{J_r}$		
$\frac{B_5}{M}$	$\frac{B_5}{M}$	$\frac{B_5L_5}{J_p}$	$\frac{B_5L_5}{J_p}$	$\frac{B_5D_2}{J_r}$	$\frac{B_5D_1}{J_r}$		
$\frac{B_6}{M}$	$\frac{B_6}{M}$	$\frac{B_6L_6}{J_p}$	$\frac{B_6L_6}{J_p}$	$\frac{B_6D_2}{J_r}$	$\frac{B_6D_1}{J_r}$		

The estimation technique used is the prediction-error method (pem) as described in [7]. The algorithm estimates the unknown parameters of a state-space structure by minimizing the quadratic

norm of the prediction error using a gradient-descent method (iterative damped Gauss-Newton method). The technique allows to fix parameters that are considered robust and to estimate the others. See table 2.1 for a list of parameters to be estimated.

Four different sets of equations are necessary to describe the force acting on the sled. Therefore, there are four different B-matrices having the same structure but different non-linear terms. The estimation problem must be divided into four different problems to deal with this. Each sub-problem represents a small section with a maximum length of $(\frac{1}{2}h - d)$. With the sled travelling at relative high speed (15 [m/s]), a high sampling frequency is required to obtain enough data points for the estimation process. Using high-speed data-acquisition cards sampling frequencies as high as 65 [kHz] per channel can be achieved. This provides sufficient data points to get a well-defined estimation problem. The matlab m-file used for the parameter estimation of the first set of equations is shown in appendix C.

2.5 Model Implementation

The model as described in section 2.3 has been implemented in Matlab/Simulink as a user-defined C-code s-function using the ode1 fixed-step solver with a stepsize of 2.5e-4 [s]. Model outputs are levitation height, pitch, roll and levitation coil currents. Because of the complexity of the model, only the current of one coil at a time is displayed. The only input to the simulation is the speed during launch. A constant acceleration profile is used in the simulation but the velocity function can be chosen arbitrarily. The velocity increases linearly until the maximum velocity is reached. Some of the model parameters (see table 2.2) have been estimated as accurate as possible, but most estimates must be redone using real-time measurements of the launch[1]. The initial vertical position of the sled is -12.5 mm below the null-flux axis of the coils. The C-code of the s-function is included in appendix B.

Table 2.2: Estimates of model parameters

Symbol	Description	Value		Symbol	Description	Value	
R	coil resistance	0.638	[Ohm]	dT_p	pitch disturbance	0.01	[Nm]
L	coil inductance	0.837e-3	[H]	dT_r	roll disturbance	0.01	[Nm]
N	coil windings	100	[-]	L_1	magnet 1 distance	0.1905	[m]
d	gap width	0.6e-2	[m]	L_2	magnet 2 distance	0.1143	[m]
b	coil width	4.48e-2	[m]	L_3	magnet 3 distance	0.0381	[m]
h	magnet width	5.08e-2	[m]	L_4	magnet 4 distance	-0.0381	[m]
B	field strength	1.25	[T]	L_5	magnet 5 distance	-0.1143	[m]
M	sled mass	6.0	[kg]	L_6	magnet 6 distance	-0.1905	[m]
J_p	inertia pitch	0.01437	[kgm^2]	D_1	right row distace	0.0386	[m]
J_r	inertia roll	0.0288	[kgm^2]	D_2	left row distance	-0.0386	[m]

[1]The magnets are placed as: $B_1 = B$, $B_2 = -B$, $B_3 = B$, $B_4 = -B$,$B_5 = B$,$B_6 = -B$

Chapter 3

Model Simulation Results

Several simulations are done with the simulation model described in section 2.3 and 2.5. For each simulation the maximum velocity along the guideway is changed. In this chapter the dynamic behavior of the sled is discussed using the simulation results.

3.1 Sled Dynamics

As seen in figure 3.1 and 3.2 the dynamic behavior of the levitation height is highly underdamped and can be unstable beyond certain speed. The frequency of the vibration increases as the maximum speed of the sled increases. This behavior can be explained by the fact that the time needed for a magnet to pass a coil decreases as the speed increases. Earlier studies using frequency domain approximations ([1],[3],[4]) concluded that the lowest frequency in the system was related to the time needed for a magnet to pass a coil.

As the maximum velocity increases the lift forces acting on the sled increase too. The sled does not completely levitate at 15 and 20 [m/s]. When the sled hits the rail guidance during launch the s-function will keep the sled inside the boundaries of the guidance but the collision of the wheels of the sled with the guidance is hard to model. This results in a discontinuity in levitation height, pitch and roll. The model is not valid when a guidance wheel hits the guidance rail and the levitation, pitch and roll can behave in a non-realistic manner. A good example is the roll at 30 [m/s] in figure 3.6. Nevertheless the dynamic behavior is realistic and agrees with earlier research.

The mean value of both the pitch and roll is not equal to zero. In case of the roll, this is fully caused by the disturbance due to the unbalance of the sled. If the disturbance in the model is removed, the roll remains zero during the launch. This also indicates that the roll dynamics is not influenced by either vertical or pitch dynamics. The pitch, on the other hand, tends to be negative during flight (the front of the sled is lower than the back). This indicates that the pitch dynamics is not independent of the vertical dynamics.

3.2 Coil Currents

In addition, the currents in the levitation coils are calculated during simulation. The currents in the first and second coils, when the six magnets of one side of the sled pass, are shown in figure 3.7. The acceleration part of each velocity profile is the same so the currents in the first two coils are the same for all simulation results.

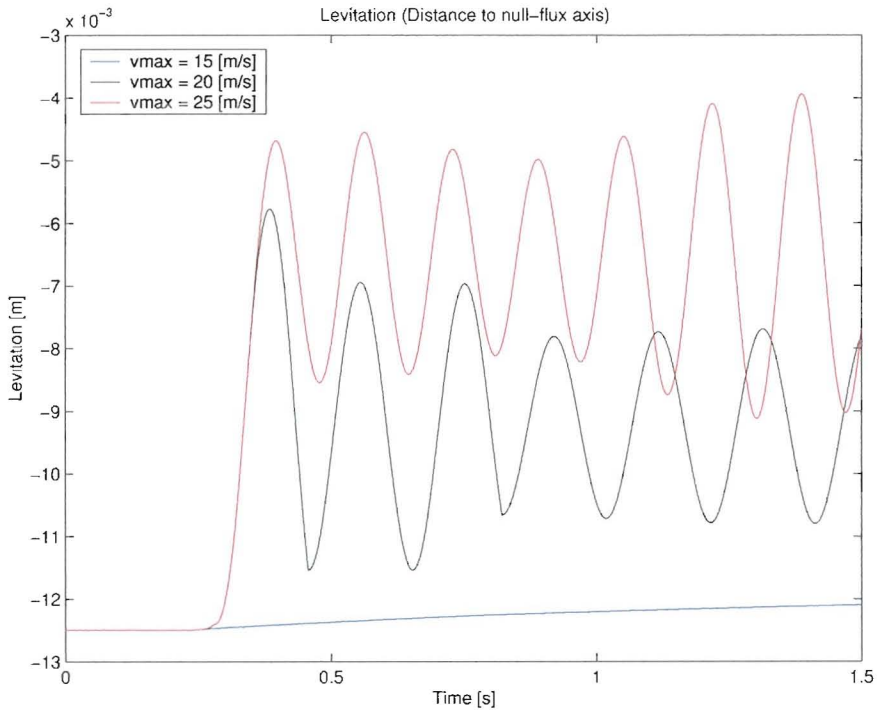

Figure 3.1: Levitation height 3-DOF model

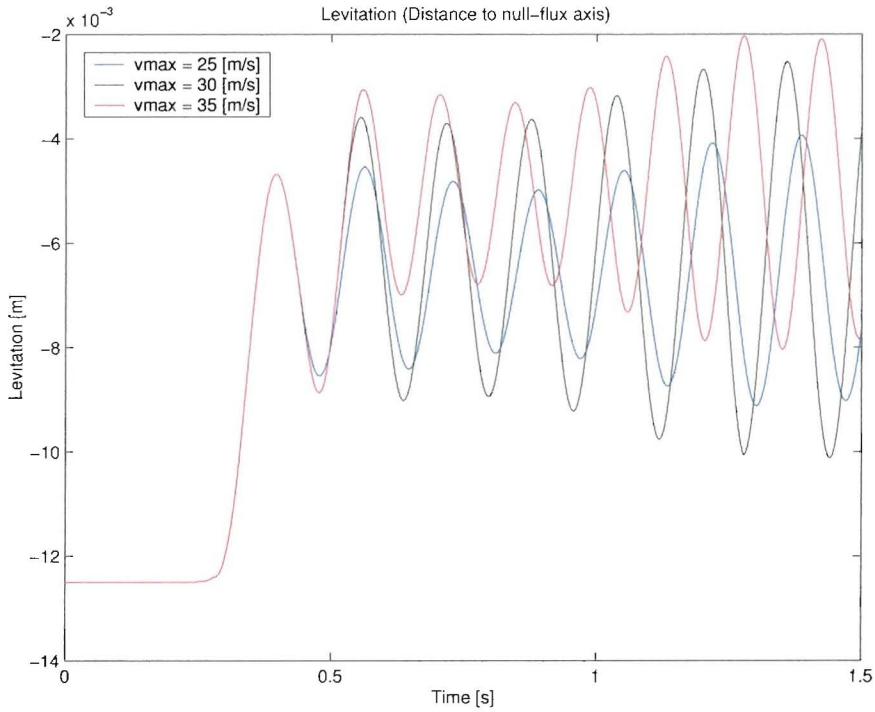

Figure 3.2: Levitation height 3-DOF model

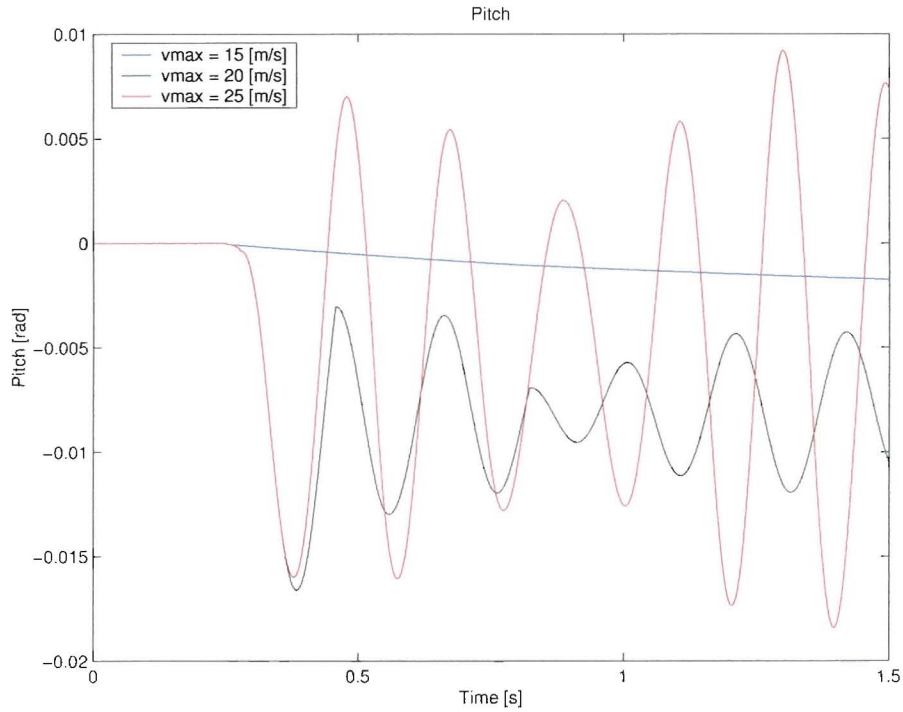

Figure 3.3: Pitch 3-DOF model

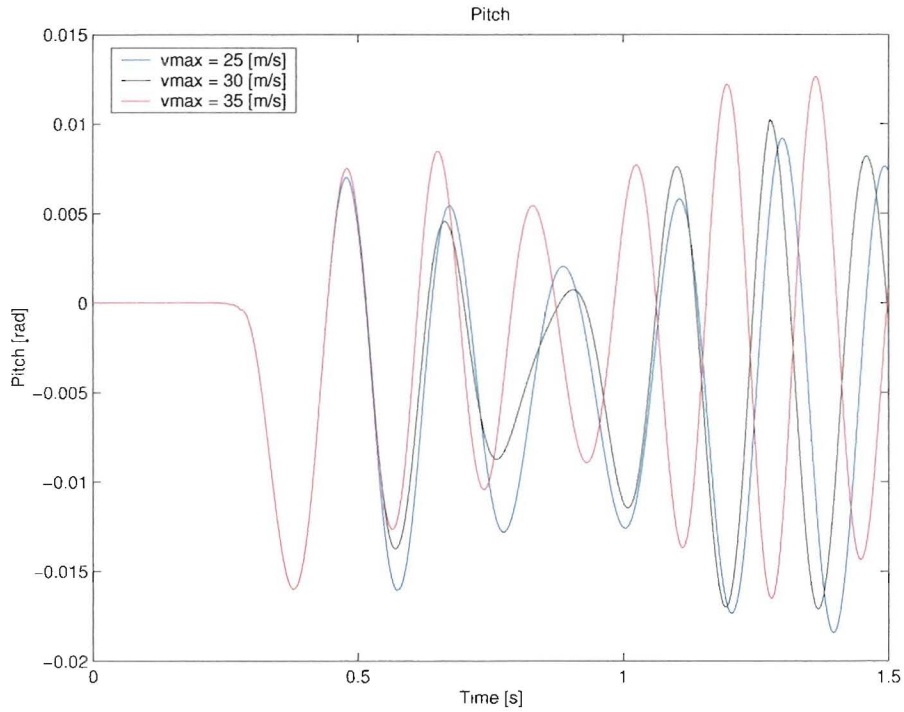

Figure 3.4: Pitch 3-DOF model

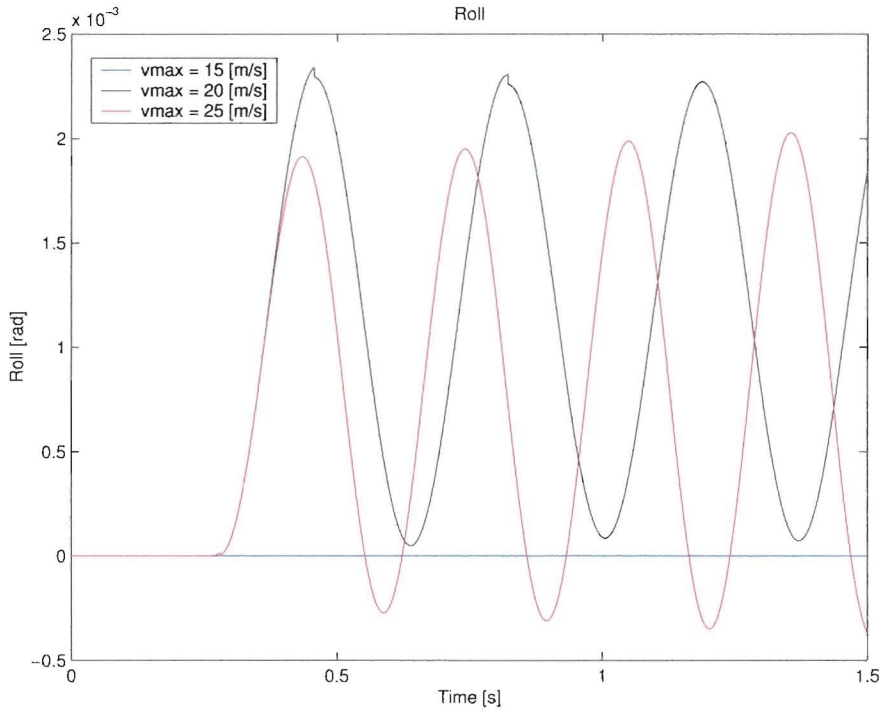

Figure 3.5: Roll 3-DOF model

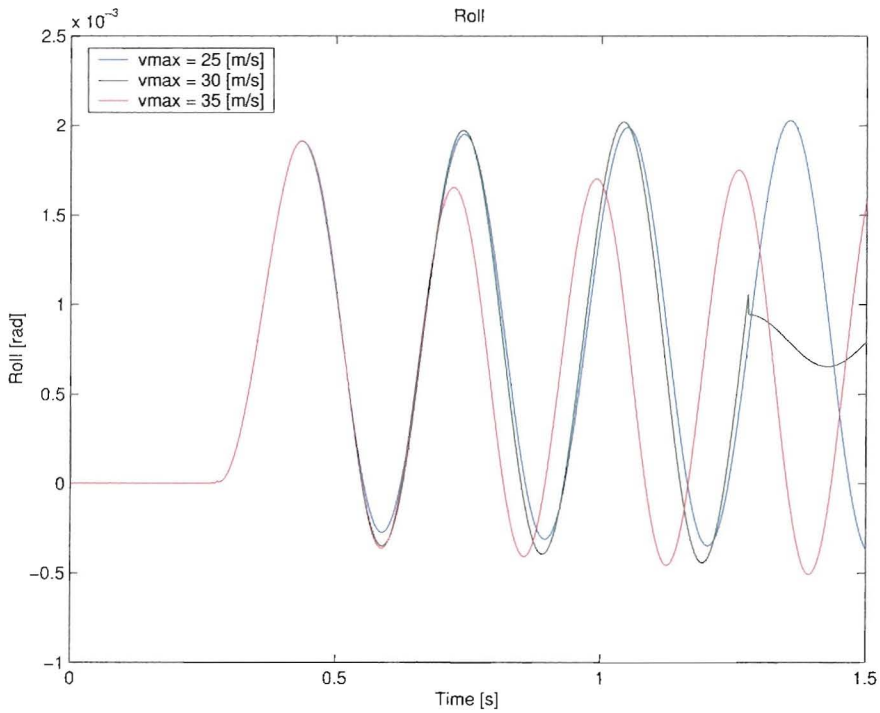

Figure 3.6: Roll 3-DOF model

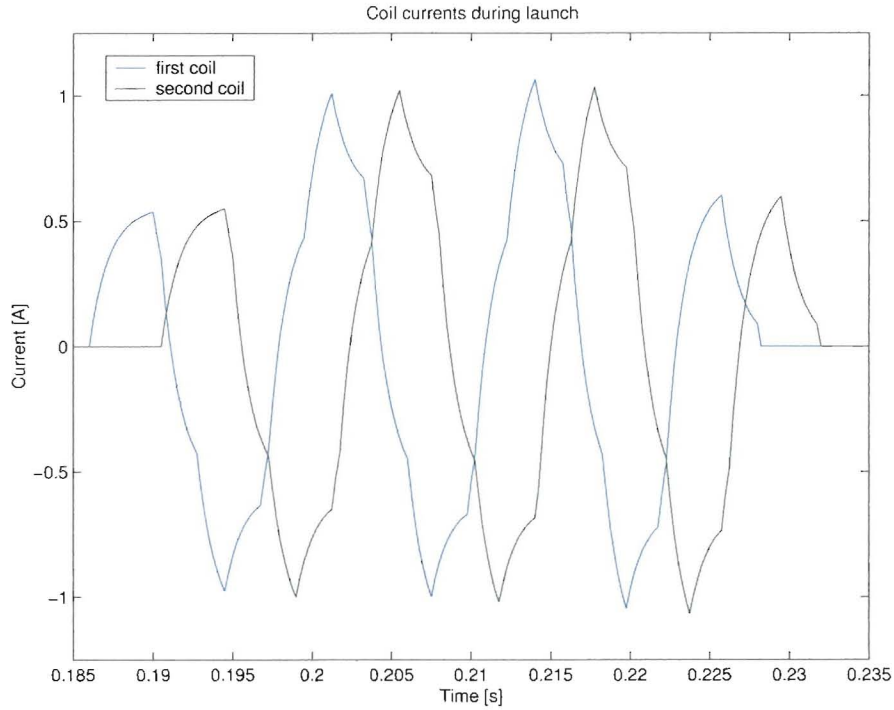

Figure 3.7: Current in coil one and two

Chapter 4

Model Validation

The simulation results agree with the physical insight we have of the system. However, in order to take the next step, active control of the figure-8 coils, this model must be validated. The interaction between the magnets and the coil and the prediction of the coil currents must be accurate enough to design and implement a controller. In addition, some of the parameters in the model such as the moments of inertias and weight unbalance need to be estimated.

4.1 Experimental Setup

A schematic of the experiment is included in figure 4.1 to describe the experiment.

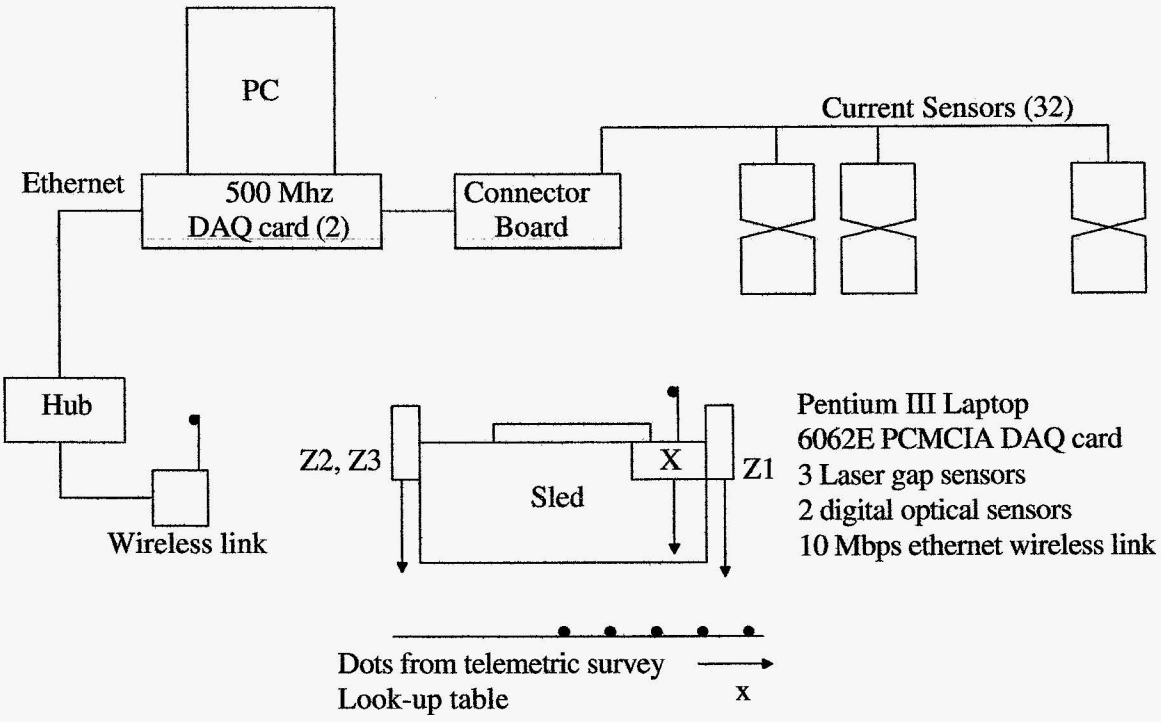

Figure 4.1: Schematic of model verification experiment

A segment of the track, towards the end of the propulsion section, has been instrumented with magneto-restrictive current sensors. During launch, a digital optical sensor on each side of the

sled counts the dots on the track. These dots are from a telemetric survey and a look-up table is used to determine the x position and speed and to correct the z position of the sled. In order to detect a dot, it needs to be marked with high reflective tape. Not all dots are marked but only the dots in the same section as the coils that are connected to the current sensors. The dots are spaced 1 inch (0.0254 [m]) apart and a total of 92 dots are marked, 46 on each side .

On three points of the sled laser gap sensors measure the height. The three measurements are used to compute the levitation height, pitch and roll of the sled. The currents of 32 coils, 16 coils on each side of the track opposite to each other, will be measured during launch. The measurement of the coil currents will be synchronized with the measurements on the sled. The coils are located in the part of the track, where the sled reaches it maximum speed.

4.1.1 Limitations of the Experimental Setup

The resolution of the position measurement cannot be improved because of the high speed of the sled and the large gap between the sensor on the sled and the top of the track, due to the levitation. In addition, the high reflective tape is put on manually and is therefore compromising accuracy. In order to get accurate position data for model validation, it would be desirable to have a factor 10 higher accuracy (one tenth of an inch).

All sensors are are powered by the battery of the laptop on top of the sled. The ground of the sensors is connected to the minus of the battery. The ground of the battery is very noisy and that noise is affecting the output of the laser gap sensors. Therefore, the output signal of the height sensors is very noisy which is greatly affecting the accuracy of the height measurement.

To calculate the height, pitch and roll, using the three height sensors, geometric relations are used. The geometric parameters used for the calculations are hard to measure accurately, especially the distance of the top of the track to the geometric null-flux axis of the coils.
The consequences of these limitations are discussed later in this chapter.

4.2 Coil Current Measurements

First some simple experiments are done to verify small parts of the model or certain assumptions. By comparing the induced currents with the model the flux and voltage equations can be validated. Therefore, the sled is pushed passed a coil by hand. The sled does not levitate but has a constant offset to the null-flux axis (lower boundary of the guidance rail). The coil current is measured and compared with a simulated current. Unfortunately it is not possible to measure speed accurately when pushed by hand. Therefore, the amplitude of the currents can not be compared, only the shape of the current graphs. The measured current is shown in figure 4.3 and the simulated current in figure 4.2.

The shape of the measured current is very similar to the shape of the simulated current. This is a good indication that the flux equations and voltage equations are valid. The amplitude of the measured current (0.1 [V] equals 0.6 [A]) differs from the simulated current. This is due to a difference in speed and parameter estimation errors. These problems can be solved using the parameter estimation technique described in chapter 2.

Another assumption made is that the mutual induction of the figure-8 coils is negligible. This assumption has been verified by FEM analysis at NASA [1]. A simple way to verify this experimentally is to induce a current in one coil by moving a magnet vertically and measure the currents in that coil end the coils next to it. The result of this experiment is displayed in figure 4.4. The assumption that the mutual inductance is negligible seems to be correct since the voltage signal

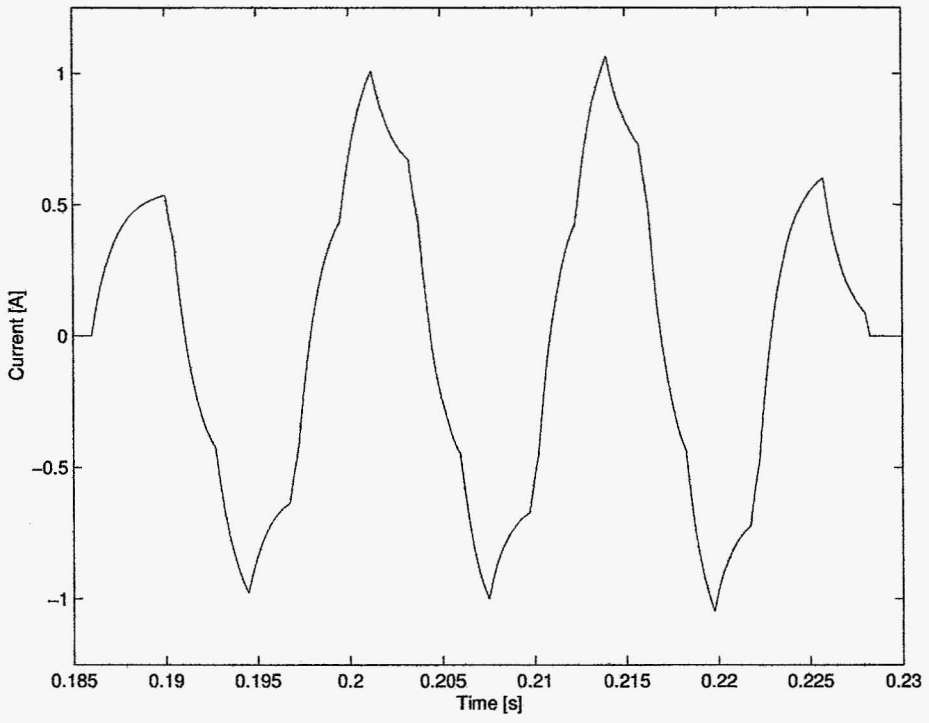

Figure 4.2: Simulated current induced by passing sled

from the current sensors of the coils does not exceed the noise level of the sensor, except for the coil that is excited by the magnet.

These results are quite promising but the most important part of the model has still not been validated. The equations that describe the forces on the sled and its dynamics can only be validated by parameter estimation of the state-space model.

4.3 Validation of the State-Space Model

To estimate the parameters of the state-space model, simultaneous measurements of the sled dynamics and the coil currents are necessary. The sampling frequency to capture the data is 65 [kHz].

First the raw data from the laser gap sensors is analyzed. Laser sensor z_1 is located at the front of the sled on the right side, z_2 at the back of the sled on the right side and z_3 at the back of the sled on the left. The measurement range of the sensors is not the same for each sensor so the measurements can not be compared as absolute values. The launch starts at 0.33 [s]. The front of the sled (z_1) starts levitating at 0.8 [s]. The vibration as predicted in the simulations is clearly visible for the front of the sled. The back of the sled (z_2 and z_3) is not lifted. Although a vibration with a small amplitude is visible for sensors z_2 and z_3, the amplitude of the signal is not much higher than the amplitude of the signal in the first part of the launch (0.4-0.7 [s]), when there is no levitation at all. The changes can easily be caused by the turning of the sled (pitch), since the front is levitating without a doubt. The braking section is reached at 1.23 [s]. The front of the sled hits the bottom of the guidance rail and the back of the sled is bumping up. At 2.7 [s] the sled comes to a complete stop. The noise on the sensor signals is quite substantial. Especially when the voltages of the laser sensors are converted to levitation, pitch and roll as shown in figure 4.6. Therefore, the noise problem must be solved and the measurements need to be redone to improve its accuracy.

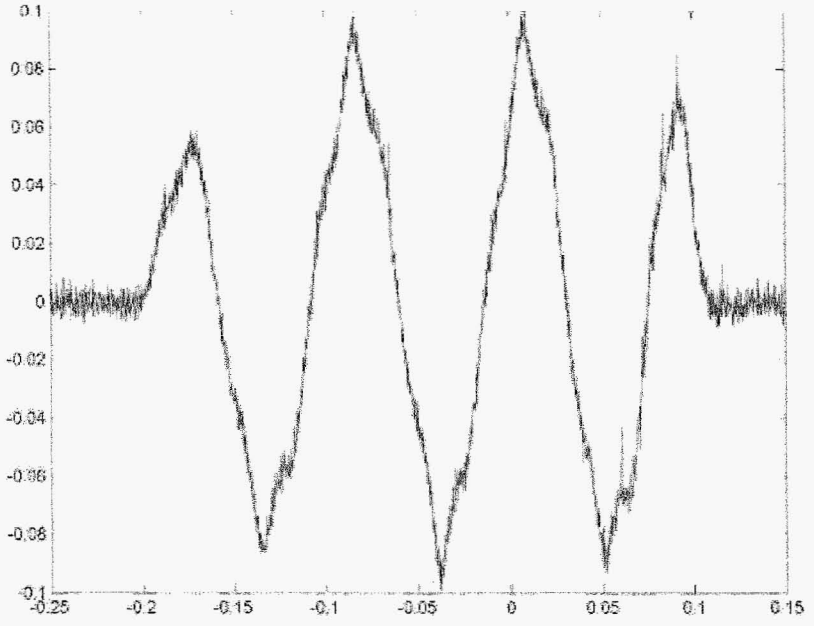

Figure 4.3: Measured current induced by passing sled (x-axis: time [s], y-axis: voltage [V])

To check whether the sled is really levitating the profile of lower guidance rails is measured in the measurement section of 1.2 [m]. Then the height measurement of each sensor during flight is compared with the guidance profile measurement. The result is displayed in figure 4.7. This figure clearly shows that the front of the sled (z_1) is levitating but the back of the sled is not (z_2 and z_3). This is a serious problem, because the model is only valid when the sled is not hitting the guidance rails. Collisions are hard to model and are not part of the state-space model.

The data in figure 4.8, 4.9 and 4.10 displays the measurement section (1.2 [m]). For the parameter estimation only a small part of the data is necessary ($\frac{1}{2}h - d = 0.019$[m]). Not only the dynamics of the sled are measured but also the x-position of the sled on the track. The marked dots that are detected during launch are displayed in figure 4.11. The line is a first order fit through the detected dots. The slope of the line is equal to the speed of the sled which is 15.3 [m/s].

The currents of the levitation coils are also measured in that section. The currents of each side of the track are displayed in figure 4.12 and 4.13. In figure 4.14 the current of coil 4 on the left row is displayed. The shape has altered a because of the dynamics of the sled. The front of the sled is levitated and closer to the null-flux axis. The induced current is therefore smaller. Then the sled passes and the magnets are further from the null-flux axis and the induced current amplitude increases. The sharp edges of the induced current as in figure 4.3 and 4.2 disappear due to the changes in levitation height, pitch and roll of the sled as it passes the magnet. The amplitude of the current is much higher as in the simulation.

As discussed earlier the data of the height sensors is very noisy. Therefore the noise will have a huge influence in that small dataset needed for the estimation. To illustrate this problem a dataset to estimate the parameters of the first set of equations is displayed in figure 4.15, 4.16 and 4.17.

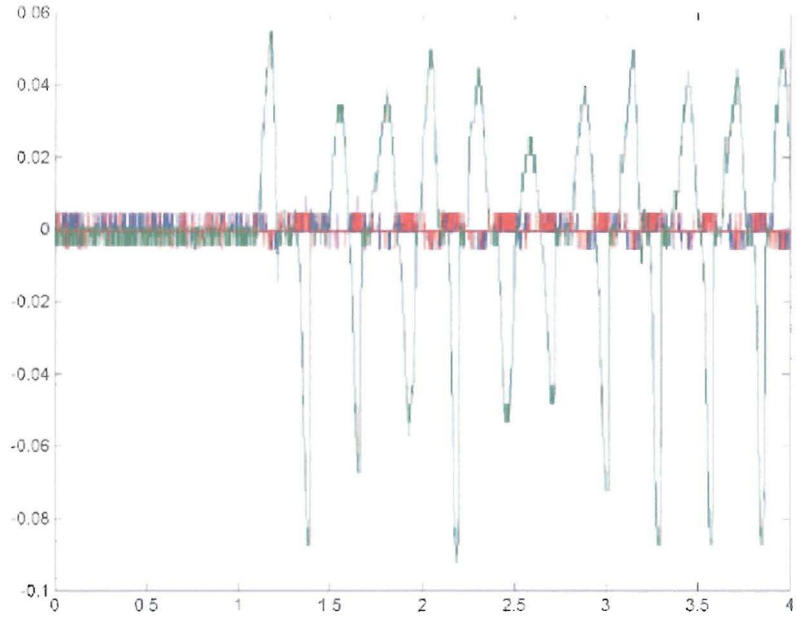

Figure 4.4: Measured current in three coils (x-axis: time [s], y-axis: voltage [V])

When this dataset is used for the parameter estimation routine the estimated parameters do not make sense. An option to improve the data is to fit a spline through the levitation, pitch and roll measurements and then cut out the section for the parameter estimation. This approach has the same problem and does not lead to realistic parameters.

With this data it will not be possible to estimate the parameters. Several launches have been done but all the datasets had the same problem. In order to verify whether the model is valid some problems need to be solved. This is discussed in chapter 7.

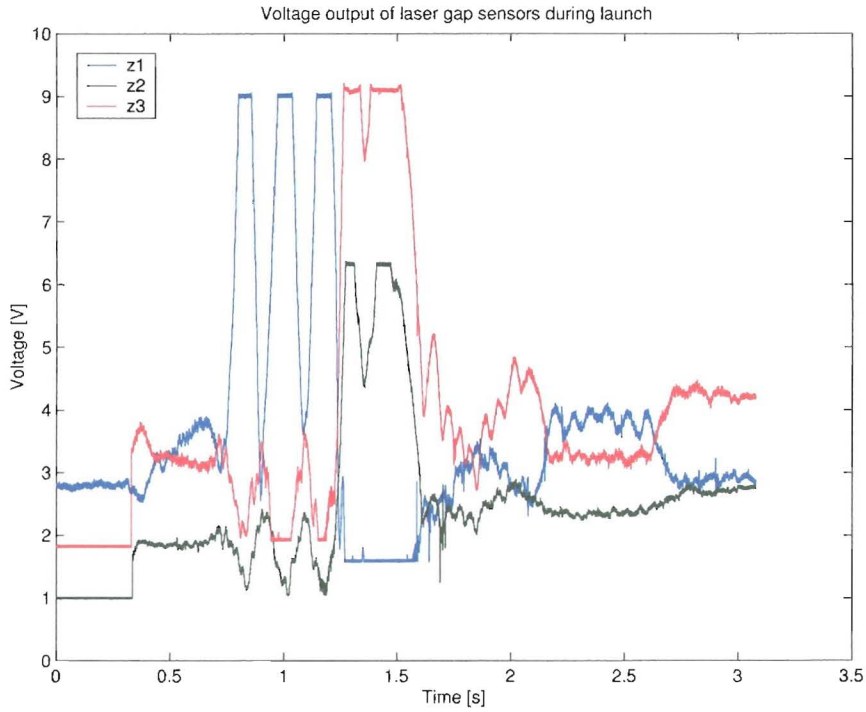

Figure 4.5: Voltage output of the laser gap sensors during launch

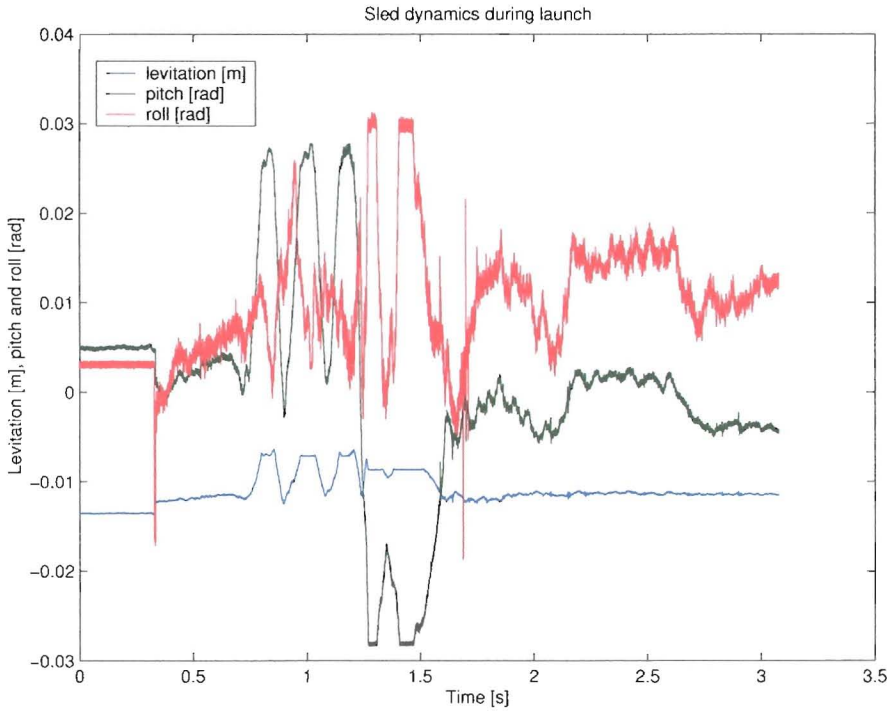

Figure 4.6: Dynamics of the sled during launch

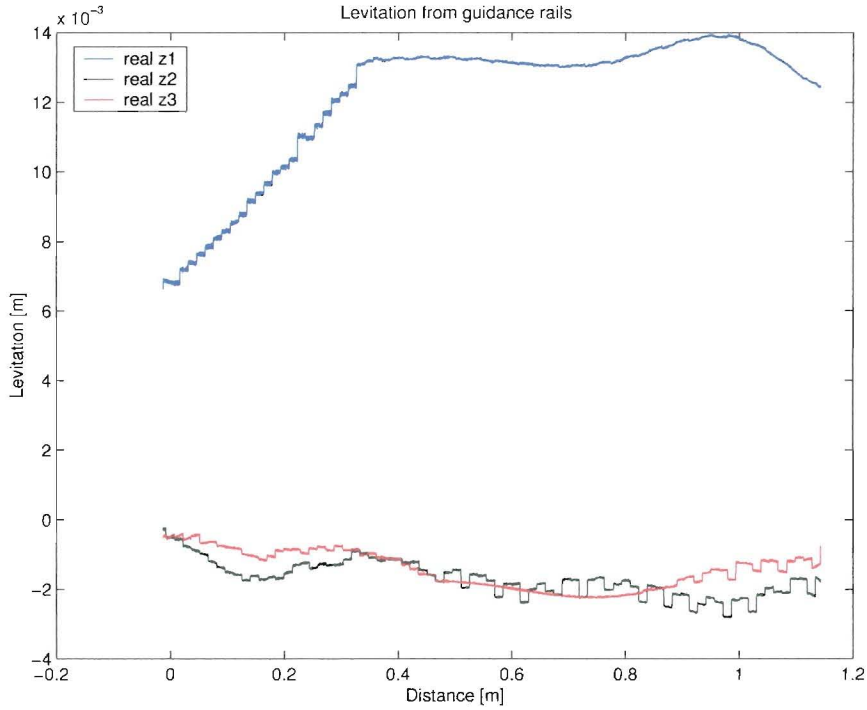

Figure 4.7: Height of each sensor with respect to the lower guidance rails

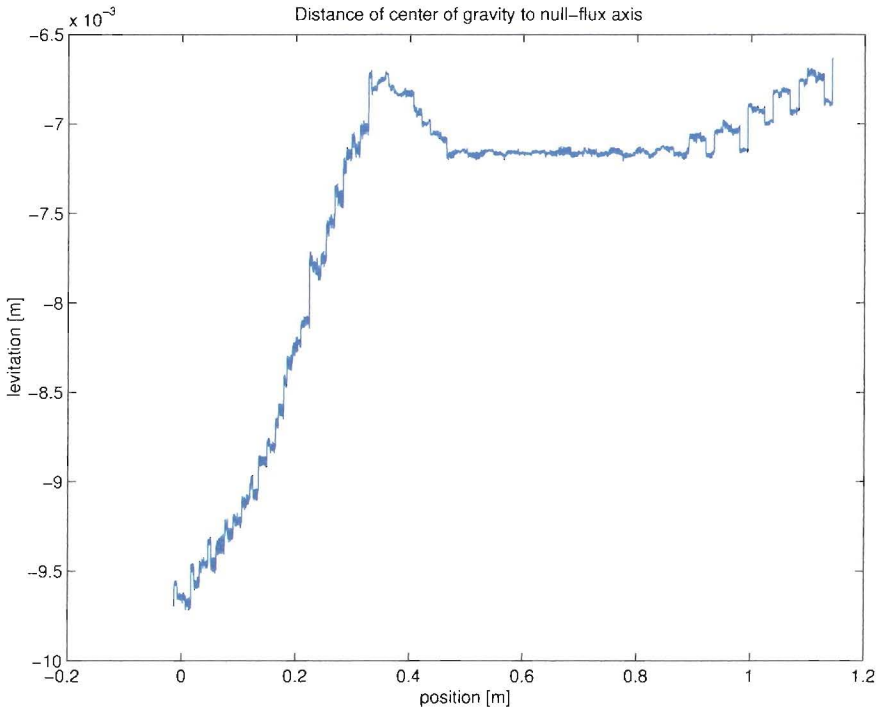

Figure 4.8: Levitation of the sled in the measurement section

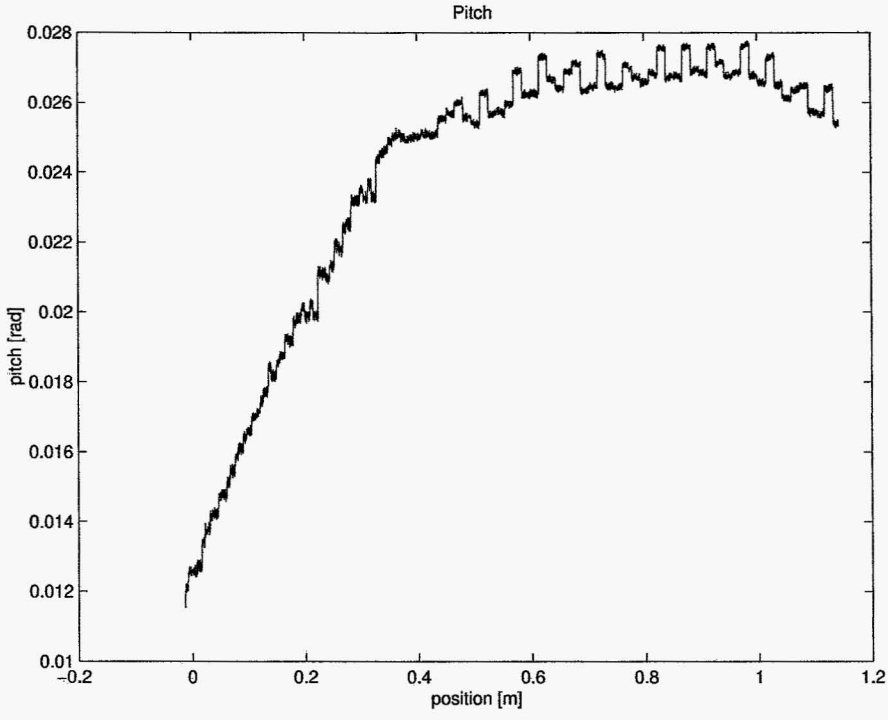

Figure 4.9: Pitch of the sled in the measurement section

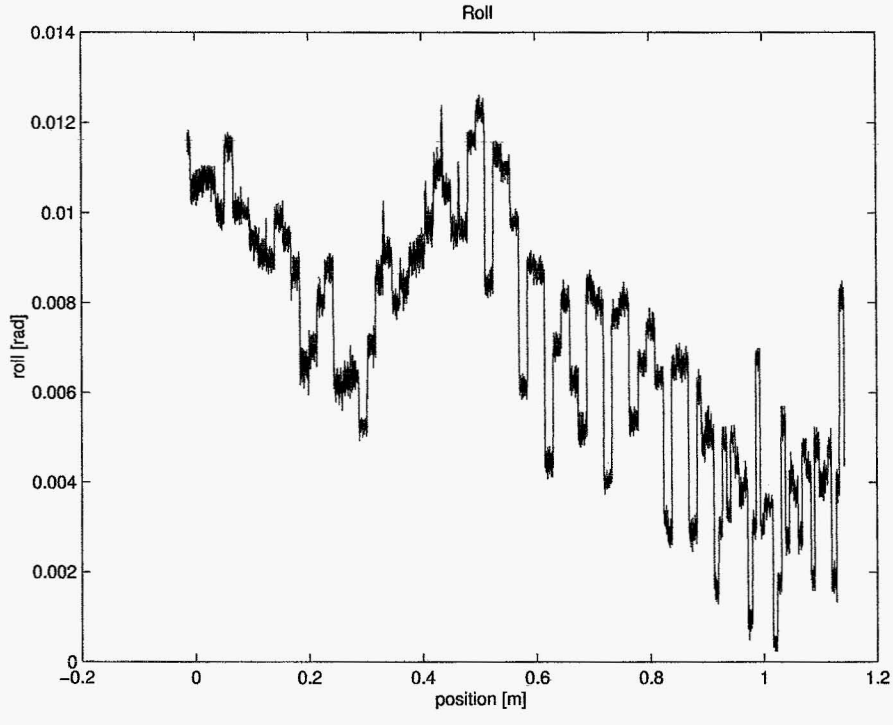

Figure 4.10: Roll of the sled in the measurement section

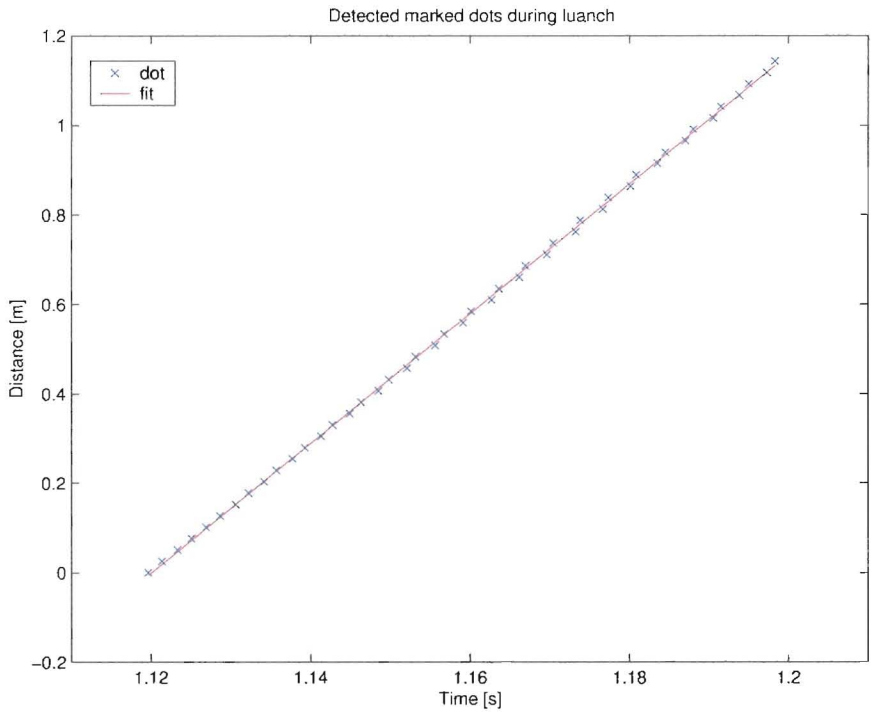

Figure 4.11: Levitation of the sled in the measurement section

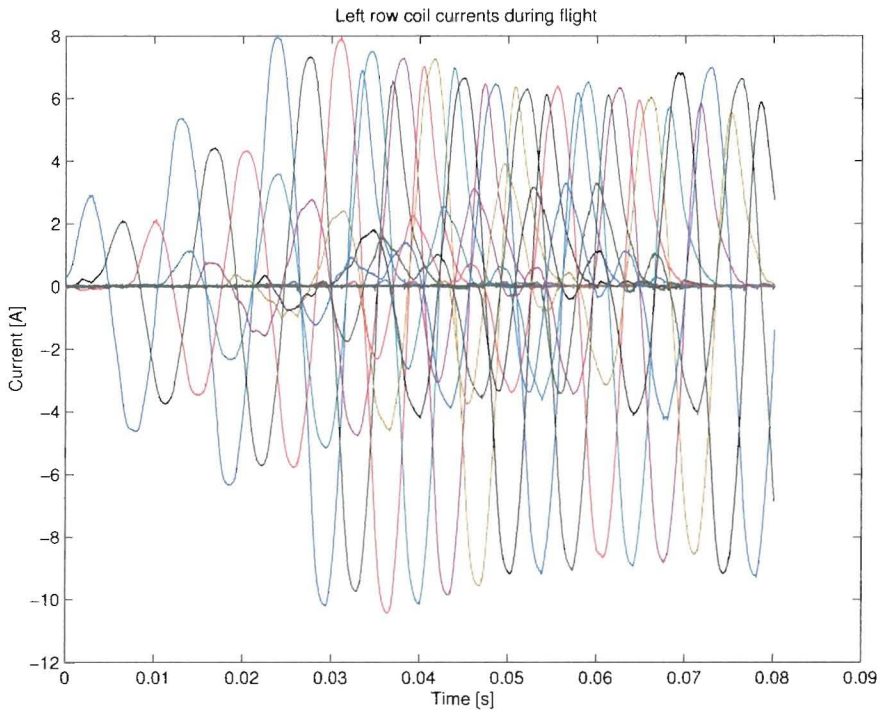

Figure 4.12: Left row coil currents during launch

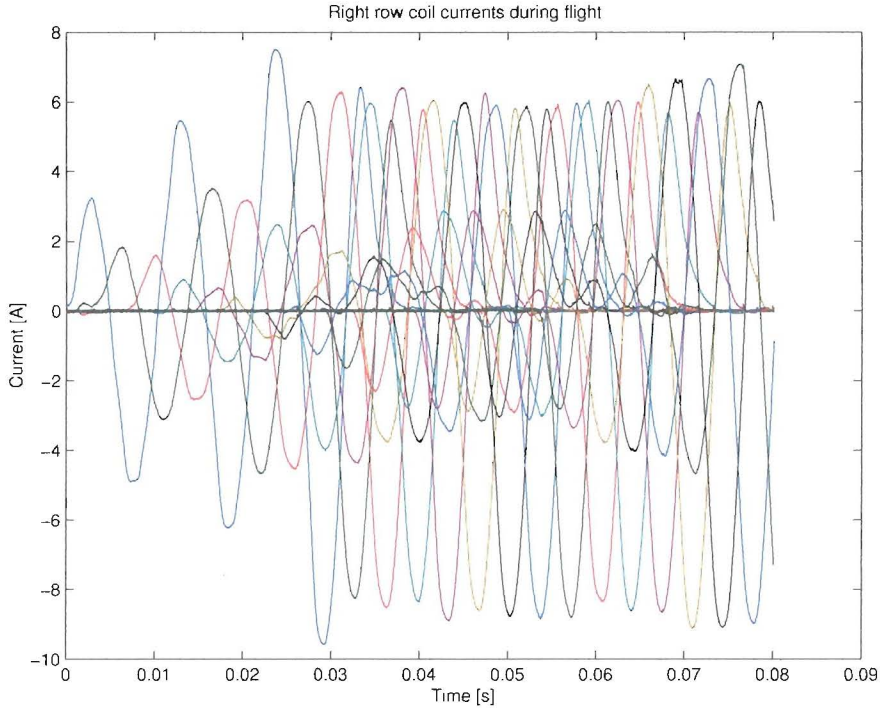

Figure 4.13: Right row coil currents during launch

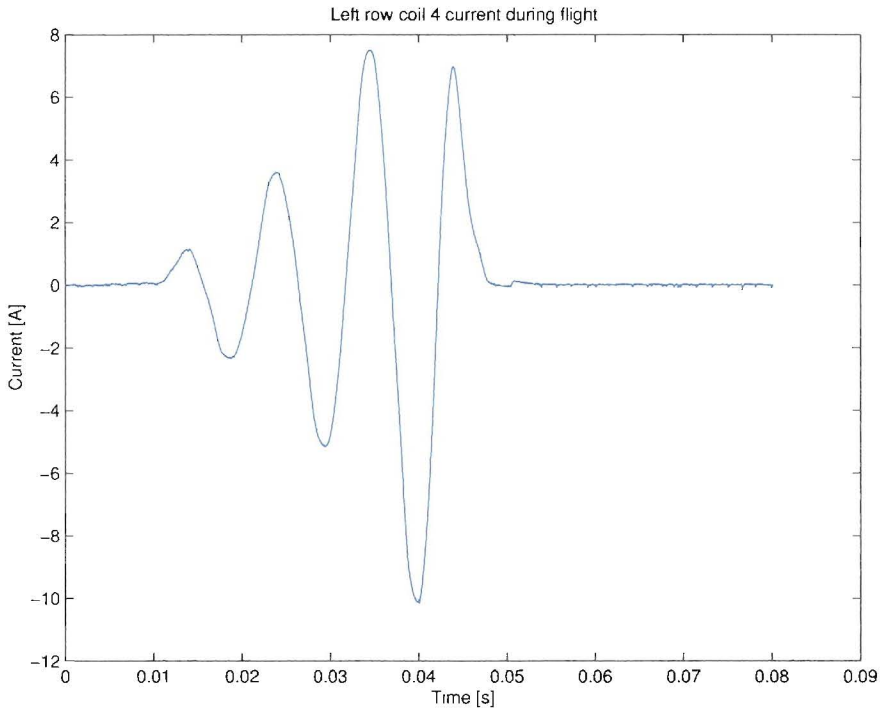

Figure 4.14: Current of coil 4 on the left row

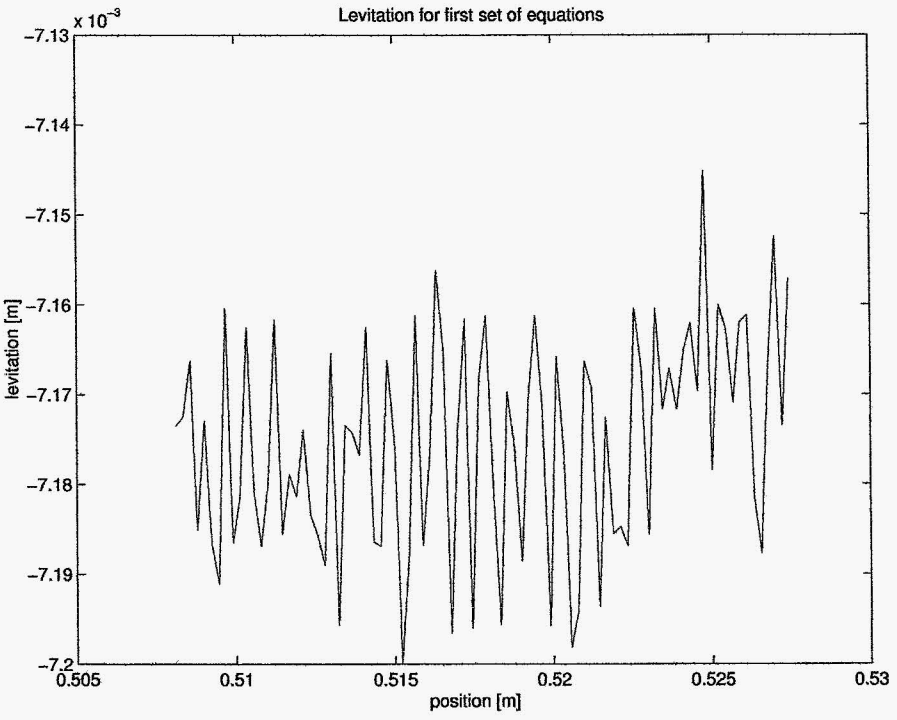

Figure 4.15: Levitation of the sled in the estimation section

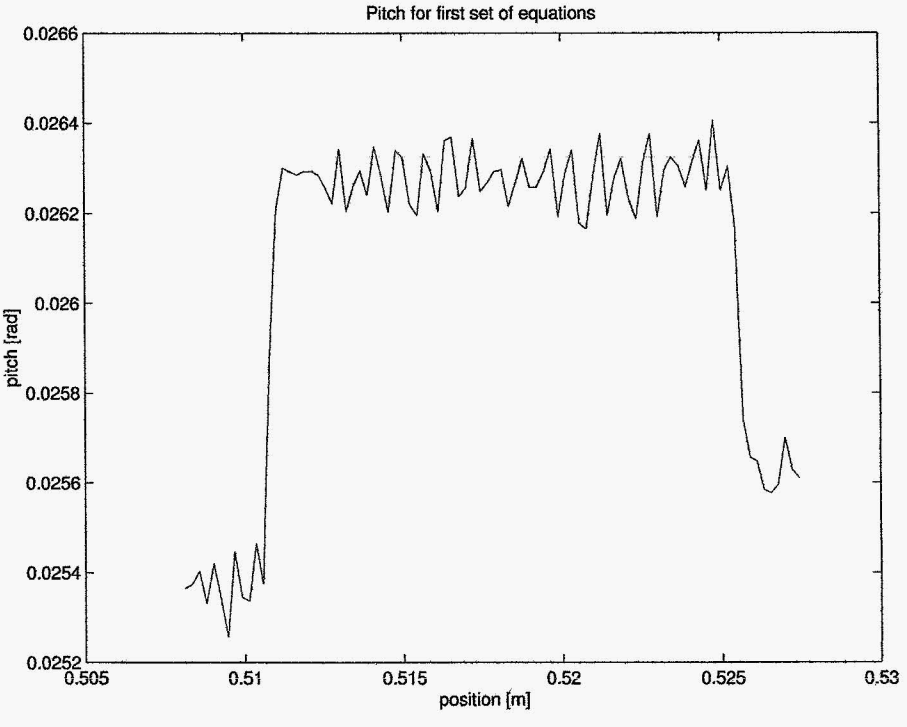

Figure 4.16: Pitch of the sled in the estimation section

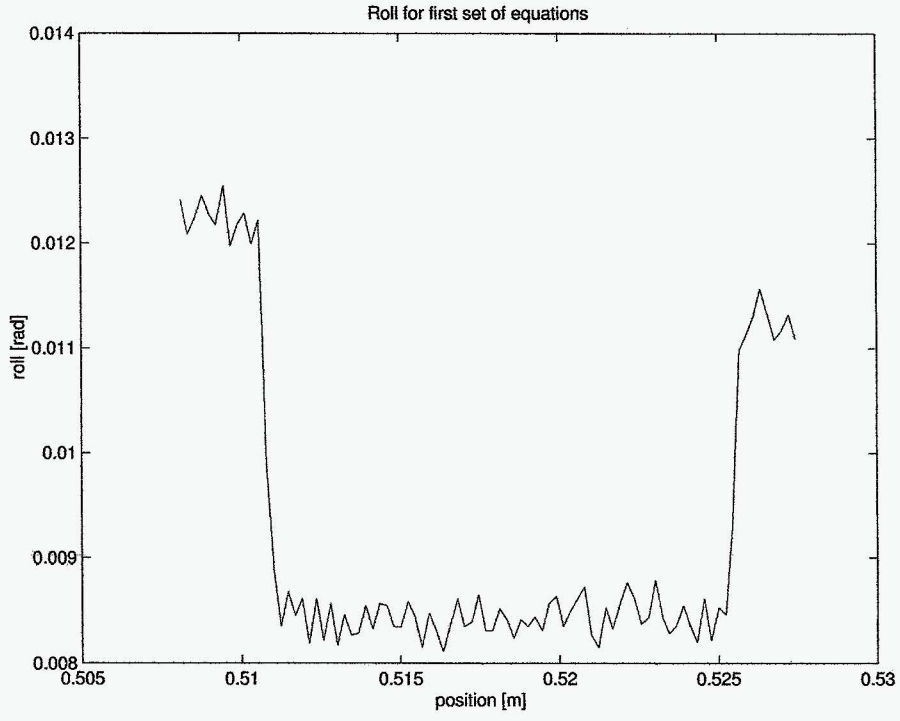

Figure 4.17: Roll of the sled in the estimation section

Chapter 5

Feedback Control of 3-DOF Model

Although the model is not completely verified the possibility of controlling such a complex system is studied. Before the development of a controller is started the B-matrix of the state-space studied more thoroughly.

5.1 Input Mapping

The B-matrix has a complicated structure. It is a 6x20 matrix containing all the non-linear dynamics. Not only it depends on the horizontal position (and therefore on time), but also on the state itself because the torque on pitch and roll depends on the cosine of these angles. Another interesting property of the B-matrix is that each element tells what influence each input has on each state. The sign of the element determine whether it has positive or negative influence on the particular state. Its value shows how much influence that particular input has on the corresponding state. This property is used to map the 20 input currents from 3 inputs for each DOF. Furthermore all forces depend linearly on the current in the coils. This implies that the state space notation does not need to be linearized with respect to $\vec{u}$. During launch all 3 DOF will be measured real-time. Therefore it is assumed that all 3 DOF are known during flight and that its derivatives must be estimated.

To simplify the controller design it is necessary to reduce the number of inputs to three. In order to do so a linear combination of inputs is made for each DOF. If all actuators had the same effect for all time it could be done by using all 20 coils for levitation (all with the same sign). The front coils (positive sign) and the back coils (negative sign) could control the pitch and the left (positive sign) and the right coils (negative sign) for roll. Unfortunately it is not this simple because the interaction of each coil with the sled is time dependent. Therefore the B-matrix is used to distribute the controller output for each DOF.

$$\begin{aligned} \vec{u}(t) &= T(t)\vec{v}(t) \\ \dot{\vec{x}}(t) &= A\vec{x}(t) + B(\vec{x}(t),t)T(\vec{x}(t),t)\vec{v}(t) \\ \dot{\vec{x}}(t) &= A\vec{x}(t) + \hat{B}(\vec{x}(t),t)\vec{v}(t) \end{aligned} \tag{5.1}$$

To scale the control problem properly it is desirable for the new matrix $\hat{B}$ to have ones on the diagonal. To obtain this form the transform matrix T must have the following form

$$T = \begin{bmatrix} \frac{B_2^T}{||B_2||^2} & \frac{B_4^T}{||B_4||^2} & \frac{B_6^T}{||B_6||^2} \end{bmatrix} \tag{5.2}$$

In this matrix B_2, B_4 and B_6 are the rows of matrix B that correspond with the height, pitch and roll acceleration respectively. When $\hat{B}$ is calculated the diagonal terms are indeed equal to 1

and all other terms are equal to zero, except for the cross terms corresponding with height and pitch. These terms are smaller than one and vary in time but have known bounds.

Because the matrix B is time dependent, so is the mapping matrix T. This implies that every time a controller output is calculated the matrix T must be calculated to map the controller outputs.

5.2 Linear Control

The possibility to design a linear robust H_∞ controller is studied. For this control strategy the input mapping is used. To compute the controller, first the states of the state-space model are rearranged so that

$$\hat{B} = \begin{bmatrix} O \\ \hat{H} \end{bmatrix} \tag{5.3}$$

with $\hat{H}$ a square positive definite matrix. This changes the state-space notation into:

$$\begin{aligned} \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} &= \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} O \\ \hat{H} \end{bmatrix} \vec{v} \\ x_1 &= [z \quad p \quad r]^T \\ x_2 &= [\dot{z} \quad \dot{p} \quad \dot{r}]^T \end{aligned} \tag{5.4}$$

All terms of the A-matrix are zero except A_{12} which has ones on the diagonal. In order to design a robust controller, first the model error is studied. In general the plant has the following state-space representation, using the input mapping discussed before.

$$\begin{aligned} \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} &= \begin{bmatrix} O & I \\ O & O \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} O \\ \hat{H} \end{bmatrix} \vec{v} \\ \vec{y} &= C\vec{x} + O\vec{v} \end{aligned} \tag{5.5}$$

The model errors are located in matrix $\hat{H}$ not in matrix A, C or D. The matrix $\hat{H}$ of the nominal plant is defined as:

$$\hat{H}_n = \begin{bmatrix} 1 & c_1 & 0 \\ c_2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.6}$$

The constants c_1 and c_2 are in reality time-dependent but have known bounds. The absolute value is less than one but the sign can be either positive or negative. In time the real plant changes and $\hat{H}$ becomes:

$$\hat{H}_r = \begin{bmatrix} 1 & c_1 + dc_1 & 0 \\ c_2 + dc_2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.7}$$

Using an additive uncertainty to describe the perturbation, the error plant is defined by the model of the nominal plant parallel with minus the real plant.

$$\begin{aligned} A_e &= \begin{bmatrix} A_n & O \\ O & A_r \end{bmatrix} \\ B_e &= \begin{bmatrix} O & \hat{H}_n & O & \hat{H}_r \end{bmatrix}^T \\ C_e &= [C \quad -C] \\ D_e &= O \end{aligned} \tag{5.8}$$

The frequency response function of the error plant can now be computed.

$$P_e = C_e(sI - A_e)^{-1}B_e + D_e = \begin{bmatrix} 0 & \frac{-dc_1}{s^2} & 0 \\ \frac{-dc_1}{s^2} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{5.9}$$

The error is not bounded and goes to infinity when s goes to zero. Furthermore, parameters dc_1 and dc_2 are not small but only one order smaller than the diagonal terms. Therefore, an additive uncertainty approach to design a robust controller that is capable of the performance requested for this problem is not the right approach. Another approach is to describe the model perturbations using a multiplicative uncertainty. However, regarding the large variations in $\hat{H}$, a non-linear controller is the best option to consider.

5.3 Sliding Mode Control

The control concept of sliding mode converts a multiple order control problem to a first order control problem. The variable of interest is the value of the sliding surface s. The objective of the controller is to get s equal to zero. Several choices for the definition of the sliding surface can be made but in this case, a linear first-order sliding surface is chosen including an integrant term to avoid steady state errors (e.g. due to gravity). Furthermore, this control technique is robust for model errors and disturbances.

For the sliding mode controller the same state-space model is used as for the linear case.

$$\begin{aligned} \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} &= \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} O \\ \hat{H} \end{bmatrix} \vec{v} \\ x_1 &= [z \quad p \quad r]^T \\ x_2 &= [\dot{z} \quad \dot{p} \quad \dot{r}]^T \end{aligned} \tag{5.10}$$

Now the tracking error is defined:

$$\begin{aligned} \dot{x}_d &= Ax_d + \hat{B}v_d \\ e &= x - x_d \end{aligned} \tag{5.11}$$

In addition, the error can be converted to terms of e_1 and e_2 ($e_2 = \dot{e}_1$) just as the state:

$$\begin{bmatrix} \dot{e}_1 \\ \dot{e}_2 \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} + \begin{bmatrix} O \\ \hat{H} \end{bmatrix} (v - v_d) \tag{5.12}$$

The sliding surfaces for each DOF are defined using the form proposed in this section. This definition results in the following equations for s and $\dot{s}$:

$$\begin{aligned} s &= e_2 + 2Le_1 + L^2 \int_0^t e_1 dt \\ \dot{s} &= A_{21}e_1 + A_{22}e_2 + \hat{H}(v - v_d) + 2L(A_{11}e_1 + A_{12}e_2) + L^2 e_1 \end{aligned} \tag{5.13}$$

This can be simplified to:

$$\dot{s} = 2LA_{12}e_2 + L^2 e_1 + \hat{H}(v - v_d) \tag{5.14}$$

Now the control law is chosen to achieve $\dot{s} = 0$ in finite time.

$$v = v_d - \hat{H}^{-1}(2LA_{12}e_2 + L^2 e_1) - u_s \tag{5.15}$$

Then a suitable control law is chosen for u_s so that $\dot{s} = 0$ is an exponentially stable equilibrium point.

$$\begin{aligned} \dot{s} &= -u_s \\ u_s &= \Lambda sign(s) \end{aligned} \tag{5.16}$$

The sign has some disadvantages when used in real-time control, therefore a arctan function is used. In addition, the control law is rephrased to allow tuning of the performance for each DOF.

$$\begin{bmatrix} v_z \\ v_p \\ v_r \end{bmatrix} = \begin{bmatrix} v_{d,z} \\ v_{d,p} \\ v_{d,r} \end{bmatrix} - \hat{H}^{-1} \left(2 \begin{bmatrix} L_1 & 0 & 0 \\ 0 & L_2 & 0 \\ 0 & 0 & L_3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{e}_z \\ \dot{e}_p \\ \dot{e}_r \end{bmatrix} + \begin{bmatrix} L_1^2 & 0 & 0 \\ 0 & L_2^2 & 0 \\ 0 & 0 & L_3^2 \end{bmatrix} \begin{bmatrix} e_z \\ e_p \\ e_r \end{bmatrix} \right)$$
$$- \begin{bmatrix} \Lambda_1 & 0 & 0 \\ 0 & \Lambda_2 & 0 \\ 0 & 0 & \Lambda_3 \end{bmatrix} \begin{bmatrix} \frac{2}{\pi} \arctan(\phi s_z) \\ \frac{2}{\pi} \arctan(\phi s_p) \\ \frac{2}{\pi} \arctan(\phi s_r) \end{bmatrix} \tag{5.17}$$

$$\begin{bmatrix} v_{d,z} \\ v_{d,p} \\ v_{d,r} \end{bmatrix} = \hat{H}^{-1} \begin{bmatrix} \ddot{z}_d \\ \ddot{p}_d \\ \ddot{r}_d \end{bmatrix} \tag{5.18}$$

5.4 Controller Implementation

The controller is implemented in Matlab/Simulink as a user-defined s-function using the ode1 fixed-step solver with a stepsize of 2.5e-4 [s]. The inputs of the controller are the speed of the sled, the error for each DOF, the controller parameters L, the pitch and roll angle and the acceleration of the trajectory of each DOF for the feedforward term. The outputs are the 20 currents for the coils.

The calculation of the controller outputs desires the derivative of the error. A first order filter is used to calculate these derivatives.

$$x = \frac{1}{k}\dot{x}_f + x_f \tag{5.19}$$

The filtered error is equal to the real error for all frequencies in the filter bandwidth. So the choice of k depends on the desired bandwidth of the dynamics and is limited by the sample frequency. In this case the value of k is 300 and the bandwidth of the filter is almost 100 [Hz] (300 [rad/s]). This bandwidth is sufficient because the natural frequencies of the state-space model are zero. Unfortunately the filter does include phase-loss above 100 [Hz] but that was not a problem in the simulations. The following equation is used to calculate the derivatives of the error for each DOF.

$$\dot{e}_f = k(e - e_f) \tag{5.20}$$

The model s-function as in appendix B must be adapted. The coil dynamics are replaced and the currents for the force equations are inputs of the system. The c-code of the controller and the model for feedback control are listed in appendix D.

Chapter 6

Controller Simulation Results

Two simulations are performed using the controller. The first simulation has no disturbances. The trajectories set for the sled are zero for pitch and roll and a z-trajectory to lift the sled from its initial position to the null-flux axis, using a 3rd order trajectory. These simulations have been done with a speed profile of the sled that accelerates with a constant acceleration to a maximum speed of 25 meters per second. The parameters used for the simulations are listed in table 6.1

Table 6.1: Controller parameters

Parameter	Value
L_1	25
L_2	20
L_3	20
Λ_1	45
Λ_2	20
Λ_3	20
ϕ	50

The z-error during the trajectory is acceptable (0.15 mm maximum) and the error goes to zero in finite time. Also the errors in pitch and roll are acceptable. Furthermore the error in roll is 10 times smaller as the other two errors because its dynamics are decoupled and the controller only has to compensate for the unbalance. The computed currents necessary to control the vehicle have a maximum of 1 [A] and the rise times are realizable. The steady-state oscillations are due to approximation errors used to switch x back to zero when a new levitation coil is reached. They cannot be reduced by tuning the controller parameters.

The second simulation has one sample (T = 2.5e-4s) time delay on all measurements and 2 samples time delay on the actuator signal. Also all measurements have band-limited white band noise with a bandwidth of 300 Hz and quantizer noise due to discretization. The error goes to the noise level. The currents needed to control the sled dynamics are still acceptable with a maximum of 2.5 [A].

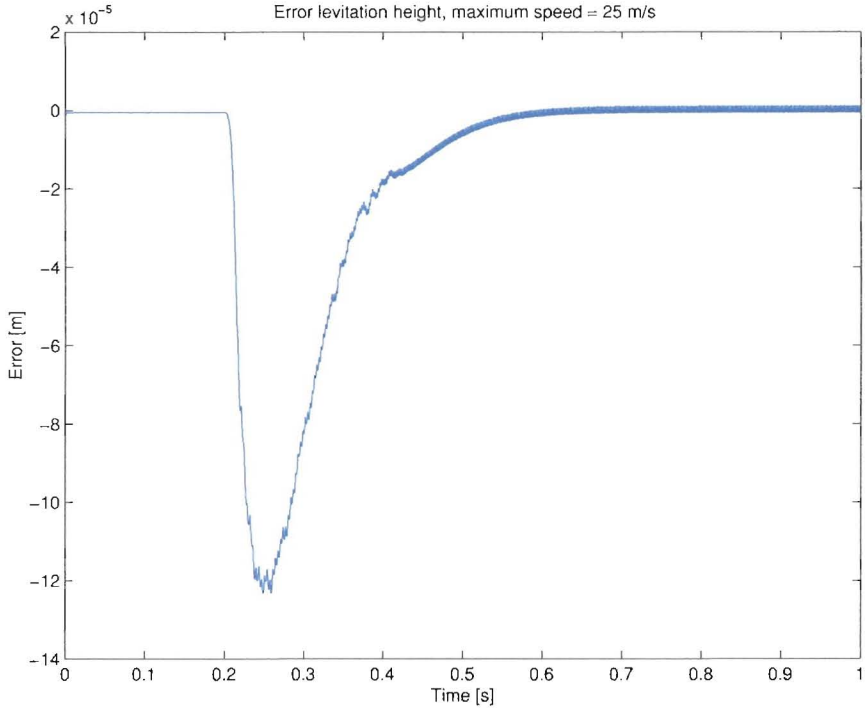

Figure 6.1: Levitation error without disturbance

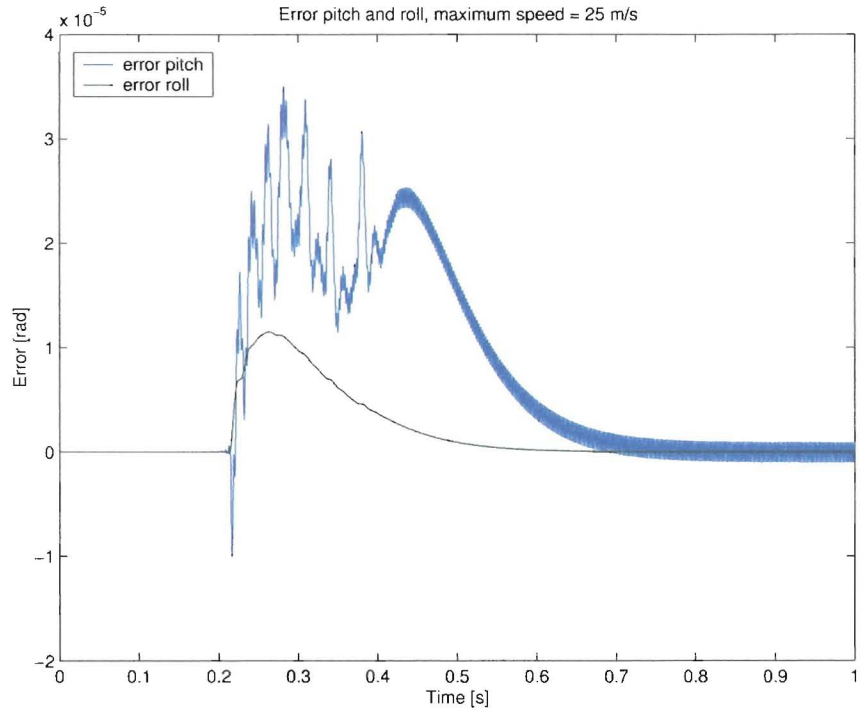

Figure 6.2: Pitch and roll error without disturbance

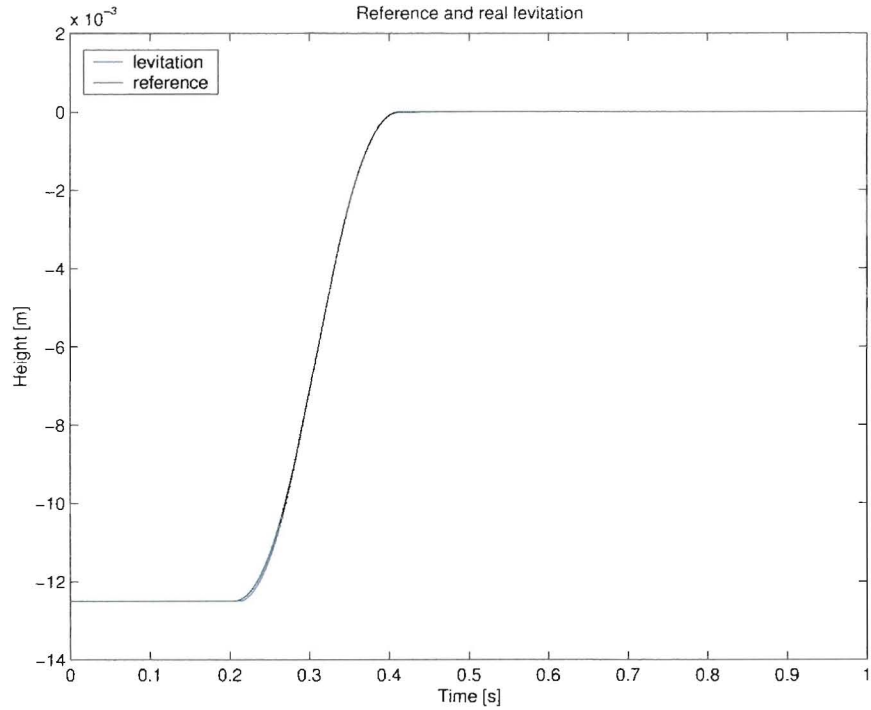

Figure 6.3: Levitation and reference without disturbance

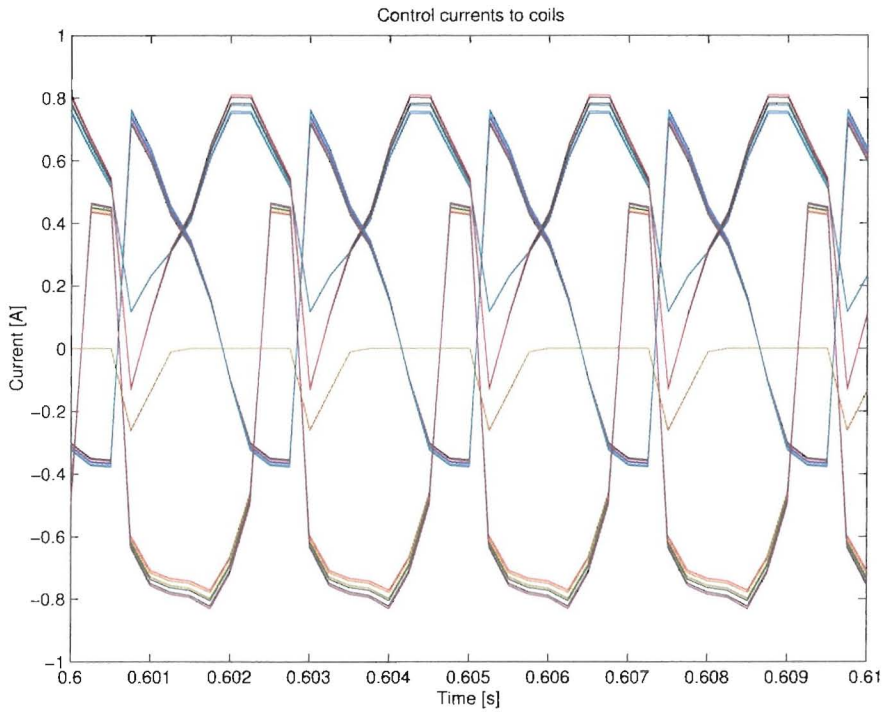

Figure 6.4: Steady-State controller currents without disturbance

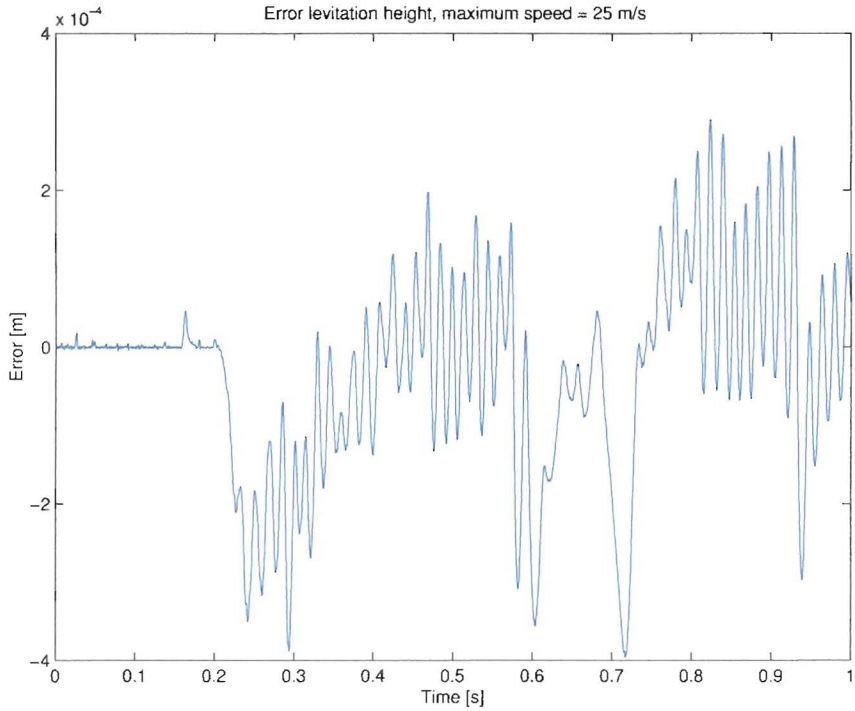

Figure 6.5: Levitation error with disturbance

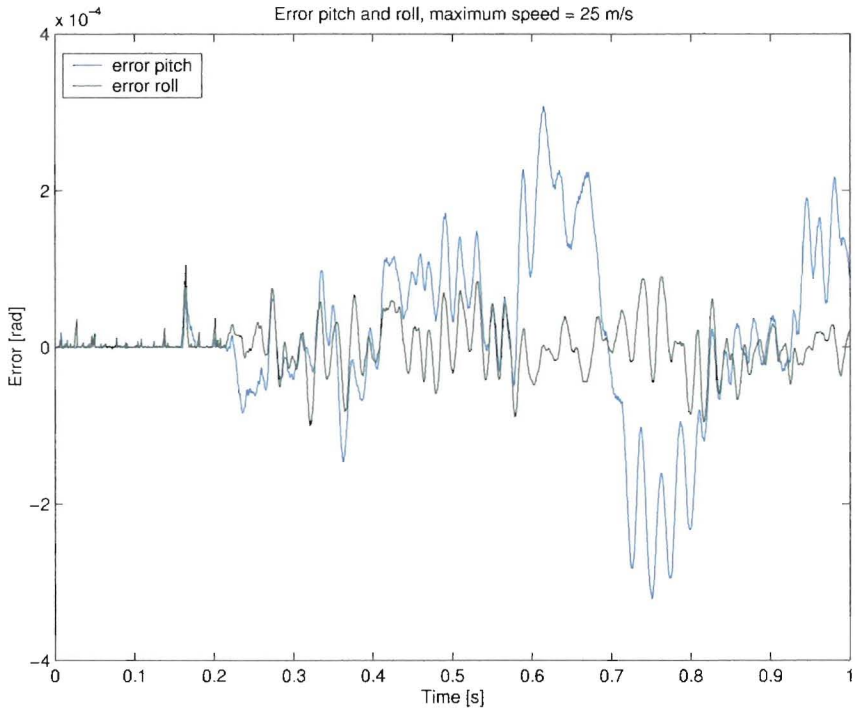

Figure 6.6: Pitch and roll error with disturbance

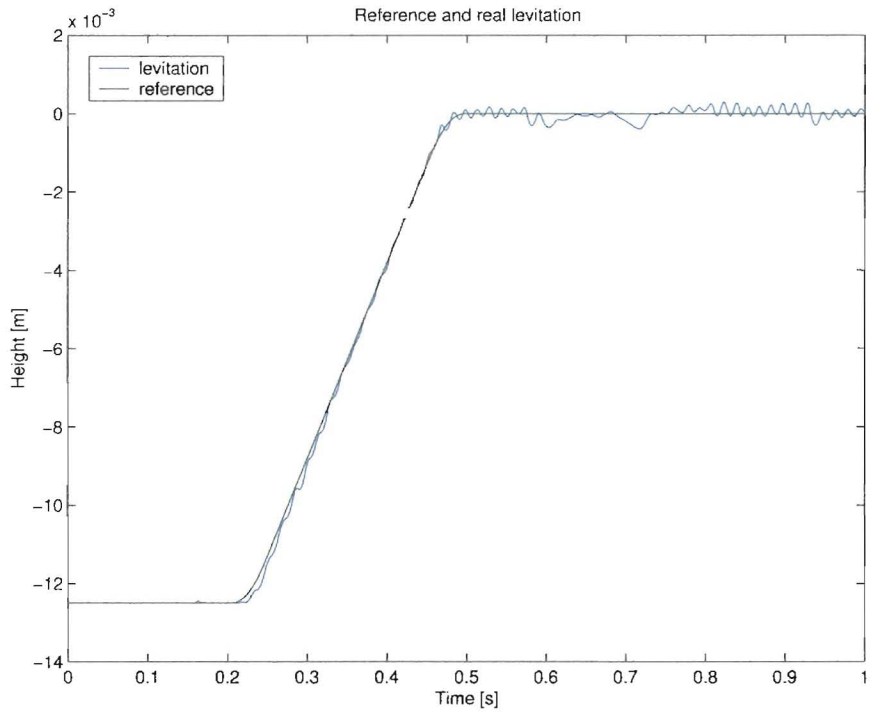

Figure 6.7: Levitation and reference with disturbance

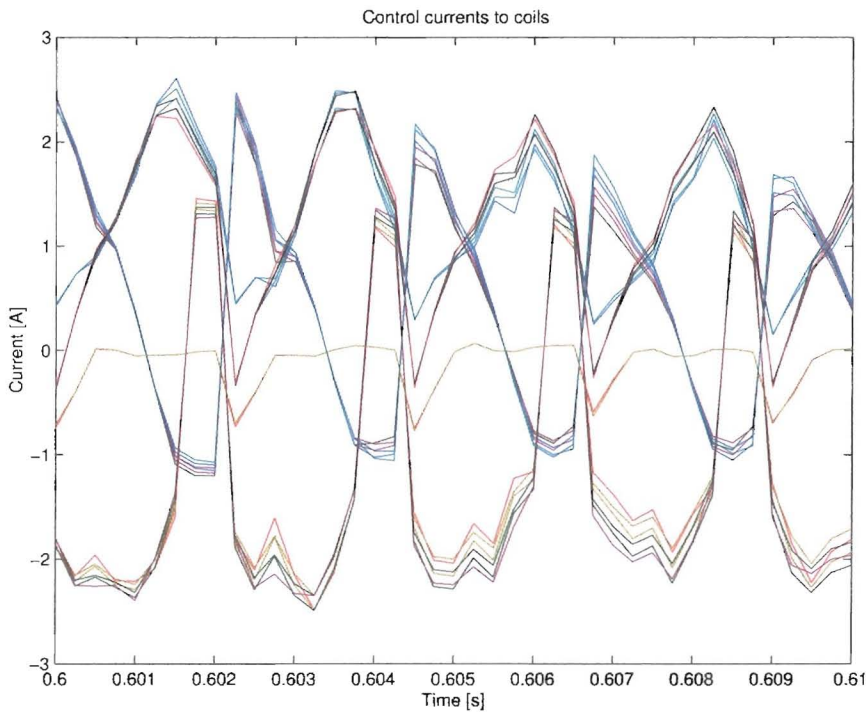

Figure 6.8: Steady-State controller currents with disturbance

Chapter 7

Conclusions and Future Research

7.1 Conclusions

The model of the 3-DOF EDS Maglev is definitely a step in the right direction. The dynamics of the sled in simulation are conform the result of earlier research. The part of the model that describes the geometry of the magnets and coils with respect to the induced currents is correct except for some parameters that must be estimated. To validate the force part of the model, some improvements must be made and some problems solved. This will be discussed in section 7.2. The sliding mode controller with input mapping is working properly for the model. The maximum tracking error is approximately 1 percent. The controller is robuust for noise, delays and model errors. If the model can be validated then this would be a good start for realtime feedback control of the sled dynamics. Future research for feedback control is discussed in section 7.3.

7.2 Future Research for Model Validation

As discussed earlier the sled is not levitating completely. The mass of all the sensors, laptop and battery is probably the cause. The permanent magnets are over 3 years old and the field strength has decreased over the years. To overcome these problems a new sled is build with new, stronger permanent magnets and with lighter construction materials. Hopefully the new sled will levitate completely with all the sensors and the laptop, then it will be possible to validate the model. Another problem that must be solved is the noise on the signals of the laser gap sensors. To improve the accuracy of the data it will be desirable to increase the resolution of the x-position measurement. A technique to improve the accuracy is to use multiple optical sensors to detect the dots. The dots are 1 inch apart, for example by using two sensors (0.5 inch apart) the accuracy can be increased with a factor 2 to 0.5 inch.

7.3 Future Research for Feedback Control

In the future this controller will be implemented on a test track from NASA. Each figure-8 coils of the track will be cut open and connected to a computer controlled current source. A schematic of the controller implementation is found in figure 7.1. In this schematic a laptop is used and data is transmitted using the wireless link. It might very well be possible that the delay caused by the time needed to transmit data from the sled to the desktop is too high. Another option is to use a DSP on the sled too and to use an FM radio to transmit the data from the sled to the desktop.

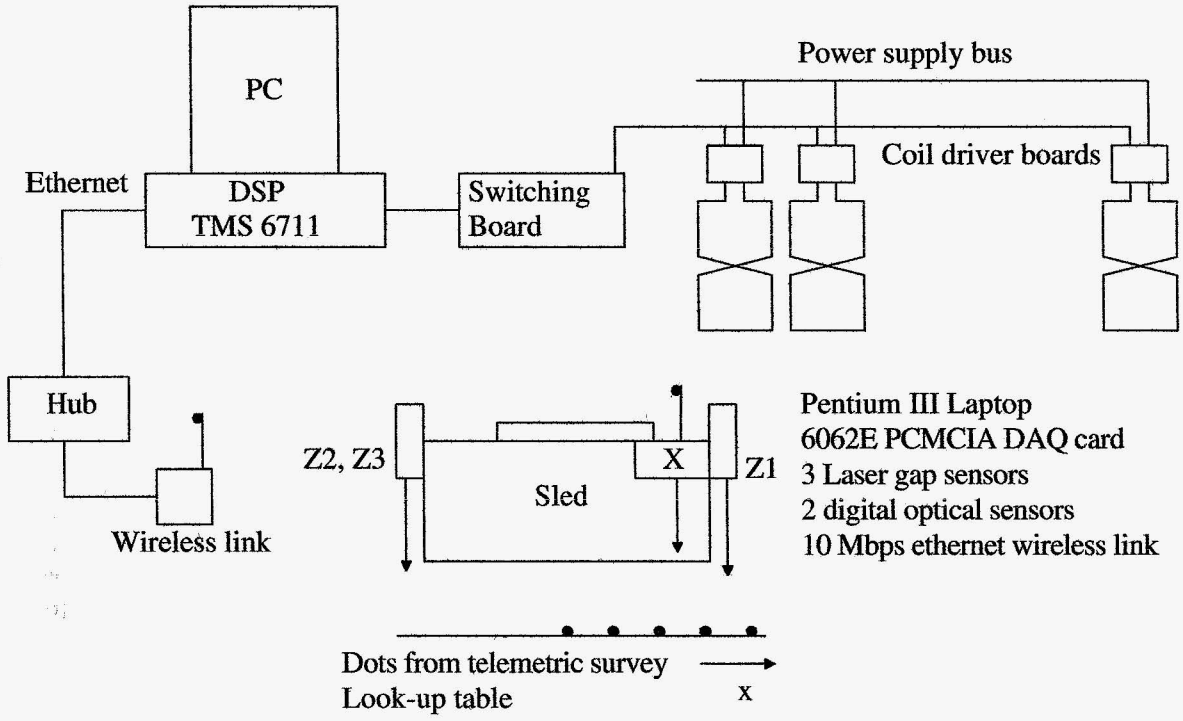

Figure 7.1: Schematic of controller implementation

Appendix A

Flux and Voltage equations of 3-DOF Model

1. First set of flux equations: $0 \leq x \leq \frac{1}{2}h - d$ (A.1)

$\Phi_a = 2B_1\delta_1 x$
$\Phi_b = 2B_1\delta_1(b-x)$
$\Phi_c = 2B_2\delta_2(\frac{1}{2}h + x)$
$\Phi_d = 2B_2\delta_2(\frac{1}{2}h - d - x) + 2B_3\delta_3 x$
$\Phi_e = 2B_3\delta_3(b-x)$
$\Phi_f = 2B_4\delta_4(\frac{1}{2}h + x)$
$\Phi_g = 2B_4\delta_4(\frac{1}{2}h - d - x) + 2B_5\delta_5 x$
$\Phi_h = 2B_5\delta_5(b-x)$
$\Phi_i = 2B_6\delta_6(\frac{1}{2}h + x)$
$\Phi_j = 2B_6\delta_6(\frac{1}{2}h - d - x)$

2. Second set of flux equations: $\frac{1}{2}h - d \leq x \leq \frac{1}{2}h$ (A.2)

$\Phi_a = 2B_1\delta_1 x$
$\Phi_b = 2B_1\delta_1(b-x)$
$\Phi_c = 2B_2\delta_2 b$
$\Phi_d = 2B_3\delta_3 x$
$\Phi_e = 2B_3\delta_3(b-x)$
$\Phi_f = 2B_4\delta_4 b$
$\Phi_g = 2B_5\delta_5 x$
$\Phi_h = 2B_5\delta_5(b-x)$
$\Phi_i = 2B_6\delta_6 b$
$\Phi_j = 0$

3. Third set of flux equations: $\frac{1}{2}h \leq x \leq b$ (A.3)

$\Phi_a = 2B_1\delta_1 x$
$\Phi_b = 2B_1\delta_1(b-x) + 2B_2\delta_2(x - \frac{1}{2}h)$
$\Phi_c = 2B_2\delta_2(b - x + \frac{1}{2}h)$
$\Phi_d = 2B_3\delta_3 x$
$\Phi_e = 2B_3\delta_3(b-x) + 2B_4\delta_4(x - \frac{1}{2}h)$
$\Phi_f = 2B_4\delta_4(b - x + \frac{1}{2}h)$
$\Phi_g = 2B_5\delta_5 x$
$\Phi_h = 2B_5\delta_5(b-x) + 2B_6\delta_6(x - \frac{1}{2}h)$
$\Phi_i = 2B_6\delta_6(b - x + \frac{1}{2}h)$
$\Phi_j = 0$

4. Fourth set of flux equations: $b \leq x \leq h$ (A.4)

$$\Phi_a = 2B_1\delta_1 b$$
$$\Phi_b = 2B_2\delta_2(x - \tfrac{1}{2}h)$$
$$\Phi_c = 2B_2\delta_2(b - x + \tfrac{1}{2}h)$$
$$\Phi_d = 2B_3\delta_3 b$$
$$\Phi_e = 2B_4\delta_4(x - \tfrac{1}{2}h)$$
$$\Phi_f = 2B_4\delta_4(b - x + \tfrac{1}{2}h)$$
$$\Phi_g = 2B_5\delta_5 b$$
$$\Phi_h = 2B_6\delta_6(x - \tfrac{1}{2}h)$$
$$\Phi_i = 2B_6\delta_6(b - x + \tfrac{1}{2}h)$$
$$\Phi_j = 0$$

1. First set of voltage equations: $0 \leq x \leq \frac{1}{2}h - d$ (A.5)

$$V_a = -2B_1 x\dot{\delta}_1 - 2B_1\delta_1\dot{x}$$
$$V_b = -2B_1 b\dot{\delta}_1 + 2B_1 x\dot{\delta}_1 + 2B_1\delta_1\dot{x}$$
$$V_c = -B_2 h\dot{\delta}_2 - 2B_2 x\dot{\delta}_2 - 2B_2\delta_2\dot{x}$$
$$V_d = -B_2 h\dot{\delta}_2 + 2B_2 x\dot{\delta}_2 + 2B_2\delta_2\dot{x} - 2B_3 x\dot{\delta}_3 - 2B_3\delta_3\dot{x}$$
$$V_e = -2B_3 b\dot{\delta}_3 + 2B_3 x\dot{\delta}_3 + 2B_3\delta_3\dot{x}$$
$$V_f = -B_4 h\dot{\delta} - 2B_4 x\dot{\delta}_4 - 2B_4\delta_4\dot{x}$$
$$V_g = -B_4 h\dot{\delta}_4 + 2B_4 x\dot{\delta}_4 + 2B_4\delta_4\dot{x} - 2B_5 x\dot{\delta}_5 - 2B_5\delta_5\dot{x}$$
$$V_h = -2B_5 b\dot{\delta}_5 + 2B_5 x\dot{\delta}_5 + 2B_5\delta_5\dot{x}$$
$$V_i = -B_6 h\dot{\delta}_6 - 2B_6 x\dot{\delta}_6 - 2B_6\delta_6\dot{x}$$
$$V_j = -B_6 h\dot{\delta}_6 + 2B_6 x\dot{\delta}_6 + 2B_6\delta_6\dot{x}$$

2. Second set of voltage equations: $\frac{1}{2}h - d \leq x \leq \frac{1}{2}h$ (A.6)

$$V_a = -2B_1 x\dot{\delta}_1 - 2B_1\delta_1\dot{x}$$
$$V_b = -2B_1 b\dot{\delta}_1 + 2B_1 x\dot{\delta}_1 + 2B_1\delta_1\dot{x}$$
$$V_c = -2B_2 b\dot{\delta}_2$$
$$V_d = -2B_3 x\dot{\delta}_3 - 2B_3\delta_3\dot{x}$$
$$V_e = -2B_3 b\dot{\delta}_3 + 2B_3 x\dot{\delta}_3 + 2B_3\delta_3\dot{x}$$
$$V_f = -2B_4 b\dot{\delta}_4$$
$$V_g = -2B_5 x\dot{\delta}_5 - 2B_5\delta_5\dot{x}$$
$$V_h = -2B_5 b\dot{\delta}_5 + 2B_5 x\dot{\delta}_5 + 2B_5\delta_5\dot{x}$$
$$V_i = -2B_6 b\dot{\delta}_6$$
$$V_j = 0$$

3. Third set of voltage equations: $\frac{1}{2}h \leq x \leq b$ (A.7)

$$V_a = -2B_1 x\dot{\delta}_1 - 2B_1\delta_1\dot{x}$$
$$V_b = -2B_1 b\dot{\delta}_1 + 2B_1 x\dot{\delta}_1 + 2B_1\delta_1\dot{x} + B_2 h\dot{\delta}_2 - 2B_2 x\dot{\delta}_2 - 2B_2\delta_2\dot{x}$$
$$V_c = -2B_2 b\dot{\delta}_2 - B_2 h\dot{\delta}_2 + 2B_2 x\dot{\delta}_2 + 2B_2\delta_2\dot{x}$$
$$V_d = -2B_3 x\dot{\delta}_3 - 2B_3\delta_3\dot{x}$$
$$V_e = -2B_3 b\dot{\delta}_3 + 2B_3 x\dot{\delta}_3 + 2B_3\delta_3\dot{x} + B_4 h\dot{\delta}_4 - 2B_4 x\dot{\delta}_4 - 2B_4\delta_4\dot{x}$$
$$V_f = -2B_4 b\dot{\delta}_4 - B_4 h\dot{\delta}_4 + 2B_4 x\dot{\delta} + 2B_4\delta_4\dot{x}$$
$$V_g = -2B_5 x\dot{\delta}_5 - 2B_5\delta_5\dot{x}$$
$$V_h = -2B_5 b\dot{\delta}_5 + 2B_5 x\dot{\delta}_5 + 2B_5\delta_5\dot{x} + B_6 h\dot{\delta}_6 - 2B_6 x\dot{\delta}_6 - 2B_6\delta_6\dot{x}$$
$$V_i = -2B_6 b\dot{\delta}_6 - B_6 h\dot{\delta}_6 + 2B_6 x\dot{\delta}_6 + 2B_6\delta_6\dot{x}$$
$$V_j = 0$$

4. Fourth set of voltage equations: $b \leq x \leq h$ (A.8)

$$V_a = -2B_1 b\dot{\delta}_1$$
$$V_b = -2B_1 b\dot{\delta}_1 + 2B_1 x\dot{\delta}_1 + 2B_1\delta_1\dot{x} + B_2 h\dot{\delta}_2 - 2B_2 x\dot{\delta}_2 - 2B_2\delta_2\dot{x}$$

$$V_c = -2B_2 b\dot{\delta}_2 - B_2 h\dot{\delta}_2 + 2B_2 x\dot{\delta}_2 + 2B_2\delta_2\dot{x}$$
$$V_d = -2B_3 b\dot{\delta}_3$$
$$V_e = B_4 h\dot{\delta}_4 - 2B_4 x\dot{\delta}_4 - 2B_4\delta_4\dot{x}$$
$$V_f = -2B_4 b\dot{\delta}_4 - B_4 h\dot{\delta}_4 + 2B_4 x\dot{\delta} + 2B_4\delta_4\dot{x}$$
$$V_g = -2B_5 b\dot{\delta}_5$$
$$V_h = B_6 h\dot{\delta}_6 - 2B_6 x\dot{\delta}_6 - 2B_6\delta_6\dot{x}$$
$$V_i = -2B_6 b\dot{\delta}_6 - B_6 h\dot{\delta}_6 + 2B_6 x\dot{\delta}_6 + 2B_6\delta_6\dot{x}$$
$$V_j = 0$$

The magnetic field strength of each magnet is represented by B. These equations are the same for each side of the sled.

Appendix B

C-code 3-DOF EDS Maglev Model

```
//=============================================================================
// setup
//=============================================================================

#define S_FUNCTION_NAME maglev_3d
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
#include "math.h"

// parameters
#define coil        1           // coil of which the current is an output
#define g           9.8         // gravity (acceleration) (m/s^2)
#define ground      -0.0125     // floor level of wheel-track (minimum levitation)
#define ceil        0.01        // ceil level of wheel-track (maximum levitation)

// coil parameters
#define R           0.638       // coil resistance (Ohm)
#define L           0.837e-3    // coil inductance (H)
#define N           100         // number of coil windings (-)
#define b           4.48e-2     // coil lenght (m)
#define d           0.6e-2      // gap between coils (m); (h=b+d)

// magnet parameters
#define h           5.08e-2     // height en length of magnet (m)
#define B           1.25         // magnetic field strenght (T)

// vehicle parameters
#define M           6.0         // mass of the vehicle (kg)
#define Jp          0.1437      // pitch rotational inertia (kg*m^2)
#define dTp         0.01        // torque disturbace on pitch (N*m)
#define Jr          0.0288      // roll rotational inertia (kg*m^2)
#define dTr         0.01        // torque disturbance on roll (N*m)
#define LL          0.4572      // length of vehicle (m)
#define L1          0.1905      // distance of magnet one to centre of inertia (m)
#define L2          0.1143      // distance of magnet two to c.o.i (m)
#define L3          0.0381      // distance of magnet three to c.o.i. (m)
#define L4          -0.0381     // distance of magnet four to c.o.i. (m)
#define L5          -0.1143     // distance of magnet five to c.o.i (m)
#define L6          -0.1905     // distance of magnet six to c.o.i (m)
```

```
#define DD          0.2286      // width of vehicle (m)
#define D1          0.0368      // distance from magnet row 1 to c.o.i (m)
#define D2          -0.0368     // distance from magnet row 2 to c.o.i (m)

// aerodynamic drag forces
#define rho_air     1.0         // density of air at 293 K (kg/m^3)
#define A_front     0.0129      // area of front of the sled (m^2)
#define C_front     2           // drag coefficient of flat plate (-)
#define Fd_arm      0.025       // arm of drag force on pitch (m)



#define NINPUTS     1           //number of inputs      uPtrs: input vector
// 0. velocity

#define NOUTPUTS    5           //number of outputs     yPtrs: output vector
// 0. delta
// 1. current of one single coil when the magnets pass
// 2. pitch
// 3. roll
// 4. debug

#define NCSTATES    28           //number of continuous states     xc: state vector
// 00. vertical displacement of centre of inertia   (z)
// 01. vertical velocity of centre of inertia       (z_dot)
// 02. magnet related displacement of the vehicle   (x)
// 03. current coil 1a                              (Ia1)
// 04. current coil 1b                              (Ib1)
// 05. current coil 1c                              (Ic1)
// 06. current coil 1d                              (Id1)
// 07. current coil 1e                              (Ie1)
// 08. current coil 1f                              (If1)
// 09. current coil 1g                              (Ig1)
// 10. current coil 1h                              (Ih1)
// 11. current coil 1i                              (Ii1)
// 12. current coil 1j                              (Ij1)
// 13. pitch                                        (phi)
// 14. pitch velocity                               (phi_dot)
// 15. total displacement of vehicle                (x)
// 16. roll                                         (theta)
// 17. roll velocity                                (theta_dot)
// 18. current coil 2a                              (Ia2)
// 19. current coil 2b                              (Ib2)
// 20. current coil 2c                              (Ic2)
// 21. current coil 2d                              (Id2)
// 22. current coil 2e                              (Ie2)
// 23. current coil 2f                              (If2)
// 24. current coil 2g                              (Ig2)
// 25. current coil 2h                              (Ih2)
// 26. current coil 2i                              (Ii2)
// 27. current coil 2j                              (Ij2)

#define NIWRK       2           // number of integer global variables
#define NRWRK       6           // number of real global variables
```

```
//=============================================================================
// mdlInitializeSizes
//=============================================================================

static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S,0);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {return;}

    ssSetNumContStates(S,NCSTATES);
    ssSetNumDiscStates(S,0);

    if (!ssSetNumInputPorts(S,1)) {return;}
    ssSetInputPortWidth(S,0,NINPUTS);
    ssSetInputPortDirectFeedThrough(S,0,1);

    if (!ssSetNumOutputPorts(S,1)) {return;}
    ssSetOutputPortWidth(S,0,NOUTPUTS);

    ssSetNumSampleTimes(S,1);
    ssSetNumRWork(S,NRWRK);
    ssSetNumIWork(S,NIWRK);
    ssSetNumPWork(S,0);
    ssSetNumModes(S,0);
    ssSetNumNonsampledZCs(S,0);
}


//=============================================================================
// mdlInitializeSampleTimes
//=============================================================================

static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S,0,CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S,0,0.0);
}


//=============================================================================
// mdlInitializeConditions
//=============================================================================

#define MDL_INITIALIZE_CONDITIONS

static void mdlInitializeConditions(SimStruct *S)
{
    real_T  *xc =   ssGetContStates(S);
    real_T  *dx =   ssGetdX(S);

    real_T  *prwrk  =   ssGetRWork(S);
    int_T   *piwrk  =   ssGetIWork(S);
```

```
    int_T   i;

    for(i=0;i<NCSTATES;i++){
        xc[i]=0.0;
    }
    xc[0]=ground;
    for(i=0;i<NIWRK;i++){piwrk[i]=0;}
    for(i=0;i<NRWRK;i++){prwrk[i]=0.0;}
}

//=============================================================================
// mdlOutputs
//=============================================================================

static void mdlOutputs(SimStruct *S, int_T tid)
{
    real_T  *xc     =   ssGetContStates(S);
    real_T  *yPtrs  =   ssGetOutputPortRealSignal(S,0);

    InputRealPtrsType   uPtrs   =   ssGetInputPortRealSignalPtrs(S,0);

    int_T   *piwrk  =   ssGetIWork(S);
    real_T  *prwrk  =   ssGetRWork(S);

    // output
    yPtrs[0]=xc[0];

    if      (piwrk[0]==coil)    yPtrs[1]=xc[3];
    else if (piwrk[0]==coil+1)  yPtrs[1]=xc[4];
    else if (piwrk[0]==coil+2)  yPtrs[1]=xc[5];
    else if (piwrk[0]==coil+3)  yPtrs[1]=xc[6];
    else if (piwrk[0]==coil+4)  yPtrs[1]=xc[7];
    else if (piwrk[0]==coil+5)  yPtrs[1]=xc[8];
    else if (piwrk[0]==coil+6)  yPtrs[1]=xc[9];
    else if (piwrk[0]==coil+7)  yPtrs[1]=xc[10];
    else if (piwrk[0]==coil+8)  yPtrs[1]=xc[11];
    else if (piwrk[0]==coil+9)  yPtrs[1]=xc[12];
    else                        yPtrs[1]=0;

    yPtrs[2]=xc[13];
    yPtrs[3]=xc[16];
    yPtrs[4]=prwrk[4];
}


//=============================================================================
// mdlDerivatives
//=============================================================================

#define MDL_DERIVATIVES

static void mdlDerivatives(SimStruct *S)
{
    real_T  *xc =   ssGetContStates(S);
```

```
real_T  *dx =   ssGetdX(S);

real_T  Tp,Tr,F,Fd,Td;
real_T  F_1,F1_1,F2_1,F3_1,F4_1,F5_1,F6_1;
real_T  F_2,F1_2,F2_2,F3_2,F4_2,F5_2,F6_2;
real_T  Va_1,Vb_1,Vc_1,Vd_1,Ve_1,Vf_1,Vg_1,Vh_1,Vi_1,Vj_1;
real_T  Va_2,Vb_2,Vc_2,Vd_2,Ve_2,Vf_2,Vg_2,Vh_2,Vi_2,Vj_2;
real_T  z1,z2,z3,z4;
int_T   z1_ground,z2_ground,z3_ground,z4_ground;
int_T   z1_ceil,z2_ceil,z3_ceil,z4_ceil;

real_T  B1          =   B;
real_T  B2          =   -B;
real_T  B3          =   B;
real_T  B4          =   -B;
real_T  B5          =   B;
real_T  B6          =   -B;

// displacements of magnets w.r.t. null-flux axis
real_T  d1_1,d2_1,d3_1,d4_1,d5_1,d6_1;
real_T  d1_2,d2_2,d3_2,d4_2,d5_2,d6_2;
// velocity of magnets
real_T  d1d_1,d2d_1,d3d_1,d4d_1,d5d_1,d6d_1;
real_T  d1d_2,d2d_2,d3d_2,d4d_2,d5d_2,d6d_2;

int_T   status_half,status_gap_half,status_gap;
int_T   not_half        =   0;  // (xc[2]<h/2)
int_T   half            =   1;  // (xc[2]>h/2)
int_T   not_gap_half    =   0;  // (xc[2]<h/2-d)
int_T   gap_half        =   1;  // (xc[2]>h/2-d)
int_T   not_gap         =   0;  // (xc[2]<b)
int_T   gap             =   1;  // (xc[2]>b)

InputRealPtrsType   uPtrs   =   ssGetInputPortRealSignalPtrs(S,0);

int_T   *piwrk  =   ssGetIWork(S);
real_T  *prwrk  =   ssGetRWork(S);

// count coils that are passed by the first magnet
if( h-xc[2] < 1e-4){
    piwrk[1]=piwrk[1]+1;

    if(piwrk[0]==coil-1){
        piwrk[1]=0;
    }
    else if( piwrk[0] > coil-1 & piwrk[0] < coil+10){
        if (piwrk[1]==2){
            piwrk[1]=0;
        }
    }
    else{
        piwrk[1]=0;
    }
}
```

```
// update coils when the first magnet has passed one coil
if( h-xc[2] < 1e-4 ){
    xc[12]=xc[11];
    dx[12]=dx[11];
    xc[11]=xc[10];
    dx[11]=dx[10];
    xc[10]=xc[9];
    dx[10]=dx[9];
    xc[9]=xc[8];
    dx[9]=dx[8];
    xc[8]=xc[7];
    dx[8]=dx[7];
    xc[7]=xc[6];
    dx[7]=dx[6];
    xc[6]=xc[5];
    dx[6]=dx[5];
    xc[5]=xc[4];
    dx[5]=dx[4];
    xc[4]=xc[3];
    dx[4]=dx[3];
    xc[3]=0.0;
    dx[3]=0.0;
    xc[27]=xc[26];
    dx[27]=dx[26];
    xc[26]=xc[25];
    dx[26]=dx[25];
    xc[25]=xc[24];
    dx[25]=dx[24];
    xc[24]=xc[23];
    dx[24]=dx[23];
    xc[23]=xc[22];
    dx[23]=dx[22];
    xc[22]=xc[21];
    dx[22]=dx[21];
    xc[21]=xc[20];
    dx[21]=dx[20];
    xc[20]=xc[19];
    dx[20]=dx[19];
    xc[19]=xc[18];
    dx[19]=dx[18];
    xc[18]=0.0;
    dx[18]=0.0;
    xc[2]=0.0;
    dx[2]=0.0;
    piwrk[0]=piwrk[0]+1;
}

// check in which status (position )the magnet is in
if      ( h/2.0-xc[2] < 1e-4)   {status_half=half;}
else                            {status_half=not_half;}

if      ( h/2.0-d-xc[2] < 1e-4) {status_gap_half=gap_half;}
else                            {status_gap_half=not_gap_half;}
```

```
if      ( b-xc[2] < 1e-4)       {status_gap=gap;}
else                            {status_gap=not_gap;}

// calculate magnets displacements and velocities
d1_1    = L1*sin(xc[13]) + D1*sin(xc[16]) + xc[0];
d2_1    = L2*sin(xc[13]) + D1*sin(xc[16]) + xc[0];
d3_1    = L3*sin(xc[13]) + D1*sin(xc[16]) + xc[0];
d4_1    = L4*sin(xc[13]) + D1*sin(xc[16]) + xc[0];
d5_1    = L5*sin(xc[13]) + D1*sin(xc[16]) + xc[0];
d6_1    = L6*sin(xc[13]) + D1*sin(xc[16]) + xc[0];

d1_2    = L1*sin(xc[13]) + D2*sin(xc[16]) + xc[0];
d2_2    = L2*sin(xc[13]) + D2*sin(xc[16]) + xc[0];
d3_2    = L3*sin(xc[13]) + D2*sin(xc[16]) + xc[0];
d4_2    = L4*sin(xc[13]) + D2*sin(xc[16]) + xc[0];
d5_2    = L5*sin(xc[13]) + D2*sin(xc[16]) + xc[0];
d6_2    = L6*sin(xc[13]) + D2*sin(xc[16]) + xc[0];

d1d_1   = L1*xc[14]*cos(xc[13]) + D1*xc[17]*cos(xc[16]) + xc[1];
d2d_1   = L2*xc[14]*cos(xc[13]) + D1*xc[17]*cos(xc[16]) + xc[1];
d3d_1   = L3*xc[14]*cos(xc[13]) + D1*xc[17]*cos(xc[16]) + xc[1];
d4d_1   = L4*xc[14]*cos(xc[13]) + D1*xc[17]*cos(xc[16]) + xc[1];
d5d_1   = L5*xc[14]*cos(xc[13]) + D1*xc[17]*cos(xc[16]) + xc[1];
d6d_1   = L6*xc[14]*cos(xc[13]) + D1*xc[17]*cos(xc[16]) + xc[1];

d1d_2   = L1*xc[14]*cos(xc[13]) + D2*xc[17]*cos(xc[16]) + xc[1];
d2d_2   = L2*xc[14]*cos(xc[13]) + D2*xc[17]*cos(xc[16]) + xc[1];
d3d_2   = L3*xc[14]*cos(xc[13]) + D2*xc[17]*cos(xc[16]) + xc[1];
d4d_2   = L4*xc[14]*cos(xc[13]) + D2*xc[17]*cos(xc[16]) + xc[1];
d5d_2   = L5*xc[14]*cos(xc[13]) + D2*xc[17]*cos(xc[16]) + xc[1];
d6d_2   = L6*xc[14]*cos(xc[13]) + D2*xc[17]*cos(xc[16]) + xc[1];

// calculate voltages and forces for all magnets depending on the status.

// 0 <= xc[2] <= h/2-d
if      (status_half == not_half & status_gap_half == not_gap_half){
    Va_1 = -2*B1*xc[2]*d1d_1-2*B1*d1_1*(*uPtrs[0]);
    Vb_1 = -2*B1*b*d1d_1+2*B1*xc[2]*d1d_1+2*B1*d1_1*(*uPtrs[0]);
    Vc_1 = -B2*h*d2d_1-2*B2*xc[2]*d2d_1-2*B2*d2_1*(*uPtrs[0]);
    Vd_1 = -B2*h*d2d_1+2*B2*d*d2d_1+2*B2*xc[2]*d2d_1+2*B2*d2_1*(*uPtrs[0])
        -2*B3*xc[2]*d3d_1-2*B3*d3_1*(*uPtrs[0]);
    Ve_1 = -2*B3*b*d3d_1+2*B3*xc[2]*d3d_1+2*B3*d3_1*(*uPtrs[0]);
    Vf_1 = -B4*h*d4d_1-2*B4*xc[2]*d4d_1-2*B4*d4_1*(*uPtrs[0]);
    Vg_1 = -B4*h*d4d_1+2*B4*d*d4d_1+2*B4*xc[2]*d4d_1+2*B4*d4_1*(*uPtrs[0])
        -2*B5*xc[2]*d5d_1-2*B5*d5_1*(*uPtrs[0]);
    Vh_1 = -2*B5*b*d5d_1+2*B5*xc[2]*d5d_1+2*B5*d5_1*(*uPtrs[0]);
    Vi_1 = -B6*h*d6d_1-2*B6*xc[2]*d6d_1-2*B6*d6_1*(*uPtrs[0]);
    Vj_1 = -B6*h*d6d_1+2*B6*d*d6d_1+2*B6*xc[2]*d6d_1+2*B6*d6_1*(*uPtrs[0]);
    //
    F1_1 = xc[3]*2*B1*xc[2]+xc[4]*2*B1*(b-xc[2]);
    F2_1 = xc[5]*2*B2*(h/2.0+xc[2])+xc[6]*2*B2*(h/2.0-d-xc[2]);
    F3_1 = xc[6]*2*B3*xc[2]+xc[7]*2*B3*(b-xc[2]);
    F4_1 = xc[8]*2*B4*(h/2.0+xc[2])+xc[9]*2*B4*(h/2.0-d-xc[2]);
```

```
    F5_1 = xc[9]*2*B5*xc[2]+xc[10]*2*B5*(b-xc[2]);
    F6_1 = xc[11]*2*B6*(h/2.0+xc[2])+xc[12]*2*B6*(h/2.0-d-xc[2]);
    //
    //
    Va_2 = -2*B1*xc[2]*d1d_2-2*B1*d1_2*(*uPtrs[0]);
    Vb_2 = -2*B1*b*d1d_2+2*B1*xc[2]*d1d_2+2*B1*d1_2*(*uPtrs[0]);
    Vc_2 = -B2*h*d2d_2-2*B2*xc[2]*d2d_2-2*B2*d2_2*(*uPtrs[0]);
    Vd_2 = -B2*h*d2d_2+2*B2*d*d2d_2+2*B2*xc[2]*d2d_2+2*B2*d2_2*(*uPtrs[0])
        -2*B3*xc[2]*d3d_2-2*B3*d3_2*(*uPtrs[0]);
    Ve_2 = -2*B3*b*d3d_2+2*B3*xc[2]*d3d_2+2*B3*d3_2*(*uPtrs[0]);
    Vf_2 = -B4*h*d4d_2-2*B4*xc[2]*d4d_2-2*B4*d4_2*(*uPtrs[0]);
    Vg_2 = -B4*h*d4d_2+2*B4*d*d4d_2+2*B4*xc[2]*d4d_2+2*B4*d4_2*(*uPtrs[0])
        -2*B5*xc[2]*d5d_2-2*B5*d5_2*(*uPtrs[0]);
    Vh_2 = -2*B5*b*d5d_2+2*B5*xc[2]*d5d_2+2*B5*d5_2*(*uPtrs[0]);
    Vi_2 = -B6*h*d6d_2-2*B6*xc[2]*d6d_2-2*B6*d6_2*(*uPtrs[0]);
    Vj_2 = -B6*h*d6d_2+2*B6*d*d6d_2+2*B6*xc[2]*d6d_2+2*B6*d6_2*(*uPtrs[0]);
    //
    F1_2 = xc[18]*2*B1*xc[2]+xc[19]*2*B1*(b-xc[2]);
    F2_2 = xc[20]*2*B2*(h/2.0+xc[2])+xc[21]*2*B2*(h/2.0-d-xc[2]);
    F3_2 = xc[21]*2*B3*xc[2]+xc[22]*2*B3*(b-xc[2]);
    F4_2 = xc[23]*2*B4*(h/2.0+xc[2])+xc[24]*2*B4*(h/2.0-d-xc[2]);
    F5_2 = xc[24]*2*B5*xc[2]+xc[25]*2*B5*(b-xc[2]);
    F6_2 = xc[26]*2*B6*(h/2.0+xc[2])+xc[27]*2*B6*(h/2.0-d-xc[2]);
}

// h/2-d <= xc[2] <= h/2
else if (status_half == not_half & status_gap_half == gap_half){
    Va_1 = -2*B1*xc[2]*d1d_1-2*B1*d1_1*(*uPtrs[0]);
    Vb_1 = -2*B1*b*d1d_1+2*B1*xc[2]*d1d_1+2*B1*d1_1*(*uPtrs[0]);
    Vc_1 = -2*B2*b*d2d_1;
    Vd_1 = -2*B3*xc[2]*d3d_1-2*B3*d3_1*(*uPtrs[0]);
    Ve_1 = -2*B3*b*d3d_1+2*B3*xc[2]*d3d_1+2*B3*d3_1*(*uPtrs[0]);
    Vf_1 = -2*B4*b*d4d_1;
    Vg_1 = -2*B5*xc[2]*d5d_1-2*B5*d5_1*(*uPtrs[0]);
    Vh_1 = -2*B5*b*d5d_1+2*B5*xc[2]*d5d_1+2*B5*d5_1*(*uPtrs[0]);
    Vi_1 = -2*B6*b*d6d_1;
    Vj_1 = 0.0;
    //
    F1_1 = xc[3]*2*B1*xc[2]+xc[4]*2*B1*(b-xc[2]);
    F2_1 = xc[5]*2*B2*b;
    F3_1 = xc[6]*2*B3*xc[2]+xc[7]*2*B3*(b-xc[2]);
    F4_1 = xc[8]*2*B4*b;
    F5_1 = xc[9]*2*B5*xc[2]+xc[10]*2*B5*(b-xc[2]);
    F6_1 = xc[11]*2*B6*b;
    //
    //
    Va_2 = -2*B1*xc[2]*d1d_2-2*B1*d1_2*(*uPtrs[0]);
    Vb_2 = -2*B1*b*d1d_2+2*B1*xc[2]*d1d_2+2*B1*d1_2*(*uPtrs[0]);
    Vc_2 = -2*B2*b*d2d_2;
    Vd_2 = -2*B3*xc[2]*d3d_2-2*B3*d3_2*(*uPtrs[0]);
    Ve_2 = -2*B3*b*d3d_2+2*B3*xc[2]*d3d_2+2*B3*d3_2*(*uPtrs[0]);
    Vf_2 = -2*B4*b*d4d_2;
    Vg_2 = -2*B5*xc[2]*d5d_2-2*B5*d5_2*(*uPtrs[0]);
    Vh_2 = -2*B5*b*d5d_2+2*B5*xc[2]*d5d_2+2*B5*d5_2*(*uPtrs[0]);
```

```
    Vi_2 = -2*B6*b*d6d_2;
    Vj_2 = 0.0;
    //
    F1_2 = xc[18]*2*B1*xc[2]+xc[19]*2*B1*(b-xc[2]);
    F2_2 = xc[20]*2*B2*b;
    F3_2 = xc[21]*2*B3*xc[2]+xc[22]*2*B3*(b-xc[2]);
    F4_2 = xc[23]*2*B4*b;
    F5_2 = xc[24]*2*B5*xc[2]+xc[25]*2*B5*(b-xc[2]);
    F6_2 = xc[26]*2*B6*b;
}

// h/2 <= xc[2] <= b
else if (status_half == half & status_gap == not_gap){
    Va_1 = -2*B1*xc[2]*d1d_1-2*B1*d1_1*(*uPtrs[0]);
    Vb_1 = -2*B1*b*d1d_1+2*B1*xc[2]*d1d_1+2*B1*d1_1*(*uPtrs[0])
        +B2*h*d2d_1-2*B2*xc[2]*d2d_1-2*B2*d2_1*(*uPtrs[0]);
    Vc_1 = -2*B2*b*d2d_1-B2*h*d2d_1+2*B2*xc[2]*d2d_1+2*B2*d2_1*(*uPtrs[0]);
    Vd_1 = -2*B3*xc[2]*d3d_1-2*B3*d3_1*(*uPtrs[0]);
    Ve_1 = -2*B3*b*d3d_1+2*B3*xc[2]*d3d_1+2*B3*d3_1*(*uPtrs[0])
        +B4*h*d4d_1-2*B4*xc[2]*d4d_1-2*B4*d4_1*(*uPtrs[0]);
    Vf_1 = -2*B4*b*d4d_1-B4*h*d4d_1+2*B4*xc[2]*d4d_1+2*B4*d4_1*(*uPtrs[0]);
    Vg_1 = -2*B5*xc[2]*d5d_1-2*B5*d5_1*(*uPtrs[0]);
    Vh_1 = -2*B5*b*d5d_1+2*B5*xc[2]*d5d_1+2*B5*d5_1*(*uPtrs[0])
        +B6*h*d6d_1-2*B6*xc[2]*d6d_1-2*B6*d6_1*(*uPtrs[0]);
    Vi_1 = -2*B6*b*d6d_1-B6*h*d6d_1+2*B6*xc[2]*d6d_1+2*B6*d6_1*(*uPtrs[0]);
    Vj_1 = 0.0;
    //
    F1_1 = xc[3]*2*B1*xc[2]+xc[4]*2*B1*(b-xc[2]);
    F2_1 = xc[4]*2*B2*(xc[2]-h/2.0)+xc[5]*2*B2*(b-xc[2]+h/2.0);
    F3_1 = xc[6]*2*B3*xc[2]+xc[7]*2*B3*(b-xc[2]);
    F4_1 = xc[7]*2*B4*(xc[2]-h/2.0)+xc[8]*2*B4*(b-xc[2]+h/2.0);
    F5_1 = xc[9]*2*B5*xc[2]+xc[10]*2*B5*(b-xc[2]);
    F6_1 = xc[10]*2*B6*(xc[2]-h/2.0)+xc[11]*2*B6*(b-xc[2]+h/2.0);
    //
    //
    Va_2 = -2*B1*xc[2]*d1d_2-2*B1*d1_2*(*uPtrs[0]);
    Vb_2 = -2*B1*b*d1d_2+2*B1*xc[2]*d1d_2+2*B1*d1_2*(*uPtrs[0])
        +B2*h*d2d_2-2*B2*xc[2]*d2d_2-2*B2*d2_2*(*uPtrs[0]);
    Vc_2 = -2*B2*b*d2d_2-B2*h*d2d_2+2*B2*xc[2]*d2d_2+2*B2*d2_2*(*uPtrs[0]);
    Vd_2 = -2*B3*xc[2]*d3d_2-2*B3*d3_2*(*uPtrs[0]);
    Ve_2 = -2*B3*b*d3d_2+2*B3*xc[2]*d3d_2+2*B3*d3_2*(*uPtrs[0])
        +B4*h*d4d_2-2*B4*xc[2]*d4d_2-2*B4*d4_2*(*uPtrs[0]);
    Vf_2 = -2*B4*b*d4d_2-B4*h*d4d_2+2*B4*xc[2]*d4d_2+2*B4*d4_2*(*uPtrs[0]);
    Vg_2 = -2*B5*xc[2]*d5d_2-2*B5*d5_2*(*uPtrs[0]);
    Vh_2 = -2*B5*b*d5d_2+2*B5*xc[2]*d5d_2+2*B5*d5_2*(*uPtrs[0])
        +B6*h*d6d_2-2*B6*xc[2]*d6d_2-2*B6*d6_2*(*uPtrs[0]);
    Vi_2 = -2*B6*b*d6d_2-B6*h*d6d_2+2*B6*xc[2]*d6d_2+2*B6*d6_2*(*uPtrs[0]);
    Vj_2 = 0.0;
    //
    F1_2 = xc[18]*2*B1*xc[2]+xc[19]*2*B1*(b-xc[2]);
    F2_2 = xc[19]*2*B2*(xc[2]-h/2.0)+xc[20]*2*B2*(b-xc[2]+h/2.0);
    F3_2 = xc[21]*2*B3*xc[2]+xc[22]*2*B3*(b-xc[2]);
    F4_2 = xc[22]*2*B4*(xc[2]-h/2.0)+xc[23]*2*B4*(b-xc[2]+h/2.0);
    F5_2 = xc[24]*2*B5*xc[2]+xc[25]*2*B5*(b-xc[2]);
```

```
    F6_2 = xc[25]*2*B6*(xc[2]-h/2.0)+xc[26]*2*B6*(b-xc[2]+h/2.0);
}

// b <= xc[2] <= h
else if (status_half == half & status_gap == gap){
    Va_1 = -2*B1*b*d1d_1;
    Vb_1 = -2*B2*xc[2]*d2d_1-2*B2*d2_1*(*uPtrs[0])+B2*h*d2d_1;
    Vc_1 = -2*B2*b*d2d_1-B2*h*d2d_1+2*B2*xc[2]*d2d_1+2*B2*d2_1*(*uPtrs[0]);
    Vd_1 = -2*B3*b*d3d_1;
    Ve_1 = -2*B4*xc[2]*d4d_1-2*B4*d4_1*(*uPtrs[0])+B4*h*d4d_1;
    Vf_1 = -2*B4*b*d4d_1-B4*h*d4d_1+2*B4*xc[2]*d4d_1+2*B4*d4_1*(*uPtrs[0]);
    Vg_1 = -2*B5*b*d5d_1;
    Vh_1 = -2*B6*xc[2]*d6d_1-2*B6*d6_1*(*uPtrs[0])+B6*h*d6d_1;
    Vi_1 = -2*B6*b*d6d_1-B6*h*d6d_1+2*B6*xc[2]*d6d_1+2*B6*d6_1*(*uPtrs[0]);
    Vj_1 = 0.0;
    //
    F1_1 = xc[3]*2*B1*b;
    F2_1 = xc[4]*2*B2*(xc[2]-h/2.0)+xc[5]*2*B2*(b-xc[2]+h/2.0);
    F3_1 = xc[6]*2*B3*b;
    F4_1 = xc[7]*2*B4*(xc[2]-h/2.0)+xc[8]*2*B4*(b-xc[2]+h/2.0);
    F5_1 = xc[9]*2*B5*b;
    F6_1 = xc[10]*2*B6*(xc[2]-h/2.0)+xc[11]*2*B6*(b-xc[2]+h/2.0);
    //
    //
    Va_2 = -2*B1*b*d1d_2;
    Vb_2 = -2*B2*xc[2]*d2d_2-2*B2*d2_2*(*uPtrs[0])+B2*h*d2d_2;
    Vc_2 = -2*B2*b*d2d_2-B2*h*d2d_2+2*B2*xc[2]*d2d_2+2*B2*d2_2*(*uPtrs[0]);
    Vd_2 = -2*B3*b*d3d_2;
    Ve_2 = -2*B4*xc[2]*d4d_2-2*B4*d4_2*(*uPtrs[0])+B4*h*d4d_2;
    Vf_2 = -2*B4*b*d4d_2-B4*h*d4d_2+2*B4*xc[2]*d4d_2+2*B4*d4_2*(*uPtrs[0]);
    Vg_2 = -2*B5*b*d5d_2;
    Vh_2 = -2*B6*xc[2]*d6d_2-2*B6*d6_2*(*uPtrs[0])+B6*h*d6d_2;
    Vi_2 = -2*B6*b*d6d_2-B6*h*d6d_2+2*B6*xc[2]*d6d_2+2*B6*d6_2*(*uPtrs[0]);
    Vj_2 = 0.0;
    //
    F1_2 = xc[18]*2*B1*b;
    F2_2 = xc[19]*2*B2*(xc[2]-h/2.0)+xc[20]*2*B2*(b-xc[2]+h/2.0);
    F3_2 = xc[21]*2*B3*b;
    F4_2 = xc[22]*2*B4*(xc[2]-h/2.0)+xc[23]*2*B4*(b-xc[2]+h/2.0);
    F5_2 = xc[24]*2*B5*b;
    F6_2 = xc[25]*2*B6*(xc[2]-h/2.0)+xc[26]*2*B6*(b-xc[2]+h/2.0);
}

// the first meter of the track does not have levitation coils.
// this if-loop implements this non linearity.
if (1-xc[15]>0){
    Va_1=0.0; Vb_1=0.0; Vc_1=0.0; Vd_1=0.0; Ve_1=0.0; Vf_1=0.0; Vg_1=0.0;
    Vh_1=0.0; Vi_1=0.0; Vj_1=0.0;
    Va_2=0.0; Vb_2=0.0; Vc_2=0.0; Vd_2=0.0; Ve_2=0.0; Vf_2=0.0; Vg_2=0.0;
    Vh_2=0.0; Vi_2=0.0; Vj_2=0.0;
    piwrk[0]=0;
}
else if (1+h-xc[15]>0){
    Vb_1=0.0; Vc_1=0.0; Vd_1=0.0; Ve_1=0.0; Vf_1=0.0; Vg_1=0.0; Vh_1=0.0;
```

```
    Vi_1=0.0; Vj_1=0.0;
    Vb_2=0.0; Vc_2=0.0; Vd_2=0.0; Ve_2=0.0; Vf_2=0.0; Vg_2=0.0; Vh_2=0.0;
    Vi_2=0.0; Vj_2=0.0;
}
else if (1+2*h-xc[15]>0){
    Vc_1=0.0; Vd_1=0.0; Ve_1=0.0; Vf_1=0.0; Vg_1=0.0; Vh_1=0.0; Vi_1=0.0; Vj_1=0.0;
    Vc_2=0.0; Vd_2=0.0; Ve_2=0.0; Vf_2=0.0; Vg_2=0.0; Vh_2=0.0; Vi_2=0.0; Vj_2=0.0;
}
else if (1+3*h-xc[15]>0){
    Vd_1=0.0; Ve_1=0.0; Vf_1=0.0; Vg_1=0.0; Vh_1=0.0; Vi_1=0.0; Vj_1=0.0;
    Vd_2=0.0; Ve_2=0.0; Vf_2=0.0; Vg_2=0.0; Vh_2=0.0; Vi_2=0.0; Vj_2=0.0;
}
else if (1+4*h-xc[15]>0){
    Ve_1=0.0; Vf_1=0.0; Vg_1=0.0; Vh_1=0.0; Vi_1=0.0; Vj_1=0.0;
    Ve_2=0.0; Vf_2=0.0; Vg_2=0.0; Vh_2=0.0; Vi_2=0.0; Vj_2=0.0;
}
else if (1+5*h-xc[15]>0){
    Vf_1=0.0; Vg_1=0.0; Vh_1=0.0; Vi_1=0.0; Vj_1=0.0;
    Vf_2=0.0; Vg_2=0.0; Vh_2=0.0; Vi_2=0.0; Vj_2=0.0;
}
else if (1+6*h-xc[15]>0){
    Vg_1=0.0; Vh_1=0.0; Vi_1=0.0; Vj_1=0.0;
    Vg_2=0.0; Vh_2=0.0; Vi_2=0.0; Vj_2=0.0;
}
else if (1+7*h-xc[15]>0){
    Vh_1=0.0; Vi_1=0.0; Vj_1=0.0;
    Vh_2=0.0; Vi_2=0.0; Vj_2=0.0;
}
else if (1+8*h-xc[15]>0){
    Vi_1=0.0; Vj_1=0.0;
    Vi_2=0.0; Vj_2=0.0;
}
else if (1+9*h-xc[15]>0){
    Vj_1=0.0;
    Vj_2=0.0;
}

// calculate forces and torques
F_1     =   N*(F1_1+F2_1+F3_1+F4_1+F5_1+F6_1);
F_2     =   N*(F1_2+F2_2+F3_2+F4_2+F5_2+F6_2);
F       =   F_1+F_2;
Tp      =   N*((F1_1+F1_2)*L1 + (F2_1+F2_2)*L2 + (F3_1+F3_2)*L3 + (F4_1+F4_2)*L4
            + (F5_1+F5_2)*L5 + (F6_1+F6_2)*L6)*cos(xc[13]);
Tr      =   (F_1*D1+F_2*D2)*cos(xc[16]);
Fd      =   0.5*rho_air*(*uPtrs[0])*(*uPtrs[0])*A_front*C_front;
Td      =   Fd*Fd_arm;


// differential equations
dx[0]   =   xc[1];
dx[1]   =   F/M - g;
dx[2]   =   *uPtrs[0];
dx[3]   =   1/L*(Va_1-R*xc[3]);
dx[4]   =   1/L*(Vb_1-R*xc[4]);
```

```
dx[5]   =   1/L*(Vc_1-R*xc[5]);
dx[6]   =   1/L*(Vd_1-R*xc[6]);
dx[7]   =   1/L*(Ve_1-R*xc[7]);
dx[8]   =   1/L*(Vf_1-R*xc[8]);
dx[9]   =   1/L*(Vg_1-R*xc[9]);
dx[10]  =   1/L*(Vh_1-R*xc[10]);
dx[11]  =   1/L*(Vi_1-R*xc[11]);
dx[12]  =   1/L*(Vj_1-R*xc[12]);
dx[13]  =   xc[14];
dx[14]  =   (Tp+dTp+Td)/Jp;
dx[15]  =   *uPtrs[0];
dx[16]  =   xc[17];
dx[17]  =   (Tr+dTr)/Jr;
dx[18]  =   1/L*(Va_2-R*xc[18]);
dx[19]  =   1/L*(Vb_2-R*xc[19]);
dx[20]  =   1/L*(Vc_2-R*xc[20]);
dx[21]  =   1/L*(Vd_2-R*xc[21]);
dx[22]  =   1/L*(Ve_2-R*xc[22]);
dx[23]  =   1/L*(Vf_2-R*xc[23]);
dx[24]  =   1/L*(Vg_2-R*xc[24]);
dx[25]  =   1/L*(Vh_2-R*xc[25]);
dx[26]  =   1/L*(Vi_2-R*xc[26]);
dx[27]  =   1/L*(Vj_2-R*xc[27]);

// check position of the corners (wheels) of the vehicle
z1      =   xc[0]+0.5*LL*sin(xc[13])+0.5*DD*sin(xc[16]);
z2      =   xc[0]-0.5*LL*sin(xc[13])+0.5*DD*sin(xc[16]);
z3      =   xc[0]-0.5*LL*sin(xc[13])-0.5*DD*sin(xc[16]);
z4      =   xc[0]+0.5*LL*sin(xc[13])-0.5*DD*sin(xc[16]);

z1_ground=0;    z2_ground=0;    z3_ground=0;    z4_ground=0;
z1_ceil=0;      z2_ceil=0;      z3_ceil=0;      z4_ceil=0;

if(z1<ground)   {z1=ground; z1_ground=1;}
if(z1>ceil)     {z1=ceil;   z1_ceil=1;}
if(z2<ground)   {z2=ground; z2_ground=1;}
if(z2>ceil)     {z2=ceil;   z2_ceil=1;}
if(z3<ground)   {z3=ground; z3_ground=1;}
if(z3>ceil)     {z3=ceil;   z3_ceil=1;}
if(z4<ground)   {z4=ground; z4_ground=1;}
if(z4>ceil)     {z4=ceil;   z4_ceil=1;}

// correct position if upper- or lower-bound is reached
if( z1_ground == 1 | z1_ceil == 1){
    xc[0]   =   z1 - 0.5*(z1-z2) - 0.5*(z1-z4);
    xc[1]   =   0.0;
    xc[13]  =   asin((z1-z2)/LL);
    xc[14]  =   0.0;
    xc[16]  =   asin((z1-z4)/DD);
    xc[17]  =   0.0;
}
if (z2_ground == 1 | z2_ceil == 1){
    xc[0]   =   z2 + 0.5*(z1-z2) - 0.5*(z2-z3);
    xc[1]   =   0.0;
```

```
        xc[13]  =   asin((z1-z2)/LL);
        xc[14]  =   0.0;
        xc[16]  =   asin((z2-z3)/DD);
        xc[17]  =   0.0;
    }
    if (z3_ground == 1 | z3_ceil == 1){
        xc[0]   =   z3 + 0.5*(z4-z3) + 0.5*(z2-z3);
        xc[1]   =   0.0;
        xc[13]  =   asin((z4-z3)/LL);
        xc[14]  =   0.0;
        xc[16]  =   asin((z2-z3)/DD);
        xc[17]  =   0.0;
    }
    if (z4_ground == 1 | z4_ceil == 1){
        xc[0]   =   z4 - 0.5*(z4-z3) + 0.5*(z1-z4);
        xc[1]   =   0.0;
        xc[13]  =   asin((z4-z3)/LL);
        xc[14]  =   0.0;
        xc[16]  =   asin((z1-z4)/DD);
        xc[17]  =   0.0;
    }

    // debug outputs
    prwrk[0]=z1;
    prwrk[1]=z2;
    prwrk[2]=z3;
    prwrk[3]=z4;
    prwrk[4]=Tp;
    prwrk[5]=Td;
}


//=============================================================================
// mdlTerminate()
//=============================================================================

static void mdlTerminate(SimStruct *S)
{
}

#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Appendix C

M-File for parameter estimation

```
%   parameter estimation of the maglev model
%   first set of force equations

%   geometric constants and estimates used for non-linear terms
h   = 5.08e-2;      % width of magnet [m]
b   = 4.48e-2;      % width of coil [m]
d   = 0.6e-2;       % gap between coils [m]
N   = 100.0;        % number of coil windings (-)
B   = 1.25;         % magnetic field strenght (T)

L1  = 0.1905;       % distance of magnet one to centre of inertia (m)
L2  = 0.1143;       % distance of magnet two to c.o.i (m)
L3  = 0.0381;       % distance of magnet three to c.o.i. (m)
L4  = -0.0381;      % distance of magnet four to c.o.i. (m)
L5  = -0.1143;      % distance of magnet five to c.o.i (m)
L6  = -0.1905;      % distance of magnet six to c.o.i (m)
D1  = 0.0368;       % distance from magnet row 1 to c.o.i (m)
D2  = -0.0368;      % distance from magnet row 2 to c.o.i (m)

M   = 6.0;          % mass of the vehicle (kg)
Jp  = 0.1437;       % pitch rotational inertia (kg*m^2)
Jr  = 0.0288;       % roll rotational inertia (kg*m^2)

%   definition of magnets
B1  = B;
B2  = -B;
B3  = B;
B4  = -B;
B5  = B;
B6  = -B;

%   measurements from workspace (sled)
%   x   position measurement
%   v   velocity measurement
%   z   height measurement
%   p   pitch measurement
%   r   roll measurement

fs  =   65000;      % sampling frequency
```

```
ts  =   1/fs;       % sample time
t   =   t-t(1)*ones(size(t));

%   measured currents from workspace (figure-8 coils)
%   1=right row, 2=left row
% ia1 ib1 ic1 id1 ie1 if1 ig1 ih1 ii1 ij1 ia2 ib2 ic2 id2 ie2 if2 ig2 ih2 ii2 ij2

%convert b, d and h to vectors
b=b*ones(size(x));
d=d*ones(size(x));
h=h*ones(size(x));

%   construction of the 60 inputs
%height                                  estimated constant
ua1=2*x.*N.*ia1;                         %B1/M   right row
ub1=2*(b-x).*N.*ib1;                     %B1/M
uc1=2*(h./2.0+x).*N.*ic1;                %B2/M
ud1=(2*B2*(h./2.0-d-x)+2*B3*x).*N.*id1; %1/M
ue1=2*(b-x).*N.*ie1;                     %B3/M
uf1=2*(h./2.0+x).*N.*if1;                %B4/M
ug1=(2*B4*(h./2.0-d-x)+2*B5*x).*N.*ig1; %1/M
uh1=2*(b-x).*N.*ih1;                     %B5/M
ui1=2*(h./2.0+x).*N.*ii1;                %B6/M
uj1=2*(h./2.0-d-x).*N.*ij1;              %B6/M
ua2=2*x.*N.*ia2;                         %B1/M   left row
ub2=2*(b-x).*N.*ib2;                     %B1/M
uc2=2*(h./2.0+x).*N.*ic2;                %B2/M
ud2=(2*B2*(h./2.0-d-x)+2*B3*x).*N.*id2; %1/M
ue2=2*(b-x).*N.*ie2;                     %B3/M
uf2=2*(h./2.0+x).*N.*if2;                %B4/M
ug2=(2*B4*(h./2.0-d-x)+2*B5*x).*N.*ig2; %1/M
uh2=2*(b-x).*N.*ih2;                     %B5/M
ui2=2*(h./2.0+x).*N.*ii2;                %B6/M
uj2=2*(h./2.0-d-x).*N.*ij2;              %B6/M

u=[ua1,ub1,uc1,ud1,ue1,uf1,ug1,uh1,ui1,uj1,ua2,ub2,uc2,ud2,ue2,uf2,ug2,uh2,ui2,uj2];

%pitch                                    estimated constant
ua1p=2*N*x.*cos(p).*ia1;                 %B1*L1/Jp   right row
ub1p=2*N*(b-x).*cos(p).*ib1;             %B1*L1/Jp
uc1p=2*N*(h./2.0+x).*cos(p).*ic1;        %B2*L2/Jp
ud1p=N*(2*B2*(h./2.0-d-x).*L2+2*B3*x).*cos(p).*id1; %1/Jp
ue1p=2*N*(b-x).*cos(p).*ie1;             %B3*L3/Jp
uf1p=2*N*(h/2.0+x).*cos(p).*if1;         %B4*L4/Jp
ug1p=N*(2*B4*(h./2.0-d-x).*L4+2*B5*x).*cos(p).*ig1;  %1/Jp
uh1p=2*N*(b-x).*cos(p).*ih1;             %B5*L5/Jp
ui1p=2*N*(h./2.0+x).*cos(p).*ii1;        %B6*L6/Jp
uj1p=2*N*(h./2.0-d-x).*cos(p).*ij1;      %B6*L6/Jp
ua2p=2*N*x.*cos(p).*ia2;                 %B1*L1/Jp   left row
ub2p=2*N*(b-x).*cos(p).*ib2;             %B1*L1/Jp
uc2p=2*N*(h./2.0+x).*cos(p).*ic2;        %B2*L2/Jp
ud2p=N*(2*B2*(h./2.0-d-x).*L2+2*B3*x).*cos(p).*id2; %1/Jp
ue2p=2*N*(b-x).*cos(p).*ie2;             %B3*L3/Jp
uf2p=2*N*(h./2.0+x).*cos(p).*if2;        %B4*L4/Jp
```

```
ug2p=N*(2*B4*(h./2.0-d-x).*L4+2*B5*x).*cos(p).*ig2; %1/Jp
uh2p=2*N*(b-x).*cos(p).*ih2;              %B5*L5/Jp
ui2p=2*N*(h./2.0+x).*cos(p).*ii2;         %B6*L6/Jp
uj2p=2*N*(h./2.0-d-x).*cos(p).*ij2;       %B6*L6/Jp

u=[u,ua1p,ub1p,uc1p,ud1p,ue1p,uf1p,ug1p,uh1p,ui1p,uj1p,...
    ua2p,ub2p,uc2p,ud2p,ue2p,uf2p,ug2p,uh2p,ui2p,uj2p];

%roll                                     estimated constant
ua1r=2*x.*N.*cos(r).*ia1;                 %B1*D1/Jr   right row
ub1r=2*(b-x).*N.*cos(r).*ib1;             %B1*D1/Jr
uc1r=2*(h./2.0+x).*N.*cos(r).*ic1;        %B2*D1/Jr
ud1r=(2*B2*(h./2.0-d-x)+2*B3*x).*N*D1.*cos(r).*id1;   %1/Jr
ue1r=2*(b-x).*N.*cos(r).*ie1;             %B3*D1/Jr
uf1r=2*(h./2.0+x).*N.*cos(r).*if1;        %B4*D1/Jr
ug1r=(2*B4*(h./2.0-d-x)+2*B5*x).*N*D1.*cos(r).*ig1;   %1/Jr
uh1r=2*(b-x).*N.*cos(r).*ih1;             %B5*D1/Jr
ui1r=2*(h./2.0+x).*N.*cos(r).*ii1;        %B6*D1/Jr
uj1r=2*(h./2.0-d-x)*N.*cos(r).*ij1;       %B6*D1/Jr
ua2r=2*x*N*D2.*cos(r).*ia2;                  %B1*D2/Jr  left row
ub2r=2*(b-x)*N.*cos(r).*ib2;              %B1*D2/Jr
uc2r=2*(h./2.0+x)*N.*cos(r).*ic2;         %B2*D2/Jr
ud2r=(2*B2*(h./2.0-d-x)+2*B3*x).*N*D2.*cos(r).*id2;  %1/Jr
ue2r=2*(b-x)*N.*cos(r).*ie2;              %B3*D2/Jr
uf2r=2*(h./2.0+x)*N.*cos(r).*if2;         %B4*D2/Jr
ug2r=(2*B4*(h./2.0-d-x)+2*B5*x).*N*D2.*cos(r).*ig2;  %1/Jr
uh2r=2*(b-x)*N.*cos(r).*ih2;              %B5*D2/Jr
ui2r=2*(h./2.0+x)*N.*cos(r).*ii2;         %B6*D2/Jr
uj2r=2*(h./2.0-d-x)*N.*cos(r).*ij2;       %B6*D2/Jr

u=[u,ua1r,ub1r,uc1r,ud1r,ue1r,uf1r,ug1r,uh1r,ui1r,uj1r,...
    ua2r,ub2r,uc2r,ud2r,ue2r,uf2r,ug2r,uh2r,ui2r,uj2r];

%define an additional input to add gravity to the estimation model
ugravity=-9.8*ones(size(x));
%unbalance pitch estimation
udtp=ones(size(x));
%unbalance roll estimation
udtr=ones(size(x));
%aerodynamic drag on pitch estimation (A*Cl*arm)
rho=1.0; %density of air
udragp=0.5*rho*v.^2;

%input matrix containing 64 inputs
u=[u,ugravity,udtp,udtr,udragp];

%outputmatrix
y=[z,p,r];

%state-space formulation
%A-matrix (6x6)
A=[0 1 0 0 0 0;0 0 0 0 0 0;0 0 0 1 0 0;0 0 0 0 0 0;0 0 0 0 0 1;0 0 0 0 0 0];

%B-matrix (6x61)
```

```
b0=zeros(1,20);
bz=[B1/M B1/M B2/M 1/M B3/M B4/M 1/M B5/M B6/M B6/M...
     B1/M B1/M B2/M 1/M B3/M B4/M 1/M B5/M B6/M B6/M];
bp=[B1*L1/Jp B1*L1/Jp B2*L2/Jp 1/Jp B3*L3/Jp B4*L4/Jp 1/Jp B5*L5/Jp B6*L6/Jp B6*L6/Jp... B1*L1/Jp B1
br=[B1*D1/Jr B1*D1/Jr B2*D1/Jr 1/Jr B3*D1/Jr B4*D1/Jr 1/Jr B5*D1/Jr B6*D1/Jr B6*D1/Jr... B1*D2/Jr B1
B=[b0,b0,b0,0,0,0,0;...
   bz,b0,b0,1,0,0,0;...
   b0,b0,b0,0,0,0,0;...
   b0,bp,b0,0,1,0,1;...
   b0,b0,b0,0,0,0,0;...
   b0,b0,br,0,0,1,0];

%C-matrix (3x6)
C=[1 0 0 0 0 0;0 0 1 0 0 0;0 0 0 0 1 0];
%D-matrix (3x61)
D=zeros(3,64);
%K-matrix (6x3)
K=zeros(6,3);

%x0
z0=z(1);
z0dot=(z(2)-z(1))/ts;
p0=p(1);
p0dot=(p(2)-p(1))/ts;
r0=r(1);
r0dot=(r(2)-r(1))/ts;
x0=[z0;z0dot;p0;p0dot;r0;r0dot];

%initialize model for estimation
m0=idss(A,B,C,D,K,x0,'Ts',0);
m0.As=[0 1 0 0 0 0;0 0 0 0 0 0;0 0 0 1 0 0;0 0 0 0 0 0;0 0 0 0 0 1;0 0 0 0 0 0];
bnan=[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN];
m0.Bs=[b0,b0,b0,0,0,0,0;...
       bnan,b0,b0,1,0,0,0;...
       b0,b0,b0,0,0,0,0;...
       b0,bnan,b0,0,NaN,0,NaN;...
       b0,b0,b0,0,0,0,0;...
       b0,b0,bnan,0,0,NaN,0];
m0.Cs=[1 0 0 0 0 0;0 0 1 0 0 0;0 0 0 0 1 0];
m0.Ds=zeros(3,64);
m0.Ks=zeros(6,3);
m0.x0s=[z0;z0dot;p0;p0dot;r0;r0dot];
m0.Ts=0;

%initialize data set for estimation
dataset=iddata(y,u,ts);
%fit model to data
mfit=pem(dataset,m0,'InitialState','Fixed');
%convert idmodel to ssmodel
ssfit=ss(mfit);
%get state-space matrices
[Af,Bf,Cf,Df]=ssdata(ssfit);

%get parameter estimates
```

```
bzfit=Bf(2,[1:20]);
bpfit=Bf(4,[21:40]);
brfit=Bf(6,[41:60]);

disp('DISTURBANCE ESTIMATES')
unbalance_pitch=Bf(4,62)
unbalance_roll=Bf(6,63)
drag_coefficient=Bf(4,64)

disp('RIGHT COIL ROW ESTIMATES')
disp(['B1 = ',num2str(M*bzfit(1))])
disp(['B1 = ',num2str(M*bzfit(2))])
disp(['B2 = ',num2str(M*bzfit(3))])
disp(['M  = ',num2str(1/bzfit(4))])
disp(['B3 = ',num2str(M*bzfit(5))])
disp(['B4 = ',num2str(M*bzfit(6))])
disp(['M  = ',num2str(1/bzfit(7))])
disp(['B5 = ',num2str(M*bzfit(8))])
disp(['B6 = ',num2str(M*bzfit(9))])
disp(['B6 = ',num2str(M*bzfit(10))])
disp('---')
disp(['B1*L1/Jp = ',num2str(bpfit(1))])
disp(['B1*L1/Jp = ',num2str(bpfit(2))])
disp(['B2*L2/Jp = ',num2str(bpfit(3))])
disp(['Jp       = ',num2str(1/bpfit(4))])
disp(['B3*L3/Jp = ',num2str(bpfit(5))])
disp(['B4*L4/Jp = ',num2str(bpfit(6))])
disp(['Jp       = ',num2str(1/bpfit(7))])
disp(['B5*L5/Jp = ',num2str(bpfit(8))])
disp(['B6*L6/Jp = ',num2str(bpfit(9))])
disp(['B6*L6/Jp = ',num2str(bpfit(10))])
disp('---')
disp(['B1*D1/Jr = ',num2str(brfit(1))])
disp(['B1*D1/Jr = ',num2str(brfit(2))])
disp(['B2*D1/Jr = ',num2str(brfit(3))])
disp(['Jr       = ',num2str(1/brfit(4))])
disp(['B3*D1/Jr = ',num2str(brfit(5))])
disp(['B4*D1/Jr = ',num2str(brfit(6))])
disp(['Jr       = ',num2str(1/brfit(7))])
disp(['B5*D1/Jr = ',num2str(brfit(8))])
disp(['B6*D1/Jr = ',num2str(brfit(9))])
disp(['B6*D1/Jr = ',num2str(brfit(10))])
disp('---')
disp('LEFT COIL ROW ESTIMATES')
disp(['B1 = ',num2str(M*bzfit(11))])
disp(['B1 = ',num2str(M*bzfit(12))])
disp(['B2 = ',num2str(M*bzfit(13))])
disp(['M  = ',num2str(1/bzfit(14))])
disp(['B3 = ',num2str(M*bzfit(15))])
disp(['B4 = ',num2str(M*bzfit(16))])
disp(['M  = ',num2str(1/bzfit(17))])
disp(['B5 = ',num2str(M*bzfit(18))])
disp(['B6 = ',num2str(M*bzfit(19))])
disp(['B6 = ',num2str(M*bzfit(20))])
```

```
disp('---')
disp(['B1*L1/Jp = ',num2str(bpfit(11))])
disp(['B1*L1/Jp = ',num2str(bpfit(12))])
disp(['B2*L2/Jp = ',num2str(bpfit(13))])
disp(['Jp       = ',num2str(1/bpfit(14))])
disp(['B3*L3/Jp = ',num2str(bpfit(15))])
disp(['B4*L4/Jp = ',num2str(bpfit(16))])
disp(['Jp       = ',num2str(1/bpfit(17))])
disp(['B5*L5/Jp = ',num2str(bpfit(18))])
disp(['B6*L6/Jp = ',num2str(bpfit(19))])
disp(['B6*L6/Jp = ',num2str(bpfit(20))])
disp('---')
disp(['B1*D2/Jr = ',num2str(brfit(11))])
disp(['B1*D2/Jr = ',num2str(brfit(12))])
disp(['B2*D2/Jr = ',num2str(brfit(13))])
disp(['Jr       = ',num2str(1/brfit(14))])
disp(['B3*D2/Jr = ',num2str(brfit(15))])
disp(['B4*D2/Jr = ',num2str(brfit(16))])
disp(['Jr       = ',num2str(1/brfit(17))])
disp(['B5*D2/Jr = ',num2str(brfit(18))])
disp(['B6*D2/Jr = ',num2str(brfit(19))])
disp(['B6*D2/Jr = ',num2str(brfit(20))])

%   plot fitted model and data
Y=sim(mfit,iddata([],u,ts));
t=[0:ts:200*ts-ts].';

%   levitation
figure
plot(t,Y(:,1),'b-')
hold
plot(t,z,'r--')
hold off
title('fitted model and levitation height')
legend('fitted model','data')
xlabel('time [s]')
ylabel('height [m]')

%   pitch
figure
plot(t,Y(:,2),'b-')
hold on
plot(t,p,'r--')
hold off
title('fitted model and pitch')
legend('fitted model','data')
xlabel('time [s]')
ylabel('pitch [rad]')

%   roll
figure
plot(t,Y(:,3),'b-')
hold on
plot(t,r,'r--')
```

```
hold off
title('fitted model and roll')
legend('fitted model','data')
xlabel('time [s]')
ylabel('roll [rad]')
```

The parameter estimation files for the other three set of equations are basically the same as this file. The non-linear equations will be slightly different as are the estimated parameters.

Appendix D

C-code for Feedback Control Simulation

D.1 C-code Sliding Mode Controller

```
//=============================================================================
// setup
//=============================================================================

#define S_FUNCTION_NAME smciprff
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
#include "math.h"

#define h           5.08e-2     // width of magnet [m]
#define b           4.48e-2     // width of coil [m]
#define d           0.6e-2      // gap between coils [m]
#define N           100.0       // number of coil windings (-)
#define B           1.25        // magnetic field strenght (T)

#define L1          0.1905      // distance of magnet one to centre of inertia (m)
#define L2          0.1143      // distance of magnet two to c.o.i (m)
#define L3          0.0381      // distance of magnet three to c.o.i. (m)
#define L4          -0.0381     // distance of magnet four to c.o.i. (m)
#define L5          -0.1143     // distance of magnet five to c.o.i (m)
#define L6          -0.1905     // distance of magnet six to c.o.i (m)
#define D1          0.0368      // distance from magnet row 1 to c.o.i (m)
#define D2          -0.0368     // distance from magnet row 2 to c.o.i (m)

#define M           6.0         // mass of the vehicle (kg)
#define Jp          0.1437      // pitch rotational inertia (kg*m^2)
#define Jr          0.0288      // roll rotational inertia (kg*m^2)

#define phi         50.0        // parameter for sign function estimate (2/pi*atan(phi*s))
#define Kgain       300.0       // filter gain for velocity estimate

#define pi          3.14159265358979

#define NINPUTS     15          // number of inputs      uPtrs: input vector
```

```
// 0. velocity
// 1. error delta
// 2. error pitch
// 3. error roll
// 4. L1, sliding parameter for delta
// 5. L2, sliding parameter for pitch
// 6. L3, sliding parameter for roll
// 7. Labda1, switching parameter for delta
// 8. Labda2, switching parameter for pitch
// 9. Labda3, switching parameter for roll
// 10. pitch
// 11. roll
// 12. z acceleration
// 13. p acceleration
// 14. r acceleration

#define NOUTPUTS    20          // number of outputs     yPtrs: output vector
// 0-19. control outputs

#define NCSTATES    7           // number of continuous states
// 0. x
// 1. error delta  (filtered for velocity estimate)
// 2. error pitch  (filtered for velocity estimate)
// 3. error roll   (filtered for velocity estimate)
// 4. integrant error delta
// 5. integrant error pitch
// 6. integrant error roll

#define NIWRK       0           // number of integer global variables
#define NRWRK       3           // number of real global variables
// 0. error_dot delta   (from filter)
// 1. error_dot pitch   (from filter)
// 2. error_dot roll    (from filter)


//=============================================================================
// includes
//=============================================================================

#include "invcal.c"
// calculates matrix inversion of 3x3 matrix

//=============================================================================
// mdlInitializeSizes
//=============================================================================

static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S,0);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {return;}

    ssSetNumContStates(S,NCSTATES);
    ssSetNumDiscStates(S,0);
```

```
    if (!ssSetNumInputPorts(S,1)) {return;}
    ssSetInputPortWidth(S,0,NINPUTS);
    ssSetInputPortDirectFeedThrough(S,0,0);

    if (!ssSetNumOutputPorts(S,1)) {return;}
    ssSetOutputPortWidth(S,0,NOUTPUTS);

    ssSetNumSampleTimes(S,1);
    ssSetNumRWork(S,NRWRK);
    ssSetNumIWork(S,NIWRK);
    ssSetNumPWork(S,0);
    ssSetNumModes(S,0);
    ssSetNumNonsampledZCs(S,0);
}


//=============================================================================
// mdlInitializeSampleTimes
//=============================================================================

static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S,0,CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S,0,0.0);
}


//=============================================================================
// mdlInitializeConditions
//=============================================================================

#define MDL_INITIALIZE_CONDITIONS

static void mdlInitializeConditions(SimStruct *S)
{
    real_T  *xc =   ssGetContStates(S);
    real_T  *dx =   ssGetdX(S);

    real_T  *prwrk  =   ssGetRWork(S);
    int_T   *piwrk  =   ssGetIWork(S);
    int_T   i;

    for (i=0;i<NCSTATES;i++){
        xc[i]=0.0;
    }
    for (i=0;i<NRWRK;i++){
        prwrk[i]=0.0;
    }
}

//=============================================================================
// mdlOutputs
//=============================================================================
```

```
static void mdlOutputs(SimStruct *S, int_T tid)
{
    real_T  *xc     =   ssGetContStates(S);
    real_T  *yPtrs  =   ssGetOutputPortRealSignal(S,0);

    InputRealPtrsType   uPtrs   =   ssGetInputPortRealSignalPtrs(S,0);

    int_T   *piwrk  =   ssGetIWork(S);
    real_T  *prwrk  =   ssGetRWork(S);

    real_T  Blin[3][20],T[20][3],A[3][3];
    real_T  s[3],e1[3],e2[3],e3[3],L[3],v[3],vs[3],labda[3];
    real_T  sum0,sum1,sum2,sum3,sum4,sum5,sum6;
    real_T  x,cosp,cosr;
    int_T   i,j,k;

    int_T   status_half,status_gap_half,status_gap,status;
    int_T   not_half        =   0;  // (xc[2]<h/2)
    int_T   half            =   1;  // (xc[2]>h/2)
    int_T   not_gap_half    =   0;  // (xc[2]<h/2-d)
    int_T   gap_half        =   1;  // (xc[2]>h/2-d)
    int_T   not_gap         =   0;  // (xc[2]<b)
    int_T   gap             =   1;  // (xc[2]>b)

    real_T  B1          =   B;
    real_T  B2          =   -B;
    real_T  B3          =   B;
    real_T  B4          =   -B;
    real_T  B5          =   B;
    real_T  B6          =   -B;

    // check in which status (position )the magnet is in
    if      ( h/2.0-xc[0] < 0)      {status_half=half;}
    else                            {status_half=not_half;}
    if      ( h/2.0-d-xc[0] < 0)    {status_gap_half=gap_half;}
    else                            {status_gap_half=not_gap_half;}
    if      ( b-xc[0] < 0)          {status_gap=gap;}
    else                            {status_gap=not_gap;}

    if  (status_half==not_half  & status_gap_half==not_gap_half)    status=1;
    if  (status_half==not_half  & status_gap_half==gap_half)        status=2;
    if  (status_half==half      & status_gap==not_gap)              status=3;
    if  (status_half==half      & status_gap==gap)                  status=4;

    x       =   xc[0];
    cosp    =   cos(*uPtrs[10]);
    cosr    =   cos(*uPtrs[11]);

    // calculate B matrix at given postion, depending on status
    if (status==1){
        // levitation
        Blin[0][0]=2*B1*x*N/M;
        Blin[0][1]=2*B1*(b-x)*N/M;
        Blin[0][2]=2*B2*(h/2.0+x)*N/M;
```

```
Blin[0][3]=(2*B2*(h/2.0-d-x)+2*B3*x)*N/M;
Blin[0][4]=2*B3*(b-x)*N/M;
Blin[0][5]=2*B4*(h/2.0+x)*N/M;
Blin[0][6]=(2*B4*(h/2.0-d-x)+2*B5*x)*N/M;
Blin[0][7]=2*B5*(b-x)*N/M;
Blin[0][8]=2*B6*(h/2.0+x)*N/M;
Blin[0][9]=2*B6*(h/2.0-d-x)*N/M;
Blin[0][10]=2*B1*x*N/M;
Blin[0][11]=2*B1*(b-x)*N/M;
Blin[0][12]=2*B2*(h/2.0+x)*N/M;
Blin[0][13]=(2*B2*(h/2.0-d-x)+2*B3*x)*N/M;
Blin[0][14]=2*B3*(b-x)*N/M;
Blin[0][15]=2*B4*(h/2.0+x)*N/M;
Blin[0][16]=(2*B4*(h/2.0-d-x)+2*B5*x)*N/M;
Blin[0][17]=2*B5*(b-x)*N/M;
Blin[0][18]=2*B6*(h/2.0+x)*N/M;
Blin[0][19]=2*B6*(h/2.0-d-x)*N/M;

// pitch
Blin[1][0]= 2*N*B1*x*L1*cosp/Jp;
Blin[1][1]=2*N*B1*(b-x)*L1*cosp/Jp;
Blin[1][2]=2*N*B2*(h/2.0+x)*L2*cosp/Jp;
Blin[1][3]=N*(2*B2*(h/2.0-d-x)*L2+2*B3*x*L3)*cosp/Jp;
Blin[1][4]=2*N*B3*(b-x)*L3*cosp/Jp;
Blin[1][5]=2*N*B4*(h/2.0+x)*L4*cosp/Jp;
Blin[1][6]=N*(2*B4*(h/2.0-d-x)*L4+2*B5*x*L5)*cosp/Jp;
Blin[1][7]=2*N*B5*(b-x)*L5*cosp/Jp;
Blin[1][8]=2*N*B6*(h/2.0+x)*L6*cosp/Jp;
Blin[1][9]=2*N*B6*(h/2.0-d-x)*L6*cosp/Jp;
Blin[1][10]=2*N*B1*x*L1*cosp/Jp;
Blin[1][11]=2*N*B1*(b-x)*L1*cosp/Jp;
Blin[1][12]=2*N*B2*(h/2.0+x)*L2*cosp/Jp;
Blin[1][13]=N*(2*B2*(h/2.0-d-x)*L2+2*B3*x*L3)*cosp/Jp;
Blin[1][14]=2*N*B3*(b-x)*L3*cosp/Jp;
Blin[1][15]=2*N*B4*(h/2.0+x)*L4*cosp/Jp;
Blin[1][16]=N*(2*B4*(h/2.0-d-x)*L4+2*B5*x*L5)*cosp/Jp;
Blin[1][17]=2*N*B5*(b-x)*L5*cosp/Jp;
Blin[1][18]=2*N*B6*(h/2.0+x)*L6*cosp/Jp;
Blin[1][19]=2*N*B6*(h/2.0-d-x)*L6*cosp/Jp;

// roll
Blin[2][0]=2*B1*x*N*D1*cosr/Jr;
Blin[2][1]=2*B1*(b-x)*N*D1*cosr/Jr;
Blin[2][2]=2*B2*(h/2.0+x)*N*D1*cosr/Jr;
Blin[2][3]=(2*B2*(h/2.0-d-x)+2*B3*x)*N*D1*cosr/Jr;
Blin[2][4]=2*B3*(b-x)*N*D1*cosr/Jr;
Blin[2][5]=2*B4*(h/2.0+x)*N*D1*cosr/Jr;
Blin[2][6]=(2*B4*(h/2.0-d-x)+2*B5*x)*N*D1*cosr/Jr;
Blin[2][7]=2*B5*(b-x)*N*D1*cosr/Jr;
Blin[2][8]=2*B6*(h/2.0+x)*N*D1*cosr/Jr;
Blin[2][9]=2*B6*(h/2.0-d-x)*N*D1*cosr/Jr;
Blin[2][10]=2*B1*x*N*D2*cosr/Jr;
Blin[2][11]=2*B1*(b-x)*N*D2*cosr/Jr;
Blin[2][12]=2*B2*(h/2.0+x)*N*D2*cosr/Jr;
```

```
    Blin[2][13]=(2*B2*(h/2.0-d-x)+2*B3*x)*N*D2*cosr/Jr;
    Blin[2][14]=2*B3*(b-x)*N*D2*cosr/Jr;
    Blin[2][15]=2*B4*(h/2.0+x)*N*D2*cosr/Jr;
    Blin[2][16]=(2*B4*(h/2.0-d-x)+2*B5*x)*N*D2*cosr/Jr;
    Blin[2][17]=2*B5*(b-x)*N*D2*cosr/Jr;
    Blin[2][18]=2*B6*(h/2.0+x)*N*D2*cosr/Jr;
    Blin[2][19]=2*B6*(h/2.0-d-x)*N*D2*cosr/Jr;
}

if (status==2){
    // levitation
    Blin[0][0]=2*B1*x*N/M;
    Blin[0][1]=2*B1*(b-x)*N/M;
    Blin[0][2]=2*B2*b*N/M;
    Blin[0][3]=2*B3*x*N/M;
    Blin[0][4]=2*B3*(b-x)*N/M;
    Blin[0][5]=2*B4*b*N/M;
    Blin[0][6]=2*B5*x*N/M;
    Blin[0][7]=2*B5*(b-x)*N/M;
    Blin[0][8]=2*B6*b*N/M;
    Blin[0][9]=0.0;
    Blin[0][10]=2*B1*x*N/M;
    Blin[0][11]=2*B1*(b-x)*N/M;
    Blin[0][12]=2*B2*b*N/M;
    Blin[0][13]=2*B3*x*N/M;
    Blin[0][14]=2*B3*(b-x)*N/M;
    Blin[0][15]=2*B4*b*N/M;
    Blin[0][16]=2*B5*x*N/M;
    Blin[0][17]=2*B5*(b-x)*N/M;
    Blin[0][18]=2*B6*b*N/M;
    Blin[0][19]=0.0;

    //pitch
    Blin[1][0]=2*N*B1*x*L1*cosp/Jp;
    Blin[1][1]=2*N*B1*(b-x)*L1*cosp/Jp;
    Blin[1][2]=2*N*B2*b*L2*cosp/Jp;
    Blin[1][3]=2*N*B3*x*L3*cosp/Jp;
    Blin[1][4]=2*N*B3*(b-x)*L3*cosp/Jp;
    Blin[1][5]=2*N*B4*b*L4*cosp/Jp;
    Blin[1][6]=2*N*B5*x*L5*cosp/Jp;
    Blin[1][7]=2*N*B5*(b-x)*L5*cosp/Jp;
    Blin[1][8]=2*N*B6*b*L6*cosp/Jp;
    Blin[1][9]=0.0;
    Blin[1][10]=2*N*B1*x*L1*cosp/Jp;
    Blin[1][11]=2*N*B1*(b-x)*L1*cosp/Jp;
    Blin[1][12]=2*N*B2*b*L2*cosp/Jp;
    Blin[1][13]=2*N*B3*x*L3*cosp/Jp;
    Blin[1][14]=2*N*B3*(b-x)*L3*cosp/Jp;
    Blin[1][15]=2*N*B4*b*L4*cosp/Jp;
    Blin[1][16]=2*N*B5*x*L5*cosp/Jp;
    Blin[1][17]=2*N*B5*(b-x)*L5*cosp/Jp;
    Blin[1][18]=2*N*B6*b*L6*cosp/Jp;
    Blin[1][19]=0.0;
```

```
    // roll
    Blin[2][0]=2*B1*x*N*D1*cosr/Jr;
    Blin[2][1]=2*B1*(b-x)*N*D1*cosr/Jr;
    Blin[2][2]=2*B2*b*N*D1*cosr/Jr;
    Blin[2][3]=2*B3*x*N*D1*cosr/Jr;
    Blin[2][4]=2*B3*(b-x)*N*D1*cosr/Jr;
    Blin[2][5]=2*B4*b*N*D1*cosr/Jr;
    Blin[2][6]=2*B5*x*N*D1*cosr/Jr;
    Blin[2][7]=2*B5*(b-x)*N*D1*cosr/Jr;
    Blin[2][8]=2*B6*b*N*D1*cosr/Jr;
    Blin[2][9]=0.0;
    Blin[2][10]=2*B1*x*N*D2*cosr/Jr;
    Blin[2][11]=2*B1*(b-x)*N*D2*cosr/Jr;
    Blin[2][12]=2*B2*b*N*D2*cosr/Jr;
    Blin[2][13]=2*B3*x*N*D2*cosr/Jr;
    Blin[2][14]=2*B3*(b-x)*N*D2*cosr/Jr;
    Blin[2][15]=2*B4*b*N*D2*cosr/Jr;
    Blin[2][16]=2*B5*x*N*D2*cosr/Jr;
    Blin[2][17]=2*B5*(b-x)*N*D2*cosr/Jr;
    Blin[2][18]=2*B6*b*N*D2*cosr/Jr;
    Blin[2][19]=0.0;
}

if (status==3){

    //levitation
    Blin[0][0]=2*B1*x*N/M;
    Blin[0][1]=(2*B1*(b-x)+2*B2*(x-h/2.0))*N/M;
    Blin[0][2]=2*B2*(b-x+h/2.0)*N/M;
    Blin[0][3]=2*B3*x*N/M;
    Blin[0][4]=(2*B3*(b-x)+2*B4*(x-h/2.0))*N/M;
    Blin[0][5]=2*B4*(b-x+h/2.0)*N/M;
    Blin[0][6]=2*B5*x*N/M;
    Blin[0][7]=(2*B5*(b-x)+2*B6*(x-h/2.0))*N/M;
    Blin[0][8]=2*B6*(b-x+h/2.0)*N/M;
    Blin[0][9]=0.0;
    Blin[0][10]=2*B1*x*N/M;
    Blin[0][11]=(2*B1*(b-x)+2*B2*(x-h/2.0))*N/M;
    Blin[0][12]=2*B2*(b-x+h/2.0)*N/M;
    Blin[0][13]=2*B3*x*N/M;
    Blin[0][14]=(2*B3*(b-x)+2*B4*(x-h/2.0))*N/M;
    Blin[0][15]=2*B4*(b-x+h/2.0)*N/M;
    Blin[0][16]=2*B5*x*N/M;
    Blin[0][17]=(2*B5*(b-x)+2*B6*(x-h/2.0))*N/M;
    Blin[0][18]=2*B6*(b-x+h/2.0)*N/M;
    Blin[0][19]=0.0;

    //pitch
    Blin[1][0]=2*N*B1*x*L1*cosp/Jp;
    Blin[1][1]=N*(2*B1*(b-x)*L1+2*B2*(x-h/2.0)*L2)*cosp/Jp;
    Blin[1][2]=2*N*B2*(b-x+h/2.0)*L2*cosp/Jp;
    Blin[1][3]=2*N*B3*x*L3*cosp/Jp;
    Blin[1][4]=N*(2*B3*(b-x)*L3+2*B4*(x-h/2.0)*L4)*cosp/Jp;
    Blin[1][5]=2*N*B4*(b-x+h/2.0)*L4*cosp/Jp;
```

```
    Blin[1][6]=2*N*B5*x*L5*cosp/Jp;
    Blin[1][7]=N*(2*B5*(b-x)*L5+2*B6*(x-h/2.0)*L6)*cosp/Jp;
    Blin[1][8]=2*N*B6*(b-x+h/2.0)*L6*cosp/Jp;
    Blin[1][9]=0.0;
    Blin[1][10]=2*N*B1*x*L1*cosp/Jp;
    Blin[1][11]=N*(2*B1*(b-x)*L1+2*B2*(x-h/2.0)*L2)*cosp/Jp;
    Blin[1][12]=2*N*B2*(b-x+h/2.0)*L2*cosp/Jp;
    Blin[1][13]=2*N*B3*x*L3*cosp/Jp;
    Blin[1][14]=N*(2*B3*(b-x)*L3+2*B4*(x-h/2.0)*L4)*cosp/Jp;
    Blin[1][15]=2*N*B4*(b-x+h/2.0)*L4*cosp/Jp;
    Blin[1][16]=2*N*B5*x*L5*cosp/Jp;
    Blin[1][17]=N*(2*B5*(b-x)*L5+2*B6*(x-h/2.0)*L6)*cosp/Jp;
    Blin[1][18]=2*N*B6*(b-x+h/2.0)*L6*cosp/Jp;
    Blin[1][19]=0.0;

    //roll
    Blin[2][0]=2*B1*x*N*D1*cosr/Jr;
    Blin[2][1]=(2*B1*(b-x)+2*B2*(x-h/2.0))*N*D1*cosr/Jr;
    Blin[2][2]=2*B2*(b-x+h/2.0)*N*D1*cosr/Jr;
    Blin[2][3]=2*B3*x*N*D1*cosr/Jr;
    Blin[2][4]=(2*B3*(b-x)+2*B4*(x-h/2.0))*N*D1*cosr/Jr;
    Blin[2][5]=2*B4*(b-x+h/2.0)*N*D1*cosr/Jr;
    Blin[2][6]=2*B5*x*N*D1*cosr/Jr;
    Blin[2][7]=(2*B5*(b-x)+2*B6*(x-h/2.0))*N*D1*cosr/Jr;
    Blin[2][8]=2*B6*(b-x+h/2.0)*N*D1*cosr/Jr;
    Blin[2][9]=0.0;
    Blin[2][10]=2*B1*x*N*D2*cosr/Jr;
    Blin[2][11]=(2*B1*(b-x)+2*B2*(x-h/2.0))*N*D2*cosr/Jr;
    Blin[2][12]=2*B2*(b-x+h/2.0)*N*D2*cosr/Jr;
    Blin[2][13]=2*B3*x*N*D2*cosr/Jr;
    Blin[2][14]=(2*B3*(b-x)+2*B4*(x-h/2.0))*N*D2*cosr/Jr;
    Blin[2][15]=2*B4*(b-x+h/2.0)*N*D2*cosr/Jr;
    Blin[2][16]=2*B5*x*N*D2*cosr/Jr;
    Blin[2][17]=(2*B5*(b-x)+2*B6*(x-h/2.0))*N*D2*cosr/Jr;
    Blin[2][18]=2*B6*(b-x+h/2.0)*N*D2*cosr/Jr;
    Blin[2][19]=0.0;
}

if (status==4){
    //levitation
    Blin[0][0]=2*B1*b*N/M;
    Blin[0][1]=2*B2*(x-h/2.0)*N/M;
    Blin[0][2]=2*B2*(b-x-h/2.0)*N/M;
    Blin[0][3]=2*B3*b*N/M;
    Blin[0][4]=2*B4*(x-h/2.0)*N/M;
    Blin[0][5]=2*B4*(b-x-h/2.0)*N/M;
    Blin[0][6]=2*B5*b*N/M                                    ;
    Blin[0][7]=2*B6*(x-h/2.0)*N/M;
    Blin[0][8]=2*B6*(b-x-h/2.0)*N/M;
    Blin[0][9]=0.0;
    Blin[0][10]=2*B1*b*N/M;
    Blin[0][11]=2*B2*(x-h/2.0)*N/M;
    Blin[0][12]=2*B2*(b-x-h/2.0)*N/M;
    Blin[0][13]=2*B3*b*N/M;
```

```
    Blin[0][14]=2*B4*(x-h/2.0)*N/M;
    Blin[0][15]=2*B4*(b-x-h/2.0)*N/M;
    Blin[0][16]=2*B5*b*N/M;
    Blin[0][17]=2*B6*(x-h/2.0)*N/M;
    Blin[0][18]=2*B6*(b-x-h/2.0)*N/M;
    Blin[0][19]=0.0;

    //pitch
    Blin[1][0]=2*N*B1*b*L1*cosp/Jp;
    Blin[1][1]=2*N*B2*(x-h/2.0)*L2*cosp/Jp;
    Blin[1][2]=2*N*B2*(b-x-h/2.0)*L2*cosp/Jp;
    Blin[1][3]=2*N*B3*b*L3*cosp/Jp;
    Blin[1][4]=2*N*B4*(x-h/2.0)*L4*cosp/Jp;
    Blin[1][5]=2*N*B4*(b-x-h/2.0)*L4*cosp/Jp;
    Blin[1][6]=2*N*B5*b*L5*cosp/Jp;
    Blin[1][7]=2*N*B6*(x-h/2.0)*L6*cosp/Jp;
    Blin[1][8]=2*N*B6*(b-x-h/2.0)*L6*cosp/Jp;
    Blin[1][9]=0.0;
    Blin[1][10]=2*N*B1*b*L1*cosp/Jp;
    Blin[1][11]=2*N*B2*(x-h/2.0)*L2*cosp/Jp;
    Blin[1][12]=2*N*B2*(b-x-h/2.0)*L2*cosp/Jp;
    Blin[1][13]=2*N*B3*b*L3*cosp/Jp;
    Blin[1][14]=2*N*B4*(x-h/2.0)*L4*cosp/Jp;
    Blin[1][15]=2*N*B4*(b-x-h/2.0)*L4*cosp/Jp;
    Blin[1][16]=2*N*B5*b*L5*cosp/Jp;
    Blin[1][17]=2*N*B6*(x-h/2.0)*L6*cosp/Jp;
    Blin[1][18]=2*N*B6*(b-x-h/2.0)*L6*cosp/Jp;
    Blin[1][19]=0.0;

    //roll
    Blin[2][0]=2*B1*b*N*D1*cosr/Jr;
    Blin[2][1]=2*B2*(x-h/2.0)*N*D1*cosr/Jr;
    Blin[2][2]=2*B2*(b-x-h/2.0)*N*D1*cosr/Jr;
    Blin[2][3]=2*B3*b*N*D1*cosr/Jr;
    Blin[2][4]=2*B4*(x-h/2.0)*N*D1*cosr/Jr;
    Blin[2][5]=2*B4*(b-x-h/2.0)*N*D1*cosr/Jr;
    Blin[2][6]=2*B5*b*N*D1*cosr/Jr;
    Blin[2][7]=2*B6*(x-h/2.0)*N*D1*cosr/Jr;
    Blin[2][8]=2*B6*(b-x-h/2.0)*N*D1*cosr/Jr;
    Blin[2][9]=0.0;
    Blin[2][10]=2*B1*b*N*D2*cosr/Jr;
    Blin[2][11]=2*B2*(x-h/2.0)*N*D2*cosr/Jr;
    Blin[2][12]=2*B2*(b-x-h/2.0)*N*D2*cosr/Jr;
    Blin[2][13]=2*B3*b*N*D2*cosr/Jr;
    Blin[2][14]=2*B4*(x-h/2.0)*N*D2*cosr/Jr;
    Blin[2][15]=2*B4*(b-x-h/2.0)*N*D2*cosr/Jr;
    Blin[2][16]=2*B5*b*N*D2*cosr/Jr;
    Blin[2][17]=2*B6*(x-h/2.0)*N*D2*cosr/Jr;
    Blin[2][18]=2*B6*(b-x-h/2.0)*N*D2*cosr/Jr;
    Blin[2][19]=0.0;
}

sum0=0.0;
sum1=0.0;
```

```
sum2=0.0;

// calculate norm of each column
for (i=0;i<20;i++){
    sum0=sum0+Blin[0][i]*Blin[0][i];
    sum1=sum1+Blin[1][i]*Blin[1][i];
    sum2=sum2+Blin[2][i]*Blin[2][i];
}

// calculate T matrix to transform 3 inputs to 20
for (i=0;i<3;i++){
    for (j=0;j<20;j++){
        if (i==0)   T[j][i]=1/sum0*Blin[i][j];
        if (i==1)   T[j][i]=1/sum1*Blin[i][j];
        if (i==2)   T[j][i]=1/sum2*Blin[i][j];
    }
}

// calculate B matrix for 3 inputs
for (i=0;i<3;i++){
    for (j=0;j<3;j++){
        sum3=0.0;
        for(k=0;k<20;k++){
            sum3=sum3+Blin[i][k]*T[k][j];
        }
        A[i][j]=sum3;
    }
}

// calculate inverse of B matrix for controller
minv(A);

// calculate controller outputs
for (i=0;i<3;i++){
    e1[i]   =   *uPtrs[i+1];
    e2[i]   =   prwrk[i];
    e3[i]   =   xc[4+i];
    L[i]    =   *uPtrs[i+4];
    labda[i]=   *uPtrs[i+7];
    s[i]    =   L[i]*L[i]*e3[i]+2*L[i]*e1[i]+e2[i];
    vs[i]   =   L[i]*s[i];
}

for (i=0;i<3;i++){
    sum4    =   0.0;
    sum5    =   0.0;
    for(j=0;j<3;j++){
        sum4=sum4+A[i][j]*vs[j];
        sum5=sum5+A[i][j]*(*uPtrs[12+j]);
    }
    v[i]=sum5-sum4-labda[i]*2.0/pi*atan(phi*s[i]);
}

// transform 3 to 20 controller outputs
```

```
    for (i=0;i<20;i++){
        sum6=0.0;
        for (j=0;j<3;j++){
            sum6=sum6+T[i][j]*v[j];
        }
        yPtrs[i]=sum6;
        //printf("output value = %d \n\n",sum6);
    }
}

//=============================================================================
// mdlDerivatives
//=============================================================================

#define MDL_DERIVATIVES

static void mdlDerivatives(SimStruct *S)
{
    real_T  *xc =   ssGetContStates(S);
    real_T  *dx =   ssGetdX(S);

    real_T  *prwrk  =   ssGetRWork(S);

    InputRealPtrsType   uPtrs   =   ssGetInputPortRealSignalPtrs(S,0);

    if (5.08e-2-xc[0]<0)    xc[0]=0.0;

    dx[0]=*uPtrs[0];

    dx[1]=Kgain*(*uPtrs[1]-xc[1]);
    dx[2]=Kgain*(*uPtrs[2]-xc[2]);
    dx[3]=Kgain*(*uPtrs[3]-xc[3]);

    prwrk[0]=dx[1];
    prwrk[1]=dx[2];
    prwrk[2]=dx[3];

    dx[4]=*uPtrs[1];
    dx[5]=*uPtrs[2];
    dx[6]=*uPtrs[3];
}

//=============================================================================
// mdlTerminate()
//=============================================================================

static void mdlTerminate(SimStruct *S)
{
}

#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
```

```
#endif
```

D.1.1 C-code for Matrix Inversion

```
//calculate inverse of 3x3 matrix
//Algorithm: Gauss-Jordan Elimination

void minv(real_T mat[][3])
{
    int_T   i,j,k,ki,irow,m,n;
    real_T  max,temp,factor;
    real_T  Bmat[3][6];
    real_T  tol =   1e-8;

    //assemble matrix for G-J elimination
    m=3;
    n=2*m;
    for (i=0;i<m;i++){
        for(j=0;j<m;j++){
            if (i==j)   {Bmat[i][j+m]=1.0;}
            else        {Bmat[i][j+m]=0.0;}
            Bmat[i][j]=mat[i][j];
        }
    }
    i=0;
    j=0;

    //main loop
    while ( (i<m) & (j<n) ){
        // check which is the largest pivot
        max=0.0;
        for (k=i;k<m;k++){
            if (fabs(Bmat[k][j])>max){
                max=fabs(Bmat[k][j]);
                irow=k;
            }
        }
       //printf("The max value: %g \n",max);

        // zero out column if max is neglegible
        if (max<tol){
            for (k=i;k<m;k++){
                Bmat[k][j]=0.0;
            }
            j=j+1;
        }
        // if not, perform elimination
        else {
            // swap i-th and irow-th row
            for (k=j;k<n;k++){
                temp=Bmat[i][k];Bmat[i][k]=Bmat[irow][k];Bmat[irow][k]=temp;
            }
```

```
            // divide the pivot row by the pivot element
            factor=Bmat[i][j];
            for (k=j;k<n;k++){
                Bmat[i][k]=Bmat[i][k]/factor;
            }

            // substract multiples of the pivot row from all other rows.
            if (i>0){
                for (ki=0;ki<i;ki++){
                    factor=Bmat[ki][j];
                    for (k=j;k<n;k++){
                        Bmat[ki][k]=Bmat[ki][k]-factor*Bmat[i][k];
                    }
                }
            }
            for (ki=i+1;ki<m;ki++){
                factor=Bmat[ki][j];
                for (k=j;k<n;k++){
                    Bmat[ki][k]=Bmat[ki][k]-factor*Bmat[i][k];
                }
            }

            i=i+1;
            j=j+1;
        }
    }
    for (i=0;i<m;i++){
        for (j=0;j<m;j++){
            mat[i][j]=Bmat[i][j+m];
        }
    }
}
```

D.2 C-code 3-DOF Model for Feedback Control

```
//=============================================================================
// setup
//=============================================================================

#define S_FUNCTION_NAME maglev_3dssd
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
#include "math.h"

// parameters
#define coil        1           // coil of which the current is an output
#define g           9.8         // gravity (acceleration) (m/s^2)
#define ground      -0.0125     // floor level of wheel-track (minimum levitation)
#define ceil        0.01        // ceil level of wheel-track (maximum levitation)

// coil parameters
#define R           0.638       // coil resistance (Ohm)
```

```
#define L           0.837e-3    // coil inductance (H)
#define N           100         // number of coil windings (-)
#define b           4.48e-2     // coil lenght (m)
#define d           0.6e-2      // gap between coils (m); (h=b+d)

// magnet parameters
#define h           5.08e-2     // height en length of magnet (m)
#define B           1.25        // magnetic field strenght (T)

//vehicle parameters
#define M           6.0         // mass of the vehicle (kg)
#define Jp          0.1437      // pitch rotational inertia (kg*m^2)
#define dTp         0.01        // torque disturbace on pitch (N*m)
#define Jr          0.0288      // roll rotational inertia (kg*m^2)
#define dTr         0.01        // torque disturbance on roll (N*m)
#define LL          0.4572      // length of vehicle (m)
#define L1          0.1905      // distance of magnet one to centre of inertia (m)
#define L2          0.1143      // distance of magnet two to c.o.i (m)
#define L3          0.0381      // distance of magnet three to c.o.i. (m)
#define L4          -0.0381     // distance of magnet four to c.o.i. (m)
#define L5          -0.1143     // distance of magnet five to c.o.i (m)
#define L6          -0.1905     // distance of magnet six to c.o.i (m)
#define DD          0.2286      // width of vehicle (m)
#define D1          0.0368      // distance from magnet row 1 to c.o.i (m)
#define D2          -0.0368     // distance from magnet row 2 to c.o.i (m)

// aerodynamic drag forces
#define rho_air     1.0         // density of air at 293 K (kg/m^3)
#define A_front     0.0129      // area of front of the sled (m^2)
#define C_front     2           // drag coefficient of flat plate (-)
#define Fd_arm      0.025       // arm of drag force on pitch (m)

#define NINPUTS     21          //number of inputs      uPtrs: input vector
// 0. velocity
// 1-20 currents

#define NOUTPUTS    3           //number of outputs     yPtrs: output vector
// 0. delta
// 1. pitch
// 2. roll

#define NCSTATES    7            //number of continuous states     xc: state vector
// 00. magnet related displacement of the vehicle   (x)
// 01. vertical displacement of centre of inertia   (z)
// 02. vertical velocity of centre of inertia       (z_dot)
// 03. pitch                                        (phi)
// 04. pitch velocity                               (phi_dot)
// 05. roll                                         (theta)
// 06. roll velocity                                (theta_dot)

#define NIWRK       2           // number of integer global variables
#define NRWRK       6           // number of real global variables
```

```
//=============================================================================
// mdlInitializeSizes
//=============================================================================

static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S,0);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {return;}

    ssSetNumContStates(S,NCSTATES);
    ssSetNumDiscStates(S,0);

    if (!ssSetNumInputPorts(S,1)) {return;}
    ssSetInputPortWidth(S,0,NINPUTS);
    ssSetInputPortDirectFeedThrough(S,0,1);

    if (!ssSetNumOutputPorts(S,1)) {return;}
    ssSetOutputPortWidth(S,0,NOUTPUTS);

    ssSetNumSampleTimes(S,1);
    ssSetNumRWork(S,NRWRK);
    ssSetNumIWork(S,NIWRK);
    ssSetNumPWork(S,0);
    ssSetNumModes(S,0);
    ssSetNumNonsampledZCs(S,0);
}


//=============================================================================
// mdlInitializeSampleTimes
//=============================================================================

static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S,0,CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S,0,0.0);
}


//=============================================================================
// mdlInitializeConditions
//=============================================================================

#define MDL_INITIALIZE_CONDITIONS

static void mdlInitializeConditions(SimStruct *S)
{
    real_T  *xc =   ssGetContStates(S);
    real_T  *dx =   ssGetdX(S);

    real_T  *prwrk  =   ssGetRWork(S);
    int_T   *piwrk  =   ssGetIWork(S);
    int_T   i;
```

```
    for(i=0;i<NCSTATES;i++){
        xc[i]=0.0;
    }
    xc[1]=ground;
    for(i=0;i<NIWRK;i++){piwrk[i]=0;}
    for(i=0;i<NRWRK;i++){prwrk[i]=0.0;}
}

//=============================================================================
// mdlOutputs
//=============================================================================

static void mdlOutputs(SimStruct *S, int_T tid)
{
    real_T  *xc     =   ssGetContStates(S);
    real_T  *yPtrs  =   ssGetOutputPortRealSignal(S,0);

    InputRealPtrsType   uPtrs   =   ssGetInputPortRealSignalPtrs(S,0);

    int_T   *piwrk  =   ssGetIWork(S);
    real_T  *prwrk  =   ssGetRWork(S);

    // output
    yPtrs[0]=xc[1];
    yPtrs[1]=xc[3];
    yPtrs[2]=xc[5];
}


//=============================================================================
// mdlDerivatives
//=============================================================================

#define MDL_DERIVATIVES

static void mdlDerivatives(SimStruct *S)
{
    real_T  *xc =   ssGetContStates(S);
    real_T  *dx =   ssGetdX(S);

    real_T  Tp,Tr,F,Fd,Td;
    real_T  F_1,F1_1,F2_1,F3_1,F4_1,F5_1,F6_1;
    real_T  F_2,F1_2,F2_2,F3_2,F4_2,F5_2,F6_2;
    real_T  z1,z2,z3,z4;
    int_T   z1_ground,z2_ground,z3_ground,z4_ground;
    int_T   z1_ceil,z2_ceil,z3_ceil,z4_ceil;

    real_T  B1          =   B;
    real_T  B2          =   -B;
    real_T  B3          =   B;
    real_T  B4          =   -B;
    real_T  B5          =   B;
    real_T  B6          =   -B;
```

```
int_T   status_half,status_gap_half,status_gap;
int_T   not_half        =   0;  // (xc[0]<h/2)
int_T   half            =   1;  // (xc[0]>h/2)
int_T   not_gap_half    =   0;  // (xc[0]<h/2-d)
int_T   gap_half        =   1;  // (xc[0]>h/2-d)
int_T   not_gap         =   0;  // (xc[0]<b)
int_T   gap             =   1;  // (xc[0]>b)

InputRealPtrsType   uPtrs   =   ssGetInputPortRealSignalPtrs(S,0);

int_T   *piwrk  =   ssGetIWork(S);
real_T  *prwrk  =   ssGetRWork(S);

// update coils when the first magnet has passed one coil
if( h-xc[0] < 0.0 ){
    xc[0]=0.0;
}

// check in which status (position )the magnet is in
if      ( h/2.0-xc[0] < 0.0)    {status_half=half;}
else                            {status_half=not_half;}

if      ( h/2.0-d-xc[0] < 0.0)  {status_gap_half=gap_half;}
else                            {status_gap_half=not_gap_half;}

if      ( b-xc[0] < 0.0)        {status_gap=gap;}
else                            {status_gap=not_gap;}

// calculate voltages and forces for all magnets depending on the status.

// 0 <= xc[0] <= h/2-d
if      (status_half == not_half & status_gap_half == not_gap_half){
    F1_1 = (*uPtrs[1])*2*B1*xc[0]+(*uPtrs[2])*2*B1*(b-xc[0]);
    F2_1 = (*uPtrs[3])*2*B2*(h/2.0+xc[0])+(*uPtrs[4])*2*B2*(h/2.0-d-xc[0]);
    F3_1 = (*uPtrs[4])*2*B3*xc[0]+(*uPtrs[5])*2*B3*(b-xc[0]);
    F4_1 = (*uPtrs[6])*2*B4*(h/2.0+xc[0])+(*uPtrs[7])*2*B4*(h/2.0-d-xc[0]);
    F5_1 = (*uPtrs[7])*2*B5*xc[0]+(*uPtrs[8])*2*B5*(b-xc[0]);
    F6_1 = (*uPtrs[9])*2*B6*(h/2.0+xc[0])+(*uPtrs[10])*2*B6*(h/2.0-d-xc[0]);
    //
    F1_2 = (*uPtrs[11])*2*B1*xc[0]+(*uPtrs[12])*2*B1*(b-xc[0]);
    F2_2 = (*uPtrs[13])*2*B2*(h/2.0+xc[0])+(*uPtrs[14])*2*B2*(h/2.0-d-xc[0]);
    F3_2 = (*uPtrs[14])*2*B3*xc[0]+(*uPtrs[15])*2*B3*(b-xc[0]);
    F4_2 = (*uPtrs[16])*2*B4*(h/2.0+xc[0])+(*uPtrs[17])*2*B4*(h/2.0-d-xc[0]);
    F5_2 = (*uPtrs[17])*2*B5*xc[0]+(*uPtrs[18])*2*B5*(b-xc[0]);
    F6_2 = (*uPtrs[19])*2*B6*(h/2.0+xc[0])+(*uPtrs[20])*2*B6*(h/2.0-d-xc[0]);
}

// h/2-d <= xc[0] <= h/2
else if (status_half == not_half & status_gap_half == gap_half){
    F1_1 = (*uPtrs[1])*2*B1*xc[0]+(*uPtrs[2])*2*B1*(b-xc[0]);
    F2_1 = (*uPtrs[3])*2*B2*b;
    F3_1 = (*uPtrs[4])*2*B3*xc[0]+(*uPtrs[5])*2*B3*(b-xc[0]);
    F4_1 = (*uPtrs[6])*2*B4*b;
    F5_1 = (*uPtrs[7])*2*B5*xc[0]+(*uPtrs[8])*2*B5*(b-xc[0]);
```

```
    F6_1 = (*uPtrs[9])*2*B6*b;
    //
    F1_2 = (*uPtrs[11])*2*B1*xc[0]+(*uPtrs[12])*2*B1*(b-xc[0]);
    F2_2 = (*uPtrs[13])*2*B2*b;
    F3_2 = (*uPtrs[14])*2*B3*xc[0]+(*uPtrs[15])*2*B3*(b-xc[0]);
    F4_2 = (*uPtrs[16])*2*B4*b;
    F5_2 = (*uPtrs[17])*2*B5*xc[0]+(*uPtrs[18])*2*B5*(b-xc[0]);
    F6_2 = (*uPtrs[19])*2*B6*b;
}

// h/2 <= xc[0] <= b
else if (status_half == half & status_gap == not_gap){
    F1_1 = (*uPtrs[1])*2*B1*xc[0]+(*uPtrs[2])*2*B1*(b-xc[0]);
    F2_1 = (*uPtrs[2])*2*B2*(xc[0]-h/2.0)+(*uPtrs[3])*2*B2*(b-xc[0]+h/2.0);
    F3_1 = (*uPtrs[4])*2*B3*xc[0]+(*uPtrs[5])*2*B3*(b-xc[0]);
    F4_1 = (*uPtrs[5])*2*B4*(xc[0]-h/2.0)+(*uPtrs[6])*2*B4*(b-xc[0]+h/2.0);
    F5_1 = (*uPtrs[7])*2*B5*xc[0]+(*uPtrs[8])*2*B5*(b-xc[0]);
    F6_1 = (*uPtrs[8])*2*B6*(xc[0]-h/2.0)+(*uPtrs[9])*2*B6*(b-xc[0]+h/2.0);
    //
    F1_2 = (*uPtrs[11])*2*B1*xc[0]+(*uPtrs[12])*2*B1*(b-xc[0]);
    F2_2 = (*uPtrs[12])*2*B2*(xc[0]-h/2.0)+(*uPtrs[13])*2*B2*(b-xc[0]+h/2.0);
    F3_2 = (*uPtrs[14])*2*B3*xc[0]+(*uPtrs[15])*2*B3*(b-xc[0]);
    F4_2 = (*uPtrs[15])*2*B4*(xc[0]-h/2.0)+(*uPtrs[16])*2*B4*(b-xc[0]+h/2.0);
    F5_2 = (*uPtrs[17])*2*B5*xc[0]+(*uPtrs[18])*2*B5*(b-xc[0]);
    F6_2 = (*uPtrs[18])*2*B6*(xc[0]-h/2.0)+(*uPtrs[19])*2*B6*(b-xc[0]+h/2.0);
}

// b <= xc[0] <= h
else if (status_half == half & status_gap == gap){
    F1_1 = (*uPtrs[1])*2*B1*b;
    F2_1 = (*uPtrs[2])*2*B2*(xc[0]-h/2.0)+(*uPtrs[3])*2*B2*(b-xc[0]+h/2.0);
    F3_1 = (*uPtrs[4])*2*B3*b;
    F4_1 = (*uPtrs[5])*2*B4*(xc[0]-h/2.0)+(*uPtrs[6])*2*B4*(b-xc[0]+h/2.0);
    F5_1 = (*uPtrs[7])*2*B5*b;
    F6_1 = (*uPtrs[8])*2*B6*(xc[0]-h/2.0)+(*uPtrs[9])*2*B6*(b-xc[0]+h/2.0);
    //
    F1_2 = (*uPtrs[11])*2*B1*b;
    F2_2 = (*uPtrs[12])*2*B2*(xc[0]-h/2.0)+(*uPtrs[13])*2*B2*(b-xc[0]+h/2.0);
    F3_2 = (*uPtrs[14])*2*B3*b;
    F4_2 = (*uPtrs[15])*2*B4*(xc[0]-h/2.0)+(*uPtrs[16])*2*B4*(b-xc[0]+h/2.0);
    F5_2 = (*uPtrs[17])*2*B5*b;
    F6_2 = (*uPtrs[18])*2*B6*(xc[0]-h/2.0)+(*uPtrs[19])*2*B6*(b-xc[0]+h/2.0);
}


// calculate forces and torques
F_1     =   N*(F1_1+F2_1+F3_1+F4_1+F5_1+F6_1);
F_2     =   N*(F1_2+F2_2+F3_2+F4_2+F5_2+F6_2);
F       =   F_1+F_2;
Tp      =   N*((F1_1+F1_2)*L1 + (F2_1+F2_2)*L2 + (F3_1+F3_2)*L3 + (F4_1+F4_2)*L4
            + (F5_1+F5_2)*L5 + (F6_1+F6_2)*L6)*cos(xc[3]);
Tr      =   (F_1*D1+F_2*D2)*cos(xc[5]);
Fd      =   0.5*rho_air*(*uPtrs[0])*(*uPtrs[0])*A_front*C_front;
Td      =   Fd*Fd_arm;
```

```
// differential equations
dx[0]   =   *uPtrs[0];
dx[1]   =   xc[2];
dx[2]   =   F/M-g;
dx[3]   =   xc[4];
dx[4]   =   (Tp+dTp+Td)/Jp;
dx[5]   =   xc[6];
dx[6]   =   (Tr+dTr)/Jr;

// check position of the corners (wheels) of the vehicle
z1      =   xc[1]+0.5*LL*sin(xc[3])+0.5*DD*sin(xc[5]);
z2      =   xc[1]-0.5*LL*sin(xc[3])+0.5*DD*sin(xc[5]);
z3      =   xc[1]-0.5*LL*sin(xc[3])-0.5*DD*sin(xc[5]);
z4      =   xc[1]+0.5*LL*sin(xc[3])-0.5*DD*sin(xc[5]);

z1_ground=0;    z2_ground=0;    z3_ground=0;    z4_ground=0;
z1_ceil=0;      z2_ceil=0;      z3_ceil=0;      z4_ceil=0;

if(z1<ground)   {z1=ground; z1_ground=1;}
if(z1>ceil)     {z1=ceil;   z1_ceil=1;}
if(z2<ground)   {z2=ground; z2_ground=1;}
if(z2>ceil)     {z2=ceil;   z2_ceil=1;}
if(z3<ground)   {z3=ground; z3_ground=1;}
if(z3>ceil)     {z3=ceil;   z3_ceil=1;}
if(z4<ground)   {z4=ground; z4_ground=1;}
if(z4>ceil)     {z4=ceil;   z4_ceil=1;}

// correct position if upper- or lower-bound is reached
if( z1_ground == 1 | z1_ceil == 1){
    xc[1]  =   z1 - 0.5*(z1-z2) - 0.5*(z1-z4);
    xc[2]  =   0.0;
    xc[3]  =   asin((z1-z2)/LL);
    xc[4]  =   0.0;
    xc[5]  =   asin((z1-z4)/DD);
    xc[6]  =   0.0;
}
if (z2_ground == 1 | z2_ceil == 1){
    xc[1]  =   z2 + 0.5*(z1-z2) - 0.5*(z2-z3);
    xc[2]  =   0.0;
    xc[3]  =   asin((z1-z2)/LL);
    xc[4]  =   0.0;
    xc[5]  =   asin((z2-z3)/DD);
    xc[6]  =   0.0;
}
if (z3_ground == 1 | z3_ceil == 1){
    xc[1]  =   z3 + 0.5*(z4-z3) + 0.5*(z2-z3);
    xc[2]  =   0.0;
    xc[3]  =   asin((z4-z3)/LL);
    xc[4]  =   0.0;
    xc[5]  =   asin((z2-z3)/DD);
    xc[6]  =   0.0;
}
if (z4_ground == 1 | z4_ceil == 1){
```

```
        xc[1]  =   z4 - 0.5*(z4-z3) + 0.5*(z1-z4);
        xc[2]  =   0.0;
        xc[3]  =   asin((z4-z3)/LL);
        xc[4]  =   0.0;
        xc[5]  =   asin((z1-z4)/DD);
        xc[6]  =   0.0;
    }

    // debug outputs
    prwrk[0]=z1;
    prwrk[1]=z2;
    prwrk[2]=z3;
    prwrk[3]=z4;
    prwrk[4]=0.0;
    prwrk[5]=0.0;
}


//=============================================================================
// mdlTerminate()
//=============================================================================

static void mdlTerminate(SimStruct *S)
{
}

#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Bibliography

[1] Dill, J. and Meeker, D. Maglifter tradeoff study and subscale system demonstrations. NAS-98069-1362, Foster-Miller, Inc, Waltham MA, December 2000.

[2] He, J.L., Rote, D.M. and Coffey, H.T. Applications of the dynamic circuit theory to maglev suspension systems. IEEE transactions on magnetics, vol. 29, no. 6, pp 4153-4164, November 1993.

[3] Kent, R.D. Designing with Null flux coils. IEEE transactions on magnetics, vol. 33, no. 5, pp 4327-4334, September 1997.

[4] Kent, R.D. Analysis of an electro dynamic maglev system. IEEE transactions on magnetics, vol. 35, no. 5, pp 4259-4267, September 1999.

[5] He, J.L., Rote, D.M. and Coffey, H.T. Analysis of the combined maglev levitation propulsion and guidance system. IEEE transactions on magnetics, vol. 31, no. 2, pp 981-987, March 1995.

[6] Jacobs, W.A. Magnetic launch assist-NASA's vision for the future. IEEE transactions on magnetics, vol. 37, no. 1, pp 55-57, January 2001.

[7] Ljung, L., System Identification, Theory for the user, Prentice Hall PTR Upper Saddle River, New Jersey, 1987. ISBN 0-13-881640-9

[8] Thompson, M.T., Thornton, R.D., Flux-Canceling Maglev Suspension, IEEE Transactions on Magnetics, Vol. 35, no. 3, p1956-1975, May 1999.

[9] He, J., Coffey, H., Magnetic Damping Forces in Figure-Eight-Shaped Null-Flux Coil Suspension Systems, IEEE Transactions on Magnetics, Vol. 33, no. 5, p4230-4232, September 1997

[10] Ohashi, S., Ohsaki, H., Masada, E., Effect of the Active Damper Coil System on the Lateral Displacement of the Magnetically Levitated Bogie, IEEE Transactions on Magnetics, Vol. 35, no. 5, p4001-4003, September 1999

[11] Brunelli, B., Casadei, D., Serra, G., Tani, A., Active Damping Control for Electrodynamic Suspension Systems without Mechanical Transducers, IEEE Transactions on Magnetics, Vol. 32, no. 5, p5055-5057, September 1996

[12] Ohashi, S., Effect of the Damper Coils on the Rotational Motion of the Superconducting Magnetically Levitated Bogie, IEEE Transactions on Magnetics, Vol. 36, no. 5, p3680-3682, September 2000