ISSN 0280-5316
ISRN LUTFD2/TFRT--5763--SE

Control of a Spider Crane

Thomas Johansson

Department of Automatic Control
Lund University
December 2005

Department of Automatic Control
Lund Institute of Technology
Box 118
SE-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
December 2005

Document Number
ISRNLUTFD2/TFRT--5763--SE

Author(s)
Thomas Johansson

Supervisor
Dominique Bonvin at EPFL, Lausanne in Schweiz Karl-Erik Årzén at Automatic Control in Lund

Sponsoring organization

Title and subtitle
Control of a Spider Crane. (Reglering av en spindelkran)

Abstract
The "Spider Crane" is a crane structure that has been developed at the department for automatic control at "Ecole Polytechnique F´ed´erale de Lausanne". This crane structure does not have a moving mechanical structure, which enables it to move the load much faster than a conventional crane. In a previous thesis project a control algorithm for the crane behavior has been developed. This control algorithm is based on flatness conversion of the reference trajectory of the load to reference signal for the crane motors to follow. This master thesis project proposes a new control algorithm that separates
the overall control of the crane behavior, from the control of the crane motors. Two different control approaches for the control of the motors are examined. One is based on tracking a ramp reference for the position of the motor. The second one, which has been implemented on the real system uses a cascade position controller for tracking the position reference and a feed forward controller to compensate for the disturbance the crane causes the motor. This control algorithm is tested on a reduced model of the real
system. The experimental results suggest that with this control approach it is possible to separate the overall control of the crane from the control of the crane motors.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
69

Recipient's notes

Security classification

Contents

Chapter 1

Introduction

At the department for automatic control at "Ecole Polytechnique Fédérale de Lausanne" a new crane structure called "Spider Crane" has been designed. The "Spider Crane" is constructed so as to achieve fast movement of the load with high precision. In a conventional crane, the mechanical structure needs to be moved to change the position of the load. This puts limitations on how fast the load can move. In the "Spider Crane" the mechanical structure is fixed. The positioning of the load is instead done by pulling three cables which are connected to the cable that suspends the load. This structure allows a much faster movement compared to a normal crane.

In a previous master thesis project a full dynamical model of the crane structure (apart from the DC motors which controls the cables) was developed. In that project, a flatness based control scheme for positioning the load (given a reference trajectory) was also developed. The controller was designed under the assumption that the DC motors could deliver a perfect torque. No particular care was taken in order to accurately model the DC motor.

In this project the control of the crane and the control of the DC motors will be separated. The main problem with the current motor is that its time constant is very fast. This is a problem since the motion planning for the crane load necessitate a good number of operations which imposes a slow sampling time. The time constant of the motor is to fast with respect to this sampling time.

The objective is to find a scheme which controls both the DC motors and the crane, using multirate sampling technique, i.e. a fast sampling time to handle the DC motor and a slow one for motion planning.

Chapter 2

Crane model

The spider crane mechanical structure consists of four fixed pylons Fig. 2.1. The three small pylons are all connected to the ring. The cable that is connected to the tall pylon runs through the ring and suspends the load. By adjusting the length of this cable the height of the load is controlled. And by adjusting the length of the cables connected to the ring the position of the load in the horizontal plane is controlled.

Every cable is connected to a DC motor, which controls the length of the cable. By using this crane structure, it is possible to move the load without moving the mechanical structure of the crane. A dynamical model, which describes the whole crane structure (apart from the DC motors) has been developed in a previous master thesis project[1]. The characteristics of this model are

- The input to the system are the forces in the cables
- The output of the system is the position of the load (x, y, z) and the height of the ring
- The system is flat

2.1 Simplified crane model

In order to answer the question: "Is it possible to do localized control of the DC motors, in order to control the crane?" it is not necessary to do it on the whole crane structure. It is sufficient as a first step to do it on a simplified model, with only one of the small crane pylons. Therefore a simplified model has been developed using Lagrangian mechanics [2].
The reason for using Lagrangian mechanics instead of Newton's mechanics is that the flatness calculation in the next section will be easier.

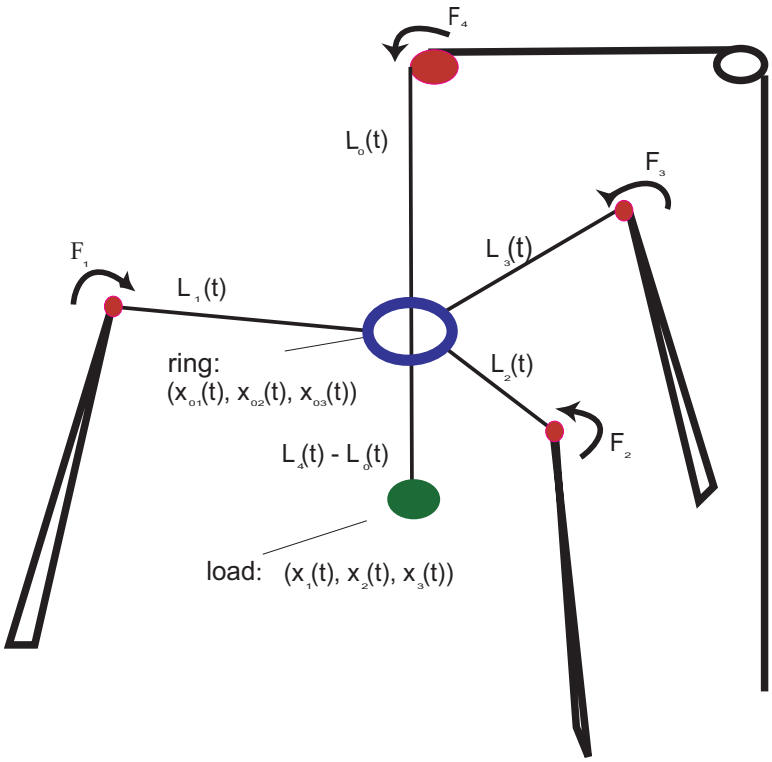

Figure 2.1: Model of the Spider Crane

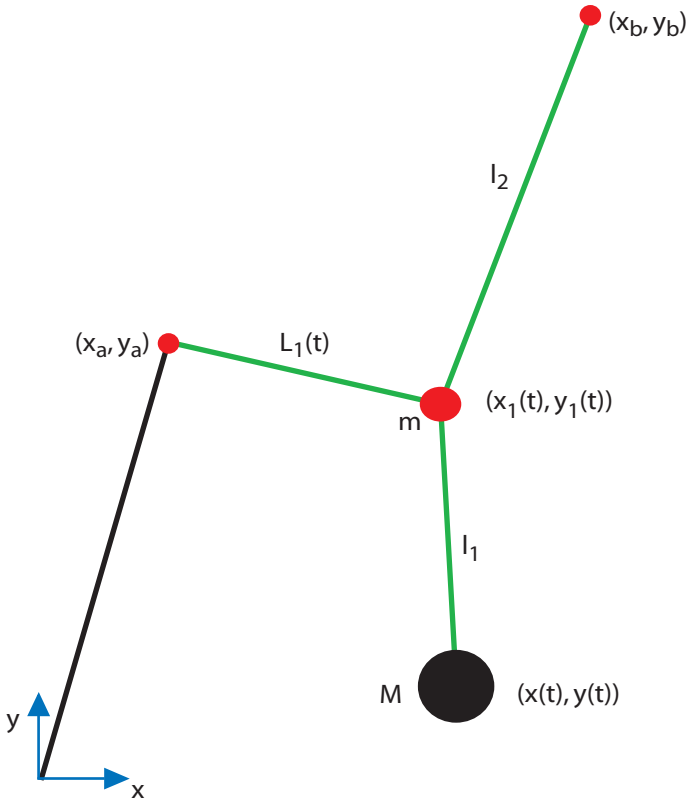

Figure 2.2: Model of the crane with only one pylon.

The following geometrical constraints for the cables are derived from Fig. 2.2.

$$(x_1 - x_B)^2 + (y_1 - y_B)^2 - l_2^2 = 0 \tag{2.1}$$

$$(x_1 - x_A)^2 + (y_1 - y_A)^2 - L_1^2 = 0 \tag{2.2}$$

$$(x_1 - x)^2 + (y_1 - y)^2 - l_1^2 = 0 \tag{2.3}$$

The total kinetic energy and potential energy of the system are

$$E_{kin} = \frac{M\dot{x}^2}{2} + \frac{M\dot{y}^2}{2} + \frac{m\dot{x_1}^2}{2} + \frac{m\dot{y_1}^2}{2} + \frac{m_L\dot{L_1}^2}{2} \tag{2.4}$$

$$E_{pot} = gMy + gmy_1 \tag{2.5}$$

The Lagrangian is given by

$$L = E_{kin} - E_{pot} \tag{2.6}$$

The only external force is T_c which is applied to the cable of length L_1 by the motor.

The Lagrangian equations of motion are given by

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q_i}}\right) - \frac{\partial L}{\partial q_i} = \sum_{j=1}^{3} \lambda_j \frac{\partial C_j}{\partial q_i} F_{ext} \tag{2.7}$$

and q_i is one of the components of q

$$q = (x, x_1, y, y_1, L_1) \tag{2.8}$$

Introducing (2.6),(2.1),(2.2) and (2.3) in (2.7) one get the equations of motions

$$\begin{aligned}
M\ddot{x} &= -2\lambda_3(x_1 - x) && (2.9)\\
m\ddot{x_1} &= 2\lambda_1(x_1 - x_B) + 2\lambda_2(x_1 - x_A) + 2\lambda_3(x_1 - x) && (2.10)\\
M\ddot{y} &= -gM - 2\lambda_3(y_1 - y) && (2.11)\\
m\ddot{y_1} &= -gm + 2\lambda_1(y_1 - y_B) + 2\lambda_2(y_1 - y_A) + 2\lambda_2(y_1 - y) && (2.12)\\
m_L\ddot{L_1} &= -2\lambda_2 L_1 + T_c && (2.13)
\end{aligned}$$

The movement of the load is considered to be small in y in the experimental setup, and the following assumptions are made

$$y \approx \bar{y} \text{ which is constant} \Rightarrow \dot{y} = 0, \ddot{y} = 0. \tag{2.14}$$

Note that this assumption only makes the calculation a bit easier, and it is possible to solve the problem without this simplification.

To get a system of differential equations that describes the behavior of the system, one has to solve the above equations for $\ddot{x}$ and $\ddot{x}_1$. By using 2.14 together with (2.11) one gets

$$\lambda_3 = -\frac{gM}{2l_1}. \tag{2.15}$$

Equations 2.15 and 2.9 are combined to get

$$\ddot{x} = \frac{g(x_1 - x)}{l_1} \tag{2.16}$$

Equations (2.10), (2.12) and (2.13) are combined and solved for $\ddot{x}_1, \lambda_1$ andλ_2. This yields the following system

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{x}_1 \\ \ddot{x}_1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{-g}{l_1} & 0 & \frac{-g}{l_1} & 0 \\ 0 & 0 & 0 & 1 \\ \text{solve} & (2.10) & (2.12) & \text{and } (2.13) \end{bmatrix} \cdot \begin{bmatrix} x \\ \dot{x} \\ x_1 \\ \dot{x}_1 \end{bmatrix} \tag{2.17}$$

2.2 Trajectory generation and flatness

The trajectory for the load is given as a position reference in cartesian coordinates (x, y, z). This trajectory needs to be converted to corresponding cable length and cable force, in order to have a useful reference for the control of the DC motors. Due to the fact that the system is flat[1] it is possible to do this.

Definition 1 *A system $\bar{\dot{x}} = f(\bar{x}, \bar{u}))$ with k inputs $\bar{u}$ and l states $\bar{x}$ is flat if there exists an output $\bar{y}$ with the same dimension, and satisfying*

- *The components of y are independent*
- *$\bar{x}$ and $\bar{u}$ can be expressed as a function of $\bar{y}$ and its n derivatives*

$$\bar{x} = \phi(y, \dots\ y^{n-1}), \bar{u} = \Gamma(y, \dots\ y^{n-1}) \tag{2.18}$$

with ϕ and γ that satisfy $\dot{\phi} = f(\phi, \gamma)$.

This means that given the output of the system (i.e. the position of the load and the height of the ring), it is possible to obtain, without integrating a differential equation, the length of cables, velocity of cables, acceleration of cables and the corresponding forces in the cables.

2.2.1 Flatness calculation

Given the reference point (x, y) for the simplified model, the purpose is to find the corresponding cable length (which will be the reference for the position control of the motor), and the applied force in the cable. The same assumption that y is constant as was the case for the crane model is also made here.

Combining (2.15) with (2.10) and (2.12), one gets the following system of equations

$$\begin{aligned} 2(x_1 - x_B)\lambda_1 + 2(x_1 - x_A)\lambda_2 &= m\ddot{x_1} + gM(x_1 - x)/l_1 \quad (2.19) \\ 2(y_1 - y_B)\lambda_1 + 2(y_1 - y_A)\lambda_2 &= g(M + m) \quad (2.20) \end{aligned}$$

The unknown x_1 is found if (2.15) is combined with the first equation in (2.9).

$$x_1 = \frac{l_1 x''}{g} + x \quad (2.21)$$

From (2.21) we also get

$$\begin{aligned} \dot{x_1} &= \frac{l_1 x'''}{g} + x' \quad (2.22) \\ \ddot{x_1} &= \frac{l_1 x''''}{g} + x'' \quad (2.23) \end{aligned}$$

Now it is possible to solve the system (2.19-2.20) for λ_1 and λ_2.

In (2.9) one find that the input force to the system is

$$T_c = 2\lambda_2 L_1 + m_L \ddot{L_1} \quad (2.24)$$

To solve this equation, an expression for L_1 and $\ddot{L_1}$ has to be found. L_1 is solved using (2.2).

$$L_1 = \sqrt{(x_1 - x_A)^2 + (y_1 - y_A)^2} \quad (2.25)$$

$\ddot{L_1}$ in (2.24) is found by taking the the second derivative of (2.25) and combining with (2.14)

$$\ddot{L_1} = \frac{\ddot{x_1}^2 + (x_1 - x_A) - \dot{L_1}^2}{L_1} \quad (2.26)$$

The reference position for the motor is therefore given by (2.25), and the force in the cable is given by (2.24).

Chapter 3

DC motor

3.1 Theoretical model

The model that describes the crane does not take the dynamics of the DC motors into account. A way to improve the control of the crane system is to take these dynamics into account. The model of the DC motor, which is identified, consists of a DC motor with a pulley connected to the shaft. This is due to the fact that one would like to control the length and velocity of the cables, which is the same as the position and velocity of a point on the edge of the pulley.

A simple model [3] of a DC motor that relates the applied voltage to the velocity of the motor is illustrated in Fig. 3.1. This model is described by the following differential equations

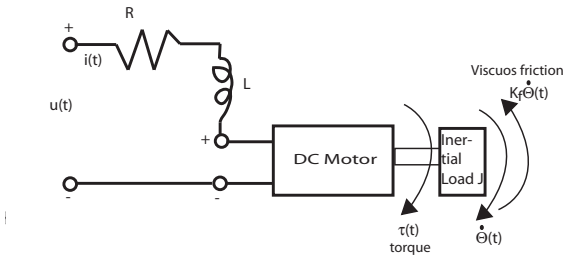

Figure 3.1: A simple model of a DC motor.

This system is described by the following differential equations

$$\frac{di}{dt} = -\frac{R}{L}i(t) - \frac{K_m}{L}\dot{\theta}(t) + \frac{1}{L}u(t) \tag{3.1}$$

$$\frac{d\dot{\theta}}{dt} = -\frac{k_f}{J}\dot{\theta}(t) + \frac{K_m}{J}i(t) \tag{3.2}$$

where R = resistance, L = inductance, K_m and K_f are motor constants, $\dot{\theta}$ the velocity of the pulley, I = current, U = applied voltage.

The differential equations above corresponds to a second order transfer function,

$$G(s) = \frac{K}{(s + T_e)(s + T_{mec})} \tag{3.3}$$

which is the transfer function from applied voltage to velocity of the "pulley"/cable. K is related to the dc-gain, T_e is the electrical time constant and T_{mec} is the mechanical time constant.

3.2 Experiment and estimation

The inertial load on the DC motors will change when the load is moved. This is due to the fact that the forces in the cables are dependent on the position of the load. The two extreme cases for the inertial load on the DC motor are

- Weight of load is not connected to the DC motor
- The whole weight of the load is connected to one DC motor.

The small inertial load corresponds to the pulley with a mass of 0.095 g. As a coarse estimate of the inertial load when the whole weight of the load is connected to the motor a pulley with a mass of 0.53 g is used.

A step response experiment is done for the two different cases. To avoid non-linearity's/artifacts, the step response experiment is done with a bias of 1 V. Thus the step input is 1-3 V.

The mechanical time constant of the motor without the pulley mounted is 6 ms [4]. As a rule of thumb the sampling period should be about 10-20 times faster than the dominating time constant of the system [5]. This gives an upper bound on the sampling period at 0.6 ms. Due to limitations in the software interface that controls the crane system, it is not possible to sample faster than 1 ms. This means that the measurements of the motor might be aliased.

It is desirable to have a simple model that describes the DC motor. Therefore a first order model is also estimated apart from the expected second order model. These two continuous time models are forced to fit the discreet time measurement data. The first order model (3.4) is fitted to the experimental step response data that's been filtered through a low pass filter with a cutoff frequency of 30 Hz to remove the measurement noise.

$$G(s) = \frac{\beta}{s + \alpha}. \tag{3.4}$$

The coefficients α and β are estimated with the "Least Squares Method". Since the experiment data are sampled, equation (3.4) is converted to a discrete-time version. By using Tustin's approximation on the transfer function (3.4) one gets

$$G(z) = \frac{Y(z)}{U(z)} = \frac{\alpha}{\frac{2}{h}\frac{z-1}{z+1} + \beta} \tag{3.5}$$

This discrete transfer-function is converted to a difference equation between input $u(k)$ and output $y(k)$

$$\left(\frac{\alpha}{\frac{2}{h}\frac{z-1}{z+1} + \beta} \right) y(k) = \alpha u(k), \tag{3.6}$$

and after some manipulations

$$y(k+1) - y(k) + \frac{h\beta}{2} y(k+1) + \frac{h\beta}{2} y(k) = \frac{h\alpha}{2} u(k) + \frac{h\alpha}{2} u(k+1). \tag{3.7}$$

introduce $a = \frac{h\beta}{2}, b = \frac{h\alpha}{2}$ and

$$\hat{y}(k) = y(k+1) - y(k) \tag{3.8}$$

$$\bar{u}(k) = u(k+1) + u(k) \tag{3.9}$$

$$\bar{y}(k) = y(k+1) + y(k) \tag{3.10}$$

then (3.7) becomes

$$\hat{y}(k) = a\bar{y}(k) + b\bar{u}(k) \tag{3.11}$$

This equation is linear and the coefficients are estimated with the least squares method. The regressor vector R and states θ are

$$\theta = \begin{bmatrix} a & b \end{bmatrix} \text{ and } R = \begin{bmatrix} \bar{y}(k) & \bar{u}(k) \end{bmatrix}^T \tag{3.12}$$

$$\hat{\theta} = \left(R^T R \right)^{-1} R^T \hat{y} \tag{3.13}$$

Thus a and b are estimated from 3.13, and thus one gets also a estimate of α and β.

The coefficients for the second order model are estimated from the same step response data. Instead of doing the least squares estimation "by hand" (as for the first order model), Matlab System Identification Toolbox is used. Since the model structure is known 3.3 the "Process Model" mode in 'System Identification Toolbox' has been used.

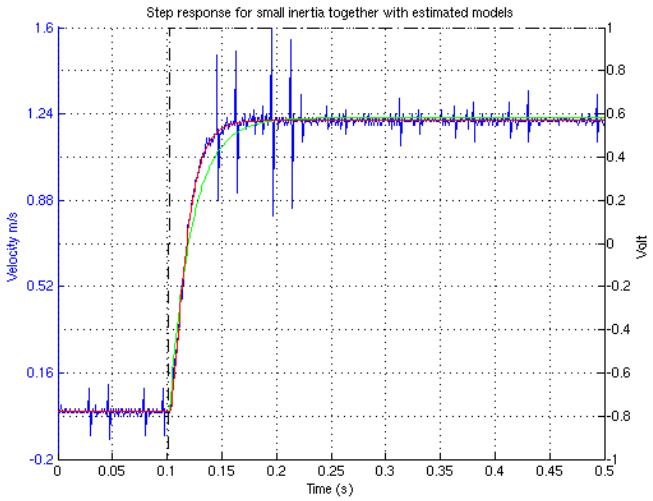

Figure 3.2: Step response for small inertia and estimated models. First order model *green*, second order model *red*, measurement *blue*.

3.2.1 Estimation Results

In Fig. 3.3 and Fig. 3.2 it is seen that both the first and second order models capture the dc-gain for the system. The second order model captures the rise time of the system much better. A close comparison between the characteristics of the models and the real experiment data is seen in Table. 3.2.1

The estimated transfer functions for the small inertial load are

$$G_{small}(s) = \frac{29.62}{s + 48.41} \tag{3.14}$$

$$G_{small}(s) = \frac{10240}{(s^2 + 280.9s + 16880)} \tag{3.15}$$

$$\tag{3.16}$$

The estimated transfer functions for the big inertial load are

$$G_{big}(s) = \frac{14.89}{s + 23.54} \tag{3.17}$$

$$G_{big}(s) = \frac{3153}{(s^2 + 174.2s + 5054)} \tag{3.18}$$

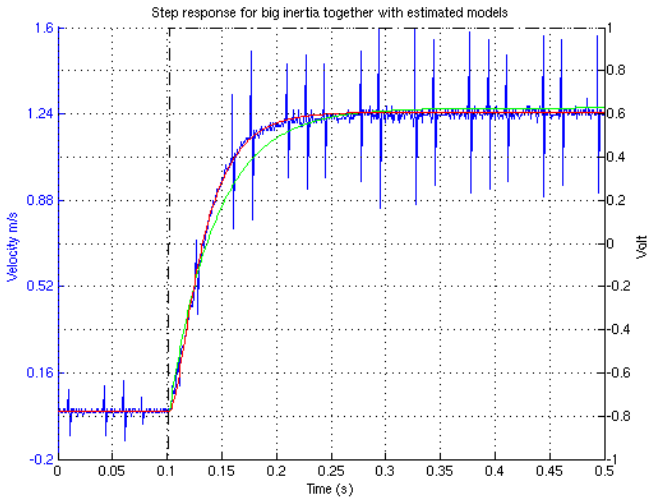

Figure 3.3: Step response for big inertia and estimated models. First order model *green*, second order model *red*, measurement *blue*.

Model	dcgain	rise time (ms)
Small Inertia	0.61	30
first order model	0.61	45
2:e order model	0.61	0.29
Big Inertia	0.62	62
first order model	0.63	0.93
2:e order model	0.62	0.63

3.3 Discrete-time model

At the beginning of the project it was considered to be advantageous to estimate a discrete-time model for the motor. Since this model has been used in the "ramp" design approach, a short description of how it was estimated will follow.

The same step response data which have been used for the estimation of the continuous-time model have been used here as well. The discrete-time model was estimated from the step response with the aid of Matlab's System Identifications Toolbox. Different type of ARX- and state space model structures was tried. The estimated models were chosen according to the following criteria

- dcgain
- rise-time
- model is minimum-phase
- low order model preferred

The model that gave the best correspondence with the real data was an ARX221 type structure for both the small and big inertia models.

$$G_{small}(z) = \frac{0.2819}{z^2 - 1.113z + 0.1594} \tag{3.19}$$

$$G_{big}(z) = \frac{0.0168}{z^2 - 1.09z + 0.1174} \tag{3.20}$$

Model	dcgain	risetime (ms)
$G_{small}(z)$	0.61	38
$G_{big}(z)$	0.62	70

Remark: The procedure which uses the same data for both estimation and validation of the model structures is not the correct approach for System Identification. Also when one estimates the parameters of an ARX structure, the excitation signal should preferably be a PRBS signal which, in theory, fully excites the system. Experiments were also done with different PRBS signals, but it was not possible to excite the system sufficiently well at low frequencies. This meant that these models where worse than the "wrongfully" identified discrete-time models above. Even though the discrete-time models are not totally correct, they can be used to show the concepts and problems with the "ramp" control approach.

3.4 Nominal model

The load on the DC motor changes when the load moves. Therefore a nominal model is estimated from the two extreme cases mentioned before. The nominal model is taken to be the mean of the two extreme cases.

The transfer function that corresponds to the nominal model is estimated from the Bode plot. It is assumed that the nominal model has the same transfer function structure as (3.3). This transfer function can be converted to the frequency domain by inserting $s = j\omega$ in (3.3)

$$G(j\omega) = \frac{b_0}{-\omega^2 + a_1 j\omega + a_2} = \frac{W(j\omega)}{U(j\omega)} \tag{3.21}$$

where $W(j\omega)$ is the output of the system and $U(j\omega) = 1$. Given the magnitude and phase in the bode plot one gets

$$W(j\omega) = magnitude \cdot e^{j \cdot phase} \tag{3.22}$$

By rearranging (3.21) one gets

$$b_0 = -\omega^2 W(j\omega) + a_1 j\omega W(j\omega) + a_2 W(j\omega). \tag{3.23}$$

This equation is linear, the coefficients can be estimated with the "Least Squares Method" (3.13).

The nominal transfer function is

$$G_{nom}(s) = \frac{6699}{s^2 + 232.6s + 1096} \tag{3.24}$$

The same approach is used for the discrete-time model, and the resulting nominal transfer function is

$$G_{nom}(z) = \frac{0.2261z + 0.0001477}{z^2 - 1.103z + 0.1402} \tag{3.25}$$

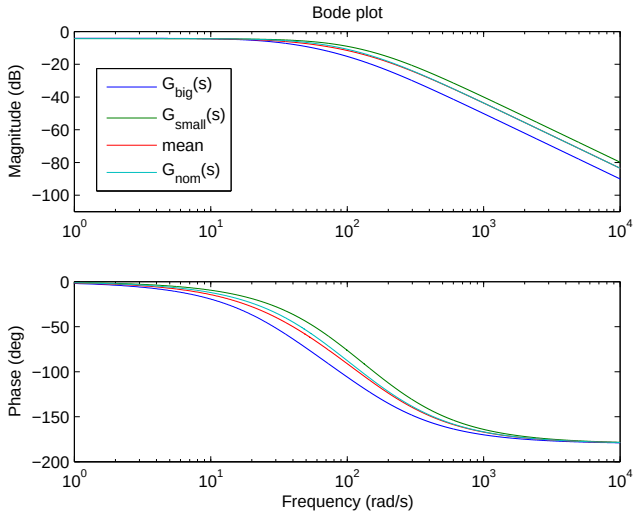

Figure 3.4: Bode plot of continuous-time model of the DC motor

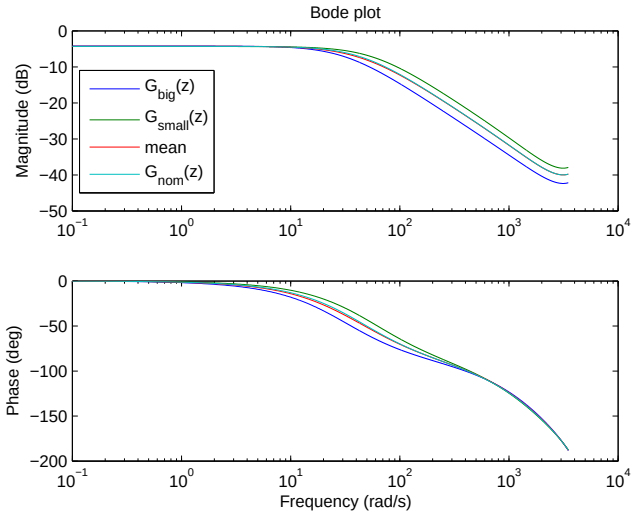

Figure 3.5: Bode plot of discrete-time model of the DC motor

Chapter 4

Control Strategy

The objective is to track a position reference for the cable, which is the same as the position of the motor. This reference signal for the motor is given by flatness conversion of the position reference of the load. It takes about 10-20 ms to do the flatness calculation for the whole crane system. This means that it is not possible to do position control for the whole system faster than this. One way to overcome this problem is to separate the control of the crane and the DC motors. By doing this, it is possible to sample the DC motors faster so as to satisfy the Shannon's sampling theorem.

Since the reference is only updated every 20 ms, one has to interpolate the reference in between these time instants in order to be able to use the fast sampling period for the DC motors. By using linear interpolation, a "smoother" acceleration of the load is obtained than what is possible for a step. Also the risk of inducing oscillations of the load decreases compared to "step" interpolation. Thus the controller to be designed should fulfill the following

- Track a ramp reference without a stationary error after 5-10 ms
- Reject load disturbances

4.1 Model match design

In order to track a ramp without any stationary error, the model $H_m(z)$ must fulfill $H_m(1) = 1$. For a ramp reference $Y_{ramp}(z)$, the relation between

$Y_{ramp}(z)$ and the output of the system $Y(z)$ is

$$Y_{ramp}(z)-Y(z) = Y_{ramp}(z)-H_m(z)Y_c(z) = Y_{ramp}(1-H_m(z)) = \frac{hz}{(z-1)^2}(1-H_m(z)) \tag{4.1}$$

This relation will only go to zero if [6]

$$\sum_{i=1}^{n} \frac{1}{1-p_i} = \sum_{j=1}^{m} \frac{1}{1-z_j} \tag{4.2}$$

where p_i are the zeros and z_j are the poles of $H_m(z)$.

The transfer function (3.5) is given between the voltage and the velocity. Since the control is done on the position, an integrator is added to (3.5).

$$G_{nom}(z) = \frac{0.0002261z + 1.477 \cdot 10^{-7}}{z^3 - 2.103z^2 + 1.243z - 0.1402}. \tag{4.3}$$

It is only possible to match $H(z) = B(z)/A(z)$ to $H_m(z) = B_m(z)/A_m(z)$ if the following conditions are fulfilled[7]

1. $F(z)$ need to have all poles inside the unit circle.
2. Deg $A_m(z)$-deg$B_m(z) \geq$ Deg $A_m(z)$-deg$B(z)$
3. All zeros of $B(z)$ outside the unit circle are retained in $B_m(z)$

A possible $H_m(z)$ that fulfills these conditions are

$$H_m(z) = \frac{z - p_1}{(z - z_1)(z - z_2)(z - z_3)} \tag{4.4}$$

To find a suitable $H_m(z)$ the poles were chosen to lie inside the unit circle and the pole was chosen so that $H_m(z)$ fulfills the condition (4.2) for tracking of a ramp. Through trial and error (so as to obtain a good $H_m(z)$ in order to track a ramp), suitable values for the poles were found.

$$H_m(z) = \frac{z - 0.8357}{(z - 0.1)(z - 0.3917)(z - 0.7)} \tag{4.5}$$

As can be seen in Fig. 4.1 the tracking error for $H_m(s)$ is less than 0.1 mm after just 20. But a drawback with $H_m(z)$ can be seen in the bode plot Fig. 4.2. At high frequencies there is strong amplification. Another drawback is that the tracking error will increase slowly with time (see Fig. 4.3). But since the increase is less than 0.1 mm after 10 s, the model was considered to be good enough to be used for a model matching design approach.

The model matching problem is solved for a second degree of freedom control structure Fig. 4.4.

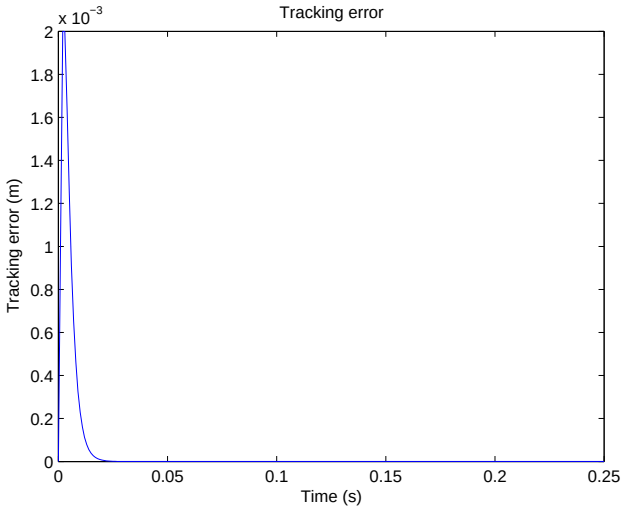

Figure 4.1: Tracking error for a ramp reference

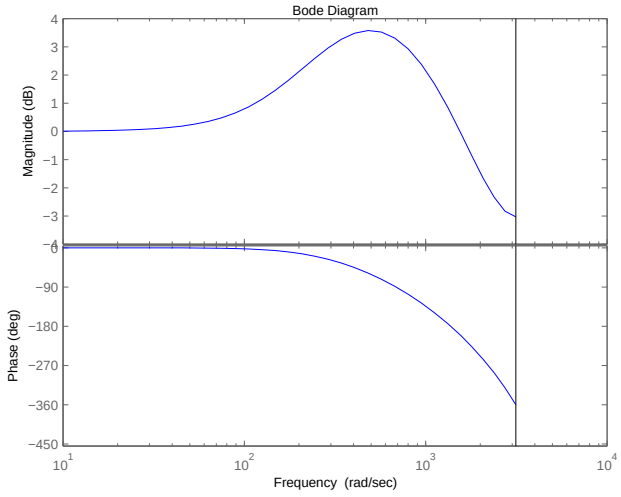

Figure 4.2: Bode plot of $H_m(z)$

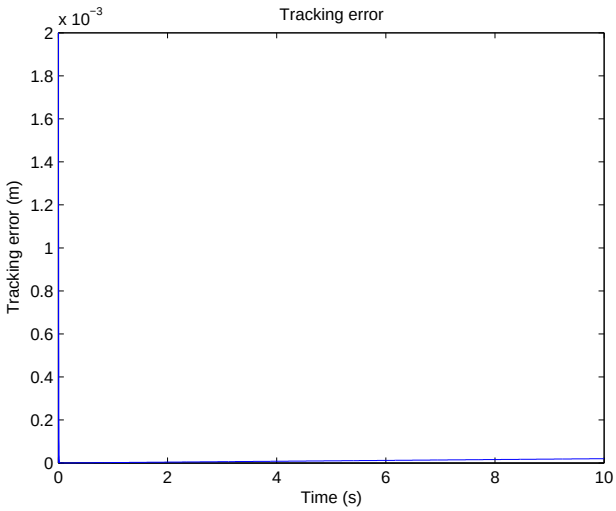

Figure 4.3: Tracking error for a ramp reference after 10 s

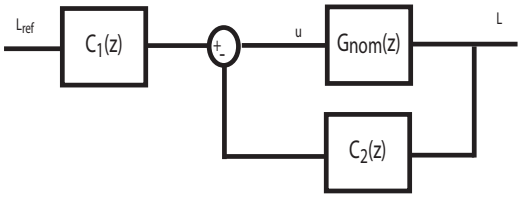

Figure 4.4: Second degree of freedom control structure used for the model matching problem

The algorithm [7] below is used to solve the model matching problem

The objective is to find two proper controllers $C_1(z) = L(z)/D(z)$ and $C_2(z) = M(z)/D(z)$.

First

$$\frac{H_m(z)}{B(z)} = \frac{B_m(z)}{A_m(z)B(z)} := \frac{\bar{B}_m(z)}{\bar{A}_m(z)} \tag{4.6}$$

this is transformed to

$$H_m(z) = \frac{\bar{B}_m(z)B(z)}{\bar{A}_m(z)} = \frac{L(z)B(z)}{D(z)A(z) + M(z)B(z)} \tag{4.7}$$

in order for $C_2(z)$ to be proper (4.7) rewritten to

$$H_m(z) = \frac{\bar{B}_m(z)\hat{A}(z)B(z)}{\bar{A}_m(z)\hat{A}(z)} = \frac{L(z)B(z)}{D(z)A(z) + M(z)B(z)} \tag{4.8}$$

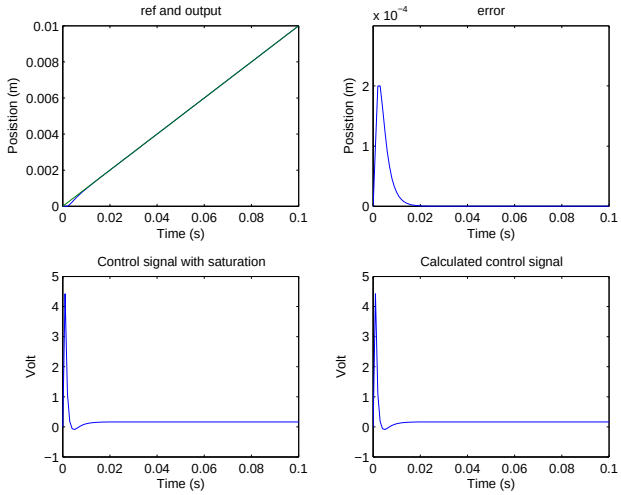

Figure 4.5: *Left upper:* Tracking of the green reference signal, *Right upper:* Tracking error, *Left bottom:* Control signal fed to the motor which is saturated at $pm5$ V, *Right bottom:* Control signal without saturation

where $\hat{A}(z)$ is a arbitrary Hurwitz polynomial such that the degree of $\bar{A}(z)\hat{A}(z)$ is $2n-1$, where n is the degree of the denominator of $H(z)$.

In this case, n = 3 and since $\bar{A}(z)$ has degree a degree of 4, $\hat{A}(z) = (z{+}0.3)$ is introduce to fulfill the condition above. Set $L(z) = \bar{A}(z)\hat{A}(z)$, $D(z)$ and $M(z)$ is solved from the Diophantine equation in the denominator of (4.8)

$$D(z)A(z) + M(z)B(z) = \hat{A}(z)\bar{A}(z) \tag{4.9}$$

The controllers are

$$C_1 = \frac{4.423 * 10^4 z^2 - 5.023 * 10^4 z + 1.109 * 10^4}{z^2 + 0.6178z + 0.003992} \tag{4.10}$$

$$C_2 = \frac{3.463 * 10^4 z^2 - 3.37 * 10^4 z + 4155}{z^2 + 0.6178z + 0.003992}. \tag{4.11}$$

4.1.1 Simulation

The suggested control structure is simulated in Matlab, with a ramp that has a slope of 0.1 m per second. The effect of a load disturbance of 5 Volts and a measurement noise of 1 mm is studied.

As can be seen in Fig. 4.5 the tracking of the reference signal is relatively good just after 10 ms. And in the Fig. 4.6 one sees that the controller rejects

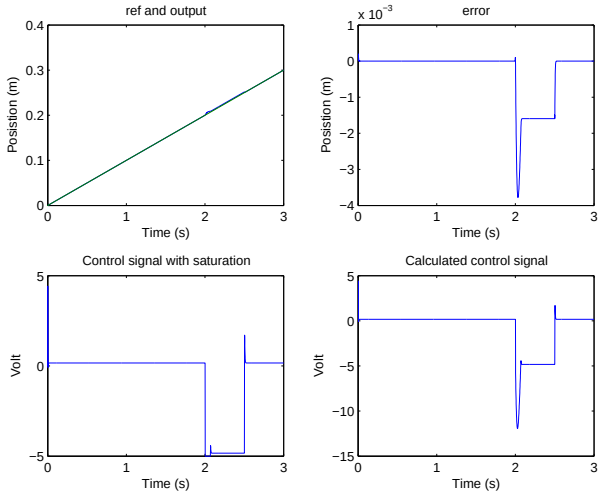

Figure 4.6: *Left upper:* Tracking of the green reference signal,*Right upper:* Tracking error, *Left bottom:* Control signal fed to the motor which is saturated at ±5 V,*Right bottom:* Control signal without saturation

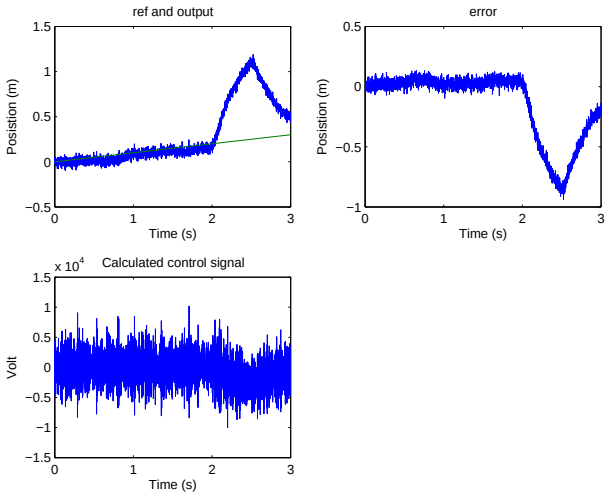

Figure 4.7: Tracking behavior when measurement noise of 1 mm is added. *Left upper:* Tracking of the green reference signal, *Right upper:* Tracking error, *Left bottom:* Control signal fed to motor which is saturated at ±5 V, *Right bottom:* Control signal without saturation

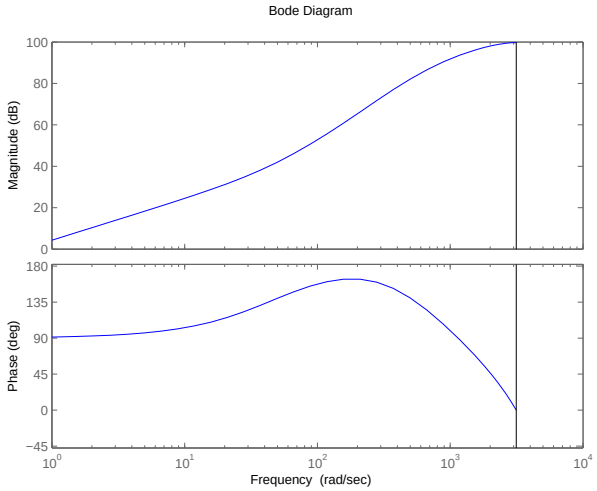

Figure 4.8: Bode plot of the transfer function $N2C(z) = \frac{C_2(z)}{1+C_2(z)H}N(z)$ from noise to command signal

the load disturbance (in the shape of a bump that is added between 2 and 2.5 s).

When a measurement noise of 1 mm is added, the control signal will blow up and saturate Fig. 4.7. The reason for this behavior is that the controller $C_2(z)$ will amplify high frequencies Fig. 4.8.

The problem with the ramp approach with the selected model (4.4) is that high frequencies is amplified in order to track the ramp within 10-20 ms. And since the open-loop model of the DC motor does not have a high gain at high frequencies, the controller $C_2(z)$ will have high gain for high frequencies in order to match $H(z)$ to $H_m(z)$. This makes the controller extremely sensitive to noise, as can be seen in Fig. 4.7. The control signal blows up which makes the controller saturate (± 5 Volt). Therefore this control approach has not been implemented on the real system.

Chapter 5

Reduced bandwidth design

The objective is to track a position reference for the cable (i.e. this is the same as controlling the position of the motor). From the flatness calculations, one gets the reference position of the cable, but it is also possible to get the force which is applied to the cable in order to move the load. This means that one also knows the influence of the load on the motor at every time instant along the trajectory. Therefore one can see the crane as a disturbance which acts on the motor. If one could find a transfer function that describes how the force in the cable influences the velocity of the motor, it would be possible to use a feed-forward controller to compensate for the crane.

The force in the cable will act as a torque T_{crane} on the motor. If this torque is added to the theoretical model (3.1) it is possible to get a transfer function that describes how the crane acts on the motor.

$$\frac{di}{dt} = -\frac{R}{L}i(t) - \frac{K_m}{L}\dot{\theta}(t) + \frac{1}{L}u(t) \tag{5.1}$$

$$\frac{d\dot{\theta}}{dt} = -\frac{k_f}{J}\dot{\theta}(t) + \frac{K_m}{J}i(t) + \frac{T_{crane}}{J} \tag{5.2}$$

The Laplace transform of this is

$$sI(s) = -\frac{R}{L}I(s) - \frac{K_m}{L}\dot{\theta}(s) + \frac{1}{L}U(s) \tag{5.3}$$

$$s\dot{\theta}(s) = -\frac{k_f}{J}\dot{\theta}(s) + \frac{k_m}{J}I(s) + \frac{T_{crane}}{J} \tag{5.4}$$

If the first equation in (5.3) is inserted in the second one

$$s\dot{\theta}(s) = -\frac{k_f}{J}\dot{\theta}(s) + \frac{k_m}{J(sL+R)}(U(s) - k_m\dot{\theta}(s)) + \frac{T_{crane}}{J} \tag{5.5}$$

In (5.5) there are two different cases $U(s) = 0$ and $T_{crane} = 0$.

$U(s) = 0$

$$s\dot{\theta}(s) + \frac{k_f}{J}\dot{\theta}(s) + k_m \frac{k_m}{J(sL+R)}(\dot{\theta}(s)) = \frac{T_{crane}}{J} \tag{5.6}$$

This is rewritten as

$$\dot{\theta}(s) = \frac{(s + R/L)}{s^2 + \left(\frac{R}{L} + \frac{k_f}{J}\right)s + \frac{1}{JL}\left(k_f R + k_m^2\right)} T_{crane} \tag{5.7}$$

$T_{crane} = 0$ is inserted in (5.5) and after some manipulations one get

$$\dot{\theta}(s) = \frac{\frac{k_m}{JL}}{s^2 + \left(\frac{R}{L} + \frac{k_f}{J}\right)s + \frac{1}{JL}\left(k_f R + k_m^2\right)} U \tag{5.8}$$

This equation has the same structure as the model identified (3.3) of the DC motor. Thus the coefficients in (5.8) are known. This means that the coefficients of the denumerator in (5.7) are also known, since it has the same denumerator as (5.8). And the relation R/L in the denominator of (5.7) is also known, if one uses the values of R and L form the data sheet of the motor (ref to data sheet here).

$$\dot{\theta}(s) = \frac{6699}{s^2 + 232.6s + 1096} U \tag{5.9}$$

$$\dot{\theta}(s) = \frac{(s + 6280)}{s^2 + 232.6s + 1096} T_{crane} \tag{5.10}$$

Where the torque $T_{crane} = r_{pully} \cdot F_{crane}$. This means that it is possible to use a feed-forward controller to compensate for the crane effect on the motor, and a feedback position controller to control the length of the cables.

5.1 Position control

It is hard to do a good position controller directly on the motor in the fast time scale (1 ms). One of the problem in doing position control directly on the motor is that it is easy to get the motor to rattle. This reduces the life of the motor, and in this case it would probably induce oscillations of the load. Another problem is that it is not possible to have a position controller which has a faster sampling time than the one needed to do the flatness calculations. The minimum sampling period for the flatness calculation is about 20 ms for the full crane system. The rise time of the motor is about 37 ms for the nominal model. Since one should sample the motor at about 4 -10 times the rise time [5], a sampling time of 20 ms is too large. This

means that the motor has to be made slower. This can be done buy using the following cascade control structure see Fig 5.7.

A way to artificially make the motor slower is to do velocity control with a sampling period of 1 ms that reduces the bandwidth (equivalent to a longer rise time). In order to have a sampling period of 20 ms for the position controller, the rise time of the 'slow' motor should be about 90-100 ms. The velocity controller is designed with loop-shaping using the 'Siso Toolbox' interface. The desired characteristic of the "new" slow motor, are

- A rise time of about 90-100 ms.

- The dc-gain of the closed-loop system should be around 0.6, which is the dc-gain of the open-loop system.

The designed controller has the following transfer function

$$C_{vel}(s) = \frac{2.6}{0.14s + 1} \tag{5.11}$$

This gives the following transfer function for the closed-loop system

$$G_{slow}(s) = \frac{1.244 * 10^5}{s^3 + 239.8s^2 + 1262s + 2.027 * 10^5} \tag{5.12}$$

which is the new slow motor.

Parameter	Value
Risetime	95 ms
dcgain	0.61
bandwidth	3.62 Hz

The design criteria for the position controller are based on the dynamics of the crane. The time constant of the crane is about 0.9 s, this implies that the rise time for the position control of the motor should be about 100 ms. The controller is designed with the following criteria in mind

- The rise time for a step should be about 100 ms

- The overshoot should be small (less than 10 %)

- Reject a bump disturbance

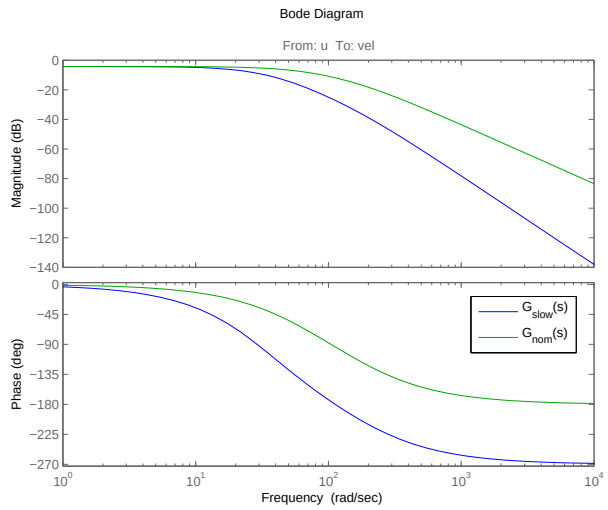

Figure 5.1: Bode plot of $G_{slow}(s)$ (blue)

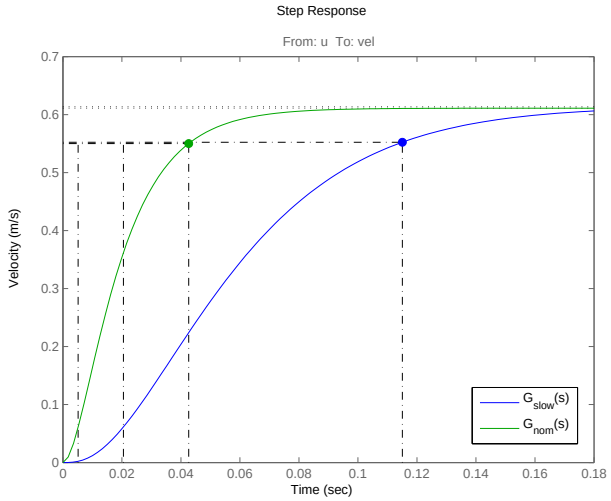

Figure 5.2: Step plot of $G_{nom}(s)$ (green) and closed loop velocity feedback (blue)

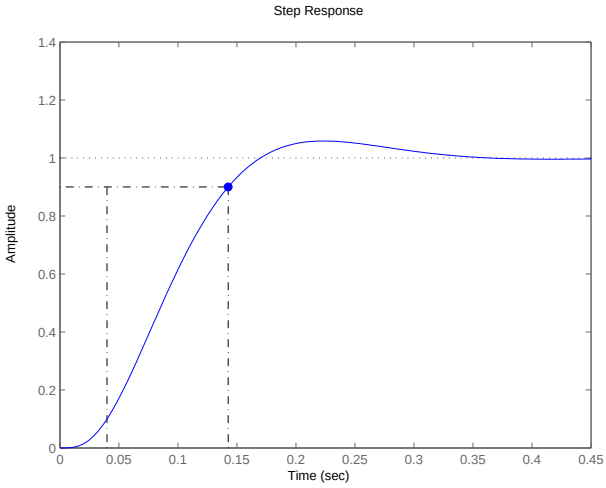

Figure 5.3: Step-response of closed loop system with position control

The position controller is designed with a combination of loop shaping and root locus design using the "Siso Tool" in Matlab's control system toolbox. The transfer function of the position controller is

$$C_{pos}(s) = \frac{46.63s + 14142}{s + 72.27} \tag{5.13}$$

The step response in of the closed loop-system Fig. 5.3 is fast with a 103 ms rise time, which is close to specification, and the overshoot is smaller then the specified 10 %. And in Fig. 5.4 one sees that the closed-loop system has a high gain margin and phase margin, which means that the closed loop system is robust to uncertainties in the model. The controller is also able to reject a 'bump' disturbance see Fig. 5.5. In Fig. 5.6 one sees that measurement noise with a amplitude of 1 mm will be amplified in the closed-loop system, and the tracking error will be quite big.The only good thing is that the control signal does not blow up and saturate the motor.

5.1.1 Feed forward controller

The full control scheme Fig 5.7 consist of a flatness block that gives the reference position of the cable/motor and the corresponding force in the cable. The above mentioned cascade structure is used for position control, and a feed-forward part. Since the crane acts as a disturbance on the velocity of the motor, one realizes from Fig 5.7 that the following condition has to

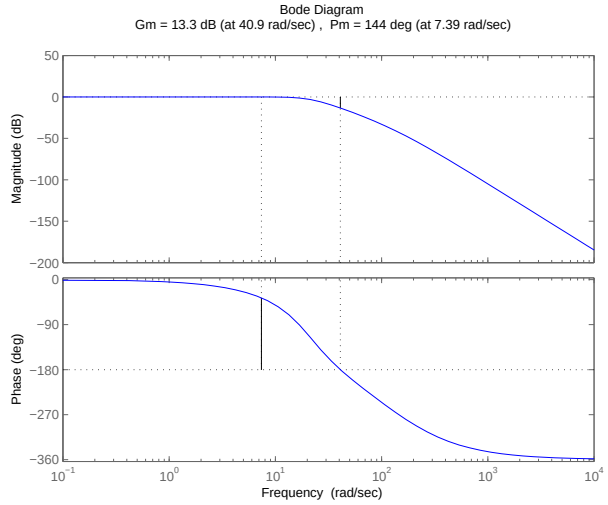

Figure 5.4: Bode plot of the resulting closed loop with position control

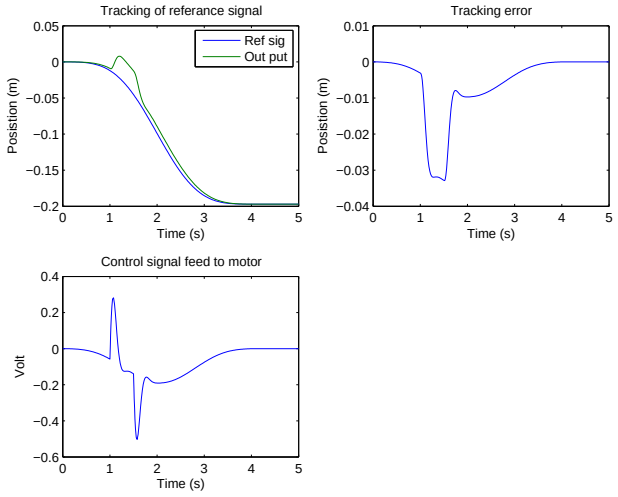

Figure 5.5: *Left upper:* Tracking performance for the closed loop system and rejection of a bump disturbance of amplitude 0.5 that enters the system between 1-1.5 s, *Right Upper:* Tracking error, *Left Lower:* Control signal feed to the motor

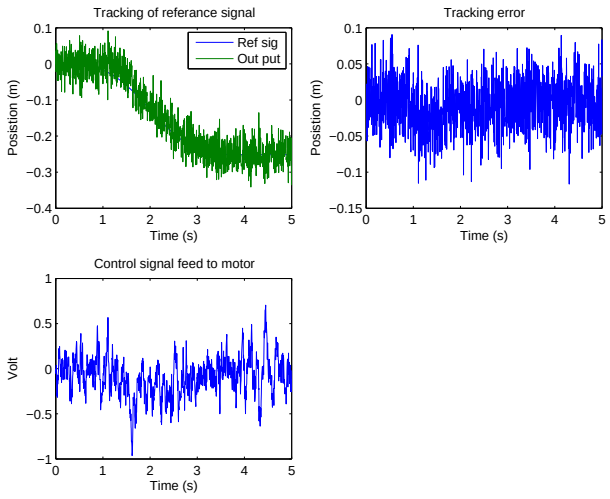

Figure 5.6: *Left upper:* Tracking performance for the closed loop system with 1 mm measurement noise added, *Right Upper:* Tracking error, *Left Lower:* Control signal feed to the motor

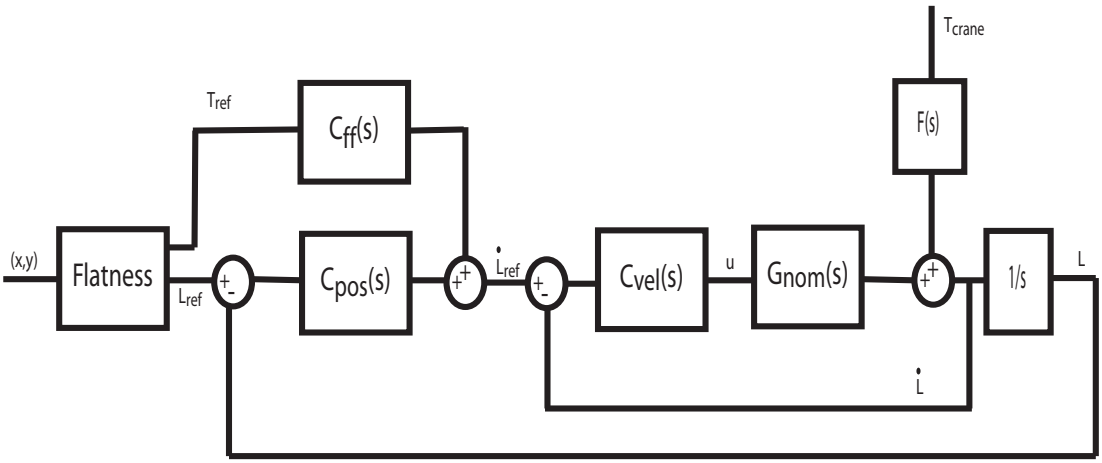

Figure 5.7: Control scheme. L_{ref} is the same as θ

be fulfilled for the feed-forward $C_{ff}(s)$ part to be able to compensate for the crane disturbance.

$$T_{ref}(s)C_{ff}(s)C_{vel}(s)G_{nom}(s) + T_{ref}(s)r_{pully}F(s) = 0 \tag{5.14}$$

where r_{pulley} has been added to convert the force T_{ref} to a torque on the motor since the transfer function $F(s)$ (5.10) is a transfer function from torque to velocity.This gives

$$C_{ff}(s) = -\frac{F(s)}{C_{vel}(s)G_{nom}(s)} \tag{5.15}$$

$$C_{ff}(s) = \frac{-0.0035s^4 - 22.83s^3 - 5314s^2 - 2.77 \cdot 10^5 s - 1.721 \cdot 10^6}{1.742 \cdot 10^4 s^2 + 4.052 \cdot 10^6 s + 1.909 \cdot 10^8} \tag{5.16}$$

For the cascade position controller to track a reference signal perfectly, one normally inverts the closed-loop system. In this case, it is not necessary to do this, since the closed-loop system has the shape of a low-pass filter Fig. 5.4, which means that the slow varying reference signal will just pass through.

Chapter 6

Simulation

The suggested control structure is set up with "Simulink" in Matlab. The flatness block and the model of the crane are implemented as S-Functions. The crane model needs the force in the cable as an input. The input force can be found by feeding the signal $\dot{L}$ trough $r_{pulley}/F(s)$, since $F(s)$ is the transfer function from the torque applied on motor by the crane to the velocity of the motor $\dot{L}$. (The r_{pulley} in the numerator is for converting the torque of the motor to the corresponding cable force). From Fig 5.7 an expression for the F_{cable} is found.

$$\begin{aligned} F_{cable} &= [(1 + C_{vel}(s)G_{nom}(s) + C_{pos}(s)C_{vel}(s)G_{nom}(s)/s)\dot{L} \\ &\quad -(T_{ref}(s)C_{ff}(s) + L_{ref}C_{pos}(s))C_{vel}(s)G_{nom}(s)]\frac{r_{pulley}}{F(s)} \end{aligned} \quad (6.1)$$

The force that is fed to the control structure is just the output of the crane model. Since all transfer functions in Simulink have to be causal for the simulation to work , extra poles must be added to the feed-forward controller. This is done by adding two poles at at -250 which are faster than the fastest of (5.16). This gives the new feed forward controller

$$C_{ff}(s) = \frac{-0.01382s^4 - 90.80s^3 - 2.098 \cdot 10^4 s^2 - 1.096 \cdot 10^6 s - 6.93 \cdot 10^6}{s^4 + 757.6s^3 + 2.018s^2 + 2.175 \cdot 10^7 s + 7.536 \cdot ^8} \quad (6.2)$$

The same is also done for $1/F(s)$ where an additional pole at -7000 is added.

The setup for the simulation of the crane system was according to the simplified crane model. The initial position of the load and other physical properties of the system are given in Table 6

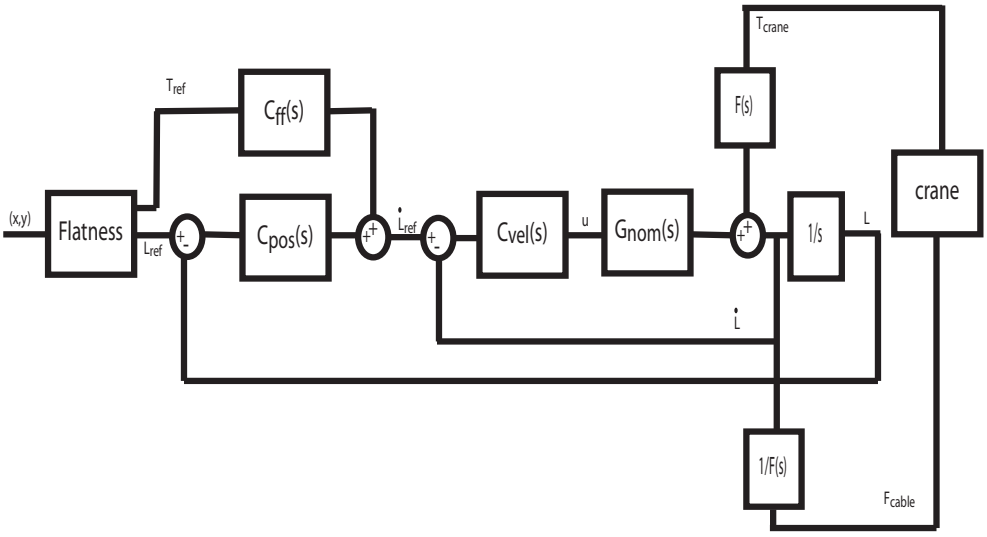

Figure 6.1: Simulation setup

Parameter	value	unit
x_a	0.05	m
y_a	0.57	m
x_b	0.58	m
y_b	2.62	m
$\bar{y_1}$	0.61	m
y_1	0.61	m
l_1	0.37	m
g	9.82	m/s^2
M	0.277	kg
m	0.005	kg
m_L	$3 \cdot 10^{-6}$	kgm^2

As a crude estimate of m_L the inertia of the pulley is used.

In order to see whether the suggested control scheme works, one has to move the load quite fast. Otherwise, it is not possible to see the effect of the flatness part, since the pulling point and the load will follow each other. In the experiment, the position of the load is moved 10 cm in 1 s, according to the trajectory in Fig. 6.2.

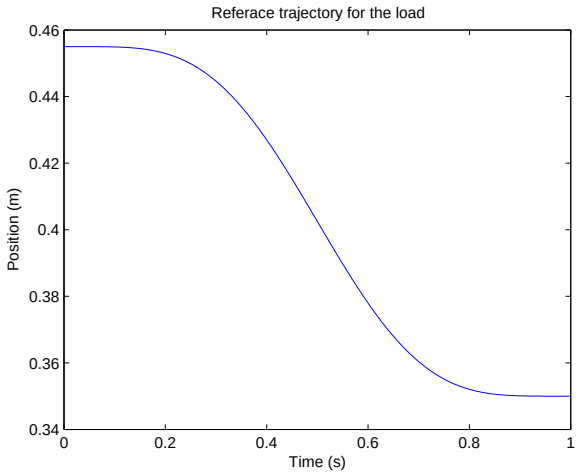

Figure 6.2: Reference trajectory for the load.

6.1 Results

Due to problems with the initial state of the simulation model, it has not been possible to do a full simulation where the interaction between the motor and the crane model works. The problem is that the transfer function that is supposed to give the input force to the crane model does not give the correct output when the motor and crane model are connected together, and it is also very sensitive to the initial condition of the whole model. Therefore the T_{ref} given by the flatness has been used as the input to the crane model during the simulation to simulate the effect of the crane system on the motor.

In the right plot in Fig. 6.3, one sees that the position of the load follows the trajectory perfectly given the corresponding input force from the flatness calculations.

The plots in Fig. 6.4 show the reference trajectory for the cable and how well the motor tracks this trajectory. The physical interpretation of this trajectory is that the cable will accelerate fast at the beginning. This means that the position of the load will fall behind the position of "pulling point" (x_1, y_1) in Fig 6.4. Therefore the cable is "slowed down" which is the same as relaxing the cable a bit, in order for the load to "catch up". This is the "dip" at 0.6 seconds in the figure. Then when the load has "caught up" with the pulling point, the cable accelerates again in order to stop the load at the desired position. Also note that the movement of cable will lag about 0.1 s behind the reference trajectory.

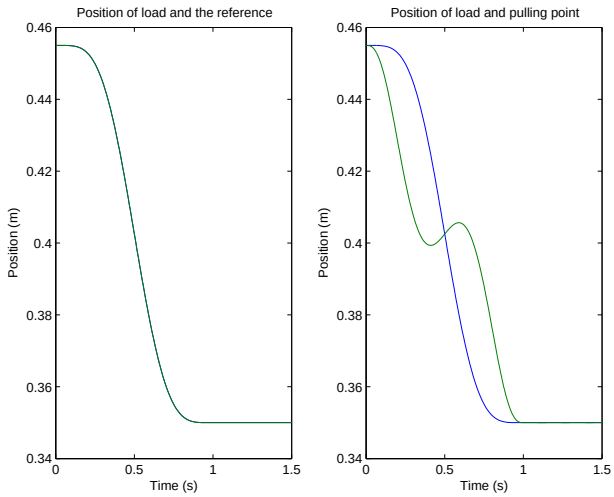

Figure 6.3: *Right:* The movement of load along the reference trajectory.*Left:* The movement of the load *(blue)* and pulling point *(green)*

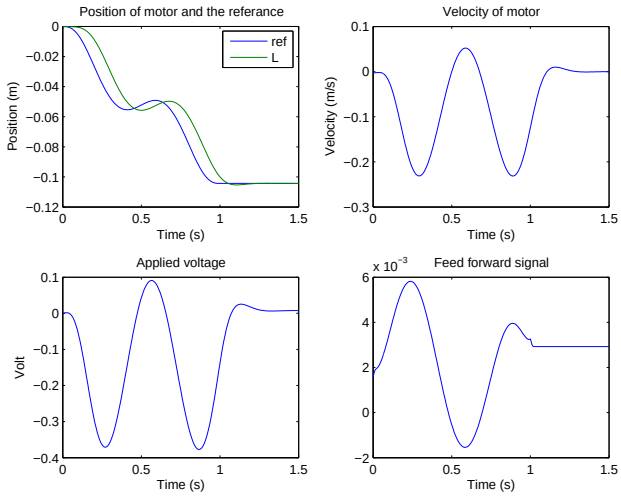

Figure 6.4: The behavior of the motor given the reference trajectory for moving the load.

Chapter 7

Implementation

The software interface, which is used to control the crane model, is a combination of a LabVeiw interface which runs a real-time kernel. In the current setup of the LabView interface, it is only possible to have one thread for which the sampling time can be set arbitrary above 0.5 ms. Due to this, it is not possible to have one thread for the velocity controller with a sampling period of 1 ms and another one for the flatness calculation and position controller running at 20 ms. Therefore the whole control algorithm is implemented in only one thread running at 1 ms. The slow sampling rate used for the position controller is just implemented to run once for every 20 runs of the fast 1 ms thread.

7.1 Experiment on the crane model

The experimental setup of the crane system was done according to the simplified crane model. All parameters and the reference trajectory were the same as in the simulation part.

7.1.1 Results

With the designed control scheme, it is possible to move the load along the given trajectory. Fig. 7.1 shows the reference trajectory for the cable given by the flatness block. Note, that the results are very similar to the results of the simulation.

The final positioning error of the load is about 0.5 cm. This error is probably mainly due to the fact that all parameters in the model are not correctly identified, and that the model is too simple. But a another small contributing fact may be the small stationary error for the cable position also seen in Fig 7.1. When the load arrives at it is destination it will oscillate

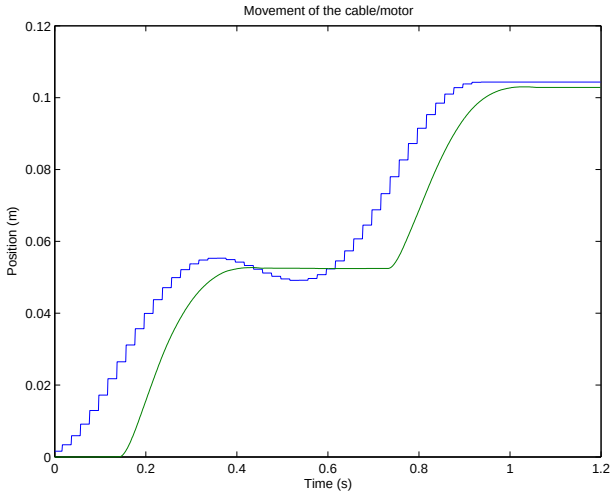

Figure 7.1: Movement of the cable/position of motor for the given reference trajectory

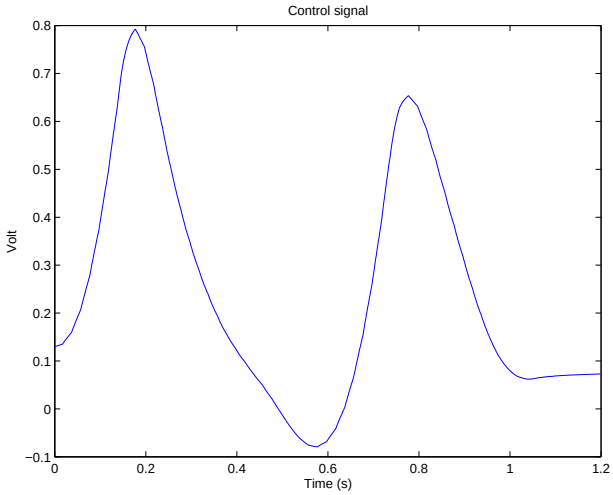

Figure 7.2: Control signal.

a bit. The main factor behind this is that the model of the system is too simple.

Chapter 8

Conclusion

This master thesis project has resulted in the identification of two different models for the DC motors which are controlling the crane. This resulted in (i) a continuous-time version that describes the dynamics of the DC motor well, and (ii) a discrete-time version which was not that good. The main problem with the identification of the model for the motors was that the motors were very fast. This meant that the software interfaced connected to the motors could not sample fast enough, without inducing aliasing of the measurement data. This meant that the identified models were not totally reliable, and also that it would be hard to control the motors.

Using the continuous-time model of the DC motor, it was possible to design a control scheme based on flatness together with a cascade position controller with feed-forward. Due to time limitations, it has not been possible to implement the localized controller with the full scale crane system. The main difference between the full-scale system and the simple model is that flatness calculations are more complex. In the full-crane system, one can see that the DC motors run independently of each other. This means that it is reasonable to assume that it is possible to use localized control scheme for the full-crane model as well. The main difference would be that the feed-forward term that compensates for the crane behavior on the motors would be more complicated.

For the discrete-time model, a control approach based on ramp tracking (within the 20 ms it takes for the flatness reference to be updated) was tried. The designed controller needed to be very aggressive, in order to track the ramp reference within 20 ms. Due to the characteristics of the model of the motor, the controller needed to amplify high frequencies a lot. This leads to a controller that did not work in the presence of measurement noise.

Chapter 9

Discussion

The designed localized controller worked quite well for the simple model. No problems with noise have been noticed in the real experiment. But, since the simulations of the motor showed that the control structure was a bit sensitive to noise, this might be something one can improve upon. Also, it would be good to do a controller which is a bit faster compared to the current one. This might reduce the time delay seen in the simulations and in the experiment Fig. 7.1.

Since the flatness calculation also gives the reference velocity for the cable, it might be a good idea to implement a controller which uses both the position and the velocity as reference signals. In my view this should give a controller which has the ability to control the cables better.

The concept of taking the nominal model of the motor as the mean of the two extreme cases *(only pulley connected or the full load connected to motor)* is not totally correct. A better approach would have been to identify one model for the motor with only the pulley connected, since the effect of the load is canceled by the feed forward term in the controller for the reduced bandwidth design.

At the moment it is not possible to run two different control threads together with the LabView interface. This is something that has to be addressed in order to implement the proposed control scheme for the whole crane, which has one fast velocity controller at 1 ms and then a slow position controller.

Bibliography

[1] Buccieri, Davide (2003) *Commande et aide au pilotage d'une grue 3D ne comportant aucune pièce mobile*, Laboratioire d'automatique a Ecole Polytechnique Fédérale de Lausanne.

[2] Kiss, B., Mullhaupt, Ph. and Lévine, J. (1999) *Modelling, Flatness and Simulation of a Class of Cranes*. Periodica Polytechnica, Electrical Engineering, vol. 43, pp 215-225.

[3] Matlab documentation www.mathworks.com.

[4] www.maxonmotor.com, product nr 118797.

[5] Åström, K.J and Wittenmark, B. (1997) *Computer Controlled Systems, Theory and design (Third Edition)*. Prentice Hall, Ch 3.5 p 110.

[6] Longchamp, R. (1995) *Commande numrique de systemes dynamiques*,PPUR, p 308.

[7] Chen ,Chi-Tsong (1999) *Linear System Theory and Design (Third Edition)*. Oxford University Press, 9.3,p 285-288.

Appendix A

Matlab code for simulation

A.1 Simplified crane model

Listing from source file `cranemodelwL.m`

```
function [sys,x0,str,ts] = cranemodel(t,x,u,flag,x0,mL
    ,M,m,xa,ya,xb,yb,y1bar,l1,y1,g)
% Statespace model of simplifyed spidercrane
%  x = (x,dx/dt,x1,dx1/dt)


%Parameters
%mL,M, m, xa, ya, xb, yb, y1bar, l1 = 0.22, y1 = 0.61,
     g = 9.81


switch flag,

  %%%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%%
  case 0,
     [sys,x0,str,ts]=mdlInitializeSizes(x0);

  %%%%%%%%%%%%%%%%%
  % Derivatives %
  %%%%%%%%%%%%%%%%%
  case 1,
```

```
    sys=mdlDerivatives(t,x,u,mL,M,m,xa,ya,xb,yb,y1bar,
        l1,y1,g);

  %%%%%%%%%%%%
  % Outputs %
  %%%%%%%%%%%%
  case 3,
    sys=mdlOutputs(t,x,u,mL,M,m,xa,ya,xb,yb,y1bar,l1,
        y1,g);

  %%%%%%%%%%%%%%%%%%%%%
  % Unhandled flags %
  %%%%%%%%%%%%%%%%%%%%%
  case { 2, 4, 9 },
    sys = [];

  %%%%%%%%%%%%%%%%%%%%%
  % Unexpected flags %
  %%%%%%%%%%%%%%%%%%%%%
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end
% end csfunc

%
%=============================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample
    times for
% the S-function.
%=============================================================
%
function [sys,x0,str,ts]=mdlInitializeSizes(x0)

sizes = simsizes;
sizes.NumContStates  = 6;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 6;
sizes.NumInputs      = 1;
```

```
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;

sys = simsizes(sizes);
x0  = [x0  0.3523];
str = [];
ts  = [0  0];

% end mdlInitializeSizes
%
%=============================================================
% mdlDerivatives
% Return the derivatives for the continuous states.
%=============================================================
%
function sys=mdlDerivatives(t,x,u,mL,M,m,xa,ya,xb,yb,
    y1bar,l1,y1,g)

%states

sx = x(1);
sxd = x(2);
sx1 = x(3);
sx1d = x(4);

L1 = sqrt((sx1-xa)*(sx1-xa)+(y1bar-ya)*(y1bar-ya));
L1d = (sx1-xa)*sx1d/L1;


T1 = u; %input force

A = [m   -2*(sx1-xb)  -2*(sx1-xa);...
     0  2*(y1-yb)  2*(y1-ya);...
     mL*(sx1-xa)/L1  0  2*L1];

B = [-g*M*(sx1-sx)/l1;  g*m+g*M;  T1-mL*(sx1d*sx1d-L1d*
    L1d)/L1];

%solve Ax=B;
par = mldivide(A,B);
```

```
sx1dd = par(1);
lambda1 = par(2);
lambda2 = par(3);

sxdd = g*(sx1-sx)/l1;

sys = [sxd sxdd sx1d sx1dd 0 0];

% end mdlDerivatives
%
%=============================================================
% mdlOutputs
% Return the block outputs.
%=============================================================
%
function sys=mdlOutputs(t,x,u,mL,M,m,xa,ya,xb,yb,y1bar
    ,l1,y1,g)

sx = x(1);
sxd = x(2);
sx1 = x(3);
sx1d = x(4);


L1 = sqrt((sx1-xa)*(sx1-xa)+(y1bar-ya)*(y1bar-ya));
L1d = (sx1-xa)*sx1d/L1;
T2 = u;

A = [m  -2*(sx1-xb) -2*(sx1-xa);...
     0  2*(y1-yb)  2*(y1-ya);...
     mL*(sx1-xa)/L1  0  2*L1];

B = [-g*M*(sx1-sx)/l1;  g*m+g*M;  T2-mL*(sx1d*sx1d-L1d*
    L1d)/L1];
%solve Ax=B;
par = mldivide(A,B);
```

```
sx1dd = par(1);
lambda1 = par(2);
lambda2 = par(3);

sxdd = g*(sx1-sx)/l1;

%only used for simulation purpurs
%update T2motor and L1
L1d2 = (-L1d*L1d + sx1d*sx1d + (sx1-xa)*sx1dd)/L1;
T2motor = mL*L1d2 + 2*lambda2*L1;

sys = [sx sxd sx1 sx1d T2motor L1];

% end mdlOutputs
```

A.2 Flatness calculation

Listing from source file `flatnesswLrefcont.m`

```
function [sys,x0,str,ts] = flattness(t,x,u,flag,Tinit,
    mL,M,m,xa,ya,xb,...
     yb,y1bar,l1,y1,g)

%S-function for continoius time flatness calculation
    of referance for
%simplifzed crane model
%The output is the lenght referance of cable and the
    force in the cable
%The input signal are u and it's four first derivitves
%Parameters
%mL,M, m, xa, ya, xb, yb, y1bar, l1 = 0.22, y1 = 0.61,
     g = 9.81

switch flag,

  %%%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%%
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes(Tinit);
```

```
%%%%%%%%%%%%%%%%%
% Derivatives %
%%%%%%%%%%%%%%%%%
case 1,
  sys=mdlDerivatives(t,x,u,mL,M,m,xa,ya,xb,yb,y1bar,
      l1,y1,g);

%%%%%%%%%%
% Update %
%%%%%%%%%%
case 2,
  sys=mdlUpdate(t,x,u,mL,M,m,xa,ya,xb,yb,y1bar,l1,y1
      ,g);

%%%%%%%%%%%
% Outputs %
%%%%%%%%%%%
case 3,
  sys=mdlOutputs(t,x,u,mL,M,m,xa,ya,xb,yb,y1bar,l1,
      y1,g);

%%%%%%%%%%%%%%%%%%%%%%%%%
% GetTimeOfNextVarHit %
%%%%%%%%%%%%%%%%%%%%%%%%%
case 4,
  sys=mdlGetTimeOfNextVarHit(t,x,u);

%%%%%%%%%%%%%%
% Terminate %
%%%%%%%%%%%%%%
case 9,
  sys=mdlTerminate(t,x,u);

%%%%%%%%%%%%%%%%%%%%%%
% Unexpected flags %
%%%%%%%%%%%%%%%%%%%%%%
otherwise
  error(['Unhandled flag = ',num2str(flag)]);
```

```
end

% end sfuntmpl

%
%
    ==========================================================

% mdlInitializeSizes
% Return the sizes, initial conditions, and sample
    times for
%the S-function.
%
    ==========================================================

%
function [sys,x0,str,ts]=mdlInitializeSizes(Tinit)


sizes = simsizes;

sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 2;
sizes.NumInputs      = 5;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;   % at least one sample time
    is needed

sys = simsizes(sizes);

%
% initialize the initial conditions
%
x0  = [];%Tinit;

%
% str is always an empty matrix
%
str = [];
```

```
%
% initialize the array of sample times
%
ts  = [0  0];

% end mdlInitializeSizes

%
%=============================================================
% mdlDerivatives
% Return the derivatives for the continuous states.
%=============================================================
function sys=mdlDerivatives(t,x,u,mL,M,m,xa,ya,xb,yb,
    y1bar,l1,y1,g)
sys = [];

% end mdlDerivatives

%
%=============================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and
     major
%time step requirements.
%=============================================================
%
function sys=mdlUpdate(t,x,u,mL,M,m,xa,ya,xb,yb,y1bar,
    l1,y1,g)

sys = [];
% end mdlUpdate

%
%=============================================================
% mdlOutputs
% Return the block outputs.
%=============================================================
%
```

```
function sys=mdlOutputs(t,x,u,mL,M,m,xa,ya,xb,yb,y1bar
    ,l1,y1,g)

%referance trajectorz and it's derivitives
xref = u(1);
xd1 = u(2);
xd2 = u(3);
xd3 = u(4);
xd4 = u(5);


%caltulate the x1 and it's first and second derivitive

x1 = l1*xd2/g+xref;
x1d1 = l1*xd3/g+xd1;
x1d2 = l1*xd4/g+xd2;


%L1 and it's derivitives
L1 = sqrt((x1-xa)*(x1-xa)+(y1bar-ya)*(y1bar-ya));
L1d1 = (x1-xa)*x1d1/L1;
L1d2 = (-L1d1*L1d1 + x1d1*x1d1 + (x1-xa)*x1d2)/L1;

%Algebraic soultion of Ax=B
%lambda1 = (2*(y1bar-ya)*(m*x1d2 +g*M*(x1-xref)/l1) -
    2*(x1-xa)*(m+M)*g)/(4*(x1-xb)*(y1bar-ya)...
%           -4*(y1-yb)*(x1-xa));
%lambda2 = (-2*(y1bar-yb)*(m*x1d2 +g*M*(x1-xref)/l1) +
     2*(x1-xb)*(m+M)*g)/(4*(x1-xb)*(y1bar-ya)...
%           -4*(y1-yb)*(x1-xa));


A = [2*(x1-xb)   2*(x1-xa);2*(y1bar-yb) 2*(y1bar-ya)];
B = [(m*x1d2+g*M*(x1-xref)/l1) g*(M+m)]';

lambdas = mldivide(A,B);
lambda2 = lambdas(2);


Tc = mL*L1d2 + 2*lambda2*L1;
```

```
sys = [Tc L1];


% end mdlOutputs

%
%=============================================================
% mdlGetTimeOfNextVarHit
% Return the time of the next hit for this block.
    Note that the result is
% absolute time.  Note that this function is only used
     when you specify a
% variable discrete-time sample time [-2 0] in the
    sample time array in
% mdlInitializeSizes.
%=============================================================
function sys=mdlGetTimeOfNextVarHit(t,x,u)

sampleTime = 0.01;     % Example, set the next hit to
    be one second later.
sys = t + sampleTime;

% end mdlGetTimeOfNextVarHit

%
%=============================================================
% mdlTerminate
% Perform any end of simulation tasks.
%=============================================================
%
function sys=mdlTerminate(t,x,u)

sys = [];

% end mdlTerminate
```

A.3 Generation of reference signal

Listing from source file `LrefwInital.m`

```
%Caclulates the referance signal for the simplifzed
    crane model and
%stets the model parmeters. It's also set the intial
    condtions of the crane
%model

close all;

%Calculate referance signal
T=1.0;
Tstop = T; %only used for the referance gerantion in
    simulink
r = 0.35; %final value
xstart = 0.455;


% Constrainst for 9 order ploynomial

% coefficiants for f(t) and derivitivs at time 0
f0 = [0 0 0 0 0 0 0 0 0 1];
f0d1 = [0 0 0 0 0 0 0 0 1 0];
f0d2 = [0 0 0 0 0 0 0 2 0 0];
f0d3 = [0 0 0 0 0 0 3*2 0 0 0];
f0d4 = [0 0 0 0 0 4*3*2 0 0 0 0];

% coefficiants for f(t) and derivitivs at time T
fT = [power(T,9) power(T,8) power(T,7) power(T,6)
    power(T,5) ...
     power(T,4) power(T,3) power(T,2) power(T,1) 1];

fTd1 = [9*power(T,8) 8*power(T,7) 7*power(T,6) 6*power
    (T,5) 5*power(T,4) ...
     4*power(T,3) 3*power(T,2) 2*power(T,1) 1 0];

fTd2 = [9*8*power(T,7) 8*7*power(T,6) 7*6*power(T,5)
    6*5*power(T,4) ...
```

```
    5*4*power(T,3) 4*3*power(T,2) 3*2*power(T,1) 2 0
        0];

fTd3 = [9*8*7*power(T,6) 8*7*6*power(T,5) 7*6*5*power(
    T,4) 6*5*4*power(T,3) ...
     5*4*3*power(T,2) 4*3*2*power(T,1) 3*2 0 0 0];

fTd4 = [9*8*7*6*power(T,5) 8*7*6*5*power(T,4) 7*6*5*4*
    power(T,3) ...
     6*5*4*3*power(T,2) 5*4*3*2*power(T,1) 4*3*2 0 0 0
        0];


A = [f0;f0d1;f0d2;f0d3;f0d4;fT;fTd1;fTd2;fTd3;fTd4];
B = [xstart 0 0 0 0 r 0 0 0 0 ]';

%Ploynomial coeffciants
ycoeff = mldivide(A,B);
ycoeff = ycoeff';

%calculate coefficiant for derivitives
yd1coeff = [9 8 7 6 5 4 3 2 1 0].*ycoeff;
yd1coeff = yd1coeff(1:end-1); %one order less after
    taken derivitive
yd2coeff = [9*8 8*7 7*6 6*5 5*4 4*3 3*2 2 0 0].*ycoeff
    ;
yd2coeff = yd2coeff(1:end-2);
yd3coeff = [9*8*7 8*7*6 7*6*5 6*5*4 5*4*3 4*3*2 3*2 0
    0 0].*ycoeff;
yd3coeff = yd3coeff(1:end-3);
yd4coeff = [9*8*7*6 8*7*6*5 7*6*5*4 6*5*4*3 5*4*3*2
    4*3*2 0 0 0 0].*ycoeff;
yd4coeff = yd4coeff(1:end-4);


hslow = 0.001;
t = 0:hslow:T;
y = polyval(ycoeff,t);

% derivitives of y
```

```
yd1 = polyval(yd1coeff,t);
yd2 = polyval(yd2coeff,t);
yd3 = polyval(yd3coeff,t);
yd4 = polyval(yd4coeff,t);


%final value
yfinal = y(end);
yd1final = yd1(end);
yd2final = yd2(end);
yd3final = yd3(end);
yd4final = yd4(end);


%plot the derivitives of y
figure(1);
subplot(2,2,1);
plot(t,yd1);
title('yd1');
subplot(2,2,2);
plot(t,yd2);
title('yd2');
subplot(2,2,3);
plot(t,yd3);
title('yd3');
subplot(2,2,4);
plot(t,yd4);
title('yd4');

%System constants

mL = 0.00003;
M = 0.277;
m = 0.005;
xa = 0.05;
ya = 0.57;
xb = 0.58;
yb = 2.62;
y1bar = 0.61;
l1 = 0.37; %old 0.22;
```

```
y1 = 0.61;
g = 9.81;

% System varibales
x = 0;
xd1 = 0;
xd1 = 0;
xd2 = 0;
xd3 = 0;
xd4 = 0;

%x1 = 0.455;
x1d1 = 0; %x1 dot
x1d2 = 0; %x1 dot dot

lamda1 = 0;
lamda2 = 0;

L1 = 0;
L1d1 = 0; %l1 dot
L1d2 = 0; %l2 dot dot



T1 = zeros(1,length(y));

%Flatness calculation
for i=1:length(y)
    x = y(i);

    %calulate derivitives
    xd1 = yd1(i);
    xd2 = yd2(i);
    xd3 = yd3(i);
    xd4 = yd4(i);


    %caltulate the x1

    x1 = l1*xd2/g+x;
```

```
    x1d1 = l1*xd3/g+xd1;
    x1d2 = l1*xd4/g+xd2;


    L1 = sqrt((x1-xa)*(x1-xa)+(y1bar-ya)*(y1bar-ya));
    L1d1 = (x1-xa)*x1d1/L1;
    L1d2 = (-L1d1*L1d1 + x1d1*x1d1 + (x1-xa)*x1d2)/L1;


    A = [2*(x1-xb)   2*(x1-xa);2*(y1bar-yb) 2*(y1bar-ya
        )];
    B = [(m*x1d2+g*M*(x1-x)/l1) g*(M+m)]';

    lambdas = mldivide(A,B);
    lambda2 = lambdas(2);

    lambdaEq(i,:) = lambdas';


    lambda1 = (2*(y1bar-ya)*(m*x1d2 +g*M*(x1-x)/l1) -
        2*(x1-xa)*(m+M)*g)/(4*(x1-xb)*(y1bar-ya)...
          -4*(y1-yb)*(x1-xa));
    lambda2c = (-2*(y1bar-yb)*(m*x1d2 +g*M*(x1-x)/l1)
        + 2*(x1-xb)*(m+M)*g)/(4*(x1-xb)*(y1bar-ya)...
          -4*(y1-yb)*(x1-xa));


    lambdaCode(i,:) = [lambda1 lambda2c];

    T1(1,i) = mL*L1d2 + 2*lambda2*L1;

    %save inital conditions for flatness
    if i==1
        FlatInit = [T1(1,1) L1]
        x0 = [xstart 0 xstart 0 T1(1,1)]; %initial
            conditions for crane
        L01 = L1;
    end
end
```

```
Tinit = T1(1,1); %intial value of the force in the
    model


%plot the force and refance trajectory given from
    flatness
figure(2);
subplot(2,1,1);
plot(t,T1);
title('T');

subplot(2,1,2);
plot(t,y);
title('ref');

%to compare different calculation method of the
    lambdas
figure(3);
subplot(2,1,1);
plot(t',lambdaEq(:,1),t',lambdaCode(:,1));
title('lambda 1');
subplot(2,1,2);
plot(t',lambdaEq(:,2),t',lambdaCode(:,2));
title('lambda 2');
```

Appendix B

Source code of implemented controller

B.1 Control algorithm

Listing of the control algorithm in source file `CommandChris.c`

```
// Full controll structure with feed forward

counter++; //update counter every time the thread runs
Consigne3=counter; // only used for plotting purpose

// run position and feed forward controller at every
   20 ms
if (counter%20==0){
        time = counter*h;

        if(counter>1000) // reference trajectory only
           defined for 1 s
             time = 1.0;

        uref = -(Lref(time) - L01); // get length
           reference of cable at time h*counter

        // time shift uref
        uref_4 = uref_3;
        uref_3 = uref_2;
        uref_2 = uref_1;
        uref_1 = uref;
```

```
        forceRef = Fref(time); // get the reference
            force in cable

        // update old forceRef
        forceRef_4 = forceRef_3;
        forceRef_3 = forceRef_2;
        forceRef_2 = forceRef_1;
        forceRef_1 = forceRef;


        // Position control
        epos = uref-PositionMoteur4;
        uvref = 0.2357*uvref_1 + 46.63*epos -31.68*
            epos_1;

        //feed forward part
        uvref_ff = -a3ff*ff_1 -a2ff*ff_2 -a1ff*ff_3 -
            a0ff*ff_4 +b4ff*forceRef +b3ff*forceRef_1 +
            b2ff*forceRef_2 + b1ff*forceRef_3 +b0ff*
            forceRef_4;

        // update old feed forward output
        ff_4 = ff_3;
        ff_3 = ff_2;
        ff_2 = ff_1;
        ff_1 = uvref_ff;

        // add output of feed forward and position
            control
        uvref = uvref +uvref_ff;
        epos_1 = epos;
        uvref_1 = uvref;

}


Consigne2 = forceRef; // only used for plotting
    purpose
Consigne3 = uref;// only used for plotting purpose
```

```
//velocity control;
evel = uvref-VitesseMoteur4;
u = 0.9929*u_1+0.01851*evel_1;
u_1 = u;
evel_1 = evel;

//Saturation
if(u > 5.0){
        u = 5.0;
}
else if (u<-5.0){
        u = -5.0;
}

Consigne4 = u; // Consigne4 is feed to motor
```

B.2 Flatness and reference generation

Listing from source file `LrefwInital.m`

```
#include <math.h>
#include "fp.h"



// coefficients for referance trajectory and it's
   derivitives
static double xcoeff[10] =
   {0.455,0,0,0,0,-13.23,44.1,-56.7,33.075,-7.35};
static double xd1coeff[9] =
   {0,0,0,0,-66.15,264.6,-396.9,264.6,-66.15};
static double xd2coeff[8] =
   {0,0,0,-264.6,1323,-2381.4,1852.2,-529.2};
static double xd3coeff[7] =
   {0,0,-793.8,5292,-11907,11113,-3704.4};
static double xd4coeff[6] =
   {0,-1587.6,15876,-47628,55566,-22226};
```

```
// model parameters
static double mL = 0.00001;
static double M = 0.5;
static double m = 0.005;
static double xa = 0.05;
static double ya = 0.57;
static double xb = 0.58;
static double yb = 2.62;
static double y1bar = 0.61;
static double l1 = 0.37; //0.22;
static double y1 = 0.61;
static double g = 9.81;


//static double L01 = 0.4; //cable lengh a initial
    posistion
// model variables


// gives the referance lenght of cable as a divations
    from initial posistion to motor
double Lref(double time);
double Lref(double time)
{


        double x = 0.0;
        double xd1 = 0.0; // first derivitive
        double xd2 = 0.0;
        double xd3 = 0.0;
        double xd4 = 0.0;

        double x1 = 0.0;
        double x1d1 = 0.0;
        double x1d2 = 0.0;

        double lambda1 = 0.0;
        double lambda2 = 0.0;
```

```
        double L1 = 0.0; // lenght of cabel contected
            to motor
        double L1d1 = 0.0;
        double L1d2 = 0.0;

        int i;



        // calculate current value of x and it's
            derivitives
        for (i=0;i<=9;i++)
                x = x + xcoeff[i]*pow(time,i);

        for (i=0;i<=8;i++)
                xd1 = xd1 + xd1coeff[i]*pow(time,i);

        for (i=0;i<=7;i++)
                xd2 = xd2 + xd2coeff[i]*pow(time,i);

        for (i=0;i<=6;i++)
                xd3 = xd3 + xd3coeff[i]*pow(time,i);

        for (i=0;i<=5;i++)
                xd4 = xd4 + xd4coeff[i]*pow(time,i);

        //calculate x1 and it's derivitives
        x1 = l1*xd2/g + x;
        x1d1 = l1*xd3/g + xd1;
        x1d2 = l1*xd4/g + xd2;

        //calculate L1
        L1 = sqrt((x1-xa)*(x1-xa) + (y1bar-ya)*(y1bar-
            ya));

        return L1;
}

// gives the expected force in cable at the given
    referance posistion
```

```
double Fref(double time);
double Fref(double time){


        double x = 0.0;
        double xd1 = 0.0; // first derivitive
        double xd2 = 0.0;
        double xd3 = 0.0;
        double xd4 = 0.0;

        double x1 = 0.0;
        double x1d1 = 0.0;
        double x1d2 = 0.0;

//      double lambda1 = 0.0;
        double lambda2 = 0.0;

        double L1 = 0.0; // lenght of cabel contected
            to motor
        double L1d1 = 0.0;
        double L1d2 = 0.0;

        int i;
        // calculate current value of x and it's
            derivitives
        for (i=0;i<=9;i++)
                x = x + xcoeff[i]*pow(time,i);

        for (i=0;i<=8;i++)
                xd1 = xd1 + xd1coeff[i]*pow(time,i);

        for (i=0;i<=7;i++)
                xd2 = xd2 + xd2coeff[i]*pow(time,i);

        for (i=0;i<=6;i++)
                xd3 = xd3 + xd3coeff[i]*pow(time,i);

        for (i=0;i<=5;i++)
                xd4 = xd4 + xd4coeff[i]*pow(time,i);
```

```
        //calculate x1 and it's derivitives
        x1 = l1*xd2/g + x;
        x1d1 = l1*xd3/g + xd1;
        x1d2 = l1*xd4/g + xd2;

        //calculate L1
        L1 = sqrt((x1-xa)*(x1-xa) + (y1bar-ya)*(y1bar-
            ya));
        L1d1 = (x1-xa)*x1d1/L1;
        L1d2 = (-L1d1*L1d1 + x1d1*x1d1 + (x1-xa)*x1d2)
            /L1;

        //lambda1 = (2*(y1bar-ya)*(m*x1d2 + g*M*(x1-x)
            /l1) - 2*(x1-xa)*(m+M)*g)/(4*(x1-xb)*(y1bar
            -ya) - 4*(y1-yb)*(x1-xa));
        lambda2 = (-2*(y1bar-yb)*(m*x1d2 + g*M*(x1-x)/
            l1) + 2*(x1-xb)*(m+M)*g)/(4*(x1-xb)*(y1bar-
            ya) - 4*(y1-yb)*(x1-xa));

        return (mL*L1d2 + 2*lambda2*L1);
}
```