CSD Labs Technical Report
Control of a Rolling Mill

Nathan Dwek – Thomas Lapauw

December 9, 2015

Contents

Chapter 1

Introduction

1.1 Description of the plant

The goal of this project is to control the rolling of a metallic strip using a rolling mill. The plant is composed of two DC motor driven single rolls which unwind and wind up the sheet of metal. Between those two, the strip passes through a pair of rolls driven by a third DC motor, in order to have its thickness reduced and made uniform.

The actuators of the plant are the three DC motors, which are controlled through their armature current. The speed of those motors are measured using three velocity sensors. There are also two traction sensors to measure the tension in the strip left and right of the middle pair of rolls. Finally, a thickness sensor measures the thickness of the metallic strip on both sides as well.

The first requirement is to control the traction of the metallic strip. When this is achieved, a more advanced requirement is to control the thickness of the metallic strip.

1.2 Broad design of the controller

The plant will be controlled using a numerical controller implemented in matlab. To set this up, eight ADC and two DAC ports are available, along with second order Butterworth filters with various ω_c and an analog computer.

Intuitively, we know that the traction of the strip between two rolls will probably mostly depend on the difference of speed between the rolls. Similarly, we also know that the thickness reduction at the middle pair of rolls probably mostly depends on the difference between the traction of the strip that is fed into the pair of rolls and the traction of the sheet that is pulled out of it. With this in mind, we propose to control this process using a cascade controller.

First, we will control the speed of the DC motors using the current as input. Then, we will control the traction of the metallic strip using the speed of the motors as input. Finally, we will control the thickness of the metallic sheet using the tractions of the strips as input.

Moreover, since we know that the traction will depend on a *difference* of speeds, we also propose to simplify this scheme by modulating the speed of only one motor during the operation. One motor is designated as “master”: it will be controlled with a constant reference with the only goal of spinning at the setpoint speed as steadily as possible. The other is designated as “slave”: its reference will vary during operation in order to control the traction of the strip and it should have adequate transient response properties.

1.3 Structure of this report

First, in chapter 2, we will describe the experimental setup which was used throughout this project. This report will then follow the chronological order of the work that was done during the labs. We used a bottom up approach where the inner loops are first implemented in order to then design the outer loops.

As stated before, to simplify the controller, only one motor – the slave – is controlled with a dynamic reference while the other – the master – is kept at a speed which should be as steady as possible. In chapter 3, the dynamics of the master motor are identified and a controller is designed to achieve zero steady state error and perturbation rejection. Then, in chapter 4, the dynamics of the slave motor are identified and a controller is designed to achieve reasonably fast reference tracking.

In chapter 5, the relation between both motors’ speeds and the traction of the metallic strip is modelled and identified, and a controller is designed to stabilize the system and reduce the steady state error.

Finally, chapter 6 describes some considerations in controlling the thickness when rolling the metal strip. Since this objective is not attainable in the course of the lab sessions due to other constraints and complexity this will be done theoretically.

In appendix A the matlab code for the controller can be found.

Chapter 2

Description of the Experimental Setup

2.1 General Wiring

The setup is wired as described in figure 2.1. We use the maximum recom-

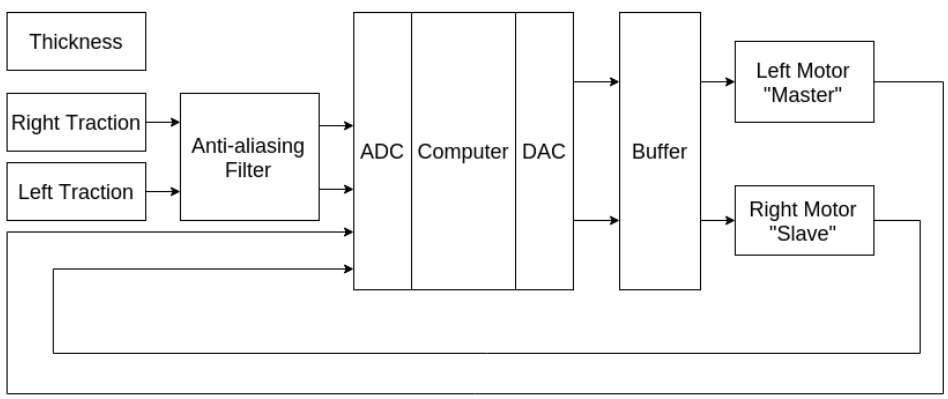

Figure 2.1: Wiring for the control of the rolling mill

mended sampling frequency $f_s = 100\,\mathrm{Hz}$, which is a common value for the control of a DC motor.

2.1.1 Anti-aliasing filtering

To be perfectly rigorous, we should filter every input before sampling it. However, we only have a limited amount of single signal, second order Butterworth filters at our disposition, each with a different f_c. With the chosen f_s, those constraints on the available f_c, and the fact that second order Butterworth filters have a large transition band, only two of the available f_c values seem actually useful:

- $f_c = 40\,\text{Hz}$, which provides 37.1 % attenuation at 50 Hz and 62.9 % rejection at 100 Hz.
- $f_c = 20\,\text{Hz}$, which provides 62.9 % attenuation at 50 Hz and 80.4 % rejection at 100 Hz.

The other available filters either do not actually prevent aliasing, or have a too high passband attenuation. Even the two most adequate filters have a non ideal passband gain, and thus introduce a significant delay in the feedback loop. For this reason we choose not to use them in the inner loop, which should be fast, but rather in the outer loop. This is why only the two traction measurements are filtered in figure 2.1.

Chapter 3

Control of the Master Motor

In this chapter, we will identify and design a controller for the master motor of the rolling mill. During the operation, this motor will be controlled with a constant reference and it should keep its velocity as steady as possible.

3.1 Master motor

The master motor is the one that pulls the strip and winds it up on a spool. The left motor was chosen as master since its setpoint velocity is higher than that of the right motor. This is necessary since when the strip is compressed to reduce its thickness it extends, which means the speed of the winding motor should be higher than the feeding motor. Figure 3.1 shows the static characteristic of the left motor and table 3.1 shows the currents and speed for different operating points.

Armature current [A]	Angular velocity [RPM]
5.4	530.7
5.7	649.8

Table 3.1: Operating points of the left motor

3.2 Identification of the Transfer Function

The transfer function of the left motor is identified with a least square approximation using matlab. To do this, the measured velocity is sampled for an input step between the setpoints, since that is the region where we want

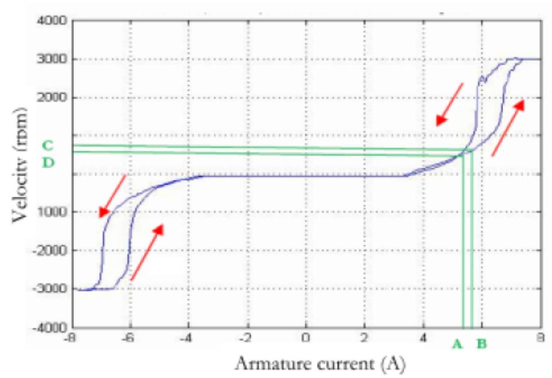

Figure 3.1: Static characteristic of the left motor

to linearise the behaviour of the motor. We choose to fit the system with a first order transfer function which is common for a DC motor with big inertial load.

Figure 3.2 shows the measured and fitted step responses. We see that the system is indeed heavily dominated by a single slow real pole, and that it is well approximated by transfer function 3.1.

$$LM(s) = \frac{5.398}{3.642S + 1} \tag{3.1}$$

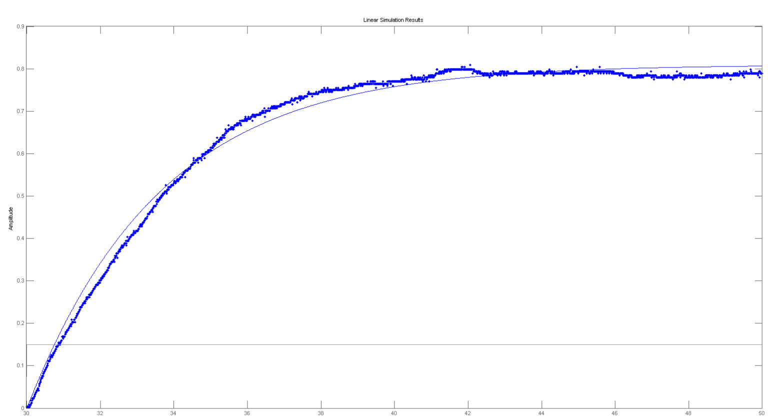

Figure 3.2: Sampled and fitted step response of the left motor

3.3 Design of a PI Controller

The master motor is controlled with a constant reference and should keep its velocity as steady as possible despite possible disturbance. For this reason, we choose to control it with a PI controller, which should provide asymptotic stability, zero steady state error, and perturbation rejection.

We use the root locus method to tune the PI controller. First we choose $\frac{K_I}{K_P}$ so that the zero of the controller cancels the pole of the motor. This theoretically ensures that the closed loop has a single negative real pole that can then be arbitrarily placed by adjusting K_P. In the following, we detail the calculations used to determine the gains using this method.

First, the transfer functions of the left motor and the PI controller are transformed to make the poles and zeroes immediately visible.

$$\begin{aligned} LM(s) &= \frac{5.398}{3.642s + 1} \\ &= \frac{1.482}{s + 0.274} \end{aligned}$$

$$\begin{aligned} PI(s) &= K_p + \frac{K_i}{s} \\ &= K_p \cdot \frac{s + \frac{K_i}{K_p}}{s} \end{aligned}$$

To cancel the pole of the motor, we hence use $\frac{K_i}{K_p} = 0.294$.

We then choose K_P using a root locus. The open loop is given by:

$$\begin{aligned} OL(s) &= \frac{1.482}{s + 0.274} \cdot \frac{s + 0.274}{s} \\ &= \frac{1.482}{s} \end{aligned}$$

Figure 3.3 shows that $K_P > 0$ can theoretically be chosen arbitrarily. Non linearities, higher order effects and actuator saturation will determine its ideal value.

Figures 3.4 – 3.6 show the result of simulink simulations for several values of K_P. We see that actuator saturation will probably not be the deciding constraint[1]. We thus need to implement the controller to be able to experimentally tune its gain.

[1]The blue curve represents the small signal input on each of the plots. The setpoint value is 2.7 V and the input range is [−10 V - 10 V].

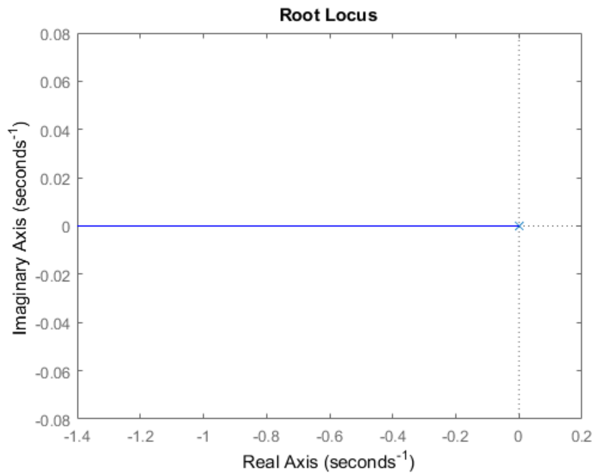

Figure 3.3: Root locus plot for the left motor with a PI controller

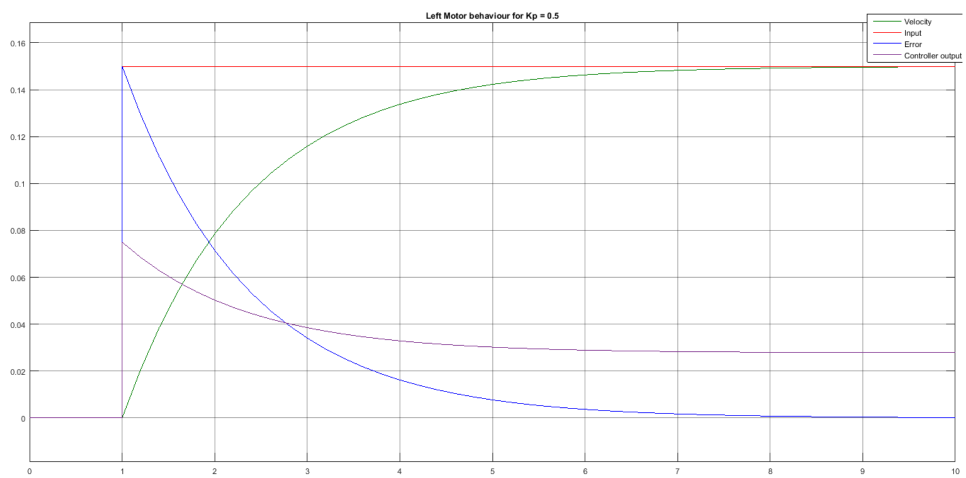

Figure 3.4: Simulink simulation of the left motor behaviour for $K_p = 0.5$

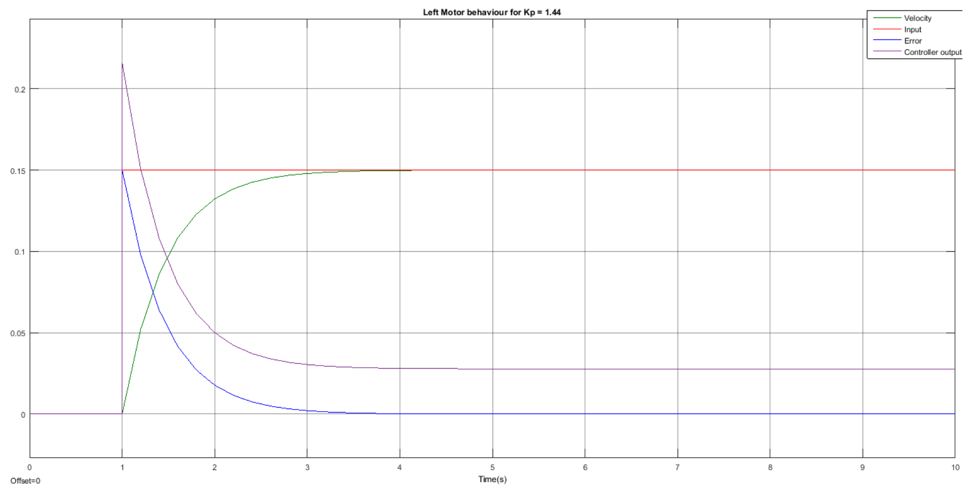

Figure 3.5: Simulink simulation of the left motor behaviour for $K_p = 1.44$

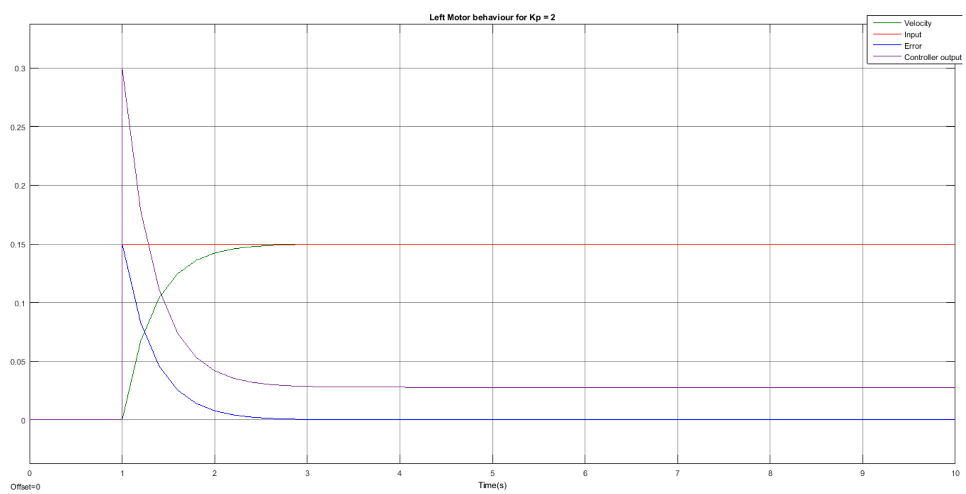

Figure 3.6: Simulink simulation of the left motor behaviour for $K_p = 2$

3.4 Tuning of the PI Controller

Figures 3.7 – 3.9 show experimental runs of the master motor with different values of K_P. Based on these measurements $K_P = 2$ was chosen. $K_P = 0.5$ has a longer settling time with a larger overshoot than $K_p = 2$, and $K_P = 0.5$ has a longer, higher overshoot. Since $\frac{K_I}{K_P} = 0.294$, $K_I = 0.588$. None of the simulations have overshoot while all the practical experiments do. This is because the transfer function of the motor was reduced to a first order system while in practice it behaves as a non linear system of higher order.

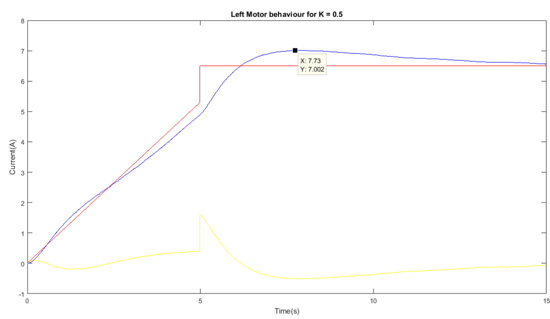

Figure 3.7: Response of the left motor behaviour for $K_p = 0.5$ to the red curve as input.

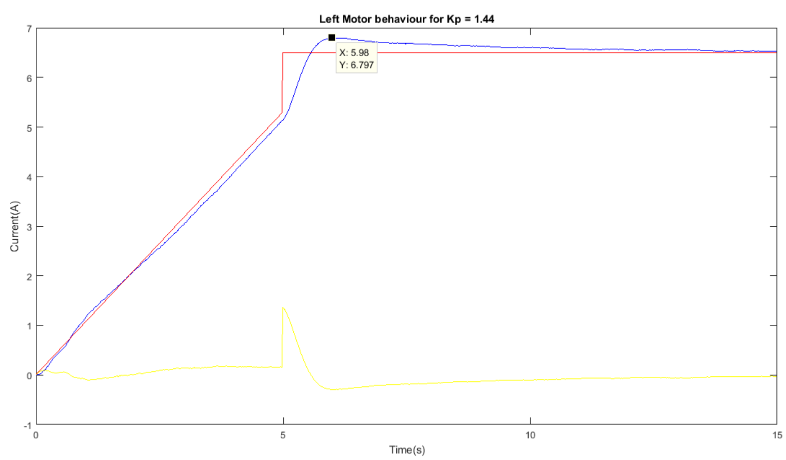

Figure 3.8: Response of the left motor behaviour for $K_p = 1.44$ to the red curve as input.

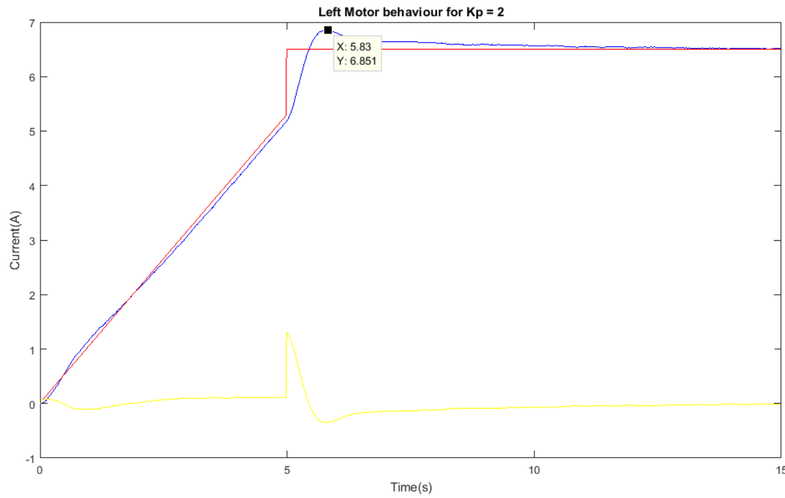

Figure 3.9: Response of the left motor behaviour for $K_p = 2$ to the red curve as input.

We observe that we indeed obtain zero steady state error, and that disturbances of order higher than 1 seem to not affect the master velocity too much, which is very good.

3.5 Conclusion

We decide to control the master motor with a PI controller and the following values:

$$K_P = 2$$
$$K_I = 0.588$$

This allows us to generate a steady, precise master speed around which we will then modulate the slave speed in order to control the traction of the metallic strip. In the next chapter, we will identify and design a controller for the slave motor, which will serve as an inner loop for the traction control.

Chapter 4

Control of the Slave Motor

In this chapter, we will identify and design a controller for the slave motor of the rolling mill. The process is analogous to the master motor, first the motor is identified around the operating point, then a controller is designed which is then simulated and experimentally tested. The reference of the slave motor is the output of the traction controller and the slave motor should thus have good dynamic reference tracking.

4.1 Slave Motor

The right motor is chosen as the slave motor, the reasoning behind this was already explained in chapter 3. Figure 4.1 shows the motor's static characteristic and table 4.1 shows the currents for different operating points.

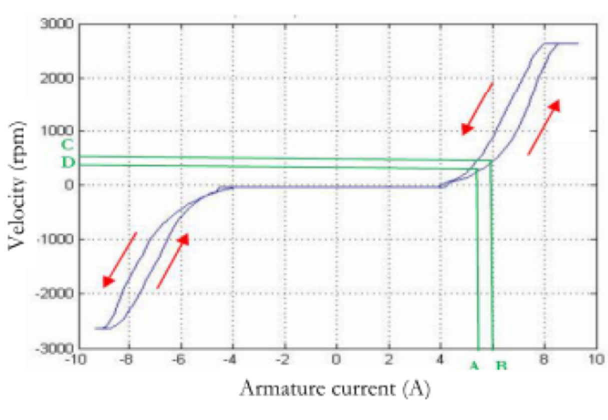

Figure 4.1: Static characteristic of the slave motor

Armature current [A]	Angular velocity [RPM]
5.4	244
5.7	370

Table 4.1: Operating points of the right motor

4.2 Identification of the Transfer Function

The slave motor is identified the same way the master motor was. Again, we decide to approximate with a first order system, for the same reasons. Figure 4.2 shows that transfer function 4.1 indeed fits the sample data very well.

$$RM(s) = \frac{7.128}{6.0665s + 1} \tag{4.1}$$

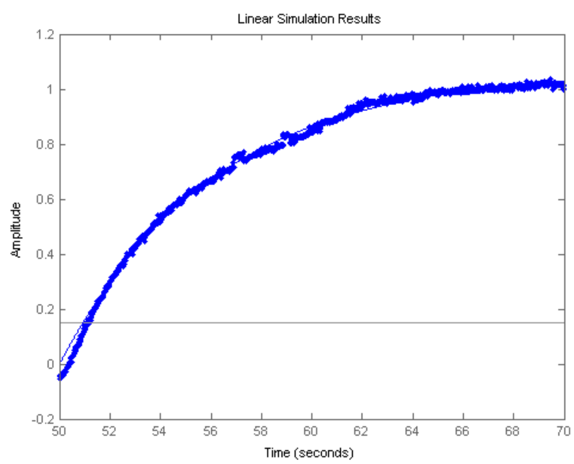

Figure 4.2: Sampled and fitted step response of the slave motor

4.3 Controller Design

The closed loop needs to be controllable and fast enough to be able to track the tension reference generated by the tension controller. To do this a proportional controller with gain K was chosen. This does not guarantee zero steady state error, but in the case of the slave loop, this is unimportant since it is part of a cascade controller. Furthermore, adding an integrator to the controller slows the response down, which deteriorates tracking.

Figure 4.3 shows a root locus plot for the right motor. Again, K can theoretically be chosen arbitrarily. It was chosen based on a couple simulations (figures 4.4, 4.5 and 4.6) and practical experiments for different gains (figures 4.7, 4.8 and 4.9). None of the simulations have overshoot while some of the practical experiments do (depending on K). This is because the transfer function of the motor was reduced to a first order system while in practice it is a non-linear system of higher order.

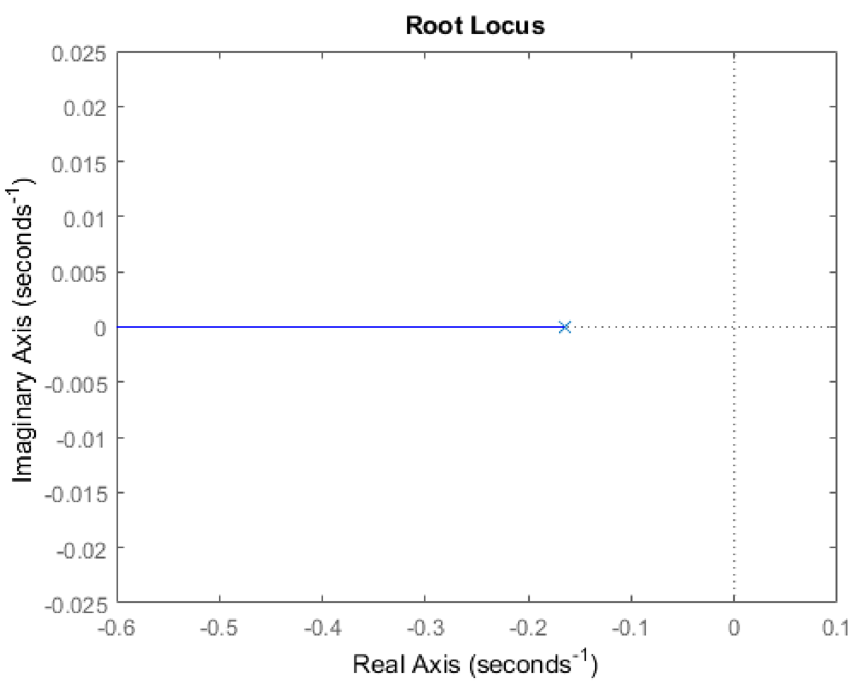

Figure 4.3: Root locus plot for the right motor

$K = 3$ is chosen since higher gains introduce the risk of putting the actuator in saturation, or at least in a heavily non linear zone.

4.4 Conclusions

A proportional controller was chosen to obtain good reference properties. In a cascade architecture, the steady state error of intern signals is unimportant

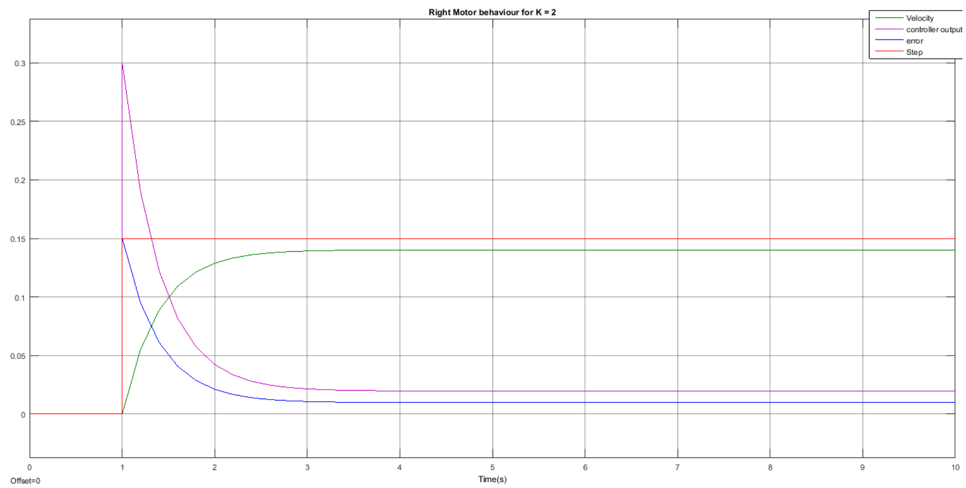

Figure 4.4: Simulink simulation of the right motor behaviour for $K = 2$

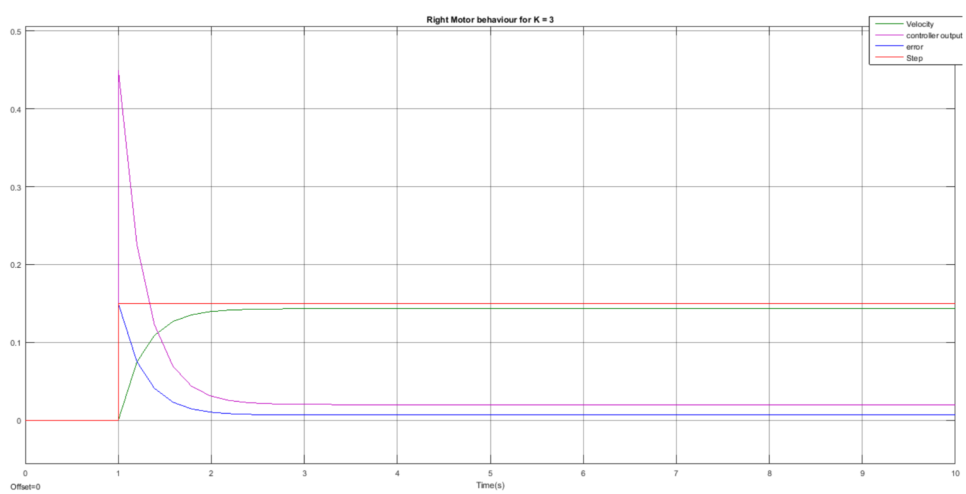

Figure 4.5: Simulink simulation of the right motor behaviour for $K = 3$

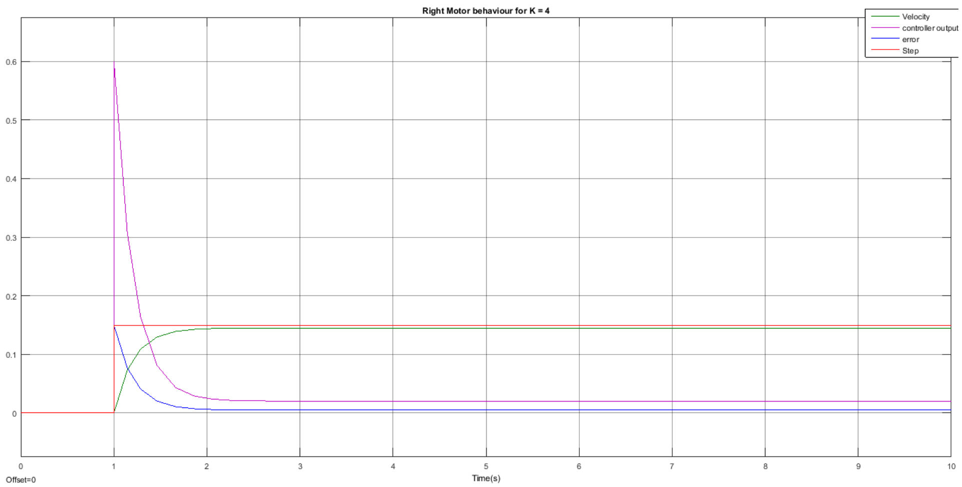

Figure 4.6: Simulink simulation of the right motor behaviour for $K = 4$

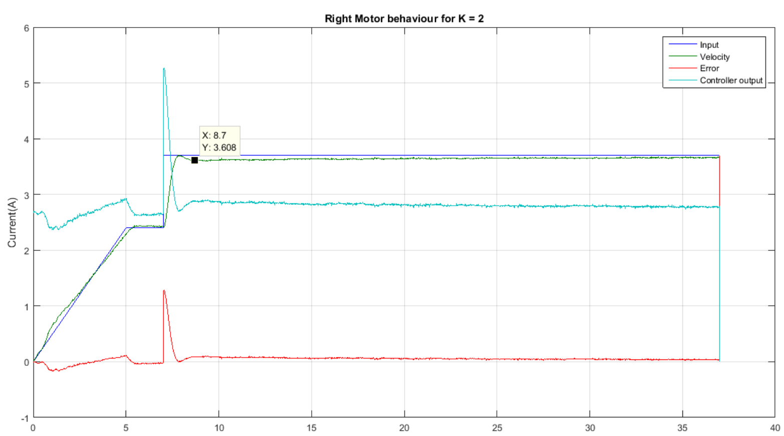

Figure 4.7: Response of the left motor behaviour for $K = 2$ to the blue curve as input.

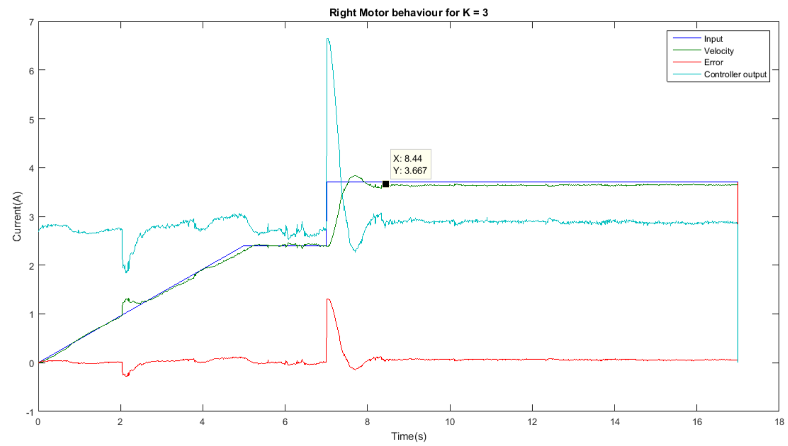

Figure 4.8: Response of the left motor behaviour for $K = 3$ to the blue curve as input.

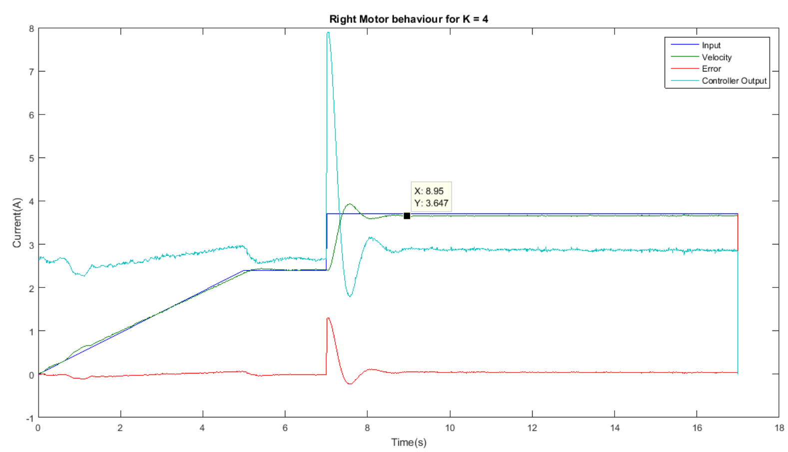

Figure 4.9: Response of the left motor behaviour for $K = 4$ to the blue curve as input.

(but those then need to be monitored along with their reference), so we do not need to introduce an integrator in the controller, which would slow down the response. The outer loop will adjust the slave speed reference in order to obtain zero steady state error on the traction.

As we did for the master motor, the behaviour was identified using a step between two operating points. Using this approximated, transfer function, the controller gain is tuned using simulations and then real-world experiments. Finally a gain of $K = 3$ is selected. We will now design the outer loop controller which will generate the slave speed reference. In order to do this, we will need a good estimation of the closed loop pole of the slave loop. This approximation can be found using a root locus, as we did to design the controller, but it is better to also have an experimental value, both to verify the design and to increase the accuracy of the rest of the calculations. Using the same method as we did in the beginning of this chapter, we find that the closed loop transfer function of the slave motor can be approximated by 4.2

$$\begin{aligned} Slave(s) &= \frac{3.9442}{s + 4.118} \\ &= \frac{0.9577}{0.2428s + 1} \end{aligned} \tag{4.2}$$

Chapter 5

Modelling and Control of the Traction

5.1 Modelling of the Traction of the Metallic Strip

As presented in the introduction, we know from a physical intuition that the traction in the metallic strip depends mainly on the difference of speed between the rolls, rather than on each of the speeds individually. For this reason, we only have to determine $Trac(s)$ as described in figure 5.1, where ω_i is the speed of a motor, and f is the traction of the metallic strip[1].

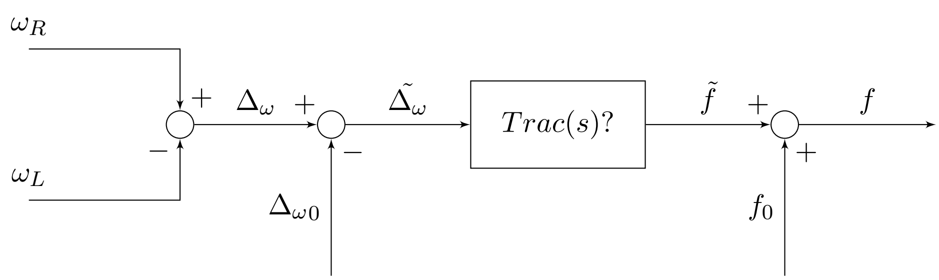

Figure 5.1: Simple gray-box model of the traction of the metallic strip

Furthermore, we also know that $Trac(s)$ should contain a close-to-perfect integrator. Indeed, if we increase Δ_ω slightly from the setpoint $\Delta_{\omega 0}$, we ex-

[1] Since we do not control the middle pair of rolls, we are not able to control the left and right traction separately. For this reason, in the following, we only care about controlling the right traction, which is denoted f but sometimes f_R in the matlab figures. With this setup, the left traction has roughly the same evolution as the right traction, multiplied by a coefficient slightly lower than 1.

pect the tension in the metallic strip to rise indefinitely until breakage. This is also confirmed by the experience: we observe that the system's response to a real world pulse is really close to a step, as showed in figure 5.2. Fi-

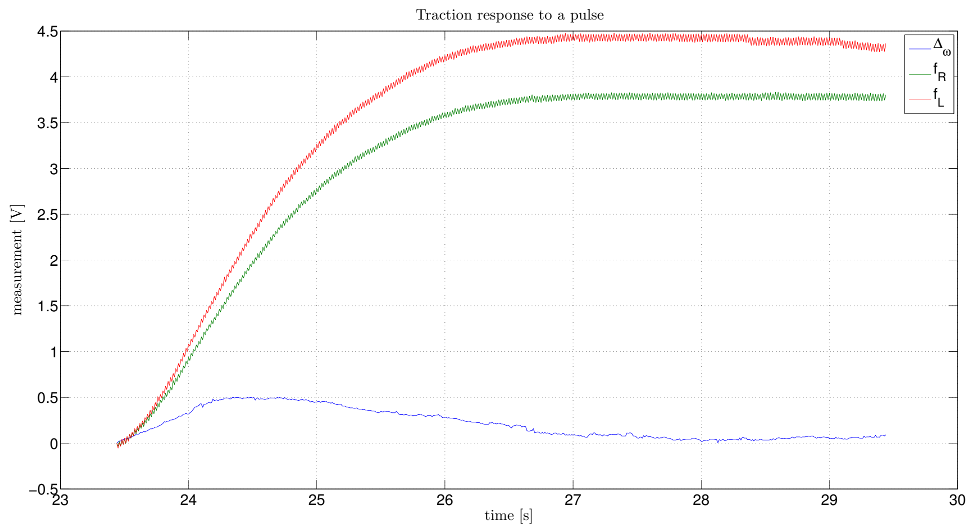

Figure 5.2: Traction response to a real world pulse

nally, this is supported by a second order numerical approximation of the dynamics, which always yields a pole that is very close to zero.

However, this leads the rest of the optimisation problem to be badly conditioned. This means that the second pole is not reliably placed, and that the final result does not fit the real response accurately. Moreover, the obtained transfer function, with a very large K and p_2, takes a long time to simulate with simulink.

To solve this, we refine our gray-box approximation by artificially introducing an integrator in the system we want to identify. We then compute its response to $\tilde{\Delta_\omega}(t)$ and try to determine the rest of the dynamics based on this new input and the observed response, as shown in figure 5.3, where $\frac{1}{s} \cdot H(s) = Trac(s)$. In the figure, we also use the fact that only ω_L is supposed to contribute to $\tilde{\Delta_\omega}$, since the master velocity is chosen steady. $\tilde{\omega_R}$ is thus considered as a disturbance.

Fitting a simple first order transfer function to $H(s)$ is not easy because the dynamics between $\int_0^t \tilde{\omega_L}(t)$ and $f(t)$ is very fast, as shown in figure 5.4. This leads to very large poles which are not well approximated and difficult to simulate, as experienced previously.

To solve this, we try to fit $H(s)$ with a transfer function of the form

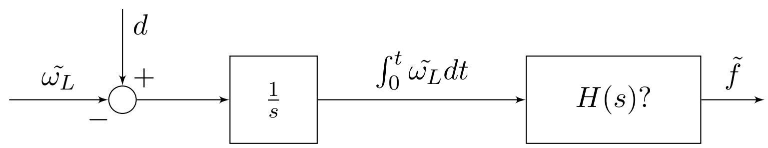

Figure 5.3: Gray-box model of the traction of the metallic strip

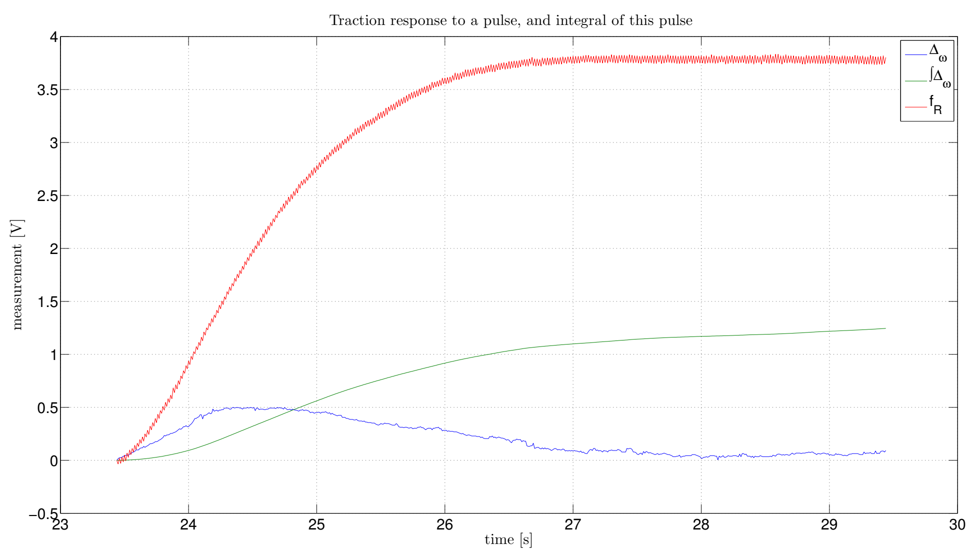

Figure 5.4: Right traction response to a pulse and output of the intermediate integrator

$K\frac{s-z_1}{s-p_2}$, because the introduction of a zero speeds up a step response, which would bring p_2 reasonably closer to the origin. The result is shown in figure 5.5, where we see that $Trac(s)$ as given in equation 5.1 seems to fit the experience very well.

$$\begin{aligned} H(s) &= 13.096 \cdot \frac{s+0.9221}{s+4.063} \\ Trac(s) &= 13.096 \cdot \frac{s+0.9221}{s(s+4.063)} \end{aligned} \tag{5.1}$$

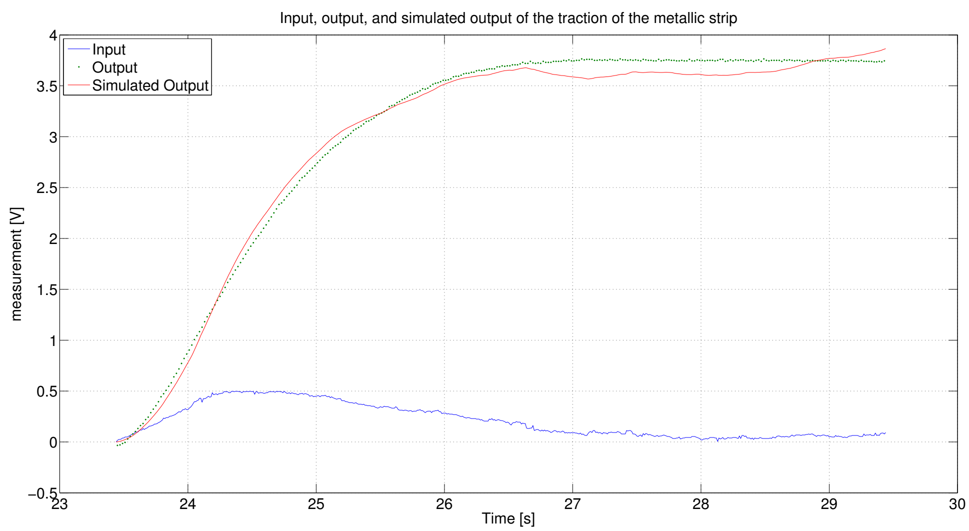

Figure 5.5: Input, output, and simulated output of the traction of the metallic strip

5.2 Control of the Traction

5.2.1 Simple P Controller

Since there already is an integrator in the system, and since we do not have specifications on the transient response, we first try to control the traction with a simple P controller, which should provide asymptotic stability and zero steady-state error. Figure 5.6 shows the root locus of the traction for this controller, with the approximation that the inner slave speed loop is fast enough to be considered perfectly transparent. The root locus shows

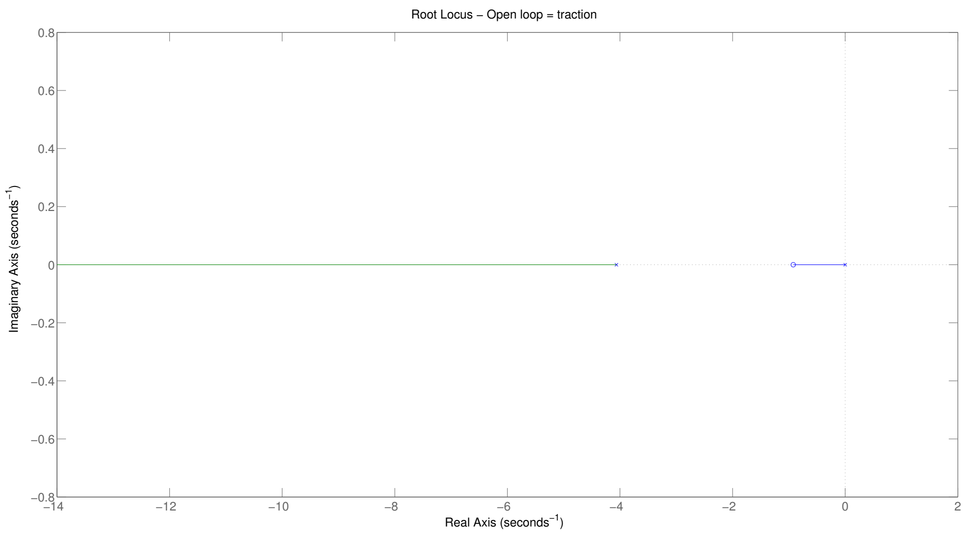

Figure 5.6: Root locus of the traction

that $k_P < 0$[1] can be theoretically chosen as high as desired while keeping the closed-loop poles on the real axis.

Figure 5.7 shows the closed-loop traction response to a constant reference, with an arbitrary gain of -1, as simulated by simulink. First, we observe that the closed-loop poles are not real, which is due to non linearities and higher order effects. More importantly, we see that the steady-state error is not cancelled.

Comparing the measured traction and the master speed, we observe that the initial ramp on the master motor is not compensated. In steady state, the master speed also creates an offset on the traction. This is because the right velocity is actually a disturbance, as was shown in figure 5.3. This disturbance is not rejected because the integrator is in the plant, and not in the controller.

To achieve perturbation rejection, an integrator should be added to the controller. However, this is not a good solution as is, because the open loop would then contain two integrators. This would greatly reduce the phase margin and thus make the closed loop system nearly unstable and slow its response down.

[1] The $\tilde{\omega_L}$ input is inverting, as shown in figure 5.3

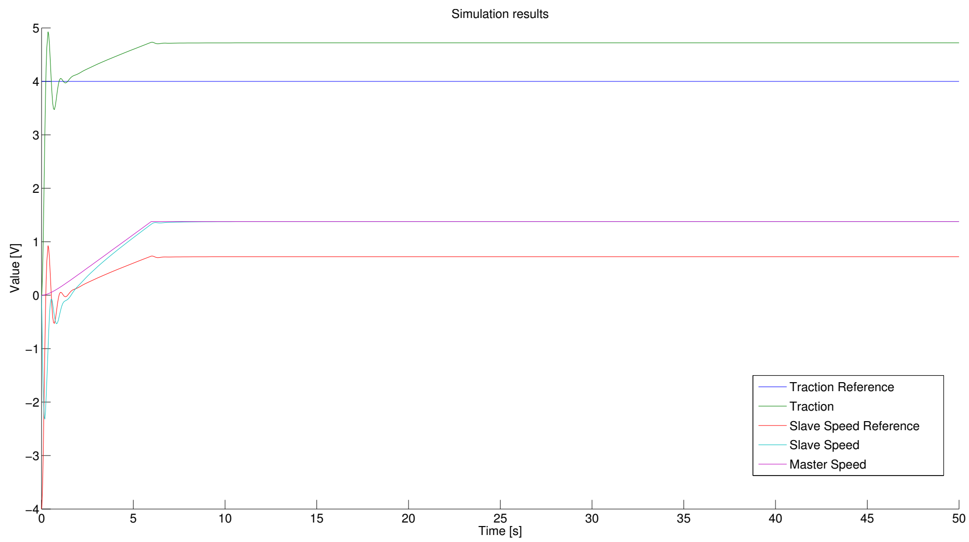

Figure 5.7: Simulation of the rolling mill operation with a P controller on the traction

5.2.2 Second Loop: PI Controller

Rather then adding an integrator to the existing controller, we can add a second external loop with a PI, as shown in figure 5.8 ,to achieve pertur-

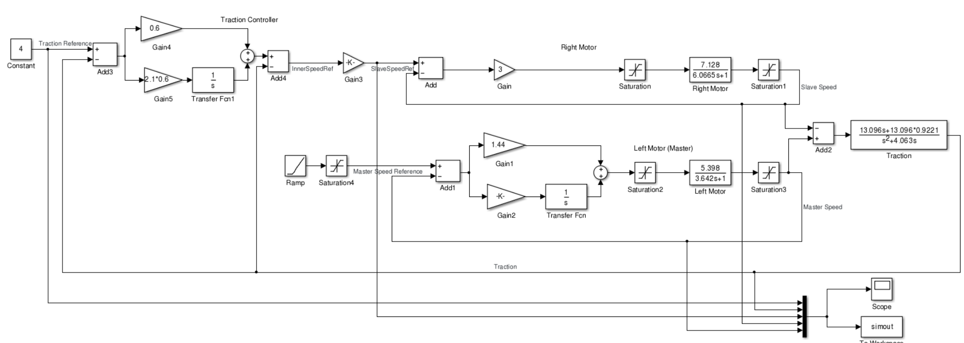

Figure 5.8: Final controller architecture

bation rejection while preserving the phase margin. In this subsection, the reasoning behind this second loop will be explained. In the following, we do not neglect the effect of the inner slave speed loop anymore. However, we do make the approximation that the traction is a essentially an integrator[1]. The plant that is controlled using an inner P and an outer PI is thus approximated by the series of the closed loop transfer function of the slave motor and an integrator – the transfer function of the traction.

With that in mind, the closed loop poles of the inner traction loop are given by figure 5.9 ,which shows the root locus of $Slave(s) \cdot Trac_{approx}(s)$.

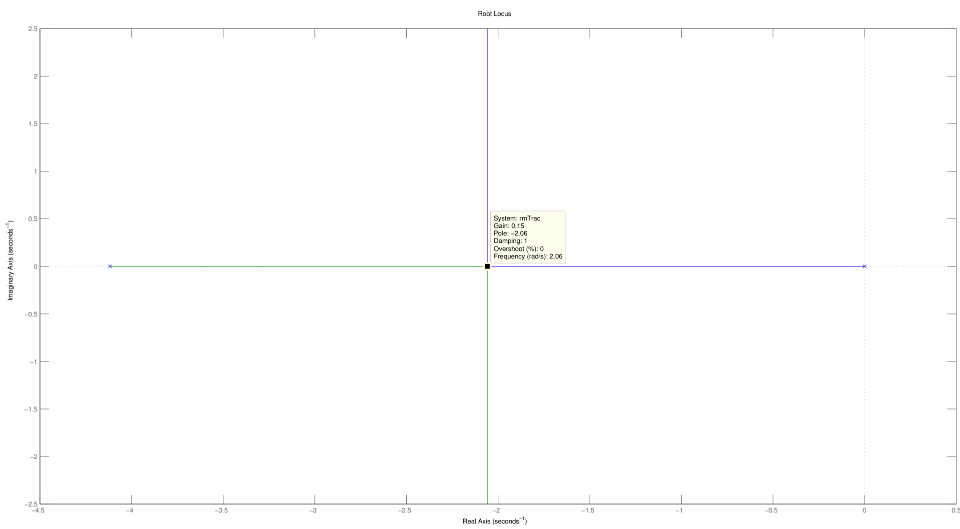

Figure 5.9: Root locus of the inner traction loop

We observe that using a P controller, it is possible to move the integrator pole into the LHP, which is the essence of feedback control. This is the reason why we can now afford to add a PI outer loop: for an adequate choice of $K_{p_{inner}}$, the plant, which is given by closed loop transfer function of the inner traction loop, can have two real, negative poles. This means that adding a PI to it will ensure a reasonable phase margin, as well as zero steady state error and perturbation rejection.

On figure 5.9 we choose $K_{p_{inner}} \lessapprox 0.15$ to obtain the fastest possible real poles for the inner loop. This yields a pair of real poles around -2. There is now left to determine the gains of the outer PI, given those inner loop poles to complete the controller design of our rolling mill.

To do so, we use the classical approach of cancelling the slowest pole in the plant with the zero from the PI, and then determining a first $K_{p_{outer}}$ with a root locus. The resulting plot is shown in figure 5.10. From the root locus, we can extract two reasonable first values: $K_{p_{outer}} \lessapprox 0.52$ would

[1]Neglecting the higher order zeroes and poles in $Trac(s)$ given in equation 5.1, we obtain $Trac_{approx}(s) = \frac{7.16}{s}$.

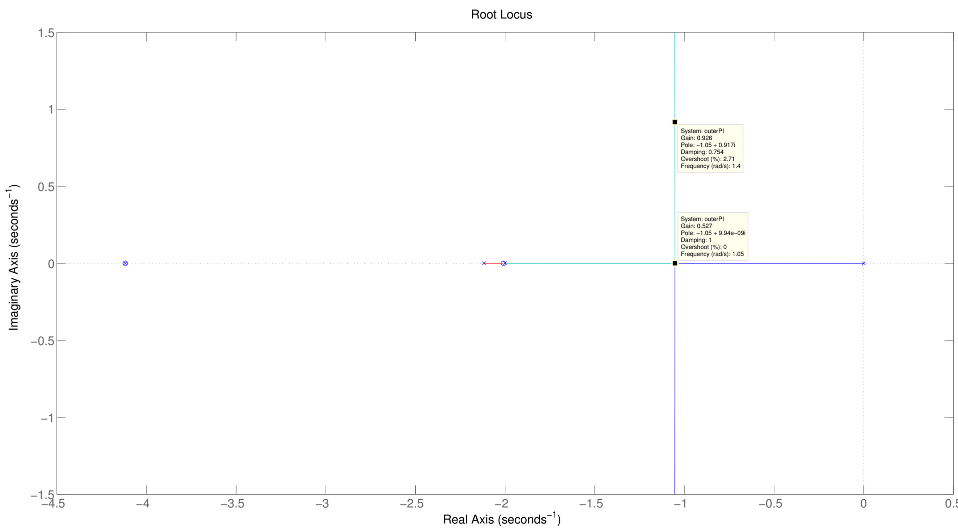

Figure 5.10: Root locus of the outer traction loop

provide the fastest real poles and $K_{p_{outer}} \approx 0.93$ would provide a damping of 0.7 – any value inbetween could also be a good first choice. K_i is then determined accordingly using $K_i = |z_{PI}| \cdot K_{P_{outer}}$. The root locus also shows us that if the poles of the inner loop are not perfectly known and z_{PI} cannot be placed perfectly, $p_{inner_{fast}} < z_{PI} < p_{inner_{slow}}$ is better than $p_{inner_{fast}} < p_{inner_{slow}} < z_{PI}$, because the latter pole-zero configuration would make the closed loop poles leave the real axis, as shown in figure 5.11.

With this controller, we should be able to move the integrating pole from the plant to the controller. The total closed loop should thus be stable and have zero steady-state error, perturbation rejection and a reasonable phase margin. In the following section, we will verify this design using simulink simulations and real world experiences and tune the gains if necessary.

5.3 Tuning of the PI Controller

Figure 5.12 shows the the simulated output of the rolling mill with the following gains:

$$
\begin{aligned}
K_{p_{inner}} &= -0.14 \\
K_{p_{outer}} &= 0.6 \\
K_i &= 1.26
\end{aligned}
$$

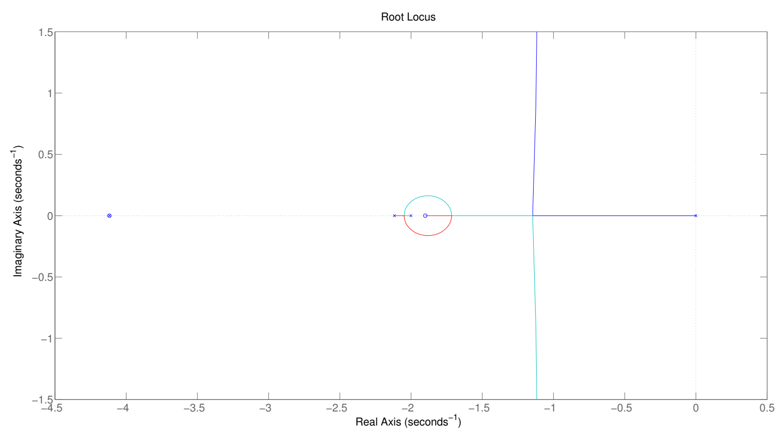

Figure 5.11: Root locus of the outer traction loop with a bad pole-zero configuration

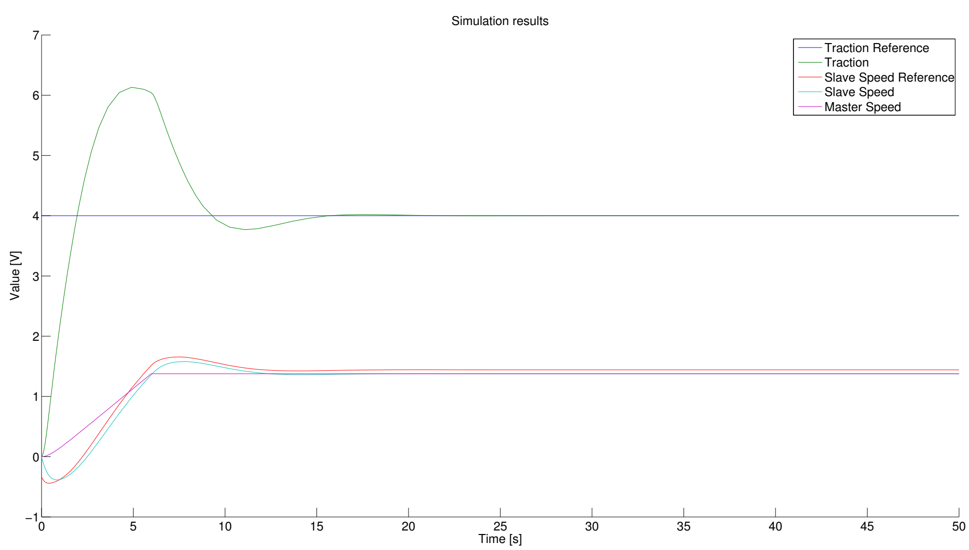

Figure 5.12: Simulation of the rolling mill operation with a P - PI controller on the traction. $K_{p_{outer}} = 0.6$.

We see that all the desired properties are achieved. However, the effect of the master speed ramp is still present, which is normal since a single integrator in the controller is able to reject perturbation of order ≤ 1. This is supported by the experience, as shown in figure 5.13.

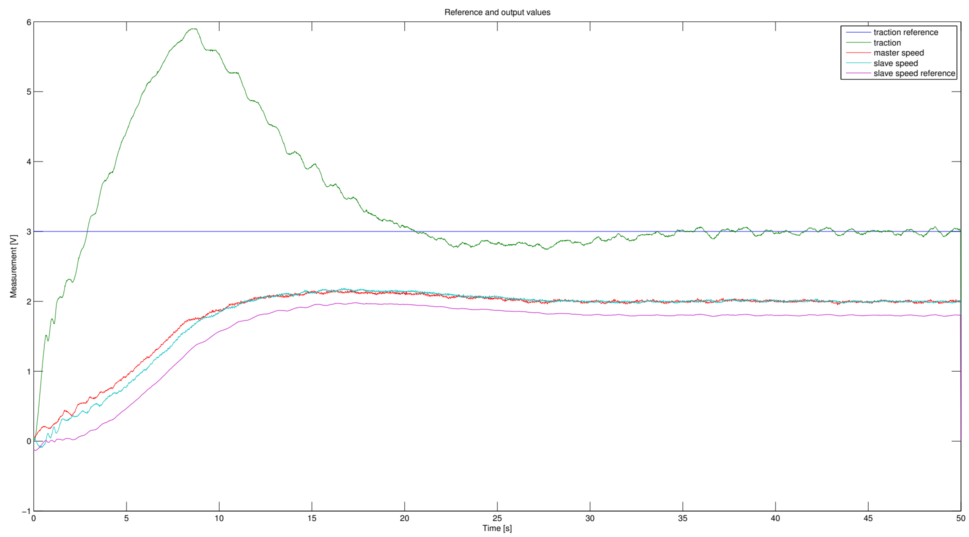

Figure 5.13: Operation of the rolling mill. $K_{p_{outer}} = 0.327$.

As a result, the controller should be tuned first and foremost to limit the overshoot induced by the initial ramp, because a measured traction $> 6\,\mathrm{V}$ is dangerous for the plant. We should thus increase $K_{P_{outer}}$ until the traction overshoot is low enough, and then check if the resulting lower damping is still satisfying.

Fortunately, figure 5.14 shows that values 5.2 - 5.4 along with an initial ramp on the traction provide an acceptable overshoot and sufficient damping. Figure 5.15 finishes the validation this design by showing that the actuators do not go into saturation and are reasonably sollicitated during normal operation.

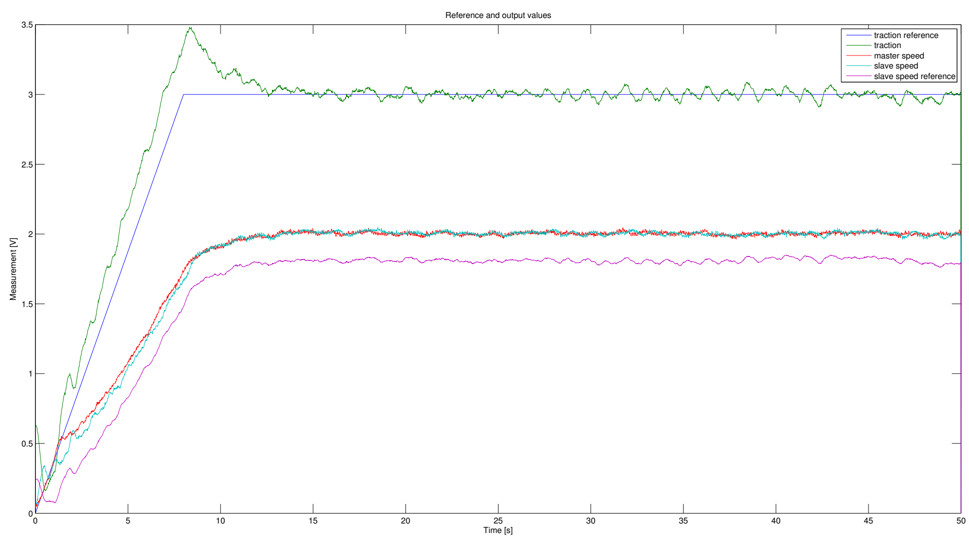

Figure 5.14: Operation of the rolling mill. $K_{p_{outer}} = 2$.

$$K_{p_{inner}} = -0.14 \tag{5.2}$$
$$K_{p_{outer}} = 2 \tag{5.3}$$
$$K_i = 4.2 \tag{5.4}$$

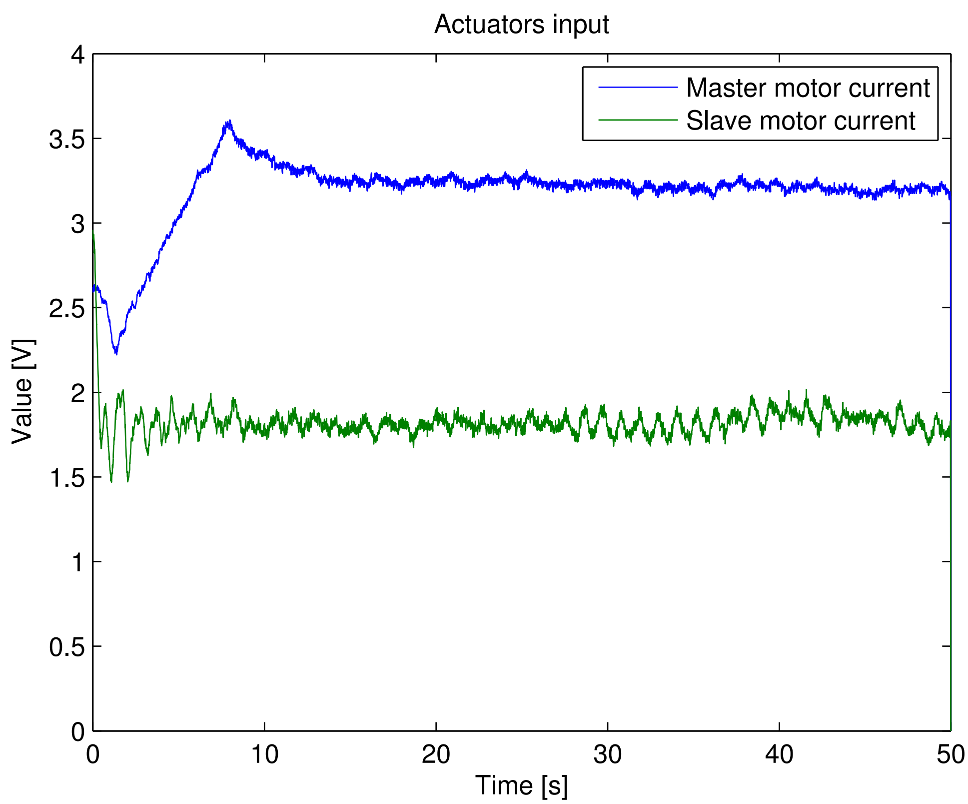

Figure 5.15: Actuators values during operation with the final controller

Chapter 6

Thickness control

In this section some considerations will be made about the more advanced objective. Since it is not attainable in the course of five lab sessions only some theoretical reflections will be made. The more advanced objective was to be able to control the output thickness of the metallic strip.

6.1 Workings of the Complete Rolling Mill Plant

The full workings of the mill are as follows: one motor unwinds the metal strip from a spool. The metal strip passes between two cylinders driven by a second motor. The metal is compressed between these too cylinders and is subsequently wound up by the third motor. The thickness is mainly controlled by the distance between the two rolls but also depends the other parameters of the plant in a non linear way. This distance can be controlled manually by a crank.

We have three motors as actuators, one manual setting, and multiple sensors:

- Thickness sensor before the rolls
- Thickness sensor after the rolls
- Traction sensor before the rolls
- Traction sensor after the rolls
- Velocity sensors for each of the motors
- Rolling force sensor

The technical reason the objective is not attainable is that there are not enough DAC ports to control every motor independently. In terms of required research and time, this project would also be more fitted as a master thesis. Still, in the following, we will present a possible approach to try to control the thickness of the metallic strip.

6.2 Controller considerations

First, the plant has to be biased around a setpoint depending on the desired output thickness. The bias parameters are the master speed, which is now the speed of the middle pair of rolls, the spacing between the middle rolls and the traction left and right of the middle pair of rolls. Those biasing parameters can be found in a table describing the rolling mill. Around this setpoint, the thickness has to be controlled using small signal variations of the left and right traction.

To control the thickness, one should use a cascade controller on top of the traction controller we already designed during this report. However, the existing controller must be adapted to use the middle motor as master motor, and duplicated in order to independently control the left and right traction. From there on, one has to see if it is possible to again simplify the controlling scheme by fixing a traction on a steady value and only controlling the other one around a master value. A worse case would be that both tractions have to be dynamically controlled simultaneously in order to ensure the plant works properly. In the worst case, the traction variations needed to control the thickness are so large that the speed setpoint has to be changed during the operation. In that case, the controller has to implement a lookup table in order to control the non linear parameters.

With such a setup, it should be at least possible to control the thickness of the metallic strip around a constant reference. However, as we showed, tracking a dynamic reference would be an even more advanced requirement due to the heavy non linearity of the plant. However, there are already ports available to control the spacing of the middle rolls during operation rather than using a manual crank, which should make this possible.

Appendix A

Controller code

```
%controller.m
function [time, reference, output, input, innerTractionRefArray, rSpeedRefArray] = controller()

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %Control System Design Lab: Overall Controller Loop
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %%
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %Setup
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  T_S = 0.01;%Set the sampling time.


  function [nSamples, time, reference, output, input] = referenceGenerator()
    %REF(1,:) = traction
    %REF(2,:) = master speed

    %traction
    EXP_LENGTH = 50;%sec
    TRACTION_REF = 3;
    TRAC_SETUP_TIME = 8;

    tracRamp = 0:T_S*TRACTION_REF/TRAC_SETUP_TIME:TRACTION_REF;
    reference(1,:) = [tracRamp TRACTION_REF*ones(1, EXP_LENGTH/T_S - length(tracRamp))];

    %master
    SETUP_TIME = 8;%seconds
    SETUP_POINT = 2;%V Left Motor

    ramp = 0:T_S*SETUP_POINT/SETUP_TIME:SETUP_POINT;
    rampLength = length(ramp);

    reference(2,:) = [ramp ones(1,length(reference(1,:))-rampLength)*SETUP_POINT];

    nSamples = length(reference);
    time=0:T_S:(nSamples-1)*T_S;
    output = zeros(2, nSamples);
    input = zeros(8, nSamples);
  end
```

```
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Controllers
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function current = lmController(speedRef, measuredSpeed)
  if speedRef < 0
    FRICTION_COMPENSATOR  = -2.7;
  else
    FRICTION_COMPENSATOR  = 2.7;
  end
  KP = 2;
  KI = KP/3.642;

  persistent errorIntegral;
  if isempty(errorIntegral)
    errorIntegral = 0;
  end

  error = speedRef - measuredSpeed;
  errorIntegral = errorIntegral + error*T_S;

  current = FRICTION_COMPENSATOR + KP*error + KI*errorIntegral;
end

function current = rmController(speedRef, measuredSpeed)
  if speedRef < 0
    FRICTION_COMPENSATOR  = -2.1;
  else
    FRICTION_COMPENSATOR  = 2.1;
  end
  KP = 3;

  error = speedRef - measuredSpeed;
  current = FRICTION_COMPENSATOR + KP*error;
end

function speedRef = innerLoopSpeed(traction, innerTractionRef)
  K = -0.123;
  speedRef = K*(innerTractionRef-traction);
end

function innerTractionRef = outerLoopTraction(traction, tractionRef)
  persistent errorIntegral;
  if isempty(errorIntegral)
    errorIntegral = 0;
  end
  K = 2;
  ZERO = 1.9;
  KI = ZERO*K;
  error = tractionRef - traction;
  errorIntegral = errorIntegral + error*T_S;
  innerTractionRef = K*(error) + KI*errorIntegral;
end
```

```
  %%
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %Main Loop
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  [nSamples, time, reference, output, input] = referenceGenerator();
  %precompute the reference and generate arrays of according size
  innerTractionRefArray = zeros(1,nSamples);
  rSpeedRefArray = zeros(1,nSamples);
  i=1;
  while i < nSamples
    tic %Begins the first strike of the clock.
    [input(1,i), input(2,i), input(3,i), input(4,i), input(5,i), ...
      input(6,i), input(7,i), input(8,i)]  = anain; %Acquisition of the measurements.

    rmVelocity = input(5,i); %saving the inputs in a variable for ease of working
    lmVelocity = input(4,i);
    lTraction = input(3,i);
    rTraction = input(2,i);

    if(lTraction > 6 || rTraction > 6) % safety measures if traction is to high
      lmCurrent = 0;
      rmCurrent = 0;
      i = nSamples + 1;
    else
      lmCurrent = lmController(reference(2,i), lmVelocity);

      %Cascade loops
      innerTractionRef = outerLoopTraction(rTraction, reference(1,i));
      rSpeedRef = innerLoopSpeed(rTraction, innerTractionRef);
      rmCurrent = rmController(rSpeedRef, rmVelocity);
      innerTractionRefArray(i) = innerTractionRef;
      rSpeedRefArray(i) = rSpeedRef;
    end

    output(2,i) = rmCurrent;
    output(1,i) = lmCurrent;
    anaout(lmCurrent, rmCurrent);

    if toc > T_S
      disp('Sampling time too small');%Test if the sampling time is too small.
    else
      while toc <= T_S
        %Does nothing until the second strike of the clock reaches the sampling time set.
      end
    end
    i=i+1;
  end
end
```

```
%main.m
openinout;
[time, reference, output, input, innerTractionRef, rSpeedRef] = controller();
anaout(0,0);
closeinout;
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Plots
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure %Open a new window for plot.
plot(time,reference(1,:), time, input(2,:), time,...
  input(4,:), time, input(5,:), time, rSpeedRef); %Plot the experiment (input and output).
legend('traction reference','traction','master speed',...
  'slave speed', 'slave speed reference');
title('Reference and output values')
xlabel('Time [s]');ylabel('Measurement [V]');

figure;
plot(time, output(1,:), time, output(2,:));
legend('Master motor current', 'Slave motor current');
title('Actuators input')
xlabel('Time [s]');ylabel('Value [V]');
```