Low Level Spindle Control on an Autonomous Mate for IntelliGent Operations, AMIGO

N.E. Bilterijst
0616995

University of Technology, Eindhoven
Department of Mechanical Engineering,
Control Systems Technology

Supervisor:
Ir. J. Elfring

June 29, 2011
Figure 1: AMIGO, Autonomous Mate for IntelliGent Operations
Contents

1 Introduction ............................................ 4

2 Spindle specifications .................................. 5
   2.1 Hardware .......................................... 6
   2.1.1 EtherCAT stack and amplifiers ................. 6
   2.1.2 Lifting mechanism .............................. 6
   2.2 Spindle requirements .............................. 8

3 System Identification .................................. 9
   3.1 Frequency response function estimation .......... 9
   3.2 Closed loop theory ................................ 9
   3.3 Measurement ...................................... 11
      3.3.1 Experiment setup ............................. 11
      3.3.2 Analysis ..................................... 12

4 Controller .............................................. 17
   4.1 Loop shaping ...................................... 17
   4.2 Feed Forward ..................................... 21
   4.3 Resulting behavior ............................... 24

5 C++ implementation .................................... 26
   5.1 I/O interface ..................................... 26
   5.2 Orocos components ............................... 26
   5.3 Safety features .................................. 31

6 ROS Package ........................................... 33
   6.1 Amigo_spindle_controller package ............... 33
   6.2 The Operating Program Script .................... 33
      6.2.1 Starting up the controller on Amigo .......... 34

7 Summary .................................................. 36
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1 Matlab file for the FRF analysis</td>
<td>38</td>
</tr>
<tr>
<td>A.2 FRF results</td>
<td>42</td>
</tr>
<tr>
<td>A.2.1 Bodediagram Plant</td>
<td>42</td>
</tr>
<tr>
<td>A.2.2 Bodediagram Open Loop</td>
<td>43</td>
</tr>
<tr>
<td>A.2.3 Bodediagram Sensitivity</td>
<td>44</td>
</tr>
<tr>
<td>A.2.4 Bodediagram Process Sensitivity</td>
<td>45</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Amigo, which is an acronym for Autonomous Mate for IntelliGent Operations, is a prototype of a service robot. Amigo participates in the Robocup @Home competition. In this competition robots, service robots in particular, are exposed to a variety of challenges.

These challenges are mainly focussed on tasks which a specific service robot should be able to do. Such as recognizing objects or people and grasping objects. An example in which both of these aspects are used is that Amigo should be able to pick up a glass of water from a table or a cabinet and hand it over to a person lying on a hospital bed. But in other cases it can be desired to pick up an object placed on the ground. If Amigo must be able to pick up an object that can be placed on the ground and an object placed on for instance a higher table, the arms with which Amigo is equipped are not sufficient to reach both objects. Therefore a spindle has been installed that can move the torso up and down. The arms are connected to the torso and the spindle becomes an extension for the arms.

With the spindle at it’s lowest position Amigo is able to pick up objects from the ground. And at it’s highest position objects can be easily picked up from high tables or cabinets. Since Amigo is built as a service robot, it needs to look friendly and not intimidating. Being able to set Amigo to a lower position makes it look much more friendly.

The low level control of the spindle is the subject of this report. By designing a controller for the spindle certain aspects are of importance. There are certain requirements for the controller and hardware properties which must be taken into account and some safety measures must be implemented to assure the spindle to go up and down safely. These aspects that were of importance to come up with the final design of the controller are discussed in the following chapters.
Chapter 2

Spindle specifications

The hardware specifications of the spindle will be discussed in this chapter. It will give more insight into how the spindle is connected and how the controller should be implemented to function properly. Also the purpose and the requirements for the controller performance will be the subject of this chapter. In figure 2.1 Amigo is shown at it’s lowest and at it’s highest spindle position.
2.1 Hardware

This section is about the hardware required to use the spindle installed on Amigo.

2.1.1 EtherCAT stack and amplifiers

Amigo is equipped with three pc’s running on Linux having the Robot Operating System (ROS) installed. The pc’s contain the ROS software packages that have been written to control Amigo’s features and are connected to an Beckhoff EtherCAT stack with an ethernet cable. The EtherCAT stack contains both digital and analog input and output ports. The analog output ports are connected with the amplifiers situated in the base of the robot and the amplifiers power the actuators on Amigo. This way the pc’s can communicate with the actuators on Amigo.

2.1.2 Lifting mechanism

The lifting mechanism consists of a ball screw spindle mechanism. An exploded view of this mechanism is depicted in Figure 2.2. One end of the spindle is supported by an axial support and the other end by a deep groove roller bearing. The axial support is bolted to the motor spindle support. The motor assembly consists of four Maxon products, namely an actuator, a gearhead, an encoder and the brake. The assembly is mounted on this support. The power from the motor is transferred to the spindle via a tooth belt drive train. The mechanism has a stroke of $435 \text{ mm}$. However, the actual measured stroke of the mechanism is less than $435 \text{ mm}$, because of the endstops that are implemented in the spindle at it’s maximum and it’s minimum position. The actual measured stroke is $413 \text{ mm}$. A more detailed description can be found on the wiki page of AMIGO [6].

One of the digital output ports on the EtherCAT stack is connected with a brake for the spindle. The brake locks the spindle in it’s place, and is required to prevent Amigo’s torso from falling down when no power is sent to the actuators by the amplifiers.
Figure 2.2: Exploded view of the spindle.

Specifications:

- Spindle screw pitch (*pitch*): 0.004 [m]
- Number of encodersteps (*encsteps*): 2000 [-]
- Transmission ratio (*transmission*): 4.8 [-]

The formula used to calculate the conversion factor from encodersteps to the SI system is given by equation (2.1).

\[
\frac{\text{pitch}}{\text{transmission} \cdot \text{encsteps}} \quad [m] \quad (2.1)
\]
2.2 Spindle requirements

To achieve a controller with both high functionality and a certain degree of safety, the requirements for the spindle should be determined.

Designing a controller that makes the torso of Amigo moves up and down very fast with high acceleration and deceleration, does not make Amigo look friendly but rather aggressive. That is the least we want to achieve with a service robot who’s work environment should be a hospital room, a household or among children. It needs to move up and down with a smooth trajectory. This is one argument to implement a limitation in maximum velocity and acceleration. Another argument for the limitation is that if a task has to be performed when, for instance a glass of water must be picked up and the movement of the torso is very aggressive, it will probably spill a lot of water. As a result, the glass of water is empty before it reaches its destination. However, moving up and down should be done within reasonable time. A maximum velocity of 0.05 \( [m/s] \) and a maximum acceleration of 0.02 \( [m/s^2] \) are considered to be reasonable.

Besides the smooth trajectory, the controller should also be accurate. If an object must be grasped from a cabinet or table the spindle is used as an extension for the arms. Grasping objects gets more difficult if the accuracy of the spindle controller is not sufficient. In an ideal situation there is no steady state error in the control of the spindle. However, a steady state error of about 2-3 \( [mm] \) is sufficient for reliable grasping. Therefore this is another requirement.

In order for the controller to be safe, some safety features should be implemented. The spindle must slow down when it reaches it’s end positions, it must stop moving when an unknown object blocks the spindle and it is not allowed to move down when it results in the arms touching the base.

In chapter 5 about the implementation of the controller, the methods implementing these requirements will be explained.
Chapter 3
System Identification

To be able to design a controller that meets the requirements some sort of model of the system is desired. A frequency response measurement to determine the frequency response is an often used technique to identify the system. This chapter explains in more detail how the system identification has been performed.

3.1 Frequency response function estimation

A frequency response function measurement is usually performed by making use of the cross- and autopowerspectra. The power spectrum of a signal in a given frequency band can be calculated by integrating over positive and negative frequencies. The cross powerspectrum is a power spectrum between two signals, the autopowerspectrum is the powerspectrum of a single signal. In the open-loop case the measured crosspowerspectrum is divided by the measured autopowerspectrum to create an estimator for the frequency response function. In a closed-loop situation it is not required to look at the input and the output. Comparing the input and output will give a biased result when extraneous noise is present. When performing an analysis with the sensitivity this problem can be eliminated.

The system identification of the spindle mechanism is performed in the closed loop environment. Without a basic controller the behavior of the system is not stable and therefore it can not be analyzed.

3.2 Closed loop theory

This section covers the general theory about frequency response estimation in the closed loop environment. The next section describes the practical ap-
approach of the frequency response estimation for the spindle.

There are two different methods to identify a system in the closed loop case, the direct and indirect method. The direct method is performed in the same way as in the open loop case. But as mentioned before, a disadvantage of this method is the distortion in the estimated frequency response function because the autopowerspectrum of the extraneous noise influences the estimation. Also the controller must be known in advance.

When identifying the system using the indirect method, three signals must be measured during the experiment, namely the system output $y$, the noise $w$ and the control output $u$. The noise will be added after the control output $u$ and before the plant, see Figure 3.1.

![Figure 3.1: A closed loop control system in a schematic view.](image)

The sensitivity function can be defined as in equation (3.1) and can be estimated by dividing the measured crosspowerspectrum of the noise $w$ and the plant input $v$ by the measured autopowerspectrum of the noise $w$.

$$S = \frac{1}{1 + PC} = \frac{S_{wv}}{S_{ww}} \quad (3.1)$$

With $S_{wv}$ the crosspowerspectrum from $w$ and $v$, and $S_{ww}$ the autopowerspectrum from $w$. The process sensitivity function can be defined as in equation (3.2) and can be estimated by dividing the measured crosspowerspectrum of the noise $w$ and the output $y$ by the measured autopowerspectrum of $w$.

$$PS = \frac{P}{1 + PC} = \frac{S_{wy}}{S_{ww}} \quad (3.2)$$
With $S_{wy}$ the crosspowerspectrum from $w$ and $y$, and $S_{ww}$ the autopowerspectrum from $w$. The plant can then be obtained by dividing the estimated process sensitivity by the estimated sensitivity:

$$P = \frac{PS}{S} \quad (3.3)$$

The coherence of the sensitivity given by equation (3.4) gets lower when more extraneous correlated noise is present, therefore the coherence can be considered as a good indicator for the quality of the measurement. The same holds for the coherence for the process sensitivity given by equation (3.5). However, if the estimation of the system is done by dividing the process sensitivity by the sensitivity, it is harder to determine the coherence since we are dealing with three different signals. A suggestion for the actual coherence is a combination of the individual coherence of the process sensitivity respectively the sensitivity. More information about the theory can be found in the report about Frequency response measurements by Johan Boot [3].

$$\gamma_{wu}^2 = \frac{S_{wu}}{|C|^2 S_{rr}} \quad (3.4)$$

$$\gamma_{wy}^2 = \frac{|H|^2 S_{ww}}{|C|^2 |H|^2 S_{rr} + |H|^2 S_{ww}} \quad (3.5)$$

### 3.3 Measurement

This section will cover the practical implementation of the technique mentioned in the last section to identify the behavior of the spindle.

#### 3.3.1 Experiment setup

To obtain a good model for the dynamical behavior of the system, the spindle must be in a continuous motion while performing the frequency response function measurement. A reference trajectory has been created, which can be seen as a saw-tooth profile between 10 [cm] and 30 [cm] of the spindle mechanism’s stroke (Figure 3.2). The actual stroke is 413 [mm], the extra margin on the bottom as well as on the top has been chosen this way to give the spindle some space to overshoot during the measurement. To assure the continuous and as smooth as possible motion, a simple controller has
been used to make it a closed loop system. The controller consists of a proportional gain and to overcome a large portion of the friction some feed forward has been applied. After the system is brought into motion, random noise is inserted into the system after the controller output and before the plant (as shown in figure 3.1). The random noise is not completely random but it’s limited to a certain minimum and maximum.

Due to the randomness of the noise the system response can be measured over a high range of frequencies. The response to high frequent signals can be measured in a very short amount of time, while very low frequency responses need a lot of time to measure. So longer measurements give more reliable information at all frequencies, but this especially accounts for smaller frequencies. The system response is measured for about 2.5 minutes and the control output $u$, the error $e$ and the noise $w$ are stored in a data file for later analysis.

![Figure 3.2: The sawtooth signal used during the measurements.](image)

### 3.3.2 Analysis

*Matlab* possesses a function to determine the sensitivity and it’s coherence and a function to determine the process sensitivity and it’s coherence, see appendix A.1.

The measurement data points are time sampled discrete, instead of continuous, analysis of the discrete data is usually done with the use of a window. In this case the so called *Hann*-window. The window determines the distribution of the measured points over the frequency range. $nfft$ is the length of the window, if it is larger you have more frequency points in the window. $noverlap$ is the amount of overlap between the segments and is set to 50% ($nfft/2$). Choosing a longer window, a higher $nfft$, gives more precise information at a particular frequency due to more frequency points in the window. But if the measurement was not taken long enough, there is less
measurement information and less overlapping segments can be used. Thus the result could be less accurate.
When the sensitivity and process sensitivity are known the plant can be determined. Figure 3.3 shows the bodediagram of the system. The hann-window has been chosen in such a way that there are less frequency points in the lower frequency range but the result of frequency response of the system in this range is more averaged out.

The bodediagram shows that the measurement is not that reliable, because there is a lot of delay present already at very low frequencies. Some variables are logged multiple times while reporting with the same value in the data. The values of the variables are analyzed at a certain timestamp, but due to the multiple logging the wrong timestamp can be assigned to a certain value of a variable. That can create some delay when analyzing the signals of the different variables. This is in fact a problem with its cause at a bad A/D-conversion. This problem needs to be eliminated.

The corresponding values of the error and the control output with the unique values in the measurement data of the added noise are used as data for the analysis. The other not corresponding values are deleted from the file. This way a substantial improvement is made with the elimination of the delay. Figure 3.5 shows the bodediagram of the new improved Frequency Responce Function measurement.

Also the coherence of the sensitivity and process sensitivity show that the measurement is not completely reliable, the coherence of the first analysis is shown in figure 3.4. The coherence of the second analysis shows much better reliability, see figure 3.6. The coherence should approach 0.8 for a reliable measurement, we can see that at a frequency range of about 8 [Hz] till approximately 30 [Hz] the coherence does approach 0.8. Thus we can say that the bodediagram of the plant is reliable enough in this specified region.

In appendix A.1 the complete Matlab-file is shown. The output of the file are four figures; the bodediagram of the sensitivity and the process sensitivity both with its coherence, the closed loop and the open loop bodediagram (see Appendix A.2).

The dominant effects of Amigo’s torso can easily be reviewed in the bodediagram, a slope of -2 and a phase of 180° is clearly present in the amplitude between approximately 1 [Hz] and 100 [Hz] which represents a large effect of the mass in the dynamical behavior.
Figure 3.3: The bode diagram of the system.

Figure 3.4: The coherence of the sensitivity (first figure) and the process sensitivity (second figure).
Figure 3.5: The bodediagram of the system after filtering the data.

Figure 3.6: The coherence of the sensitivity (first figure) and the process sensitivity (second figure).
Chapter 4

Controller

4.1 Loop shaping

The design of the controller is done with the toolbox `shapeit` in Matlab. The output of the analysis for the FRF, calculated by the matlab file, can easily be loaded into Shapeit. Once loaded, controller components can be added to acquire a controller that meets the general closed loop stability margins. These margins are the phase margin, gain margin and modulus margin and should generally meet these values:

- Modulus Margin \(< 6\) [dB]
- Phase Margin \(> 30^\circ\)
- Gain Margin \(> 6\) [dB]

The main rule in control technology is to get the phase margin right first, and then improve performance. When looking at the bodediagram of the plant, figure 3.5, we can see that phase must be added to make it stable. Phase can be added by applying a differential action, the slope in the amplitude gets reduced to \(-1\) and adds \(90^\circ\) phase. A normal PD controller would add these values over the complete frequency range. However, that is not desired when adding differential action on this plant. Adding phase at higher frequencies makes it unstable and the sensitivity gets too high.

A solution is adding a Lead/Lag filter instead of using a PD-controller. The Lead/Lag filter adds phase at a defined frequency interval. The spindle moves at low speeds, with low acceleration and deceleration. Therefore the bandwith will probably be in the order of 1 to 10 Hz. So the interval is chosen in such way that at the bandwidth phase will be added, the zero of
the Lead/Lag is set to 1.6 [Hz] and the pole to 60 [Hz].

Now the phase margin is correct the performance can be increased. Increasing the proportional action of the controller improves the performance, at least to a certain extent. When the gain becomes too high, the system can become unstable or the sensitivity gets too high and then the modulus margin requirement will not be met.

High frequent noise in the system creates some disturbances at higher frequencies. If the gain is increased also these frequencies will be amplified, but that is not desired because it will lead to worse sensitivity and thus worse performance. These higher frequencies must be suppressed. This can be achieved by using a first order low pass filter, it damps the frequencies higher than it’s defined breakpoint. The pole is set to 20 [Hz]. When the filter is applied, the gain can be increased a little more for better performance.

Eventually the spindle is required to have no or a little steady state error. The implementation of an integrator can assure the absence of steady state errors. However, implementation of an integrator is hard because it reduces the phase margins by 90°. Certainly with this plant, there already is a lot of delay. That is why the differential action was necessary, but adding an integrator would eliminate the effects of the differentiator. It was not possible to implement an integrator over the whole frequency range. A weak integrator with a zero at 0.3 [Hz] is implemented. The integrator does not have much effect on the phase margin around the bandwidth, but it is just enough to eliminate the steady state errors of the spindle. The final design of the controller is shown in figure 4.1.

And the values of the controller components are summarized below:

- Gain 40
- Lead/Lag zero: 1.6 [Hz], pole: 60 [Hz]
- First order low pass pole: 20 [Hz]
- Integrator zero: 0.3 [Hz]
Controller margins and stability

The nyquist plot of the controlled system (figure 4.2) shows it is stable, because the point -1 stays left of the line. The controller margins can be determined from the open loop bodediagram and the sensitivity bodediagram (figure 4.3 and 4.4 respectively).

The controller closed loop stability margins for this controlled system and the minimum margins they must meet, are as follows:

- Bandwith \(4.34 \text{ [Hz]}\)
- Modulus Margin \(5.8 \text{ [dB]}\) \(< 6 \text{ [dB]}\)
- Phase Margin \(52.0^\circ\) \(> 30^\circ\)
- Gain Margin \(13.2 \text{ [dB]}\) \(> 6 \text{ [dB]}\)

The controller is stable and it meets the margins, thus it can be regarded as a good controller at least in theory. However, the coherence of the process sensitivity (figure 3.6) shows that at a bandwith of \(4.34 \text{ [Hz]}\) the measurement is less reliable. It is only reliable at a frequency range of \(8 \text{ [Hz]}\) till approximately \(30 \text{ [Hz]}\). Whether these margins actually meet the requirements is not really clear, extensive testing has shown however that the applied controller is stable and robust. In section 4.3 the test results with the controller and the practical performance of the spindle are discussed.
Figure 4.2: Nyquist plot of the final stable controller.

Figure 4.3: Bodediagram of the open loop.
4.2 Feed Forward

The spindle mechanism entails a lot of friction and to significantly improve the controller’s performance this friction must be overcome in advance. A solution to this is adding feed forward to the system. In an ideal situation the feed forward will get the spindle at the desired height without much input from the controller itself.

The feed forward can be divided into separate parts and are listed below:

- Static friction feed forward
- Dynamic friction feed forward
- Gravity feed forward
- Acceleration feed forward
Static friction

The friction in the spindle mechanism can be divided in a static part and a dynamic part. When the spindle is not in motion it has a certain static friction that needs to be overcome before the spindle starts to move, that is the purpose of the feed forward for the static friction. The output of the feed forward consists of a constant multiplied by the direction of the velocity.

Determination of the constant for static friction also depends on the constant for the gravity compensation. A combination between these two constants can be determined by measuring the output of the system at the moment the spindle starts to move down and at the moment the spindle starts to move up. The ratio between the gravity and static friction when going down can be showed in a simplified form as in equation (4.1). The force of the static friction and the force generated by the controller to keep it at it’s place should be equal to the force of the gravity. When the spindle starts moving up equation (4.2) applies. With $F_{\text{controller}_{\text{down}}}$ and $F_{\text{controller}_{\text{up}}}$ representing the output of the controller when it starts to move down respectively up, $F_{\text{w\_static}}$ and $F_{\text{gravity}}$ can then be calculated.

\[ F_{\text{w\_static}} + F_{\text{controller}_{\text{down}}} = F_{\text{gravity}} \]  
\[ F_{\text{gravity}} - F_{\text{w\_static}} = F_{\text{controller}_{\text{up}}} \]

Dynamic friction

When the spindle is in motion there is also a dynamic friction present in the system. The feed forward for the dynamic friction will overcome this and it consists of a constant multiplied by the magnitude (thus also direction) of the velocity.

It is determined by looking at the average output of the system while the spindle is moving up with a constant velocity. Equation (4.3) represents the forces acting on the spindle during the motion at a constant velocity. When the forces for static friction and gravity are determined as explained above the force of dynamic friction can be determined. And a constant for compensating this dynamic friction can be derived.

\[ F_{\text{w\_static}} + F_{\text{w\_dynamic}} + F_{\text{gravity}} = F_{\text{controller}} \]
Gravity

Without the feed forward for the gravity and there would be no static friction the spindle will fall down. The feed forward for compensating the gravity is basically a constant output added to the controller output at all times, regardless whether it is in motion or not. Determination of this constant is described above.

Acceleration

For smoother acceleration and deceleration feed forward for the acceleration has been added. It consists of a constant multiplied by the magnitude of the acceleration. Determination is done by looking at the average output of the controller during constant acceleration and deceleration.

The values determined for the feed forward described in the sections above are based on estimated physical quantities. The parameters have been tuned using a trial and error method while looking closely at the performances and the behavior of the spindle. The tuned parameters are:

- Static friction 0.05 [V]
- Dynamic friction 0.4 [V]
- Gravity 0.07 [V]
- Acceleration 0.3 [V]
4.3 Resulting behavior

Figure 4.5 shows the time domain performance of the spindle with the controller designed in the previous sections. The steady state error of the spindle eventually goes almost to zero, in fact about 0.01 [mm]. This can be seen in figure 4.6 of a close up of the error. This is good because the required steady state error was 2 - 3 [mm]. During movement an error of less than 1.0 [mm] is measured, and even then the large part of the error is during acceleration and deceleration. The error of 5.0 [mm] around the 1x10^4th timestamp is part of the homing procedure and thus an expected error. The error has to build up to 5 [mm] before it is assumed to be homed. The figure shows that the controller is working great on the system and that it performs a good following behavior.

![Figure 4.5: The performance of the spindle.](image)
Figure 4.6: A close up of the error, clearly showing the steady state error.
Chapter 5

C++ implementation

This chapter discusses the implementation of the controller and its features in the programming language C++. It describes how the code is built and the reasons why it is built this way.

5.1 I/O interface

As mentioned in section 2.1 the pc’s are connected with the EtherCAT stack and the stack is connected to the amplifiers which power the encoders. Some code has to be written to let the pc’s communicate with the stack, this is done with Orocos and stands for Open RObot COntrol Software[7]. It is a realtime toolkit for programming robot software and it runs under ROS.

5.2 Orocos components

The controller is built up from multiple orocos components each with its own function. A scheme of all the orocos components in the controller is shown in figure 5.1. The function as well as how it works for each component, will shortly be explained in the following subsections.
Figure 5.1: The controller structure built up with the separate orocos components.
Soem, DigitalOuts, AnalogOuts and ReadSpindleEncoders

These components are used to read or control the actual hardware components of Amigo.

The Soem component is the driver for the EtherCAT stack, DigitalOuts is the component that sends a boolean to the spindle brake. If it sends out true the spindle brake is off, and when it is false the spindle brake is on. The encoders are powered by the amplifiers which receive the controller output by the AnalogOuts component and the values of the encoder can be read by the ReadSpindleEncoders component. These encoder values are converted to a relative position from the spindle.

SpindleHoming

When the controller is started this component initially has a boolean homed set to false. As long as it is false the component follows a homing procedure. It changes the setpoint to a setpoint which is larger than the actuation stroke of the spindle and the velocity is initially set to $1 \text{ cm/s}$. The component listens to the error and when it exceeds $5 \text{ mm}$ the boolean homed is set to true, the maximum velocity and acceleration are reset in the SpindleReferenceGenerator component. The spindle is homed and from now on the homing procedure gets skipped. The remaining function of the SpindleHoming component is forwarding the setpoint and applying a correction factor in position calculation.

ReadSpindleSetpoint

A message that contains information for the spindle setpoint is published on a ROS topic and the function of the ReadSpindleSetpoint component is to read the message and retrieve the spindle setpoint. When it has read the message it will send the setpoint over the output port.

SpindleReferenceLimiter

It receives the setpoint from ReadSpindleSetpoint and it limits the setpoint when it exceeds the minimum or maximum value of the spindle’s actuation stroke. Also a small safety margin of $5 \text{ mm}$ is used when limiting the setpoint to the maximum or minimum. This is required if for some reason the controller would create a little overshoot. It also reads two ROS topics on which the tip positions of both arms are published. A certain ’no collision-zone’ is defined in the component, when one of the arm tips is reaching that zone the reference gets limited to a safe position where the arm are in a safe
position. The spindle can only move up and a warning message is published to the Orocos deployer. These safety features will also be discussed in section 5.3.

**CalculatePosition**

This component is a simple subtraction component and it subtracts the correction factor calculated while homing from the relative position to achieve the absolute spindle position, which is sent over it’s output port.

**PublishPosition**

The absolute position of the spindle must be published on a ROS topic so that other packages running on Amigo are aware of the spindle’s position. This component publishes the position to the right ROS topic.

**SpindleReferenceGenerator**

To assure a smooth motion with maximum velocity and acceleration from the current position to the setpoint an interpolator is used. That is the function of the SpindleReferenceGenerator component, it acts as an interpolator. With given velocity and acceleration as properties it calculates a smooth reference trajectory. The reference positions, velocities and accelerations are sent over the output ports.

**CalculateError**

Also a simple subtraction component which subtracts the current position from the reference position. The result is the position error and is sent out over the output port.

**SpindleGain, SpindleLeadLag, SpindleFirstOrderLowPass and SpindleIntegrator**

These components are the controller components that are multiplied by the error to create the controller output desired to eliminate the position error. In section 4.1 the controller filters are discussed.
CalculateFFW

In this component the feed forward is calculated that needs to be added to the controller output before it is sent to the amplifiers. It uses the reference velocity and acceleration from the SpindleReferenceGenerator to calculate the acceleration feed forward, the static friction compensation and the dynamic friction compensation.

CalculateOutput

A simple addition component, that adds the feed forward to the controller output. The total output is sent over the output port and the AnalogOuts component reads the value and sends it to the amplifiers.

SpindleSafety

A safety component that registers the error. If for an unknown reason the error exceeds a defined limit, which is set at 2 \([\text{cm}]\), the component will perceive the error as unsafe and it will send a boolean to the DigitalOuts component and to the OutputLimiter component so the brakes can be applied and the output can be set to zero.

OutputLimiter

It receives the controller output over one of it’s input ports and limits the controller output to 1 \([\text{V}]\) for safety reasons. The other input port receives the boolean from the SpindleSafety component and sets the output to 0 if the boolean is false.
5.3 Safety features

The safety features are more or less explained in the previous section. In this section we will summarize how the safety features are implemented. And some figures will explain the behavior of the safety features.

- A smooth reference trajectory with maximum velocity and acceleration is assured by the reference generator. Figure 4.5 is a good example of the smooth trajectory after the homing procedure.

- The spindle can not run into it’s endstops. Because of the homing, the exact position of the spindle can be derived and the incoming references are limited to the maximum and minimum value of the actuation stroke with a extra safety margin. Figure 5.2 shows the homing procedure and the limitation of the setpoint. The homing starts with a slow upwards movement until the error is large enough. The endstop limitation is shown in the right figure. The red line represents the given setpoint, which is too low in this case. The spindle’s lowest position with the safety margin is 0.005 \([m]\). Altough the setpoint is given at 0.0, the spindle shuts off the reference to 0.005. In this second figure also the effect of the integrator becomes clear.

- If for some reason the spindle gets blocked by an object, or the arms block the spindle movement because they are hitting for instance a table. Then that would result in an increasing error. The spindle brake is applied and the output to the actuators is set to 0 when the error gets larger than the defined 2 \([cm]\).

- If the controller receives a setpoint that would result in the arms colliding with the base, it detects the setpoint and changes it to a safe setpoint so that the arm positions get just above the base. The controller creates a smooth trajectory to this position and publishes a warning message to the deployer that the setpoint is limited. See figure 5.3. This safe position is defined as 35 \([cm]\) above Amigo’s baseframe. In the figure the red line, which represents the position of the arm tip, is limited to 35 \([cm]\). The setpoint of the spindle (blue line) would bring the arm position below that treshold, so the setpoint is changed to the setpoint of the spindle after safety limitation (green line) where the arm tip is at a position of 35 \([cm]\).
Figure 5.2: The left figure shows the homing. The right figure shows the endstop limitation.

Figure 5.3: The arm safety in action.
Chapter 6

ROS Package

6.1 Amigo_spindle_controller package

The *amigo_spindle_controller* package contains the controller with the safety features to safely operate the spindle on Amigo. If the arms are not turned on the controller will receive no information from the arms and a warning message will be published to the screen. It will tell you that the arm safety is not functioning.

The package is built up from different orocos components as described in chapter 5, most of the components are used from the *orocos_components* package. Some components have been written specifically for the spindle controller, they are included in the *amigo_spindle_controller* package.

6.2 The Operating Program Script

The orocos components are launched in a deployer and get connected with each other by a Operating Program Script, a (*ops*) file. In this script all the commands necessary to get the components connected and running are listed. Several sections can be distinguished in the script. The first section loads and configures all the necessary components, the second section makes all the connections between the components and connects some of the components to a ROS topic and the last section starts all the components.
6.2.1 Starting up the controller on Amigo

Launching the controller on Amigo can be done in two different ways. It is possible to launch only the spindle controller on Amigo, but it is also possible to launch all the required hardware controllers for Amigo at the same time. There are two different launch files for launching the controllers. Most of the time the complete hardware launch will be used.

Complete Amigo hardware startup

A launch file is included in the `amigo_hardware_startup` package, `start_amigo2.launch`. It starts all the hardware controllers connected to Amigo’s second pc, so also the spindle controller. The launch file loads the `spindle_controller.ops` file into the deployer which contains the startup information. The procedure is as follows, and enter the following commands in a terminal:

Make sure that the package is updated from the SVN:

```bash
$ roscd amigo_spindle_controller
$ svn up
```

Make the package:

```bash
$ rosmake
```

Launch the controllers by running the launch file:

```bash
$ roslaunch amigo_hardware_startup start_amigo2.launch
```

A popup window will appear where the deployer is loaded, error or warning messages will be published here. The controller is running and it will start it’s homing procedure, after homing it will move down to a safe position of 35 \([\text{cm}]\) where the arms are not able to hit the base.

Normally the setpoints for the spindle are received by the `inverse_kinematics` package. However, it is possible to send the spindle up or down without this package. A message must be sent over the topic `/spindle_controller/spindle_coordinates` and the message is of the `amigo_msgs/spindle_setpoint` type. The message consists of a vector containing three doubles and a boolean.
Only the first double is of importance for giving a setpoint, it represents the required setpoint for the spindle in [m]. Sending it to a position of, for instance 20 [cm] requires the command:

```
$ rostopic pub -1 /spindle_controller/spindle_coordinates amigo_msgs/spindle_setpoint - - 0.20 0.0 0.0 false
```

**Amigo only spindle hardware startup**

Almost the same applies for launching only the spindle controller as for all the hardware controllers. The only difference is another launch file that must be launched. The required launch file is located in the `amigo_spindle_controller` package itself. So launching it requires the command:

```
$ roslaunch amigo_spindle_controller start_spindle_controller.launch
```

Now only the spindle controller gets launched exactly in the same way as described in the last section, the other controllers remain off. Pay attention to the fact that at this moment there is no arm safety! Moving the spindle up and down requires a little extra attention. Giving a setpoint happens the same way by sending a message over the ROS topic.
Chapter 7

Summary

The package now contains a accurate and safe controller for the spindle. With a steady state error of less than 0.01 $[\text{mm}]$ and an accuracy of less than 1 $[\text{mm}]$ during movement, the spindle can be moved up or down to a given setpoint and can be used as an extension of Amigo’s arms.

There are some improvements that can be made to make the controller better. This mainly lies in the improvement of the system identification. The coherence of the performed system identification shows it has been a pretty reliable measurement. But at the same time it also shows that there is still room for improvement.

Now only random noise is used to identify the system. Adding specific sinusses with certain frequencies can make the measurement reliable at that particular frequencies. The spindle could also be moved up and down on different ranges of the actuation stroke, this would improve insight about lineairity of the system. Reduction methods for the delay created by the A/D-conversion could be used to reduce the effect of the delay. This would further increase measurement reliability.

However, the results have shown that the spindle controller is neat enough for its requirements. And therefore no steps have been made to improve reliabilty of the system identification.
Bibliography


Appendix A

A.1 Matlab file for the FRF analysis

% authors: Paul Maas
% date: 11/09/2007

% http://alexandria.tue.nl/repository/books/612380.pdf
% The sensitivity can be identified by dividing the measured
% crosspowerspectrum of w and v by the measured autopowerspectrum of w.
% By dividing the measured crosspowerspectrum of w and y by the measured
% autopowerspectrum of w the process sensitivity is obtained. Now we can
% divide the process sensitivity function by the sensitivity function in
% order to identify the plant;
% e: error
% u: control output
% v: control output+noise
% y: system output
% w: noise
clear all; clc

% Import the file
newData1 = importdata(’28022011metFFW.dat’);

% Create new variables in the base workspace from those fields.
for i = 1:size(newData1.colheaders, 2)
assignin(’base’, genvarname(newData1.colheadersi), newData1.data(;;i));
end

/TimeStamp(1) = TimeStamp(2) - 0.001;
TimeStamp = TimeStamp - TimeStamp(1);
starttime = 12;
endtime = 612;
Fs = 100;

%%
close all
error_pos = SpindleController0x2Eerror_pos(163000:325000);
control_output = SpindleController0x2Econtrol_output(163000:325000);
o noise = SpindleController0x2Enoise(163000:325000);

%e=error_pos;
%u=control_output;
%w = noise;
%v = u+w;

vv = noise;
vv1 = [vv(2:end);vv(end)];
index = find(vv = vv1);
e = error_pos(index);
u = control_output(index);
w = noise(index);
v=u+w;

Ts=0.004;
Fs=1/Ts;
Fmin = 0.01;
Fmax = 1/Ts;
nfft=4*1024;
noverlap=nfft/2;

% ========= Sensitivity [frf from w to v]
[C,F]=cohere(w,v,nfft,Fs,hann(nfft),nooverlap);
[Hsens,F] = tfestimate(w,v,hann(nfft),nooverlap,nfft,Fs);
angle_Hsens = atan2(imag(Hsens),real(Hsens));

figure(1);
subplot(3,1,1);
semilogx(F,abs(C));
title('sensitivity');
grid;
ylabel('coherence');
xlim([Fmin Fmax])

subplot(3,1,2);
semilogx(F,20*log10(abs(Hsens)));
grid;
ylabel('mag in [dB]');
xlim([Fmin Fmax])

subplot(3,1,3);
semilogx(F,angle(Hsens)/pi*180);
grid;
xlabel('frequency in [Hz]')';
ylabel('phase in [deg]');
xlim([Fmin Fmax])

% ======== Process Sensitivity [frf from w to e]
[C,F]=cohere(w,e,nfft,Fs,hann(nfft),noverlap);
[Hpsens,F] = tfestimate(w,e,hann(nfft),noverlap,nfft,Fs);
Hpsens=-Hpsens;
angle_Hpsens = atan2(imag(Hpsens),real(Hpsens));

figure(2);
subplot(3,1,1);
semilogx(F,abs(C));
title('process sensitivity')
grid;
ylabel('coherence');
xlim([Fmin Fmax])

subplot(3,1,2);
semilogx(F,20*log10(abs(Hpsens)));
grid;
ylabel('mag in [dB]');
xlim([Fmin Fmax])

subplot(3,1,3);
semilogx(F,angle_Hpsens/pi*180);
grid;
xlabel('frequency in [Hz]');
ylabel('phase in [deg]');
xlim([Fmin Fmax])

% ============ open loop frf
Hol=1./Hsens-1;
angle_Hol = atan2(imag(Hol),real(Hol));

figure(3);
subplot(2,1,1);
semilogx(F,20*log10(abs(Hol)));
title('Open loop')
grid;
ylabel('mag in [dB]');
xlim([Fmin Fmax])

subplot(2,1,2);
semilogx(F,angle_Hol/pi*180);
grid;
xlabel('frequency in [Hz]');
ylabel('phase in [deg]');
xlim([Fmin Fmax])

% ============ Plant [mechanics frf]
Hmech=Hpsens./Hsens;
angle_Hmech = angle_Hpsens-angle_Hsens;

figure(4);
title('Plant')
subplot(2,1,1);
semilogx(F,20*log10(abs(Hmech)));
grid;
ylabel('mag in [dB]');
xlim([Fmin Fmax])

subplot(2,1,2);
semilogx(F,angle_Hmech/pi*180);
grid;
xlabel('frequency in [Hz]');
ylabel('phase in [deg]');
xlim([Fmin Fmax])

41
A.2 FRF results

A.2.1 Bodediagram Plant

Figure A.1: Bodediagram Plant.
A.2.2 Bodediagram Open Loop

Figure A.2: Bodediagram Open Loop.
A.2.3 Bodediagram Sensitivity

Figure A.3: Bodediagram Sensitivity.
A.2.4 Bodediagram Process Sensitivity

Figure A.4: Bodediagram Process Sensitivity.