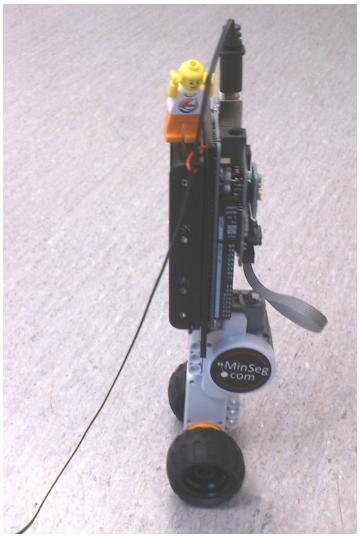
MinSeg balancing using pole-placement

This version: 2020-03-05



REGLERTEKNIK
AUTOMATIC CONTROL
LINKÖPING

Name:	
P-number:	
Date:	
Passed:	

Chapter 1

Introduction

The purpose of this lab is to balance the MinSeg robot. To develop the balancing controller, we will use model-based state-space methods using pole-placement.

Figure 1.1: Examples of systems (SegWay, JAS39 gripen, Proton Rocket, Plasma Tokamak) with unstable dynamics just as our MinSeg. All these applications require stabilizing controllers to function.

1.1 Hardware set-up

The lab is based on three main hardware components.

To begin with, we have a standard desktop computer. This computer is used to automatically develop and deploy code using MATLAB and SIMULINK models.

To supply power to the DC-motor, perform measurements and compute the control input, we use a board with an Arduino micro-controller which runs the auto-generated code. It also communicates with the desktop computer and thus allows us to look at measurements. The Arduino board with the motor and wheels is called the MinSeg.

1.2 Troubleshooting

The wheels turn slowly and/or erratically Make sure the tires do not rub against the motor. You can pull the wheels apart as they slide on the wheel axis.

Complaints about COM port or connection when downloading to board

- Make sure USB cable is firmly attached on both ends.
- Disconnect USB-cable and connect it again.
- Make sure you only have one simulink model open
- If it still does not work after several tries, follow the instructions in `fixcomport.pdf`.

Nothing happens/strange things happen The introductory tasks are performed in external mode with a slow sampling rate (MATLAB keeps connection to the board and can plot signals), while balancing requires a higher sampling rate which forces us to run in normal mode (no communication with MATLAB). Hence, make sure you are in the correct mode as specified in the task, and that you use the deploy button when in normal mode (instead of running with green run-button).

Common mistakes. Wrong sign in feedback when computing control input. Wrong order on states when computing control input. Missing to set sample-time in discrete-time integrator to -1. Sample-time variable TS not set to 0.006 when stabilizing.

Chapter 2

Preparation

The questions below, and all questions in the document marked as **Preparation** must be done before attending the lab. Note that there are additional preparation exercises in Chapter 3.

Solutions to all questions should be available upon request from the lab assistant.

The scheduled time spent with the laboratory equipment is only a small part of the complete lab, as a major part is spent on the theoretical material during preparations. When the lab starts, it is assumed you have done all preparations, and have a clear idea of the tasks that will be performed during the lab.

Preparation 1 *Read Sections 8.1-8.4, 8.8 and 9.1-9.2 in the course book by Ljung & Glad.*

Preparation 2 *A simple model of an inverted pendelum (which is a simplified model of our balancing robot) with a force applied to the base is given by*

$$\dot{x}(t) = \begin{pmatrix} 0 & 1 \\ 25 & 0 \end{pmatrix} x(t) + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u(t) \tag{2.1}$$

$$y(t) = \begin{pmatrix} 1 & 0 \end{pmatrix} x(t) \tag{2.2}$$

where $y(t) = x_1(t)$ is the angle of the pendelum, $x_2(t)$ is angular rate (rotational velocity), and $u(t)$ is the applied force. Show that the open-loop poles are given by ± 5.

Preparation 3 *A state-feedback controller $u(t) = -l_1 x_1(t) - l_2 x_2(t) + l_0 r(t)$ is used. Show that the closed-loop system is given by*

$$\dot{x}(t) = \begin{pmatrix} 0 & 1 \\ 25 - l_1 & -l_2 \end{pmatrix} x(t) + \begin{pmatrix} 0 \\ l_0 \end{pmatrix} r(t) \tag{2.3}$$

$$y(t) = \begin{pmatrix} 1 & 0 \end{pmatrix} x(t) \tag{2.4}$$

Preparation 4 *Find l_1 and l_2 such that the closed-loop poles are placed in -1 and -2.*

Preparation 5 *Now assume we try to design a controller which only uses the angle $x_1(t)$, i.e., l_2 is forced to be zero and no derivative feedback is used. Effectively, we use a P-controller. Show that it is impossible to stabilize the pendelum using this approach.*

Preparation 6 *Read the complete lab-pm. There are some theoretical questions in the pm which you are supposed to complete as preparation.*

Chapter 3

The lab

As explained above, in this lab we will finally balance the MinSeg, and to do so we will use state-space methods for control.

Items labeled **Preparation** are questions you are supposed to solve and fill out before attending the lab.

Items labeled **Task** are questions you solve when attending the lab and have access to the hardware

3.1 MinSeg modeling

Complete physical modelling of the MinSeg is beyond the scope of this lab, but let us look at the basic physics which is used to derive a model.

In principle, we have two parts, two wheels on a joint axle and a solid body. The forces acting on the robot are

1. The torque on the wheels causing them to rotate. This torque is generated by our control input $u(t)$ via the DC-motor, and the physics describing how the wheels behave when voltage is applied on the DC-motor was investigated in depth in LAB1. A positive input voltage $u(t)$ generates a torque in the positive $\theta(t)$ direction causing the wheels to rotate clock-wise.
2. Friction force which prevents the wheel from slipping on the surface.
3. Gravitational force.

Relevant physical states of the MinSeg are

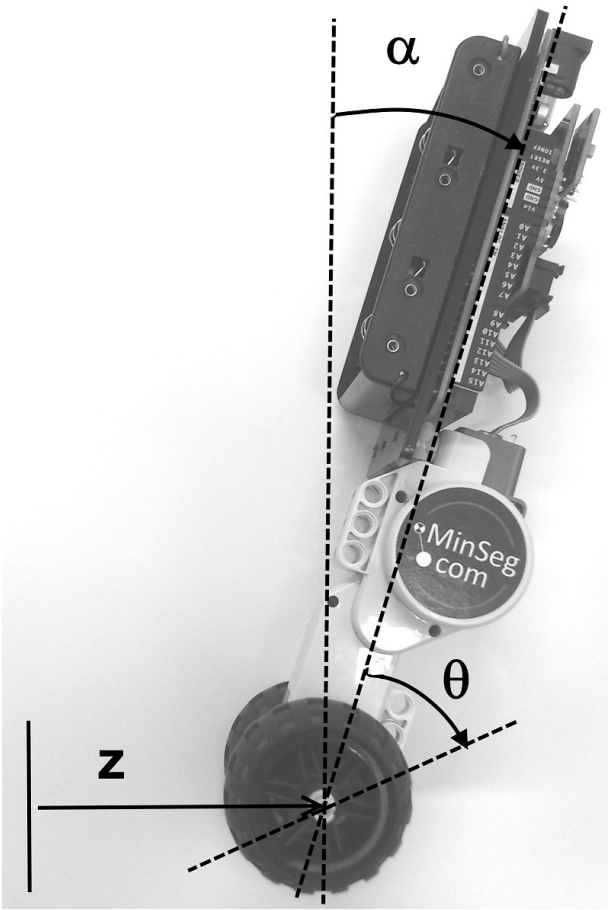


Figure 3.1: Definition of tilt-angle $\alpha(t)$ (positive clock-wise), wheel rotation angle $\theta(t)$ (positive clock-wise, relative to initial position), and position of center of wheel $z(t)$, relative to initial position.

1. The position $z(t)$ of the center of the wheel (relative to the initial position). To balance the MinSeg at a fixed place (the initial position where we started the MinSeg), we want this value to be 0.

2. The velocity $\dot{z}(t)$ of the center of the wheel.

3. The tilt-angle $\alpha(t)$ relative to straight up (this is the angle we want to keep at 0)

4. The angular rate $\dot{\alpha}(t)$.

As we will see, these 4 states are the states required to create a mathematical state-space model of the MinSeg, and used in a state-feedback $u(t) = f(z(t), \dot{z}(t), \alpha(t), \dot{\alpha}(t))$. The control goal is to have all states converge to 0, i.e., standing straight up with no movements, on the initial position where it was released.

3.2 Sensors and measurements

To implement the control law, we must have measurements of the 4 states. However, the only direct measurements we have are

1. The angular rate $\dot{\alpha}(t)$ is obtained from a **gyro** (which has noise and calibration error)
2. The angle $\alpha(t)$ can be reconstructed from **accelerometer** signals when moving slowly (with noise and calibration errors).
3. The wheel angle $\theta(t)$ can be measured (with limited precision) using a rotary encoder. This is the measurement we have used in previous labs.

One way to proceed could be to develop an observer[1] which estimates the 4 states from the available measurements. However, we will proceed with a slightly different approach, to reduce complexity of the design task.

Angle and angular rate measurements

The MinSeg is equipped with a so called inertial measurement unit (IMU) which gives measurements of angular rates (rotational velocities) and translational accelerations in 3D. You typically have a similar device in your smart phone.

The gyro allows us to pick up the rotational rate of the MinSeg in three directions, and one of these is the rate $\omega(t) = \dot{\alpha}(t)$. In practice this measurement is not perfect, and a typical problem is calibration errors, which means that it does not give the value 0 despite the device being completely still. If we let $\omega_m(t)$ denote the gyro measurement, and assume a constant calibration error b we have

$$\omega_m(t) = \dot{\alpha}(t) + b \tag{3.1}$$

A first task in the lab will be to find an approximate value of b for your device. If we know this calibration error, we can simply subtract it from the measurements. We call this estimate of the calibration error $\hat{b}$.

The gyro measurement of the angular rate can be exploited for more. We need the tilt-angle $\alpha(t)$, and exploit the fact that $\alpha(t) = \int_0^t \omega(\tau)d\tau + \alpha(0)$. In other words, integrating the measured angular rate gives us the angle. This works in theory, but in practice we use the measured angular rate, and we don't know the initial angle $\alpha(0)$ but have to set

[1] A concept not covered in the courses TSRT21 and TSIU61

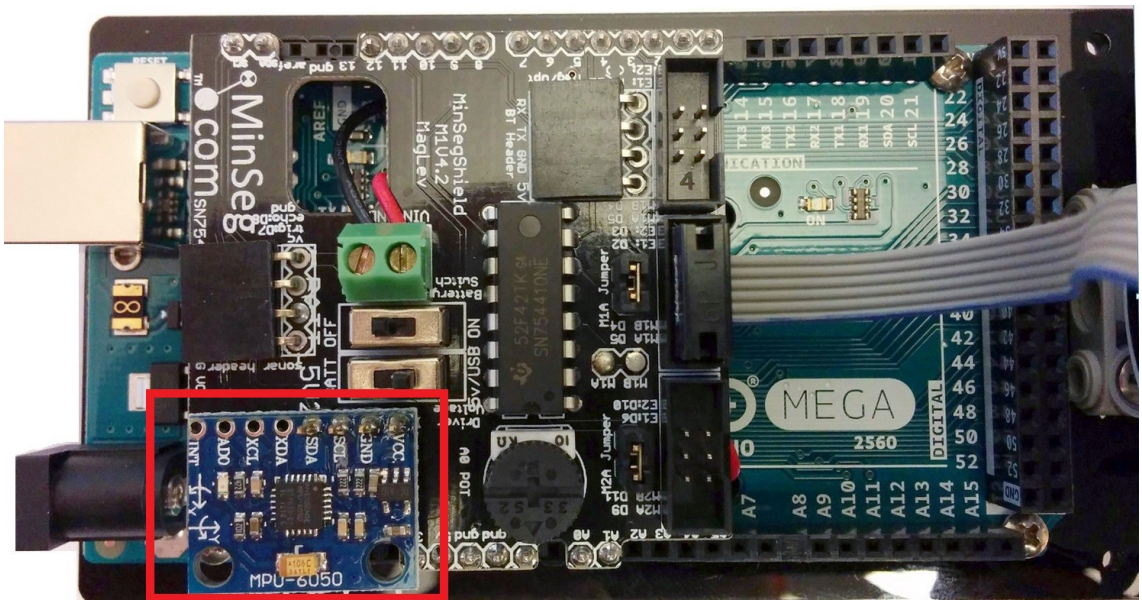


Figure 3.2: The accelerometer and gyro are placed on the blue board (MPU 6050) on the bottom left of the figure. The sensors are (with the setup in this lab) capable of measuring up to $4g$ acceleration and an angular rate of $250°/s$. The blue board combining an accelerometer and a gyro is called an IMU (inertial measurement unit). Retail price in the order of 50SEK. An IMU with similar performance was at least 100x as expensive 20 years ago, and 10 times as large.

it to 0. Let $\alpha_g(t)$ denote the angle estimate obtained by integrating the calibrated gyro measurement $\omega_m(t) - \hat{b}$

$$\alpha_g(t) = \int_0^t (\omega(\tau) + b - \hat{b}) d\tau = \alpha(t) - \alpha(0) + (b - \hat{b})t \tag{3.2}$$

Not only is the estimate wrong due to the unknown initial condition, we will also have an error in the angle estimate which grows with time, unless we have managed to calibrate the device perfectly (never possible in practice) and the initial angle really was $\alpha(0) = 0$. Hence, we cannot trust this angle estimate on a long time-scale. This phenomena is called linear drift, and the approach to estimate the angle is called dead-reckoning. It does however detect *changes* in the angle very well.

Another approach to obtain a measurement of the angle is to look at accelerometer signals, called $a_x(t)$, $a_y(t)$ and $a_z(t)$.

The accelerometer measures forces on the body causing accelerations. In practice this means that when the device is completely still, there should be a total acceleration of $g = \sqrt{a_x(t)^2 + a_y(t)^2 + a_z(t)^2}$. The three measurement directions are aligned with the geometry of the MinSeg in the way that when the MinSeg is lying flat on the table

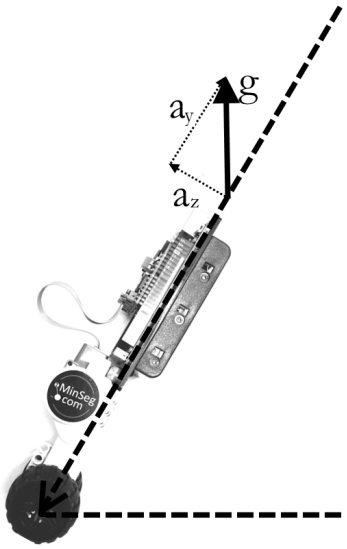


Figure 3.3: The acceleration relative to free-fall will be picked up in a stationary situation on the two accelerometer signals $a_y(t)$ and $a_z(t)$ allowing us to compute the tilt-angle θ

($\alpha(t) = -\pi/2$), $a_z(t) = -g$ and all other accelerations are 0, and when the MinSeg is standing straight up ($\alpha(t) = 0$) we have $a_y(t) = -g$ with all other measurements 0. By geometry, it follows that the tilt-angle estimated from the accelerometer is given by $\alpha_a(t) = \arctan(a_y(t)/a_z(t))$. However, this reasoning only works when the MinSeg is completely still. When the MinSeg is moving around, this relation no longer holds, and the more rapid the movements are, the worse the estimate will be.

We thus have two estimates of the angle, $\alpha_g(t)$ computed from the gyro, and $\alpha_a(t)$ computed from the accelerometers. The estimate $\alpha_g(t)$ can not be trusted on a long time-scale as it has a slowly growing error. On a short time-scale it works well though and measures *changes* in the angle very well. The signal $\alpha_a(t)$ on the other hand has complementary properties. If the MinSeg is relatively still, the estimate can be trusted, but if it moves too quickly, the signal is no longer valid and is disturbed by the movements. Another way to say the same thing is that there are low-frequency errors on $\alpha_g(t)$ and high-frequency errors on $\alpha_a(t)$. A clever way to combine these two signals is a so called complementary filter which filters out the trustworthy parts and sums them up. Let us denote this final estimate $\alpha_c(t)$ where c stands for complementary, and introduce a low-pass filter $L(s)$ and high-pass filter $H(s) = 1 - L(s)$. In Laplace notation, we have

$$\alpha_c(s) = H(s)\alpha_g(s) + L(s)\alpha_g(s) = (1 - L(s))\alpha_g(s) + L(s)\alpha_a(s) \tag{3.3}$$

The filter used in this lab is $L(s) = \frac{1}{s/3+1}$, i.e., a filter with bandwidth 3 rad/s (or time-constant 0.33s). Hence, movements occurring on a time-scale much faster than 0.33s is predominantly taken from the gyro (quick changes), and slower movements are mainly extracted from the accelerometer signal.

The details concerning the implementation and tuning of this complementary filter is

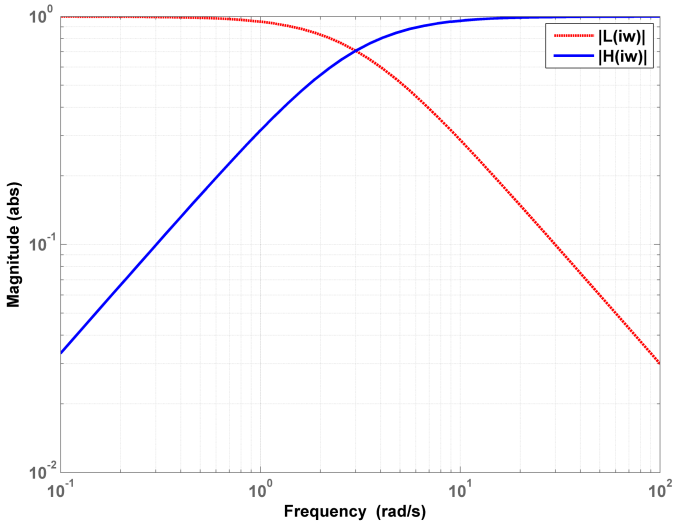


Figure 3.4: Amplitude gains of the low- and high-pass filters used to remove noise on the angle estimated from accelerometer, and removing slowly varying components from the gyro-based angle estimate.

not part of the lab.

Position and velocity measurements

To obtain $z(t)$ and $\dot{z}(t)$ we need some geometry. Consider the case when the MinSeg is kept at a constant tilt-angle $\alpha(t)$ (such as straight up) and the wheel rotates $\theta(t)$. The center of the wheel will then move $R\theta(t)$ along the surface, where R is the radius of the wheel. Similarly, consider the case when the wheel is kept fixed to the body, but the whole MinSeg is tilted $\alpha(t)$ with the wheel rotating on the surface (for example from standing straight up to lying down). As the wheel is fixed to the body, the wheel center will once again move $R\alpha(t)$ along the surface. Combining this, we have

$$z(t) = R(\alpha(t) + \theta(t)) \tag{3.4}$$

The velocity of the center of the wheel is consequently given by

$$\dot{z}(t) = R(\dot{\alpha}(t) + \dot{\theta}(t)) \tag{3.5}$$

The angular velocity $\dot{\alpha}(t)$ is available from the gyro, while $\dot{\theta}(t)$ must be created by numerically differentiating the wheel encoder signal. With that, we have all necessary signals required for the state-feedback control law.

Preparation 7 *Assume the worst possible measurement error of the motor angle* $\theta(t)$ *is* $\pm 0.5°$*, and that we create an estimate of* $\dot{\theta}(t)$ *using a simple Euler approximation* $(\theta(t) - \theta(t - T_S))/T_S$ *where the sampling-time* T_S *is 0.006 seconds. The wheel radius* R *is* 0.02 *meter. Assume the measurement of* $\dot{\alpha}(t)$ *is perfect with no error. Show that the error in*

the estimate of the velocity $\dot{z}(t)$ can be up to 0.058 *m/s due to the measurement error.* ***Hint:*** *If $\theta(t)$ and $\theta(t-T_S)$ each can be measured with a precision of $\pm 0.5^\circ$, what is the worst-case error in the difference $\theta(t)-\theta(t-T_S)$?.*

3.3 State-space model

By modelling the pendelum mechanics in combination with the the electrical motor, one arrives at a nonlinear model describing the robot. After linearization in the stationary point corresponding to standing straight up, a linear approximation is given by

$$\ddot{z} = -703\dot{z}(t) - 5.2\alpha(t) + 14.8\dot{\alpha}(t) + 33.2u(t) \tag{3.6}$$

$$\ddot{\alpha} = 3135\dot{z}(t) + 58.4\alpha(t) - 65.8\dot{\alpha}(t) - 148u(t) \tag{3.7}$$

Preparation 8 *Define the state vector*

$$x(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix} = \begin{bmatrix} z(t) \\ \dot{z}(t) \\ \alpha(t) \\ \dot{\alpha}(t) \end{bmatrix} \tag{3.8}$$

and output of interest $y(t) = \alpha(t)$. Derive the state-space matrices A, B, C, D *in a state-space model of the MinSeg $\dot{x}(t) = Ax(t) + Bu(t)$, $y(t) = Cx(t) + Du(t)$ using 3.6 and 3.7. Remember $\dot{x}_1 = \frac{d}{dt}x_1(t) = \frac{d}{dt}z(t) = \dot{z}(t) =$?...*

Preparation 9 *Based on physical insight on the uncontrolled MinSeg, which statement must be true?*

1. *At least 1 eigenvalue of* A *has a positive real part.*
2. *All eigenvalues of* A *have negative real parts.*

Task 1 (Check model) *Double-click the icon* ***Copy minseg files*** *on the Desktop. Template files will be copied to a directory locally on the computer, and a link to this directory will be created on the Desktop. Open this directory and go to the directory* ***Balance****.Open and run* ***labsetup.m*** *in the* MATLAB *editor. Make sure you understand all the variables which are defined. Check that the definition of* A, B, C, *and* D *are consistent with your model in preparation 8 (after you have run the file, the variables are available in the workspace, so you can simply type the name of the variable in the command window, and it will be displayed)*

3.4 Measurements and sensor calibration

Our first step is to calibrate the sensors in order to remove the main calibration errors on the gyro signal and accelerometer signals.

Task 2 (Calibration of gyro) *Open the file* `template1.slx`. *Take some time to understand the general logic. Your first task is to calibrate the gyro sensor. Assemble the robot as illustrated on the front-page and place the robot with the battery holder down on the table. Confirm USB-cable is attached, compile, download and start the code by pressing the green run button.*

In the block ***Gyro calibation****, compensate for the calibration error as described in the text in the model. How large is the calibration error (rad/s)?*

Task 3 (Detect IMU version) *In the block* ***Angle estimate from accelerometer****, setup the two gains in the yellow blocks for the IMU version (MPU-6050 or ITG/MPU) used on your board, as described in the text next to the two gains. Which version do you have?*

Task 4 (Calibration of accelerometer) *The accelerometer is typically not mounted precisely straight on the board, which will give a slightly wrong angle estimate. This can also be seen as a calibration error. In the block* ***Angle estimate from accelerometer****, compensate for the the calibration errors in the* ***two*** *accelerometer signals as described in the model. How large are the two calibration errors* (m/s^2)*?*

Task 5 (Check angle estimate) *Study the final angle estimate and various raw and filtered versions available in a plot in the block* ***Complementary filter****. Tilt the robot from flat on the table* $(-\pi/2)$ *to* 0 *(standing straight up) and confirm that it is reasonably calibrated. When you move it around, you should see*

> ***Yellow:*** *The estimate we use. Smooth, correct angle and reacts quickly.*
> ***Blue:*** *Low-pass filtered* $\alpha_a(t)$*, smooth, correct angle statically, but lags behind.*
> ***Red:*** *High-pass filtered* $\alpha_g(t)$*, moves around 0 statically, reacts quickly.*
> ***Green:*** *Raw* $\alpha_a(t)$*, correct angle statically, reacts quickly but noisy.*
> ***Purple:*** *Raw* $\alpha_g(t)$*, incorrect angle statically, reacts quickly.*

3.5 Safety first!

Before we start to develop and test the balancing controller, we introduce a safety feature which will force the motor to turn off if the absolute value of the tilt angle is too large.

To reduce clutter with too many wires in the model, we use so called **From** and **Goto** ports. Instead of drawing a wire from point A to point B, one can draw a wire from A to a **Goto** port and give it a label, and then draw a wire from a **From** port to the point B. As you can see in Figure 3.5, all inputs to the controller blocks are immediately directed to three labeled **Goto** blocks. If we need these anywhere, we just add a new **From** port at that point. Of course, if you want to draw wires, you can do this. It will be rather messy in the end though.

Task 6 (Add safety switch) *Implement the model in Figure 3.5. Our goal is to let through the lower signal when the tilt-angle* $|\alpha(t)|$ *is sufficiently small, and send through the upper signal (*0 *volts) when the angle is beyond our safety zone. Effectively, we want the robot to shut off if it is tilted too much and we cannot save it. Later on the lower input will be our feedback control signal, but for now we simply use a constant signal* $u(t) = 2$ *to see if the safety switch works.*

- *The* ***Switch*** *is found under* ***Signal routing****.*
- *The* ***abs*** *operator is found under* ***Math operations****.*
- *The* ***From*** *port is found under* ***Signal routing****.*
- *The* ***constant*** *blocks are found under* ***Commonly used*** *(or copied).*

Set the label in the ***From*** *block to* ***alpha****. In the* ***Switch****, set the threshold to* ***pi/6*** *(*30°*).*

In terms of standard programming, the switch implements the code

if $|\alpha(t)| > 30^\circ$
$u(t) = 0$
else
$u(t) = 2$
end

Remember to save your model after every change!

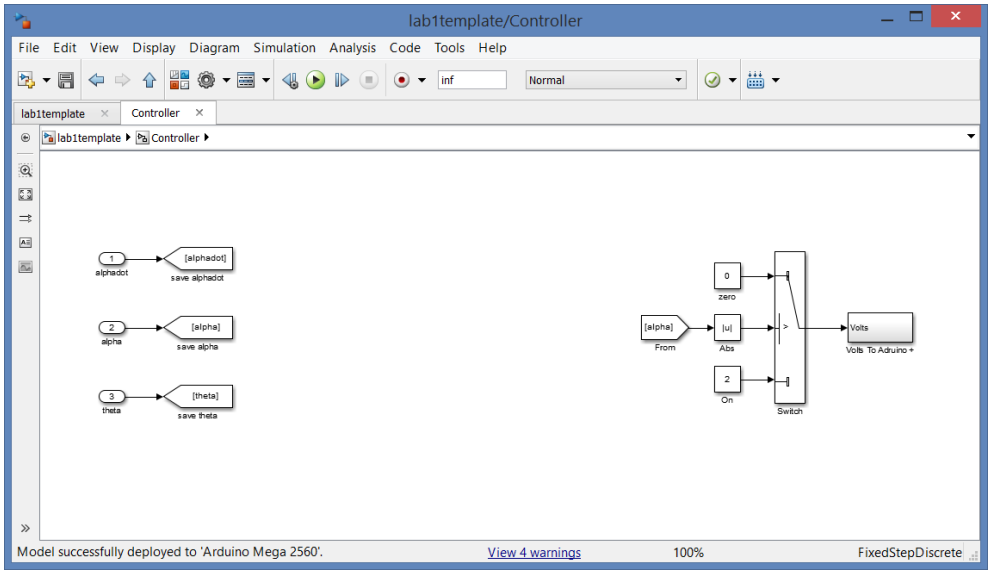


Figure 3.5: Safety switch to ensure robot is turned off if tilt-angle becomes too large. Note the use of **From** and **Goto**. When $\alpha(t)$ enters the controller block, it goes to a **Goto** port with the label **alpha**. When we need the value of $\alpha(t)$ in the model, we pull it from a **From** port, and avoid a wire going all over the model. Do not be confused by the fact that SIMULINK uses the name u in the display of the function block for the absolute value. It is simply SIMULINK standard that inputs to functions are displayed as u.

Task 7 (Test of safety switch) *Run your* SIMULINK *model. When the robot is close to straight up, the wheels should start to move, and when the angle is too large, it should shut off. Note that this also is a way to check that your angle estimator works. Does it turn off when the angle is large enough, in a similar way when tilting both forward and backward?*

3.6 P-control

At first, one might think it is trivial to balance the robot (and similarly a SegWay). A common misconception is to think that we can stabilize it by driving forwards if the robot is tilting forwards, and drive backwards if it is tilting backwards. In control language, this is a P-controller as the input is proportional to the controlled variable,

$$u(t) = K_P \alpha(t) \tag{3.9}$$

The reference signal is zero, and with the sign-conventions in the model and assumed positive K_P, it appears reasonable to have positive input voltage if the tilt-angle is positive (positive input generates torque in clock-wise direction on wheels and generating acceleration in the positive z-direction as explained in Figure 3.1). Make sure you understand this and agree from a simple intuitive physical point of view!

Task 8 (Setup for normal mode) *Change sample-time **TS** to* 0.006 *in the script **labsetup** and run the script. With this fast sampling (used during stabilization), we will no longer be able to run in external mode and look at signals, so the mode has to be changed to **Normal** as indicated in Figure 3.6.*

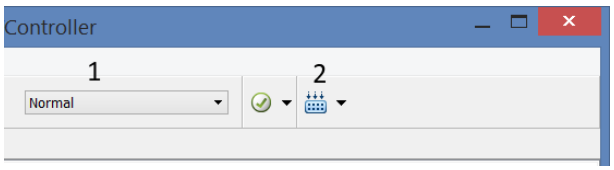


Figure 3.6: When in stabilization mode with fast sampling, we must set it to **Normal** mode (1) and deploy to robot (2) without communication when running the controller.

Task 9 (P-controller) *Implement a P-controller* $u(t) = K_P \alpha(t)$ *(i.e., change your controller block which currently has a constant input signal* $u(t) = 2$ *and implement* $u(t) = K_P \alpha(t)$ *instead). Start with the gain* $K_P = 100$.

*Compile, download and start the controller by pressing the **Deploy to hardware** button indicated in Figure 3.6 with (2) (you will no longer use the green run button). Raise the robot and see if it works (**hold your hands around it!**). Try some other gains to see if you can improve the behavior (you must deploy new code after every change). What happens?*

Task 10 (P-controller theory) *How does your result relate to **Preparation 5**?*

As you will note, it is impossible to stabilize the robot using the tilt-angle only. A more advanced controller is necessary. We predicted this in the preparations on a simplified model, but we can see this using the full model also. The setup is described by

$$\dot{x}(t) = \mathrm{A}x(t) + \mathrm{B}u(t) \tag{3.10}$$

$$u(t) = \mathrm{K}_p\alpha(t) = \begin{bmatrix} 0 & 0 & \mathrm{K_P} & 0 \end{bmatrix} x(t) \tag{3.11}$$

so the closed-loop model will be

$$\dot{x}(t) = (\mathrm{A} + \mathrm{B}\begin{bmatrix} 0 & 0 & \mathrm{K_P} & 0 \end{bmatrix})x(t) \tag{3.12}$$

Hence, we can easily check the eigenvalues of the closed-loop model.

Task 11 *Compute the eigenvalues for the closed-loop systems you tested. You can do this in* MATLAB *with* `eig(A + B*[0 0 Kp 0])`*. Try some different* $\mathrm{K_P}$*. Which important property do you see?*

It is possible to stabilize the angle using a PD-like controller, i.e., taking $\dot{\alpha}(t)$ into account also. However, it is trickier than what one would believe, as any remaining calibration error in the angle will lead to problems which has to be taken care of. We will not proceed in that direction, but instead go straight to a more advanced but more general state-space approach.

3.6.1 Computation of position and velocity

To implement the state-feedback controller, we must have all states available. We have estimates of $\dot{\alpha}(t)$ and $\alpha(t)$ from the gyro and the complementary filter, but $z(t)$ and $\dot{z}(t)$ are missing. A general approach is to develop an observer[2] of the states, but for simplicity we use a more direct approach where we use an Euler approximation together with (3.4) and (3.5) to obtain estimates of $z(t)$ and $\dot{z}(t)$ based on the estimates of $\alpha(t)$, $\dot{\alpha}(t)$, and the measurement of the wheel angle $\theta(t)$

Task 12 *Add code in the **Controller** block to compute the velocity $\dot{z}(t)$ according to equation (3.5), as illustrated in Figure 3.7. Angular rate $\dot{\alpha}(t)$ is already available, but the rotational velocity $\dot{\theta}(t)$ of the wheel has to be computed using an Euler approximation from the encoder value $\theta(t)$, as we have done in previous labs. The result is kept unconnected for now and will be used later.*

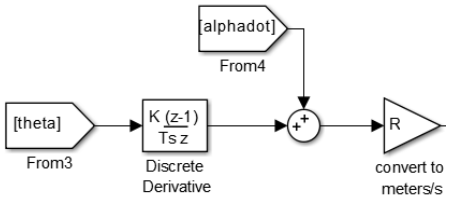


Figure 3.7: Computation of $\dot{z}(t) = R(\dot{\alpha}(t) + \dot{\theta}(t))$.

Task 13 *Add code in the **Controller** block to compute the position $z(t)$ according to equation (3.4). The result is kept unconnected for now and will be used later.*

[2] A concept not included in the courses TSIU61 and TSRT21

3.6.2 State-feedback and pole-placement

The important concept in this lab is that we are going to compute controller gains by using a model.

The four states $z(t)$, $\dot{z}(t)$, $\alpha(t)$,$\dot{\alpha}(t)$ describe the current state of the robot, and it is natural to think that a controller which uses all this information, instead of simply the angle $\alpha(t)$, should be able to perform much better. However, the complexity in trying to manually guess or tune how these 4 states should be used in a feedback-law would lead to hours of testing. Instead, a structured approach is required, and pole-placement is one such model-based approach.

A simple control structure is a linear control law. With the four states, denoted $x(t)$ when placed in a column vector, and a reference angle $r(t)$, a natural control-law is

$$u(t) = -l_1 z(t) - l_2 \dot{z}(t) - l_3 \alpha(t) - l_4 \dot{\alpha}(t) + l_0 r(t) = -\mathrm{L}x(t) + l_0 r(t) \tag{3.13}$$

The negative sign is simply a sign-convention used in the control field. If we use this control law on the system $\dot{x}(t) = \mathrm{A}x(t) + \mathrm{B}u(t)$, the closed-loop system is given by

$$\dot{x}(t) = (\mathrm{A} - \mathrm{BL})x(t) + \mathrm{B}l_0 r(t) \tag{3.14}$$

Since stability and transient performance of the closed-loop system is determined by the poles of the closed-loop system, and poles are equivalent to the eigenvalues in the state-space model, all we have to do is to come up with a choice of L such that the eigenvalues of $\mathrm{A} - \mathrm{BL}$ have suitable properties. The most direct method to do this is *pole-placement* (there are alternative more advanced methods available).

In MATLAB, if we want to compute a matrix L to place the 4 poles (we have 4 states in the model, hence 4 poles), we use the command **place**. For instance, if we want one pole in -1, one in -2, and a complex pair in $-2 \pm i$, we write

```
L = place(A,B,[-1 -2 -2-i -2+i])
```

Hence, instead of manually trying to guess which values the feedback gains l_1, l_2, l_3 and l_4 should have, we propose a reasonable placement of the poles. Trivially, all of them should have negative real part (otherwise the closed-loop system is unstable). Beyond that, it is not obvious. Typically, one has to resort to some physical insight on the time-scales and typical behavior in a well-behaved closed-loop system. In some applications, engineering experience gives direct hints on what suitable pole locations are. Importantly though, coming up with reasonable placements of the poles is easy compared to trying to guess reasonable values on L directly.

In our case, we roughly speaking have two different time-scales. The core stabilization of the tilt-angle happens on a very fast time-scale (milliseconds), meaning that the associated closed-loop pole will be far into the left half plane. The stabilization of the

position of the robot happens on a much slower time-scale (seconds), meaning that some poles are much slower and closer to the origin.

As a first try, try placing one pole around -0.1 to -1 (the slow) and one pole at -500 to -1500 (the fast), and the remaining between -5 and -10.

Task 14 (Compute feedback gains) *Add code to **labsetup** to compute a state-feedback gain, and run the file*

```
wanted = [? ? ? ?]      % We want closed-loop poles here
L = place(A,B,wanted)   % Compute feedback gains
eig(A-B*L)              % Check that we achieved desired poles!
```

*Note that there is a limitation in **place** in that you cannot have two identically placed poles. Where did you place the poles, and what is the resulting feedback gain* L*?*

Task 15 (Understand L**)** *Look at the values of the matrix* L *and recall that the control law is* $u(t) = -\mathrm{L}x(t)$ *(the reference is* 0*). Let us try to understand the logic in the control law, in particular the sign of some of the gains, and see if it matches your intuition. Remember, the controller aims to to drive all states to* 0*, and recall how an input generates torque on the wheels and the definition of angles in Section 3.1.*

1. *When the angle* $\alpha(t)$ *is positive and all other states are zero, show that the robot will accelerate to the right. Does this make sense intuitively?*

2. *If the position* $z(t)$ *is positive and all other states are 0, show that the robot initially will accelerate to the right (away from the target position!). Does this make sense intuitively? Hint: if you move to the right, in which direction will the MinSeg fall...*

Task 16 (Simulate closed-loop system) *Since we have the model, we can simulate and see how the closed-loop system will behave, before we actually test in practice. We create a model from the reference angle (with* $l_0 = 0$*) to the position* $z(t) = x_1(t)$ *(type **help ss** if you are unfamiliar with the command **ss**)*

```
Gcz = ss(A-B*L,B*0,[1 0 0 0],0);
```

We can for example simulate to see how far the robot will move when it stabilizes the robot from an initial condition where the robot angle is 10°*. To simulate the system from a given initial condition, we use the command* ***initial****.*

```
x0 = [0 0 10*pi/180 0];
initial(Gcz,x0)
```

A plot will be generated which shows a simulation of the position $z(t)$*. How far does the simulation predict that the robot will move when catching the initially tilted robot using your control law?*

Time to implement the controller! All we have to do is to implement the computation $u(t) = -l_1 z(t) - l_2 \dot{z}(t) - l_3 \alpha(t) - l_4 \dot{\alpha}(t) = -\mathrm{L}x(t)$. All states are available in the **Controller block** now, the gains have been computed, so everything is available. How you implement the computation is up to you. Two alternatives are shown in Figure 3.8. The right alternative uses a direct scalar description, while the left figure shows how you first can combine the 4 states in a column vector using a **Mux** (found under **Commonly used blocks**) followed by a suitable matrix multiplication (change the field **Multiplication** in the **Gain** to **Matrix K*u**)

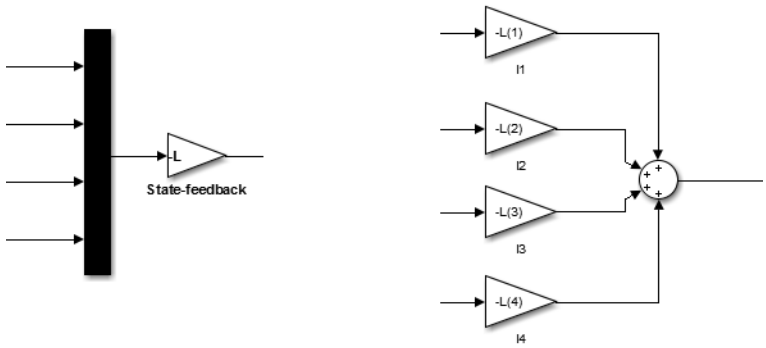


Figure 3.8: Alternative ways to represent $u(t) = -\mathrm{L}x(t)$. Coming from the left should be states in suitable order, and exiting on the right is the computed input which should be connected to drive the motors. Note that you can use the variable L in the blocks.

Task 17 (Implement state-feedback) *Implement the state-feedback controller. Make sure you use the correct states and correct sign. Don't copy numbers manually, but use the variables directly in the blocks.*

Task 18 (Deploy!) *Moment of truth. Deploy the controller. Stand the robot up, hold it carefully, and see if it stabilizes. It wants to move away a bit initially (we will understand this later) so let it move around while you support it gently.For some pole-placements it may happen it wants to move away so far that it runs out of cable, and then you will have to lift it up and move it back quickly. Sometimes it can be convenient to temporarily turn off (and reset) the MinSeg while standing up, and you can do that by pressing the small button in the top right. Does it stabilize? (If not even close, check your computation of $u(t)$ and check that all signs are correct and that you using using the correct states. Hold it in the air, do the wheels start turning in the correct direction when you tilt it).*

Task 19 (Improve) *Try to improve the performance by testing other pole-placement configurations. There is no best answer, and the model is only an approximation so theoretical predictions are not necessarily perfect. A typical way to calm the controller down is to make the slow pole slower (this pole is connected to the position $z(t)$, hence if slower the controller will not put as much effort on trying to move the MinSeg to the correct position. However, the slow pole also influences the wandering problem.)*

Task 20 (Practical performance) *Assuming the model of the robot is correct, theory tells us that the robot is asymptotically stable and all states should converge to zero. Your robot will most likely continue to move around. What could the causes be?* ***Hint:*** *For one reason, think of preparation 7. Additionally, what do you know about the gyro signal after it has been calibrated, is it zero when still? For another reason, check the wheels (are they firmly attached to motor?). Finally, what did we learn about the input to the DC-motor when the requested voltage was small or huge on the PID lab?*

When we design a controller based on a model, one might think that it only will work when the real system behaves precisely as the model predicts. Having a margin for model error is called robustness, and we will now practically test the robustness by changing properties of the real system, without changing the model or control law. Use the controller tuning you were most satisfied with above.

Challenge 1 (Small wheels) *There should be extra wheels available in the lab. Change your wheels to a smaller set of wheels, and see if the controller still stabilizes the system. Is your controller robust enough?*

Challenge 2 (Large wheels) *Change your wheels to a larger set of wheels, and see if the controller still stabilizes the system. Is your controller robust enough?*

3.6.3 Estimating non-zero balance angle using integral action

If we start the robot at a particular position, it should go back to that initial point as it corresponds to $z(t) = 0$ (the closed-loop system should be asymptotically stable which means all states converge to 0 if there is no reference). What you typically see is that the robot wants to move to some other position. This can be explained by the fact that the robot is not at balance at 0°. If you carefully look at the robot when balancing, you will see that it is slightly tilted backwards. If the robot is standing still, we know all derivatives are 0, and the input must be zero. Hence, if the robot ends up in a steady-state position $z^\star$ at balance angle $\alpha^\star$, it must satisfy

$$u(t) = 0 = -l_1 z^\star - l_3 \alpha^\star \tag{3.15}$$

Preparation 10 (Theoretical steady-state position) *Given a feedback matrix* L *and (for example)* $\alpha^\star = -2^\circ = -2\pi/180$*, what position would theory predict it will end up in?*

Task 21 (Practical steady-state position) *With a controller you are satisfied with and the original wheels, hold the reset button (small button upper right corner on the board) and place the robot in balancing position. Note where you have placed the robot, release the reset button and let the robot start the stabilization. Can you see that the robot consistently moves to a position away from the initial starting point? How far, approximately?*

If we knew the balance angle $\alpha^\star$, we could simply change the control law to

$$u(t) = -l_1 z(t) - l_2 \dot{z}(t) - l_3 \left(\alpha(t) - \alpha^\star\right) - l_4 \dot{\alpha}(t) \tag{3.16}$$

and the only possible stationary point would be $z(t) = 0, \alpha(t) = \alpha^\star$.

Can we teach the control law what the non-zero balance angle $\alpha^\star$ is, and compensate for this? Yes, by integral action!

If the robot ends up in a non-zero positive position $z(t)$, 3.15 reveals that the robot is tilted slightly backwards at steady-state[3], and we are trying to achieve the wrong balance angle. To find out if we end up in the wrong position, we integrate $z(t)$. Effectively, we define an estimate of the balance angle as $\hat{\alpha}^\star(t) = -\mathrm{K_I} \int_0^t z(\tau) d\tau$. As long as $z(t)$ stays positive, the value of $\hat{\alpha}^\star(t)$ will continue to decrease. As the controller then sees a smaller angle error $\alpha(t) - \alpha^\star(t)$, it can move towards the initial zero position. The only

[3] or the accelerometer angle estimate is poorly calibrated, giving us a wrong value of the actual angle

possible stationary point (nothing changes) is $\alpha = \hat{\alpha}^\star = \alpha^\star$ and $z(t) = 0$ (since $\hat{\alpha}^\star(t)$ would keep changing if $z(t)$ converged to a non-zero value). If we use a too large gain in the integrator, we can easily become unstable or oscillatory, as we overcompensate and update the adjustment too fast.

Hence, the new control law is

$$u(t) = -l_1 z(t) - l_2 \dot{z}(t) - l_3 \left(\alpha(t) - (-\mathrm{K_I} \int_0^t z(\tau) d\tau) \right) - l_4 \dot{\alpha}(t) \quad (3.17)$$

$$= -l_1 z(t) - l_2 \dot{z}(t) - l_3 \left(\alpha(t) + \mathrm{K_I} \int_0^t z(\tau) d\tau \right) - l_4 \dot{\alpha}(t) \quad (3.18)$$

Task 22 (Integral action to the rescue) *Add a discrete-time integrator and rework the control computation to implement 3.18. A suitable gain is around* 0.1 *or so. Don't forget to set the sample-time in the integrator to* -1 *which means it uses the same sample-time as all other blocks. Repeat the experiment where you place the robot standing up at an initial point and start the robot. Does the robot come back to the initial point after some initial movements?*

Task 23 (Integral tuning) *Experiment with a much larger and smaller integral gain (change by a factor of* 10*). How do the different controllers perform?*

The slightly non-zero balance angle is a fixed geometric property of your robot. The value of the integral should converge to a fixed value, and an alternative could have been to figure out this fixed value, and simply use that constant value of $\alpha^\star$ instead of estimating it on-line. However, by estimating the balance angle, we can cope with a changing environment and robot.

3.7 Summary and reflections

Summarize and reflect on what you have seen and learned in this lab.

Question	**Answer**
1. State-feedback control design through pole-placement requires a model	Yes No
2. State-feedback control requires a perfect model	Yes No
3. State-feedback control requires perfect measurement of all states	Yes No
4. For the closed-loop system to be stable	All eigenvalues of A – BL have to be real All eigenvalues of A – BL have to have negative real part All eigenvalues of A – BL have to have positive real part All eigenvalues of A have to be real All eigenvalues of A have to have negative real part All eigenvalues of A have to have positive real part
5. A gyro measures	an acceleration an angle an angular rate (rotational velocity)
5. High-frequency noise can be removed with	a low-pass filter a high-pass filter
6.Slowly varying and constant components in a signal can be removed with	a low-pass filter a high-pass filter

Most unclear to me is still: