ELEC2540: Control Systems Laboratory

Closed-Loop Control Design of an Electronic Compass

University of Leeds
Summer 2018 session

Contents

1 Review

Please **review** the following items before the lab:

- Basic theory of proportional-integral (PI) control
- Basic programming of mbeds

Figure 1: Traditional magnetic compass (left) and compass on a Smartphone with a magnetometer (right)
Source: www.wikipedia.com

2 Introduction

A compass is a device to determine a specific direction. Its most common application is to determine a direction relative to the cardinal directions North, South, East and West. Compasses are widely used for orientation and navigation. The first use of compasses for navigation dates back to the 11th century in China and to the 13th century in Europe and Persia.

The most common type of compass is the magnetic compasses. It consists of a 铁磁性指针 ferromagnetic pointer mounted on a low-friction pivot point, see Figure 1. The pointer 枢轴 then orients itself towards the Earth's magnetic field. As the field lines of the Earth's magnetic field are oriented *approximately* in a geographical North-South orientation, a magnetic compass allows to obtain a relatively accurate estimate of the "geographic North", also called "true North". Depending on where the compass is located on the Earth surface, the deviation between the "magnetic North" provided by a magnetic compass and the "geographic North" may vary. Likewise, if the compass is placed close to another magnetic field, e.g., a machine, this will also affect the accuracy of the obtained "magnetic North".

In recent years, solid-state compasses have become more and more popular in mobile devices, such as smartphones. Solid-state compasses usually consist of a magnetometer, which provides information of the Earth's magnetic field to a microprocessor. In a smartphone, the actual direction can then be displayed by an app, see Figure 1.

In this lab, we will design an electronic compass. To this end, we will use a magnetometer to measure the direction of the Earth's magnetic field. The magnetometer is mounted on a printed-circuit-board (PCB), which serves as our pointer. The PCB is fixed on a servo motor, which we will use to move the pointer. The servomotor is powered by a battery pack. Your main task is to implement and design a P(I) controller, which orients the pointer towards North and, hence, can be used as an electronic compass for orientation and navigation purposes. You will implement this controller on an mbed microcontroller. The mbed is also mounted on the PCB and powered by the battery pack. The overall system is shown in Fig. 2.

1. magnetometer
2. PCB
3. servo motor
4. battery pack
5. mbed

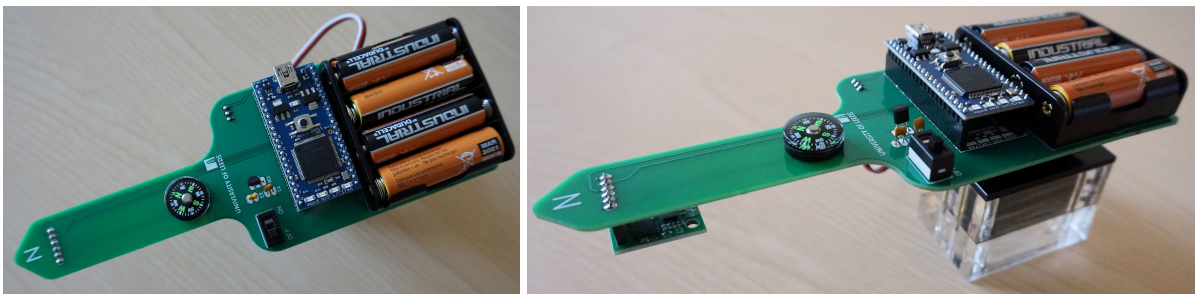

Figure 2: Electronic compass used in the lab

2.1 Tasks

The lab is divided into four main tasks:

1) Control-engineering classification of electronic compass components and signals
2) Compass position control via P control
3) Compass position control via PI control
4) Anti-windup scheme for integral controllers with actuator saturation

Ziegler-Nichols method

?

Important. You need to submit a lab report. The specific tasks to be included in the report are highlighted in the manuscript and summarised in Section 9.

2.2 Learning Objectives

After successful completion of this lab, you should . . .

- . . . be able to classify the role of hardware equipment and physical signals in control-engineering terms;
- . . . implement and tune a P(I) controller by using the Ziegler-Nichols method;
- . . . explain the advantages and need of integral control in many engineering applications;
- . . . implement an anti-windup scheme for an integral controller.

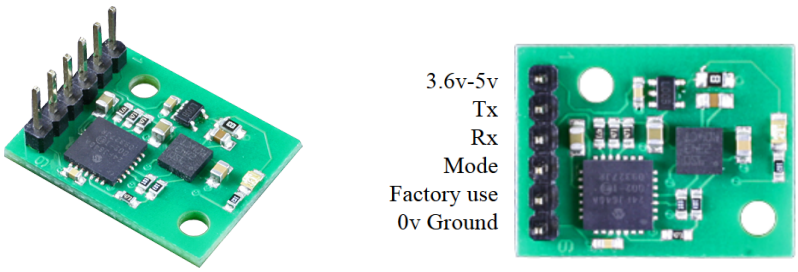

Figure 3: Compass module CMPS11 connected in serial mode
Source: www.robot-electronics.co.uk

3 Equipment and Components Supplied

In this section, the main hardware components used in the lab are introduced.

3.1 Compass Module

The compass module CMPS11 used is a tilt-compensated compass module, see Figure 3. It has a 3-axis magnetometer, a 3-axis gyro and a 3-axis accelerometer. You will only use the magnetometer in this lab. The compass module provides an angle measurement between 0°-359.9° and 0° represents the "magnetic North". The accuracy of the compass module is 3°-4° and it has a resolution of 0.1°.

In our setup, the compass is connected to a printed-circuit-board (PCB) in serial mode (mode pin connected to ground). Then, the angle can be read from the compass using the command "0x13". This command gives back the angle in the range 0-3599 as two bytes, with the first byte being the high byte. The CMPS11 module needs to be powered at 3.6 - 5V and its nominal current is 25mA.

It is important to note that the compass module itself does only provide an angle measurement, but does not tell us visually, where North is. We can, however, read this angle measurement digitally and use it to design a feedback controller, which orients our pointer, i.e. the PCB, to North.

For further details, you may have a look at the data sheet of the compass module, which is available at

https://www.robot-electronics.co.uk/htm/cmps11doc.htm.

3.2 Servomotor and Battery Block

Servomotors are common actuators with high precision in position feedback. In our case, we use a rotary servomotor to drive the pointer (PCB). The servomotor is controlled via a puls-width-modulation (PWM) signal and can be powered with 4.8V-6V. We use a 5V battery block as power source in the lab.

For further details, you may have a look at the data sheet of the servomotor, which is available at

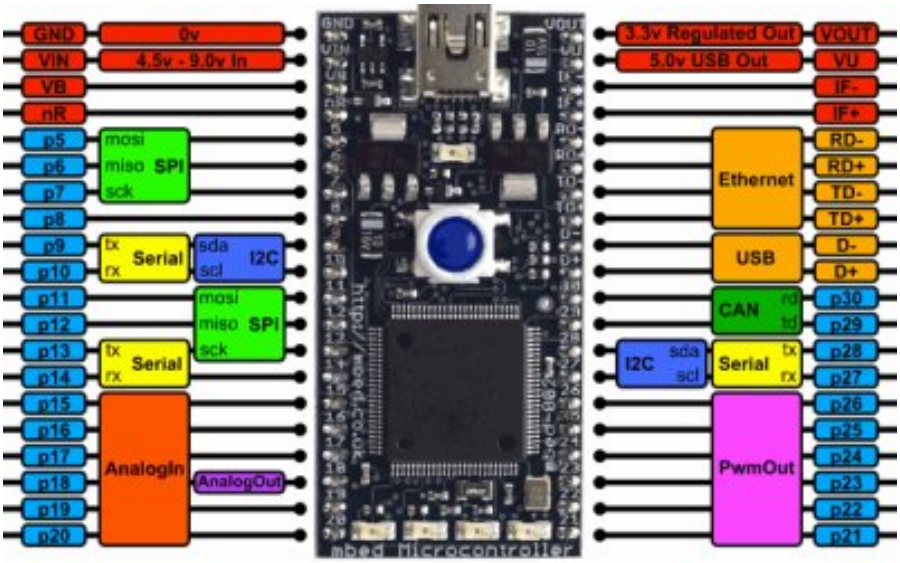

Figure 4: The mbed LPC1768
Source: https://developer.mbed.org/platforms/mbed-LPC1768/

3.3 Microcontroller

We use a mbed LPC1768 mbed microcontroller in this lab. The mbed and its commonly used interfaces and their locations are shown in the pinout in Figure 4. The mbed is also powered by the 5V battery block. For further details, you may have a look at

https://developer.mbed.org/platforms/mbed-LPC1768/

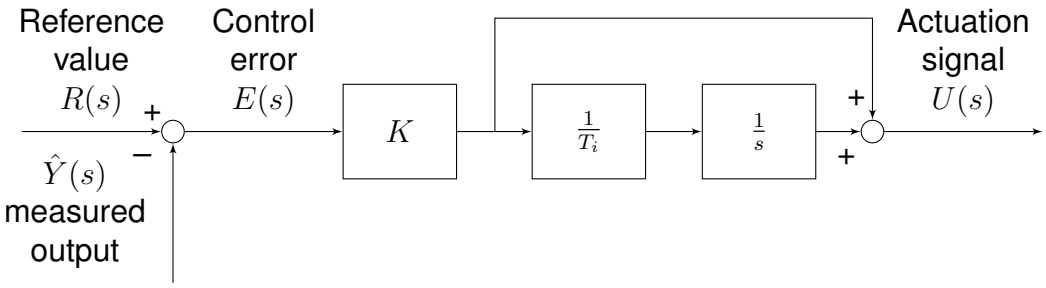

Figure 5: PI controller in Laplace-domain

4 Proportional-Integral (PI) Control

PI controllers are a type of feedback controllers, which are very often used in industrial control systems. In many cases, the control performance can be improved by adding a derivative (D) part to the PI controller, yielding a PID controller. But for the purpose of this lab, a well-tuned PI controller provides a good enough control performance.

4.1 The Standard PI Controller in the Time- and Laplace-Domain

In the time-domain, the standard form of a PI controller is

$$u(t) = K\left(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau\right), \tag{4.1}$$

where

- t is the time,
- $e(t) = r(t) - \hat{y}(t)$ is the control error between the measured plant output $\hat{y}$ and the reference r at time t,
- $u(t)$ the actuator signal at time t, 执行器信号
- K is the (positive) feedback gain,
- T_i is the (positive) integration time constant.

In the Laplace-domain and by denoting the Laplace variable as usual by s, the PI controller in (4.1) becomes

$$U(s) = K\left(E(s) + \frac{1}{T_i s}E(s)\right),$$

with $E(s) = R(s) - \hat{Y}(s)$. The control scheme is shown in Figure 5.

4.2 Digital Implementation of a PI Controller

The aim of this lab is to implement and tune a PI controller to operate an electronic compass. Therefore, the PI controller (4.1) should be implemented on a microcontroller, in the present case on an mbed. Yet, the controller (4.1) is represented in what is called *continuous time*. In a continuous time representation, time is viewed as a continuous variable. Another way of looking at this, is to view a period of time being split into infinitesimally short sections.

However, a microcontroller is not capable of performing calculations in infinitesimally short time intervals. But the microcontroller can perform calculations every certain time interval, typically called sampling time. Therefore, the implementation a PI controller on a microcontroller requires the continuous-time PI algorithm in (4.1) to be *discretised*. There are several methods to do this, but discussing these here would be out of the scope of this lab. Instead, we will discuss one particular common form of a *discretised* PI controller. To this end, the sampling interval is denoted by Δt. Now, assume we are at a point in time $t_k \geq 0$. Then, the control error at t_k is given by $e(t_k) = r(t_k) - \hat{y}(t_k)$. The next point in time at which the microcontroller can provide an updated actuator signal u is

$$t_{k+1} = t_k + \Delta t.$$

Note that for calculating $u(t_{k+1})$, the microcontroller can use the current control error $e(t_k)$ (this is proportional control), but also all previous control errors at all previous sampling times $t_k, t_{k-2}, \ldots, t_1$, where $t_1 = 0$ (this is integral control). Hence, the discrete-time version of the PI controller (4.1) is given by

$$\begin{aligned} u(t_{k+1}) &= K\left(e(t_k) + \frac{1}{T_i}\left(e(t_k)\Delta t + e(t_{k-1})\Delta t + \ldots + e(t_1)\Delta t\right)\right) \\ &= K\left(\underbrace{e(t_k)}_{\text{proportional control}} + \underbrace{\frac{1}{T_i}\sum_{i=1}^{k} e(t_i)\Delta t}_{\text{integral control}}\right). \end{aligned} \tag{4.2}$$

This discrete-time PI controller (4.2) can be implemented on a microcontroller.

Hint: For the implementation, the following equivalent representation of (4.2) may be helpful

$$\begin{aligned} u(t_{k+1}) &= K\left(e(t_k) + \frac{1}{T_i} e_{\text{Int}}(t_k)\right), \\ e_{\text{Int}}(t_k) &= e(t_k)\Delta t + e_{\text{Int,old}}(t_k), \\ e_{\text{Int,old}}(t_k) &= \sum_{i=1}^{k-1} e(t_i)\Delta t. \end{aligned} \tag{4.3}$$

4.3 Some General Design Rules for PI Controllers

As you certainly can imagine, the implementation of the PI algorithm (4.2) is only one step in designing a controller for a specific task. But in order to achieve a satisfactory closed-loop performance, one also needs to select appropriate numerical values for the feedback gain K and the integration time constant T_i. It is obvious that, though many physical and industrial systems can be controlled using PI controllers, in most cases the parameters K and T_i will have different numerical values depending on the physical properties of the system to be controlled. The process of determining these values is called *controller design* or *controller tuning*.

Hence, controller tuning means to adjust the parameters K and T_i, such that the closed-loop system consisting of the plant in feedback with the PI controller (see Fig. 6) satisfies certain performance requirements. Two examples for performance requirements are speed and accuracy of the control system.

Here are some general guidelines for tuning of PI controllers taken from [1].

- Increasing proportional gain K decreases stability.
- Error decays more rapidly if integration time T_i is decreased.
- Decreasing integration time T_i decreases stability.

4.4 The Ziegler-Nichols Method

Proper tuning of PI controllers can be a very challenging task. Therefore, there have been developed a variety of methods which provide a structured and systematic way for tuning. One of the most prominent PI design methods is the one developed by Ziegler and Nichols in 1942 [1]. Even today, the Ziegler-Nichols method or modifications of it are widely used in practice. The main reason for this may be their simplicity.

In the following, the *Ziegler-Nichols method based on the frequency response of the plant dynamics* is introduced based on [1, Chapter 6.2] [1]. This method relies on determining the so-called *ultimate* gain K_u and ultimate period T_u of the system. The specific design steps are as follows.

1. Connect a PI controller as given in (4.1), respectively (4.2), in feedback with the plant.
2. Set the parameters such that the control action is purely proportional, i.e., $1/T_i = 0$ (you can also simply remove the integral part).
3. Set the feedback gain K to a low value, e.g., $K = 0.01$.
4. Increase K slowly until the closed-loop system starts to oscillate consistently.

开始始终摆动

[1] It is said to be based on the frequency response, because the design uses the information of the point, where the Nyquist curve of the systems' open-loop transfer function intersects the negative real axis.

Controller	K/K_u	T_i/T_u
P	0.5	
PI	0.4	0.8

Table 1: Controller parameters for the Ziegler-Nichols method [1]

5. The gain at which this occurs is the ultimate gain K_u.
6. The period of this consistent oscillation is the ultimate period T_u.
7. Ziegler and Nichols now provided some formulas to determine the controller parameters K and T_i from the ultimate gain K_u and ultimate period T_u. These formulas are listed in Table 1.

Attention! The Ziegler-Nichols method is a *heuristic* PI design method, which uses a simple characterisation of the system dynamics. Therefore, it may not necessarily give control parameters yielding an excellent control performance. In fact, it often only provides a moderately good control performance. It has, however, the advantage of providing simple tuning formulas based on direct experiments. It also may provide a good starting point for a more extensive control design procedure.

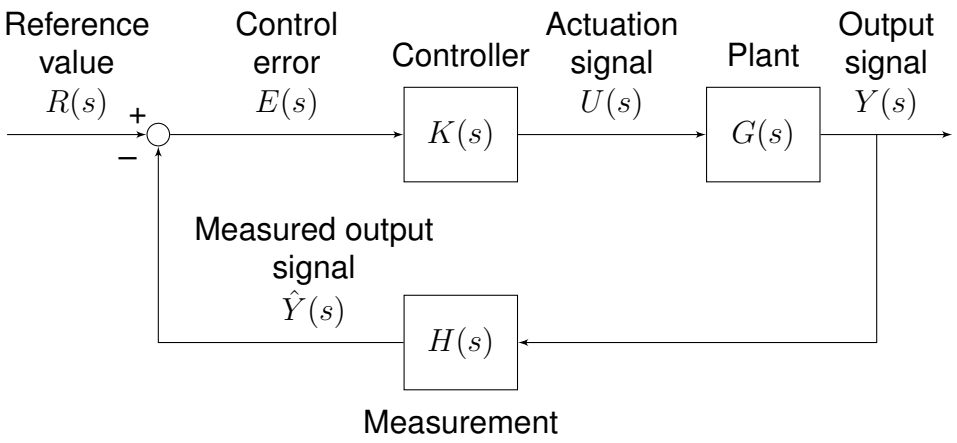

Figure 6: Basic control loop in Laplace-domain

5 Task 1: Control-Engineering Classification of System Components and Signals

Report task. A standard closed-loop control system is shown in Figure 6. Classify the different components and signals of the electronic compass system displayed in Figure 2 in the control engineering terms used in Figure 6, i.e. which components represent the controller, which the plant, what is the output signal, what the actuation signal. . . .

6 Task 2: Compass Position Control via Proportional Control

In this section you will implement and design a P controller to orient the pointer (PCB) of the electronic compass towards North.

2.1) Download the file "compass_code.cpp" from the ELEC2540 VLE. "Right click" link to allow file to be saved to a location indicated. Use "save target as" and "all files" to save it your M drive.

2.2) Connect the mbed to the PC via the USB cable.

2.3) Open a web browser and navigate to "https://developer.mbed.org/". ~~Log-in with your existing credentials. If you don't have an account yet, sign up for one.~~ After logging in, click on the link "Compiler" in the upper-right corner of the page. A new window displaying your compiler workspace should have opened. Click on "Import" and then on the tab "Upload." Click on "Browse" at the bottom of your screen and navigate to the location of the file "compass_lab". Select this file and click "Open". Then click on "Import" on the right-hand side of your screen, select "Import as program" and choose an Import Name, e.g., "compass". Then click "Import". You should now have a new program called "compass" in your Program Workspace. Now, you need to import the mbed library. Right-click on your program name and select "Import Library→From Import Wizard". Search for "mbed" and, after the search is completed, choose the "mbed" library by double-clicking on it or selecting it and then clicking "Import". Now, you are ready to get started!

2.4) Familiarise yourself with the program code "compass_code.cpp".

Report task 1. self-written code; 2. values of your controller parameters; 3. ANSWER

2.5) Implement a P controller with gain K in the function "float updatePI(float pos,float target,float dt)" by using the discretised PI algorithm (4.2). The controller should be able to orient the compass' pointer towards North.

Report task

2.6) Tune the P controller, such that the pointer is oriented towards North. Start with a value for the feedback gain of $K = 0.1$. After adjusting the parameter K, you have to compile your program "compass_code.cpp" and save it on the mbed. Disconnect the USB cable before resetting the mbed to load the new program! Evaluate the control performance. The 4 LEDs on the mbed indicate how close your pointer's position is to the "magnetic North". When all 4 LEDs are on, then the pointer's position is within $\pm 0.5^\circ$ of the "magnetic North" axis. For more details, see the function "feedBack()" in the program "compass_code.cpp". Keep adjusting K until all 4 LEDs are lid up or the plant becomes unstable.

Report task

2.7) What do you observe as you increase the magnitude of K? In particular, comment on how the magnitude of K affects a) the control accuracy and b) the convergence speed to a steady-state.

收敛

Report task

2.8) Relate your observations to the theoretical explanations on P controllers and their steady-state behaviour given in the lecture. Assume that the plant of the electronic compass system can be represented by a stable transfer function $G(s)$, i.e., $G(s)$ has all its poles in the left half-plane.

7 Task 3: Compass Position Control via PI Control

In this section you will add an integral part to your P controller and use the Ziegler-Nichols method to tune the controller parameters.

1. self-written code; 2. values of your controller parameters; 3. ANSWER

Report task

3.1) Add an integral control with integration constant T_i to the function "float updatePI(float pos,float target,float dt)". Use the discretised PI algorithm (4.2).

Report task

3.2) Tune the controller parameters K and T_i by using the Ziegler-Nichols method described in Section 4.4.

Hint: When you have found the ultimate gain K_u, you need to determine the ultimate period T_u. One way of determining the period of a repetitive process is as follows. Take a fixed time interval ΔT, for example $\Delta T = 10$s. Now, count the number ΔN of periodically repetitive events occurring in the time interval ΔT. Then, the period T is given by

$$T = \frac{\Delta T}{\Delta N}.$$

Report task

3.3) Set K and T_i to the values identified in Task 3.2) in your program "compass_code.cpp". Evaluate the control performance. In particular, compare it to the performance of the proportional controller of Task 2.6). If necessary, adjust the control parameters to improve the control performance.

Report task

3.4) Decrease the magnitude of T_i. What do you observe? In particular, comment on how the magnitude of T_i affects a) the control accuracy and b) the convergence speed to a steady-state.

Report task

3.5) Relate your observations to the theoretical explanations on PI controllers and their steady-state behaviour given in the lecture. Assume that the plant of the electronic compass system can be represented by a stable transfer function $G(s)$, i.e., $G(s)$ has all its poles in the left half-plane.

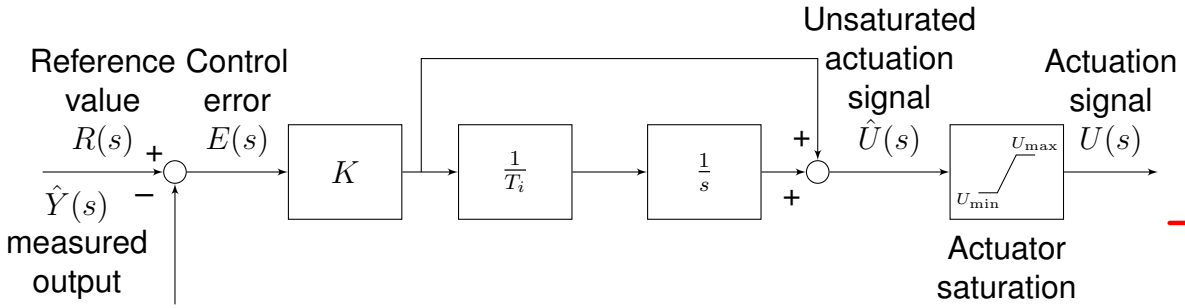

Figure 7: PI controller with actuator saturation. The saturation upper and lower limits are denoted by $U_{\max}$, respectively $U_{\min}$.

8 Task 4: Integrator Windup and Anti-Windup

The servo used to control the compass pointer has a maximum and minimum speed, which can't be exceeded. This means that the actuator is limited. In fact, most real-world actuators are limited (e.g., a valve can only be fully opened or closed, a motor can only generate a minimum or maximum torque). A PI controller with actuator saturation is shown in Figure 7. While the actuator operates within its limits, the system's behaviour can still be analysed using linear control theory. But this is not the case, when the actuator reaches its lower or upper limit. Then the feedback loop is interrupted and the actuator remains at its limit independently of the control error. This is particularly dangerous when an integral feedback term is employed. The reason for this is that, in absolute values, the integral error will keep increasing even so the actuator is already operating at its limit. Therefore, it is often said that the "integrator winds up". As a consequence, the control error needs to have opposite sign for a long time to drive the actuation signal back within its physical limits. Integrator windup can lead to serious performance deterioration, such as overshoot or oscillations, and even instability. Algorithms, which take actuator saturation into account, are usually called "anti-windup". A standard anti-windup scheme is shown in Figure 8.

Report task

4.1) Implement the anti-windup scheme in Figure 8 in the function "antiwindup" in the file "compass_code.cpp". Set $T_t = \sqrt{T_i}$.

4.2) Compare the difference between the system's response with and without anti-windup (to see any remarkable difference, you might need to run the system with large control errors).

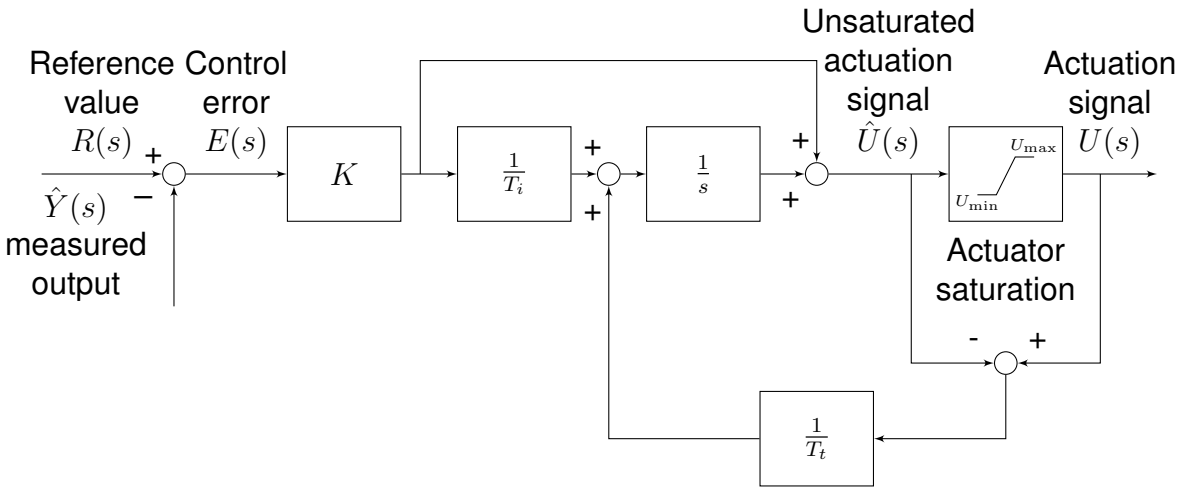

Figure 8: PI controller with anti-windup

9 Report Tasks

- Your report should include answers to the following tasks: 1, 2.5, 2.6, 2.7, 2.8, 3.1, 3.2, 3.3, 3.4, 3.5, 4.1 and 4.2.
- Please include your **self-written code**, whenever this was part of the task!
- Please include the **numeric values of your controller parameters**, whenever their design was part of the task!
- Please keep your answers short and precise (**no more than 200 words per task**)!

References

[1] Karl Johan Åström and Tore Hägglund. *Advanced PID control*. ISA-The Instrumentation, Systems, and Automation Society; Research Triangle Park, NC 27709, 2006.