IN3140/IN4140 - Assignment 4

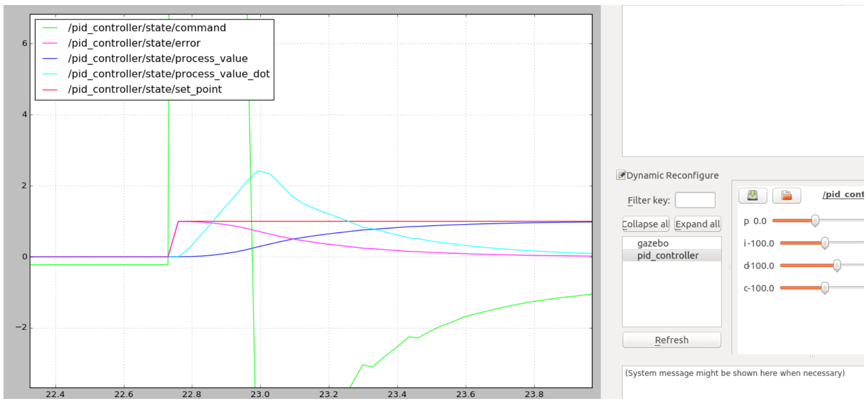
Due: 24 April 2020, 23:59 (24h)

Introduction

In this assignment we will look more closely at control of a robot. We will continue to work with the CrustCrawler robot, and by the end of this assignment you will get to test your controller in a simulation environment modeling the real robot.

We will design controllers for a single joint of the simplified three-link CrustCrawler robot. Since Joint 2 does most of the work, it has the most important controller to tune and will therefore be the one we are working with in this assignment. We can imagine the link of the CrustCrawler robot as an inverted pendulum (Figure 2).

For this assignment you will need to use the virtual machine (VM) with ***ROS*** installed. The VM has all required packages installed, but you may need to download custom assignment packages. You can also use your own computer, but there will be no time allocated for ***ROS*** installation. At the end of this assignment, you will have relevant experience for working with robotic systems in the real world.


/pid_controller/state/command
/pid_controller/state/error
/pid_controller/state/process_value
/pid_controller/state/process_value_dot
/pid_controller/state/set_point
Dynamic Reconfigure
Filter key:
Collapse all
Expand all
gazebo
pid_controller
Refresh
/pid_cont

p 0.0
i -100.0
d 100.0
c -100.0

(System message might be shown here when necessary)

Figure 1: Control of CrustCrawler Joint

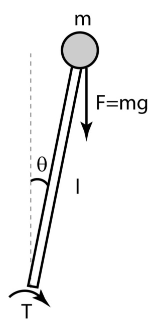


Figure 2: Inverted pendulum model

Task 1: Control Theory (40%)

We begin where we left off in **Task 2 (Dynamics II) of Assignment 3**. There, you derived an equation of motion for Joint 2. We shall now work further with this equation.

a) 10% Make the equation of motion for Joint 2 independent (of other joints). Justify your method.

HINT *Besides figuring out which variables are constant, remember from chapter 7.3 that the quadratic terms represent the effect induced on Joint i by other joints.*

If you are unable to solve this task for some reason related to Assignment 3, you are allowed to use the generic inverted pendulum torque equation in place of τ_2 for all later tasks (by doing so, you will get **no points** for this subtask and make controller tuning more difficult for yourself):

$$\tau_{pendulum} = ml^2\ddot{\theta} - mgl\sin(\theta) \tag{1}$$

where $m = m_2 + m_3$ and $l = L_2 + L_3$.

b) 10% Transform the following independent joint control equation from time domain to Laplace domain:

$$u(t) = J\ddot{\theta}(t) + B\dot{\theta}(t) + D(t) \tag{2}$$

Next, write out the terms in the equation you obtained in (a) that correspond to the terms J, B and D in the above equation.

c) 10% Draw a closed-loop block diagram for equation (2) in Laplace domain, using only simple blocks. Add a PD-controller to the block diagram and derive the transfer function between the input $\theta_d(s)$ and the output $\theta(s)$.

d) 10% With the PD-controller, the closed-loop system is now second order, and hence the step response is given by the closed-loop natural frequency ω and damping ratio ζ. Given the requirements of a natural frequency of 6 and a critically damped system, find values for K_P and K_D.

HINT *See Chapter 6.3 (which you should read thoroughly) in the textbook.*

Next, we will implement the PD-controller and test the values for K_D and K_P that you just calculated. A bit of preparation is in order before we can begin.

Task 2: Setup (5%)

We assume that you have initialized your workspace in accordance with the lectures. The first thing we will need is to download dependencies for the assignment code. We can discover the dependencies of a package by reading the ***package.xml***. Most of our dependencies are already installed, but we still need to download these packages:

```
cd /path/to/your/workspace/src/
git clone https://github.uio.no/INF3480/crustcrawler_simulation.git
git clone https://github.uio.no/INF3480/crustcrawler_pen.git
```

This will download the CrustCrawler simulation packages containing a description of the robot and how to simulate it within Gazebo. The second package is a helper package for our reduced robot arm with a pen attachment.

The next step is to copy the ***pid_assignment*** folder, which is part of the assignment code, into our source directory. **You will find the zipped package on the course page, in the "Mandatory assignments" section.** You should now be ready to simulate CrustCrawler.

a) 5% We will start by testing the simulation. Run

```
roslaunch pid_assignment setup.launch
```

Do not forget to build and source the workspace. You should now see Gazebo starting with the CrustCrawler inside. To run the assignment code launch

```
roslaunch pid_assignment pid.launch
```

This should open an ***rqt*** window where we can tune the controller. If you try to change values now, nothing will happen because the controller is not implemented. **Your delivery on this task should be a screenshot showing the functioning setup.**

Task 3: Simulation I (35%)

We will now work with controlling the position of Joint 2. In the file 'pid.py' you will find the skeleton implementation which is called by the simulation.

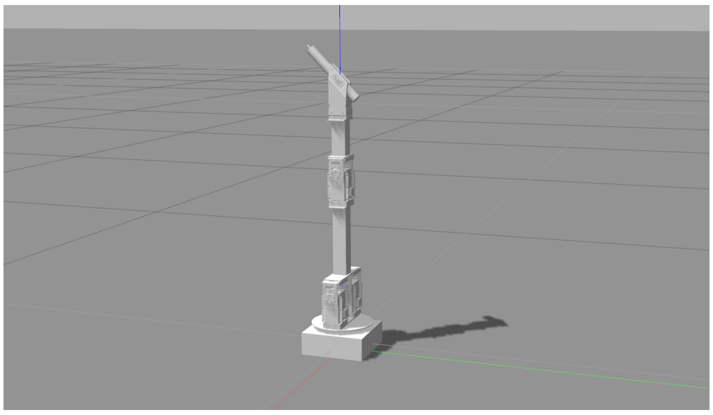


Figure 3: CrustCrawler robot in simulation environment

a) 15% Implement the PD-controller. The return value from the function '*__call__(...)*' should be your computed control effort. Plot the model with the PD-controller. Did the response of the model have a settling time as expected? Why or why not? Include an image of the plot.

HINT *The settling time is the time the response takes to settle withing* 5% *of the steady state value, and is given by the following equation*

$$t_s = \frac{4}{\zeta\omega} \tag{3}$$

b) 10% Tune your controller so that the model has the lowest possible settling time without overshooting. What is the new values of K_P and K_D? What is the steady state error of the system? Include an image of the plot after tuning.

c) 5% Decide if the steady state error in task 3 b) is sufficiently small, or if the controller needs further improvement. Justify your answer.
If the steady state error is regarded as too large, use your knowledge of control theory to implement a controller that will adjust for steady state error, tune it and add an image of the plot.
If the steady state error is regarded as small enough then briefly describe what controller you would have implemented if it wasn't.

d) 5% Expand the closed-loop block diagram and function you made earlier in Task 1, with regards to Task 3c).

Task 4: Simulation II (20%)

Until now, we have worked with a PD-controller. For many industrial purposes, the improvement of the derivative term comes at a cost. In factories, the process value signal is often noisy, and the derivative of noise leads to instability (remember that the derivative action works on the rate of change of the error). In

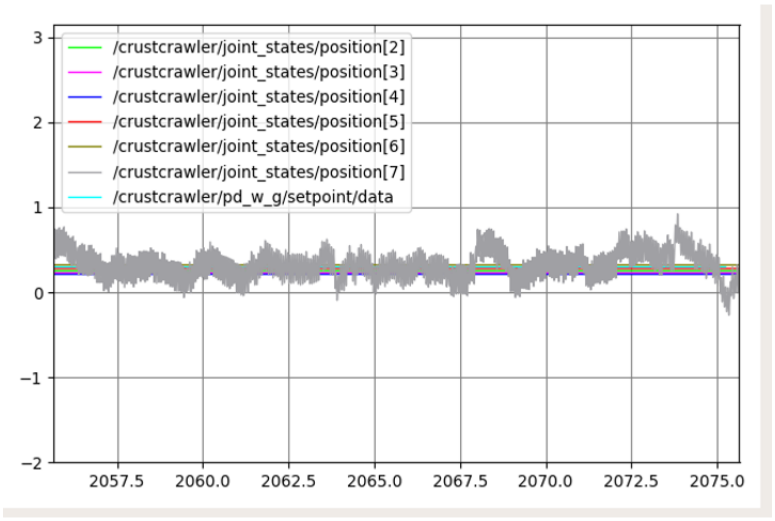


Figure 4: Instability caused by derivative term on Joint 6 of six-link CrustCrawler (with $K_P = 2$ and $K_D = 0.5$)

figure 4 you can see that the derivative action on Joint 6 of the full CrustCrawler robot causes instability. Here, the gray graph is the position of Joint 6, plotted over time.

For processes that do not benefit from the derivative action, a PI-controller is often chosen instead. In fact, it is the most used controller in the industry as well as the focus of this task.

We have already used the dynamics of the system to calculate controller gains, but we are not always so fortunate as to have a model of the system to compute controller gains from. We will now assume that we know nothing about the system other than that overshoot and oscillations are tolerated.

One popular method that can be used to tune our controller experimentally is called ***Ziegeler-Nichol's first method***. This method gives us rough estimates of what the different parameters should be. The reason for its popularity is that it can be used without having a model of the system, which is just what we need.

This method uses another form of PID called ***standard form*** which gives the parameters a clear physical meaning:

$$H(s) = K_p(e(s) + e(s)\frac{1}{T_i s} + e(s)T_d s) \tag{4}$$

where we have that

- K_p: The proportional constant is the ***controller gain***.
- T_d: The ***derivative time*** is the length of time into the future for which the error is estimated and taken into account. In other words, we predict

the error value at T_d seconds (or samples) in the future, assuming that the loop control remains unchanged.
The relationship to derivative gain: $K_d = K_p T_d$

- T_i: The ***integral time*** is the length of time for which past errors has been taken into account. In other words, the integral component adjusts the error value to compensate for the sum of all past errors, with the intention of completely eliminating them in T_i seconds (or samples).
The relationship to integral gain: $K_i = \frac{K_p}{T_i}$
- The equation only holds when $T_d << T_i$.

Ziegeler-Nichol's first method can briefly be described as follows:

1. Time parameters are set to; $T_i \approx \infty$ and $T_d = 0$. The regulator is then a P-regulator.
2. The gain K_p is gradually increased, while small disturbances are applied to the process (as steps), until the output signal is a harmonic oscillation (or at least a weakly damped oscillation).
3. We note the value of the proportional constant which gives harmonic oscillations (often also called critical oscillations) as ***critical gain***, K_{pk}.
4. The ***period*** of the harmonic oscillation is noted as

$$T_k = \frac{1}{f_k} = \frac{2\pi}{\omega_{180}} \tag{5}$$

where ω_{180} is the frequency of the harmonic oscillation. In other words, T_k is the time of one period of the harmonic oscillation.
5. Based on the noted values K_{pk} and T_k, the time parameters can be calculated according to table 1 (see appendix), and thus the method works like an autotune for the controller.

a) 10% Implement a PI-controller in the standard form.

NOTE *The Ziegeler-Nichol's transfer function is designed for continuous signals, while the signal from the simulation actually is discrete. For discretization, multiply T_i by the sampling time Δt. It's not necessary for this assignment; but if you were to implement the derivative action in a discrete system in the future, the discretization will be* $(current\_error - previous\_error)/\Delta t$.

b) 10% Tune the controller using Ziegeler-Nichol's first method.

Appendix

- Measured masses of links, with motors included.
  - $cm1 = 0.3833\ kg$
  - $cm2 = 0.2724\ kg$
  - $cm3 = 0.1406\ kg$, with pen.
- Inertia tensors

$$\mathbf{I_1} = \begin{bmatrix} 2.5135 & 0 & 0 \\ 0 & 0.9198 & 0 \\ 0 & 0 & 1.8316 \end{bmatrix} \tag{6}$$

$$\mathbf{I_2} = \begin{bmatrix} 3.0675 & 0 & 0 \\ 0 & 0.2234 & 0 \\ 0 & 0 & 2.9577 \end{bmatrix} \tag{7}$$

$$\mathbf{I_3} = \begin{bmatrix} 0.1171 & 0 & 0 \\ 0 & 2.1680 & 0 \\ 0 & 0 & 2.1680 \end{bmatrix} \tag{8}$$

Controller	K_p	T_i	T_d
P	$0.5K_{pk}$	∞	0
PI	$0.45K_{pk}$	$0.85T_k$	0
PID	$0.6K_{pk}$	$0.5T_k$	$0.12T_k$

Table 1: Formulas for Ziegeler-Nichol's first method

REQUIREMENTS:

Obtain a total score of at least 40%.
Each student must hand in their own assignment, and you are required to have read the following declaration on student submissions at the department of informatics: https://www.uio.no/studier/eksamen/obligatoriske-aktiviteter/mn-ifi-obliger-retningslinjer.html

IMPORTANT! Name the pdf file;

"in3140_oblig4_*your_username.pdf*"

Submit your assignment at https://devilry3.ifi.uio.no.
Your submission must include:

1.) A pdf-document with answers to the questions.

2.) The code asked for in Task 3 and 4.

3.) **A README.txt containing a short reflection on the assignment**; what was dificult, what was easy, was there anything you could have done better?

Where you have used MATLAB, Python or other tools to compute an answer, your approach and solution must be illustrated and explained thoroughly in the pdf file. By illustrating, it's meant that you must paste the block diagrams, the images of the functions you were asked to implement, and the images of the output printed to terminal, ***in the PDF file***.
The files containing the code must also be delivered and named;

"in3140_oblig4_taskXX_*your_username.py .m .cpp etc*"

You are free to use whatever programming languages and tools you are familiar with, but for this assignment we recommend using Python, as the provided simulator is built using Python. ROS supports only Python and C++ nodes, so you are limited to these two languages for creating ROS nodes.

Deadline: 24 April 2020, 23:59 (24h)

You can use the slack channel *assignment 4* for general questions about the assignment, and the channels *control_theory*, *matlab_and_python* and *ros* for discussion. Slack team domain is; https://inf34804380robotics.slack.com
Do not hesitate to contact us if you have any further questions.

Artem Chernyshov - *artemch@uio.no*
Daniel Sander Isaksen - *daniesis@uio.no*