INF3480/INF4380 - Assignment 3(b)

Justinas Mišeikis, Eirik Kvalheim and Jørgen Nordmoen

Due: 3. May 2018, 12:00 (24h)

Introduction

In this assignment we will look more closely at control of a robot. We will continue to work with the CrustCrawler and by the end of this assignment you will get to test your controller on the real robot. We will start with PID control theory, before testing path generation.

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. Read the text carefully to fully understand what is required to run and implement. Good luck!

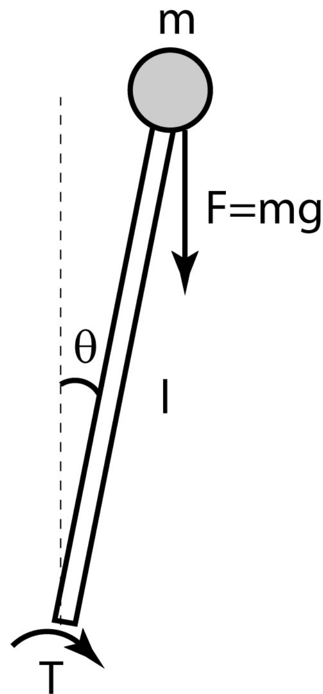


Figure 1: Inverted pendulum model.

Task 2 ($\frac{1}{3}$)

In this task we will try to design a controller for the single joint of a simplified CrustCrawler using a PID controller. We can imagine the simplified CrustCrawler as an inverted pendulum as shown in figure 1.

The dynamic model is

$$2ml^2\ddot{\theta} - mglsin\theta = \tau_m \tag{1}$$

The motor and gears have some internal resistance. We will therefore use a simplified model of the internal resistance given by

$$\tau_m = \tau - c\dot{\theta} \tag{2}$$

The output of the motor is τ_m. This is equal to the torque the motor is set to output (τ) subtracted the internal damping ($c\dot{\theta}$). The torque is a function of the current supplied to the motor. We model this as a linear function $\tau = nu$ where n is the ratio between torque and current and u is the current. This gives a model over the motors as

$$\tau_m = nu - c\dot{\theta} \tag{3}$$

Before we get going lets go through a bit of setup to run the code and simulation.

Setup

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. If you remember, from the ROS lecture, 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 the custom CrustCrawler packages.

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

This will download the CrustCrawler simulation packages which contains a description of the robot and how to simulated 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 this assignment code, into our source directory. You should now be ready to simulate the CrustCrawler!

Assignment

a) 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 PID controller. If you try

to change values now nothing will happen because the controller is not implemented.

We will now test a proportional controller (the 'P' in PID) to control the position of the pendulum. We assume that we can set u to any value. To implement the P-controller we set

$$u = K_p e \tag{4}$$

where $e = \theta_d - \theta$ and θ_d is the desired angle.

b) Implement the controller in Python. In the file 'src/pid.py' you will find the skeleton implementation which is called by the simulation. The return value from the function '*__call__(...)*' should be the same as the current task sub-step. Implement the calculation in equation 4 in the function 'step_b' (do not forget to return u). Then simply close the 'rqt' window (if it is open) and run 'roslaunch pid_assignment pid.launch' again. To change the desired angle of the arm change the 'data' value in the 'Message Publisher' pane. Test different values of P by changing the value in the 'Dynamic Reconfigure' pane. Remember to use the 'self.p' and 'self.error' values so that these are output correctly to the GUI! Test a few values for K_p and report on which values seem to give a good result.

Next we will try the PD-controller. This controller is defined as

$$u = K_p e + K_d \dot{e} \tag{5}$$

where $\dot{e} = \dot{\theta}_d - \dot{\theta}$ and $\dot{\theta}_d$ is the desired angle velocity, which we will assume is zero. Making $\dot{e} = -\dot{\theta}$.

c) Implement the controller in the function 'step_c' and try different values of K_d. Remember to change which function is returned in '*__call__(...)*'. Does this new controller improve over the one in *b)*? Describe the difference and the values of K_d that you found.

It is now time for the full PID-controller. This is defined as

$$u = K_p e + K_i \int e(t)dt + K_d \dot{e} \tag{6}$$

d) Implement this new controller (function 'step_d') and try different values for K_i. Do you observe any improvements in performance? Describe the improvements (or lack thereof) and the values of K_i that resulted in good/bad performance.

To compensate for the non-linearities on the model we will make a controller that removes the non-linearities from our model. To do this we will partition our controller into three parts, one containing the PID control law, α and β. Where α and β account for the non-linearities. This control law can be written as

$$u = \alpha u' + \beta \tag{7}$$

where u implements the PID control law. Combining (1), (3) and (7) we get

$$\frac{1}{n}\left(2ml^2\ddot{\theta} + c\dot{\theta} - mglsin\theta\right) = \alpha u' + \beta \tag{8}$$

To simplify the controller we assume that the internal damping in the motor is zero. This is a linear term and we will be able to compensate for this by using the PID-controller. The new model is thus

$$\frac{1}{n}\left(2ml^2\ddot{\theta} - mglsin\theta\right) = \alpha u' + \beta \tag{9}$$

We want to make the system a unit mass system where $\ddot{\theta} = u'$. To obtain this we use

$$\alpha = \frac{1}{n}2ml^2 \tag{10}$$

$$\beta = -\frac{1}{n}mglsin\theta \tag{11}$$

This gives us a controller that looks like

$$u = \frac{1}{n}2ml^2u' - \frac{1}{n}mglsin\theta \tag{12}$$

Since we want to tune the constants of the controller we replace the constants in equation 12 with generic ones

$$u = K_1u' - K_csin\theta \tag{13}$$

We want to use a PID-controller, so we insert the PID-controller in (6) for u' and set $K_1 = 1$. This gives the non-linear controller

$$u = K_pe + K_i\int e(t)dt + K_d\dot{e} - K_csin\theta \tag{14}$$

where the parameters m and l have been incorporated into the control parameters K_p, K_i, K_d and K_c.

e) Implement this last controller in Python and try different values for K_c. What improvements do you observe this time around? Which values of K_c gives good performance?

Task 3 ($\frac{2}{3}$)

In this final task you will implement a trajectory generator for the CrustCrawler in order to make the robot draw a circle. First this circle will be drawn in the horizontal plane and then later on we will experiment with rotation of this plane. Everything that is needed to solve this task has been explained in lectures (Chapter 5 - *Path and Trajectory Planning* is not needed). We will first test the code in software and then later you will get to test it on the real robot! The task is to draw circles on a board and for this to work we need to calculate a stream of positions along a trajectory that the robot can follow. The focus of this task will be to create a ROS package and seeing how the same code can work in both simulation and in the real-world.

Note that task *a)* accounts for $\frac{1}{2}$ of Task 3 and *b), c), d)* account for $\frac{1}{2}$

Software

We will start this task by creating a ROS package.

a) Create a package that depends on *rospy, actionlib, control_msgs, crustcrawler_pen_gazebo* and *trajectory_msgs*.

b) Implement the path planning node as described here.

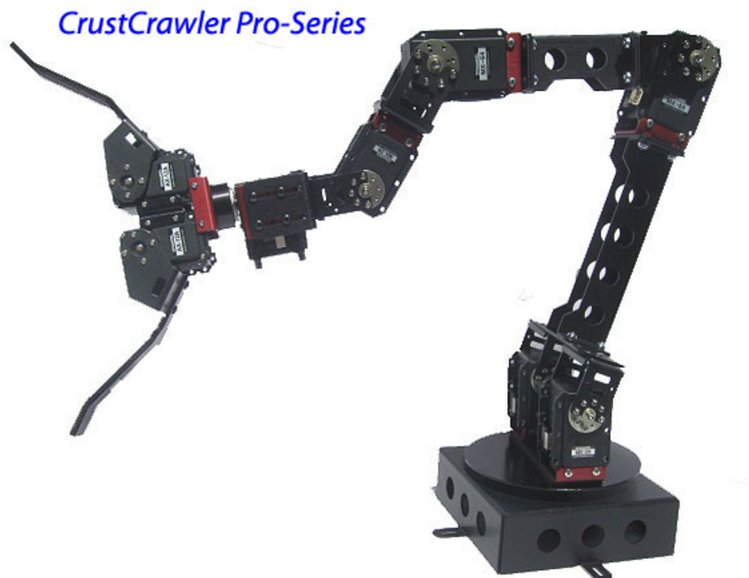
c) Implement rotation of the drawn circle in the function *'rotate_path(...)'*. The arguments *angle* and *axis* describe the rotation.

d) Test different arguments for the circle. What do you observe when you add more points to be drawn?

If you have gotten this far, congratulations! You are ready to test your code on the real CrustCrawler!

Hardware

The final part of this assignment is to demonstrate your code on the real hardware. We have set aside group sessions where you will get a chance to test your code on the real CrustCrawler. You must first demonstrate that your code works in the simulator before you will be able to run the code on the CrustCrawler.



Requirements:
Each student must hand in their own assignment, and you are required to have read the following declaration to student submissions at the department of informatics: http://www.ifi.uio.no/studinf/skjemaer/declaration.pdf

IMPORTANT: Name the pdf file: "inf3480-oblig3-*your_username.pdf*". All deadline and devilry3 questions are to be directed to Nikolai (email below).

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

- A pdf-document with answers to the questions.
- A README.txt containing a short reflection on the assignment; what was difficult, what was easy, was there anything you could have done better?

If you have used MATLAB, Sympy or other tools for computing an answer, your solution and approach must be illustrated and explained thoroughly in the pdf file. The files containing the code must also be named and delivered.

Deadline: 3. May 2018, 12:00 (24h)

You can use the slack channel *assignment 3* for general questions about the assignment. Do not hesitate to contact us if you have any further questions.

Eirik Kvalheim - *eirikval@mail.uio.no*
Daniel Sander Isaksen - *daniesis@mail.uio.no*
Sadegh Hosseinpoor - *sadeghh@mail.uio.no*
Fredrik Ebbesen - *fredreb@mail.uio.no*
Nikolai René Berg *nikolber@mail.uio.no*