INF3480 - spring 2015

Compulsory exercise 3

Introduction

In this exercise we will keep working on the X2 robot. You should know it well by now, so we refer to the previous exercises for details on the robot. The exercises in this assignment will involve PID control, dynamics, and path generation for the robot. You will get the opportunity to try out some of the results from this exercise on a real robot.

1) PID

In Oblig 2 you modeled joint 2 of the X2 robot as an inverted pendulum. In this Oblig you will try to design a controller for the joint. The model you found in the previous Oblig was

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

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

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

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

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

Use the following values in the rest of the exercise. $m = 13.5292$, $n = 0.2460$, $c = 24.3219$, $g = 9.81$ and $l = 0.23893$.

a) Open the Simulink model of the joint. Load the parameters by running the Matlab-script `joint_model_param.m`. Try different initial conditions and watch the response. Is the system stable?

We will now try a proportional controller 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 Simulink and try different values of K_p. Double click on the "Controller" block and implement your controller there. Is it possible to get a stable system using the P-controller? What value of K_p gives good performance? Why is there a stationary deviation?

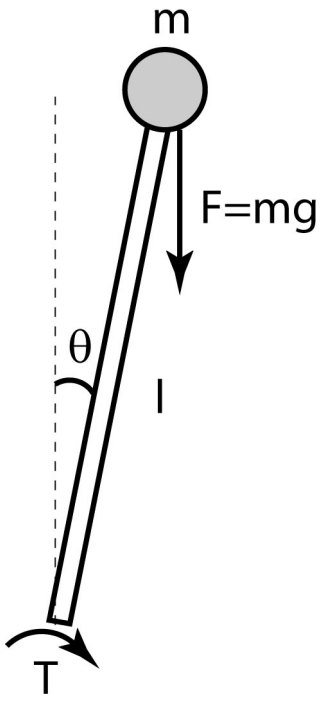

Figure 1: The inverted pendulum

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

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

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

c) Implement the controller in Simulink and try different values of K_d. Are there any improvements on the performance of the system? What are the improvements? What value of K_d gives good performance?

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

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

d) Implement the controller in Simulink and try different values of K_i. Are there any improvements on the performance of the system? What are the improvements? What value of K_i gives good performance?

To compensate for the non-linearities in the model we will make a controller that removes the non-linearities from our model. To do this we will partition our controller into two parts, one

containing the PID control law and one canceling 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

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

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 using the PID controller. The new model is

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

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

$$\alpha = \tfrac{1}{n}ml^2 \tag{10}$$
$$\beta = -\tfrac{1}{n}mgl\sin\theta \tag{11}$$

This gives the controller

$$u = \tfrac{1}{n}ml^2u' - \tfrac{1}{n}mgl\sin\theta \tag{12}$$

We want to tune the constants of the controller. Therefore we replace the constants in the above equation with generic constants

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

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

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

where the parameters m and l have been incorporated into the controller parameters K_p, K_i, K_d and K_c. This control law is the one that is implemented on the real robot.

e) Implement the new controller in Simulink and try different values of K_c. Are there any improvements on the performance of the system? What are the improvements? What value of K_c gives good performance?

2) Dynamics

In this exercise we will further develop the model of the X2 robot. We will be using the *Euler Lagrange Equations* to derive a model. This method uses the kinetic energy and the potential energy to derive the dynamic model. The general formula for kinetic energy of a can be written as

$$K = \tfrac{1}{2}m\boldsymbol{v}^T\boldsymbol{v} + \tfrac{1}{2}\boldsymbol{\omega}^T\boldsymbol{J}\boldsymbol{\omega} \tag{15}$$

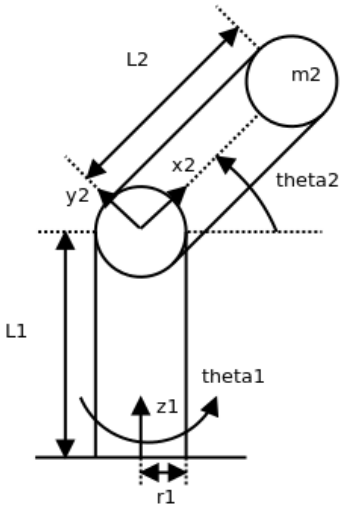

Figure 2: Joint 1 and 2 of the X2 robot

The inner product of the velocities produces a scalar value. The velocity vector $\boldsymbol{v}$ is a three dimensional vector and can be expressed in any reference frame. In this exercise the velocities must be relative to the base frame. The kinetic energy for rotations is given in the last term, where $\boldsymbol{J}$ is the inertia tensor and $\boldsymbol{\omega}$ is the angular speed.

In our model we will model joint 1 as a cylinder with radius r_1, height L_1 and an evenly distributed mass of m_1, see Figure 2. Rotating the cylinder around z_1 gives a moment of inertia of

$$J_{1,zz} = \frac{m_1 r_1^2}{2} \tag{16}$$

The rest of the inertia tensor $\boldsymbol{J}_1$ is zero. The second joint is modeled as a point mass of m_2 located L_2 from the origo of frame 2 along the x_2 axis. To find the kinetic energy of this mass one must find the velocities of this mass in all directions. Since it is a point mass the inertia tensor $\boldsymbol{J}_2$ is zero.

a) Derive the kinetic energy equations K_1 and K_2 for the masses m_1 and m_2 respectively.

The general formula for potential energy can be written as

$$P = mgh \tag{17}$$

where m is the mass, g is the gravitation constant (9.81) and h is height above a defined zero point.

b) Derive the potential energy equations P_1 and P_2 for the masses m_1 and m_2 respectively. Use the base of the robot as zero point.

c) Derive the dynamic model of the X2 robot using the Euler Lagrange Equations.

We will now expand the model to include all three joints, see Figure 3. The third joint is modeled in the same manner as the second joint.

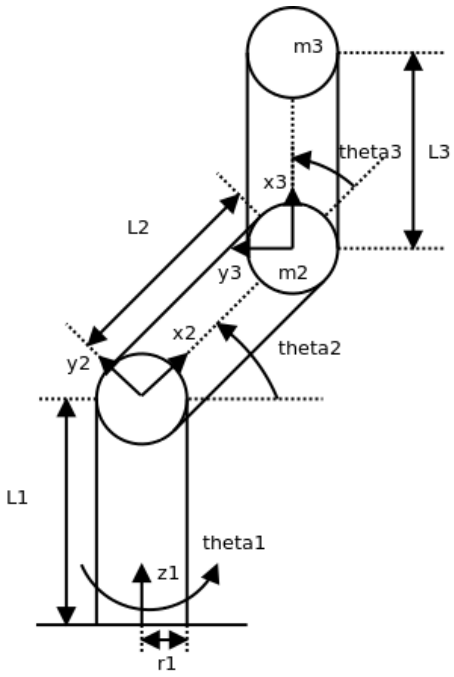

Figure 3: Joint 1, 2 and 3 of the X2 robot

d) Derive the kinetic and potential energy equations K_3 and P_3 for the third joint. You can use the jacobian in Equation 18 to find the velocities for the m_3 mass.

e) Derive the dynamic model of the three joint X2 robot using the Euler Lagrange Equations. Note that K_1, K_2, P_1 and P_2 do not change by introducing the third joint. They must however be used when deriving the dynamic model.

$$J = \begin{bmatrix} -s_1(L_2c_2 + L_3c_{23}) & -c_1(L_2s_2 + L_3s_{23}) & -c_1(L_3s_{23}) \\ c_1(L_2c_2 + L_3c_{23}) & -s_1(L_2s_2 + L_3s_{23}) & -s_1(L_3s_{23}) \\ 0 & (L_2c_2 + L_3c_{23}) & L_3c_{23} \\ 0 & s_1 & s_1 \\ 0 & -c_1 & -c_1 \\ 1 & 0 & 0 \end{bmatrix} \tag{18}$$

3) Robot model

We have developed a simulation of the robot. This is included with the download of this paper. You will use the model when you are working with this exercise. The robot should draw certain shapes on a board, and for this to work it needs to receive a stream of positions along a trajectory. This is what you will program in matlab, test out on the simulation and later transfer to the real robots. The animation works with any angle, but the real robot will only accept angles between -180 and 180 degrees, so you should limit your angles to this range.

Simulation

The simulation is developed in Max/MSP/Jitter, and is set up to receive UDP messages from matlab in the same way as the actual robot does. It is important that you get things working in the simulation before we try it on the X2 robot. This is to prevent the X2 robot from getting damaged.

Matlab and the necessary software to run the robot animation is installed in the room "2428 Lisp", and the setup is tested and works fine there.

If you want to run the animation on your own computer, you need QuickTime and Java Runtime Environment installed, and if you are on a Windows PC, you will maybe need Microsoft .NET framework 3.x which you get from Windows Update.

A Mac version is available for download from the course website.

MATLAB

Three matlab scripts are provided along with this exercise. You can use these to communicate with the robot (or write your own if you prefer).

The script `activateAnimation.m` sets up three UDP connections to localhost on ports 7701, 7702, 7703. These are the ports where the simulation is listening for commands.

The script `deActivateAnimation.m` closes the three UDP connections.

The function `setRobotAngles.m` takes three angles (in degrees) as input, and sends these to the animation via the UDP connections.

Basic usage

a) Start the animation (the file `x2model.exe`)

b) Click the button next to "load all robot-parts" to load all the robot files.

c) Type `activateAnimation` in the matlab prompt to setup the udp-connection.

d) Type for instance `setRobotAngles([90 90 -90])` to set the robot to another configuration.

e) Try other configurations.

f) Close the UDP-connection by typing `deActivateAnimation` when you are done.

You should start this exercise by exploring the animation.

In exercise 3, X, Y and Z coordinates are given. You are free to rotate the base coordinate system around the Z_0 axis, and so you may have to swap and/or revert the X and Y axes to fit your matlab code. In either case, the point is that you should make the robot draw on the drawing board.

a)

Write a path generator in matlab that calculates stepwise each new discrete cartesian point on a circular trajectory that the robot is supposed to move along. Combine this path generator with your inverse kinematics function.
These parameters are input to the path generator:

- Circle radius
- Origo position

These variables must be easily controllable in the code:

- The time delay between each position that is sent to the robot (the robot needs to get a position, then move, then get a new position, etc.). You can use the matlab function `pause(seconds)` to create a pause between each robot step.
- The resolution of the path generator (i.e. the length of each step).
- The position of the centre of the drawing board.
- The orientation of the drawing board.
- The static DH-parameters (in practice meaning only the length of the links).

You might have to add offsets of +/- 90 or 180 degrees to the theta values in your matlab code to make your function compatible with the model and the robot

b)

Use the drawing function in the animation, and draw a circle that is parallel to the Y_0Z_0-plane with the center at $X = 45cm$, $Y = 0cm$, $Z = 20cm$ (in the base coordinate frame of the robot), and a radius of 5 cm. In the simulation the white dot marks the center point on the drawing board.

Draw the same circle rotated around the Y axis by 30, 60 and 90 degrees.
Take screenshots of the drawings (use printscreen).

c)

When we are working on the robot, we will project the tilted circles onto a vertical drawing board as shown in Figure 4 on the next page. Make your matlab function able to do this sort of projection (e.g. by locking the X-coordinate.)

d)

A regular industrial robot uses both the inverse kinematics and forward kinematics. Why do we not use the forward kinematics in this exercise?

4) Real Robot

The X2 robot can be interfaced in Matlab by using the supplied Matlab functions. The Robot should be set in the initial position shown in Figure 5 before it is used. Now all the joint angles are set to zero. You must find a way to convert the angels to your convention. Joint 1 has a positive rotation in the clockwise direction. The two other joint has the direction defined on the robot. The following functions are provided:

- openRobot
- closeRobot

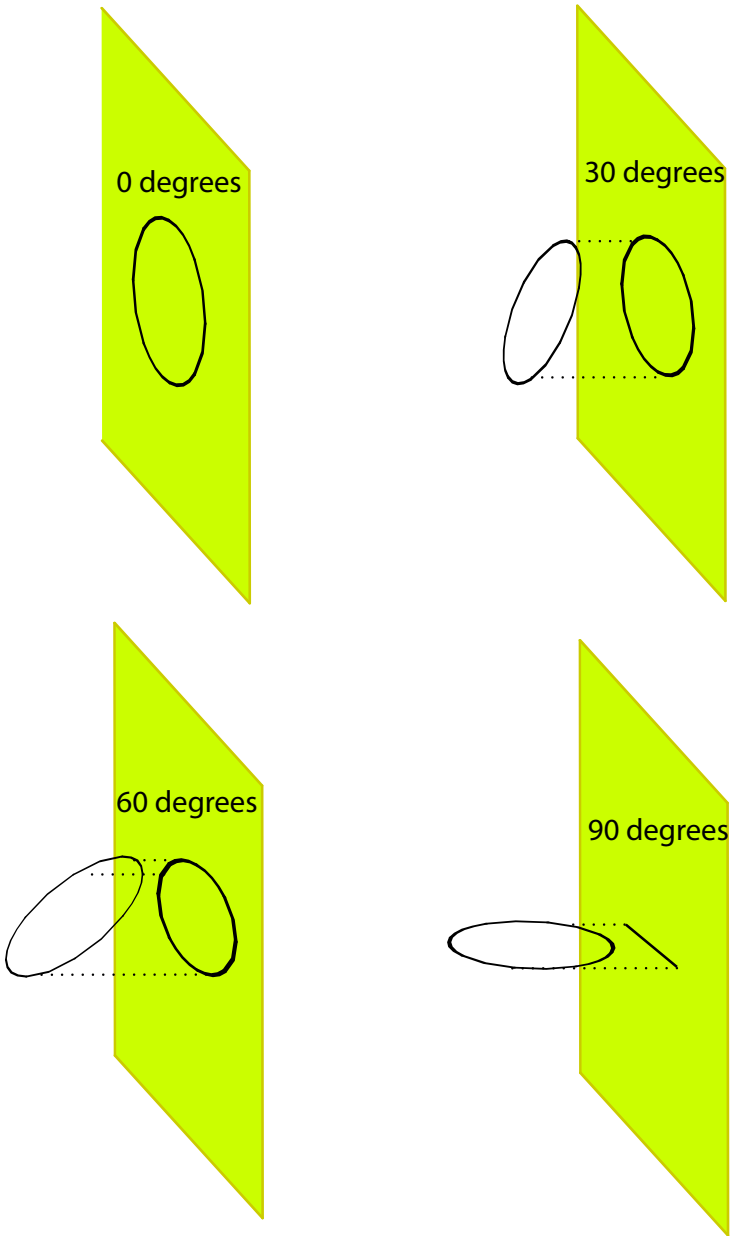

Figure 4: Example of rotating circles in space and projecting them on the drawing board

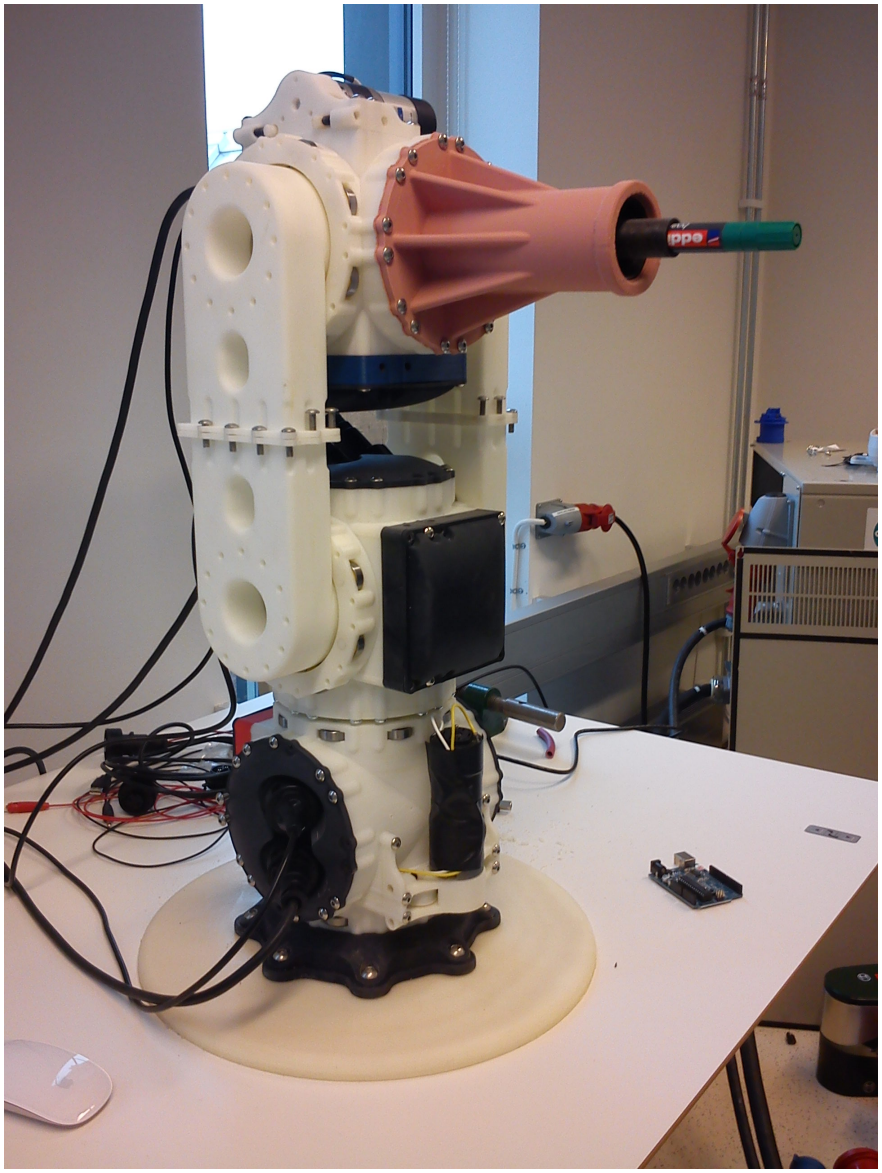

Figure 5: The initial position of the robot

- toggleMotor
- setRobotAngles
- resetJoints
- getStatus

To open the interface you should use `robot = openRobot(com_j1, com_j2, com_j3)`. This function returns a struct that must be used with alle the other functions. The input of the functions are the COM-ports to communicate with each joint. Be patient when calling this function, it will take more than five seconds to complete the call. The following line should open the joints correctly

```
robot = openRobot('COM6','COM4','COM5');
```

The interface must be closed using `closeRobot(robot)`. If not you will not be able to open the interface again. If your code fails before closing you should first try to restart Matlab. If this do not work you should restart the computer.

To turn the motors of the robot on and off use the function `toggleMotor(robot)`. It will be displayed if they are turned on or off. To move the robot use the function `setRobotAngles(robot, theta1, theta2, theta3)`. The input is three setpoints for the joint positions in degrees. Note that joint 2 only can move between 0 degrees and 90 degrees. This is to avoid the cables to collide. You should use the `pause` Matlab command after setting the setpoint in order for the robot to move to the correct position.

If you started the robot in an incorrect initial position you may reinitialize the robot. Turn off the motors and move the robot to the initial position. Then use the function `resetJoints(robot)` to reset the robot.

The function `getStatus(robot)` will return status information about the tree joints. Control parameters, measured and target position will be shown.

a) Use the drawing functions develop in assignment 3 to draw circles on the real board.

Requirements:
Each student must hand in their own assignment, and you are required to have read the following requirements to student submissions at the department of informatics: http://www.mn.uio.no/ifi/english/studies/admin/mandatory-assignments/assignments-guidelines.html

Submit your assignment on https://devilry.ifi.uio.no/
Your submission must include:

- A pdf with answers to the questions
- The simulink files
- The matlab-functions for path generation etc. (.m-files).
  Use comments in the code to document the functions!
- Screenshots of the circles drawn by the model

Deadline: Sunday, May 15th

Don't hesitate to ask us if you have questions.
Contact information is at the course website.