cse 4360 / 5364

Homework 1- Fall 2016

Due Date: Oct. 17 2016

Forward and Inverse Kinematics

For this Exercise you have to derive the forward kinematics and a partial inverse kinematic function for a manipulator and implement them in a robot simulator. To do this you are provided with a C library containing the simulator. All your code should be written in the file *kin_fncs.c* which contains 2 main functions, *fwd_kin(theta, x)* and *inv_kin(x, theta)*. These functions are directly called by the simulator.
To start with this part of the assignment you have to retrieve the compressed tar file for your computer architecture (either for Linux, Cygwin under Windows, or OSX with X-Windows) from the web site and uncompress it to your account. This archive contains a directory naked *kin* which contains the following files:

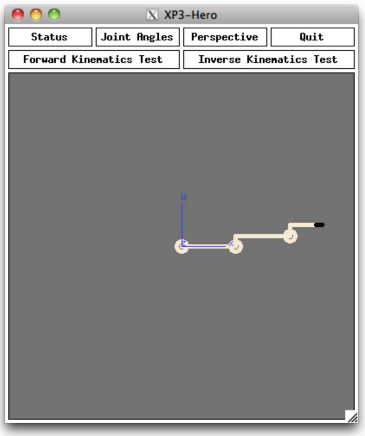
Imakefile This file is used to create a machine specific Makefile by typing *xmkmf*.

kin_fncs.c This is the file you have to edit in order to implement the kinematic functions.

lib/libKin.a This library contains the simulator and some routines to test your kinematic functions.

The Kinematic Simulator

To generate the simulator with your kinematic functions you just have to type *make*. This creates the simulator executable *Kinematics*. The simulator should look as follows:

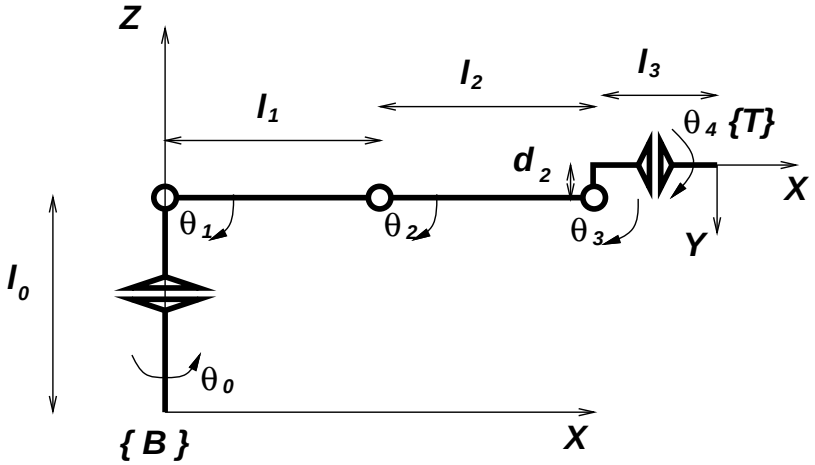


The *Joint Angles* and *Perspective* buttons at the top of the simulator window will open a set of sliders which allow you to move the individual joints of the robot and to change your viewing angle, respectively (the default perspective is a top view).
To test your forward and inverse kinematic functions you can press the *Status* button which will print the current joint angles, the actual location of the tool frame, and the results of your forward and inverse kinematic functions for the current robot configuration. In addition, the 2 buttons in the second row will test your functions in 100 random robot configurations and report if they differ from the expected result.

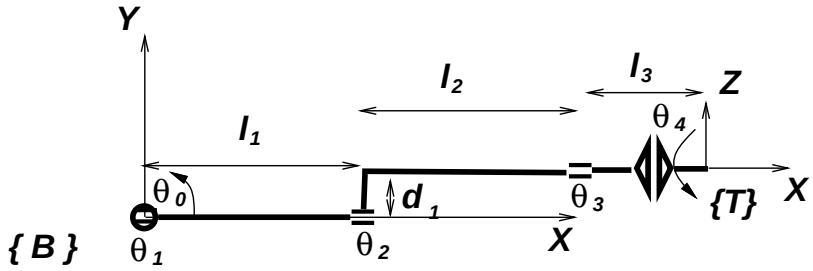
The Robot

In the first examples you are to determine the forward kinematics and a partial inverse kinematics for the following 5 degree-of-freedom robot manipulator:

Side View:



Top View:



The link length are defined as follows:

$$l_0 = 0.25m \quad l_1 = 0.2m$$
$$l_2 = 0.2m \quad l_3 = 0.15m$$

In addition there is an offset of $d_1 = 0.04m$ between joint 1 and joint 2, and an offset of $d_2 = 0.04m$ between joint 3 and joint 4 of this robot. The picture above shows the robot in the configuration where all joint angles are 0.

The simulator will actually show one additional joint (joint 5) corresponding to the opening of the gripper. This, however, is not relevant for the kinematic functions.

1. Determine and implement the forward kinematic function for this Robot.

   Here you are supposed to determine the transformation from configuration space (joint angles) to the Cartesian location of the tool frame ($\{T\}$) in base frame coordinates ($\{B\}$). You only have to provide the position. The orientation of the tool frame is not required. (HINT: What effect does the last joint, Θ_4, have on the location of the tool frame ?)

   For this part of the assignment you are to hand in a written version of the forward kinematics and your derivation (either closed form or as a sequence of transformation matrices), and the code for the forward kinematic function you implemented.

   For the implementation you have to write the function *fwd_kin()* inside the file *kin_fncs.c*. This function has the following structure:

   ```
   fwd_kin(theta, x)
   double theta[6];
   double x[3];
   {
   ...
   }
   ```

   It gets passed in an array containing the 6 joint angles *theta* (the last angle corresponds to the gripper and is of no interest here) and should calculate *x* , the 3 dimensional position of the tool frame (x[0] = x, x[1] = y, x[2] = z).

2. Determine a partial inverse kinematic function for the robot and implement it in the simulator.

   In this part of the assignment you are to derive and implement an inverse kinematic function for the robot manipulator for a specific orientation of the tool frame (if multiple solutions exist only one is required here). This orientation is given by $\Theta_4 = 0$ and the requirement that the X-axis of the tool frame is parallel to the Z-axis of the base frame but in the opposite direction. In other words, the last link of the manipulator is to point straight down. (HINT: To determine this inverse kinematics you should decompose the problem. You can determine Θ_0 and Θ_3 separately and calculate the other joint angles by analyzing the substructure formed by links 1 and 2 (l_1 and l_2). This requires moving the wrist frame to joint 3.)

   Again you are to hand in a written version of this inverse kinematics and its derivation and the code. The code here should be implemented in the function *inv_kin()*.

   ```
   inv_kin(x, theta)
   double x[3];
   double theta[6];
   {
   ...
   }
   ```

This function gets passed in the location of the tool frame (x) and has to compute a corresponding joint angle configuration (*theta*).

Dynamics and Control

In this exercise you are going to perform system identification and implement a PD controller for a 1 degree-of-freedom robot manipulator.
Again you are provided with a C library containing a dynamic simulator and a file in which you should implement the controller. For this part of the assignment, the subdirectory *dyn* contains the following files:

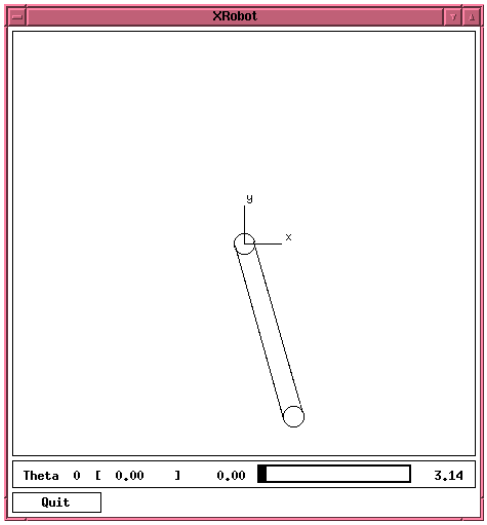
Imakefile This file is used to create a machine specific Makefile by typing *xmkmf*.

PD_control.c This is the file you have to edit in order to implement the controller and to perform the experiments required for system identification.

lib/libDyn.a This library contains the dynamic simulator.

The Dynamic Simulator

To generate the simulator you have to type *make*. This creates the simulator executable *Dynamics*. The simulator looks as follows:



The Slider on the bottom allows you to set the reference angle configuration for your controller (the reference velocity is always set to 0).

3. Determine the dynamic parameters of the robot.

   Here you are supposed to perform system identification to determine the parameters required later to perform model compensation in your PD control loop. The dynamic model of this single joint manipulator has the form

$\tau = I\ddot{\Theta} + B\dot{\Theta} + G(\Theta)$

These are caused by the inertia of the arm (I), the viscous friction in the joint (B) and by gravity (G).

To determine these elements of the dynamics you have to perform experiments with the simulated robot. Your interface here is given in the function *PD_control()* which receives all relevant information from the robot system and is called at a rate of $500Hz$.

```
double PD_control(theta, theta_dot, theta_ref, theta_dot_ref)
double theta, theta_dot, theta_ref, theta_dot_ref;
{
...
return(...);
}
```

This function receives the current joint angle (*theta*), the corresponding rotational velocity (*theta_dot*), the desired reference angle (*theta_ref*) set using the slider, and the reference velocity (*theta_dot_ref*). The output of this function should be the amount of torque you want to apply to the joint.

(HINT: To perform system identification you want to collect this data and analyze it in order to determine the system parameters).

For this part of the assignment you should hand in the system parameters found as well as a description of the method used to determine them.

4. Implement a PD controller with model compensation for the 1 degree-of-freedom robot.

   Using the parameters found above you should implement a PD controller with model compensation for this robot. Use of the model will give the system the appearance of a unit-mass system without friction or gravity and critically damped behavior should therefore be achieved by a choice of gains where $K_v = 2\sqrt{K_p}$.

   For this part of the assignment you have to hand in the controller code you implemented in the function *PD_control* in the file *PD_control.c*.