In the following problems, you will need the NAO simulator and Matlab for CLMCPLOT for other visualizations. You find relevant files and information under:

http://www-clmc.usc.edu/Teaching/TeachingIntroductionToRoboticsHomework

IMPORTANT: In your solutions of the homework, also provide intermediate steps how you derived the solution to a problem.

1. (80 Points) In this homework, you are supposed to create a complete point-to-point reaching behavior for the NAO robot. First, you will need to create a planning system for point-to-point movements for the end-effector (left hand) of the robot. Second, you will need to implement the plan on the robot simulator and visualize the outcome to confirm that it works.

a) For planning, you are to create a 3rd order spline system (a “cubic spline”). The general 3rd order spline equation is:

\[ x(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3 \]

Assume the movement duration is \( \tau \), and the movement starts at \( t=0 \). Assume that at start the movement is at \( x_0 \), and it is supposed to reach \( x_f \) at time \( t=\tau \). Determine all the constants \( c_0 \) to \( c_3 \) as a function of \( x_0 \), \( x_f \), and their derivatives, and \( \tau \). At start and end velocities are \( \dot{x}(0) = \dot{x}_0 \), \( \dot{x}(\tau) = \dot{x}_f \).

b) Using matlab, implement a planning system that creates a trajectory of 2 seconds duration, starting at position \( x=0 \) and going to a target at \( x=2 \), with zero velocity boundary conditions. Use a time step of 0.01s to compute the movement plan. Provide a print-out of your matlab code, and plots of position, velocity, and acceleration for the entire trajectory.

c) A useful way to implement the cubic spline planning system is by creating a function that takes as input variables the current state \( x(t), \dot{x}(t) \), the remaining time to go \( \tau_{\text{togo}} \), and the target state \( x_f, \dot{x}_f \). The output of the function would be \( x(t+\Delta t), \dot{x}(t+\Delta t) \), i.e., the planned state one time increment ahead. This function can be used to plan the next desired state given the current desired state, which is useful in a control loop of a robot. Note that the time-to-go \( \tau_{\text{togo}} \) needs to be decreased at every iteration of the control loop by \( \Delta t \). Implement this function in matlab, and create the results of b) using this incremental planning system. Compare the results of b) and c) and comment on the differences and similarities.

d) The webpage has a file cubic spline task.cpp. Modify this file to implement your cubic spline function from matlab. Run the task using the “setTask” command in the blue task_servo window. You will automatically collect a data file, and you should save it to disk with “saveData”. Visualize the trajectories of all 5 left arm DOFs in CLMCPLDT in matlab, i.e., print the data traces L_SFE_th and L_SFE_des_th in the top most chart of clmcplot, and then the data traces for L_SAA, L_HR, L_EB, and L_WR in the next 4 charts. Create a print-out of your visualization and include it in your homework. Comment on the quality of track-
ing for the 5 DOFs. Repeat the same for the plot, this time using joint velocities, and then another plot using joint accelerations.

e) Add three additional targets to your C-program such that the left arm creates approximately a square in Cartesian space. Note that due to the joint-space movements, the hand movement in Cartesian space is curved. Your square should end at the same point where it started. Collect data of your movement and save it such that it can be visualized with CLMCPLOT. In clmcplot, click the “PhasePlot” button, and then click LEFT\_HAND\_x and then LEFT\_HAND\_y. This creates a window that plot LEFT\_HAND\_x in the horizontal axis, and LEFT\_HAND\_y in the vertical axis. Provide this plot in your homework and comment on what is good or bad about your realization of the square. Repeat this plot for LEFT\_HAND\_x and LEFT\_HAND\_z and provide this print-out as well. Comment on the quality of your square from this view.

2) (45 Points) Consider the dynamical system:

\[
\begin{align*}
\dot{x}_1 &= b x_1 + k x_2 + x_3 \\
\dot{x}_2 &= x_1 \\
\dot{x}_3 &= \alpha (u - x_2) - \beta x_3
\end{align*}
\]

(2)

a) (5 points) Show a physical system that corresponds to the equations in (2) and explain what the parameters mean.

b) (5 points) Transform the system into the frequency domain and give the individual transfer functions and the transfer function of the complete system. Provide a block diagram that shows how the individual transfer functions are connected.

c) (15 points) Assume \(b=-1, k=-1, \alpha=1, \beta=10\). Use Simulink to build a PD controller for the system, assuming that the system should track a sinusoidal desired trajectory with 1Hz frequency. Use the “Transfer Function” building block in Simulink for this implementation. Attach a “Scope” to the simulation that simultaneously shows the true output of the system, and the desired output (note that “Scopes” accept vector inputs to display multiple trajectories superimposed). Also create a scope that shows the tracking error. Tune the gains of the PD controller to get as good as possible tracking performance. Give plots of i) your Simulink system, and ii) position and desired position as well as the tracking error of the best tracking performance (over 10 seconds simulation time). (Note, all Simulink windows allow copy/paste or direct printing of their contents). Make sure that all your plots are scaled such that they have maximal resolution in the vertical axis!

d) (5 points) In real physical systems, the state of the controlled systems needs to be measured, usually by means of sensors that add some noise to the measurement. Additionally, the sensory feedback can only be measured at a certain frequency. In order to simulate such a sensory processing, add a “zero order hold” block (from the “Discrete” library) in the feedback path way of your system (i.e., the pathway leading back from the output of the system’s transfer function to the negative feedback controller). Set the “sample time” parameter of this block to 0.005 (i.e., 200Hz). To simulate the noise in the sensory, take the “Random Number” block from the “Sources” library and sum it into the feedback pathway after the “zero order hold”. Set the “sample time” parameter of this block also to 0.005, and the variance pa-
rameter to 0.01. Now tune your PD gains to get the best performance for this more realistic control system and provide plots of the best tracking performance and tracking error as in point b-ii above. Briefly describe the difference in performance between the system in b) and the current one, and why this difference exists.

e) (15 points) In order to deal with sensory noise, signal filtering is usually employed. A very simple filter is of the form:

\[ \dot{x} = \lambda (u - x) \]

where \( \lambda \) is the time constant of this filter and regulates the amount of filtering: the smaller \( \lambda \), the more filtering is achieved. The danger of too much filtering, however, is that the filtered signal becomes delayed. Convert the filtering equation into a transfer function, and add it in the feedback pathway after the point where the noise was added. Adjust \( \lambda \) to improve the performance of your control system, and provide the same plots as in c). Briefly comment on the success of your filtering. Note that you may have to re-tune the PD controller for the filtering.