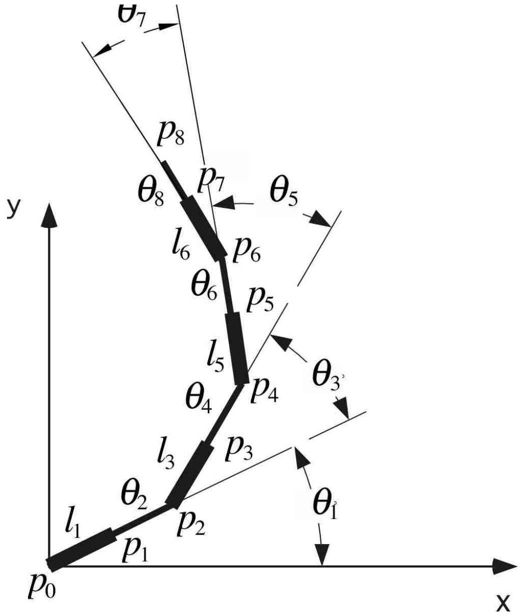
CS545–Introduction to Robotics

Homework Assignment 2 (Due April 2)

1. On the homework web page, you find four files: kinematic_arm_simulation.mdl, ArmAnimation.m, forward_kinematics.m, and inverse_kinematics.m. This Simulink simulation is a kinematic simulation of a planar 8-joint-arm (note that a "kinematic simulation" is a simulation where no mass and inertia properties are included – it basically assumes that there is a perfect inverse dynamics model in the control loop such that desired states and current states always coincide). Details of the robot arm are in the figure below: it has one revolute joint followed by a prismatic joint, and so on. For simplicity, the joints have no stops, i.e., revolute joints can rotate more than 360 degrees, and prismatic joints can be positive or negative length.



a) Write down the generic formula for the geometric Jacobian of a general robot system with *n/2* pairs of revolute/prismatic joints (that is n degrees-of-freedom) arranged in an open-chain fashion as shown for the 8 degrees-of-freedom system above – just assume that all the n links form an open-loop chain, where each revolute link is of length l_i with joint angle θ_i, each prismatic joint is solely described by the length coded in θ_i and zero link length (i.e., $l_2 = l_4 = l_6 = l_8 = 0$ in the figure above) and the origin of the i-th local coordinate system is at $\mathbf{p}_{i-1}$. The joint axes are denoted by $\mathbf{z}_i$. Note that you also need to give the part of the Jacobian that deals with orientations. Explain the symbols in the formula. Use the notation in the figure above for this formula. Note that this result will look like a partitioned matrix, similar as in the introduction of the geometric Jacobian in the textbook.

b) Expand and simplify the formula for the Jacobian in a) such that it is only a function of the joint variables and link lengths, i.e., eliminate the $\mathbf{p}_{i-1}$ variables. Take into account that this Jacobian is for a planar arm, and has only one orientation component.

c) Implement this Jacobian for the 8 DOF system above in the Matlab program inverse_kinematics.m at the indicated position by using the (vector) variables "links" and "theta" in this function. Provide a printout of the program.

d) Give the general formula of the Jacobian transpose for inverse kinematics computations, and implement this method in the "inverse_kinematics.m" program. Provide a printout of your program. Run the Simulink simulation (which implements an ellipsoid tracking task with orientation control of the endeffector at an orientation angle of zero – this is like balancing a glass of water on the endeffector) and provide printouts of the x-y-Graph, the gamma graph, and the joint trajectories in the "scope" block. How well does the method perform? Note that you may have to tune some "multipliers" for best performance. Judge from the data whether the approach is conservative and explain your opinion.

e) Give the general formula of the pseudo-inverse for inverse kinematics computations, and implement this method in the "inverse_kinematics.m" program. Provide a printout of your program. Run the Simulink simulation and provide printouts of the x-y-Graph, the gamma graph, and the joint trajectories in the "scope" block. How well does the method perform? Judge from the data whether the approach is conservative and explain your opinion.

f) Give the general formula of the pseudo-inverse with Null-space optimization for inverse kinematics computations, and implement this method in the "inverse_kinematics.m" program — use zero joint variables as a desired optimization posture (see Slide 9, Lecture-9). Note that the Null-space optimization term requires a scalar multiplier to be maximally active. Provide a printout of your program. Run the Simulink and provide printouts of the x-y-Graph, the gamma graph, and the joint trajectories in the "scope" block. How well does the method perform? Judge from the data whether the approach is conservative and explain your opinion.

g) Using a weighted pseudo-inverse allows you changing how much each degree-of-freedom should contribute to the inverse kinematics. Assume a weight vector for the eight joints of the robot as **w**=[1000 0.1 100 0.1 10 0.1 1 0.1]. Derive a weighted version of the inverse kinematics of e) and repeat subproblem e) for this weighted version. How well does the method perform?

h) You can also use the null-space optimization criterion with the weighted pseudo-inverse. Repeat the subproblem f) with the weighted pseudo-inverse, but the NON-WEIGHTED pseudo-inverse for the Null-space optimization. How well does the method perform? Is this approach mathematically correct? Give a mathematical argument for or against it.

i) How would you choose the weights and the inverse kinematics method if you don't want the prismatic joints to move at all? Give a print-out of the weights and the inverse kinematics method you chose. Implement your solution and provide the same plots as in f). Who well does this work?