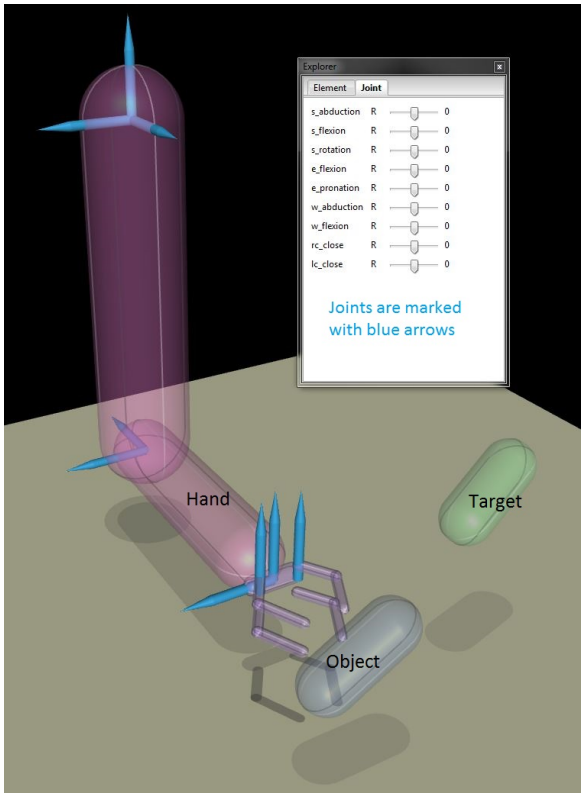
Homework 1:
Jacobian Control Methods

E. Todorov, CSE P590

Due midnight May 12, 2014

Problem statement

In this assignment you will implement two Jacobian control methods discussed in class to control position and orientation of a simulated 9 degree of freedom arm. Having these tools, you will use them to create a controller that picks up an object.



Part 1: Position control with Jacobian transpose and pseudo-inverse methods

In this part of the assignment, you will make the palm of the simulated arm reach a target marker. Let all the arm joint angles be collected in a vector θ as in the lecture notes. Let $\mathbf{x}$ be the 3d position of the end effector we wish to control (palm of the hand, in our case). As discussed in the lecture, a position Jacobian $J(\theta)$ is a matrix that maps a change in joint angles $\Delta\theta$ to a change in end effector position $\Delta\mathbf{x}$. The relationship is $\Delta\mathbf{x} = J(\theta)\Delta\theta$.

Instead, you will go in the other direction. Let the fixed target marker position be $\hat{\mathbf{x}}$. Then at any point in time, we want a palm position change that moves it to the marker: $\Delta\mathbf{x} = \hat{\mathbf{x}} - \mathbf{x}$. Given this change in position, you will calculate the corresponding change in joint angles $\Delta\hat{\theta}$. Two methods for calculating $\Delta\hat{\theta}$ are listed on pages 7 and 13 of the Jacobian methods lecture notes and it is up to you to understand and implement them. Once you calculated it, the desired joint angles are $\hat{\theta} = \theta + \Delta\hat{\theta}$.

There are several constants in the formulae in the lecture notes. Set reference pose θ_o to a zero vector and start with α set to 10^{-2}. Experiment with several values of α to see how it affects the resulting motion.

In the provided starter code, you will provide you with methods to access θ $J(\theta)$ $\hat{\mathbf{x}}$ and a method to set $\hat{\theta}$. Your task is to move the hand so that its position coincides with position of the green capsule (which you cannot collide with). Note that the full scene in the simulator contains both an arm and an additional unactuated object. θ and $\hat{\theta}$ refer only to degrees of freedom of the arm (which you can actuate) and not the unactuated object.

Part 2: Orientation control

Now instead of reaching for a particular position, you will make the palm match the orientation of the target marker. This task is very similar to position control above, with a few caveats. Let $\mathbf{r}$ be the palm orientation, which is now a quaternion. Similarly, there is an orientation Jacobian matrix $J_r(\theta)$ mapping $\Delta\theta$ to a change in orientation: $\Delta\mathbf{r} = J_r(\theta)\Delta\theta$. Somewhat counter-intuitively, $\Delta\mathbf{r}$ is a 3d vector while $\mathbf{r}$ is a quaternion. If $\hat{\mathbf{r}}$ is the desired palm orientation, then we can follow the same procedure as in part 1:

1) calculate desired change in orientation: $\Delta\mathbf{r} = \text{quatdiff}(\mathbf{r}, \hat{\mathbf{r}})$

2) calculate $\Delta\hat{\theta}$ using $J_r(\theta)$ and two formulae from lecture notes

3) set $\hat{\theta} = \theta + \Delta\hat{\theta}$

The only tricky part is calculating an orientation difference 3d vector from two quaternions. We will provide you with *quatdiff* function that does this. For details on how it works and a gentle introduction to quaternions, you can see this article. For this part you will have additional methods to access $\hat{\mathbf{r}}$ and $J_r(\theta)$.

Part 3: Combined position and orientation control

In this part, you will combine both position and orientation control, making the palm of the arm both reach the target marker and match its orientation. Very conveniently, this can be done by simply concatenating the difference vectors and Jacobians of the above two tasks. The procedure and formulae are identical to the previous two parts, except the combined desired change vector is now

$$\begin{bmatrix} \Delta\mathbf{x} \\ \Delta\mathbf{r} \end{bmatrix} \in R^6$$

and the combined Jacobian matrix is

$$\begin{bmatrix} J(\theta) \\ J_r(\theta) \end{bmatrix} \in R^{6\times|\theta|}$$

Part 4: Picking up an object

Now you can combine the two ways of controlling the arm for a more open-ended task of picking up an object. Your task is to grip the blue capsule object and move it to the green target position and orientation. One possible way to solve this is to break the task up into a sequence of subtasks. For example, move arm towards the object (using position and orientation control you implemented in previous parts), close the hand gripper to grasp the object (you will have *setGrip* function to set gripper opening and closure), and with object in hand, position and orient it towards the target. How you orchestrate these subtasks and how you set hand position and orientation targets is up to you. There is no gravity in the simulation that would cause the object to fall down, so you have ample time to grip it.

Starter code

On the Catalyst assignment page you will find a package containing Mujoco physics simulation server and a starter Visual Studio project containing a Mujoco client that you will extend.

After compiling MujocoClient, start *Mujoco.exe* (the server). You will see simulation display window. Then start *MujocoClient.exe*. The client will load a hand model we provided and run simulation forward while setting constantly setting $\hat{\theta}$ and reading quantities $\mathbf{x}$, $\mathbf{r}$, etc. $\hat{\theta}$ is set to zero in the starter code, which maintains the arm in a reference pose. You can press *F2* for access to various visualization features to help you. Read starter code and comments for more details. Your task is to extend the starter code to complete the four assignment parts.

What to submit

Submit four executables solving each of the four parts of the assignment along with your source code. The source code may be either four separate source files, or one source file with different solutions commented out. We are flexible as long as we can see all your

implementation. The executables may use whichever method for calculating $\Delta\hat{\theta}$ you found to work better. Additionally, submit a short writeup (pdf or txt) explaining the effects of various settings of α on the resulting solution trajectories and the differences you observed between two methods of calculating $\Delta\hat{\theta}$.