Advanced Robotic Manipulation Handout

CS327A (Spring 2017)

Problem Set #2 Due **Wed, Apr** 26^{th}

Guidelines:

- This homework has both problem-solving and programming components. So please start early.
- In problems that require mathematical derivations, please try to be as detailed as possible so we can reward partially correct answers.
- For problems involving numerical answers, round off answers to two significant digits after the decimal point.
- We will be using the SAI2 (Simulation and Active Interfaces) software framework, developed in the Stanford Robotics Lab for this and future homeworks. Note that the SAI2-Simulation library is currently closed source. *So please do not forward or distribute.*
- Collaboration is allowed on this homework, but each student must submit their own original content (solution and code). In case you did collaborate, please note the names of other students you have worked with.

Submission Details:

We are using Gradescope for this class. So please submit your homework write-up through the Gradescope website directly. `https://gradescope.com/courses/7467`

For code submission, we'll be providing you a submission script with instructions soon.

Software update

The source code for this homework requires the latest version of SAI2-Common and the CS327A class repository.
To update the SAI2-Common repository, from within the `sai2-common` directory that you installed as part of homework 1, run

```
$ git pull
$ cd build && cmake -DCMAKE_BUILD_TYPE=Release .. && make && cd ../
```

To update the class repository, from within the `cs327a` directory, run

```
$ git stash
$ git pull --rebase
$ git stash pop
```

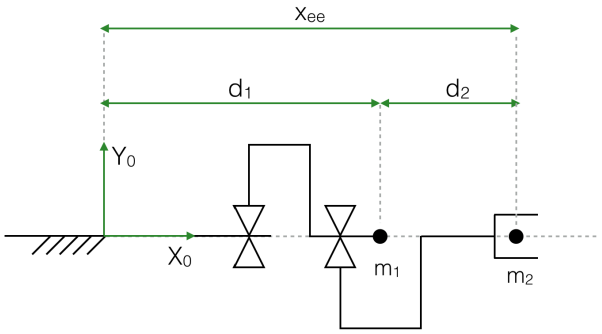
If you are asked to configure your computer for `git` with a username and email address, follow the instructions to do so, then run the above commands again.

Problem 1 - Redundant robot control

The goal of this problem is to familiarize you with the control of a robot that is redundant with respect to the task it is required to perform.

(a) Weighted instantaneous inverse kinematic solutions

Consider the PP robot shown below.



The masses for link 1 and link 2 are $m_1 = 10\,\text{kg}$ and $m_2 = 5\,\text{kg}$ respectively. Consider the task of moving along the X_0 axis. Answer the following questions:

i. Is the robot redundant with respect to the task? How many degrees of freedom does the task require? How many degrees of freedom does the robot have?

ii. Find A, the generalized co-ordinates mass matrix of the robot.

iii. Let x_{ee} be the displacement of the end-effector along the X_0 axis. Find the task Jacobian J such that

$$\dot{x}_{ee} = J\dot{q}, \quad q = \begin{bmatrix} d_1 \\ d_2 \end{bmatrix}$$

iv. Find J^+ and $[I - J^+J]$. Then express the family of "instantaneous inverse kinematic" solutions in terms of δx_{ee} and δq_0:

$$\delta q = J^+ \delta x_{ee} + [I - J^+J]\delta q_0$$

v. Find $J^\#$ and $[I - J^\#J]$ where

$$J^\# = W^{-1}J^T(JW^{-1}J^T)^{-1}, \quad W = \begin{bmatrix} w_1 & 0 \\ 0 & w_2 \end{bmatrix}.$$

Then express the family of weighted "instantaneous inverse kinematic" solutions in terms of δx_{ee} and δq_0:

$$\delta q = J^\# \delta x_{ee} + [I - J^\#J]\delta q_0$$

vi. Given a particular δx_{ee} and $\delta q_0 = 0$, qualitatively compare the solutions δq obtained with J^+ and $J^\#$ when the latter is calculated with $W = A$.

(b) Puma end-effector position control

Now let us investigate the control of task-redundant robots through joint torques. The PUMA(560) arm is a 6-DOF manipulator you are probably familiar with through CS223A and the lecture slides. With respect to the 3-DOF positioning task at the end-effector, the robot is redundant.

i. After following the instructions to upgrade the dependencies above, compile and run the source file `hw2/p1-main.cpp`. Note the difference between this and the robot behavior in homework 1.

```
$ cd bin
$ pushd ../build && cmake -DCMAKE_BUILD_TYPE=Release .. && make && popd
$ ./hw2-p1 1
```

The robot should be in free fall under gravity when you run the starter code. This is because we are using the dynamics simulation framework for this homework, but no control torques are applied by default.

ii. Implement the following PD controller in the area marked as "FILL ME IN". This controller is designed to hold the end effector tip position stationary while providing some joint damping to stabilize the simulation. Note that we do not compensate for Coriolis or centrifugal forces as in practice, they tend to destabilize the controller due to estimation errors.

$$\Gamma = J_v^T(\Lambda_v(-k_{px}(x - x_d) - k_{vx}\dot{x}) + p) + A(-k_{vj}\dot{q})$$

where J_v is the linear velocity Jacobian at the end-effector tip, x is the end-effector tip position and x_d is the end-effector tip desired position, all calculated with respect to the base frame. Use $\mathbf{x_d} = [\mathbf{-0.15\ \ 0.81\ \ 0.58}]^\mathbf{T}$, $\mathbf{k_{px} = 50}$, $\mathbf{k_{vx} = 20}$, $\mathbf{k_{vj} = 20}$. These values are already set for you in the variables `ee_des_pos`, `kpx`, `kvx` and `kvj` respectively in the starter code.

Set your joint control torques `tau` as per the above control law in case **PART1** of the switch block. Run your code with a "1" argument to the executable as shown below

```
$ pushd ../build && cmake -DCMAKE_BUILD_TYPE=Release .. && make && popd
$ ./hw2-p1 1
```

What happens if you remove the joint damping? That is, what is the difference in observed behavior between the above controller and the one below?

$$\Gamma = J_v^T(\Lambda_v(-k_{px}(x - x_d) - k_{vx}\dot{x}) + p)$$

iii. Now, let us try performing some null-space (or self-) motions. First, lets use the following pseudo-inverse based null-space torque controller. We control only the second joint position while damping the motion on other joints and compensating for gravity.

$$\Gamma = J_v^T(\Lambda_v(-k_{px}(x - x_d) - k_{vx}\dot{x}) + p) + [I - J_v^T J_v^{+T}](A(-k_{pj}(q - q_d) - k_{vj}\dot{q}) + g)$$

where

$$q_d = \begin{bmatrix} q_1 \\ -\frac{\pi}{8} + \frac{\pi}{8}\sin\frac{2\pi t}{10} \\ q_3 \\ q_4 \\ q_5 \\ q_6 \end{bmatrix}$$

and t is time in seconds. x_d is the same as the previous part. Use $k_{pj} = 50$ (available as variable `kpj`).

Set your joint control torques `tau` as per the above control law in case **PART2** of the switch block. Run your code with a "2" argument to the executable as shown below

```
$ pushd ../build && cmake -DCMAKE_BUILD_TYPE=Release .. && make && popd
$ ./hw2-p1 2
```

Does the end effector position remain stationary?

iv. Finally, let us try to perform the same null-space motion, but with the *dynamically consistent generalized Jacobian inverse*.

$$\Gamma = J_v^T(\Lambda_v(-k_{px}(x - x_d) - k_{vx}\dot{x}) + p) + [I - J_v^T \bar{J}_v^T](A(-k_{pj}(q - q_d) - k_{vj}\dot{q}) + g)$$

where x_d and q_d are the same as the previous parts.

Set your joint control torques `tau` as per the above control law in case **PART3** of the switch block. Run your code with a "3" argument to the executable as shown below

```
$ pushd ../build && cmake -DCMAKE_BUILD_TYPE=Release .. && make && popd
$ ./hw2-p1 3
```

Now, does the end effector position remain stationary? Briefly explain why the dynamically consistent generalized inverse works but the pseudo-inverse does not.

Note: The end-effector tip is located at $[-0.2 \;\; 0 \;\; 0]$ in the last joint frame, and is available through the variable `ee_pos_local` in the starter code. So, you have to use the following commands from the model interface to evaluate the position and linear velocity Jacobian at the tip of the end-effector.

```
// calculate end-effector tip position
Eigen::Vector3d ee_x;
robot->position(ee_x, ee_link_name, ee_pos_local);

// calculate end-effector tip linear velocity Jacobian
Eigen::VectorXd Jv(3, robot->dof());
robot->Jv(Jv, ee_link_name, ee_pos_local);
```

Problem 2 - Operational space trajectory tracking

In this problem, you will implement a full trajectory tracking controller on the KUKA-IIWA simulation model. Unlike the position-based controller you had implemented in homework 1, you will need to control the robot through joint torques this time around.

The desired end-effector trajectory is identical to the one you implemented in homework 1:

$$\begin{aligned} x_d &= 0 \\ y_d &= 0.5 + 0.1 \cos \frac{2\pi t}{5} \\ z_d &= 0.65 - 0.05 \cos \frac{4\pi t}{5} \end{aligned}$$

where t is time in seconds. The desired orientation trajectory is represented in Euler parameters as $\lambda_d = (\lambda_{0,d}, \lambda_{1,d}, \lambda_{2,d}, \lambda_{3,d})$ where

$$\begin{aligned} \lambda_{0,d} &= \frac{1}{\sqrt{2}} \sin \left(\frac{\pi}{4} \cos \frac{2\pi t}{5} \right) \\ \lambda_{1,d} &= \frac{1}{\sqrt{2}} \cos \left(\frac{\pi}{4} \cos \frac{2\pi t}{5} \right) \\ \lambda_{2,d} &= \frac{1}{\sqrt{2}} \sin \left(\frac{\pi}{4} \cos \frac{2\pi t}{5} \right) \\ \lambda_{3,d} &= \frac{1}{\sqrt{2}} \cos \left(\frac{\pi}{4} \cos \frac{2\pi t}{5} \right) \end{aligned}$$

(a) Desired operational space acceleration

Find $\ddot{\mathbf{x}}_d$, where $\mathbf{x}_d = [x_d \; y_d \; z_d \; \lambda_{0,d} \; \lambda_{1,d} \; \lambda_{2,d} \; \lambda_{3,d}]^T$ is the desired acceleration in operational space coordinates.

(b) Desired end-effector linear and angular accelerations

Give an expression for the linear and angular end-effector accelerations $\dot{v}$ and $\dot{\omega}$ in terms of the operational space position, velocity and acceleration.

(c) Operational space controller design

Propose an operational space controller that meets the following requirements:

- Track the desired operational space trajectory with dynamically decoupled PD control and feed-forward acceleration
- Damp the null-space motion, and
- Compensate for gravitational forces

Note that you do not need to compensate for Coriolis/centrifugal forces.

(d) Simulation

Compile the starter code located under `hw2/p2-main.cpp` as follows:

```
$ cd bin
$ pushd ../build && cmake -DCMAKE_BUILD_TYPE=Release .. && make && popd
$ ./hw2-p2
```

As with the previous problem, you should see the robot simply falling under the effect of gravity.

Now, implement your proposed controller in the area marked as "FILL ME IN" in `hw2/p2-main.cpp`. The resulting motion should be identical to the motion you observed in homework 1.

Problem 3 - Unified motion and force control

In this problem, you will once again control the KUKA-IIWA arm to track a desired motion at its end effector. However, certain directions will be controlled to apply open loop forces rather than perform motions.

The desired end-effector trajectory is identical to the one you implemented in homework 1 and the previous problem, with the exception that we no longer desire a fixed motion along the x direction. Instead, we require a force of $10N$ to be applied in the x direction while maintaining a zero moment about the y and z axes.

$$y_d = 0.5 + 0.1 \cos \frac{2\pi t}{5}$$
$$z_d = 0.65 - 0.05 \cos \frac{4\pi t}{5}$$

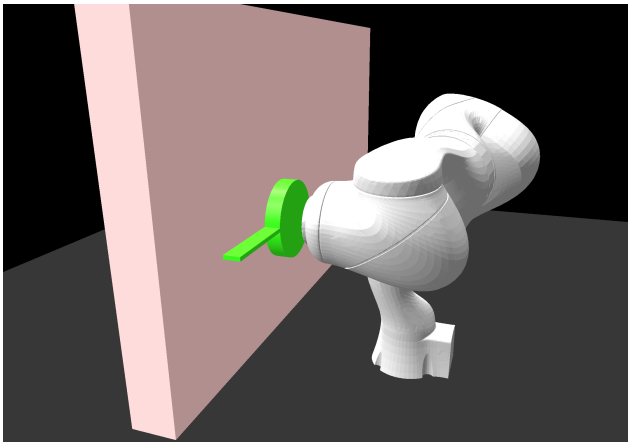
where t is time in seconds. The desired orientation trajectory is represented in Euler parameters as $\lambda_d = (\lambda_{0,d}, \lambda_{1,d}, \lambda_{2,d}, \lambda_{3,d})$ where

$$\lambda_{0,d} = \frac{1}{\sqrt{2}} \sin\left(\frac{\pi}{4} \cos \frac{2\pi t}{5}\right)$$
$$\lambda_{1,d} = \frac{1}{\sqrt{2}} \cos\left(\frac{\pi}{4} \cos \frac{2\pi t}{5}\right)$$
$$\lambda_{2,d} = \frac{1}{\sqrt{2}} \sin\left(\frac{\pi}{4} \cos \frac{2\pi t}{5}\right)$$
$$\lambda_{3,d} = \frac{1}{\sqrt{2}} \cos\left(\frac{\pi}{4} \cos \frac{2\pi t}{5}\right)$$

The desired end effector forces are constant at

$$F_{x,d} = 10$$
$$M_{y,d} = 0$$
$$M_{z,d} = 0$$

As shown below, the forces are to be balanced by contact against a plane surface located at $x = 0.05$.



(a) Force, motion selection matrices

Find the selection matrices Ω and $\bar{\Omega}$ that separate the controlled force directions from the controlled motion directions at the end-effector.

(b) Unified motion and force controller

Modify the controller from the previous problem to produce the desired response above. For controlling the end-effector forces, use feedforward control only.

(c) Simulation

Compile the starter code located under `hw2/p3-main.cpp` as follows:

```
$ cd bin
$ pushd ../build && cmake -DCMAKE_BUILD_TYPE=Release .. && make && popd
$ ./hw2-p3
```

Implement your proposed controller in the area marked as "FILL ME IN" in `hw2/p3-main.cpp`. The resulting motion should be identical to the motion you observed in homework 1 and the previous problem, except the end effector should be flat against the given surface.

Hint 1: To verify that your robot is touching the surface, you can use the readings from the simulated force sensor as follows

```
Eigen::Vector3d sensed_force;
Eigen::Vector3d sensed_moment;
force_sensor->getForce(sensed_force);
force_sensor->getMoment(sensed_moment);
```

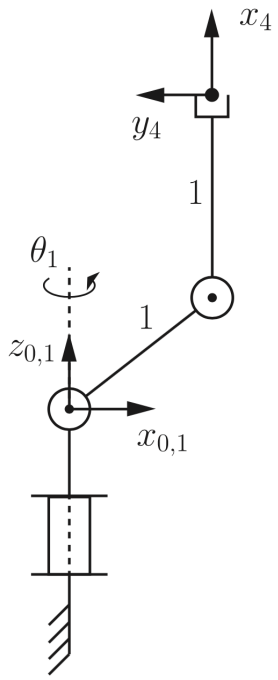
The above variables `sensed_force` and `sensed_moment` are already updated and made available to you in the control loop.

Hint 2: Due to uncompensated Coriolis and centrifugal forces at the end effector, it is possible for the end effector to lose contact from the surface intermittently, in spite of the feedforward control force. You don't have to worry about that for this homework as long as the end effector continues to follow the desired trajectory on the surface after wards.

Hint 3: You can switch between a few different pre-defined camera views by commenting and uncommenting lines 29 - 31.

Problem 4 - Singularity control

Consider the manipulator shown below.



In the configuration $\theta_1 = 0, \theta_2 = 90°$, $\theta_3 = 0°$, the manipulator is at a singularity with respect to the end-effector positioning task.

i. How many degrees of freedom does the end effector have in this configuration?

ii. What is(are) the singular direction(s) in the $\{0\}$ frame?

iii. What is(are) the direction(s) orthogonal to the singular direction(s)?

iv. What type(s) of singularity(ies) are present in this configuration?

v. Briefly describe how you would control the robot close to this configuration in case you needed the end-effector to be moved along the singular direction(s).