Advanced Robotic Manipulation Handout

CS327A (Spring 2017)

Problem Set #1

Due **Wed, Apr** 19^{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.* Installation instructions are provided below.
- 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 through the Gradescope website directly. `https://gradescope.com/courses/7467`

SAI2 Installation Instructions

Attention: Windows users

Unfortunately, Windows is currently not a supported platform for the SAI2 framework. We encourage you to install a virtual machine with "Ubuntu 16.04LTS" to use for this and future homeworks. VMware is available for Stanford students for free. Go to `https://stanford.onthehub.com/WebStore/Welcome.aspx`, login (top right), search "VMware" in the top box, select "VMware Workstation 12". Make sure you allocate at least 2Gb of memory for the virtual machine to work properly.

After installing Ubuntu, follow the instructions below for Ubuntu.

Installation: macOS Sierra

SAI2 requires a number of components to run on macOS. The following instructions will help you through it. Open Terminal.app, then run the following commands. You may skip any components you have already installed on your system.

1. Command line tools:

   ```
   $ xcode-select --install
   ```

2. Homebrew (macOS package manager):

   ```
   $ /usr/bin/ruby -e "$(curl -fsSL
   https://raw.githubusercontent.com/Homebrew/install/master/install)"
   ```

3. CMake (cross-platform build system):

   ```
   $ brew install cmake
   ```

4. Eigen3 (linear algebra):

   ```
   $ brew install eigen
   ```

5. TinyXML2 (xml parsing):

   ```
   $ brew install tinyxml2
   ```

6. GLFW (UI window manager):

   ```
   $ brew install glfw
   ```

7. Chai3d (graphics and haptics library):

   ```
   $ git clone https://github.com/chai3d/chai3d.git
   $ cd chai3d && mkdir -p build && cd build
   $ cmake -DCMAKE_BUILD_TYPE=Release .. && make -j4 && cd ../../
   ```

8. RBDL (Rigid body dynamics library): Note that RBDL must be installed on the system to be used. So you will be prompted for your system password during the "make install"

```
$ curl -O https://bitbucket.org/rbdl/rbdl/get/849d2aee8f4c.zip
$ unzip 849d2aee8f4c.zip
$ cd rbdl-rbdl-849d2aee8f4c && mkdir -p build && cd build
$ cmake -DCMAKE_BUILD_TYPE=Release -DRBDL_BUILD_ADDON_URDFREADER=ON
-DRBDL_USE_ROS_URDF_LIBRARY=OFF .. && make && sudo make install && cd ../../
```

9. SAI2-Simulation (Rigid body simulation library): Note that this library is currently closed-source. It is only being made available for students enrolled in the class. You will need to enter your SUNET-ID password when prompted.

```
$ scp <your-sunet-id>@cardinal.stanford.edu:
/afs/ir.stanford.edu/class/cs327a/sai2-simulation/sai2-simulation.zip .
$ unzip sai2-simulation.zip
$ cd sai2-simulation && mkdir -p build && cd build
$ cmake -DCMAKE_BUILD_TYPE=Release .. && make && cd ../../
```

10. SAI2-Common (Simplified robot simulation library):

```
$ git clone https://github.com/manips-sai/sai2-common.git
$ cd sai2-common && mkdir -p build && cd build
$ cmake -DCMAKE_BUILD_TYPE=Release .. && make && cd ../../
```

11. CS327A class repository (homeworks for this class): Note that instructions for particular problems are in the problem details below.

```
$ git clone https://github.com/shameekganguly/cs327a-dist.git cs327a
$ cd cs327a && mkdir -p build && cd build
$ cmake -DCMAKE_BUILD_TYPE=Release .. && make && cd ../
```

12. Robot mesh files (for visualization): You will need to enter your SUNET-ID password when prompted.

```
$ scp -r <your-sunet-id>@cardinal.stanford.edu:
/afs/ir.stanford.edu/class/cs327a/meshes/kuka_iiwa/meshes
resources/kuka_iiwa/
$ scp -r <your-sunet-id>@cardinal.stanford.edu:
/afs/ir.stanford.edu/class/cs327a/meshes/puma/meshes
resources/puma/
```

Installation: Ubuntu 16.04

SAI2 requires a number of components to run on Ubuntu as well. The following instructions will help you through it. Open command terminal, then run the following commands. You may skip any components you have already installed on your system.

1. C++ build tools:

```
$ sudo apt-get install build-essential
```

2. CMake (cross-platform build system):

```
$ sudo apt-get install cmake
```

3. Eigen3 (linear algebra):

```
$ sudo apt-get install libeigen3-dev
```

4. TinyXML2 (xml parsing):

```
$ sudo apt-get install libtinyxml2-dev
```

5. GLFW (UI window manager):

```
$ sudo apt-get install libglfw3-dev
```

6. Chai3d (graphics and haptics library):

```
$ sudo apt-get install libasound2-dev libusb-1.0.0-dev libxmu-dev libxi-dev
$ git clone https://github.com/chai3d/chai3d.git
$ cd chai3d && mkdir -p build && cd build
$ cmake -DCMAKE_BUILD_TYPE=Release .. && make -j4 && cd ../../
```

7. RBDL (Rigid body dynamics library): Note that RBDL must be installed on the system to be used. So you will be prompted for your system password during the "make install"

```
$ curl -O https://bitbucket.org/rbdl/rbdl/get/849d2aee8f4c.zip
$ unzip 849d2aee8f4c.zip
$ cd rbdl-rbdl-849d2aee8f4c && mkdir -p build && cd build
$ cmake -DCMAKE_BUILD_TYPE=Release -DRBDL_BUILD_ADDON_URDFREADER=ON
-DRBDL_USE_ROS_URDF_LIBRARY=OFF .. && make && sudo make install && cd ../../
```

8. SAI2-Simulation (Rigid body simulation library): Note that this library is currently closed-source. It is only being made available for students enrolled in the class. You will need to enter your SUNET-ID password when prompted.

```
$ scp <your-sunet-id>@cardinal.stanford.edu:
/afs/ir.stanford.edu/class/cs327a/sai2-simulation/sai2-simulation.zip .
$ unzip sai2-simulation.zip
$ cd sai2-simulation && mkdir -p build && cd build
$ cmake -DCMAKE_BUILD_TYPE=Release .. && make && cd ../../
```

9. SAI2-Common (Simplified robot simulation library):

```
$ git clone https://github.com/manips-sai/sai2-common.git
$ cd sai2-common && mkdir -p build && cd build
$ cmake -DCMAKE_BUILD_TYPE=Release .. && make && cd ../../
```

10. CS327A class repository (homeworks for this class): Note that instructions for particular problems are in the problem details below.

```
$ git clone https://github.com/shameekganguly/cs327a-dist.git cs327a
$ cd cs327a && mkdir -p build && cd build
$ cmake -DCMAKE_BUILD_TYPE=Release .. && make && cd ../
```

11. Robot mesh files (for visualization): You will need to enter your SUNET-ID password when prompted.

```
$ scp -r <your-sunet-id>@cardinal.stanford.edu:
/afs/ir.stanford.edu/class/cs327a/meshes/kuka_iiwa/meshes
resources/kuka_iiwa/
$ scp -r <your-sunet-id>@cardinal.stanford.edu:
/afs/ir.stanford.edu/class/cs327a/meshes/puma/meshes
resources/puma/
```

Problem 1 - Revision: Introduction to Robotics

(a) D-H Convention

Draw the schematic of one manipulator of each type below. Clearly label the base frame $\{0\}$ as well as each joint frame. Indicate the joint co-ordinates on the schematic.

i. RR (planar)

ii. RR (non-planar)

iii. RRP (planar)

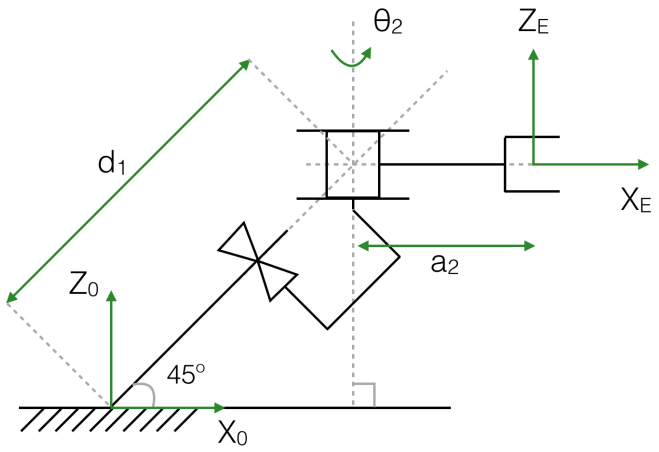
iv. PRR (non-planar)

With respect to a positioning task at the end effector, which of the manipulators above are redundant?

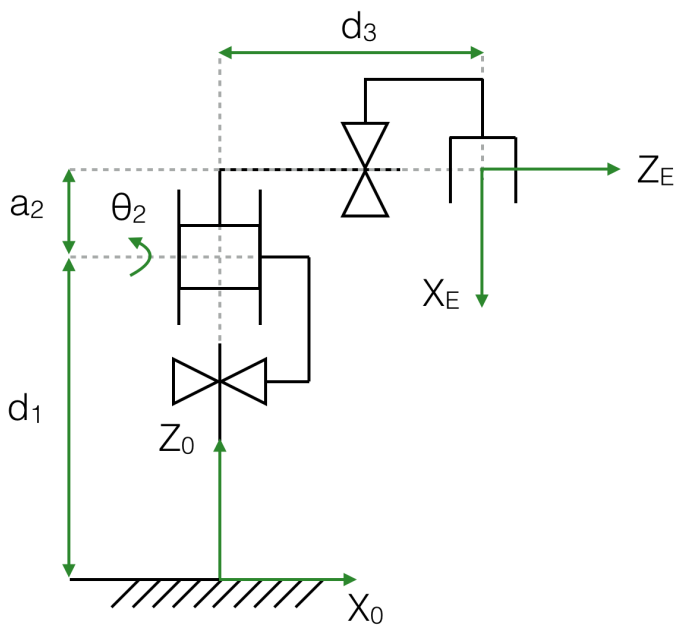
(b) Basic Jacobian

Write down the linear (3 x n) and angular velocity (3 x n) Jacobians at the end-effector for the following manipulators *in the given configuration* in frame $\{0\}$. Try to do so by using the explicit form and visualizing the resulting motion at the end-effector when each joint is moved with unit speed.

i. PR (non-planar): As shown, $d_1 = 1$, $\theta_2 = 0$ such that X_E is parallel to X_0.



ii. PRP (non-planar): As shown, $d_1 = 1$, $\theta_2 = 0$, $d_3 = 1$ such that X_E is parallel to Z_0.


d3
ZE
a2
θ2
XE
d1
Z0
X0

Problem 2 - Quaternion algebra

Unit quaternions or Euler parameters are ubiquitously used in robotics to describe rotations between coordinate frames. In this problem, we explore some of their properties as rotation operators.

As mentioned in class, quaternions are described in a 4-dimensional vector space, $\mathbb{H}$. The unit vectors for this space however are defined in the following manner, similar to complex numbers

$$h = \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} \equiv w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$$

where

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1.$$

w is called the **scalar** component of the quaternion, and (x, y, z) is called the **vector** component of the quaternion.

Multiplying two quaternions is also similar to multiplying two complex numbers:

$$\begin{aligned}(w_1 + x_1\mathbf{i} + y_1\mathbf{j} + z_1\mathbf{k})(w_2 + x_2\mathbf{i} + y_2\mathbf{j} + z_2\mathbf{k}) = \\ (w_1w_2 - x_1x_2 - y_1y_2 - z_1z_2)+ \\ (w_1x_2 + w_2x_1 + y_1z_2 - y_2z_1)\mathbf{i}+ \\ (w_1y_2 + w_2y_1 - x_1z_2 + x_2z_1)\mathbf{j}+ \\ (w_1z_2 + w_2z_1 + x_1y_2 - x_2y_1)\mathbf{k}\end{aligned}$$

Note above, that the multiplication operation is non-commutative. That is, $h_1h_2 \neq h_2h_1$ usually.

The quaternion $1 + 0\mathbf{i} + 0\mathbf{j} + 0\mathbf{k}$ is called the **identity quaternion**, because multiplying it to any quaternion leaves the quaternion unchanged. For simplicity, the identity quaternion is often denoted as 1.

(a) Conjugation

The conjugate of a quaternion h is defined as $\bar{h}$, such that

$$\bar{h} \equiv w - x\mathbf{i} - y\mathbf{j} - z\mathbf{k}$$

Show that $h\bar{h} = \bar{h}h = (w^2 + x^2 + y^2 + z^2) + 0\mathbf{i} + 0\mathbf{j} + 0\mathbf{k}$.

(b) Norm of a quaternion

Similar to complex numbers, the norm of a quaternion is defined to be the scalar component of the product of the quaternion with its conjugate.

$$||h|| \equiv \sqrt{w^2 + x^2 + y^2 + z^2} \equiv \sqrt{h\bar{h}}$$

Quaternions which possess the property that $(w^2 + x^2 + y^2 + z^2) = 1$ are called **unit quaternions** and have unit norm. Show that the product of two unit quaternions is a unit quaternion.

(c) Rotation operation as similarity transform

Let $r = [r_x\, r_y\, r_z]^T$ be a vector in the 3-D vector space, $\mathbb{R}^3$. Now, we construct a quaternion $h_r = 0 + r_x\mathbf{i} + r_y\mathbf{j} + r_z\mathbf{k}$ whose vector component is identical to the components of r. Also consider a *unit* quaternion $\lambda = \lambda_0 + \lambda_1\mathbf{i} + \lambda_2\mathbf{j} + \lambda_3\mathbf{k}$.

Show that the vector component of the triple product $h_{r'} = \lambda h_r \bar{\lambda}$ is identical to the components of the (3-dimensional) matrix-vector product

$$\begin{bmatrix} 2(\lambda_0^2 + \lambda_1^2) - 1 & 2(\lambda_1\lambda_2 - \lambda_0\lambda_3) & 2(\lambda_1\lambda_3 + \lambda_0\lambda_2) \\ 2(\lambda_1\lambda_2 + \lambda_0\lambda_3) & 2(\lambda_0^2 + \lambda_2^2) - 1 & 2(\lambda_2\lambda_3 - \lambda_0\lambda_1) \\ 2(\lambda_1\lambda_3 - \lambda_0\lambda_2) & 2(\lambda_2\lambda_3 + \lambda_0\lambda_1) & 2(\lambda_0^2 + \lambda_3^2) - 1 \end{bmatrix} \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix}$$

Also give the scalar component of the triple product $h_{r'}$.

You should already be familiar with the expression above from equation 1.25 in the course reader. To summarize, unit quaternions can be used to construct rotation operators which act through the triple product $\lambda h_r \bar{\lambda}$ on h_r in the quaternion space. In doing so, we first convert the 3-dimensional vector r into a quaternion with no scalar component, $h_r = 0 + r_x\mathbf{i} + r_y\mathbf{j} + r_z\mathbf{k}$. The triple product gives back another quaternion with no scalar component, $h_{r'} = 0 + r'_x\mathbf{i} + r'_y\mathbf{j} + r'_z\mathbf{k}$ which can be converted back to the 3-dimensional space rotated vector $r' = [r'_x\, r'_y\, r'_z]^T$.

(d) Composition of two rotations

Suppose that unit quaternions λ_1 and λ_2 represent two rotation operations. Given a 3-dimensional vector r, show that rotating r first by λ_1, and then by λ_2 is equivalent to rotating r by the resultant of the product $\lambda_2\lambda_1$.

Problem 3 - Instantaneous inverse kinematics

In this problem, you will program a "position-controlled" 7-DOF KUKA-IIWA manipulator in SAI2 to follow a desired end-effector trajectory. The desired position trajectory to be followed is represented by

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

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

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

The desired operational space coordinates are thus given by $\mathbf{x}_d = [x_d \ y_d \ z_d \ \lambda_{0,d} \ \lambda_{1,d} \ \lambda_{2,d} \ \lambda_{3,d}]^T$.

(a) Desired operational space velocity

Find an expression for the desired operational space velocity vector $\dot{\mathbf{x}}_d = [\dot{x}_d \ \dot{y}_d \ \dot{z}_d \ \dot{\lambda}_{0,d} \ \dot{\lambda}_{1,d} \ \dot{\lambda}_{2,d} \ \dot{\lambda}_{3,d}]^T$. It is sufficient to report just the individual components of the above vector.

(b) Desired end-effector linear and angular velocities

Given an expression for the E and E^+ matrices in terms of x, y, z, λ_0, λ_1, λ_2 and λ_3.

(c) Simulation: getting started

To track the desired end-effector trajectory with the "position-controlled" KUKA-IIWA in simulation, we will need to calculate the desired joint velocities $\dot{q}_d$ first, then integrate them to obtain the desired joint positions. We assume that the manipulator starts with the pose of its end-effector identical to $\mathbf{x}_d(t = 0)$. Thus, the algorithm we run at each time step is as follows

1. Evaluate $\mathbf{x}_d$ at time t
2. $[v_d^T \ \omega_d^T]^T = E^+ \dot{\mathbf{x}}_d$

3. $\dot{q}_d = J_0^{\#}[v_d^T \ \omega_d^T]^T$
4. $q \leftarrow q + \dot{q}_d \, \Delta t$

where Δt is the simulation time step and $J_0^{\#}$ is a right generalized inverse of J_0.

For this problem, use the inertia weighted right generalized inverse.

$$J_0^{\#} = A^{-1} J_0^T \left(J_0 A^{-1} J_0^T \right)^{-1}$$

Note that the last step above is already implemented for you in the starter code.

To get started, open terminal and change directory into the `cs327a` directory installed earlier. Run

```
$ mkdir -p build && cd build && cmake -DCMAKE_BUILD_TYPE=Release .. && cd ..
```

This compiles the starter code under `cs327a/hw1/` and creates the executable `cs327a/bin/hw1-p1`. Note that this executable must be run from within the `cs327a/bin` directory for the graphic models to be loaded correctly.

Run the `hw1-p1` executable to verify that the program starts and a static KUKA-IIWA manipulator is visible.

```
$ cd bin
$ ./hw1-p1
```

Press ESC to close the window.

(d) Simulation: implementation

Implement the above algorithm in the space marked as `"FILL ME IN"` in `cs327a/hw1/p1-main.cpp`. Submit your completed `cs327a/hw1/p1-main.cpp` file and any other source files you added along with your homework write-up.

Hint 1.: You will need to familiarize yourself with the popular Eigen linear algebra library used in the simulation. See `https://eigen.tuxfamily.org/dox-devel/group__QuickRefPage.html` for a quick reference. A cheat sheet is available under `https://eigen.tuxfamily.org/dox/AsciiQuickReference.txt`. If you need to use the *Eigen* :: *Quaterniond* class for quaternion operations, refer to `https://eigen.tuxfamily.org/dox/classEigen_1_1Quaternion.html`.

Hint 2.: We use RBDL (rigid body dynamics library) to model the robot. An easy to use interface class is provided for it under `sai2-common/src/model/ModelInterface.h`. This is also used in the simulation through the instantiated variable called `robot`. The following member variables of this class might be of interest to you:

```
/// \brief Joint positions
Eigen::VectorXd _q;
```

```
/// \brief Joint velocities
Eigen::VectorXd _dq;

/// \brief Mass Matrix
Eigen::MatrixXd _M;

/// \brief Inverse of the mass matrix
Eigen::MatrixXd _M_inv;
```

Also of interest are the following member functions:

```
/*** @brief Full jacobian for point on link, relative to base
* @param J Matrix to which the jacobian will be written
* @param link_name the name of the link where to compute the jacobian
* @param pos_in_link the position of the point in the link where the jacobian
* is computed (in local link frame)
*/
void J(Eigen::MatrixXd& J, const std::string& link_name,
    const Eigen::Vector3d& pos_in_link);

/**
* @brief Position from base to point in link, in base coordinates
* @param pos Vector of position to which the result is written
* @param link_name name of the link in which is the point where to compute the position
* @param pos_in_link the position of the point in the link, in local link frame
*/
void position(Eigen::Vector3d& pos, const std::string& link_name,
    const Eigen::Vector3d& pos_in_link);

/**
* @brief Rotation of a link with respect to base frame for the last updated configuration
* @param rot Rotation matrix to which the result is written
* @param link_name name of the link for which to compute the rotation
*/
void rotation(Eigen::Matrix3d& rot, const std::string& link_name);
```

!!Caution!! The Jacobian written by J is of the form $[J_\omega^T \; J_v^T]^T$ which has the linear and angular velocity Jacobians in the opposite order compared to the basic Jacobian J_0 discussed in class.

Hint 3. To compile the homework from within the `cs327a/bin` folder, run

```
$ pushd ../build && cmake -DCMAKE_BUILD_TYPE=Release .. && popd
```