Advanced Robotic Manipulation (CS 327A) Homework #5

(Spring 2022/2023) Due: **Monday, May 22, 11:59pm**

In this homework you will explore cooperative robot control through the Augmented Object and Virtual Linkage models.

Example Videos

You can find example videos for the SAI portion of this homework at this link: TO BE ADDED IN V2 HW UPDATE

You must be logged into your Stanford Google account to access the videos.

Code Submission Instructions

For Problem 2, submit your controller files to the Gradescope assignment.

Additionally, you should record a video of the simulation running in each part, upload this video somewhere accessible (Google Drive or Youtube both work well), and then include a link to the video for each part.

Alternatively, you may come by Wesley's office hours or email him to schedule a Zoom session in order to show your working simulations.

Code Update - CODE NOT YET AVAILABLE, WILL WORK ON V2 RELEASE

You can find the new homework code by navigating to the cs327a directory in your computer and following the steps below:

1. Stash your changes and pull the new updates (which will include the solutions) by using the following commands:

```
$ git stash
$ git pull
```

2. Retrieve your own changes (that you previously stashed) by typing:

```
$ git stash pop
```

3. Build the updated repo by navigating to cs327a/build and running:

```
$ cmake .. && make
```

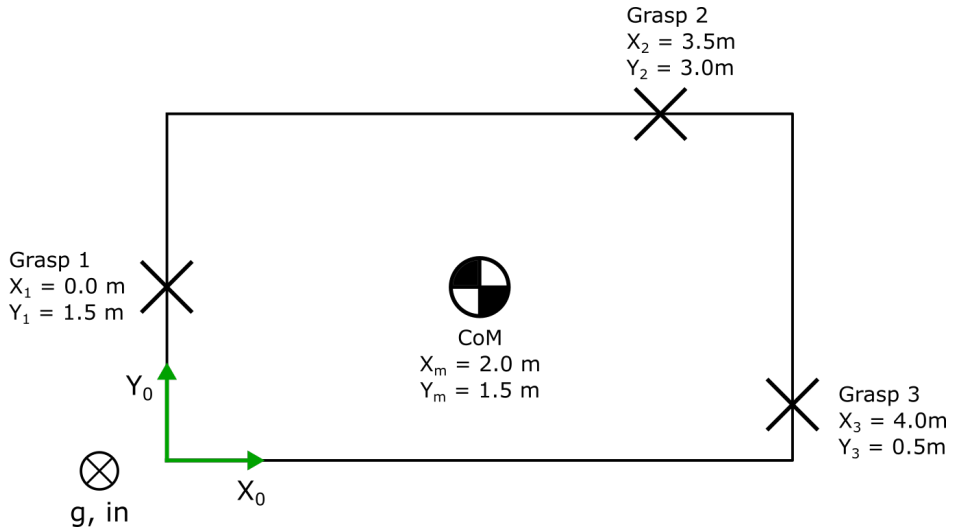
4. The new code is available in the hw5 folder

5. **NOTE:** This week's homework uses a legacy application development paradigm for SAI2 simulations, so you will only see one `.cpp` source file. It contains both the simulation and the control threads. Functionally, this won't make a difference for your implementation of the controller. Just place your code in the fill-in section as indicated.

   redis is not used for communication here between the simulator and the controller in problem 2, but one key is still published for plotting purposes.

Problem 1: Augmented Object and Virtual Linkage Warm-Up

Three students are trying to cooperatively move a rigid wooden table. Each person holds the table with only one arm, at the grasp points shown below (the figure represents the view as seen from above). You may assume the students maintain grip without slipping, and gravity pulls down in the negative Z_0 direction (into the page).



Suppose we want to simply hold the table above the ground. This requires a net vertical force of 100N, with no horizontal forces and no net moments. In order for the students to maintain grip, we also require a compression force of 5N between each of the grasp points, but no internal moments.

(a) Find the W matrix which relates resultant forces/moments on the table with the forces/moments applied by each person at the grasping points, in reference frame $\{0\}$.

(b) Find the E matrix which relates the forces applied by each person at the grasping points with internal forces.

(c) (i) Write a symbolic expression for the grasp matrix G
(ii) What are the dimensions of each of the blocks that form G?
(iii) How do you compute $\bar{E}$? You don't need to substitute with number, simply write down the symbolic expression for $\bar{E}$

(d) If the students stand still and don't use their lower bodies, we can model each arm as a 7 DOF spatial manipulator. So the whole system has 21 DOF in total whereas the object by itself has 6 DOF.

(i) How many DOF contribute to internal forces and moments between or at the grippers?
(ii) If the object is to be held stationary, can the students still move parts of their arm? If so, how many DOF of motion does each student still have with the object held stationary? The sum of the above DOF for all three students gives the total redundant DOF of motion of the system.

**** Hint **** The sum of the DOF of motion of the object, the number of internal forces and moments between the end-effectors, and the redundant DOF of motion should equal the total DOF of the system.

Problem 2 - Centralized cooperative manipulation control

For many manipulation tasks, a single robot might fall short to lift and move an object around due to limited motor capacity. As a result, multiple robots might be required to manipulate the object. In class, we studied two useful models for working with multi-robot systems:

1. **Augmented Object Model** → allows us to study how the dynamics of the system change when multiple arms grasp the same object
2. **Virtual Linkage Model** → when multiple robots hold the same object, we must explicitly control the internal forces, moments and tensions that appear in the object to (a) prevent damage to the object and (b) control slippage. The virtual linkage model provides a mathematical foundation for expressing these constraint forces as a function of the forces and moments that the manipulators apply at the grasping points.

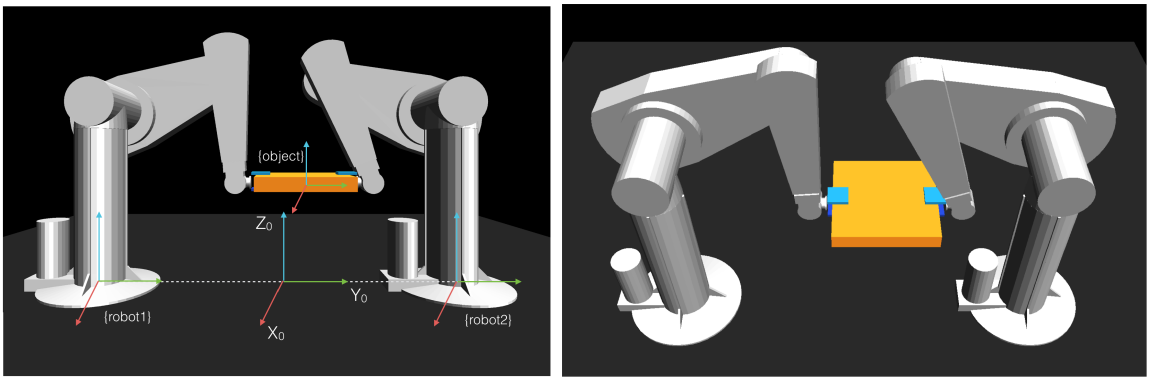
This problem is purely coding and simulation. You do not need to include any symbolic expressions in your homework write-up. Simply implement each section in code.

The deliverables you should include in your writeup are

1. A link to a video of your simulation window
2. A screenshot of the web interface window where the plotting interface shows the position of the graspsed object over time (see section (a) for more details).
3. A copy of your `hw5_p2.cpp` file, to be submitted in a seperate Gradescope assignment. This is primarily for me to be able to assign partial credit.

Alternatively, you may come by Wesley's office hours or email him to schedule a Zoom session in order to show your working simulations. You will still need to submit the code file in this case.

(a) Set up



The co-operative manipulation set-up for this problem uses two 6 DOF PUMA robots as shown below. Each PUMA is has a gripper that is modeled as a prismatic joint attached to the end-effector link.

The manipulated object is a $500g$ box and the robots grip it at two ends along the Y-axis in the local frame (see above figures). We model the object in simulation as having 6 DOF (equivalent to three prismatic joints and three revolute). The last joint frame is co-located with the center of mass frame for the object.

1. The only file that you need to fill in with your own code is `hw5_p2.cpp`, specifically in the controller function.

2. To run this problem, run the launch script

   ```
   ./launch_hw5_p2.sh
   ```

   This time the script will start the combined simulation/control application as well as the sai2-interface web plotter. This plotter works the same way as in Homework 3.

3. In the simulation window, you should see the robots slowly falling with the object. The code starts out running a grasp stabilizing controller and then switches to your designed controller once the grasp is stable, which usually takes only a few milliseconds.

4. Once you see a line in the terminal that says something like

   ```
   (#####) wsgi starting up on http://127.0.0.1:8000
   ```

   you can open up the plotter by navigating to localhost:8000 in a Chrome/Chromium or Firefox browser window.

5. Once your simulation is working, you can use this plot to verify the trajectory you have programmed for the object. Make sure to include a screenshot of this plot in your writeup.

(b) Manipulator Jacobians at object COM

In previous homeworks, the robot base frame was located at the world base frame. However, in this problem, the two robots are each located at a distance of $0.6m$ apart from the world frame along the world Y-axis and with the same orientation. ****Note**** you are provided the transformations:

- `robot1_base_frame` → from the world frame to the base frame of robot 1
- `robot2_base_frame` → from the world frame to the base frame of robot 2

To obtain the augmented object model, you will first have to obtain the basic Jacobian for each robot at the center of mass (COM) of the object. Fill in the section titled `FILL ME IN: Manipulator Jacobians` to calculate:

1. `robot1_j0_objcom_world`
2. `robot2_j0_objcom_world`

which are the basic Jacobians for the robots calculated in the end-effector frame (link `end-effector`) at the current object COM location and measured in the world frame.

**** Hints ****

- The simulation provides you the object COM frame as the variable `object_com_frame` which is measured in the world frame.
- The transformations are stored as variables of type `Eigen::Affine3d` which behave similar to 4×4 transformation matrices. For example, you can multiply them together to get chained transformations. You can also get the inverse of a transform `T` as `T.inverse()`. Lastly, you can access the translation part as a `Eigen::Vector3d` object through `T.translation()` and the rotation part as a `Eigen::Matrix3d` through `T.linear()`.
- The transformation from the base frame to a link frame can be obtained through the following `Sai2Model` function (see Sai2Model.h for more details):

  ```
  transform(Eigen::Affine3d& T, const std::string& link_name)
  ```

- Note that the objects `robot1` and `robot2` return frame-dependent measurements such as position and velocity in the robot's base frame, and not in the world frame. You will have to use the frame transformations `robot1_base_frame` and `robot2_base_frame` from the world frame to each of the robot's base frames, to measure such quantities in the world frame.

(c) Augmented object model

Update the section titled `FILL ME IN: Augmented object model` to calculate:

1. `augmented_object_inertia`: the augmented object inertia at the object COM

$$\Lambda_{\oplus}(\mathbf{x}) = \Lambda_{\mathcal{L}}(\mathbf{x}) + \sum_{i=1}^{N} \Lambda_i(\mathbf{x}).$$

2. `augmented_object_p`: the augmented object gravity vector at the object COM:

$$\mathbf{p}_{\oplus}(\mathbf{x}) = \mathbf{p}_{\mathcal{L}}(\mathbf{x}) + \sum_{i=1}^{N} \mathbf{p}_i(\mathbf{x}), \tag{1}$$

   where $\mathbf{p}_{\mathcal{L}}(\mathbf{x})$ and $\mathbf{p}_i(\mathbf{x})$ are the gravity vectors associated with the object and the ith end-effector, respectively.

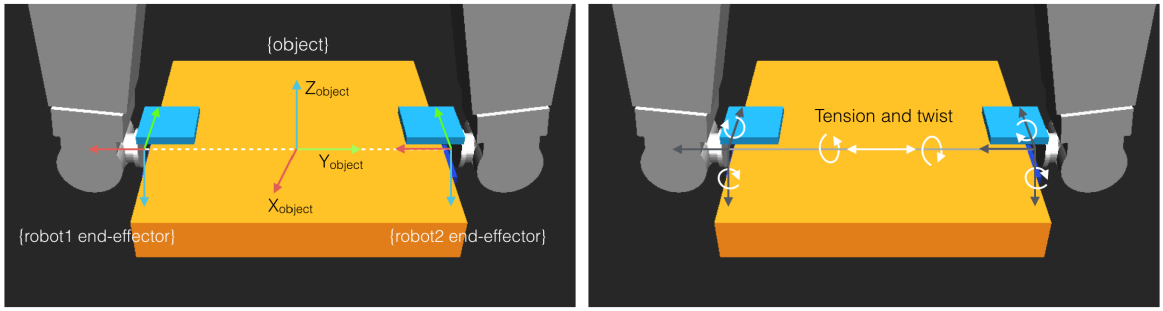
The object's inertia at its COM frame is already computed for you and saved as the variable `object_inertia`.

(d) Grasp matrix

Update the section titled `FILL ME IN: Grasp Matrix` to calculate `G`, the grasp matrix describing the relationship between the forces and moments applied at the two robot end-effectors, and the resultant and internal forces in the object.

**** Hints ****

- Since there are only two robots grasping the object, the internal forces are given by a single tension along the line joining the end-effector frames, and five internal moments. The internal moments are comprised of the end effector moments about X and Z axis in the object COM frame, as well as a twisting moment about the Y axis. The figure below qualitatively shows these quantities:



- Remember that:

$$\begin{bmatrix} \mathbf{f}_r \\ \mathbf{m}_r \\ \mathbf{t} \\ \tau \end{bmatrix} = \underbrace{\begin{bmatrix} W_f & W_m \\ \bar{E} & 0 \\ 0 & I_9 \end{bmatrix}}_{G} \begin{bmatrix} \mathbf{f} \\ \mathbf{m} \end{bmatrix}$$

Here, $\mathbf{f}_r$ and $\mathbf{m}_r$ are the resultant force and moment at some reference point O on the object, $\mathbf{t}$ is the vector of internal tensions, and τ is the vector of internal moments. The matrix G is referred to as the grasp matrix and can be computed following the procedure outlined in Chapter 9 of the course reader. The vectors $\mathbf{f}$ and $\mathbf{m}$ represent the forces and moments applied by the manipulators. The variable $\hat{\mathbf{r}}_i$ is the cross product operator corresponding to the vector $\mathbf{r}_i$ that goes from the reference point O to the ith grasp point.

(e) Force control

Design a controller that:

1. Tracks the following desired object trajectory:

$$\begin{aligned} x_{object} &= 0.0 \\ y_{object} &= 0.15 \sin\left(\frac{2\pi t}{3}\right) \\ z_{object} &= 0.4 \\ \lambda_{0,obj} &= 1.0 \\ \lambda_{1,obj} &= 0.0 \\ \lambda_{2,obj} &= 0.0 \\ \lambda_{3,obj} &= 0.0 \end{aligned}$$

2. Maintain an internal tension of $-15N$ (that is, a compression of $15N$) between the end-effectors

3. Zero internal moments

**** Hint ****

Instead of compensating for gravity on the augmented object, you can compensate for each robot separately, and for the object alone at the operational point. The following controller structure will achieve it:

$$\begin{aligned} \Gamma_1 &= {}^{world}J_{1,ee}^T \, {}^{world}F_{1,ee} + g_1 \\ \Gamma_2 &= {}^{world}J_{2,ee}^T \, {}^{world}F_{2,ee} + g_2 \\ \begin{bmatrix} f_{1,ee} \\ f_{2,ee} \\ m_{1,ee} \\ m_{2,ee} \end{bmatrix} &= G^{-1} \begin{bmatrix} \Lambda_{\oplus} F^*_{motion} + p_{obj} \\ t_{int} \\ \tau_{int} \end{bmatrix} \end{aligned}$$

where ${}^{world}J_{1,ee}$ is the basic Jacobian for the first robot at its end-effector, measured in the world frame. ${}^{world}F_{1,ee} = [f_{1,ee}^T \; m_{1,ee}^T]$ is the desired force and moment vector at the end-effector for the first robot in the world frame. F^*_{motion} is the unit mass control force on the augmented object and p_{obj} is the object gravity vector available through the variable `object_p`. t_{int} and τ_{int} are the internal tension and moments respectively. All other symbols have their standard meaning.

**** Notes about the starter code ****

- Once you have set the joint torques for each robot, `tau1` and `tau2`, an additional gripper force of $15N$ is added to each robot by the starter code. This is to maintain a rigid grip on the object. You should not need to change this.

- If your controller attempts to set a desired torque of $\geq 200Nm$ on any joint, the starter code prints a message saying `"Torque override. User asked torques beyond safe limit."` and switches to a default safe controller to prevent unwanted instability in the simulation. Similarly, if a robot loses grip on the object, the starter code prints a message saying `"Torque override. Robot 1 or 2 dropped object."` and switches to the default safe controller.
- Note that the simulation for this problem has been slowed down by a factor of 10 to allow smooth control with feed-forward forces alone at the end-effector. This should not affect your code at all, but you will notice that the robot's movements are quite sluggish.