

Lab 5: Inverse Kinematics*

EECS/ME/BIOE C106A/206A Fall 2021

Goals

By the end of this lab you should be able to:

- Use MoveIt to compute inverse kinematics solutions
 - Create a visualization of Baxter’s kinematic structure, as defined in the URDF file
 - Open and close Baxter’s grippers programmatically
 - Use MoveIt to move Baxter/Sawyer’s gripper(s) to a specified pose in the world frame and perform a rudimentary pick and place task.
-

Contents

1	Getting started with Git	2
2	Inverse Kinematics	2
2.1	Specify a robot with a URDF	2
2.2	Compute inverse kinematics solutions	2
3	Open loop manipulation with Baxter/Sawyer	3
3.1	Setting up your environment for Rethink robots	3
3.2	Recording pick-and-place positions	4
3.3	Moving the arm	4
3.4	Grippers	5

Introduction

In Lab 3, you investigated the *forward kinematics* problem, in which the joint angles of a manipulator are specified and the coordinate transformations between frames attached to different links of the manipulator are computed. Often, we’re interested in the *inverse* of this problem: Find the combination of joint angles that will position a link in the manipulator at a desired location in $SE(3)$.

It’s easy to see situations in which the solution to this problem would be useful. Consider a pick-and-place task in which we’d like to pick up an object. We know the position of the object in the stationary world frame, but we need the joint angles that will move the gripper at the end of the manipulator arm to this position. The *inverse kinematics* problem answers this question.

*Developed by Aaron Bestick and Austin Buchan, Fall 2014. Modified by Laura Hallock and David Fridovich-Keil, Fall 2017. Modified by Valmik Prabhu, Nandita Iyer, Ravi Pandya, and Phillip Wu, Fall 2018

1 Getting started with Git

Create a new workspace called `lab5` in your `ros_workspaces` directory. Refer to Lab 1 if you need a refresher on how to do this.

Our starter code for this lab is on GitHub for you to clone so that you can easily access any updates we make to the starter code. It can be found at https://github.com/ucb-ee106/lab5_starter.git. You can clone it by running

```
git clone https://github.com/ucb-ee106/lab5_starter.git
```

We also highly recommend you make a **private** GitHub repository for each of your labs just in case.

2 Inverse Kinematics

An inverse kinematics solver for a given manipulator takes the desired end effector configuration as input and returns a set of joint angles that will place the arm at this position. In this section, you'll learn how to use ROS's built-in inverse kinematics functionality.

2.1 Specify a robot with a URDF

While the `tf2` package — which you examined in Lab 3 — is the de facto standard for computing coordinate transforms for forward kinematics computations in ROS, we have several options to choose from for inverse kinematics. We'll use a package called MoveIt, which provides an inverse kinematics solver and other motion planning functionality.

Both MoveIt and `tf2` are generic software packages that can work with almost any robot. This means we need some method by which to specify a kinematic model of a given robot. The standard ROS file type for kinematic descriptions of robots is the Universal Robot Description Format (URDF), but the `baxter.urdf` file is actually deprecated and no longer exists in the `baxter_description` package of the Baxter SDK. We have instead provided the `baxter.urdf` file in the Lab 5 starter code. (Modern versions of ROS use the `xacro` package to describe the same information more cleanly. We will not examine these `xacro` files in this lab, but we encourage you to look through them if you plan to write your own URDF files for your final project.)

Task 1: It's hard to visualize the actual robot by staring at an XML file, so ROS provides a tool that creates a more informative kinematic diagram. Navigate to the folder containing your copy of the Baxter URDF and run

```
check_urdf baxter.urdf
urdf_to_graphiz baxter.urdf
```

Open the PDF file that this command creates. What do you think the blue and black nodes represent? In addition to Baxter's limbs, what are some of the other nodes included in the URDF, and how might this information be useful if you want to use Baxter's sensing capabilities?

2.2 Compute inverse kinematics solutions

To use MoveIt, you must first start the `move_group` node, which offers a service that computes IK solutions. The file `move_group.launch` in the starter code loads the URDF description of Baxter onto the ROS parameter server, then starts the `move_group` node.

Create a package called `ik` with dependencies on `rospy`, `moveit_msgs`, and `geometry_msgs`. Save the file `move_group.launch` in the new package's `/launch` subdirectory and save `baxter.urdf` in the package's `src` subdirectory. The `move_group.launch` launch file depends on your `baxter.urdf` file. Then run the launch file with `roslaunch` to start the MoveIt node. MoveIt is now ready to compute IK solutions.

If you run into an issue where ROS cannot find the `baxter_moveit_configs` package, run

```
source /scratch/shared/baxter_ws/devel/setup.bash --extend
```

in the terminal.

Connect the `base` frame from the `baxter.urdf` file to the existing `world` frame by running

```
roslaunch static_transform_publisher 0 0 0 0 0 0 world base 100
```

Task 2: Write a node that prompts the user to input (x, y, z) coordinates for a gripper configuration, then constructs a `GetPositionIK` request, submits the request to the `compute_ik` service offered by the `move_group` node, and prints the returned vector of joint angles in the terminal. You can use the file `service_query.py` in the starter code as a template for your node. A couple of tips:

1. The `GetPositionIK` service takes as input a message of type `PositionIKRequest`, which is complicated. The most important part is the `pose_stamped` field, which specifies the desired configuration of the gripper.
2. The orientation of the gripper is specified as a quaternion. Since we're not worried about rotation, you can set the four orientation parameters to any values such that their norm is equal to 1. For reference, a value of $(0.0, \pm 1.0, 0.0, 0.0)$ will have Baxter's grippers pointing straight down.

Test the IK solutions from MoveIt by plugging the returned joint angles into your forward kinematics code from Lab 3. Copy the Python files from the Lab 3 `forward_kinematics` package to the same directory as the node you just made, and uncomment the 2 lines indicated in the `service_query.py` file. This will import the function `baxter_forward_kinematics_from_angles`, run it on the angles that your IK solution returns, and print out the resulting homogenous transformation. Does the transformation match your originally specified gripper position and orientation? Why or why not?

Does the IK solver always give the same output when you specify the same end effector position? If not, why not? Any ideas why the solver sometimes fails to find a solution?

Checkpoint 1

Submit a checkoff request at tinyurl.com/106alab for a staff member to come and check off your work. At this point you should be able to:

- Explain what the different parts of the graphical representation of Baxter's URDF file represent
- Validate the output of the MoveIt IK service by solving IK for three different poses, plugging these returned joint angles back into your Lab 3 forward kinematics code, and verifying that you get approximately the original pose back
- Explain whether the solutions found by the MoveIt IK service are unique, and if not, why not

3 Open loop manipulation with Baxter/Sawyer

In this section, you'll use inverse kinematics to program a Baxter or Sawyer to perform a simple manipulation task.

First, add a `move_arm` package to your `lab5` workspace. (Include `baxter_tools` or `intera_interface` along with the standard `rospy` and `std_msgs` dependencies.) Run `catkin_make` to build your new package.

3.1 Setting up your environment for Rethink robots

If you are working with a Baxter, make a shortcut (symbolic link) in the root of your catkin workspace (the `lab5` directory) to the Baxter environment script `/scratch/shared/baxter_ws/baxter.sh` using the command

```
ln -s /scratch/shared/baxter_ws/baxter.sh [path-to-workspace]
```

If you are working with a Sawyer, make a shortcut (symbolic link) in the root of your catkin workspace (the `lab5` directory) to the Sawyer environment script `/scratch/shared/baxter_ws/intera.sh` using the command

```
ln -s /scratch/shared/baxter_ws/intera.sh [path-to-workspace]
```

Connect to Baxter/Sawyer by running the `baxter.sh` or `intera.sh` file as you did in Lab 3, then enable the robot by running

```
roslaunch baxter_tools enable_robot.py -e
```

or

```
roslaunch intera_interface enable_robot.py -e
```

depending on which type of robot you are using. (The Baxter/Sawyer may already be enabled, in which case this command will do nothing.)

3.2 Recording pick-and-place positions

Now, position the table in front of Baxter/Sawyer within the reachable workspace. Find an object that will fit in Baxter/Sawyer's gripper, and place it somewhere on the surface.

You can get the transformation from the base to the gripper by running

```
roslaunch tf_echo base [gripper]
```

where `[gripper]` is `right_gripper_tip` if working with Sawyer, or `left_gripper` or `right_gripper` if working with Baxter. If the Baxter only has one gripper installed on it, use that gripper. If the Sawyer doesn't have a gripper use `right_wrist` (and ignore future parts about grippers).

With the robot enabled, grasp the sides of Baxter/Sawyer's wrist, placing the arm in zero-g mode, where it can be moved easily by hand. (*Note:* If you have never used zero-g mode and are having trouble manipulating the robot, **ask an instructor for assistance**. You shouldn't have to use much force to move the robot around!)

Move the arm to a position where it could grasp the object on the table and record the translation component of the `tf` transform. (*Hint:* If you're using Baxter, you can try the command `roslaunch baxter_tools tuck_arms.py -u` to place Baxter in a good starting joint configuration before placing the gripper near the object.)

Next, move the arm to a different position on the table where you'd like to set the object down and record the transform of this location as well. We recommend trying to have Baxter/Sawyer's gripper pointing straight down in both of these configurations.

3.3 Moving the arm

After recording the positions to which you'd like to move the arm, you'll use the IK and path planning functionality of MoveIt to move Baxter/Sawyer's arm between different poses. Run Baxter's joint trajectory controller with the command

```
roslaunch baxter_interface joint_trajectory_action_server.py
```

or Sawyer's using

```
roslaunch intera_interface joint_trajectory_action_server.py
```

Next, in a new window, start MoveIt with

```
roslaunch baxter_moveit_config demo_baxter.launch right_electric_gripper:=true left_electric_gripper:=true
```

for Baxter, or

```
roslaunch sawyer_moveit_config sawyer_moveit.launch electric_gripper:=true
```

for Sawyer, omitting the last argument(s) if your robot does not have a gripper. On Baxter, the command above may fail depending on the types of attached grippers; if so, try

```
roslaunch baxter_moveit_config baxter_grippers.launch
```

instead. This command also opens an RViz window; ignore it for now.

Locate the `ik_example.py` file (from the starter code), place it in the `src` directory of your `move_arm` package, make it executable, then build and source your workspace. `ik_example.py` contains an example of using MoveIt to move Baxter's left arm to a specified pose. Open the file and examine it. Change all relevant statements to describe the arm you are controlling (e.g., if you are controlling Sawyer's right wrist, ensure that the relevant lines reference `right_arm` and `right_wrist`.)

Run the example with (Don't forget to ssh into the robot!):

```
roslaunch move_arm ik_example.py [robot]
```

where `[robot]` is either `sawyer` or `baxter` (all lowercase).

The file should prompt you to press enter; it will then move the specified gripper to the position (0.5, 0.5, 0.0) with orientation (0.0, 1.0, 0.0, 0.0) (gripper pointing straight down). Press enter a few times and pay attention to Baxter/Sawyer's movements, moving the arm in compliant zero-g mode between trials. Now, try uncommenting the line that only specifies a position target (with no orientation specified). Run this node again and observe the difference in behavior. Which approach is more predictable? You may want to move the table while experimenting with this behavior.

3.4 Grippers

It's easy to operate Baxter/Sawyer's grippers programmatically as part of your motion sequence. Copy the file `gripper_test.py` to the `move_arm/src/` directory. Try this by running

```
roslaunch move_arm gripper_test.py [robot]
```

where `[robot]` is either `sawyer` or `baxter` (all lowercase). This script calibrates and then opens and closes Baxter/Sawyer's right gripper.

Task 3: Make a copy of the `ik_example.py` file in the `move_arm/src/` directory. Modify your file so that it moves the arm through the series of poses that you recorded earlier using `tf_echo` and attempt to perform a pick and place. You should use code from `gripper_test.py` to open and close the grippers at an appropriate time. Can you make the arm return to the same position, repeatedly, with good enough accuracy to pick up an item?

After completing this task, you might have noticed that open loop manipulation is, in general, difficult. In particular, it's hard to estimate the position of the object accurately enough that the arm can position the gripper around it reliably. Do you have any ideas about how we might use data from the additional sensors on Baxter to perform manipulation tasks more reliably?

Checkpoint 2

Submit a checkoff request at tinyurl.com/106alab for a staff member to come and check off your work. At this point you should be able to:

- Use `tf_echo` to get the current end-effector pose for Baxter/Sawyer's arm
- Perform a pick and place task. This doesn't have to work perfectly (or at all), but at least make an attempt. If pick and place doesn't work, what could you do to improve it?