Robotic Systems Engineering Lab 06

This document contains examples and exercises on writing code for the Inverse Kinematics problem. The manipulator we are going to work on in the example is the 'Robotis Manipulator-H', while in the tasks you will have to work on the 'OpenManipulator' and the 'Youbot Manipulator'.

Example 01

In this example we compute and then code the Inverse Kinematics for the 'Robotis Manipulator-H'.

Let's suppose we have a known configuration T and a robot pose determined from the forward kinematics problem ${}^0T_N(\theta_1, \theta_2.\theta_3, d_4, \dots.)$. The latter is a function of the joint variables.

$$T = \begin{bmatrix} r_1 & r_2 & r_3 & p_x \\ r_4 & r_5 & r_6 & p_y \\ r_7 & r_8 & r_9 & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad and \quad {}^0T_N(\theta_1, \theta_2.\theta_3, d_4, \dots.) = \begin{bmatrix} r_{11} & r_{12} & r_{13} & x \\ r_{21} & r_{22} & r_{23} & y \\ r_{31} & r_{32} & r_{33} & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We need to find $(\theta_1, \theta_2, \theta_3, d_4, \dots)$ so that ${}^0T_N(\theta_1, \theta_2.\theta_3, d_4, \dots.) = \mathrm{T}$.

For this specific robot, the simplified and rearranged equations for x, y and z are:

$$x - d_6 r_{13} = d_4 c_1 c_{23} + a_2 c_1 s_2 + a_3 c_1 s_{23} \quad (1)$$

$$y - d_6 r_{23} = d_4 s_1 c_{23} + a_2 s_1 s_2 + a_3 s_1 s_{23} \quad (2)$$

$$z - d_1 - d_6 r_{33} = a_2 c_2 + a_3 c_{23} - d_4 s_{23} \quad (3)$$

Keep in mind that these equations are the simplified forms of the more complicated actual equations. If you are having trouble simplifying the equations by hand, you can use automated tools like the 'simplify' MATLAB function.

From (1) and (2) we can get:

$$\theta_1 = \operatorname{atan}\left(\frac{y - d_6 r_{23}}{x - d_6 r_{13}}\right) \quad \text{or} \quad \theta_1 = \operatorname{atan}\left(\frac{y - d_6 r_{23}}{x - d_6 r_{13}}\right) \pm \pi$$

The second value of θ_1 can be either $\operatorname{atan}\left(\frac{y - d_6 r_{23}}{x - d_6 r_{13}}\right) + \pi$ or $\operatorname{atan}\left(\frac{y - d_6 r_{23}}{x - d_6 r_{13}}\right) - \pi$. This solution needs to be checked with the joint limit of θ_1.

For simplicity, let $k = \frac{x - d_6 r_{13}}{\cos\theta_1}$ if $\sin\theta = 0$, else $k = \frac{y - d_6 r_{23}}{\sin\theta_1}$.

Hence:

$$k^2 + (z - d_1 - d_6 r_{33})^2 = a_2^2 + a_3^2 + d_4^2 + 2a_2(a_3 \cos\theta_3 - d_4 \sin\theta_3) \rightarrow$$

$$\frac{k^2 + (z - d_1 - d_6 r_{33})^2 - a_2^2 - a_3^2 - d_4^2}{2a_2\sqrt{a_3^2 + d_4^2}} = K = \cos(\theta_3 + \beta), \; where \; \beta = atan2(d_4, a_3)$$

And finally:

$$\theta_3 = \pm \operatorname{acos}(K) - \beta$$

Please note that in the example script, we subtract the offset from the solution, because we didn't take into account $arctan(0.03/0.264)$ in our inverse kinematic solutions. Therefore, we have to add it back when we're using that value to solve for θ_2.

To find θ_2 we need the value of θ_3. From (3) we have:

$$z - d_1 - d_6 r_{33} = a_2 c_2 + a_3 c_2 c_3 - a_3 s_2 s_3 - d_4 s_2 c_3 - d_4 c_2 s_3 = (a_2 + a_3 c_3 - d_4 s_3) c_2 - (a_3 s_3 + d_4 c_3) s_2 \rightarrow$$

$$\frac{z - d_1 - d_6 r_{33}}{\sqrt{A^2 + B^2}} = \cos(\theta_2 + \gamma), where\ \gamma = atan2(A, B)\ and\ A = a_3 s_3 + d_4 c_3, B = a_2 + a_3 c_3 - d_4 s_3$$

From (1) or (2):

$$k = a_2 s_2 + a_3 s_2 c_3 + a_3 s_3 c_2 + d_4 c_2 c_3 - d_4 s_2 s_3 = (a_2 + a_3 c_3 - d_4 s_3) s_2 + (a_3 s_3 + d_4 c_3) c_2 \rightarrow$$

$$\frac{k}{\sqrt{A^2 + B^2}} = \sin(\theta_2 + \gamma)$$

And finally:

$$\theta_2 = atan2(k, z - d_1 - d_6 r_{33}) - atan2(A, B)$$

The offset is subtracted out for the same reason as θ_3.

Now that we have the $\theta_1, \theta_2\ and\ \theta_3$ values, we can determine the last three joints by calculating:

$${}^3R_6(\theta_3, \theta_4, \theta_5) = {}^0R_3(\theta_1, \theta_2, \theta_3)^{-1} R,$$

For simplicity we denote: ${}^3R_6(\theta_3, \theta_4, \theta_5) = \begin{bmatrix} n_{11} & n_{12} & n_{13} \\ n_{21} & n_{22} & n_{23} \\ n_{31} & n_{32} & n_{33} \end{bmatrix}$.

And we finally have:

$$\theta_4 = atan2\left(-\frac{n_{23}}{s_5}, -\frac{n_{13}}{s_5}\right)$$

$$\theta_5 = \pm \operatorname{acos}(-n_{33})$$

$$\theta_6 = atan2\left(\frac{n_{32}}{s_5}, -\frac{n_{31}}{s_5}\right)$$

For the 'Robotis Manipulator-H'robot, we will end up with 8 possible solutions due to multiple solutions of $\theta_1, \theta_3, \theta_5$. After getting the solutions, we then have to compare the computed solutions with the joint limit whether they lie within the robot workspace. In some cases, the solution returns the joint value at the joint limits, but this still invalidates the solution because of numerical error in python/c++.

$2 \times 2^3 = 8$

Please note that this solution is merely an example. There are many ways to solve the problem which may include different mathematical techniques. There can even be solutions that already include every offset in the joint value. If they are solved correctly, however, they will return the same set of inverse kinematics solutions.

After we solve the Inverse Kinematics problem by hand, it is now time to code it.

In this lab, you will not be able to directly run any code because we are not building any executables, just an extensive class. If you want to see your code in action you can create a script (eg publisher/subscriber) that utilizes this class in order to compute any of the problems described in it, using code from the previous labs.

This is a technique often used in real life robotic systems because it is very convenient to group similar tasks in the same class. For example, you would want to group any 'Kinematics' related methods and functions together in the same class, any 'Computer Vision' algorithms in a separate class and so on. This approach is much more organised than just creating a different package for every different little task you want to do on your robot.

Notice that the structure of the 'lab06' package is slightly different than usual.

C++

The C++ part of the structure is pretty much the same. As usual, the '.cpp' file, where we define our class, is inside the 'src' folder. What we did change is the 'CMakeLists.txt' file. The 'catkin_package()' function is not empty anymore. It contains the library that the package will export. For example if a new package makes use of the 'lab06_example01' library that we just exported, all we need to do is include this library in the 'find_package()' function of the 'CMakeLists.txt' file of our new package.

Also, we added the 'add_library()' function in which we specify the exported libraries from this project (in this case 'hArmKine.cpp'). Pay attention that inside the 'add_library()' we must include the entire path under which our target library lies. Finally, as we already mentioned, we are not building any executables in this lab so there is no need for the 'add_executable()' function.

Python

In this case, the structure differs than the usual package structure. Up until now, we used to always put our python code inside the 'scripts' folder. But sometimes your python code is not the main script. For example, in this lab your python code is the library and your main script would be the script that utilizes this library. Another example is when you are defining a class or a module and you want this definition to be used in another package. In such cases the package structure must follow these rules:

1. The '[class_def].py' file must be in 'catkin_ws/src/[pkg_name]/src/[pkg_name]/[class_def].py'
2. We must create a blank python file called '__init__.py' in the same folder
3. We have to uncomment the 'catkin_python_setup()' command in 'CMakeLists.txt'
4. We must create a python file called 'setup.py' in the same directory as 'CMakeLists.txt'.

Open 'setup.py' of the 'lab06_example01' folder to take a look at the structure of the setup file and follow the same structure in your tasks.

We won't go through the code extensively since it really is just the implementation of the mathematical model we showcased at the start of this instructions sheet. However, there are some points that are worth mentioning.

- Both the '.cpp' and the '.py' classes have a method called 'init'. This method is known as a 'constructor' in Object Oriented Programming languages. In C++, it is called every time an object from the class is created and initializes the attributes of the class. In Python, it is not necessary. For the purposes of this exercise, the attributes of the class are the D-H parameters, the limits of the joints, a publisher and a subscriber.

- We have created class member functions for the ‘D-H Matrix’, the ‘Forward Kinematics’, the ‘broadcaster’ and the closed form ‘Inverse Kinematics’ for the ‘Robotis Manipulator-H’.
- We have created, but have not filled in, class member functions for the ‘Jacobian’, the ‘Singularity checker’ and the iterative form of the ‘Inverse Kinematics’. This is due to the fact that these code bits are part of the coursework.

Task 01

In this task you will be working on the ‘OpenManipulator’ robot. Using the templates and the example we have provided, fill in the methods ‘get_jacobian’, ‘inverse_kine_closed_form’, ‘inverse_kine_ite’ and ‘check_singularity’ of the class ‘open_kinematic’, grouping all kinematic functions for the ‘OpenManipulator’.

Task 02

In this task you will be working on the ‘Youbot Manipulator’ robot. Similarly to ‘Task 01’, fill in the methods ‘get_jacobian’, ‘inverse_kine_closed_form’, ‘inverse_kine_ite’ and ‘check_singularity’ of the class ‘youbot_kinematic’, grouping all kinematic functions for the ‘Youbot Manipulator’.