Lab: Computing FK and IK for a Robotic Arm

Lab Objective: Write a program in C++ implementing the forward kinematic (FK) and inverse kinematic (IK) algorithms for a robotic arm.

The document is quite long as it contains lots of basis material, please read it carefully. The overall required work is not too complex if you follow the instructions.

1 - Background

1.1 3D Object in Space

A 3D object is defined by a list of points (vertices) and edges connecting theses vertices forming triangles.
When loaded in a 3D software tool, the objects are shown on the default position, without rotation or translation. It is well illustrated below, with a simple airplane object loaded in Blender:

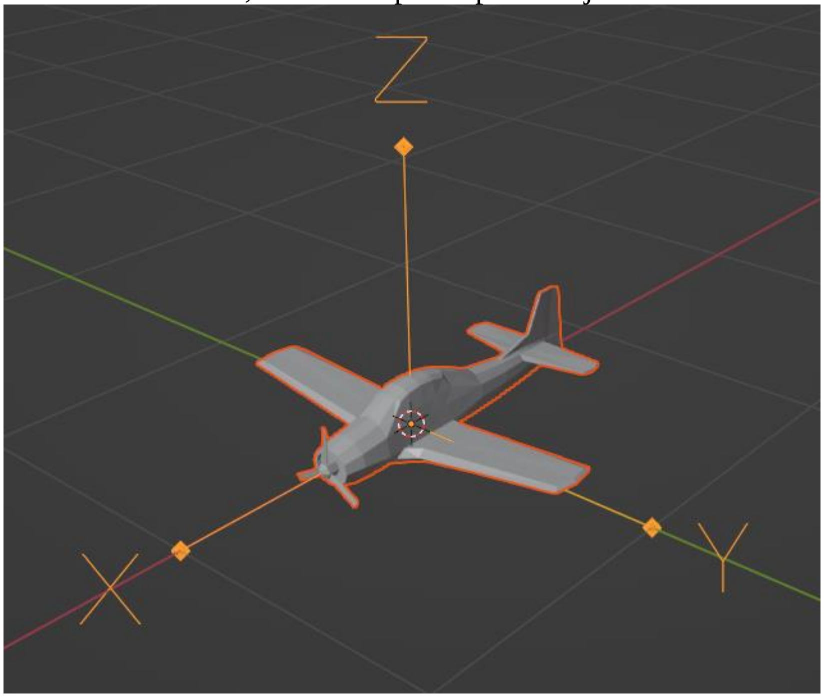


Figure 1. Default Object Position.

The object can be moved and rotated in the 3D space, so 6 values are needed to fully capture the position of the object vs the default position: 3 rotation values and 3 translation values.

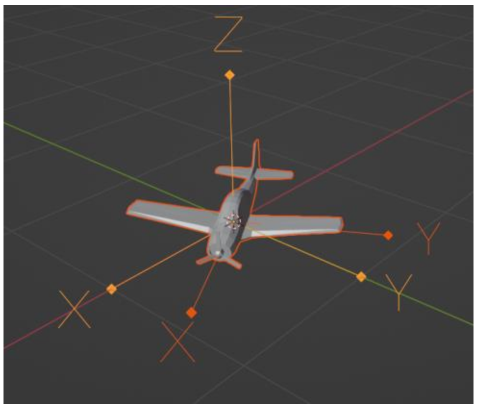
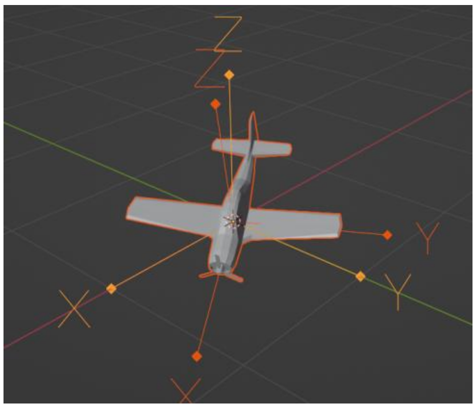
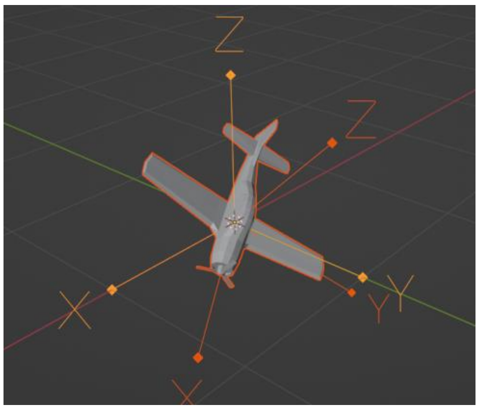
1.1.1 Rotation RPY

Rotations are defined on the *local frame* by 3 angles, done in the following order

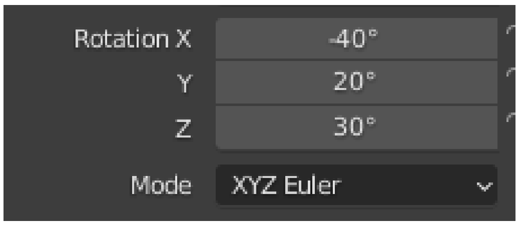
- a yaw angle (α), rotation on the **z** axis, **x** and **y** becomes **x1** and **y1**
- a pitch angle (β), rotation on the **y1** axis, **x1** and **z** becomes **x2** and **z1**
- a roll angle (γ), rotation on the **x2** axis, **y1** and **z1** become **y2** and **z2**

Local frame means that the rotation axes are linked to the object as opposed to the global frame where the rotation axes are fixed.

Example: successive yaw, pitch and roll rotations using the local frame to express the rotation.

Yaw with $\alpha = 30°$	Pitch with $\beta = 20°$	Roll with $\gamma = -40°$
		
Yaw Rotation Matrix $R_z(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$	Pitch Rotation Matrix $R_y(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix}$	Roll Rotation Matrix $R_x(\gamma) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{pmatrix}$

The overall rotation matrix can be expressed by multiplying these 3 matrices. As matrix multiplication is not commutative, we have shall only use that specific order in this document.
For reference, in Blender, rotation order can be selected, and the Blender order matching the one we have selected is called `XYZ Euler` as shown below. This specific order is also called `RPY`.



Rotation X -40°
Y 20°
Z 30°
Mode XYZ Euler

Figure 2. Blender Rotation Mode Selection.

Using the XYZ or RPY order, a vertex $V = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$ belonging to the original object, is transformed in vertex $V' = \begin{pmatrix} v'_x \\ v'_y \\ v'_z \end{pmatrix}$ using the following matrix transformation:

$$V' = R_z(\alpha) \cdot R_y(\beta) \cdot R_x(\gamma) \cdot V$$

The order of the matrix multiplications could be counter intuitive, but a rotation matrix is defined on the global reference axes, not on the local rotated axes.

1.1.2 Translation

Translation is defined by the 3 parameters t_x, t_y, t_z

Continuing with example, we show the images before and after translation

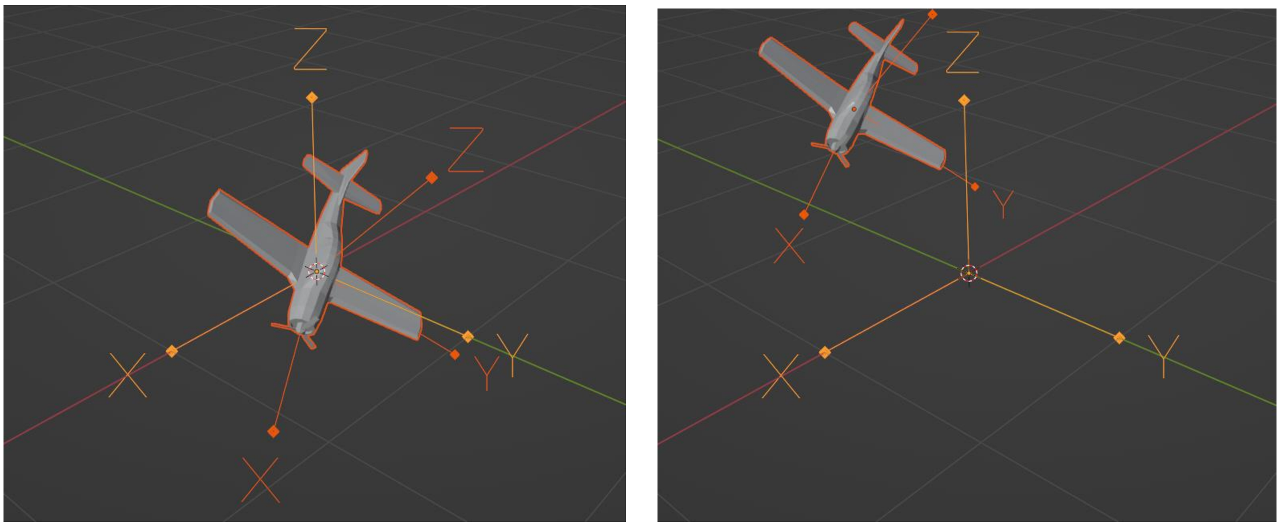
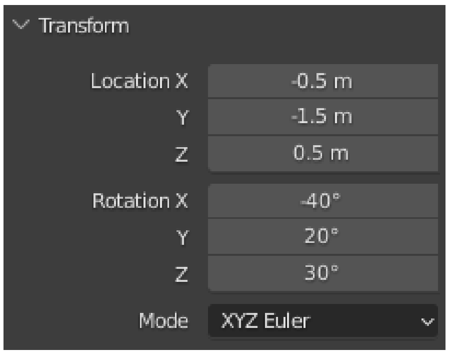


Figure 3. Object Translation Before After.

The 6 parameters are visible in Blender as:


Transform

Location X	-0.5 m
Y	-1.5 m
Z	0.5 m
Rotation X	-40°
Y	20°
Z	30°
Mode	XYZ Euler

Figure 4. Blender Translation and Rotation Parameters.

Translation cannot be expressed with a 3x3 matrix transform; we must use 4x4 matrices instead. The convention is that a 3D point, such as vertex V is represented by a 4-dimension vector:

$$V = \begin{pmatrix} v_x \\ v_y \\ v_z \\ 1 \end{pmatrix}$$

Rotation matrix are extended as shown in the example below:

$$R_x(\gamma) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\gamma) & -\sin(\gamma) & 0 \\ 0 & \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The translation matrix is defined by the following 4x4 matrix:

$$T = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The final transformation is defined:

$$V' = T \cdot R_z(\alpha) \cdot R_y(\beta) \cdot R_x(\gamma) \cdot V$$

$$V' = M \cdot V$$

2 - Robotic Arm

In this lab, we shall use a model of the Gen3 robot from Kinova robotics, which has a comprehensive documentation and 3D models available for free.

- User Guide document: here
- Specification document: here
- Git repo here

Figure 5. Kinova Gen3 Robotic Arm.

The diagram below is the full picture of the robotic arm with the names of the all the links and the name of all the joints. We use the standard term "*link*" to refers to a rigid body that forms part of the robot arm's structure.

To limit complexity, (i) the base shall always be at the origin of the reference axes, (ii) the fingers shall not move and (iii) the "*effector*" link shall be used as the reference in the remaining of the document when considering the position of the robotic arm.

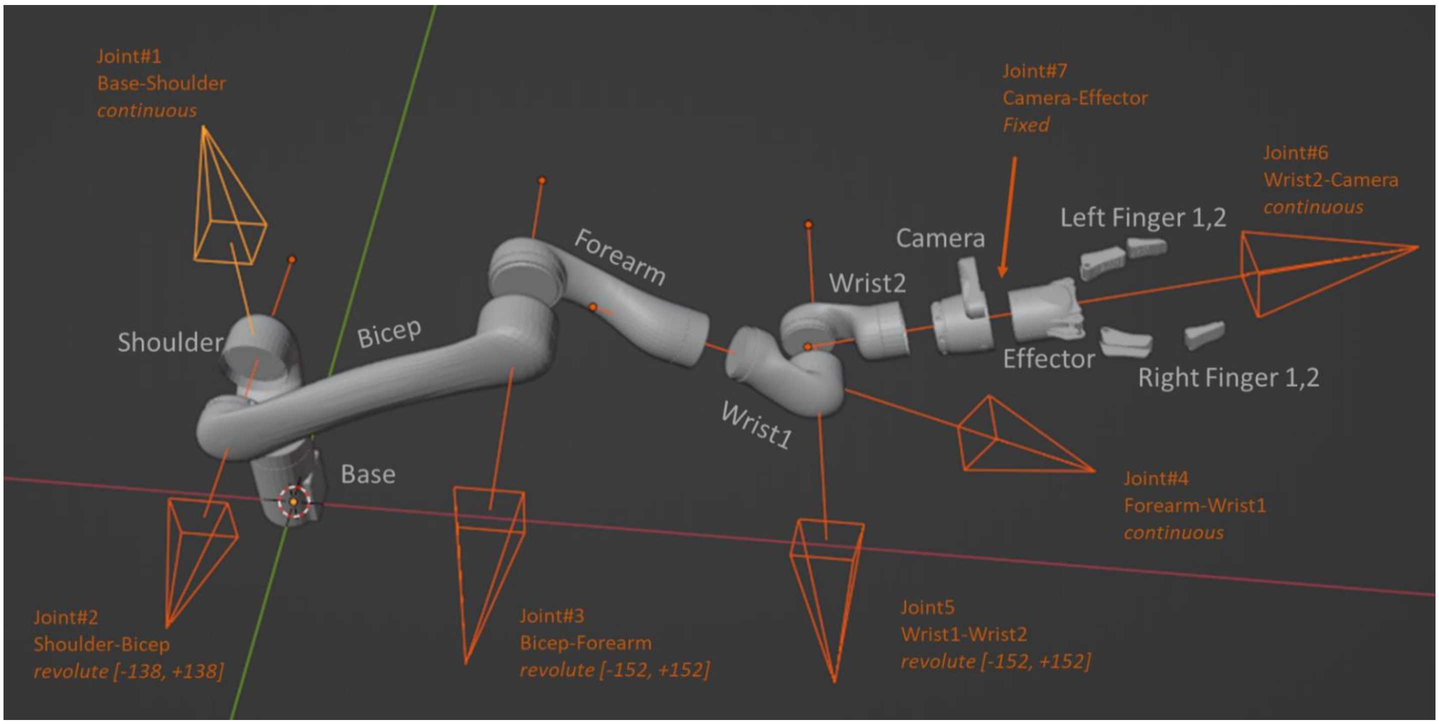


Figure 6. Robotic Arm with Links and Joints.

2.1.1 Composing Robot Links

Below is the first 4 links in default position:

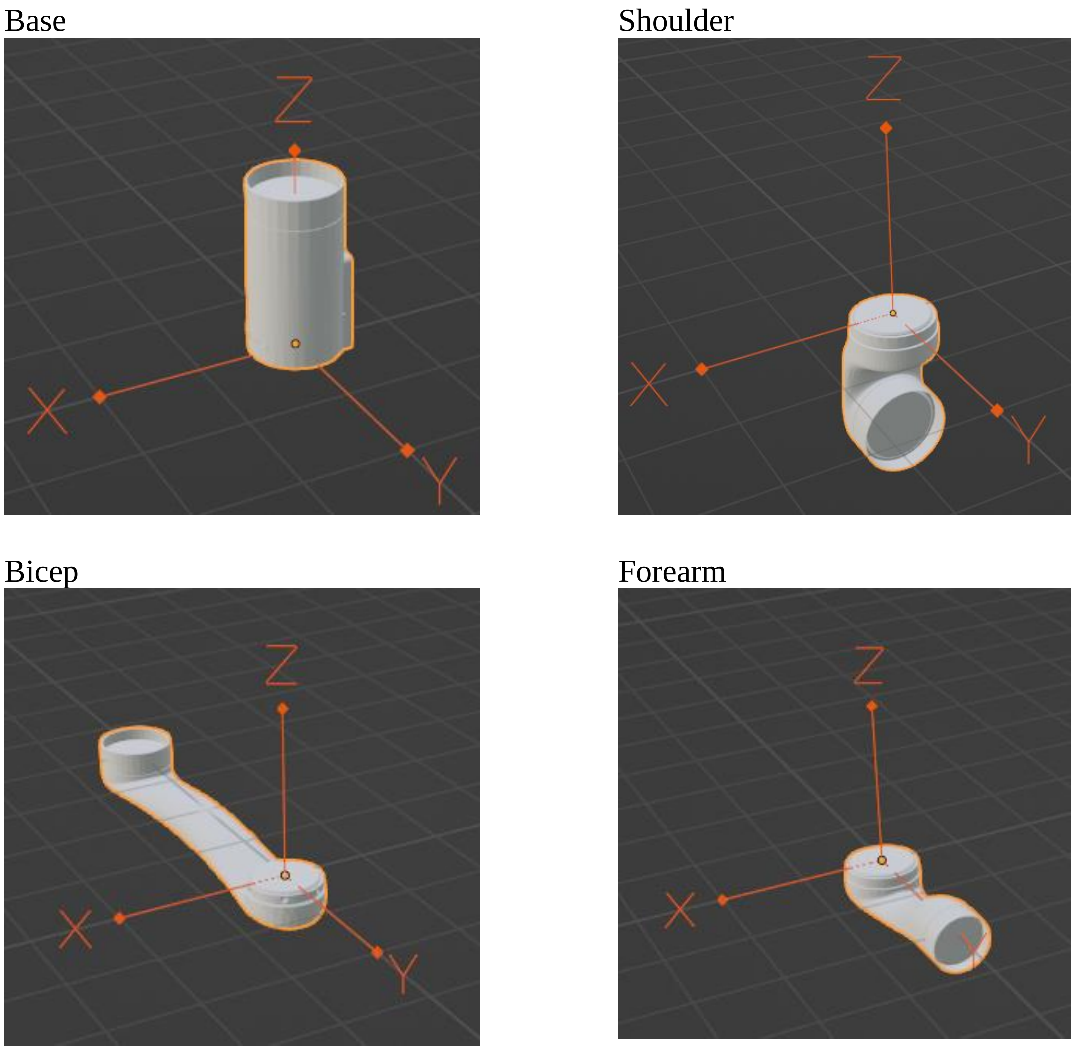


Figure 7. Links in Default Position.

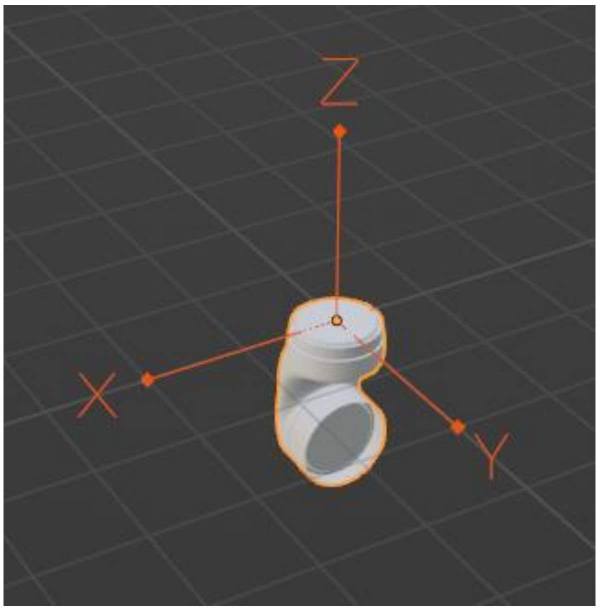
To assemble the robot, the shoulder must be first placed onto the base.
The URDF file indicates the offset and rotation which must be applied to the shoulder so that it fits exactly on top of the base:

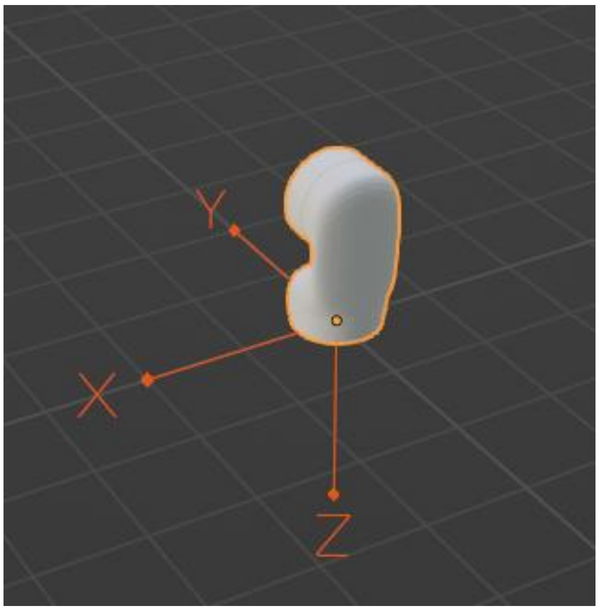
```
  <joint name="Actuator1" type="continuous">
    <origin xyz="0 0 0.15643" rpy="-3.1416 0.0 0.0" />
    <parent link="Base_Link" />
    <child link="Shoulder_Link" />
    <axis xyz="0 0 1" />
    <limit effort="39" velocity="0.8727" />
  </joint>
```

The statement `<origin...>` describes the rotation and translation which must be applied to the shoulder.

More details on URDF file format can be found here.

Shoulder before and after the rotation given by: `rpy="1.5708 0.0 0.0"`, this rotation is a 180° roll. Note how the local axis have changed as they are attached to the shoulder





After translation given by `xyz="0 0 0.15643"`

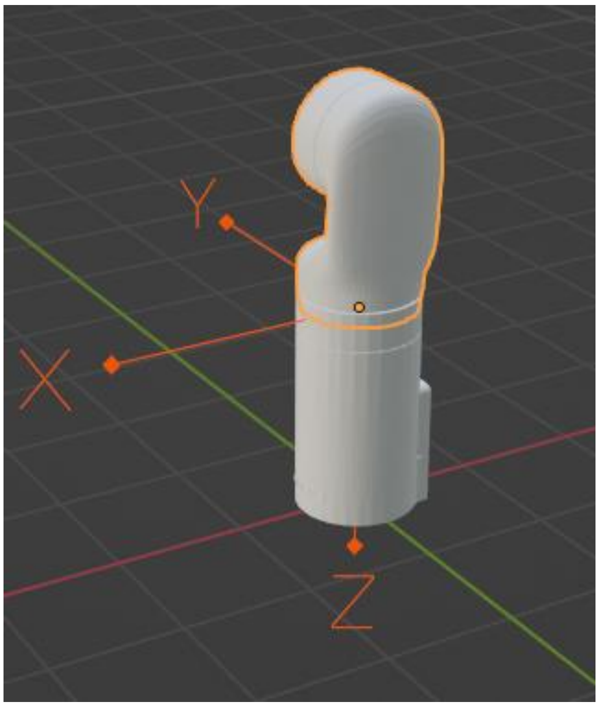


Figure 8. Composition of Links.

Using the 4x4 matrices described above and the UDRF description for joint 1, we can easily compute 2 matrices: R_1, and T_1. The overall transformation for the shoulder link is then given by:

$$Q_1 = Q_0 \cdot T_1 \cdot R_1$$

Q_0 is the transformation matrix of the base. We will assume that the base is located, as shown above, on the origin of the global space, so Q_0 is the identity matrix.

To add the bicep, we refer to the following section of the URDF:

```
  <joint name="Actuator2" type="revolute">
    <origin xyz="0 0.005375 -0.12838" rpy="1.5708 0.0 0.0" />
    <parent link="Shoulder_Link" />
    <child link="Bicep_Link" />
    <axis xyz="0 0 1" />
    <limit lower="-2.41" upper="2.41" effort="39" velocity="0.8727" />
  </joint>
```

From the above URDF, we can build two 4x4 transform matrices: R_2 and T_2

All the transforms in the URDF are related to the local reference frame of the parent.
So we have to proceed with 2 steps:

1) Apply R_2 then T_2 transforms to the bicep object.
2) Apply rotation and translation given by R_1 and T_1,

Default Position | After rotation by R_2 | After $T_2 \cdot R_2$ | After Q_1

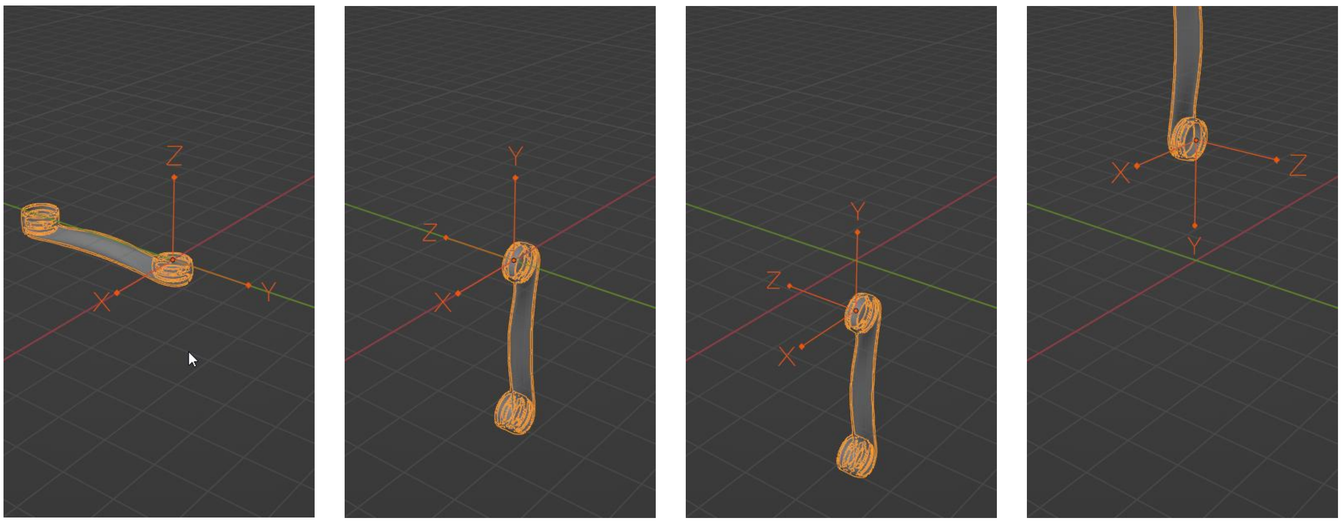


Figure 9. Step by Step Composition of Links.

Considering the global origin, it can be shown, that the bicep transform is given by:

$$Q_2 = Q_1 \cdot T_2 \cdot R_2$$

The process can be continued with the forearm:

```
  <joint name="Actuator3" type="revolute">
    <origin xyz="0 -0.41 0" rpy="3.1416 0 0" />
    <parent link="Bicep_Link" />
    <child link="ForeArm_Link" />
    <axis xyz="0 0 1" />
    <limit lower="-2.66" upper="2.66" effort="39" velocity="0.8727" />
  </joint>
```

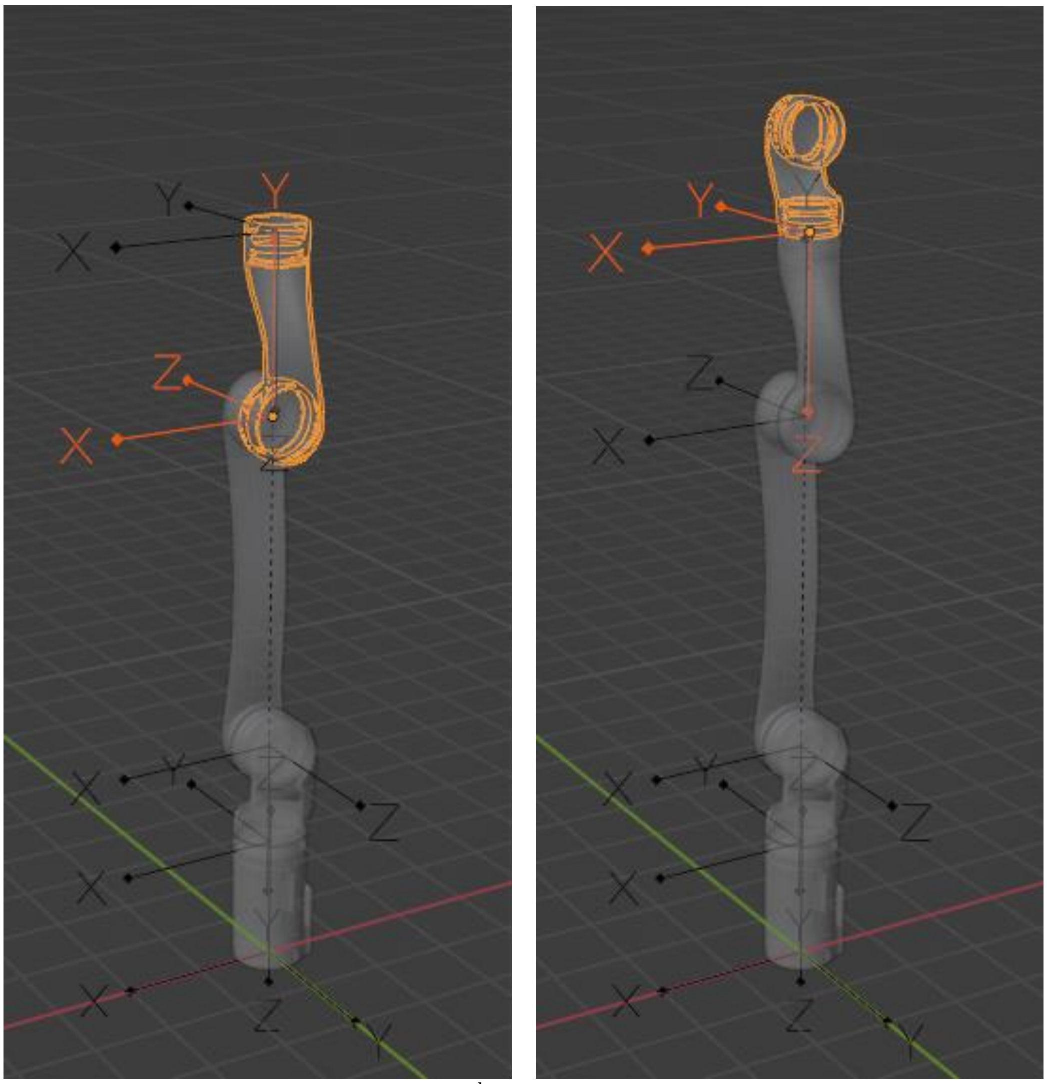


Figure 10. 4th Link Composition.

Given the URDF extract shown above for the 3rd joint, we build the rotation and transform matrices: R_3,and T_3. The global transform matrix for the forearm is given by:

$$Q_3 = Q_2 \cdot T_3 \cdot R_3$$

The transform matrix for the i^{th} link is computed equally, using the recursive rule:

$$Q_i = Q_{i-1} \cdot T_i \cdot R_i$$

2.1.2 Rotation of the Robot’s Joints

The axis of rotation is given by the URDF of the joint:

```
<joint name="Actuator2" type="revolute">
  <origin xyz="0 0.005375 -0.12838" rpy="1.5708 0.0 0.0" />
  <parent link="Shoulder_Link" />
  <child link="Bicep_Link" />
  <axis xyz="0 0 1" />
  <limit lower="-2.41" upper="2.41" effort="39" velocity="0.8727" />
</joint>
```

Rotation of the bicep shown in its default position (before and after) along the specified axis: it’s yaw of -30°.

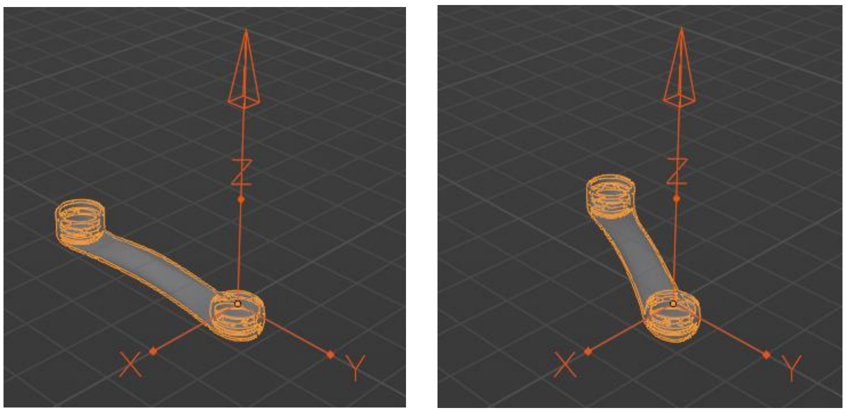


Figure 11. Link Rotation.

The rotation matrix of the i^{th} joint by an angle θ_i is expressed by $A_i(\theta_i)$.

It can be shown that transformation of the i^{th} link, is given by the following recursive definition:

$$Q_i(\theta_1, \theta_2, \dots, \theta_i) = Q_{i-1}(\theta_1, \dots, \theta_{i-1}) \cdot T_i \cdot R_i \cdot A_i(\theta_i)$$

With:

$$Q_1(\theta_1) = T_1 \cdot R_1 \cdot A_1(\theta_1)$$

Main Transform Formula

Overall robot before and after a rotation of $\theta_2 = -30\,^{\circ}$

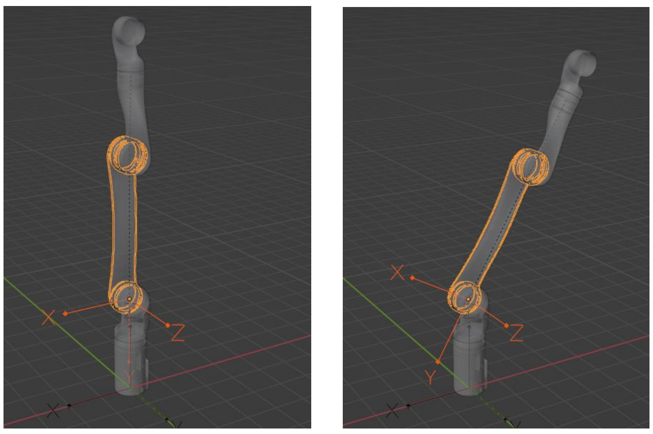


Figure 12. Link Composition with Joint Angle.

3 - Forward Kinematics (FK)

3.1 Problem Statement

Given a list of 6 joint angles ($\theta_1, \theta_2, \dots\ \theta_6$) find the 3D position of all the links composing the robot. This is a simple problem which can be solved by applying iteratively the main transformation formula of the previous section.

3.2 What must be done.

1) Install the package, compile, and cross-check that your installation is correct, see the appendix on the how to.

2) In the method `Robot::forward_kinematic(…)` add your version of the forward kinematic. The GLM package shall be used for all vector and matrix operations, check the appendix section to learn more on GLM.

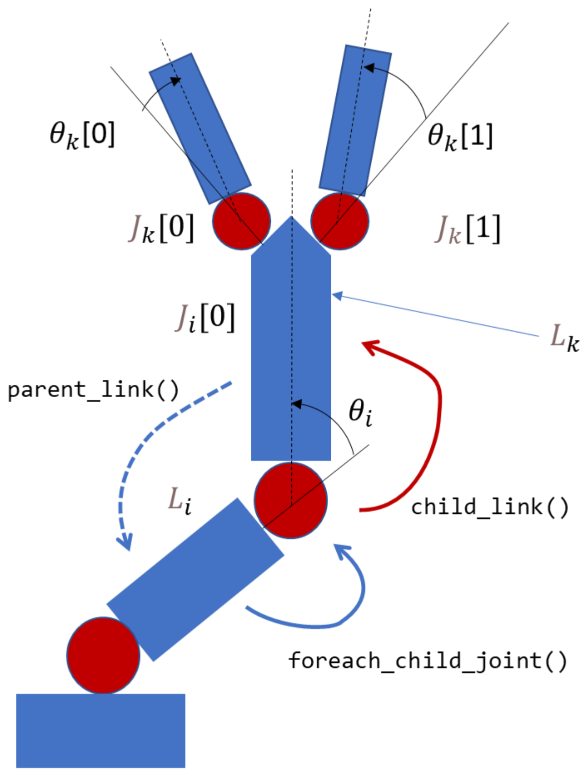
3) Validate your code against my output shown below. See next section for some hints on how to use existing methods to achieve the expected result.

```
shell> ./robot_kinematic --config ./robot.toml --angles -70,-50,90,0,-50,-90
################################################################################
##             Forward and Inverse Kinematic (Release version 1.0)            ##
##      Compiled on Sat Aug 31 18:31:52 2024 from source ID cb91eea48317+3    ##
################################################################################
[INFO]  Parsing config file ./robot.toml
multiple lines removed
[INFO]  ++++++++++++++++++++++++++++++++++++++++
[INFO]  Information for robot 'Kinova Gen3 6DF with vision'
[INFO]  Number of links     12
[INFO]  Number of joints    11
[INFO]  Degrees of freedom  6
[INFO]  Root link           L00.Base
[INFO]  Target link         L07.Effector
[INFO]  ++++++++++++++++++++++++++++++++++++++++
[INFO]  Compute forward kinematic with angles -70,-50,90,0,-50,-90
[INFO]  Target position (x, y, z) = (-0.23507285, -0.64191204, 0.30754)
[INFO]  Target rotation (r, p, y) = (0, 90, -109.99959)
Target Transform = {
  {          0,      0.9397,    -0.34201,    -0.23507}
  {          0,    -0.34201,     -0.9397,    -0.64191}
  {         -1,           0,           0,     0.30754}
  {          0,           0,           0,           1}
}
[INFO]  --------------------------------
[INFO]  Generate blender script to file my_script.py
[INFO]  Script file generated with 2 poses
```

3.3 Robot Link and Joint Methods.

The sample code already builds a robot object from a config file. TOML format has been chosen to describe the robot links and joints instead of the XML format as TOML is much simpler than XML for human to read and for a computer program to parse.

The diagram below illustrates the key method of the `Joint` and `Link` classes which form a `Robot` class. Less than 50 lines of codes are needed in the method `Robot::forward_kinematic()` to have this work.

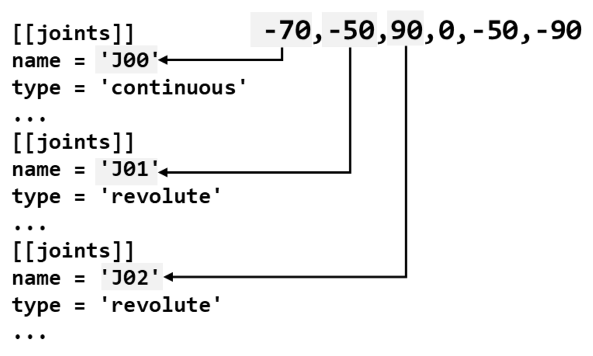


$$Q_k = \overbrace{Q_i \cdot \underbrace{T_i \cdot R_i \cdot A_i(\theta_i)}_{\text{Local Transform}}}^{\text{Global Transform for link } L_k}$$

```
class Joint {
 private:
  Link* p_child_link_;
  Link* p_parent_link_;
 public:
  Link *child_link() const;
  void set_angle(float);
  glm::mat4 get_local_transform() const;
  ...
};
```

```
class Link {
 private:
  std::vector<Joint*> pjoints_;
  Link* p_parent_link_;
 public:
  void foreach_child_joint(std::function<void(Joint*)>
callback);
  glm::mat4 get_transform() const;
  void set_transform(const glm::mat4&);
  Link *parent_link() const;
};
```

The list of angles must be applied to the non-fixed joints, based on the order found in the file. Traversing the robot links and joints from the root using the methods listed above guarantee this order.



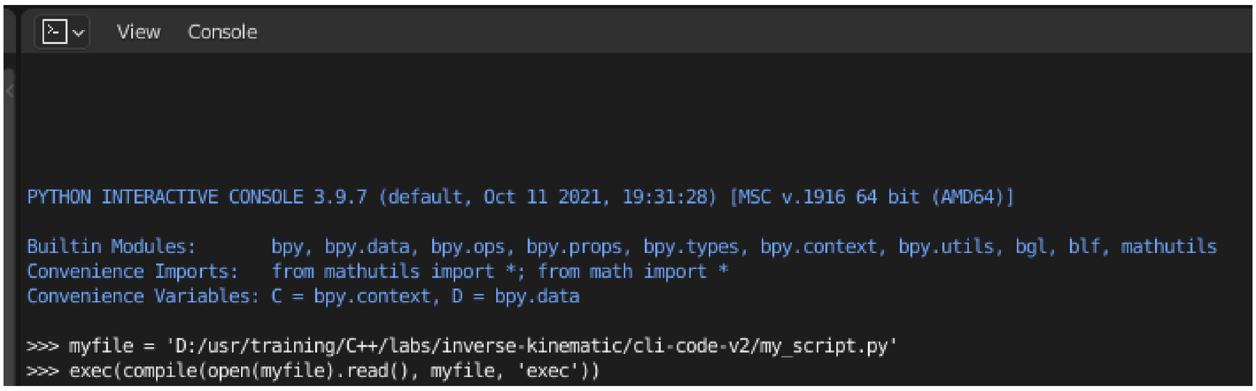
```
-70,-50,90,0,-50,-90
```

```
[[joints]]
name = 'J00'
type = 'continuous'
...
[[joints]]
name = 'J01'
type = 'revolute'
...
[[joints]]
name = 'J02'
type = 'revolute'
...
```

Figure 13. Relationship between Angles and Joints.

3.4 Visualization in Blender

You may have noted that the code generates a file called `my_script.py` which can be run in Blender python console after loading the blender project. The instructions to run the generated script are given in the script it self.
The trace is shown below:



```
PYTHON INTERACTIVE CONSOLE 3.9.7 (default, Oct 11 2021, 19:31:28) [MSC v.1916 64 bit (AMD64)]

Builtin Modules:       bpy, bpy.data, bpy.ops, bpy.props, bpy.types, bpy.context, bpy.utils, bgl, blf, mathutils
Convenience Imports:   from mathutils import *; from math import *
Convenience Variables: C = bpy.context, D = bpy.data

>>> myfile = 'D:/usr/training/C++/labs/inverse-kinematic/cli-code-v2/my_script.py'
>>> exec(compile(open(myfile).read(), myfile, 'exec'))
```

Figure 14. Blender Python Console.

After execution of the script, you will be able to see the robot in a position matching the angles that you have provided.

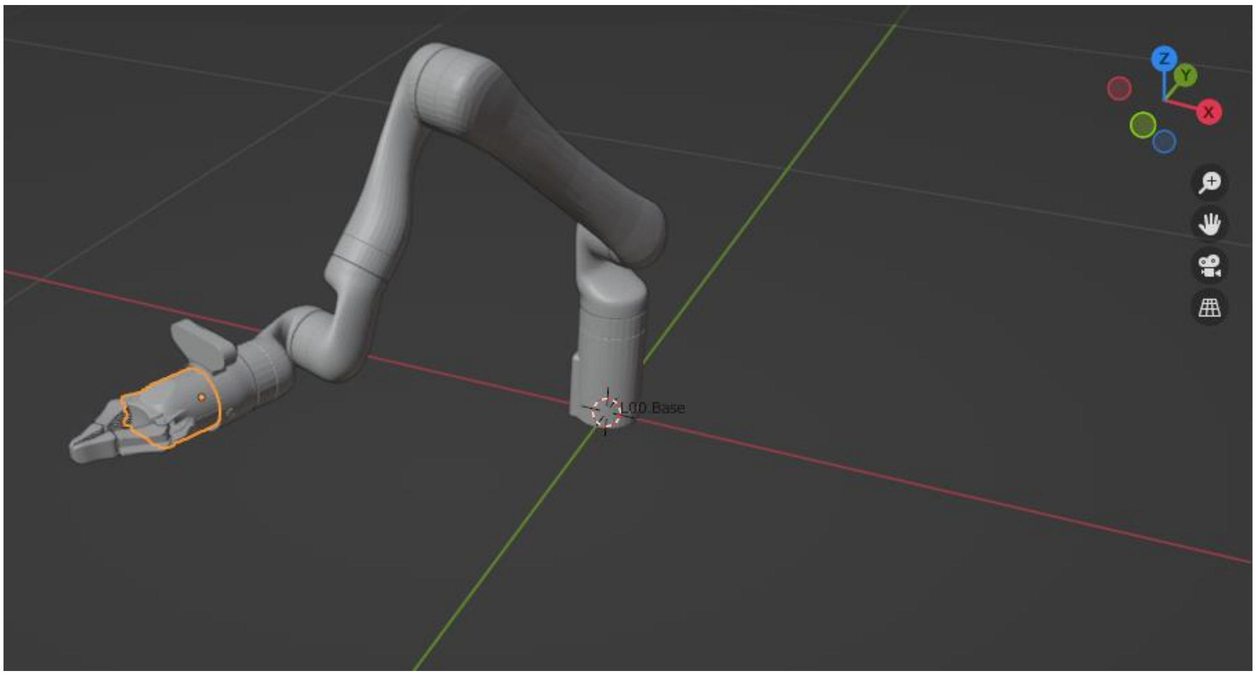


Figure 15. Robot Position for the Given Angles -70,-50,90,0,-50,-90.

4 - Inverse Kinematics (IK)

4.1 Problem Statement

Given a position T (target position) of the end effector in 3D space, i.e. the matrix Q_{eff} , find the angle values of the 6 joints $(\theta_1, \theta_2, \dots\ \theta_6)$, such that the computed position matches the target position.

IK is well-known problem which has extensively studied, many algorithms have been proposed. In this lab, we will be using the Cyclic Coordinate Descent (CCD) algorithm [more information].

CCD is an iterative algorithm, so we define T_i, the pose of the robot after the ith iteration, the pose is determined by the angles $(\theta_{1,i}, \theta_{2,i},\ \dots\ \theta_{6,i})$, we also define $\Delta_i = \|T_i, T\|$, a distance metrics between the current pose and the target pose.

,
In the simplest variant, CCD iterates as many times as needed, by selected randomly one of the 6 angles, say the kth and searching for new value of the angle, $\theta_{k,i+1}$, such as $\Delta_{i+1} < \Delta_i$, i.e. the current pose is closer to the target pose. It is possible that a change on the kth angle does not improve the solution.

A nice interactive demo can be found https://rodolphe-vaillant.fr/entry/114/cyclic-coordonate-descent-inverse-kynematic-ccd-ik

The user can move the target (T) represented by the blue cube, and the IK algorithm immediately computes the angles for the different links.

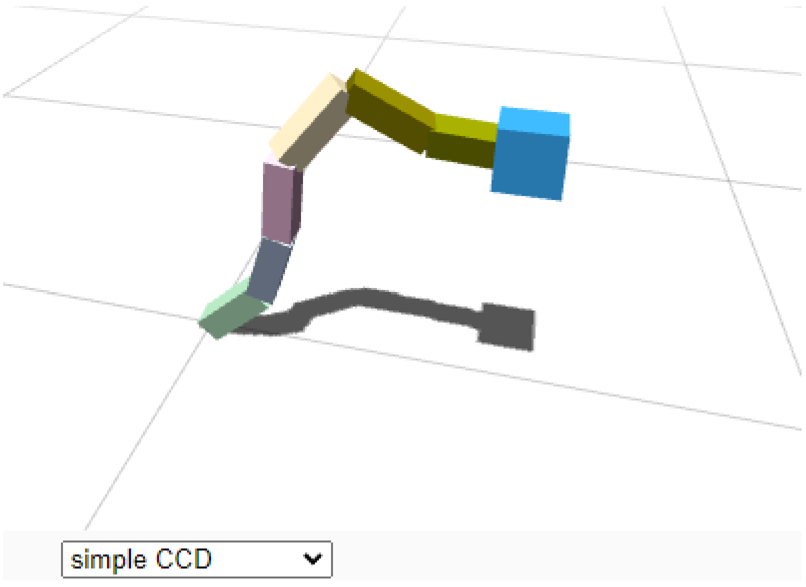


Figure 16. Inverse Kinematic of a Simple Robot.

4.2 Execution Trace

To help you debug your implementation, you will find an execution trace below.
The first part of the trace is identical to the trace shown in the forward kinematic section.

```
shell> ./robot_kinematic --config ./robot.toml --angles -70,-50,90,0,-50,-90 --xyz 0.302,-0.594,0.486 --rpy 0,90,-90
################################################################################
##           Forward and Inverse Kinematic (Release version 1.0)            ##
##     Compiled on Sat Aug 31 18:31:52 2024 from source ID cb91eea48317+3     ##
################################################################################
[INFO]  Parsing config file ./robot.toml
multiple lines removed same as forward kinematic
[INFO]  --------------------------------
[INFO]  --   Start Inverse Kinematic   --
[INFO]  --------------------------------
[INFO]  Using seed 1294732963
[INFO]  Target position is (0.302,-0.594,0.486)
[INFO]  Target rotation is (0,90,-90)
[INFO]  CCD iteration   10, cost is 0.32202473
[INFO]  CCD iteration   20, cost is 0.23252723
[INFO]  CCD iteration   30, cost is 0.17262703
[INFO]  CCD iteration   40, cost is 0.13303311
[INFO]  CCD iteration   50, cost is 0.100779735
[INFO]  CCD iteration   60, cost is 0.078131855
[INFO]  CCD iteration   70, cost is 0.06396078
[INFO]  CCD iteration   80, cost is 0.051985282
[INFO]  Final distance is 0.049736887
[INFO]  Joint angles are (-123.5,-33.5828,82.6932,-50.6,-37.5696,-46.2)
[INFO]  --------------------------------
[INFO]  --    Write Blender Script     --
[INFO]  --------------------------------
[INFO]  Generate blender script to file my_script.py
[INFO]  Script file generated with 3 poses
```

Important points:

- CCD algorithm relies on random numbers to select the next joint, every time the program is run, the results could be different. If you need to repeat the exact same results, you can use the seed option, as shown below. The trace prints the seed value, which can then be copied and pasted as the integer value
  `--seed integer       Force the seed for the random number generator`
- The number of iterations is also key for the CCD algorithm, you can control this number with the iteration option:
  `--iterations integer     Number of iterations for the CCD IK computation`
- If you need to try several variants, use the method option which can be used to enable/disable part of your code (if needed).
  `--method string      Extra option to enable variants in CCD IK computation`

4.3 Distance Metrics

It is critical that you define a correct distance metric.
If you use a simple distance between the current pose and the target pose, you will obtain a solution shown on the left-side screen capture: it’s close but the robot effector is not aligned with the target effector position shown in orange. With a proper distance metrics, used on the right-side screen capture, the computed effector position and the target position are almost identical.

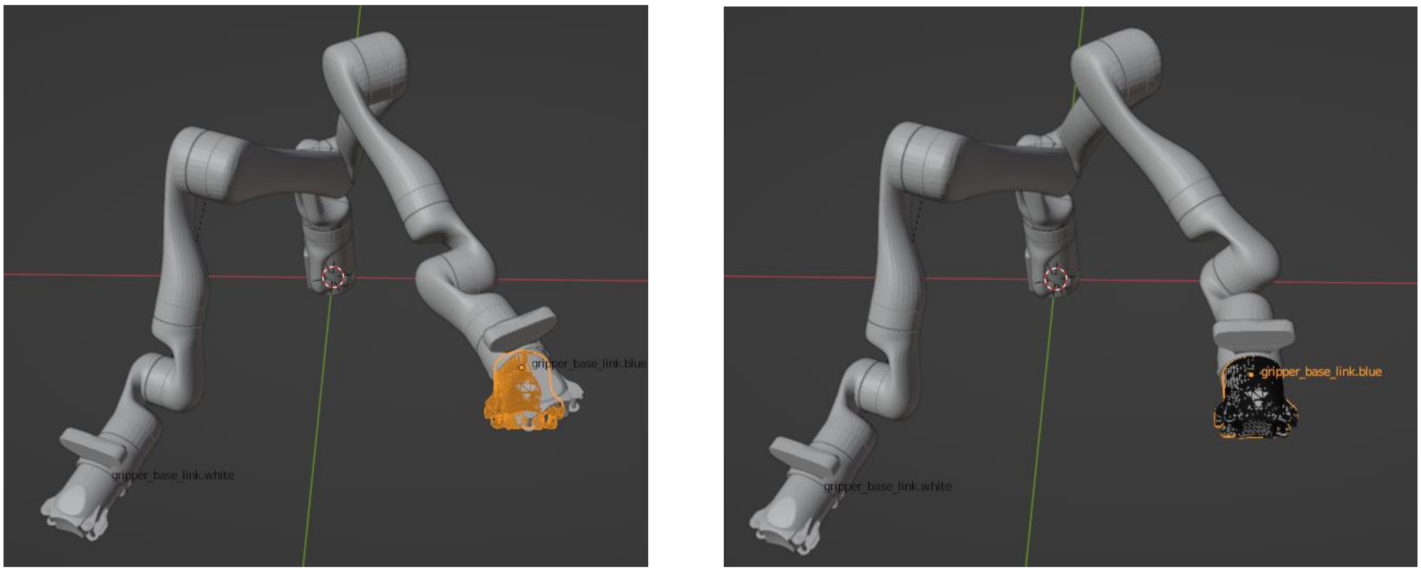


Figure 17. Incorrect and Correct Solution of the IK Problem.

Hint: don’t re-invent the wheel and have a close look at the existing code, an interesting class is already there.

4.4 Collision Detection (Optional)

Make sure that the solution is realistic. In the picture below, part of the bicep and the forearm are below the table level, this is not an acceptable solution.
You will propose a way to avoid such scenario.

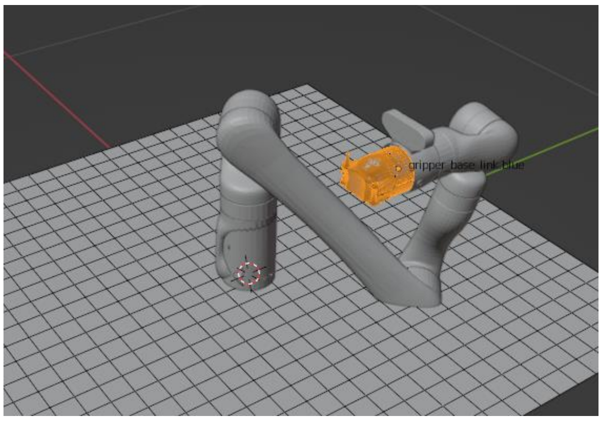


Figure 18. Incorrect Solution of the IK Problem.

4.5 Local Minimum (Optional)

The CCD algorithm belongs to the class of “greedy algorithm” and therefore can be stuck in a local minima. On the left side, it is impossible for CCD to converge toward the solution. A second try with a different random sequence, shown on the right picture, converges toward the solution.

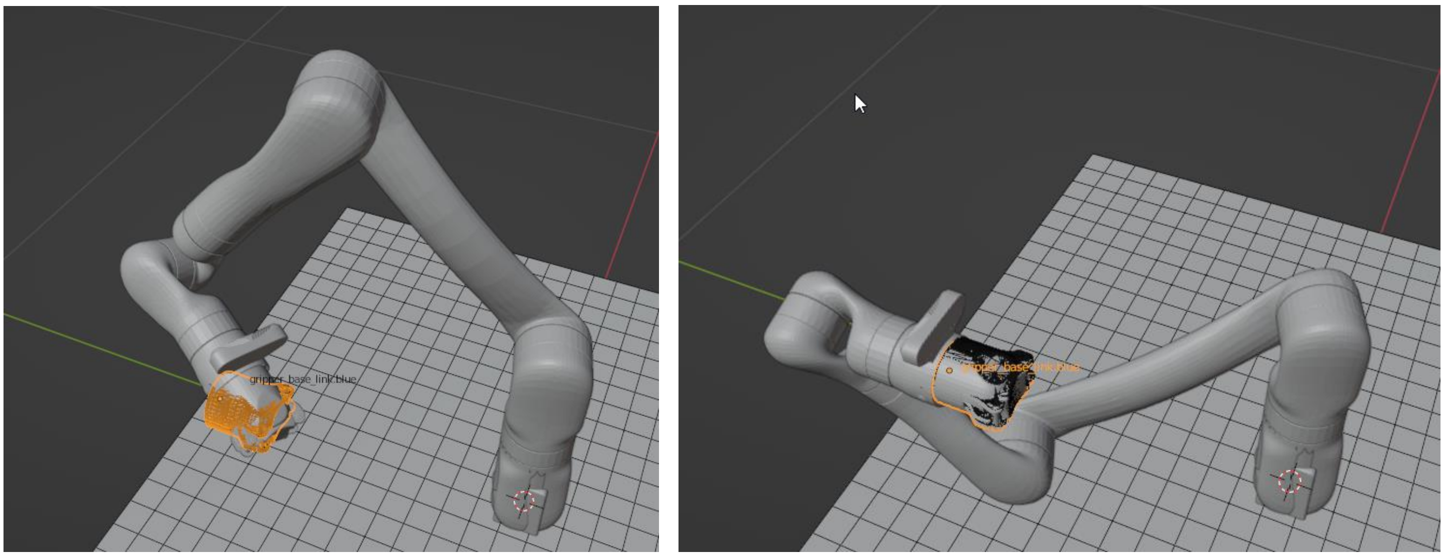


Figure 19. IK Problem, impact of the Local Minimum

Propose a solution to avoid being stuck in a local minimum.

5 - Appendix

5.1 Package Install

Pre-requisite:

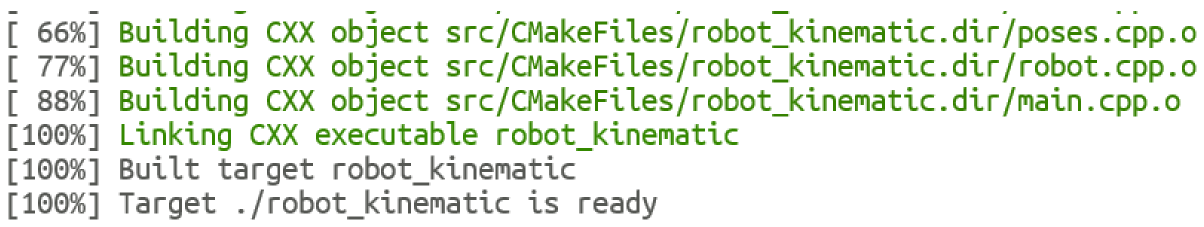
- Make sure that you have a compiler supporting C++20 features, I used g++ version 12.4
- All the robot screenshots were from Blender, I used version 3.0.

Instruction:

- The archive file `ik_package.tar.gz` or `ik_package.zip` must be extracted.
-
- Compile the program with the command
  `make release`

  During the compile, the package fmt, glm and toml are installed under the hood, so first compilation could take a bit of time.

- The last few lines on the console must be similar to these:



```
[ 66%] Building CXX object src/CMakeFiles/robot_kinematic.dir/poses.cpp.o
[ 77%] Building CXX object src/CMakeFiles/robot_kinematic.dir/robot.cpp.o
[ 88%] Building CXX object src/CMakeFiles/robot_kinematic.dir/main.cpp.o
[100%] Linking CXX executable robot_kinematic
[100%] Built target robot_kinematic
[100%] Target ./robot_kinematic is ready
```

  You are now ready to test the program.

Note: In the `source` folder, a config file, `robot.toml`, contains the description of the robotic arm taken from GEN3 (here)

- Try running the program with help option, the output shall be very similar to the one below:

```
shell> ./robot_kinematic --help
Usage: robot_kinematic [options]
Options:
-h | --help              Display this help
--version                Display the program version
--config     filename    Name of the input robot config file in toml format, default to config.toml
--blender    filename    Name of the output blender script, default to my_script.py
--angles     float,...   The angles of the joints in degrees for FK computation, can be repeated
--xyz        float,...   The target position for IK computation, 3 values
--rpy        float,...   The target rotation for IK computation, 3 values
--method     string      Extra option to enable variants in CCD IK computation
--iterations integer     Number of iterations for the CCD IK computation
--seed       integer     Force the seed used by the random number generator
```

- Try now running the program the angles and target options, your output will be very similar to the one below:

```
shell> ./robot_kinematic  --angles -70,-50,90,0,-50,-90 --xyz 0.302,-0.594,0.486 --rpy 0,90,-90
many lines deleted
[INFO]  --------------------------------
[INFO]  --   Start Forward Kinematic   --
[INFO]  --------------------------------
[INFO]  Compute forward kinematic with angles -70,-50,90,0,-50,-90
[INFO]  Not implemented, student code needed here
[INFO]  Target position (x, y, z) = (0, 0, 0)
[INFO]  Target rotation (r, p, y) = (0, 0, 0)
[INFO]  --------------------------------
[INFO]  --   Start Inverse Kinematic   --
[INFO]  --------------------------------
[INFO]  Using seed 674179356
[INFO]  Target position is (0.302,-0.594,0.486)
[INFO]  Target rotation is (0,90,-90)
[INFO]  Not implemented, student code needed here
[INFO]  Final distance after 100 iterations is 3.4028235e+38
[INFO]  Joint angles are (0,0,0,0,0,0)
[INFO]  --------------------------------
[INFO]  --    Write Blender Script     --
[INFO]  --------------------------------
[INFO]  Generate blender script to file my_script.py
[INFO]  Script file generated with 3 poses
```

You are now ready to update the two methods in file `robot.cpp`

- `Robot::forward_kinematic(…)`
- `Robot::inverse_kinematic(…)`

5.2 GLM Library

The open graphic mathematic library (GLM for short, found here) provides all the necessary function to perform the matrix operations described in the previous chapter.

Some useful examples:

```
glm::vec4 my_vec(x, y, z, 1.0f);       // create 4D vector
glm::mat4 my_matrix;                   // create a 4x4 matrix initialized to identity
glm::vec4 my_res = my_matrix * my_vec; // multiply a matrix by a vector
float alpha_d = glm::degrees(alpha);   // convert angle from radian to degree
```