# Assignment 3: Kinematic Chains

## 1.0 Connect to ROS

```
docker rm -f en613_ros2_jazzy
docker run   -p 6080:80   -p 8888:8888   -v "$pwd/assignment3:/home/ubuntu/EN613"
--name en613_ros2_jazzy   tiryoh/ros2-desktop-vnc:jazzy
```

Then launch Ubuntu in browser http://127.0.0.1:6080/

## 2.0 Failure colcon build --packages-select assignment3

```
colcon build --packages-select assignment3
Starting >>> assignment3
/usr/lib/python3/dist-packages/setuptools/dist.py:744:
SetuptoolsDeprecationWarning: Invalid dash-separated options
!!


********************************************************************************
        Usage of dash-separated 'script-dir' will not be supported in future
        versions. Please use the underscore name 'script_dir' instead.

        This deprecation is overdue, please update your project and remove
deprecated
        calls to avoid build errors in the future.

        See https://setuptools.pypa.io/en/latest/userguide/declarative_config.html
for details.

********************************************************************************

!!
  opt = self.warn_dash_deprecation(opt, section)
/usr/lib/python3/dist-packages/setuptools/dist.py:744:
SetuptoolsDeprecationWarning: Invalid dash-separated options
!!
```

```
********************************************************************************
        Usage of dash-separated 'install-scripts' will not be supported in future
        versions. Please use the underscore name 'install_scripts' instead.

        This deprecation is overdue, please update your project and remove
deprecated
        calls to avoid build errors in the future.

        See https://setuptools.pypa.io/en/latest/userguide/declarative_config.html
for details.

********************************************************************************

!!
  opt = self.warn_dash_deprecation(opt, section)
--- stderr: assignment3
/usr/lib/python3/dist-packages/setuptools/dist.py:744:
SetuptoolsDeprecationWarning: Invalid dash-separated options
!!


********************************************************************************
        Usage of dash-separated 'script-dir' will not be supported in future
        versions. Please use the underscore name 'script_dir' instead.

        This deprecation is overdue, please update your project and remove
deprecated
        calls to avoid build errors in the future.

        See https://setuptools.pypa.io/en/latest/userguide/declarative_config.html
for details.

********************************************************************************

!!
  opt = self.warn_dash_deprecation(opt, section)
---
Finished <<< assignment3 [2.29s]

Summary: 1 package finished [2.52s]
  1 package had stderr output: assignment3
```

## Fix setup.cfg

Old

```
[develop]
script-dir=$base/lib/assignment3
[install]
install-scripts=$base/lib/assignment3
```

New

```
[develop]
script_dir=$base/lib/assignment3
[install]
install_scripts=$base/lib/assignment3
```

After that run the following command to load the env variables etc.

```
$ source install/setup.bash
```

# 3.0 assignment3/assignment3.py

## Axis_angle_rot_matrix

The axis_angle_rot_matrix function in assignment3.py is designed to convert an axis-angle representation of a rotation into a 3x3 rotation matrix.

**Function Summary**

- **Input:**
  - k: A 3-element unit vector (x, y, z) representing the axis of rotation.
  - q: A scalar (float) representing the rotation angle in radians.
- **Output:**
  - A 3*3 NumPy array representing the corresponding Rotation Matrix (R).
- **Method**: Although currently unimplemented (marked with #TODO), this function is expected to implement Rodrigues' Rotation Formula, which mathematically relates an axis-angle to a rotation matrix.

**Importance in Robot Kinematics**

1. **Fundamental Building Block:** In robotics, joints typically rotate around a specific axis (e.g., the Z-axis of a joint frame). This function allows you to mathematically represent that rotation as a linear transformation (matrix) that can be applied to vectors.
2. **Homogeneous Transformations:** This function is a prerequisite for the hr_matrix (Homogeneous Representation) function in your assignment. Robot kinematics relies on chaining these matrices together to calculate the position and orientation of the end-effector relative to the base.
3. **Singularity Free:** Unlike Euler angles (Roll-Pitch-Yaw), which suffer from "Gimbal Lock" (loss of a degree of freedom at certain angles), the axis-angle representation is robust and does not have singularities for representing orientation.

# hr_matrix

The hr_matrix function stands for Homogeneous Representation Matrix. It creates a 4 * 4 transformation matrix that combines both rotation and translation into a single mathematical object.

**Function Summary**

- **Input**:
    - k: Axis of rotation (used to calculate the rotation part).
    - t: Translation vector (x, y, z) representing the offset from Frame A to Frame B.
    - q: Rotation angle (theta).
- **Output**:
    - A 4* 4 NumPy array representing the transformation T.

**Why is it important?**

- **Unified Math**: It allows us to treat rotation and translation as a single matrix multiplication.
- **Chaining Transforms**: If you have a robot arm with multiple joints, you can find the position of the hand relative to the base by simply multiplying the matrices of each link.
- **Point Transformation:** It transforms a point P_B (in Frame B coordinates) to P_A (in Frame A coordinates).

# Inverse_hr_matrix

The inverse_hr_matrix function computes the inverse of the Homogeneous Representation Matrix.

**Function Summary**

- Input:
    - k: Axis of rotation.
    - t: Translation vector from Frame A to Frame B.
    - q: Rotation angle.
- Output:

    - A 4*4 matrix representing T^{-1}, which transforms a point from Frame A to Frame B (the reverse of hr_matrix).

**Why not just use np.linalg.inv(T)?**
While you could compute T = hr_matrix(...) and then invert it numerically, calculating it analytically (using the formula above) is:

1. **More Efficient:** Transposing a rotation matrix ($R^T$) is much faster than general matrix inversion.
2. **More Numerically Stable:** It avoids potential precision errors that can accumulate in general inversion algorithms.

**Key Components**

- Rotation Part (R^T): The inverse of a rotation matrix is simply its transpose.
- Translation Part (-R^T t): The new translation vector is the negative of the original translation, rotated by the inverse rotation.

# KinematicChain.pose

The KinematicChain.pose method calculates the position of a specific joint (or a point relative to it) in the global (base) frame.

**Function Summary**

- Input:
    - Q: Array of joint angles (configuration of the robot).
    - index: The index of the joint frame you want to transform from.
    - If index is i, it computes the position of the origin of Frame i (plus offset p_i) in the global frame.
    - Default is -1 (the last joint/end-effector).
    - p_i: An optional offset vector in the index frame. Default is [0,0,0].
- Output:
    - A 3-element vector [x, y, z] representing the position in the global frame.

**How it works (Forward Kinematics)**
It uses the Chain Rule of Homogeneous Transformations.

1. Iterate: Loop from joint 0 up to index.
2. Multiply: At each step, compute the local transformation T_{j to j+1} using hr_matrix(k[j], t[j], Q[j]) and multiply it with the accumulated transformation.
3. Transform Point: Once you have the total transformation apply it to the point p_i
4. Extract: Return the first 3 components (x, y, z) of the result.

**Why is it important?**

1. **Forward Kinematics:** This is the forward kinematics function. It tells you "if I set my joints to these angles, where is my hand?"
2. **Jacobian Computation**: It is often used numerically to compute the Jacobian (by perturbing joints and seeing how the pose changes) or as part of the inverse kinematics loop (to check how close we are to the target).

## KinematicChain.jacobian

The KinematicChain.jacobian method computes the Jacobian matrix for the robot's end-effector position.

**Function Summary**

- Input:
  - Q: Current joint angles.
  - p_eff_N: Offset of the end-effector in the last joint's frame (default [0,0,0]).
- Output:
  - A 3 * N matrix (where N is the number of joints).
  - Each column J_i represents how the end-effector's position changes with respect to joint i.

**Why is it important?**

1. **Inverse Kinematics:** It's essential for numerical IK algorithms (like the Newton-Raphson method used in pseudo_inverse in our code).
2. **Singularity Analysis:** The determinant (or rank) of the Jacobian tells you if the robot is in a singular configuration (where it loses a degree of freedom).
3. Statics: It relates end-effector forces/torques to joint torques.

# 4.0 Testing our code

## How to test?

```
ros2 launch assignment3 assignment3.launch.py
```

It launched visualization as recorded in the following video.
https://www.youtube.com/watch?v=VhMOzeNQctk