Robotics in UR Robots

March 2019

This document is originally developed to be used internally, and Universal Robots A/S is not supposed to provide technical support for the information herein. Universal Robots A/S assumes no responsibility for any errors or omissions in this document.

Contents

Principle of robot motion control 4
Joint and linear spaces 6
Tool center point in linear space 8
Robot orientation 11
Pose transformation 22
Robot kinematics 27
Singularity 31
Dynamics 34
Speed and acceleration 36
Joint control 42
Repeatability 46
Force control 52

Principle of robot motion control

How can the robot move?

Motion is the **basic and essential** feature of industrial robots. There are the most important two for the robot motion: **where to move & how to get there**.

Target position

A **target position** indicates where to move. Target positions are recorded in a robot program and the robot moves to the target positions repeatedly. As the position is the primary target for industrial robots to achieve, we can say that the robots perform ***position control***.
For industrial robots, the action to record the target positions is normally called **teaching**. In UR robots, the target positions are called **Waypoint** and are taught by using hands in Freedrive mode or jogging from Teach Pendant.

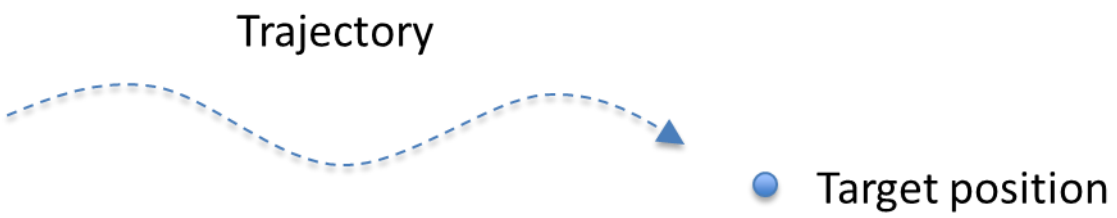


Figure 1 Target position and trajectory

Trajectory

A **trajectory** defines how to reach the target position. A robot can move linearly or by curve. In general, the robot trajectory depends on the space where the robot refers to.
In UR robots, the trajectory is specified by **Move** type on Teach Pendant. We can say that four Move types are available for UR robots: MoveJ, MoveL, MoveP and MoveC. Descriptions of each move type can be found in the User Manual.

Once the target positions are taught and trajectories are specified in the robot program, the robot can move repeatedly according to the program. For industrial robots, it is crucial to keep reaching

the target positions and repetitive tasks are what the robots can do the best. Therefore, **repeatability** is regarded as one of the most important technical specification.

Joint and linear spaces

Two different spaces for the robot

Joint space

A robotic arm is composed of links and joints. It is assumed that links are rigid body and their lengths are fixed. Then the robot takes a certain pose according to a set of joint angular positions. Here, a ***joint space*** can be defined as a space to **represent the robot in the joint coordinate system**.

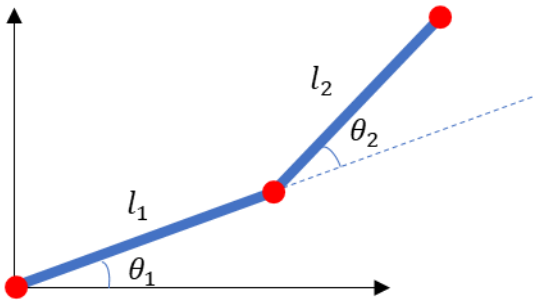


Figure 2 Two-axis link

Because the joints are directly controllable, it is easy to handle the robot in the joint space. For instance, when moving the robot from the current pose to target pose, if the trajectory is not important, we would rotate the joints from the current angles to the target angles. We can call it **joint movement**, which is **MoveJ** in UR robots. The joint movement is usually preferred for the following applications in which the target positions are important, but the trajectory is not essential.

- Pick and place
- Injection molding
- CNC & machine tending
- Packaging and palletizing
- Quality inspection (when the robot stops and inspects at each target point)

Linear space

However, humans live in a 3-dimensional ***linear space*** with X, Y, and Z axes corresponding to the Cartesian coordinate system. We would prefer to see the robot states such as position and orientation in the linear space. Most tasks are defined in the linear space as well. Let us assume a

task to polish the edge line of desk as an example. The robot may have to move linearly to make the smooth straight line. In this case, the trajectory should be defined in the linear space, which can be called **linear movement**, which is **MoveL** in UR robots. For the following applications, the linear movement would be necessary.

- Polishing
- Gluing, dispensing, and welding
- Assembly

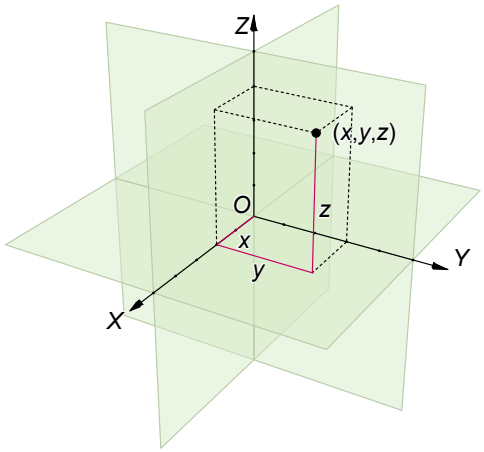


Figure 3 A three-dimensional Cartesian coordinate system[1]

Even for the applications where the joint movement is preferred as mentioned above, the linear movement would partially be needed. For instance, in the pick and place application, the robot typically moves to the approach point first, a little bit above the object and then goes down with the linear movement to pick the object. If the robot goes down with the joint movement, there is a risk to hit the object.

To move the robot in the linear space, it is necessary to convert the robot pose values in the Cartesian coordinate system to the joint positions in the joint space. On the contrary, the joint positions should be converted to the robot pose to identify the robot states in the linear space. The calculation to convert between the two different spaces is called **kinematics**.

[1] *Cartesian coordinate system*, Wikipedia (https://en.wikipedia.org/wiki/Cartesian_coordinate_system)

Tool center point in linear space

An effective point of the robot

TCP (Tool Center Point)

When operating the robot in the linear space, the position and orientation of its effective point should be identified. Then, which part of the robot is an effective point? When considering the fact that a tool, mounted on the end part of the robot called tool flange, performs meaningful tasks, it should be a ***TCP (Tool Center Point)*** to be controlled in the linear space.

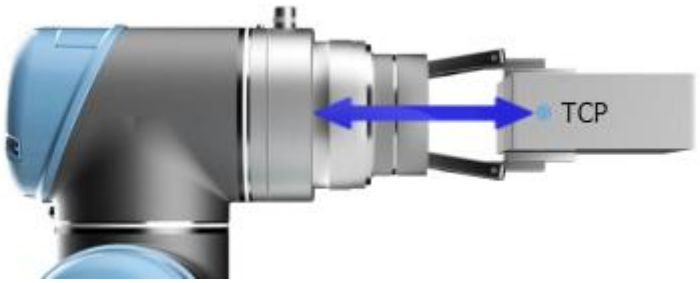
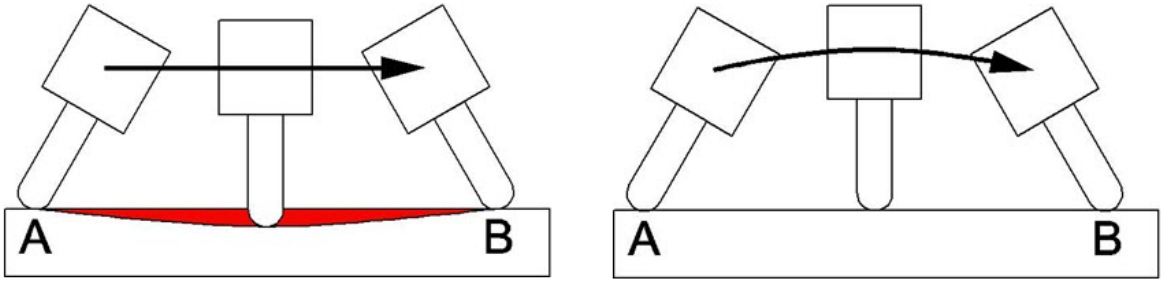


Figure 4 TCP of a gripper

Once the TCP offset is defined relative to the tool flange, the TCP position of the robot can be calculated by kinematics. It would be critical to define the TCP offset especially for the applications that require the linear movement. The best way to understand the TCP is to complete the module in UR Academy.



Tool Path Without TCP

Tool Path With TCP

Figure 5 Comparison of tool path with and without TCP[2]

[2] *Tool Center Point Control (TCP)*, FANUC America (https://www.fanucamerica.com/products/cnc/cnc-technology/tool-center-point-control)

Base and tool coordinate systems

A TCP has **position and orientation** coordinates in relation to the **base coordinate system**, a Cartesian coordinate system located with origin in the center of the robot base. As the robot is mounted on the fixed location, the base coordinate system is also called absolute or world coordinate system. In UR robots, a Cartesian coordinate system is called a **feature**. The **Base feature** is fixed with the Y axis in line with the robot cable and the upward Z axis.

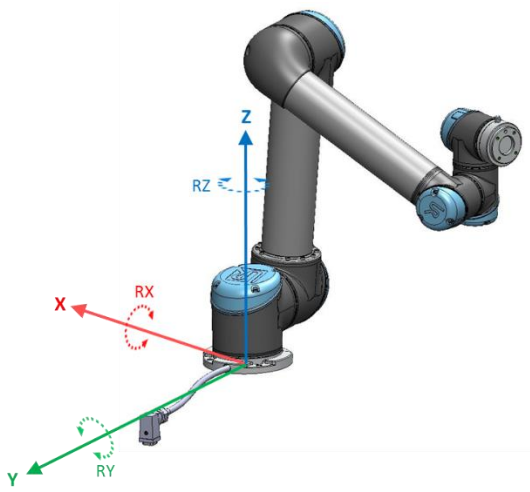


Figure 6 Base feature

As the TCP has position and orientation, the TCP also defines a coordinate system. The **tool coordinate system** makes programming simpler and more accurate.[3] In UR robots, the **Tool feature** is located with origin in the center of the current TCP and changes when the robot moves.

[3] *Robot Basics: The Importance of the Tool Center Point*, Plasma Powders and Systems, Inc. (http://www.plasmapowders.com/thermal-spray-robot-basics-the-importance-of-the-tool-center-point.htm)

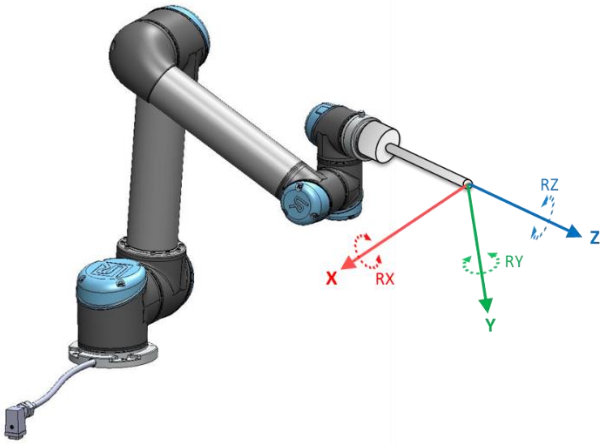


Figure 7 Tool feature

Robot orientation

How to represent the robot orientation

What is a robot orientation?

A **TCP pose** includes the **position** and **orientation**. The TCP position is the location of the TCP in the linear space and the TCP orientation indicates the directions of X, Y, and Z axes in the tool coordinate system. For example, the robot in the figure 1 and 2 is in the same position but its orientations are different. It is obvious that the orientation affects the robotic tasks. Let us assume that a spray paint tool is mounted on the robot in the figure 1 and 2. The painted area depends on the robot orientation as well as the position. Both of position and orientation are basic and crucial on the robot motion.

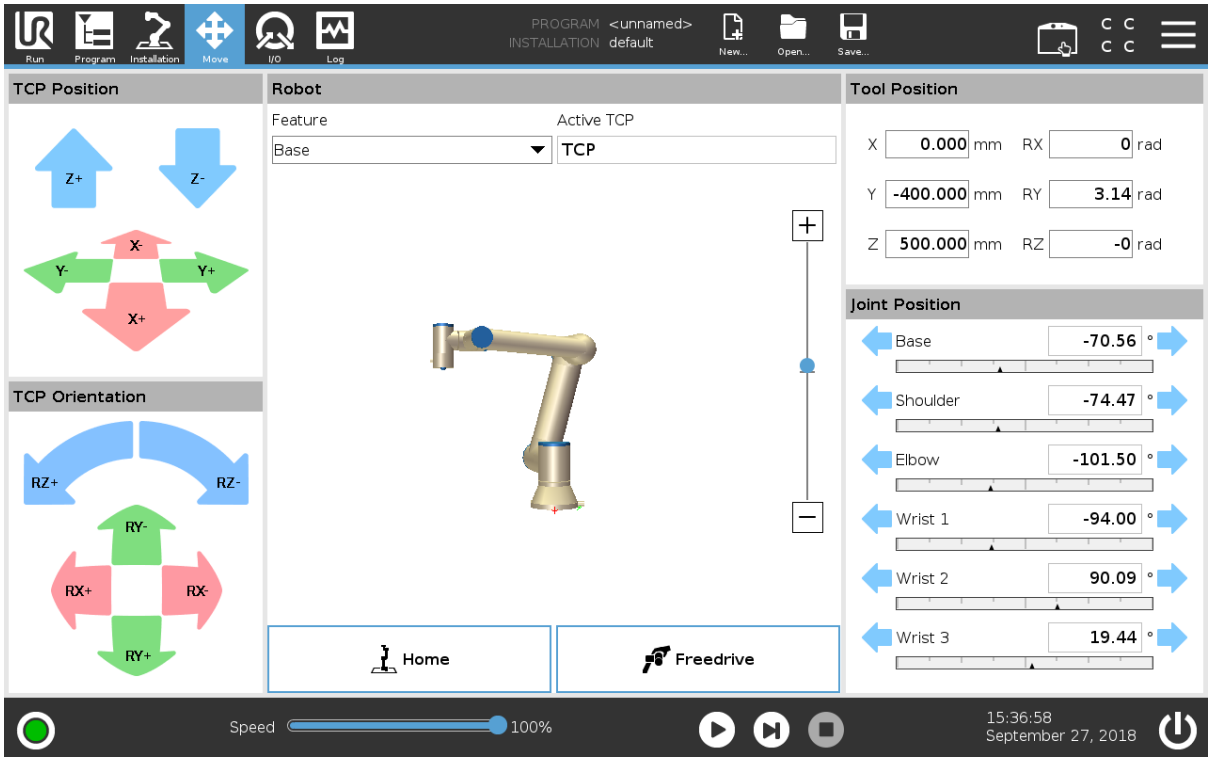


Figure 8 A robot facing down

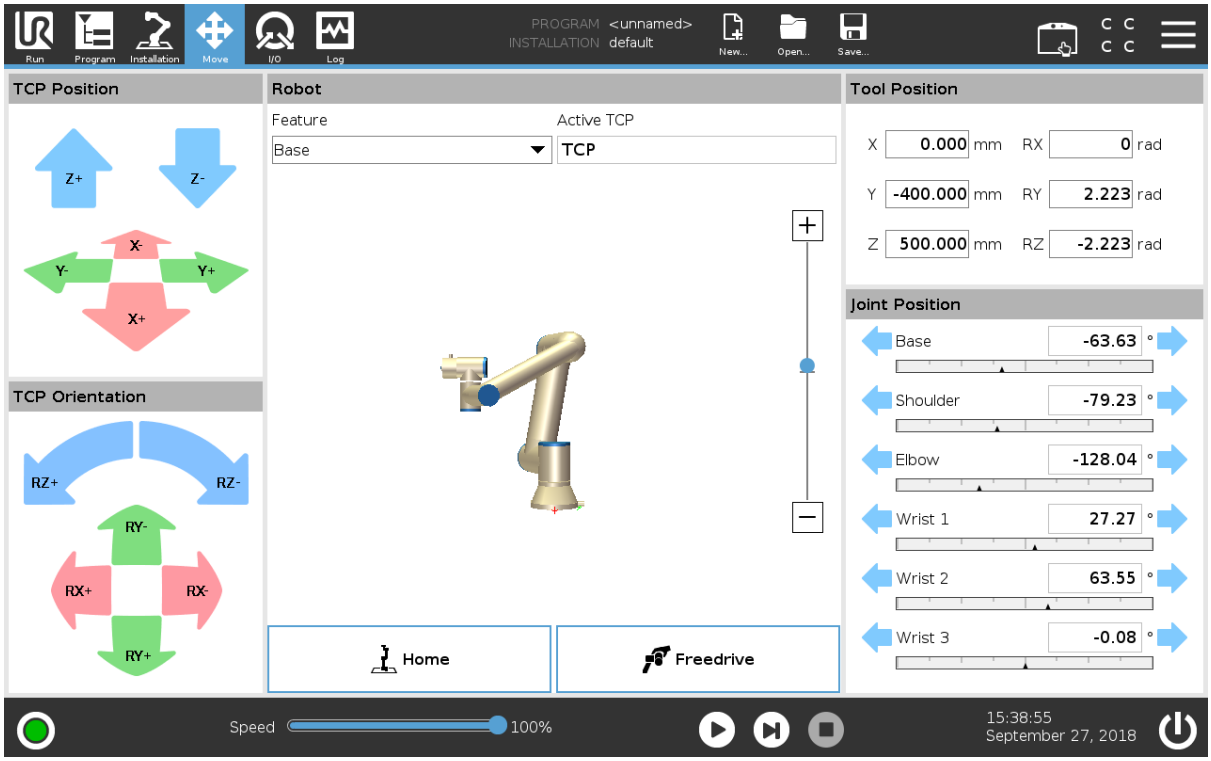


Figure 9 A robot looking aside

A position is represented as three values: X, Y, and Z coordinates. This is intuitive and easy to understand. It may not be difficult to understand the physical orientation of the robot. However, the orientation should be represented as numerical values to control the robot and it is challenging to understand. In robotics, the orientation is represented by using the following rotation concepts.

- Rotation matrix
- Euler angles
- Axis-angle representation
- Quaternion (we do not use it for UR robots)

The advanced use of robot orientation is needed for applications including but not limited to:

- 3D vision
- Multi-robot coordination
- Advanced palletizing
- Offline programming
- Welding

Rotation matrix

In linear algebra, a **rotation matrix** is a matrix that is used to **perform a rotation** in Euclidean space.[4]

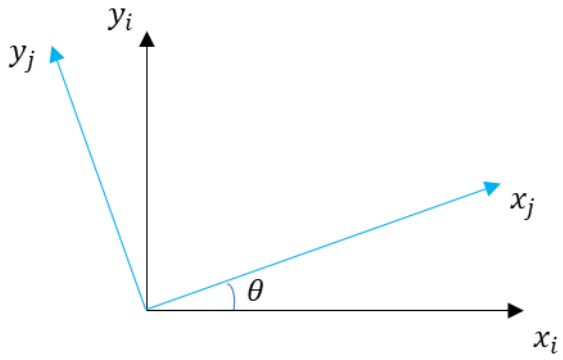


Figure 10 Rotation from the frame $\{i\}$ to $\{j\}$

Let us get started with the rotation in two dimensions. In the figure 3, corresponding to the frame $\{i\}$, the X and Y unit direction vectors of the frame $\{j\}$ are

$$u_{x_j} = \begin{bmatrix}\cos\theta \\ \sin\theta\end{bmatrix},\ u_{y_j} = \begin{bmatrix}-\sin\theta \\ \cos\theta\end{bmatrix}$$

Here, the rotation matrix from the frame $\{i\}$ frame $\{j\}$ is

$$R_{\boldsymbol{ij}} = \begin{bmatrix}u_{x_j} & u_{y_j}\end{bmatrix} = \begin{bmatrix}\cos\theta & -\sin\theta \\ \sin\theta & \cos\theta\end{bmatrix}$$

As an example, the frame $\{\mathbf{1}\}$ is rotated with π rad from the frame $\{0\}$. Then the rotation matrix is

$$R_{\mathbf{01}} = \begin{bmatrix}u_{x_1} & u_{y_2}\end{bmatrix} = \begin{bmatrix}-1 & 0 \\ 0 & -1\end{bmatrix}$$

A position with the coordinates (4, 3) in the frame $\{0\}$ can be calculated to the coordinates in the frame $\{1\}$ by using the rotation matrix.

$$p^{\{1\}} = R_{\mathbf{01}}p^{\{0\}} = \begin{bmatrix}-1 & 0 \\ 0 & -1\end{bmatrix}\begin{bmatrix}4 \\ 3\end{bmatrix} = \begin{bmatrix}-4 \\ -3\end{bmatrix}$$

[4] *Rotation matrix*, Wikipedia (https://en.wikipedia.org/wiki/Rotation_matrix)

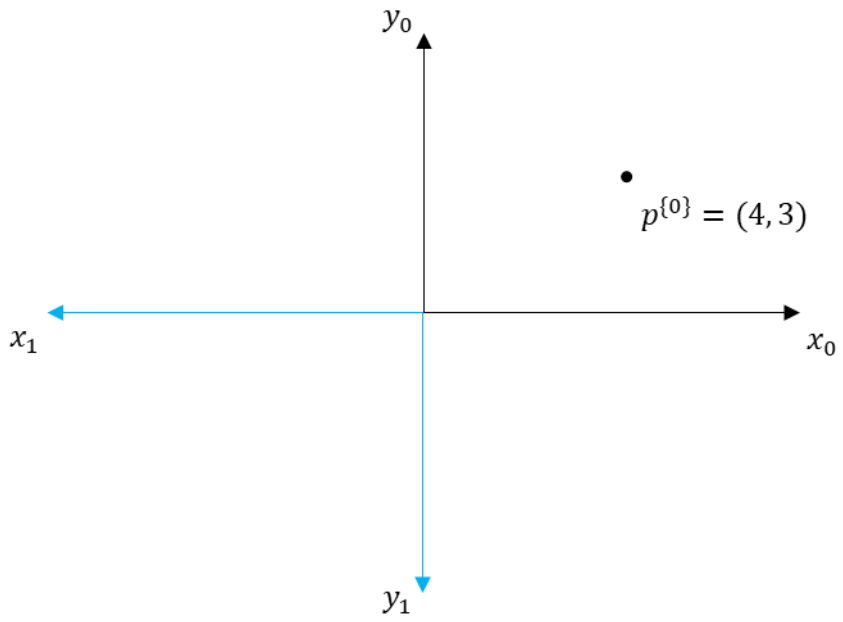


Figure 11 Rotation with π rad from the frame $\{i\}$ to $\{j\}$

As shown in the example of two-dimensional rotation matrix, the **rotation matrix** is determined by the **unit directions vectors in the reference frame**. In addition, the rotation matrix is useful to **calculate the position and orientation values in different coordinate systems**. Note that a coordinate system is composed of an origin and a frame, the directions of axes. In robotics, we use the three-dimensional (3-by-3) rotation matrix.

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

However, it is inconvenient to use 9 numerical values to represent the orientation. Therefore, other ways are also used to represent the orientation.

Euler angles

The **Euler angles** are three angles introduced to describe the **orientation of rigid body** with respect to a **fixed coordinate system**.[5] Any orientation can be achieved by composing three elemental rotations about the axes of a coordinate system. To be specific, rotate the angles of γ, β and α along the X, Y and Z axes from the reference frame, respectively then you will achieve the orientation. In the figure 5, the reference frame is shown in blue and the rotated orientation is shown in red. The angles γ, β and α can also be called **RPY (roll, pitch and yaw)**.

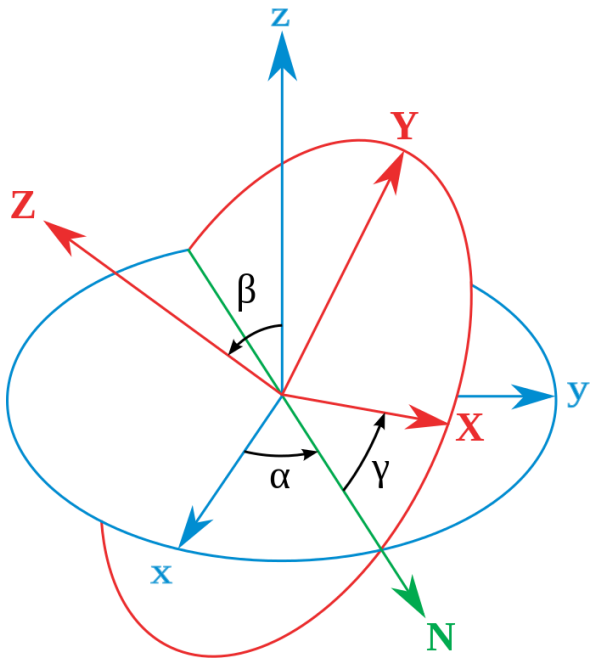


Figure 12 Proper Euler angles geometrical definition

The Euler angles are used to understand the robot orientation, but it should be converted to the rotation matrix for calculations such as kinematics and dynamics. Given Euler angles γ, β and α, the rotation matrix can be calculated by the following equations.[6]

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

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

[5] *Euler angles*, Wikipedia (https://en.wikipedia.org/wiki/Euler_angles)

[6] *Planning Algorithms*, Steven M. LaValle (http://planning.cs.uiuc.edu/node103.html)

$$
\begin{aligned}
R &= R_z(\alpha)R_y(\beta)R_x(\gamma) \\
&= \begin{bmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma \end{bmatrix} \\
&= \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}
\end{aligned}
$$

A given rotation matrix R, the Euler angles are determined as below.

$$
\begin{aligned}
\beta &= \operatorname{atan2}\left(-r_{31}, \sqrt{{r_{11}}^2 + {r_{21}}^2}\right) \\
\alpha &= \operatorname{atan2}\left(\frac{r_{21}}{\cos\beta}, \frac{r_{11}}{\cos\beta}\right) \\
\gamma &= \operatorname{atan2}\left(\frac{r_{32}}{\cos\beta}, \frac{r_{33}}{\cos\beta}\right)
\end{aligned}
$$

The Euler angles are preferred because only three values are needed and geometrically easy to understand compared to other methods. However, there are two critical disadvantages in this representation. The values are not continuous so that calculations with the Euler angles are **complicated**. In addition, the final orientation depends on the sequence of rotations. In other words, even with the same amounts of roll, pitch and yaw, the orientations will be different whether it is rotated with roll first or yaw first as shown in the figure 6, which is **confusing**. In UR robots, RPY values can be monitored in the Move tab of teach pendant. Also, URScript functions *rpy2rotvec* and *rotvec2rpy* are available to support the Euler angles.

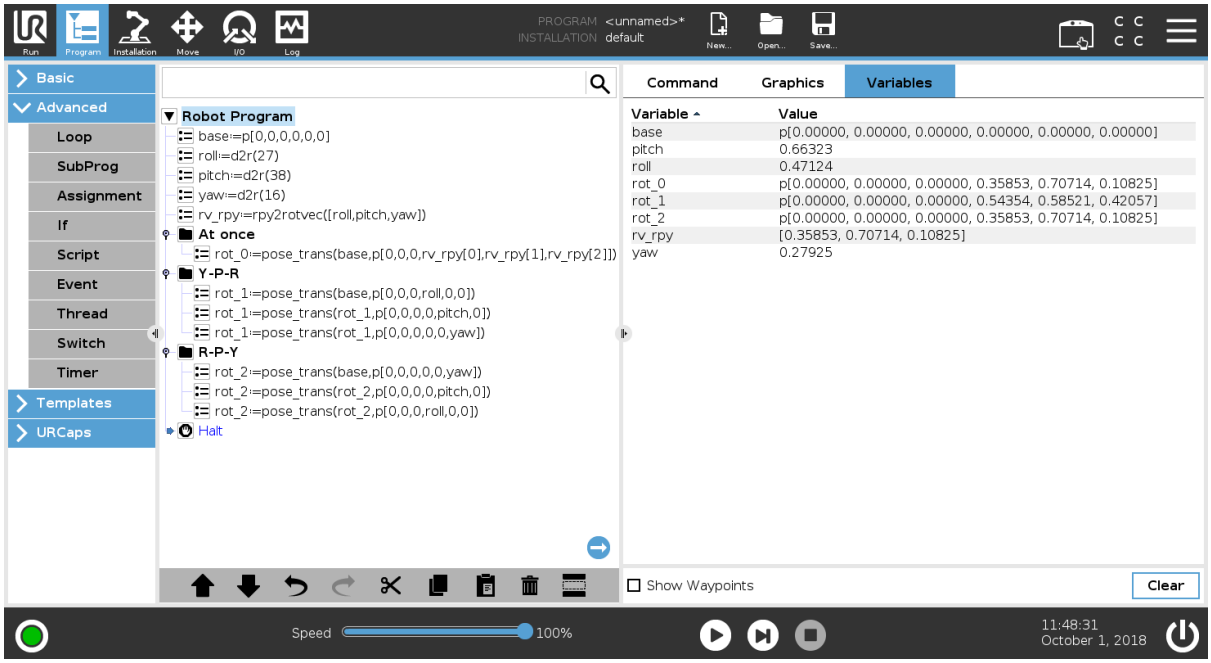


Figure 13 Orientations depends on the sequence of rotations

Rotation vector

The **axis-angle representation** parameterizes a rotation in the linear space by a unit vector u indicating the **direction of an axis of rotation**, and an angle θ describing the **magnitude of the rotation** about the axis.[7] The axis-angle representation can define the relationship between two orientations. For example, n figure 7, the blue orientation can be rotated by the angle θ along the direction vector u to the red orientation.

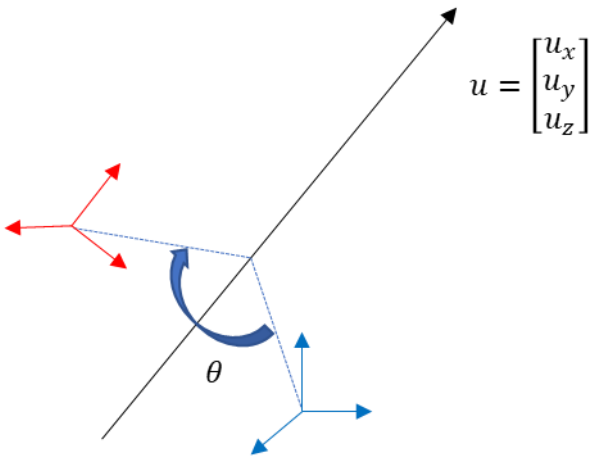


Figure 14 Axis-angle representation

If we have a fixed reference frame such as the base frame in UR robots, we can represent any orientation by four numbers: three elements for the unit direction vector and an angle. If we multiply the angle to each element of the unit direction vector, we can reduce to the three numerical values, which we can call the **rotation vector**.

$$r = \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} = \begin{bmatrix} \theta u_x \\ \theta u_y \\ \theta u_z \end{bmatrix}$$

The axis-angle representation is useful in the robot control because it is free from continuity and rotational sequence issues, the limitations of the Euler angles. However, it is hard to match between the physical orientation and the numerical values of the rotation vectors. In UR robots, we use the rotation vector to represent the orientation of the robot pose.

To convert to the rotation matrix, the rotation vector should be divided into the angle and unit direction vector.

---

[7] Axis-angle representation, Wikipedia (https://en.wikipedia.org/wiki/Axis%E2%80%93angle_representation)

$$\theta = \sqrt{{r_x}^2 + {r_y}^2 + {r_z}^2}$$
$$u = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = \begin{bmatrix} r_x/\theta \\ r_y/\theta \\ r_z/\theta \end{bmatrix}$$

Then, further calculations are needed as follows.

$$\mathrm{c} = \cos\theta$$
$$\mathrm{s} = \sin\theta$$
$$\mathrm{C} = 1 - \cos\theta$$

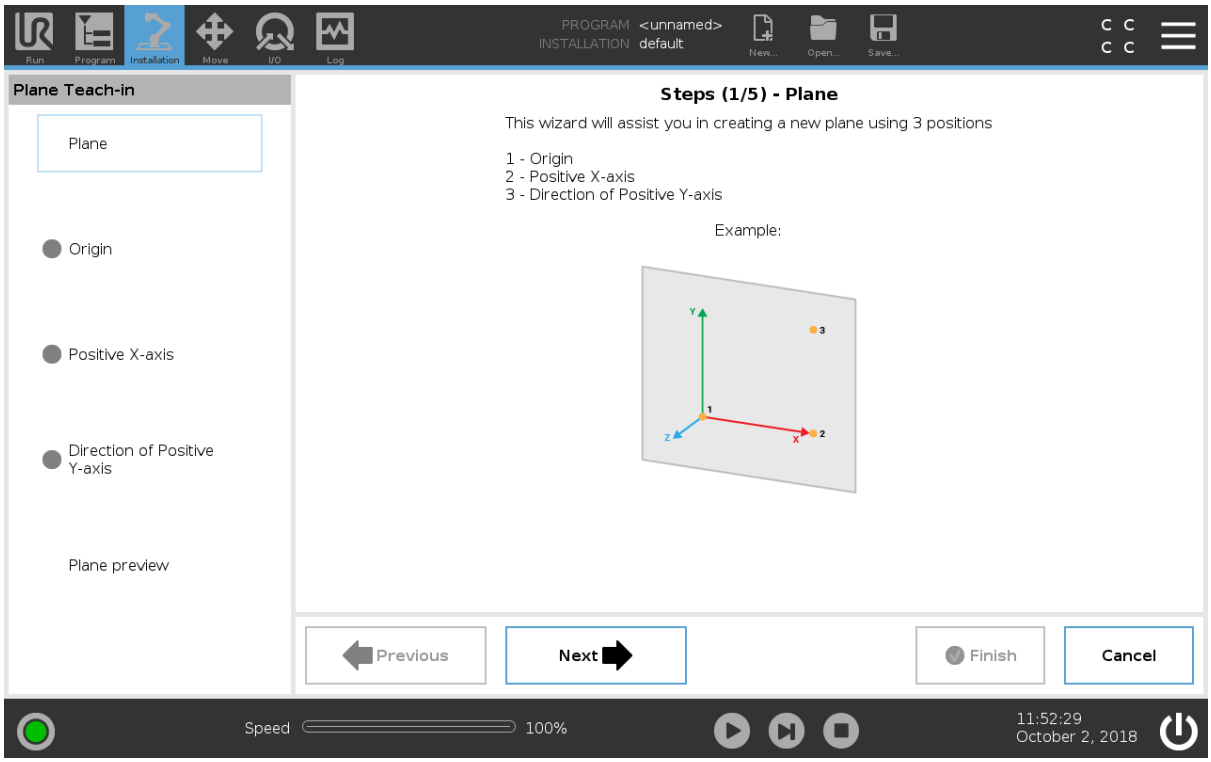
$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} = \begin{bmatrix} u_x u_x C + c & u_x u_y C - u_z s & u_x u_z C + u_y s \\ u_y u_x C + u_z s & u_y u_y C + c & u_y u_z C - u_x s \\ u_z u_x C - u_y s & u_z u_y C + u_x s & u_z u_z C + c \end{bmatrix}$$

Given a rotation matrix, the rotation vector can be calculated as follows.

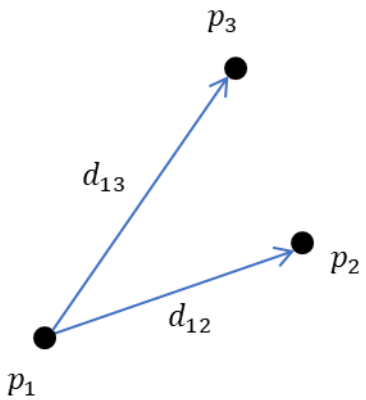
$$\theta = \mathrm{acos}\left(\frac{\mathrm{Trace}(R) - 1}{2}\right)$$
$$u = \frac{1}{2\sin\theta}\begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix}$$

Exercise: URScript function to get a feature plane from three points

(Hint: X, Y and Z axes are determined by the right-hand rule[8])

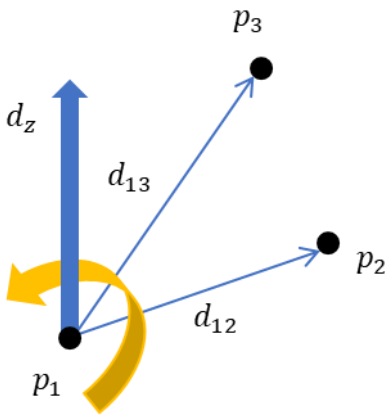


Step 1. Get direction vectors d_{12}(from p_1 to p_2) and d_{13}(from p_1 to p_3).

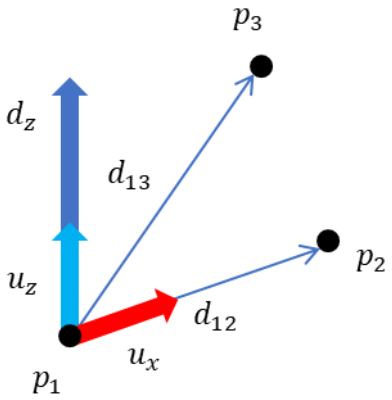


Step 2. Get the direction vector of Z axis by the cross product of d_{12} and d_{13}.
(Hint: the direction vector of Z axis is the normal vector of the plane defined by three points.)

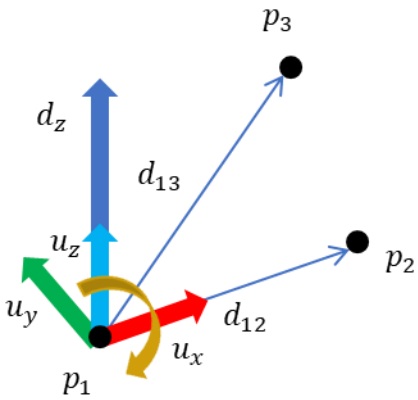
[8] *Right-hand rule*, Wikipedia (https://en.wikipedia.org/wiki/Right-hand_rule)



Step 3. Get X and Z unit direction vectors by normalizing d_{12} and d_z.
(Hint: the direction vector from p_1 to p_2 is the direction of X axis.)



Step 4. Get Y unit direction vector by the cross product of Z and X unit direction vectors.



Step 5. Get the rotation matrix from the unit direction vectors.

$$R = [u_x \quad u_y \quad u_z]$$

Step 6. Convert from rotation matrix to rotation vector
(Hint: use a URScript function *rotmat2rotvec*.)

Step 7. Get feature plane with the origin at p_1 and the frame achieved at Step 6.

Pose transformation

How to describe changes of position and orientation

Transformation matrix

You may need to move the robot from the current pose with the certain amount in position and orientation. Given a current pose and changes of position and orientation, the target pose can be calculated as follows.

$$p_{target} = p_{current} + R_{current}\Delta p$$
$$R_{target} = R_{current}\Delta R$$

The movement in position can be called **translation** and that in orientation can be represented by **rotation**. Both translation and rotation can be combined and mathematically represented as the **transformation matrix** as below.[9]

$$T = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_1 \\ r_{21} & r_{22} & r_{23} & p_2 \\ r_{31} & r_{32} & r_{33} & p_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The transformation matrix can be used not only for the movement but also to **represent a pose**. The above example to move from a current pose with changes of position and orientation can be represented by the transformation matrix as below.

$$\begin{bmatrix} R_{target} & p_{target} \\ 0 & 1 \end{bmatrix} = T_{target} = T_{current}\Delta T = \begin{bmatrix} R_{current} & p_{current} \\ 0 & 1 \end{bmatrix}\begin{bmatrix} \Delta R & \Delta p \\ 0 & 1 \end{bmatrix}$$

Pose transformation in URScript

It can be generalized for the robot movement from a pose T_1 with a certain amount of transformation T_2 and *pose_trans(Pose1, Pose2)* can return the target pose as below.

$$T_{target} = T_1 T_2 = \begin{bmatrix} R_1 & p_1 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} R_2 & p_2 \\ 0 & 1 \end{bmatrix}$$
$$p_{target} = p_1 + R_1 p_2$$

---

[9] *Modern Robotics: Mechanics, Planning, and Control*, Frank C. Park and Kevin M. Lynch

$$R_{target} = R_1 R_2$$

This is exactly the URScript function ***pose_trans*** in UR robots. In the same way, the URScript function *pose_add* can be represented as follows.

$$p_{target} = p_1 + p_2$$
$$R_{target} = R_1 R_2$$

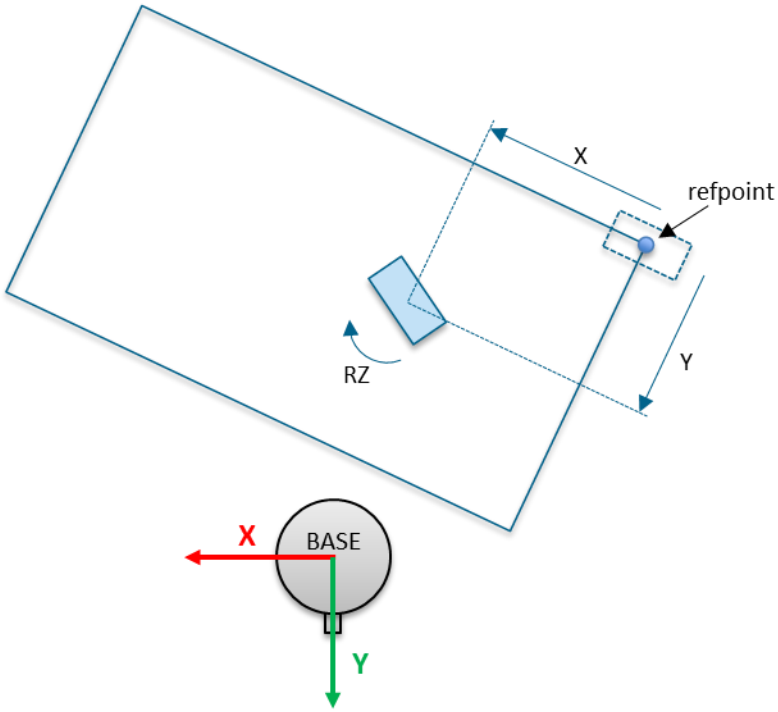
In addition, now we can understand the meaning of the URScript function pose_inv. It is just the inverse of the pose transformation matrix as below.

$$T^{-1} = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix}^{-1}$$

Exercise: 2D positioning with vision system

The target position can be achieved by using the **2D vision system**. Given the offset of p_x, p_y, r_z in the vision frame as reference, the target position in the base frame can be calculated as follows.

$$pose_{offset} = p[\, p_x, p_y, 0, 0, 0, r_z \,]$$
$$pose_{target} = pose\_trans(pose_{reference}, pose_{offset})$$



Exercise: Get TCP force in tool coordinate system

The URScript function *get_tcp_force* returns the TCP force values in the base coordinate system. Please write a script function to return the **TCP force values in the tool coordinate system**.

(Hint: You may have to **translate** the force and torque values, respectively in the inverse tool frame.)

$$f_{Base} = R_{Tool} f_{Tool}$$
$$f_{Tool} = (R_{Tool})^{-1} f_{Base}$$

$$\tau_{Base} = R_{Tool} \tau_{Tool}$$
$$\tau_{Tool} = (R_{Tool})^{-1} \tau_{Base}$$

Exercise: Rotate the TCP orientation with respect to the base frame.

In **an application with a spray gun**, the robot orientation should be rotated with an angle with respect to the base frame axes. Given a TCP pose, a rotational angle and an axis in the base frame, get the target pose.

(Hint: direction vectors of the base frame axes should be represented in the tool frame)

Robot kinematics

How to convert between the joint and linear spaces

A single pose can be described in two ways as there are two spaces: the joint and linear spaces. The calculation to convert between the joint angular positions and the TCP position and orientation is called ***kinematics***.

Forward kinematics

Each joint angle can directly be monitored by the position sensor called encoder. So, we can have the current joint positions. Now, we would like to know the TCP position and orientation of the robot in the linear space. The calculation to convert **from the joint positions to the TCP position and orientation** is called **forward kinematics**. Given joint positions and DH(Denavit-Hartenberg) parameters, the transformation matrices can be defined between adjacent joints. The forward kinematics can be calculated simply by multiplying the transformation matrices from the first to last joints.[10] In UR robots, the DH parameters are available from the support website.[11]

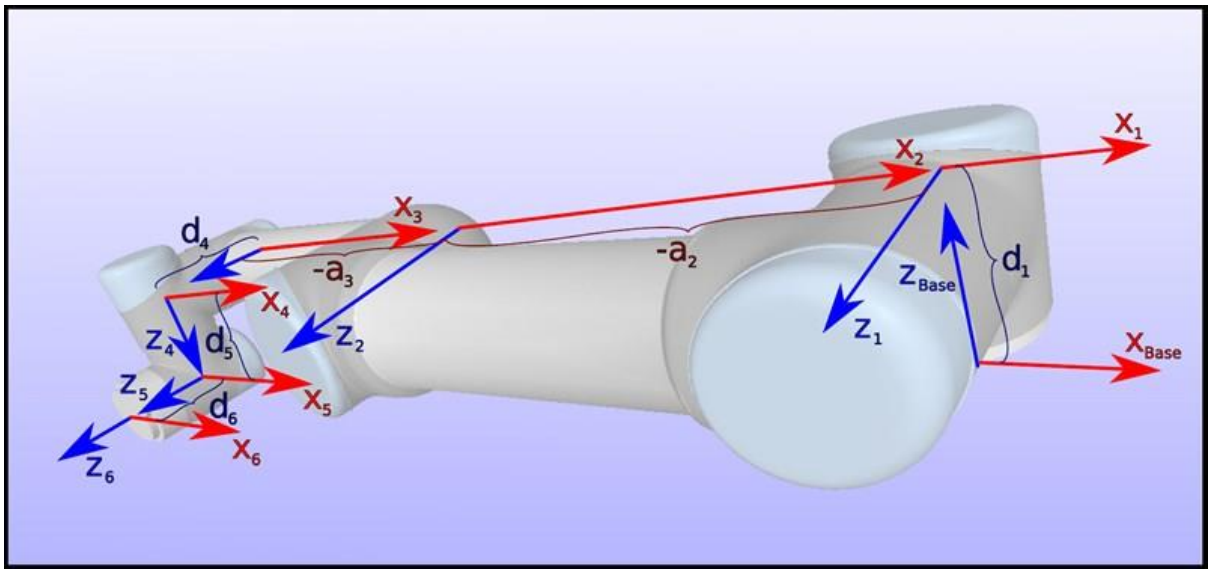


Figure 15 Frames of each joint

[10] *Forward kinematics*, Wikipedia (https://en.wikipedia.org/wiki/Forward_kinematics)

[11] *Parameters for calculations of kinematics and dynamics*, UR Support Site (https://www.universal-robots.com/how-tos-and-faqs/faq/ur-faq/parameters-for-calculations-of-kinematics-and-dynamics-45257/)

$$T_{Base,6} = T_{Base,1} \cdot T_{1,2} \cdot T_{2,3} \cdot T_{3,4} \cdot T_{4,5} \cdot T_{5,6}$$

$$T_{n-1,n} = \mathrm{Trans}_{z_{n-1}}(d_n) \cdot \mathrm{Rot}_{z_{n-1}}(\theta_n) \cdot \mathrm{Trans}_{x_n}(a_n) \cdot \mathrm{Rot}_{x_n}(\alpha_n)$$

$$\mathrm{Trans}_{z_{n-1}}(d_n) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_n \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathrm{Rot}_{z_{n-1}}(\theta_n) = \begin{bmatrix} \cos\theta_n & -\sin\theta_n & 0 & 0 \\ \sin\theta_n & \cos\theta_n & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathrm{Trans}_{x_n}(a_n) = \begin{bmatrix} 1 & 0 & 0 & a_n \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathrm{Rot}_{x_n}(\alpha_n) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_n & -\sin\alpha_n & 0 \\ 0 & \sin\alpha_n & \cos\alpha_n & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The robot controller calculates the forward kinematics all the time in UR robots, and we can get the TCP position and orientation by using the URScript function *get_actual_tcp_pose*.

Inverse kinematics

Once the TCP pose is taught, the joint angles should be calculated to control each joint position. In robotics, **inverse kinematics** makes use of the kinematics equations to **determine the joint positions** that provide **a desired TCP pose** of the robot.[12] The inverse kinematics can be calculated by the iterative optimization with Jacobian matrix.

$$x_n = f(\theta_n)$$
$$\Delta x = x_{target} - x_n$$
$$\theta_{n+1} = \theta_n + J^{-1} x_n \ (if\ \Delta x > \epsilon)$$

It is possible to yield multiple inverse kinematics solutions. In UR robots, the URScript function *get_inverse_kin()* allows to return the solution closest to the reference joint positions.

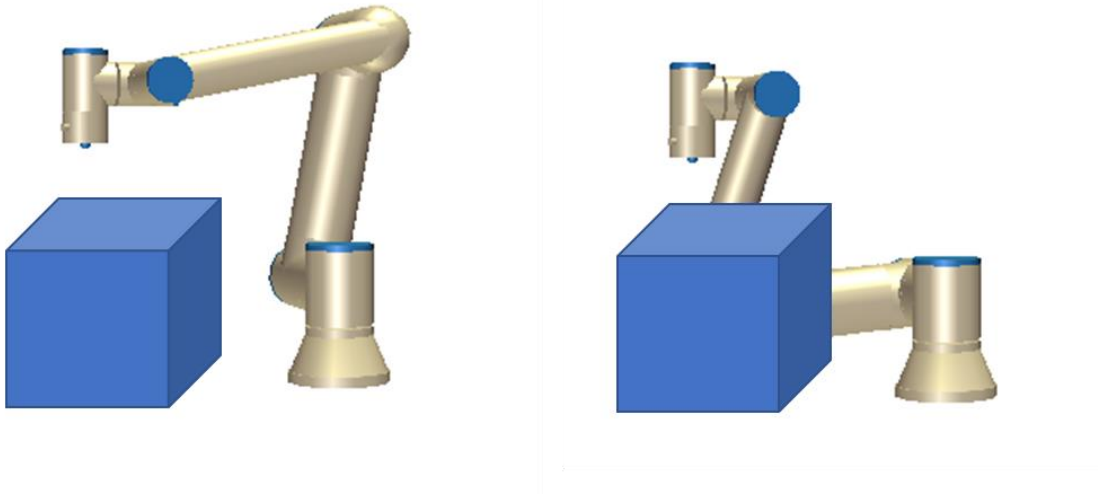
It is important to note that there is no inverse kinematics solution when the robot is in singular areas.

---

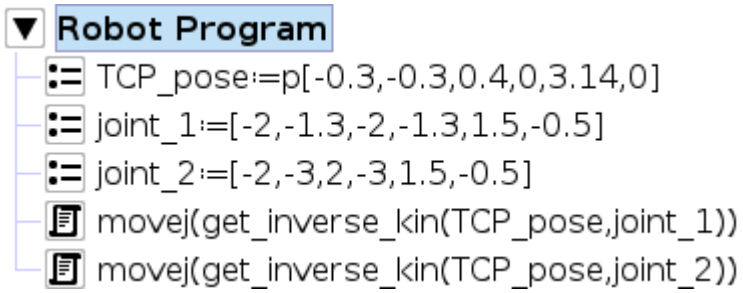
[12] *Inverse kinematics*, Wikipedia (https://en.wikipedia.org/wiki/Inverse_kinematics)

Exercise: Avoid collisions by using q_near

Assume that there is an obstacle near the robot. Given the target TCP pose, the elbow of robot would hit the obstacle depending on the joint positions. We would like to move to the joint positions corresponding to the target TCP pose to avoid the collision.



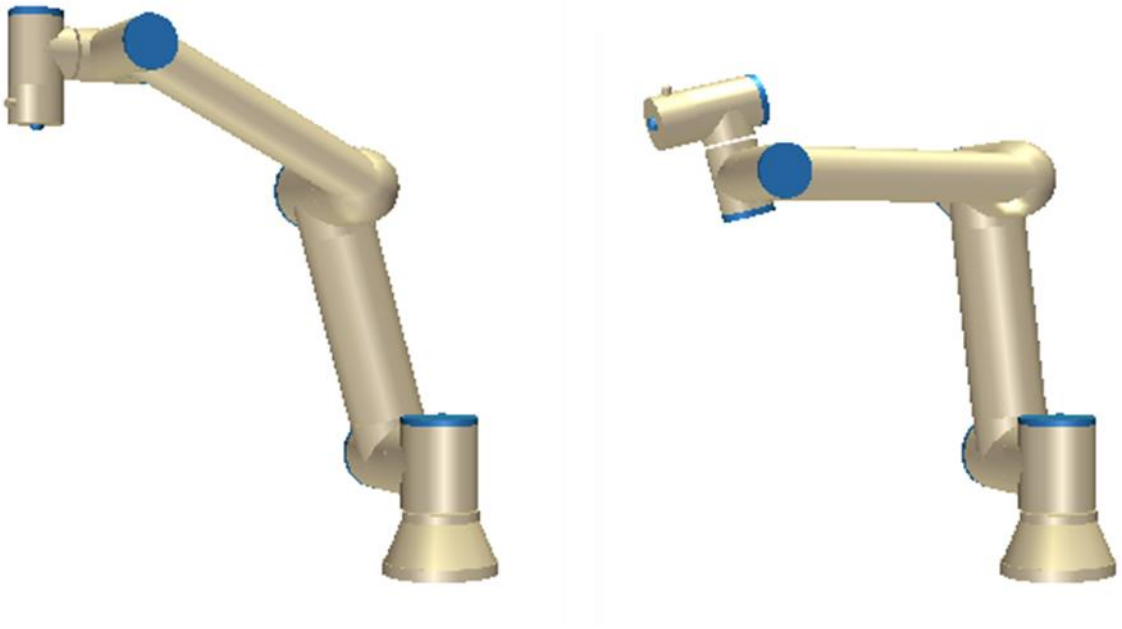
Step 1. Find two different q_near for the target TCP pose.
Step 2. Move to the two joint positions about the target TCP pose by using *movej* and *get_inverse_kin*.



Step 3. Choose the proper q_near to avoid collisions.

Exercise: Limit workspace by wrist 1 position

When the arm is stretched and close to the workspace boundary, the movement would be constrained. However, we are not able to know whether the robot is in the constrained or dexterous configuration by monitoring the TCP position. For example, the two robots below are in the same position but different orientation. The left robot may not move up any more but the right one can move to any directions.



Thus, it would be helpful to monitor the origin position of wrist 1(joint 3) frame in the Figure 1 to ignore the orientation part caused by the wrist joints. Find the wrist 1 position.

$$T_{Base,6} = T_{Base,1} \cdot T_{1,2} \cdot T_{2,3} \cdot T_{3,4} \cdot T_{4,5} \cdot T_{5,6}$$
$$T_{Base,6} = T_{Base,3} \cdot T_{3,4} \cdot T_{4,5} \cdot T_{5,6}$$
$$T_{Base,6} = T_{Base,3} \cdot T_{3,6}$$
$$T_{Base,3} = T_{Base,6} \cdot {T_{3,6}}^{-1}$$

$$T_{n-1,n} = \mathrm{Trans}_{z_{n-1}}(d_n) \cdot \mathrm{Rot}_{z_{n-1}}(\theta_n) \cdot \mathrm{Trans}_{x_n}(a_n) \cdot \mathrm{Rot}_{x_n}(\alpha_n)$$

With the single rotating angle, we can directly put the angle into the pose variable.

$$\mathrm{Trans}_{z_{n-1}}(d_n) = p[0,0,d_n,0,0,0]$$
$$\mathrm{Rot}_{z_{n-1}}(\theta_n) = p[0,0,0,0,0,\theta_n]$$
$$\mathrm{Trans}_{x_n}(a_n) = p[a_n,0,0,0,0,0]$$
$$\mathrm{Rot}_{x_n}(\alpha_n) = p[0,0,0,\alpha_n,0,0]$$

Singularity

Area where the robot is uncomfortable

Robotic arm has many advantages including the high repeatability, fast speed, and wide working range. However, there is a limitation called singularity. There are many ways to describe the singularity but easy to say, **singularity** is a pose where the **robot is uncomfortable**, and the linear movement is limited near the singular area. With respect to the kinematics, there is no inverse kinematics solution within the singular area. Mathematically, the singularity occurs when the rank of Jacobian matrix becomes less than full rank, which means there is no inverse matrix of Jacobian. Physically, the robot loses one or more degree(s) of freedom thus may not be able to move in certain directions. In actual operation, motions defined in Cartesian space that pass near singularities can produce high joint speeds, which is unexpected to an operator.[13] As the robot behavior near singularities is unexpected, it can be dangerous to the operator. It is necessary to understand and avoid the singularity. There are three types of singularities in UR robots.

Outer workspace limit

When the robot is stretched and close to the workspace boundary, the robot movement would be restricted. In the Figure 1, the spherical area with grey color is maximum reach and that with blue color is recommended working area. The robot can reach the grey area, but the movement would be limited when working in the area outside of the recommended reach. To avoid the outer workspace limit, you may arrange the equipment around the robot. If this is not possible, select a UR robot with a longer reach.

Inner workspace limit

In the Figure 1, you can find another grey area. Near the cylinder around the base axis, many positions/orientations will be physically unreachable due to the way the joints are laid out on the robot arm. If you try to move the robot linearly through the cylindrical area, the very high speed in certain joints (normally base joint) will be required even with a slow TCP speed. Layout should be changed in such a way that it is not necessary to work in or close to the central cylinder. If this is unavoidable, use MoveJ instead of MoveL. For the linear trajectory, the points between the two points will be interpolated and inverse kinematics should be calculated for all the interpolated points.

---

[13] ISO 10218-1:2011, *Robots and robotic devices – Safety requirements for industrial robots.*

However, there is no singularity issue in the joint movement as inverse kinematics is not needed.

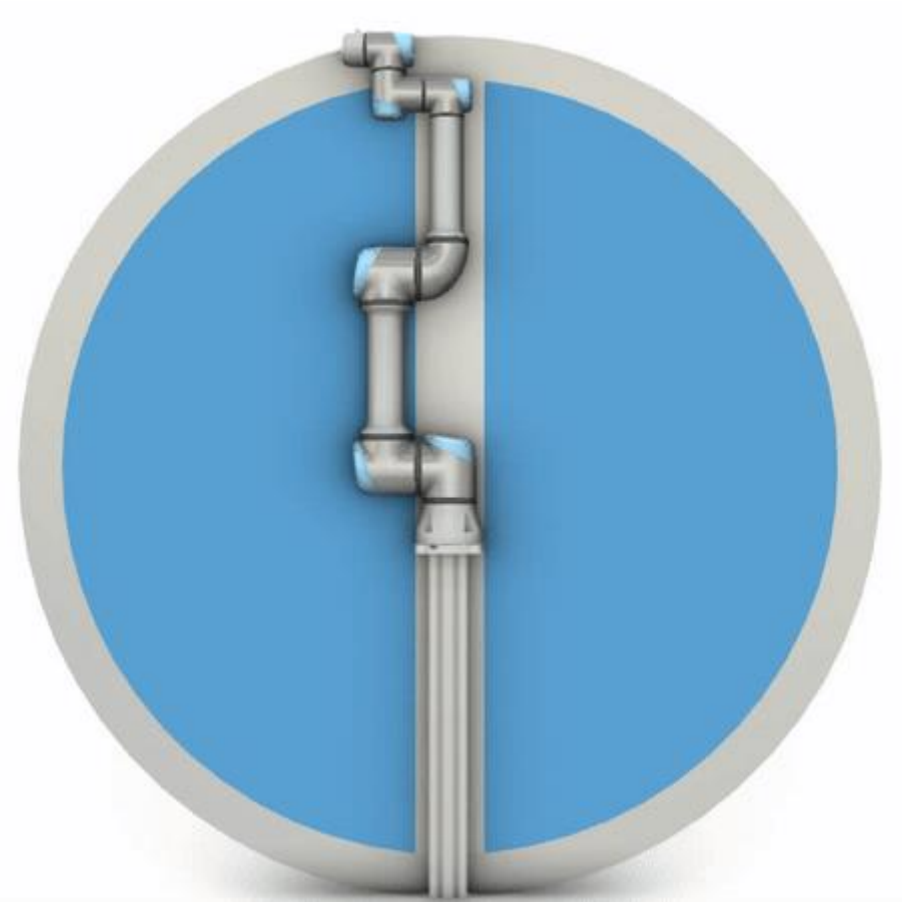


Figure 16 Recommended and maximum working area[14]

Wrist alignment singularity

In UR robots, the shoulder, elbow, and wrist 1 joints rotate in the same plane as shown by the arrows numbered 1, 2, and 3 in the figure 2. However, when we also align the movement of wrist 2 joint with the same plane by moving it to an angle of 0 or 180 degrees, the robot loses a degree of freedom and the robot movement is limited. It is recommended to re-design layout to avoid this wrist 2 joint positions. Alternatively, you may offset the direction of the tool as shown in the Figure 3, so that the tool can point horizontally without the problematic wrist alignment.

[14] *What is a singularity?*, Universal Robots Support Website (https://www.universal-robots.com/how-tos-and-faqs/faq/ur-faq/what-is-a-singularity-42311/)

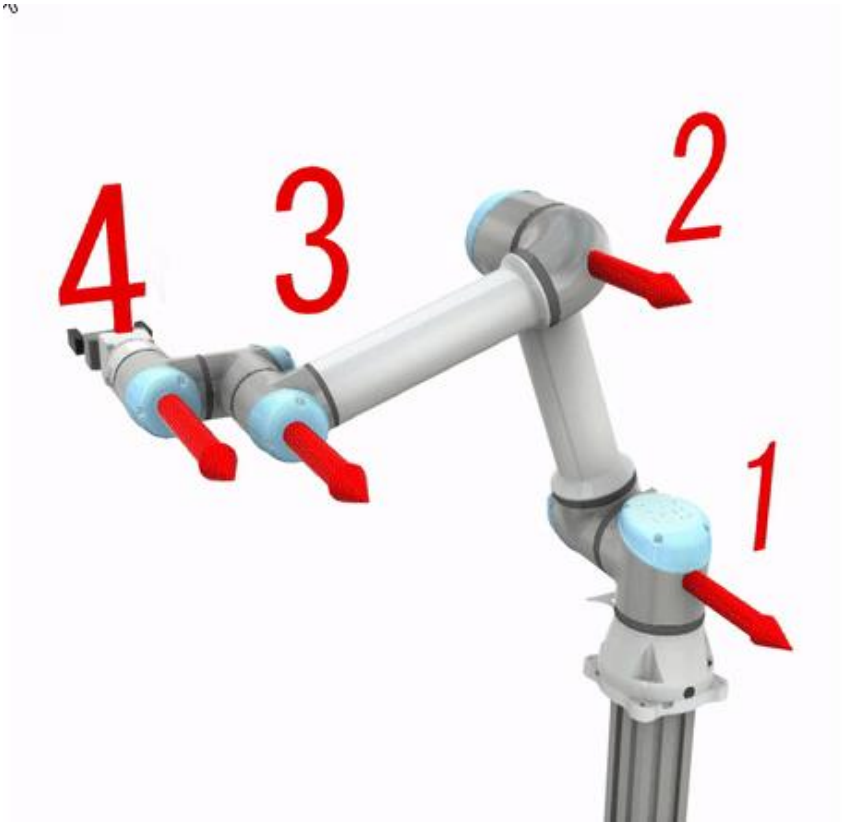


Figure 17 Wrist alignment singularity

Figure 18 Tool offset

Dynamics

Physical characteristics of the robot

When an object moves, there must be a velocity. If the object reaches a certain speed from the standstill, it means the velocity changes and an acceleration occurs. As Newton said, the force is the multiplication of mass and acceleration. The physical characteristics regarding the movement such as mass, position, velocity, acceleration and force can be analyzed by **dynamics**. The robot dynamics is very important as it enables to **calculate necessary joint torques** for each movement and **detect collisions**.

Model torques

In the robot program, we define the target position, speed and acceleration. Then the controller calculates the torques of each joint by the equations of motion as follow.[15]

$$\tau = M(\theta)\ddot{\theta} + h(\theta, \dot{\theta})$$

where $\theta, \dot{\theta}, \ddot{\theta}$ are vectors of joint position, velocity, and acceleration, $M(\theta)$ is a mass matrix containing the mass and center of gravity, and $h(\theta, \dot{\theta})$ is combination of centripetal, Coriolis, gravity and friction terms. If we have the data of the robot model, **given the payload, position, velocity, and acceleration**, the **model joint torques can be calculated**. Because the calculated joint torques are used to control the robot, **the incorrect payload setting** can cause the robot damage. For better understanding, let us consider lifting an object as an example. If you believe that the object weighs 1kg, then your muscle should be ready to hold 1kg. In case when its real weight is 10kg, your muscle and tendon would not be ready, and your shoulder would be dislocated. In contrast, if you expect that the object is 10kg, but the actual weight is 1kg, then you would use too large power and your arms would be injured. Therefore, **it is critical to set the correct payload weight and center of gravity**.

The above equation can be modified to calculate the acceleration from the given joint torques as below. The maximum allowed joint torques would affect the motion planning.

$$\ddot{\theta} = M^{-1}(\theta)\left(\tau - h(\theta, \dot{\theta})\right)$$

---

[15] *Modern Robotics: Mechanics, Planning, and Control*, Frank C. Park and Kevin M. Lynch

Actual torques

From the fact that the motor current is proportional to the motor torque, we can find the actual joint torques.

$$\tau = k_t I$$

In addition, the **joint torques can easily be converted to the TCP force** by using the Jacobian matrix.

$$\tau = J(\theta)^T F$$
$$F = (J(\theta)^T)^{-1}\tau$$

By **comparing the model and actual TCP force** values, the **collision can be detected**. If the model data is not precise, the collision detection does not work well. For example, in case when the payload weight and center of gravity are not correct, the controller might say that the collision is detected even if there was no actual collision.

Speed and acceleration

How fast the robot is?

Some people say that the UR robots are fast, but others claim the opposite. The word "fast" is subjective and would be misleading. However, it is possible to express how fast the robot is by using the objective numerical values, ***speed*** and ***acceleration***.

Joint speed

The speed of an object is the magnitude of its velocity, which is the rate of change of its position.[16] It is obvious that the joint speed represents how fast the joint position changes. The maximum joint speed is typically **determined by** the technical specifications of **motor and gear**. Most industrial robot manufacturers provide the maximum joint speed data. In UR robots, the joint speed depends on the size of joint. In general, the larger joint has the higher torque capacity but tends to be slower.

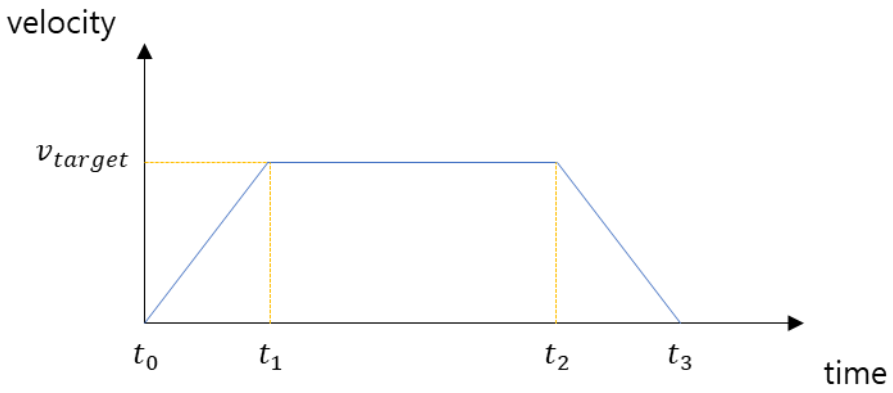


Figure 19 Speed profile

To generate the robot trajectory, the speed profile can be considered. In the Figure 1, a speed profile is described to start at t_0, accelerate the speed to reach the target velocity set in the robot program at t_1, cruise with the constant speed from t_1 to t_2, then decelerate the speed to stop at t_1. The slope is acceleration and the area of profile is the distance.

Linear speed

[16] *Speed*, Wikipedia (https://en.wikipedia.org/wiki/Speed)

A TCP linear speed is important to determine the robot can perform the specific task or not. However, we should be aware that the **linear speed is complex**. To simplify the concept, the linear speed would be the multiplication of the angular speed and the distance from the rotational axis to the TCP.

$$v = l * w$$

With the same joint speed, if the length of link is longer, then the linear speed will be higher. For example, when the base joint speed is same, the TCP speed of the left robot should be much higher than that of the right robot in the Figure 2.

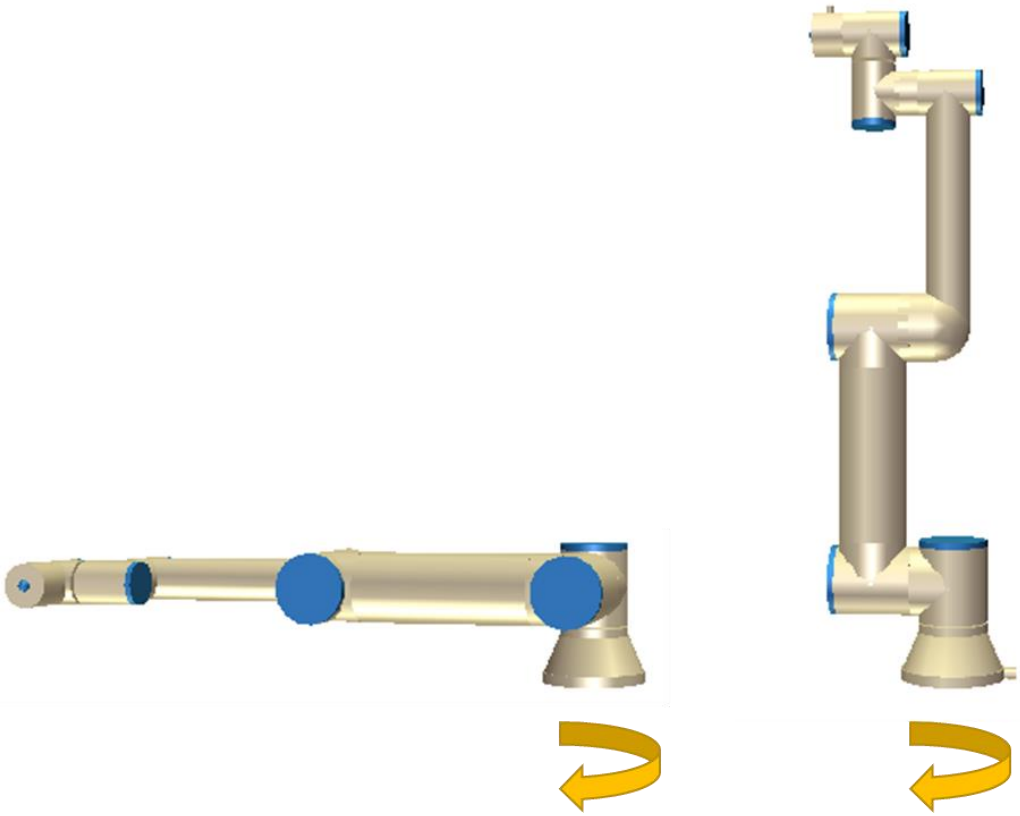


Figure 20 Stretched robotic arms

The TCP linear speed is dependent on the robot pose as well as the joint speed and the length of link. Moreover, most robotic arms have six (less or more) links and joints. Accordingly, there are thousands of combinations in the joint speed and robot pose. Jacobian matrix describes the relationship between linear and joint spaces and the TCP velocities can be calculated by using the Jacobian matrix. However, the Jacobian matrix is also a function of joint position and the matrix is usually difficult to understand.

$$J = \frac{\partial x}{\partial \theta} = \frac{\partial x}{\partial t}\frac{\partial t}{\partial \theta}$$
$$\dot{x} = J(\theta)\dot{\theta}$$

For these reasons, it is sometimes difficult to define the maximum TCP linear speed. Each robot manufacturer tends to have its own way to represent the specification of the linear speed. Some manufacturer claims the speed applicable only for certain poses. For some robots, the maximum linear speed decreases depending on the payload weight and center of gravity. The **TCP linear speed of UR robots is 1m/s** with the maximum payload. All the UR robots can have the same TCP speed as UR10 has longer links but UR3 has faster joints. Basically, this value is achievable if the moving distance is sufficient. It means, if the distance is not large enough, the speed should be decelerated before reaching the target velocity. On the contrary, it is also possible to have the higher linear speed. For instance, when the robot moves with joint movement at stretched poses, the TCP linear speed could be faster than 1m/s. It may not be a good idea to select the robot for specific applications based only on the maximum TCP speed in the tech spec sheet. For UR robots, the brief validity can be simulated with URSim.

Blending

In automation process, a cycle time can be considered as a critical performance index. To reduce the cycle time, you may try to increase the speed. However, according to the speed profile, we can find that the robot spends a lot of time on acceleration and deceleration. In UR robots, blending enables to avoid stopping and restarting at each waypoint.

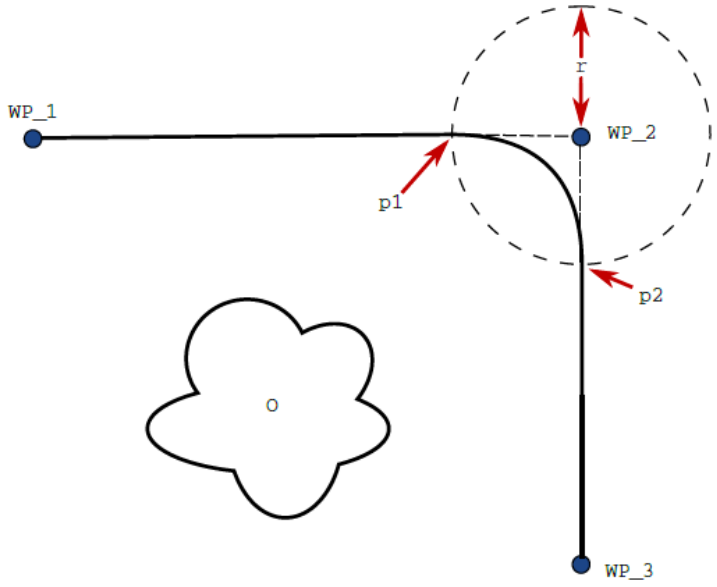


Figure 21 Description of blending

The Figure 3 shows that the robot blends over WP_2 with radius r. Instead of stopping at WP_2, the robot moves around the WP_2 by minimizing to reduce the speed.

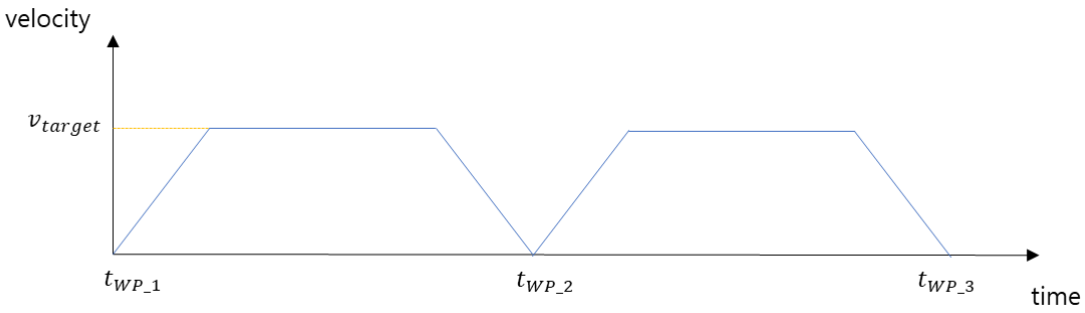


Figure 22 Speed profile without blending

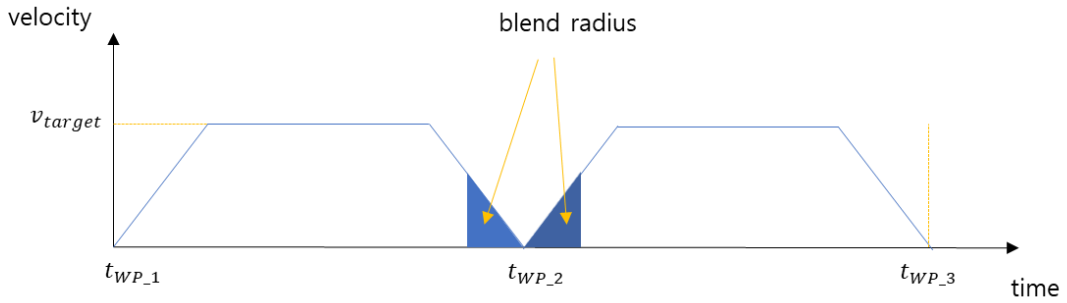


Figure 23 Speed profile with blend radius

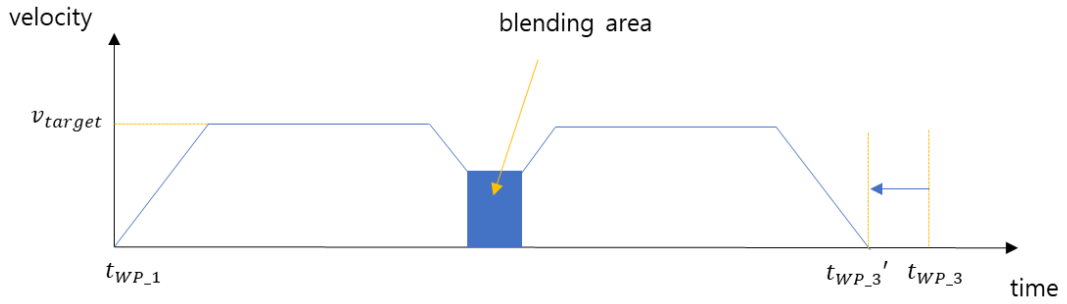


Figure 24 Speed profile with blending

The blending can be described as speed profiles in the Figure 4, 5, and 6. The cycle time can be reduced from t_{WP_3} to t_{WP_3}' by adding the blend radius to the movement.

Acceleration

Beside adding the blend radius, it could be a way to adjust the acceleration to optimize the robot movement. For example, if the moving distance is not large enough, increasing speed may never

affect the cycle time at all. The cycle time can be reduced by increasing the acceleration shown as the Figure 7.

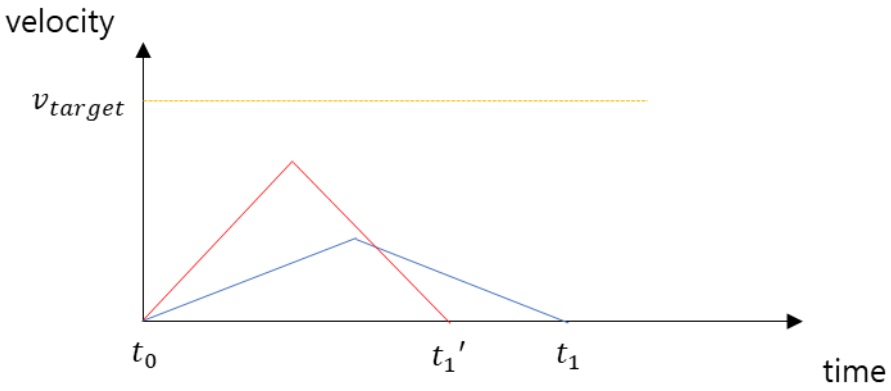


Figure 25 Speed profile with different accelerations

However, it is important to note that too high acceleration can cause protective stops. Thus, the acceleration should be optimized by avoiding the problems.

Safety limit

Some people are confused of the maximum TCP speed as safety limit and as technical specification. For example, the maximum TCP speed is 1m/s, but the default parameter of maximum TCP speed limit is 1.5m/s. However, the objectives are different. The maximum speed as technical specification is about how fast the robot can move. Thus, the robot should be able to perform the specification. On the other hand, the purpose of maximum speed for safety function is to prevent the robot to reach the speed. It is not allowed to move faster than this parameter with MoveJ as well as MoveL. If the robot is faster than the safety limit parameter, the safety system will turn off the power.

Exercise: How to optimize the acceleration?

(Hint: use the URScript function position_deviation_warning)

Narrow down the waypoint that causes large deviation
Adjust speed and acceleration for that waypoint.

Joint control

Low level control

You may be interested only in the TCP, but it is each joint to move the robot. Looking into the joint, there are three key components: motor, encoder, and gear.

Motor

A ***motor*** converts from the electrical energy to the kinetic energy and generates fast **rotational speed**. In UR robots, permanent magnet AC synchronous drive motors are used in each joint.[17] The frameless motor designed to be directly embedded in joint enables safe motion with high performance.[18] With respect to the motor drives, the vector control method is used.[19]

Figure 26 Kollmorgen TBM Direct Drive Frameless Motors

---

[17] *What type of motors are used in the robot*, UR Support Site (https://www.universal-robots.com/how-tos-and-faqs/faq/ur-faq/what-type-of-motors-are-used-in-the-robot-17270/)

[18] *Human Collaboration: The Future of Robotics*, Kollmorgen (https://www.kollmorgen.com/en-us/solutions/robotics/collaborative-robots-cobots/)

[19] *What type of motor drivers are used in the robot*, UR Support Site (https://www.universal-robots.com/how-tos-and-faqs/faq/ur-faq/what-type-of-motor-drivers-are-used-in-the-robot-17269/)

Encoder

For the precise position control, the angular position of the motor is monitored by the **position sensor** called ***encoder***. In UR robots, two encoders are used in each joint. A magnetic encoder monitors the motor position and an optical encoder monitors the position of output flange, so called joint position. The two redundant position sensors make the robot more reliable. In addition, unlike traditional robots equipped only with incremental encoders which require the battery backup, the absolute magnetic encoder disk remembers the previous position without the battery. The magnetic joint encoder has 20 bit of resolution and 0.1 degree of accuracy, and the repeatability is better than the unit of resolution.[20]

Reduction gear

A motor generates a high speed of rotation, but its power is not sufficient to handle the (relatively) heavy objects such as robot links and tools. The **reduction gear** can convert the high speed with low torque to the **low speed with high torque**. As an example of a gear with 10:1 of reduction ratio, when the motor rotates 10 revolutions, the joint output flange rotates 1 revolution. The rotational speed decreases 1/10, but the torque will be 10 times higher. You can easily imagine the principle of the gear from the bicycle. In UR robots, a **strain wave gear** is mounted in each joint. This is a type of reduction gear and well known as a ***harmonic drive***.[21]

Joint control

When the TCP path is generated, it is converted to the target positions to be controlled. In each joint, a single target position is to be achieved usually by a simple control algorithm like the PID control. The joint control is also called a **low-level control**.

In UR robots, it is basically not allowed to change any parameter in the low level. However, there is a URScript function ***servoj*** that can be used for **online control of the robot joint position**. For each joint, it calculates the error between the desired position and the actual position and adjusts the position accordingly. As input arguments, users can set the lookahead time and the gain to smoothen or sharpen the trajectory. The magnitude of the error is modified with the parameter

---

[20] *AksIM™ supports Universal Robots for smart factory automation*, Renishaw (https://www.renishaw.com/en/aksim-supports-universal-robots-for-smart-factory-automation--40389)

[21] *Strain wave gear principle*, Harmonic Drive (https://www.youtube.com/watch?v=bzRh672peNk)

gain, which can range from 100 to 2000. Its effect is similar to the P gain of a PID controller. The higher this value is, the faster reaction the robot will have. However, it is more likely to produce oscillations and shakiness. The default value is 300. The parameter **lookahead_time** plays a role in the error calculation by predicting the future position of the joint some time in advance. It can range from 0.03 to 0.2 seconds. The higher this value is, the slower the reaction will be. Nevertheless, the movement will become smoother and with fewer sharp movements. The default value is 0.1.

Exercise: How to control the robot in real time?

Note: RTDE can be used for real-time control. If you send the script function via client interfaces (30001~30003), each script should be compiled, and the controller cannot guarantee the desired time interval.

In the main program, execute the URScript servoj.
In thread, receive the joint positions via RTDE input_double_register_x in thread every 8msec.

Repeatability

How accurate the robot is?

What we expect to the industrial robot is fast and accurate motion. It is interesting to note that the accuracy can be ambiguous. In the shooting sports, if the shots are closer to the center, the scores will be higher. In the Figure 1, the left should achieve the higher score in the shooting sports. Then, can we say that the left is more accurate for the industrial robots? To answer for the question, we need to clarify the definition of accuracy. In the field of robotics, the two different terms are used to avoid confusion.

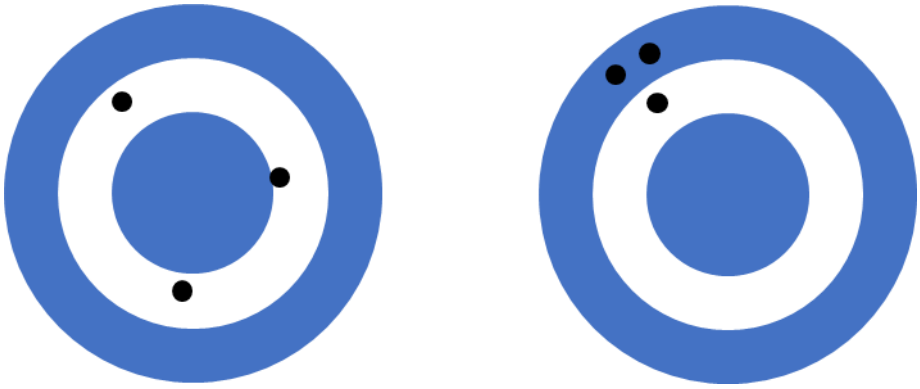


Figure 27 Results of shooting sports

Repeatability

In most cases, the physical target positions are taught, and the trajectories are defined in the robot program. Then, the robot will move to the same target positions again and again to perform the assigned tasks as long as the program is running. If the positions where robot reaches change each cycle, it would affect the tasks in a negative way. A technical specification, **repeatability**, presents the maximum deviation when the robot keeps reaching the same target position. In the Figure 1, the right one has a better repeatability because all three shots are close each other despite the lower score.

The repeatability is regarded as a primary criterion on the robot accuracy and most robot manufacturers provide the value. When measuring the repeatability, the following equations referred from ISO 9283 can be used.[22]

[22] ISO 9283:1998, *Manipulating industrial robots – Performance criteria and related test methods*

$$l_j = \sqrt{(x_j - \bar{x})^2 + (y_j - \bar{y})^2 + (z_j - \bar{z})^2}$$

$$\bar{l} = \frac{1}{n}\sum_{j=1}^{n} l_j$$

$$S_l = \sqrt{\frac{\sum_{j=1}^{n}(l_j - \bar{l})^2}{n-1}}$$

$$RP_l = \bar{l} + 3S_l$$

In general, the performance of repeatability is affected by the encoder resolution, gear backlash and motor control software. In UR robots, the gear has less than 0.1 arcmin (0.03 mrad) of repeatability[23] and the encoder has 20 bit of resolution.[24] The repeatability of UR3e and UR5e is +/-0.03mm and that of UR10e is +/-0.05mm.

Absolute accuracy

In theory, given DH parameters and joint positions, the TCP position and orientation can be calculated by the forward kinematics model. However, in reality, the model TCP pose may not be same as the physical pose. This is mainly because of the joint zero position and the mechanical clearance such as the length of link. The **absolute accuracy** represents the deviation between the model and actual poses. It is hard to guarantee the absolute accuracy due to the mechanical limitations. Thus, most robot manufacturers do not offer the data.

However, the absolute accuracy is still important for some situations/applications. For example, if you use the vision system or program the robot by using the offline simulation software, the absolute accuracy and linearity would be critical. Also, when you replace the joint or the entire robot, it should be very important to avoid re-teaching all the target points.

Kinematic calibration

In the Figure 1, the right player can fix their accuracy by adjusting the targeting. However, in the robotic arm, a little bit more complex method, kinematic calibration is required. The kinematic

---

[23] *Engineering Data HFUC 2UH*, Harmonic Drive AG (https://harmonicdrive.de/fileadmin/user_upload/2014_12_ED_1019644_HFUC_2UH.pdf)

[24] *Data sheet: AksIM off-axis rotary absolute encoder*, Renishaw (https://www.rls.si/en/aksim-off-axis-rotary-absolute-encoder)

calibration can be performed by measuring the robot position by the external devices such as laser tracker. In UR robots, we conduct the dual robot kinematic calibration during production. The two robots are connected and move together then optimize the calibration parameters. The benefit is to complete the calibration without the use of external measurement equipment.[25] Once the calibration is completed, you will be able to check the statistics data to show the absolute accuracy.[26] The DH parameters of calibrated robots should be different from the model values. Thus, if the TCP pose of real robot is different from that of URSim, this is because of the kinematic calibration.

Figure 28 Dual robot calibration

Due to the discrepancy in calibration parameters between different robots, the same joint positions may have different actual positions. It means, a robot program in a robot may not work well in another robot. To minimize the gap, when loading the robot program which is made in another robot, the controller would ask to convert the kinematics.

---

[25] *Calibration and Programming of Robots*, Universal Robots (https://patents.google.com/patent/US20130079928A1/en)

[26] *Kinematic Calibration Manual*, Universal Robots (https://s3-eu-west-1.amazonaws.com/ur-support-site/41410/calibrationManual.pdf)

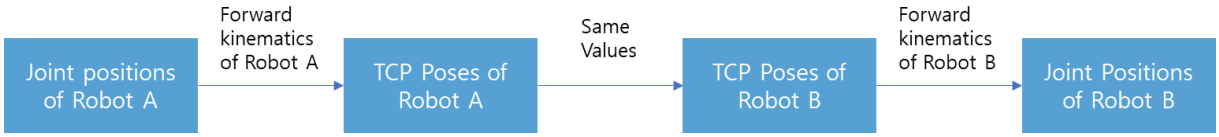


Figure 29 Process to convert kinematics

When replacing the joint or the whole robot, if it is not possible to perform the kinematic calibration, *Program Correction by Key Waypoints* can be used in UR robots. In the robot program, there should be key waypoints of which the position is critical. After replacing the joint or the entire robot, then load the robot program. What you have to do is to re-teach the key waypoints. Then, other waypoints will be adjusted automatically.

Exercise: How to calculate forward kinematics for calibrated robots

You can find the kinematic calibration parameters from *calibration.conf* file. What you have to do is just to add the delta parameters to the model DH parameters.

$$
\begin{aligned}
d_{calib} &= d + \Delta d \\
\theta_{calib} &= \theta + \Delta\theta \\
a_{calib} &= a + \Delta a \\
\alpha_{calib} &= \alpha + \Delta\alpha
\end{aligned}
$$

Note: delta_d values may be very large. This is because the Z axes of shoulder and elbow joints are parallel. However, the units are radians and meters, and the result of forward kinematics should be fine.

Exercise: Convert absolute accuracy level to repeatability level

Iterative learning

Force control

What is force control?

To understand the force control, the first thing you have to do is to remind of the principle of robot motion control. As we discussed at the beginning, the robot performs the position control, which means the position is the target to be achieved in the control loop. During the force control, the robot does not perform the position control. In other words, in case when the **force** is the **main target** to achieved **instead of the position**, it is called the ***force control***.

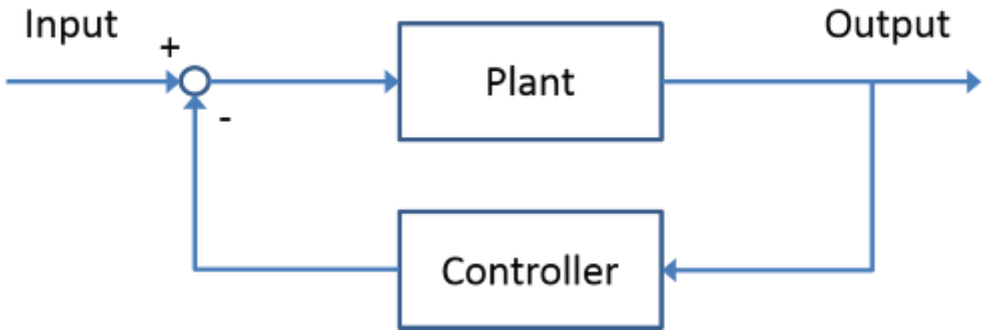


Figure 30 General control loop diagram

Motion for force control

According to the Newton's second law of motion $F = m \cdot a$, if the force exists, there should be the acceleration and the robot would move. In practice, however, the robot may not move until the force is larger than a certain amount of force, called a static friction force. Once the robot starts to move, the frictional force will change to the kinetic friction force, which is smaller than the maximum static friction force. In short, if the target force is large enough to overcome the friction force, the robot will move.

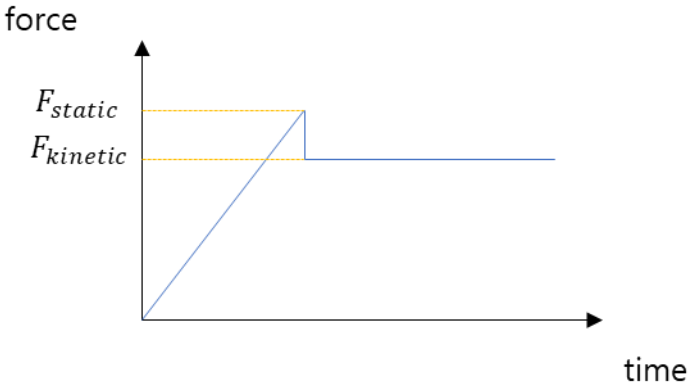


Figure 31 Friction force curve

In UR robots, a **force mode** enables the robot to perform the force control, given the direction and target force value. In the Figure 3, the target force is 30N to the left direction. During the force mode active, the robot will move to the left.

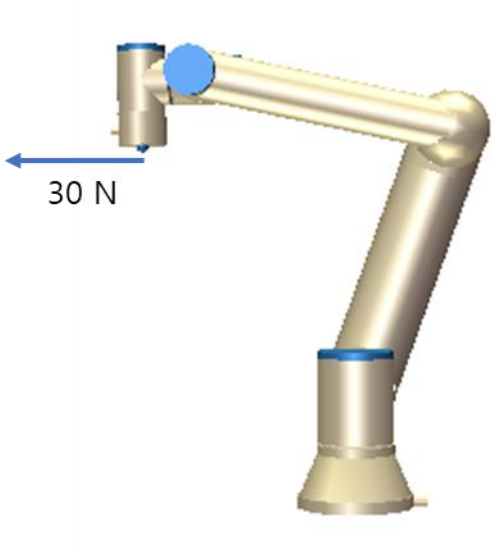


Figure 32 Force control

Compliance control

During the force control, if external force is added to the robot, what will happen? In the Figure 4, for instance, while the robot is doing the force control with the 30N of target force to the left direction, a human operator pushes the robot with 50N to the right direction. In this case, the net force will be 20N to the right direction, thus the robot would move to the right.

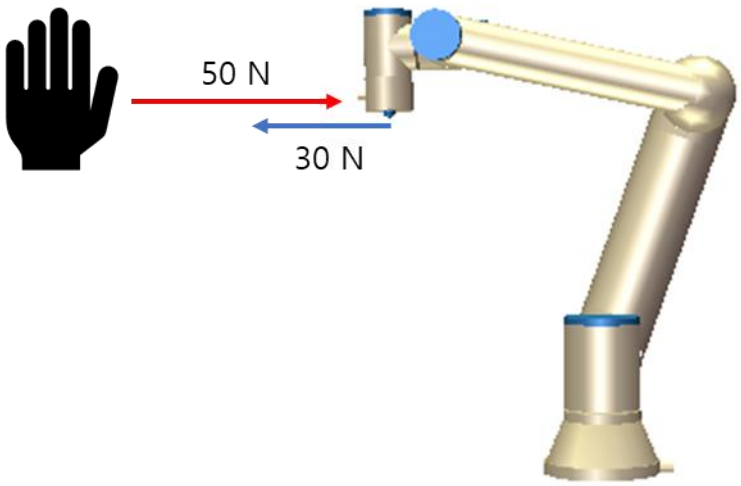


Figure 33 Compliance control

As shown in the example of Figure 4, because the target force is the primary interest of the robot, it is accepted not to reach the target position in the force control. In case when the external force is added to the robot, the robot position can be changed. In other words, the robot position can be adapted to the environment, and the force control can also be called **compliance control** due to the characteristics.

Relationship with force measurement

Some people are confused the relationship between force control and force measurement. One common misunderstanding is that the force control depends only on the force measurement. Thus, the force control should be perfect if using the precise F/T (force and torque) sensor. However, we should know that the **force control and force measurement are different**. During the force control, the force measurement will be used for the force feedback term. It is just a part and there are more processes to complete the force control. The target force and torque values in the force control are converted to the target joint torques for the joint control. In this process, the singularity affects the calculations. If the robot is near the singular area, the force control would fail due to the force singularity. If we use the **F/T sensor** in tool like e-Series, the measured force and torque values are accurate, and it can contribute to the **high quality of force control**. Though, the robot is **not free from the singularity** issue. This is the limitation of motion control for the robotic arm.

Force/Torque sensor in tool vs. Torque sensor in joint

In UR robots, the force mode is available for many different directions in the Cartesian space, but joint torque control is not available. Some robots like KUKA iiwa and Doosan offer the joint torque control. This is likely because the robots would like to show the performance of torque sensors in each joint. Many researchers want to use the joint torque control to develop their own controller as research projects. However, it is hard to find the real industrial application to require the joint torque control.

In addition, the robots with joint torque sensors have another problem. The force and torque values in tool, which is useful for the application, should be converted from the joint torque values. Like mentioned above, the singularity affects the calculation to convert between the joint torques and tool force and torque. This is bad because the singularity would affect the force measurement as well as the force control. **For industrial purposes**, we can say that **tool-oriented F/T sensor** would be **more useful**.