Individual Assignment Cover Sheet

School of Mechanical and Manufacturing Engineering

SUBMISSION DETAILS	
Course Convener: Prof. Hoang-Phuong Phan	
Course code: MTRN4230	**Course name:** Robotics
Name of this Assignment Item: Project 1	

ACADEMIC REQUIREMENTS

Before submitting this assignment, students are strongly advised to:

- Review the assessment requirements in the briefing document for the assignment from the Course Convener.
- Review the various matters related to assessment in the relevant Course Outline.
- Review the Plagiarism and Academic Integrity website at https://student.unsw.edu.au/plagiarism to ensure they are familiar with the requirements to provide appropriate acknowledgement of source materials.
- Retain a copy of this assessment for their records and in case it is misplaced, it has to be re-submitted.

If after reviewing this information there is any doubt about assessment requirements then in the first instance the student should consult with the Course Convener.

While students are generally encouraged to work with other students to enhance learning, all assignments submitted for assessment by a student must be their entire own work and they may be required to explain any or all parts of the assignment to the Course Convener or other authorised persons.

Plagiarism and Collusion are considered as Academic Misconduct and will be dealt with according to University Policy.

STUDENT DECLARATION OF ACADEMIC INTEGRITY

I declare that:

- This assessment item is entirely my own original work, except where I have acknowledged use of source material [such as books, journal articles, other published material, the Internet, and the work of other student/s or any other person/s].
- This assessment item has not been submitted for assessment for academic credit in this, or any other course, at UNSW or elsewhere.
- I have read and understood the University Rules in respect of Student Academic Misconduct.

I understand that:

- The assessor of this assessment item may, for the purpose of assessing this item, reproduce this assessment item and provide a copy to another staff member of the University.
- The assessor may communicate a copy of this assessment item to a plagiarism checking service (which may then retain a copy of the assessment item on its database for the purpose of future plagiarism checking).

STUDENT DETAILS

Student ID: 5446068	**Surname:** YU	**Given Name:** JENG-YANG
Signature of student: 余政陽	**Date of signature:** 22/7/2024	

Part A

<u>You do not need to include anything in your report for this practical part of the assessment.</u>

This part already finish and get mark during lab session.

Part B

1. Resultant homography kinematic matrices:

$$ {}^{i-1}_{i}T = \begin{bmatrix} cos\theta_i & -sin\theta_i cos\alpha_i & sin\theta_i sin\alpha_i & a_i cos\theta_i \\ sin\theta_i & cos\theta_i cos\alpha_i & -cos\theta_i sin\alpha_i & a_i sin\theta_i \\ 0 & sin\alpha_i & cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

According to the question requirements, we put the home joint configuration and the parameters in DH table into the matrix.

We can get the following result:

$$ {}^{0}_{1}T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 162.5 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

This is the transformation matrix from frame{0} to frame{1}. This matrix means rotate by 0° about z_0; translate by 0.1625m along z_0; translate by 0m along x_1; rotate by $\frac{\pi}{2}(\mathrm{rad})$ along x_1.

$$ {}^{1}_{2}T = \begin{bmatrix} 0.2588 & 0.9659 & 0 & -109.9981 \\ -0.9659 & 0.2588 & 0 & 410.5185 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

This is the transformation matrix from frame{1} to frame{2}. This matrix means rotate by -75° about z_0; translate by 0m along z_0; translate by -0.425m along x_1; rotate by $0(rad)$ along x_1.

$$ {}^{2}_{3}T = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & -392.2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

This is the transformation matrix from frame{2} to frame{3}. This matrix means rotate by 90° about z_0; translate by 0m along z_0; translate by -0.3922m along x_1;

rotate by $0(\mathrm{rad})$ along x_1.

$$ {}^{3}_{4}T = \begin{bmatrix} -0.2588 & 0 & -0.9659 & 0 \\ -0.9659 & 0 & 0.2588 & 0 \\ 0 & 1 & 0 & 133.3 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

This is the transformation matrix from frame{3} to frame{4}. This matrix means rotate by -105° about z_0; translate by 0.1333m along z_0; translate by 0m along x_1; rotate by $\frac{\pi}{2}(rad)$ along x_1.

$$ {}^{4}_{5}T = \begin{bmatrix} 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 99.7 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

This is the transformation matrix from frame{4} to frame{5}. This matrix means rotate by -90° about z_0; translate by 0.0997m along z_0; translate by 0m along x_1; rotate by $-\frac{\pi}{2}(rad)$ along x_1.

$$ {}^{5}_{6}T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 99.6 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

This is the transformation matrix from frame{5} to frame{6}. This matrix means rotate by 0° about z_0; translate by 0.096m along z_0; translate by 0m along x_1; rotate by $0(rad)$ along x_1.

Then according to the chain rule, we can get the transformation matrix from frame{0} to frame{n}.

$$ {}^{0}_{n}T = {}^{0}_{1}T * {}^{1}_{2}T * \dots * {}^{n-1}_{n}T $$

$$ {}^{0}_{2}T = \begin{bmatrix} 0.2588 & 0.9659 & 0 & -109.9981 \\ 0 & 0 & -1 & 0 \\ -0.9659 & 0.2588 & 0 & 573.0185 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

This is the transformation matrix from frame{0} to frame{2}.

$$ {}^{0}_{3}T = \begin{bmatrix} 0.9659 & -0.2588 & 0 & -488.8342 \\ 0 & 0 & -1 & 0 \\ 0.2588 & 0.9659 & 0 & 471.5096 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

This is the transformation matrix from frame{0} to frame{3}.

$$ {}_4^0T = \begin{bmatrix} 0 & 0 & -1 & -488.8342 \\ 0 & -1 & 0 & -133.3 \\ 0 & 0 & 0 & 471.5096 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

This is the transformation matrix from frame{0} to frame{4}.

$$ {}_5^0T = \begin{bmatrix} 0 & 1 & 0 & -588.5342 \\ 1 & 0 & 0 & -133.3 \\ 0 & 0 & -1 & 471.5096 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

This is the transformation matrix from frame{0} to frame{5}.

$$ {}_6^0T = \begin{bmatrix} 0 & 1 & 0 & -588.5342 \\ 1 & 0 & 0 & -133.3 \\ 0 & 0 & -1 & 371.9096 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

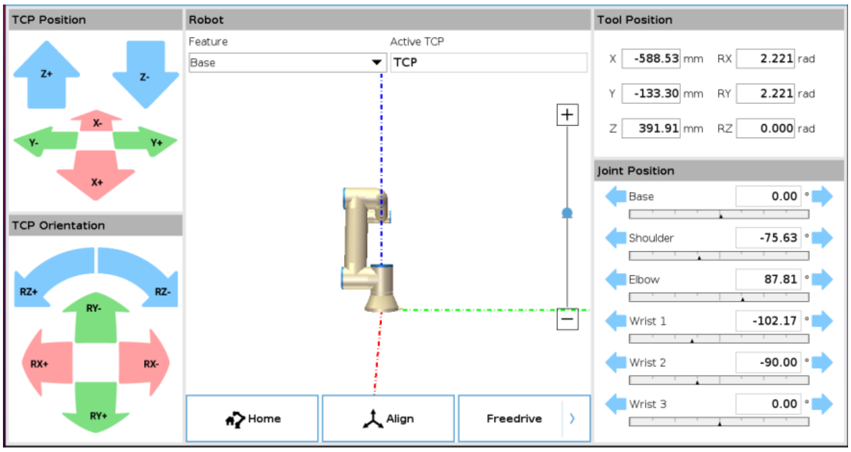
This is the transformation matrix from frame{0} to frame{6}.

These transformation matrices are crucial in robot kinematic analysis, enabling us to compute the final position and orientation of the robot's end-effector, and facilitating their application in path planning and control. Through the product of these chain transformations, we can trace and calculate the robot joints' motion paths, enabling complex motion control and simulation.

2. For this part, we use the function ***fkine*** in Matlab to attain the pose with the angles in RPY configuration. The complete code will be placed in the appendix.
   Now we get
   Pose:
   x: 0.1625m; y: 1.0165m; z: -0.1333m
   Angle:
   R: -3.1416(rad); P: 0(rad); Y: 1.5708(rad)
   Matrix result:

$$ {}_6^0T = \begin{bmatrix} 0 & 1 & 0 & -0.5885 \\ 1 & 0 & 0 & -0.1333 \\ 0 & 0 & -1 & 0.3719 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

   The unit of the final column in the matrix is the meter.
3. The screenshot showing the pose including the rotation in rpy representation is shown below.



Part C

In part C, we need to use the Jacobian matrix to calculate the robot speed limits. The path is already given, and we can use the function “movej” to get the poses, joints, joint velocities, joint accelerations, and torques.

Step 1:

Put two parameters into our **`calculateMaxLinearVelocity`** function, which are joint angles and joint velocities.

Step 2:

Jacobian matrix is the relationship of velocity between the end-effector and joint. Use the joint angles and the DH parameters of UR5e to calculate the Jacobian matrix, once we have the Jacobian matrix and the joint velocity, we can calculate the end-effector velocity.

$$\begin{bmatrix} v \\ w \end{bmatrix} = \begin{bmatrix} J_v \\ J_w \end{bmatrix} \dot{q}$$

Step 3:

After that, we can calculate the maximum linear velocity using the formula below.

$$v = \sqrt{v_x^2 + v_y^2 + v_z^2}$$

The maximum speed is around 190.0003(mm/s).

Detailed **`calculateMaxLinearVelocity`** function code is given in the appendix.

Now let’s have a look at the mathematics of Jacobian. Once we have the DH table, we can have all the quantities needed from forward kinematics.

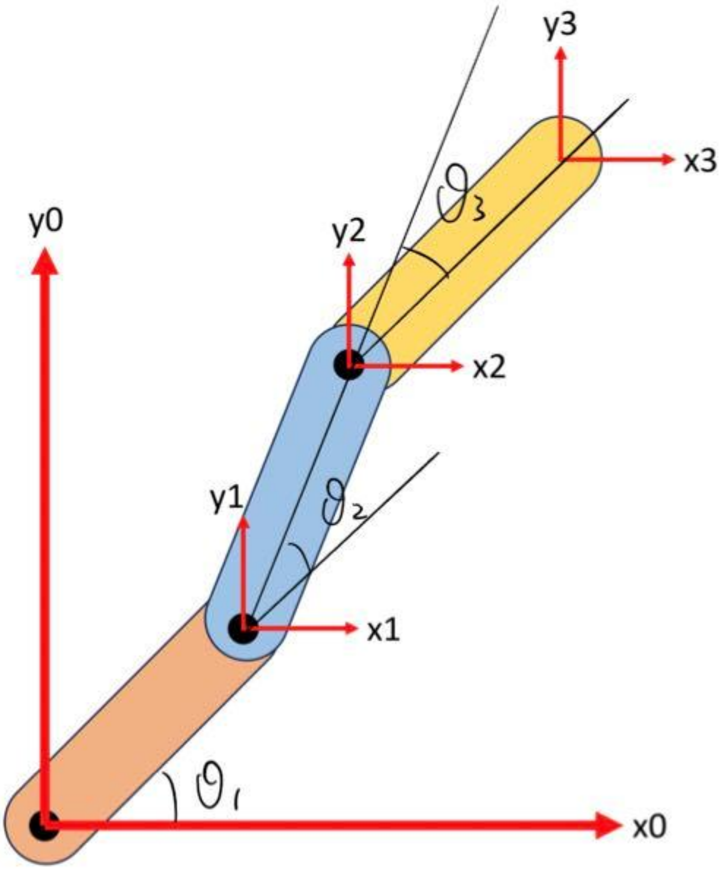
$$ {}_{i}^{0}T = \begin{bmatrix} {}_{i}^{0}R & {}_{i}^{0}o \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & o_{1,4} \\ r_{2,1} & r_{2,2} & r_{2,3} & o_{2,4} \\ r_{3,1} & r_{3,2} & r_{3,3} & o_{3,4} \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}_{n}^{0}\dot{o} = \dot{q}_i * {}_{i-1}^{0}z \times ({}_{n}^{0}o - {}_{i-1}^{0}o) $$

$$ v = {}_{n}^{0}\dot{o} $$

Part D

1. Suppose the degree in every joint is θ_1, θ_2, θ_3 respectively, as the figure shown below.



J	$\theta(deg)$	$d(m)$	$a(m)$	$\alpha(deg)$
1	θ_1	0	1	0
2	θ_2	0	1	0
3	θ_3	0	1	0

2. Calculate the Jacobian which relates joint velocities to linear velocities. Detailed code is given in the appendix.

Step 1:

Use the chain rule to get the transformation matrix. We can use function ***fkine*** in Matlab. For this problem, the calculation process is shown below.

$$ {}^{i-1}_{i}T = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 & \cos(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i) & 0 & \sin(\theta_i) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{0}_{3}T = {}^{0}_{1}T * {}^{1}_{2}T * {}^{2}_{3}T = \begin{bmatrix} {}^{0}_{i}R & {}^{0}_{i}o \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & o_{1,4} \\ r_{2,1} & r_{2,2} & r_{2,3} & o_{2,4} \\ r_{3,1} & r_{3,2} & r_{3,3} & o_{3,4} \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ = \begin{bmatrix} \cos(\theta_1+\theta_2+\theta_3) & -\sin(\theta_1+\theta_2+\theta_3) & 0 & \cos(\theta_1+\theta_2+\theta_3)+\cos(\theta_1+\theta_2)+\cos(\theta_1) \\ \sin(\theta_1+\theta_2+\theta_3) & \cos(\theta_1+\theta_2+\theta_3) & 0 & \sin(\theta_1+\theta_2+\theta_3)+\sin(\theta_1+\theta_2)+\sin(\theta_1) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

And we want to get the matrix that relates joint velocities and linear velocities so that we can calculate and get the following matrix.

$$ \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} \cos(\theta_1+\theta_2+\theta_3)+\cos(\theta_1+\theta_2)+\cos(\theta_1) \\ \sin(\theta_1+\theta_2+\theta_3)+\sin(\theta_1+\theta_2)+\sin(\theta_1) \\ \theta_1+\theta_2+\theta_3 \end{bmatrix} $$

Step 2:

Calculate the Jacobian matrix.

$$ J = \begin{bmatrix} J_v \\ J_w \end{bmatrix} = [J_1, J_2, J_3] $$

$$ J_i = \begin{bmatrix} {}^{0}_{i-1}z \times ({}^{0}_{n}o - {}^{0}_{i-1}o) \\ {}^{0}_{i-1}z \end{bmatrix}, n = 3 $$

$$ [J_1, J_2, J_3] = \begin{bmatrix} {}^{0}_{0}z \times ({}^{0}_{3}o - {}^{0}_{0}o) & {}^{0}_{1}z \times ({}^{0}_{3}o - {}^{0}_{1}o) & {}^{0}_{2}z \times ({}^{0}_{3}o - {}^{0}_{2}o) \\ {}^{0}_{0}z & {}^{0}_{1}z & {}^{0}_{2}z \end{bmatrix} $$

$$ {}^{0}_{0}z = {}^{0}_{1}z = {}^{0}_{2}z = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} $$

And we can delete the bottom 3 rows of the Jacobian, which is z, since this is three-link robot arm and we cannot control movement in the z-direction, or any of the three end effector rotations. We can use function ***jacob0*** in Matlab.

$$ \begin{bmatrix} \partial x \\ \partial y \\ \partial \theta \end{bmatrix} = J * \begin{bmatrix} \partial \theta_1 \\ \partial \theta_2 \\ \partial \theta_3 \end{bmatrix} $$

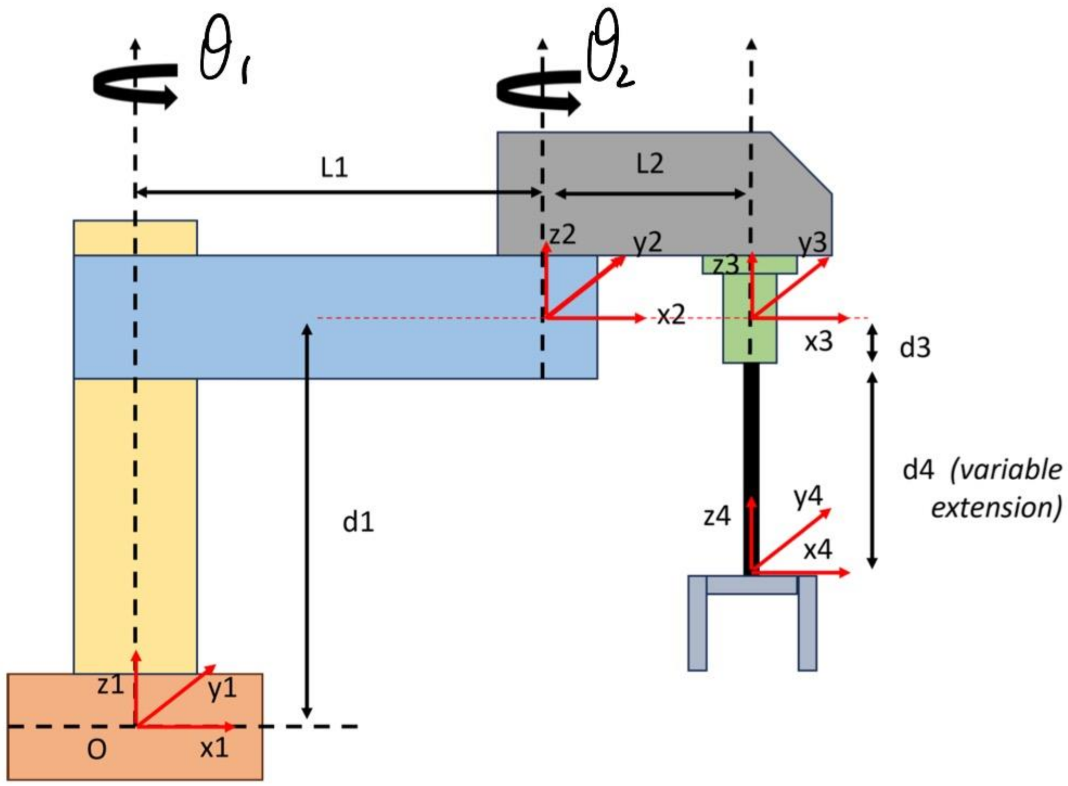
J

$$= \begin{bmatrix} -\sin(\theta_1+\theta_2+\theta_3)-\sin(\theta_1+\theta_2)-\sin(\theta_1) & -\sin(\theta_1+\theta_2+\theta_3)-\sin(\theta_1+\theta_2) & -\sin(\theta_1+\theta_2+\theta_3) \\ \cos(\theta_1+\theta_2+\theta_3)+\cos(\theta_1+\theta_2)+\cos(\theta_1) & \cos(\theta_1+\theta_2+\theta_3)+\cos(\theta_1+\theta_2) & \cos(\theta_1+\theta_2+\theta_3) \\ 1 & 1 & 1 \end{bmatrix}$$

Now we get the Jacobian matrix which relates joint velocities to linear velocities.

3. Singularity occurs when the determinant of the Jacobian matrix is zero. A robot configuration from which certain motions become unattainable. A certain configuration q is said to be singular if $\det(J(q)) = 0$. The robot may move very fast or lose some DOFs. There may be no solution or an infinite number or solutions.
   The determinant of the Jacobian matrix is $\sin(\theta_2)$. Detailed code is given in the appendix. If $\sin(\theta_2)$ is 0, then the manipulator is at a singularity, which means θ_2 is 0 or π.
   This is called elbow singularity. The singularity occurs when $\theta_2 = n\pi$, which means that the second joint (middle link) is either fully extended or fully folded back. This will loss of control in certain directions and reduce manipulability.

Part E



1. Calculate the analytical (algebraic) inverse kinematic solution

Step 1:

Get the DH table and the transformation matrix.

J	$\theta(deg)$	$d(m)$	$a(m)$	$\alpha(deg)$
1	θ_1	$d1$	$L1$	0
2	θ_2	0	$L2$	0
3	0	$d3 + d4$	0	0

$$
{}_{i}^{i-1}T = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i)\cos(\alpha_i) & \sin(\theta_i)\sin(\alpha_i) & a_i\cos(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i)\cos(\alpha_i) & -\cos(\theta_i)\sin(\alpha_i) & a_i\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}_{1}^{0}T = \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 & L_1\cos(\theta_1) \\ \sin(\theta_1) & \cos(\theta_1) & 0 & L_1\sin(\theta_1) \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}_{2}^{1}T = \begin{bmatrix} \cos(\theta_2) & -\sin(\theta_2) & 0 & L_2\cos(\theta_2) \\ \sin(\theta_2) & \cos(\theta_2) & 0 & L_2\sin(\theta_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}_{3}^{2}T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

The transformation matrix can be calculated via the chain rule.

$$
{}_{3}^{0}T = \begin{bmatrix} \cos(\theta_1+\theta_2) & -\sin(\theta_1+\theta_2) & 0 & L_1\cos(\theta_1) + L_2\cos(\theta_1+\theta_2) \\ \sin(\theta_1+\theta_2) & \cos(\theta_1+\theta_2) & 0 & L_1\sin(\theta_1) + L_2\sin(\theta_1+\theta_2) \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Step 2:

Now we can calculate the inverse kinematics.

$$x = L_1\cos(\theta_1) + L_2\cos(\theta_1+\theta_2)$$

$$y = L_1\sin(\theta_1) + L_2\sin(\theta_1+\theta_2)$$

Suppose the distance from base to the end-effector is r. $r = \sqrt{x^2 + y^2}$.

Using the Law of Cosines to solve for θ_2.

$$\cos(\theta_2) = \frac{r^2 - L_1^2 - L_2^2}{2L_1L_2}$$

$$\sin(\theta_2) = \sqrt{1 - cos^2(\theta_2)}$$

$$\theta_2 = atan2(\sin(\theta_2), \cos(\theta_2))$$

Now calculate θ_1.

$$\theta_1 = atan2(\mathrm{x}, \mathrm{y}) - atan2(L_2 \sin(\theta_2), L_1 + L_2 \cos(\theta_2))$$

In conclusion, we can get the joint rotation angles now.

$\theta_1 = atan2(\mathrm{x}, \mathrm{y}) - atan2(L_2 \sin(\theta_2), L_1 + L_2 \cos(\theta_2))$

$\theta_2 = atan2(\sin(\theta_2), \cos(\theta_2))$

2. For more complex robots such as the UR5E, the inverse kinematics equations can be very complex and may result in multiple solutions or no solution at all. Often, there are nonlinear components in the robot's kinematic model that make it difficult or impossible to solve analytically (algebraically).
Two alternative methods could be undertaken.
Optimization-Based Methods:
Based on optimization methods, set up an optimization problem with the goal of minimizing the objective function (the error between the end-effector position and the desired position) in order to find the inverse kinematics solution.
Step1: Define the objective function that typically represents the error between the real position of end-effector and the target position.
Step2: Set constraints, such as kinematic constraints, joint limits and range of motions.
Step3: Use optimization algorithms to minimize the objective function in order to find the optimal joint angle solution.
Advantages: Can solve multiple complex constraints and optimize the objectives. And capable of finding global optimal solutions (with appropriate optimization algorithms).
Disadvantages: High computational complexity, which may require a longer time to solve. And sensitive to the choice of initial guesses; the initial solution can impact the final result.

Learning-Based Methods:
Based on learning method such as machine learning or deep learning techniques to learn and predict the solution of inverse kinematics. These methods train the model via large amount of training data to achieve efficient inverse kinematics solving.
Step1: Collect the end-effector position data and joint angles data.

Step2: Use the data collected to train the machine learning model (such as neural network) so that it can predict the joint angles from the given end-effector position.
Step3: In practical use, put the new data into the trained model to get the solution of inverse kinematics.
Advantages: The trained model can compute in real-time, making it fast. And it also can handle complex nonlinear problems and high-dimensional data.
Disadvantages: Requires a large amount of training data and computational resources. The accuracy of the model depends on the quality and quantity of the training data. So the data processing and noise filtering methods need to be used.

Appendix

Part B

```
startup_rvc;
dh = [
    0   0.1625   0       pi/2;
    0   0       -0.425      0;
    0   0       -0.3922     0;
    0   0.1333       0  pi/2;
    0   0.0997       0  -pi/2;
    0   0.0996       0  0;
];
UR5e = SerialLink(dh, 'name', 'UR5e');
q = [0, -75.00, 90.00, -105.00, -90.00, 0.00];
q = deg2rad(q);
T = UR5e.fkine(q);
disp('The transformation matrix of end effector:');
disp(T);
position = T.t;
rpy_angles = tr2rpy(T);
disp('The position of end effector:');
disp(position);
disp('The RPY angle of end effector:');
disp(rpy_angles);
```

Part C- `calculateMaxLinearVelocity`

```
% calculateMaxLinearVelocity.m
% MTRN4230 Assignment 1 24T2
% Name: JENG-YANG YU
% Zid:  z5446068
```

```
%% Function you must complete
% You must implement the following function
function maxLinearVelocity =
calculateMaxLinearVelocity(jointPositions,jointVelocities)
dh = [
    0   162.5   0       pi/2;
    0   0       -425      0;
    0   0       -392.2    0;
    0   133.3    0      pi/2;
    0   99.7     0     -pi/2;
    0   99.6     0        0;
];
UR5e = SerialLink(dh, 'name', 'UR5e');
% Write your implementation here
maxLinearVelocity = 0;
% Initialize variables
v_max = 0;
% Assuming joints and jointVelocities are matrices with rows corresponding
to time steps
for i = 1:size(jointPositions, 1)
    % Get current joint positions and velocities for this time step
    q = jointPositions(i, :);
    q_dot = jointVelocities(i, :);
    % Calculate Jacobian matrix at this joint configuration
    % Replace with your actual Jacobian calculation method
    J = UR5e.jacob0(q);
    % Calculate end-effector linear velocity: v = J * q_dot'
    v_end_effector = J * q_dot';
    % Calculate magnitude of linear velocity
    v_mag = norm(v_end_effector(1:3));  % Consider only linear velocity
    % Update maximum velocity if current velocity is larger
    if v_mag > v_max
       v_max = v_mag;
    end
end
    maxLinearVelocity = v_max;
end
```

Part D

```
startup_rvc;
syms q1 q2 q3;
dh = [
    0   0   1     0;
```

```
    0   0   1     0;
    0   0   1     0;

];
q = [q1 q2 q3]
UR5e = SerialLink(dh, 'name', 'UR5e');
%J = UR5e.jacob0(q)

% syms q1 q2 q3
% expr = cos(q1 + q2 + q3)*(cos(q2 + q3) + cos(q3) + 1) + sin(q1 + q2 +
q3)*(sin(q2 + q3) + sin(q3));
% simplified_expr = simplify(expr);
% disp(simplified_expr);

J = [-sin(q1+q2+q3)-sin(q1+q2)-sin(q1) -sin(q1+q2+q3)-sin(q1+q2) -
sin(q1+q2+q3); %after simplified
    cos(q1+q2+q3)+cos(q1+q2)+cos(q1) cos(q1+q2+q3)+cos(q1+q2)
cos(q1+q2+q3);
    1 1 1]
T = UR5e.fkine(q) %transformation matrix
detj = det(J);
det_J_simplified = simplify(detj);
```