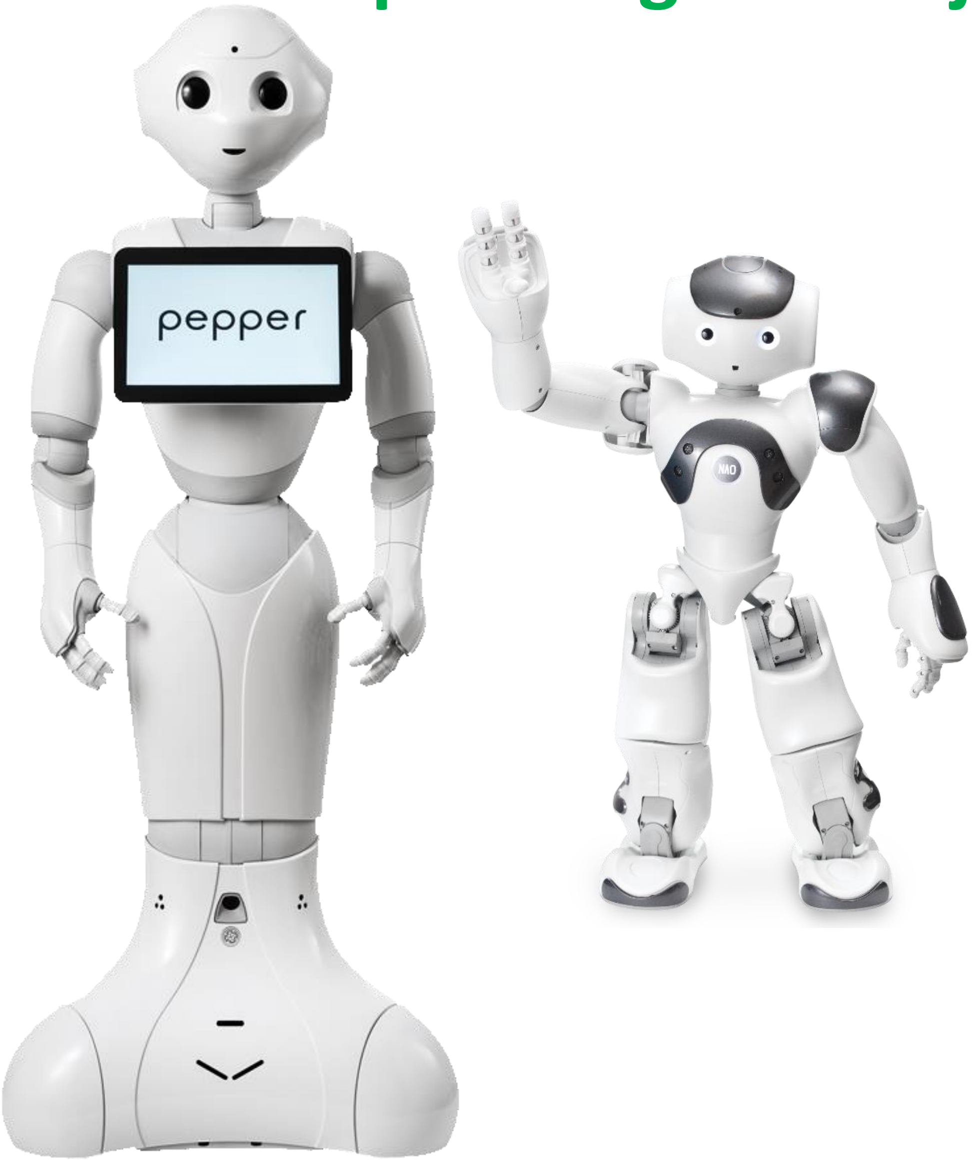
Fundamentals of Robotics

Robot Manipulators 08

Path planning and trajectory generation



Controlling path of an end effector

Path planning

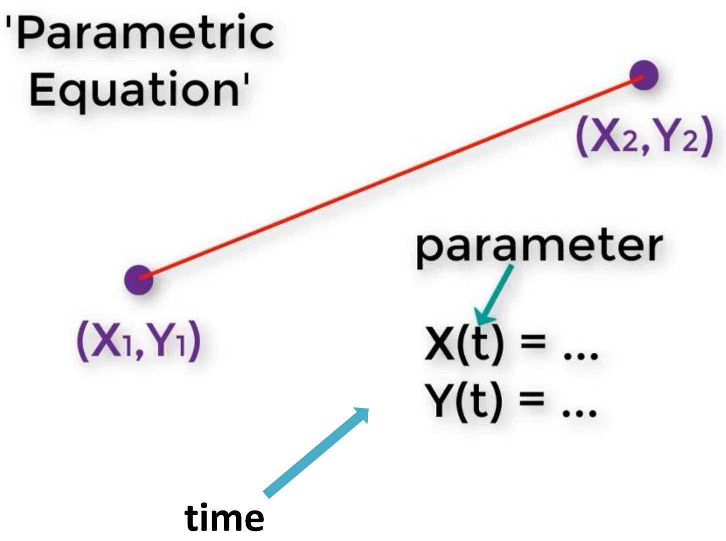
Trajectory generation

Controlling the path of an end effector

- Till now we have focused on finding the position and orientation of an end effector. However, there are many cases when the path taken by end effector is very important.
- Car assembling, cutting, welding, spraying highly depends on the path followed by an end effector.
- The process of controlling the path of an end effector is broken into two parts:
- **Path planning**: Figure out the points in space through which the end effector will pass.
- Once we figure out how to make an end effector to follow a path then the second important thing is the speed with which it reaches second point.
- **Trajectory generation**: Figure out the velocity component of the end effector motion along path (Speed + direction).

Path planning

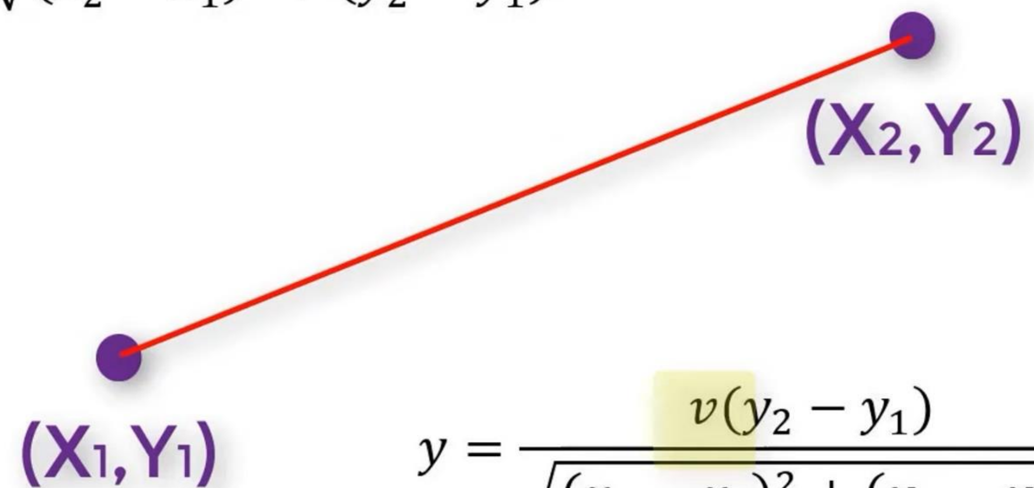
- First, we plan path and then we generate trajectory.
- For path planning we use a path planning equations called **Parametric equations**.
- Parametric equation **defines the points** on a path **relative to a parameter**.
- A **parameter** can be of different kinds. However, we would use **time** as parameter with parametric equation.
- **X(t)** gives a value of x in any point in time.
- **Y(t)** gives a value of y in any point in time.



Path planning

- Here, t is current time
- Whereas v is current velocity of the end effector.
- These equations are used to find the value of x and at any point in time.

$$x = \frac{v(x_2 - x_1)}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}} t + x_1$$

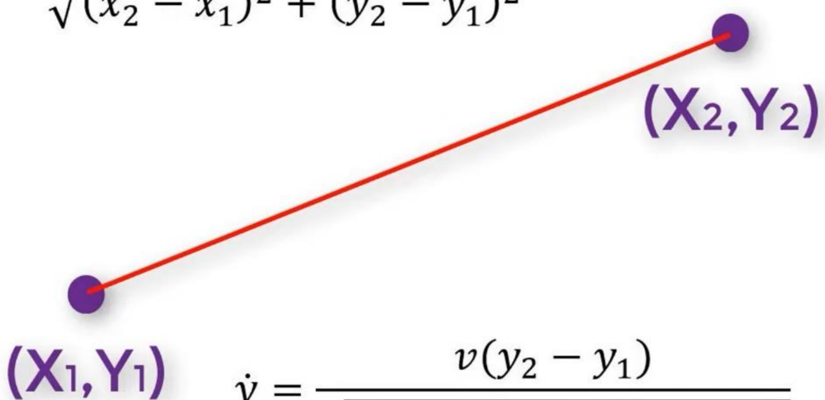


$$y = \frac{v(y_2 - y_1)}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}} t + y_1$$

Path planning

- Now, we have parametric equations for the path we want an end effector to follow.
- Its time to perform trajectory generation.
- In order, to do trajectory generation we take the time derivative of x and y to get $\dot{x}$ and $\dot{y}$.

$$\dot{x} = \frac{v(x_2 - x_1)}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}$$



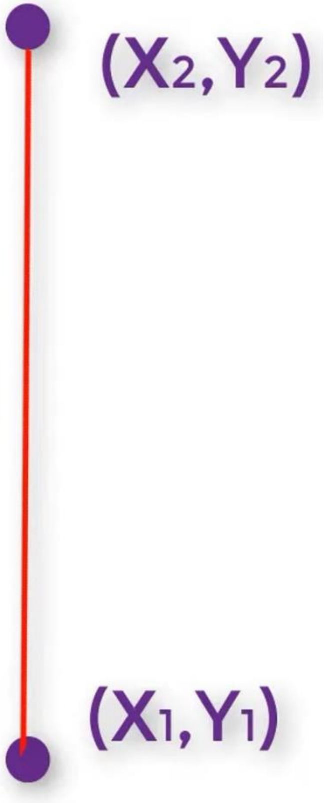
$$\dot{y} = \frac{v(y_2 - y_1)}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}$$

Path planning

- Here, the end effector is moving in a vertical straight line.

$$\dot{x} = \frac{v\cancelto{0}{(x_2 - x_1)}}{\sqrt{\cancelto{0}{(x_2 - x_1)^2} + (y_2 - y_1)^2}} = 0$$

$$\dot{y} = \frac{v(y_2 - y_1)}{\sqrt{\cancelto{0}{(x_2 - x_1)^2} + (y_2 - y_1)^2}} = v$$



Path planning

- Here, the end effector is moving in a horizontal straight line.
- We can use these two equations to get the end effector to travel in any line defined by two points (x1,y1) and (x2, y2).

$$\dot{x} = \frac{v(x_2 - x_1)}{\sqrt{(x_2 - x_1)^2 + \cancelto{0}{(y_2 - y_1)^2}}}$$

$$\dot{y} = \frac{v\cancelto{0}{(y_2 - y_1)}}{\sqrt{(x_2 - x_1)^2 + \cancelto{0}{(y_2 - y_1)^2}}}$$

Inverse Jacobian matrix

- In the previous section, we looked at how to calculate the velocities of the end effector of a robotic arm given the joint velocities. What if we want to do the reverse? We want to calculate the joint velocities given desired velocities of the end effector?

Calculate the end-effector velocities given the joint velocities

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = J \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \vdots \\ \dot{q}_n \end{bmatrix}$$

Calculate the joint velocities given the end-effector velocities

Inverse Jacobian matrix

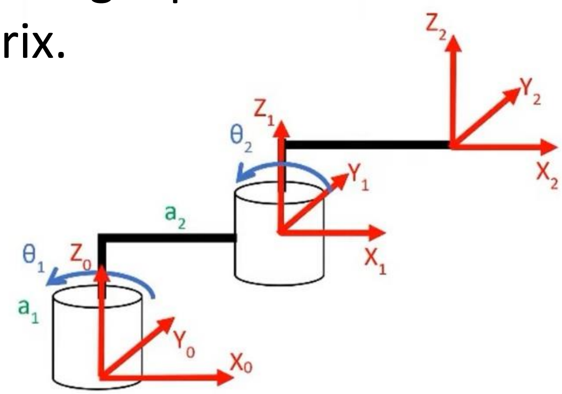
- To solve this problem, we must use the **inverse** of the Jacobian matrix.
- A matrix multiplied by its inverse is the **identity** matrix I.
- The identity matrix is the matrix version of the number 1.

$$A^{-1}A = I$$

$$J^{-1}\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = J^{-1}J\begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \vdots \\ \dot{q}_n \end{bmatrix} \Rightarrow \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \vdots \\ \dot{q}_n \end{bmatrix} = J^{-1}\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$$

Inverse Jacobian matrix

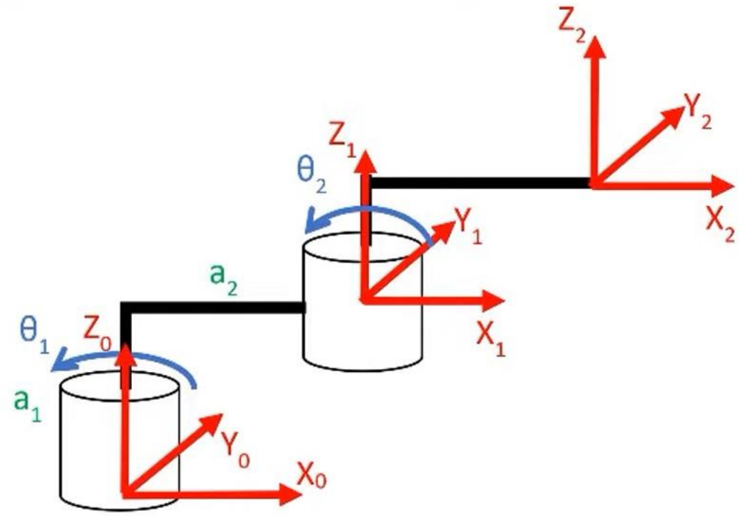
- You can only take the inverse of a **square matrix**. A square matrix is a matrix where the number of rows is equal to the number of columns.
- Suppose we have the following **two degrees of freedom** robotic arm.
- We have the following equation where the matrix with the **12 squares** is J, the Jacobian matrix.



$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} \square & \square \\ \square & \square \\ \square & \square \\ \square & \square \\ \square & \square \\ \square & \square \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}$$

Inverse Jacobian matrix

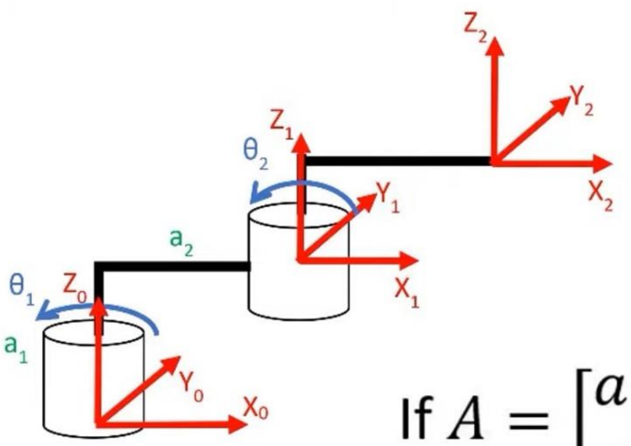
- We only have **two servo motors**. These two servo motors **control the velocity** of the end effector in only the x and y directions (e.g., we have no motion in the z direction).
- Suppose the only thing that matters to us is the **linear velocity** in the x direction and the linear velocity in the y direction. We can simplify our equation accordingly to this, where the matrix with the squares is J:



$$\begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix} = \begin{bmatrix}\square & \square\\ \square & \square\end{bmatrix}\begin{bmatrix}\dot{\theta}_1\\ \dot{\theta}_2\end{bmatrix}$$

Inverse Jacobian matrix

- To get the A^{-1}, we use the following formula:



$$\text{If } A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \text{ then } A^{-1} = \frac{1}{ad-bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} \square & \square \\ \square & \square \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}$$

Inverse Jacobian matrix

$$
\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}
=
\begin{bmatrix}
\overset{\mathbf{J}_{11}}{-a_4 S\theta_1 C\theta_2 - a_4 C\theta_1 S\theta_2 - a_2 S\theta_1} & \overset{\mathbf{J}_{12}}{-a_4 S\theta_1 C\theta_2 - a_4 C\theta_1 S\theta_2} \\
a_4 C\theta_1 C\theta_2 - a_4 S\theta_1 S\theta_2 + a_2 C\theta_1 & a_4 C\theta_1 C\theta_2 - a_4 S\theta_1 S\theta_2 \\
0 & 0 \\
\underset{\mathbf{J}_{21}}{R_0^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}} & \underset{\mathbf{J}_{22}}{R_1^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}}
\end{bmatrix}
\begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}
$$

Inverse Jacobian matrix

- J is that big matrix above. Since we are only concerned about the linear velocities in the x and y directions, this:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}$$

$$\begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} = J^{-1} \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix}$$

$$J^{-1} = \frac{1}{J_{11}J_{22} - J_{12}J_{21}} \begin{bmatrix} J_{22} & -J_{12} \\ -J_{21} & J_{11} \end{bmatrix}$$

Inverse Jacobian matrix

- Final equation

$$\begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} J_{11}^{-1} & J_{12}^{-1} \\ J_{21}^{-1} & J_{22}^{-1} \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix}$$

$$\dot{\theta}_1 = {J_{11}}^{-1}\dot{x} + {J_{12}}^{-1}\dot{y}$$

$$\dot{\theta}_2 = {J_{21}}^{-1}\dot{x} + {J_{22}}^{-1}\dot{y}$$

```
#include <VarSpeedServo.h>
// Define the number of servos
#define SERVOS 2
// Conversion factor from degrees to radians
#define DEG_TO_RAD 0.017453292519943295769236907684886
// Conversion factor from radians to degrees
#define RAD_TO_DEG 57.295779513082320876798154814105
// Create the servo objects.
VarSpeedServo myservo[SERVOS];
// Speed of the servo motors
// Speed=1: Slowest
// Speed=255: Fastest.
const int default_speed = 255;
const int std_delay = 10; // Delay in milliseconds
// Attach servos to digital pins on the Arduino
int servo_pins[SERVOS] = {3,5};
// Angle of the first servo
float theta_1 = 0;
float theta_1_increment = 0;
float theta_1_dot = 0; // rotational velocity of the first servo
// Angle of the second servo
float theta_2 = 0;
float theta_2_increment = 0;
float theta_2_dot = 0; // rotational velocity of the second servo
// Linear velocities of the end effector relative to the base frame
// Units are in centimeters per second
// If x_dot = 0.0, the end effector will move parallel to the y axis
// Play around with these numbers, and observe the motion of the end effector
// relative to the x and y axes of the base frame of the robotic arm.
float x_dot = 0.0;
float y_dot = 1.0;
// Jacobian variables
float reciprocal_of_the_determinant;
float J11;
float J12;
float J21;
float J22;
// Inverse Jacobian variables
float J11_inv;
float J12_inv;
float J21_inv;
float J22_inv;

// Link lengths in centimeters
// You measure these values using a ruler and the kinematic diagram
float a2 = 5.9;
float a4 = 6.0;

void setup() {

  Serial.begin(9600);

  // Attach the servos to the servo object
  // attach(pin, min, max  ) - Attaches to a pin
  //   setting min and max values in microseconds
  //   default min is 544, max is 2400
  // Alter these numbers until both servos have a
  //   180 degree range.
  myservo[0].attach(servo_pins[0], 544, 2475);
  myservo[1].attach(servo_pins[1], 500, 2475);

  // Set the angle of the first servo.
  theta_1 = 0.0;

  // Set the angle of the second servo.
  theta_2 = 90.0;

  // Set initial servo positions
  myservo[0].write(theta_1, default_speed, true);
  myservo[1].write(theta_2, default_speed, true);

  // Let servos get into position
  delay(3000);

}
```

```
void loop() {

  // Make sure the servos stay within their 180 degree range
  while (theta_1 <= 180.0 && theta_1 >= 0.0 && theta_2 <= 180.0 && theta_2
>= 0.0) {

    // Convert from degrees to radians
    theta_1 = theta_1 * DEG_TO_RAD;
    theta_2 = theta_2 * DEG_TO_RAD;

    // Calculate the values of the Jacobian matrix
    J11 = -a4 * sin(theta_1) * cos(theta_2) - a4 * cos(theta_1) * sin(theta_2) - a2
* sin(theta_1);
    J12 = -a4 * sin(theta_1) * cos(theta_2) - a4 * cos(theta_1) * sin(theta_2);
    J21 = a4 * cos(theta_1) * cos(theta_2) - a4 * sin(theta_1) * sin(theta_2) + a2
* cos(theta_1);
    J22 = a4 * cos(theta_1) * cos(theta_2) - a4 * sin(theta_1) * sin(theta_2);

    reciprocal_of_the_determinant = 1.0/((J11 * J22) - (J12 * J21));

    // Calculate the values of the inverse Jacobian matrix
    J11_inv = reciprocal_of_the_determinant * (J22);
    J12_inv = reciprocal_of_the_determinant * (-J12);
    J21_inv = reciprocal_of_the_determinant * (-J21);
    J22_inv = reciprocal_of_the_determinant * (J11);

    // Set the rotational velocity of the first servo
    theta_1_dot = J11_inv * x_dot + J12_inv * y_dot;

    // Set the rotational velocity of the second servo
    theta_2_dot = J21_inv * x_dot + J22_inv * y_dot;

 // Convert rotational velocity in radians per second to X radians in std_delay
milliseconds
    // Note that 1 second = 1000 milliseconds and each delay is std_delay
milliseconds
    theta_1_increment = (theta_1_dot) * (1/1000.0) * std_delay;

    // Convert rotational velocity in radians per second to X radians in std_delay
milliseconds
    // Note that 1 second = 1000 milliseconds and each delay is std_delay
milliseconds
    theta_2_increment = (theta_2_dot) * (1/1000.0) * std_delay;

    theta_1 = theta_1 + theta_1_increment;
    theta_2 = theta_2 + theta_2_increment;

    // Convert the new angles from radians to degrees
    theta_1 = theta_1 * RAD_TO_DEG;
    theta_2 = theta_2 * RAD_TO_DEG;

    Serial.println(theta_1);
    Serial.println(theta_2);
    Serial.println(" ");

    myservo[0].write(theta_1, default_speed, true);
    myservo[1].write(theta_2, default_speed, true);

    delay(std_delay); // Delay in milliseconds
  }
}
```

End