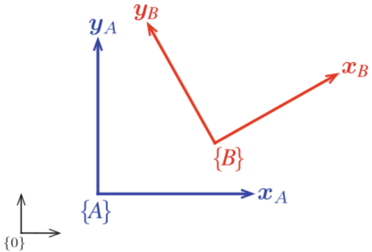
6ELEN018W - Applied Robotics
Lecture 3: Robot Motion - 2D Velocity Kinematics

Dr Dimitris C. Dracopoulos

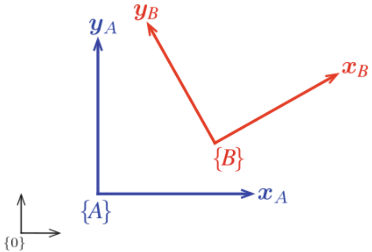
Previously - Homogeneous Transformations Matrices

2D case:



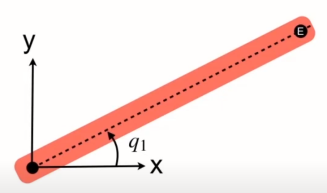
Previously - Homogeneous Transformations Matrices

2D case:

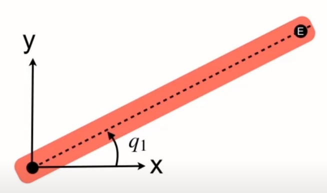


$$\begin{pmatrix} A_x \\ A_y \\ 1 \end{pmatrix} = \begin{pmatrix} cos(\theta) & -sin(\theta) & t_x \\ sin(\theta) & cos(\theta) & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} {}^B x \\ {}^B y \\ 1 \end{pmatrix} \quad (1)$$

Pose of the End-Effector - 1-Joint 2D Robot Arm

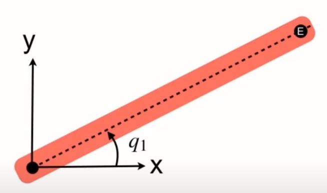


Pose of the End-Effector - 1-Joint 2D Robot Arm



$$E = \boldsymbol{R}(q_1)$$

Pose of the End-Effector - 1-Joint 2D Robot Arm



$$E = \boldsymbol{R}(q_1) \cdot \boldsymbol{T}_x(a_1)$$

$$E = \begin{pmatrix} cos(q_1) & -sin(q_1) & 0 \\ sin(q_1) & cos(q_1) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & a_1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} cos(q_1) & -sin(q_1) & a_1 cos(q_1) \\ sin(q_1) & cos(q_1) & a_1 sin(q_1) \\ 0 & 0 & 1 \end{pmatrix}$$

Pose of the End-Effector - 1-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

Pose of the End-Effector - 1-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

```
>>> from sympy import *
```

Pose of the End-Effector - 1-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

```
>>> from sympy import *

>>> q1 = Symbol('q1')
```

Pose of the End-Effector - 1-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

```
>>> from sympy import *

>>> q1 = Symbol('q1')

>>> trot2(q1)
```

Pose of the End-Effector - 1-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

```
>>> from sympy import *

>>> q1 = Symbol('q1')

>>> trot2(q1)

>>> a1=Symbol('a1')
```

Pose of the End-Effector - 1-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

```
>>> from sympy import *

>>> q1 = Symbol('q1')

>>> trot2(q1)

>>> a1=Symbol('a1')

>>> transl2(a1,0)
```

Pose of the End-Effector - 1-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

```
>>> from sympy import *

>>> q1 = Symbol('q1')

>>> trot2(q1)

>>> a1=Symbol('a1')

>>> transl2(a1,0)

>>> E = trot2(q1) @ transl2(a1, 0)
```

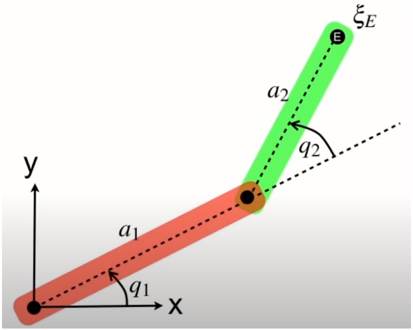
Pose of the End-Effector - 1-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

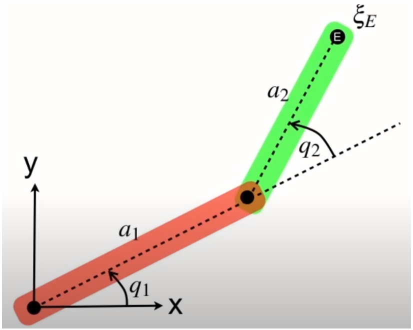
```
>>> from sympy import *

>>> q1 = Symbol('q1')

>>> trot2(q1)

>>> a1=Symbol('a1')

>>> transl2(a1,0)

>>> E = trot2(q1) @ transl2(a1, 0)
```

Demo of 1-joint arm shown in the class (see video recording)

Pose of the End-Effector - 2-Joint 2D Robot Arm

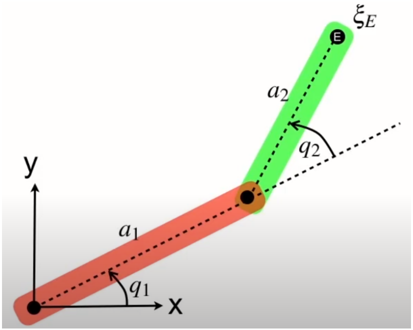


Pose of the End-Effector - 2-Joint 2D Robot Arm



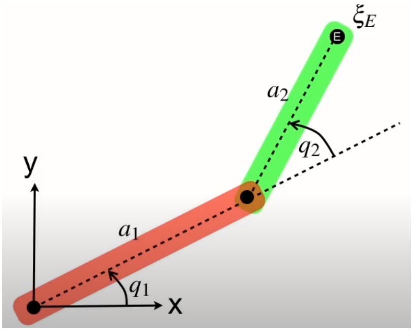
$$E = \boldsymbol{R}(q_1)$$

Pose of the End-Effector - 2-Joint 2D Robot Arm



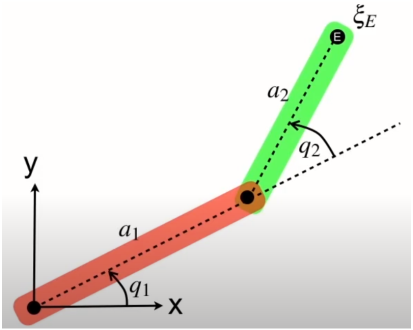
$$E = \boldsymbol{R}(q_1) \cdot \boldsymbol{T}_x(a_1)$$

Pose of the End-Effector - 2-Joint 2D Robot Arm



$$E = \boldsymbol{R}(q_1) \cdot \boldsymbol{T}_x(a_1) \cdot \boldsymbol{R}(q_2)$$

Pose of the End-Effector - 2-Joint 2D Robot Arm



$$E = \boldsymbol{R}(q_1) \cdot \boldsymbol{T}_x(a_1) \cdot \boldsymbol{R}(q_2) \cdot \boldsymbol{T}_x(a_2)$$

$$E = \begin{pmatrix} cos(q_1 + q_2)) & -sin(q_1 + q_2) & a_1 cos(q_1) + a_2 cos(q_1 + q_2) \\ sin(q_1 + q_2) & cos(q_1 + q_2) & a_1 sin(q_1) + a_2 sin(q_1 + q_2) \\ 0 & 0 & 1 \end{pmatrix}$$

Pose of the End-Effector - 2-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

Pose of the End-Effector - 2-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

```
>>> from sympy import *
```

Pose of the End-Effector - 2-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

```
>>> from sympy import *

>>> q1 = Symbol('q1')
```

Pose of the End-Effector - 2-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

```
>>> from sympy import *

>>> q1 = Symbol('q1')

>>> trot2(q1)
```

Pose of the End-Effector - 2-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

```
>>> from sympy import *

>>> q1 = Symbol('q1')

>>> trot2(q1)

>>> a1=Symbol('a1')
```

Pose of the End-Effector - 2-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

```
>>> from sympy import *

>>> q1 = Symbol('q1')

>>> trot2(q1)

>>> a1=Symbol('a1')

>>> transl2(a1,0)
```

Pose of the End-Effector - 2-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

```
>>> from sympy import *

>>> q1 = Symbol('q1')

>>> trot2(q1)

>>> a1=Symbol('a1')

>>> transl2(a1,0)

>>> q2 = Symbol('q2')
```

Pose of the End-Effector - 2-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

```
>>> from sympy import *

>>> q1 = Symbol('q1')

>>> trot2(q1)

>>> a1=Symbol('a1')

>>> transl2(a1,0)

>>> q2 = Symbol('q2')

>>> a2 = Symbol('a2')
```

Pose of the End-Effector - 2-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

```
>>> from sympy import *

>>> q1 = Symbol('q1')

>>> trot2(q1)

>>> a1=Symbol('a1')

>>> transl2(a1,0)

>>> q2 = Symbol('q2')

>>> a2 = Symbol('a2')

>>> E = trot2(q1) @ transl2(a1, 0) @ trot2(q2) @ transl2(a2, 0)
```

Pose of the End-Effector - 2-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

```
>>> from sympy import *

>>> q1 = Symbol('q1')

>>> trot2(q1)

>>> a1=Symbol('a1')

>>> transl2(a1,0)

>>> q2 = Symbol('q2')

>>> a2 = Symbol('a2')

>>> E = trot2(q1) @ transl2(a1, 0) @ trot2(q2) @ transl2(a2, 0)

E = simplify(E)
```

Pose of the End-Effector - 2-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

```
>>> from sympy import *

>>> q1 = Symbol('q1')

>>> trot2(q1)

>>> a1=Symbol('a1')

>>> transl2(a1,0)

>>> q2 = Symbol('q2')

>>> a2 = Symbol('a2')

>>> E = trot2(q1) @ transl2(a1, 0) @ trot2(q2) @ transl2(a2, 0)

E = simplify(E)
```

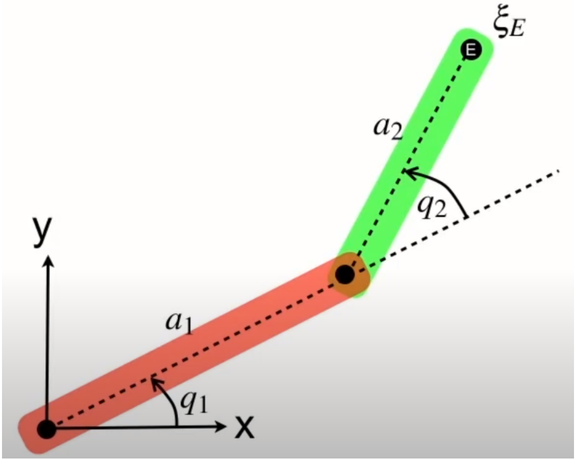
Demo of 2-joint arm shown in the class (see video recording)

Pose of the End-Effector - 2-Joint 2D Robot Arm (cont'd)

The configuration for a pose of the end-effector of the 2-joint robot arm is not unique:

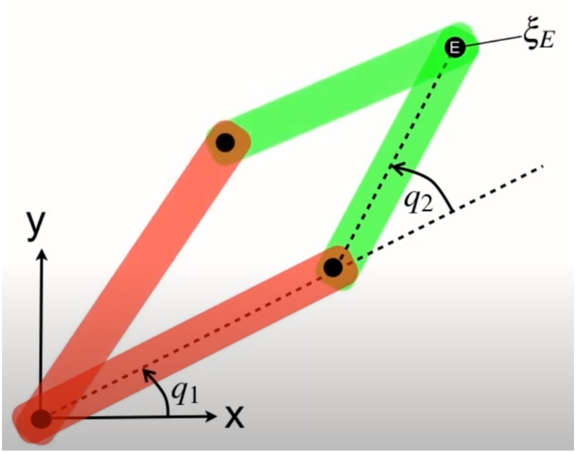
Pose of the End-Effector - 2-Joint 2D Robot Arm (cont'd)

The configuration for a pose of the end-effector of the 2-joint robot arm is not unique:

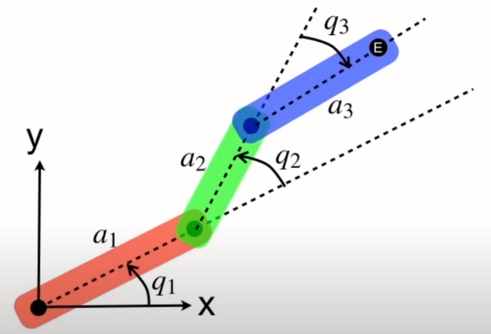


Pose of the End-Effector - 2-Joint 2D Robot Arm (cont'd)

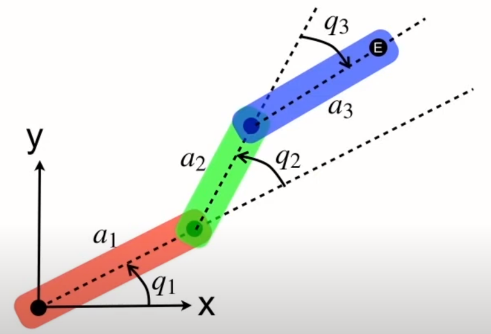
The configuration for a pose of the end-effector of the 2-joint robot arm is not unique:



Pose of the End-Effector - 3-Joint 2D Robot Arm

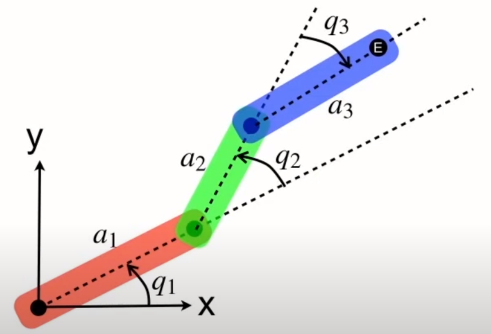


Pose of the End-Effector - 3-Joint 2D Robot Arm



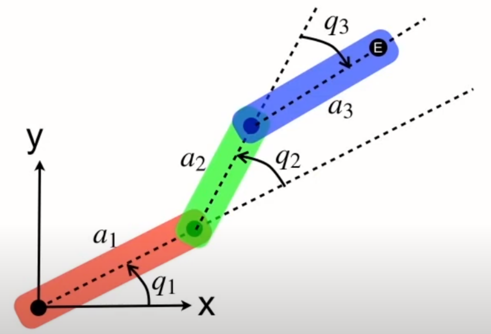
$$E = \boldsymbol{R}(q_1)$$

Pose of the End-Effector - 3-Joint 2D Robot Arm



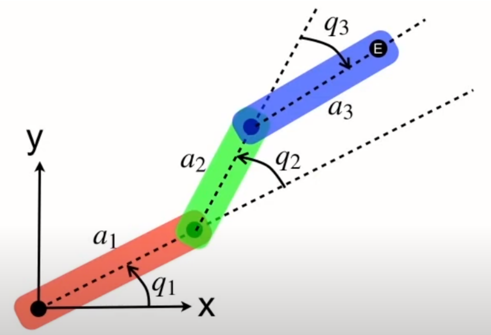
$$E = \boldsymbol{R}(q_1) \cdot \boldsymbol{T}_x(a_1)$$

Pose of the End-Effector - 3-Joint 2D Robot Arm



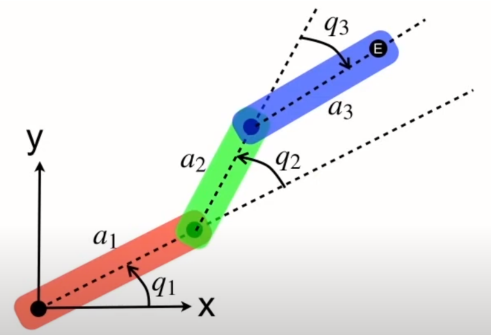
$$E = \boldsymbol{R}(q_1) \cdot \boldsymbol{T}_x(a_1) \cdot \boldsymbol{R}(q_2)$$

Pose of the End-Effector - 3-Joint 2D Robot Arm



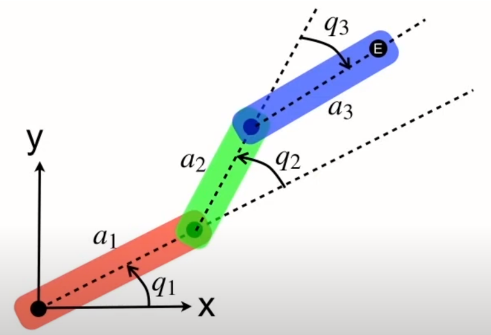
$$E = \boldsymbol{R}(q_1) \cdot \boldsymbol{T}_x(a_1) \cdot \boldsymbol{R}(q_2) \cdot \boldsymbol{T}_x(a_2)$$

Pose of the End-Effector - 3-Joint 2D Robot Arm



$$E = \boldsymbol{R}(q_1) \cdot \boldsymbol{T}_x(a_1) \cdot \boldsymbol{R}(q_2) \cdot \boldsymbol{T}_x(a_2) \cdot \boldsymbol{R}(q_3)$$

Pose of the End-Effector - 3-Joint 2D Robot Arm



$$E = \boldsymbol{R}(q_1) \cdot \boldsymbol{T}_x(a_1) \cdot \boldsymbol{R}(q_2) \cdot \boldsymbol{T}_x(a_2) \cdot \boldsymbol{R}(q_3) \cdot \boldsymbol{T}(a_3)$$

Pose of the End-Effector - 3-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

Pose of the End-Effector - 3-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

```
>>> from sympy import *
```

Pose of the End-Effector - 3-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

```
>>> from sympy import *

>>> q1 = Symbol('q1')
```

Pose of the End-Effector - 3-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

```
>>> from sympy import *

>>> q1 = Symbol('q1')
>>> trot2(q1)
```

Pose of the End-Effector - 3-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

```
>>> from sympy import *

>>> q1 = Symbol('q1')
>>> trot2(q1)

>>> a1=Symbol('a1')
```

Pose of the End-Effector - 3-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

```
>>> from sympy import *

>>> q1 = Symbol('q1')
>>> trot2(q1)

>>> a1=Symbol('a1')
>>> transl2(a1,0)
```

Pose of the End-Effector - 3-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

```
>>> from sympy import *

>>> q1 = Symbol('q1')
>>> trot2(q1)

>>> a1=Symbol('a1')
>>> transl2(a1,0)

>>> q2 = Symbol('q2')
```

Pose of the End-Effector - 3-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

```
>>> from sympy import *

>>> q1 = Symbol('q1')
>>> trot2(q1)

>>> a1=Symbol('a1')
>>> transl2(a1,0)

>>> q2 = Symbol('q2')
>>> a2 = Symbol('a2')
```

Pose of the End-Effector - 3-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

```
>>> from sympy import *

>>> q1 = Symbol('q1')
>>> trot2(q1)

>>> a1=Symbol('a1')
>>> transl2(a1,0)

>>> q2 = Symbol('q2')
>>> a2 = Symbol('a2')

>>> q3 = Symbol('q3')
```

Pose of the End-Effector - 3-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

```
>>> from sympy import *

>>> q1 = Symbol('q1')
>>> trot2(q1)

>>> a1=Symbol('a1')
>>> transl2(a1,0)

>>> q2 = Symbol('q2')
>>> a2 = Symbol('a2')

>>> q3 = Symbol('q3')
>>> a3 = Symbol('a3')
```

Pose of the End-Effector - 3-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

```
>>> from sympy import *

>>> q1 = Symbol('q1')
>>> trot2(q1)

>>> a1=Symbol('a1')
>>> transl2(a1,0)

>>> q2 = Symbol('q2')
>>> a2 = Symbol('a2')

>>> q3 = Symbol('q3')
>>> a3 = Symbol('a3')

>>> E = trot2(q1)@transl2(a1, 0)@trot2(q2)@transl2(a2, 0) \
                    @ trot2(q3) @ transl2(a3, 0)
```

Pose of the End-Effector - 3-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

```
>>> from sympy import *

>>> q1 = Symbol('q1')
>>> trot2(q1)

>>> a1=Symbol('a1')
>>> transl2(a1,0)

>>> q2 = Symbol('q2')
>>> a2 = Symbol('a2')

>>> q3 = Symbol('q3')
>>> a3 = Symbol('a3')

>>> E = trot2(q1)@transl2(a1, 0)@trot2(q2)@transl2(a2, 0) \
                    @ trot2(q3) @ transl2(a3, 0)
E = simplify(E)
```

Pose of the End-Effector - 3-Joint 2D Robot Arm (cont'd)

In Python Robotics Toolbox:

```
>>> from sympy import *

>>> q1 = Symbol('q1')
>>> trot2(q1)

>>> a1=Symbol('a1')
>>> transl2(a1,0)

>>> q2 = Symbol('q2')
>>> a2 = Symbol('a2')

>>> q3 = Symbol('q3')
>>> a3 = Symbol('a3')

>>> E = trot2(q1)@transl2(a1, 0)@trot2(q2)@transl2(a2, 0) \
                    @ trot2(q3) @ transl2(a3, 0)
E = simplify(E)
```

Demo of 3-joint arm shown in the class (see video recording)

Pose of the End-Effector - 3-Joint 2D Robot Arm (cont'd)

- Unlike the 1 and 2-joint robot arms, the 3-joint robot arm has 3 degrees of freedom and therefore it can achieve different orientations.

Pose of the End-Effector - 3-Joint 2D Robot Arm (cont'd)

- Unlike the 1 and 2-joint robot arms, the 3-joint robot arm has 3 degrees of freedom and therefore it can achieve different orientations.

The x coordinate of the end-effector is given by:

Pose of the End-Effector - 3-Joint 2D Robot Arm (cont'd)

- Unlike the 1 and 2-joint robot arms, the 3-joint robot arm has 3 degrees of freedom and therefore it can achieve different orientations.

The x coordinate of the end-effector is given by:

```
>>> E[0, 2]  #first row, third column
```

Pose of the End-Effector - 3-Joint 2D Robot Arm (cont'd)

- Unlike the 1 and 2-joint robot arms, the 3-joint robot arm has 3 degrees of freedom and therefore it can achieve different orientations.

The x coordinate of the end-effector is given by:

```
>>> E[0, 2]  #first row, third column
```

The y coordinate of the end-effector is given by:

Pose of the End-Effector - 3-Joint 2D Robot Arm (cont'd)

- Unlike the 1 and 2-joint robot arms, the 3-joint robot arm has 3 degrees of freedom and therefore it can achieve different orientations.

The x coordinate of the end-effector is given by:

```
>>> E[0, 2]   #first row, third column
```

The y coordinate of the end-effector is given by:

```
>>> E[1, 2]   #second row, third column
```

Pose of the End-Effector - 3-Joint 2D Robot Arm (cont'd)

- Unlike the 1 and 2-joint robot arms, the 3-joint robot arm has 3 degrees of freedom and therefore it can achieve different orientations.

The x coordinate of the end-effector is given by:

```
>>> E[0, 2]  #first row, third column
```

The y coordinate of the end-effector is given by:

```
>>> E[1, 2]  #second row, third column
```

The orientation of the end-effector is given by:

Pose of the End-Effector - 3-Joint 2D Robot Arm (cont'd)

- Unlike the 1 and 2-joint robot arms, the 3-joint robot arm has 3 degrees of freedom and therefore it can achieve different orientations.

The x coordinate of the end-effector is given by:

```
>>> E[0, 2]   #first row, third column
```

The y coordinate of the end-effector is given by:

```
>>> E[1, 2]   #second row, third column
```

The orientation of the end-effector is given by: $q_1 + q_2 + q_3$

The Problem of Forward Kinematics

The calculation of the position and orientation of a robot's end-effector from its joint coordinates θ_i.

The Problem of Forward Kinematics

The calculation of the position and orientation of a robot's end-effector from its joint coordinates θ_i.

- In the previous slides it has been shown how to do this in 2D spaces for:

The Problem of Forward Kinematics

The calculation of the position and orientation of a robot's end-effector from its joint coordinates θ_i.

- In the previous slides it has been shown how to do this in 2D spaces for:
  - 1-joint robot arms

The Problem of Forward Kinematics

The calculation of the position and orientation of a robot's end-effector from its joint coordinates θ_i.

- In the previous slides it has been shown how to do this in 2D spaces for:
  - 1-joint robot arms
  - 2-joint robot arms

The Problem of Forward Kinematics

The calculation of the position and orientation of a robot's end-effector from its joint coordinates θ_i.

- In the previous slides it has been shown how to do this in 2D spaces for:
  - 1-joint robot arms
  - 2-joint robot arms
  - 3-joint robot arms

using simple transformations in Mathematics which correspond to real operations in Physics!

Velocity of the End-Effector

The Problem:

Velocity of the End-Effector

The Problem:

- If the joints move at specific velocities, what is the velocity of the end-effector?

Velocity of the End-Effector

The Problem:

- If the joints move at specific velocities, what is the velocity of the end-effector?

Extremely important to control the operation of the end-effector (hand) of robots!

Velocity of the End-Effector

The Problem:

- If the joints move at specific velocities, what is the velocity of the end-effector?

Extremely important to control the operation of the end-effector (hand) of robots!

Calculation needed:

Velocity of the End-Effector

The Problem:

- If the joints move at specific velocities, what is the velocity of the end-effector?

Extremely important to control the operation of the end-effector (hand) of robots!

Calculation needed: Given the $\dot{\boldsymbol{q}}$ (time rate of change of joints angles) calculate the time rate of change of the pose of the end-effector $\dot{\boldsymbol{\xi}}_E$.

Velocity of the End-Effector

The Problem:

- If the joints move at specific velocities, what is the velocity of the end-effector?

Extremely important to control the operation of the end-effector (hand) of robots!

Calculation needed: Given the $\dot{\boldsymbol{q}}$ (time rate of change of joints angles) calculate the time rate of change of the pose of the end-effector $\dot{\xi}_E$.

- $\dot{\boldsymbol{q}}$ is the derivative of $\boldsymbol{q}$

Velocity of the End-Effector

The Problem:

- If the joints move at specific velocities, what is the velocity of the end-effector?

Extremely important to control the operation of the end-effector (hand) of robots!

Calculation needed: Given the $\dot{\boldsymbol{q}}$ (time rate of change of joints angles) calculate the time rate of change of the pose of the end-effector $\dot{\boldsymbol{\xi}}_E$.

- $\dot{\boldsymbol{q}}$ is the derivative of $\boldsymbol{q}$
- $\dot{\boldsymbol{\xi}}_E$ is the derivative of the pose (position and orientation) $\boldsymbol{\xi}_E$ of the end-effector

What is a Derivative?

What is a Derivative?

The derivative of a function measures the sensitivity of changes to the output (value) of the function, based on changes of the input (independent variable) of the function.

What is a Derivative?

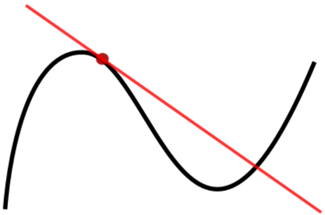
The derivative of a function measures the sensitivity of changes to the output (value) of the function, based on changes of the input (independent variable) of the function.

- The slope of the tangent line is equal to the derivative.

What is a Derivative?

The derivative of a function measures the sensitivity of changes to the output (value) of the function, based on changes of the input (independent variable) of the function.

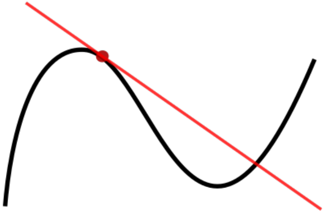
- The slope of the tangent line is equal to the derivative.



What is a Derivative?

The derivative of a function measures the sensitivity of changes to the output (value) of the function, based on changes of the input (independent variable) of the function.

- The slope of the tangent line is equal to the derivative.



$\longrightarrow$ It is also used to show the direction we need to follow and the magnitude of the step we need to take, in order to reduce an error (machine learning, etc).

Simple Numerical Calculation of Derivatives

In a numerical simulation, if we take small enough time steps, the derivative can be calculated as:

$$\frac{f(x_{t+1}) - f(x_t)}{\Delta t} \tag{2}$$

where

- $f(x_{t+1})$ is the value of function f at time $t+1$

Simple Numerical Calculation of Derivatives

In a numerical simulation, if we take small enough time steps, the derivative can be calculated as:

$$\frac{f(x_{t+1}) - f(x_t)}{\Delta t} \tag{2}$$

where

- $f(x_{t+1})$ is the value of function f at time $t+1$
- $f(x_t)$ is the value of function f at time t

Simple Numerical Calculation of Derivatives

In a numerical simulation, if we take small enough time steps, the derivative can be calculated as:

$$\frac{f(x_{t+1}) - f(x_t)}{\Delta t} \tag{2}$$

where

- $f(x_{t+1})$ is the value of function f at time $t+1$
- $f(x_t)$ is the value of function f at time t
- Δt is the time step, i.e. the difference (time elapsed) between the two successive time steps.

Simple Numerical Calculation of Derivatives

In a numerical simulation, if we take small enough time steps, the derivative can be calculated as:

$$\frac{f(x_{t+1}) - f(x_t)}{\Delta t} \tag{2}$$

where

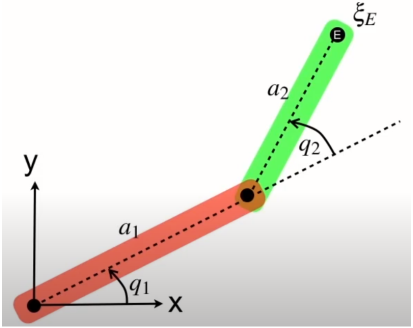
- $f(x_{t+1})$ is the value of function f at time $t+1$
- $f(x_t)$ is the value of function f at time t
- Δt is the time step, i.e. the difference (time elapsed) between the two successive time steps.

When a function f involves more than one independent variables, e.g. $f(x_1, x_2)$ the derivative with respect to one of these variables is called *partial derivative* and it is denoted as $\frac{\partial f}{\partial x_1}$, $\frac{\partial f}{\partial x_2}$, etc.:

Velocity of End-Effector in a 2-Joint Robot Arm (2D)

Relationship of the velocities of individual joints q_1 and q_2 and the velocity of the end-effector.

- It can be shown that instantaneously the velocity of the end-effector is the sum of the end effector velocity components due to motion of joint 1 and the motion due to joint 2 .



Velocity of End-Effector in a 2-Joint Robot Arm (2D) - cont'd

The position of the end-effector is given (see previous slides) by:

Velocity of End-Effector in a 2-Joint Robot Arm (2D) - cont'd

The position of the end-effector is given (see previous slides) by:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a_1 cos(q_1) + a_2 cos(q_1 + q_2) \\ a_1 sin(q_1) + a_2 sin(q_1 + q_2) \end{pmatrix} \tag{3}$$

Velocity of End-Effector in a 2-Joint Robot Arm (2D) - cont'd

The position of the end-effector is given (see previous slides) by:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a_1 cos(q_1) + a_2 cos(q_1 + q_2) \\ a_1 sin(q_1) + a_2 sin(q_1 + q_2) \end{pmatrix} \tag{3}$$

- If joint angles change over time (the robot moves):

$$q_1 = q_1(t), \quad q_2 = q_2(t)$$

- The velocity of the end-effector can be calculated by computing the derivative (using the chain rule):

$$\dot{x} = -a_1 \dot{q_1} sin(q1) - a_2(\dot{q_1} + \dot{q_2}) sin(q1 + q2) \tag{4}$$
$$\dot{y} = a_1 \dot{q_1} cos(q_1) + a_2(\dot{q_1} + \dot{q_2}) cos(q1 + q2) \tag{5}$$

where $\dot{q_1} = \frac{\partial q_1}{\partial t}, \quad \dot{q_2} = \frac{\partial q_2}{\partial t}$

Velocity of End-Effector in a 2-Joint Robot Arm (2D) - cont'd

Equations (4), (5):

$$\dot{x} = -a_1 \dot{q}_1 sin(q1) - a_2(\dot{q}_1 + \dot{q}_2) sin(q1 + q2) \quad (6)$$

$$\dot{y} = a_1 \dot{q}_1 cos(q_1) + a_2(\dot{q}_1 + \dot{q}_2) cos(q1 + q2) \quad (7)$$

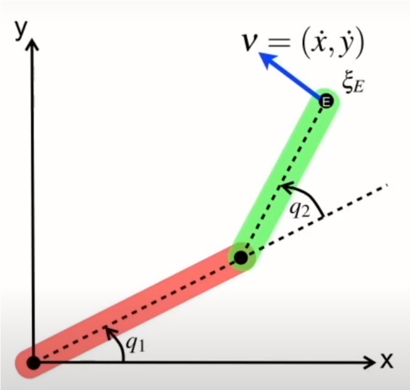
can be written in matrix form:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} -a_1 sin(q1) - a_2 sin(q1 + q2) - a_2 sin(q1 + q2) \\ a_1 cos(q_1) + a_2 cos(q1 + q2) a_2 cos(q1 + q2) \end{pmatrix} \begin{pmatrix} \dot{q}_1 \\ \dot{q}_2 \end{pmatrix}$$

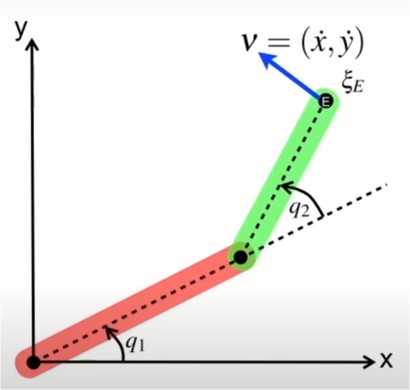
or

$$\boldsymbol{v} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} \quad (8)$$

Velocity of End-Effector in a 2-Joint Robot Arm (2D) - cont'd



Velocity of End-Effector in a 2-Joint Robot Arm (2D) - cont'd



$\boldsymbol{J}(\boldsymbol{q})$ is the Jacobian matrix of the joint angles q_1 and q_2:

$$\boldsymbol{J}(\boldsymbol{q}) = \begin{pmatrix} -a_1 sin(q1) - a_2 sin(q1+q2) - a_2 sin(q1+q2) \\ a_1 cos(q_1) + a_2 cos(q1+q2) a_2 cos(q1+q2) \end{pmatrix}$$

More on Jacobian

For a scalar value x and a scalar function f:

$$y = f(x)$$

More on Jacobian

For a scalar value x and a scalar function f:

$$y = f(x)$$

the derivative of f is:

$$\frac{df}{dx} = \frac{dy}{dx}$$

More on Jacobian

For a scalar value x and a scalar function f:

$$y = f(x)$$

the derivative of f is:

$$\frac{df}{dx} = \frac{dy}{dx}$$

The Jacobian is the equivalent for the derivative of a matrix:

More on Jacobian

For a scalar value x and a scalar function f:

$$y = f(x)$$

the derivative of f is:

$$\frac{df}{dx} = \frac{dy}{dx}$$

The Jacobian is the equivalent for the derivative of a matrix:

- the derivative of a function which has a vector as an argument and returns a vector as its result:

$$\boldsymbol{J} = \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \dots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \dots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \dots & \frac{\partial y_m}{\partial x_n} \end{pmatrix}$$

The Chain Rule for Calculating Derivatives

The *chain rule* is used to differentiate a function which has an argument another function (i.e. a composite function).

- Assuming a function: $y = f(g(x))$

The Chain Rule for Calculating Derivatives

The *chain rule* is used to differentiate a function which has an argument another function (i.e. a composite function).

- Assuming a function: $y = f(g(x))$

The chain rule states that the derivative of y with respect to x, i.e. $\frac{dy}{dx}$ can be calculated as follows:

The Chain Rule for Calculating Derivatives

The *chain rule* is used to differentiate a function which has an argument another function (i.e. a composite function).

- Assuming a function: $y = f(g(x))$

The chain rule states that the derivative of y with respect to x, i.e. $\frac{dy}{dx}$ can be calculated as follows:

1. Substitute $u = g(x)$.

The Chain Rule for Calculating Derivatives

The *chain rule* is used to differentiate a function which has an argument another function (i.e. a composite function).

- Assuming a function: $y = f(g(x))$

The chain rule states that the derivative of y with respect to x, i.e. $\frac{dy}{dx}$ can be calculated as follows:

1. Substitute $u = g(x)$. Then:

$$y = f(u)$$

The Chain Rule for Calculating Derivatives

The *chain rule* is used to differentiate a function which has an argument another function (i.e. a composite function).

- Assuming a function: $y = f(g(x))$

The chain rule states that the derivative of y with respect to x, i.e. $\frac{dy}{dx}$ can be calculated as follows:

1. Substitute $u = g(x)$. Then:

$$y = f(u)$$

2. **Chain rule:**

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx} \tag{9}$$

The Chain Rule for Calculating Derivatives

The *chain rule* is used to differentiate a function which has an argument another function (i.e. a composite function).

- Assuming a function: $y = f(g(x))$

The chain rule states that the derivative of y with respect to x, i.e. $\frac{dy}{dx}$ can be calculated as follows:

1. Substitute $u = g(x)$. Then:

$$y = f(u)$$

2. **Chain rule:**

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx} \tag{9}$$

Example:

Differentiate $y = sinx^2$:

The Chain Rule for Calculating Derivatives

The *chain rule* is used to differentiate a function which has an argument another function (i.e. a composite function).

- Assuming a function: $y = f(g(x))$

The chain rule states that the derivative of y with respect to x, i.e. $\frac{dy}{dx}$ can be calculated as follows:

1. Substitute $u = g(x)$. Then:

$$y = f(u)$$

2. **Chain rule:**

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx} \tag{9}$$

Example:

Differentiate $y = sinx^2$:

1. $u = x^2$

The Chain Rule for Calculating Derivatives

The *chain rule* is used to differentiate a function which has an argument another function (i.e. a composite function).

- Assuming a function: $y = f(g(x))$

The chain rule states that the derivative of y with respect to x, i.e. $\frac{dy}{dx}$ can be calculated as follows:

1. Substitute $u = g(x)$. Then:

$$y = f(u)$$

2. **Chain rule:**

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx} \tag{9}$$

Example:

Differentiate $y = sinx^2$:

1. $u = x^2$ then:
2. $y = sin(u)$

The Chain Rule for Calculating Derivatives

The *chain rule* is used to differentiate a function which has an argument another function (i.e. a composite function).

- Assuming a function: $y = f(g(x))$

The chain rule states that the derivative of y with respect to x, i.e. $\frac{dy}{dx}$ can be calculated as follows:

1. Substitute $u = g(x)$. Then:
$$y = f(u)$$
2. **Chain rule:**
$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx} \tag{9}$$

Example:

Differentiate $y = sinx^2$:

1. $u = x^2$ then:
2. $y = sin(u)$
3. $$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx} = cos(x^2) \cdot 2x \tag{10}$$

Calculating the Joint Velocities for a Desired End-Effector Velocity

Calculating the Joint Velocities for a Desired End-Effector Velocity

In real world, we need to specify a velocity for the end-effector.

Calculating the Joint Velocities for a Desired End-Effector Velocity

In real world, we need to specify a velocity for the end-effector.

- How do I achieve this, what velocities do I need to apply to the joints of the robot using my actuators (control motors in the joints)?

From Equation (8):

$$\boldsymbol{v} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{11}$$

Calculating the Joint Velocities for a Desired End-Effector Velocity

In real world, we need to specify a velocity for the end-effector.

- How do I achieve this, what velocities do I need to apply to the joints of the robot using my actuators (control motors in the joints)?

From Equation (8):

$$\boldsymbol{v} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{11}$$

Multiplying both sides of the equation from the left by the inverse of the Jacobian matrix:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}(\boldsymbol{q})^{-1} \cdot \boldsymbol{v} \tag{12}$$