Forward Kinematics & Jacobian (Refresher)

Technische Universität Berlin

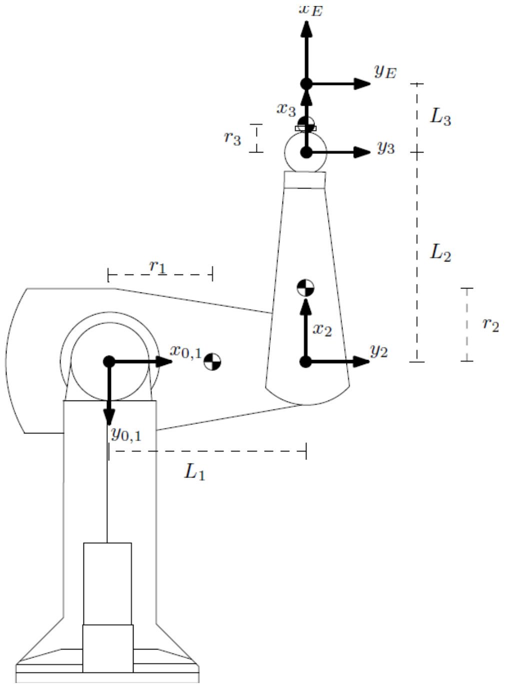
3D Homogeneous Transforms

[Lecture video]

$$\begin{pmatrix} p'_x \\ p'_y \\ p'_z \\ 1 \end{pmatrix} = \begin{bmatrix} & R(\theta) & & \begin{matrix} t_x \\ t_y \\ t_z \end{matrix} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$$

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

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



Frame to Frame Transformation

[Craig] P. 74

$$ {}^{i-1}_{i}T = R_X(\alpha_{i-1})\, D_X(a_{i-1})\, R_Z(\theta_i)\, D_Z(d_i) $$

$$ = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1} d_i \\ s\theta_i s\alpha_{i-1} & c\theta_i s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1} d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ \begin{bmatrix} \alpha_i & = & \text{the angle between } \hat{Z}_i \text{ and } \hat{Z}_{i+1} \text{ measured about } \hat{X}_i \\ a_i & = & \text{the distance from } \hat{Z}_i \text{ to } \hat{Z}_{i+1} \text{ measured along } \hat{X}_i \\ d_i & = & \text{the distance from } \hat{X}_{i-1} \text{ to } \hat{X}_i \text{ measured along } \hat{Z}_i \\ \theta_i & = & \text{the angle between } \hat{X}_{i-1} \text{ and } \hat{X}_i \text{ measured about } \hat{Z}_i \end{bmatrix} $$

Computing with HTs

[Lecture video]

$$ {}^{A}_{F}T = {}^{A}_{B}T \; {}^{B}_{C}T \; {}^{C}_{D}T \; {}^{D}_{E}T \; {}^{E}_{F}T $$

$$ {}^{A}\vec{p} = {}^{A}_{B}T \; {}^{B}_{C}T \; {}^{C}\vec{p} $$

Matrix multiplication is NOT commutative!

ORDER MATTERS !!!

forwardkinematics.cpp — C:\Users\Vincent\Clouds\tubCloud\Work\Teaching\Robotics_WS2021\forwardkinematics_WS1920\forwardkinematics — Atom

File Edit View Selection Find Packages Help

forwardkinematics.cpp | forwardkinematics.hpp

```cpp
/****************************************************/
/*****************EDIT BELOW *****************/
/****************************************************/

/*
updates the variable T0_1

<ADD EXPLANATION OF CODE>
*/
void ForwardKinematicsPuma2D::computeT0_1()
{

   // compute from
   // angles[0], angles[1], angles[2], l1, l2, and l3

   //row vector
   T0_1[0][0] = 0.0;
   T0_1[0][1] = 0.0;
   T0_1[0][2] = 0.0;
   T0_1[0][3] = 0.0;

   //row vector
   T0_1[1][0] = 0.0;
   T0_1[1][1] = 0.0;
   T0_1[1][2] = 0.0;
   T0_1[1][3] = 0.0;

   //row vector
   T0_1[2][0] = 0.0;
   T0_1[2][1] = 0.0;
   T0_1[2][2] = 0.0;
```

forwardkinematics.cpp 111:1 LF UTF-8

forwardkinematics.cpp — C:\Users\Vincent\Clouds\tubCloud\Work\Teaching\Robotics_WS2021\forwardkinematics_WS1920\forwardkinematics — Atom

File Edit View Selection Find Packages Help

forwardkinematics.cpp × | forwardkinematics.hpp

```cpp
*/
void ForwardKinematicsPuma2D::computeF()
{

   // compute from
   // angles[0], angles[1], angles[2], l1, l2, and l3

   F[0] = 0.0; //x
   F[1] = 0.0; //y
   F[2] = 0.0; //alpha
}


/*
This function updates the variable J

<ADD EXPLANATION OF CODE>
*/
void ForwardKinematicsPuma2D::computeJ()
{

   // compute from
   // angles[0], angles[1], angles[2], l1, l2, and l3

   //row vector
   J[0][0] = 0.0;
   J[0][1] = 0.0;
   J[0][2] = 0.0;

   //row vector
   J[1][0] = 0.0;
```

- # main
- f(x) print_HTransform
- f(x) print_Jacobian
- f(x) print_Position
- {c} ForwardKinematicsPuma2D
  - f(x) setJoints
  - f(x) computeT0_1
  - f(x) computeT1_2
  - f(x) computeT2_3
  - f(x) computeT3_E
  - f(x) computeT0_E
  - f(x) computeF
  - f(x) computeJ
- f(x) main

forwardkinematics.cpp 299:1 LF UTF-8 C++ GitHub Git (0) forwardkinematics.cpp — C:\Users\Vincent\Clouds\tubCloud\Work\Teaching\Robotics_WS2021\forwardkinematics_WS1920\forwardkinematics — Atom

File Edit View Selection Find Packages Help

forwardkinematics.cpp | forwardkinematics.hpp

```cpp
    //row vector
    J[2][0] = 0.0;
    J[2][1] = 0.0;
    J[2][2] = 0.0;
}



/*
Example code to test your functions:

You are free to change main() as you like
*/
int main()
{
 ForwardKinematicsPuma2D* fk = new ForwardKinematicsPuma2D();
 fk->setJoints(0.0,0.0,0.0); //example, try out different values
 cout << "********************Testing Transforms**************"<<e
 print_HTransform(fk->T0_1);
 print_HTransform(fk->T1_2);
 print_HTransform(fk->T2_3);
 print_HTransform(fk->T3_E);
 print_HTransform(fk->T0_E);
 cout << "********************Testing F***********************"<<e
 print_Position(fk->F);
 cout << "********************Testing J***********************"<<e
 print_Jacobian(fk->J);
 return 0;
}
```

- # main
- f(x) print_HTransform
- f(x) print_Jacobian
- f(x) print_Position
- {C} ForwardKinematicsPuma2D
  - f(x) setJoints
  - f(x) computeT0_1
  - f(x) computeT1_2
  - f(x) computeT2_3
  - f(x) computeT3_E
  - f(x) computeT0_E
  - f(x) computeF
  - f(x) computeJ
- f(x) main

forwardkinematics.cpp 345:1 LF UTF-8 C++ GitHub Git (0)

Visualizing the Jacobian

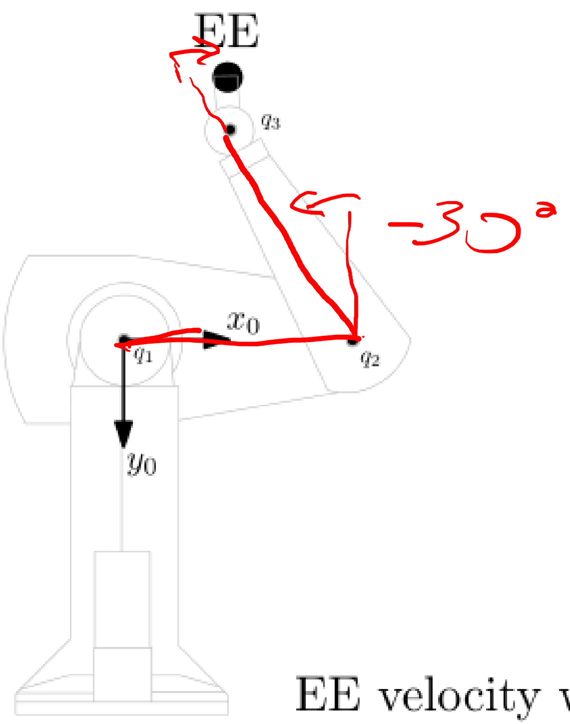
$$\delta \mathbf{x} = \begin{bmatrix} \frac{\partial f_1}{\partial q_1} & \frac{\partial f_1}{\partial q_2} & \cdots & \frac{\partial f_1}{\partial q_n} \\ \frac{\partial f_2}{\partial q_1} & \frac{\partial f_2}{\partial q_2} & \cdots & \frac{\partial f_2}{\partial q_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial q_1} & \frac{\partial f_m}{\partial q_2} & \cdots & \frac{\partial f_m}{\partial q_n} \end{bmatrix} \delta \mathbf{q} = J_{(m \times n)}(\mathbf{q}) \delta \mathbf{q}$$

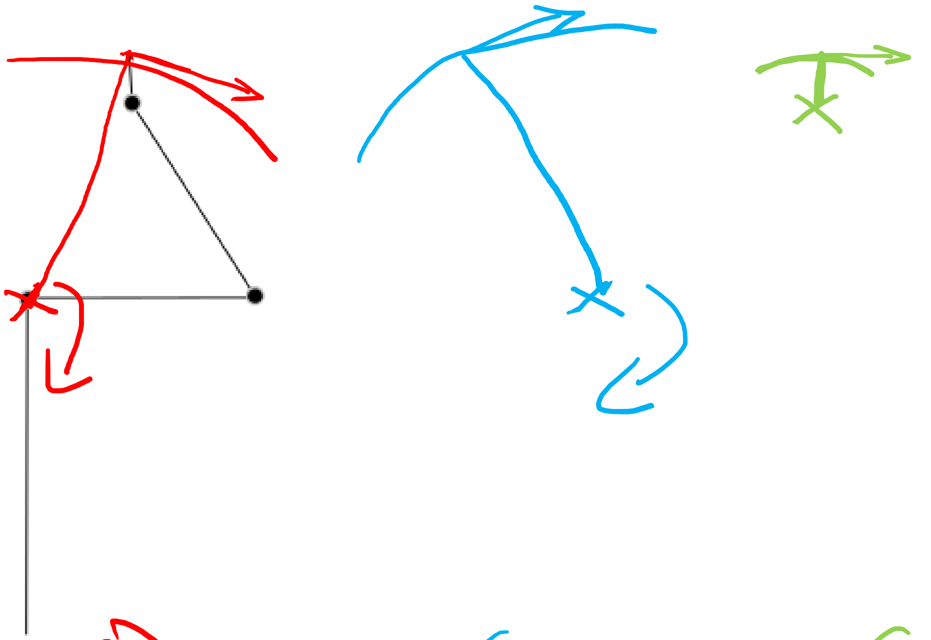
Visualizing the Jacobian

$$\delta \mathbf{x} = \begin{bmatrix} \frac{\partial f_1}{\partial q_1} & \frac{\partial f_1}{\partial q_2} & \cdots & \frac{\partial f_1}{\partial q_n} \\ \frac{\partial f_2}{\partial q_1} & \frac{\partial f_2}{\partial q_2} & \cdots & \frac{\partial f_2}{\partial q_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial q_1} & \frac{\partial f_m}{\partial q_2} & \cdots & \frac{\partial f_m}{\partial q_n} \end{bmatrix}$$

$25°$

$q = \begin{pmatrix} 0 \\ -30° \\ 25° \end{pmatrix}$





EE velocity when $\dot{q} = (1, 0, 0)$

0

0