Robotics

Assignment 1

Andreas Huber, Filip Dilber, Alexander Moser

Structure of the presentation

Overview of the package

How does it work

Explaination of the algorithmn

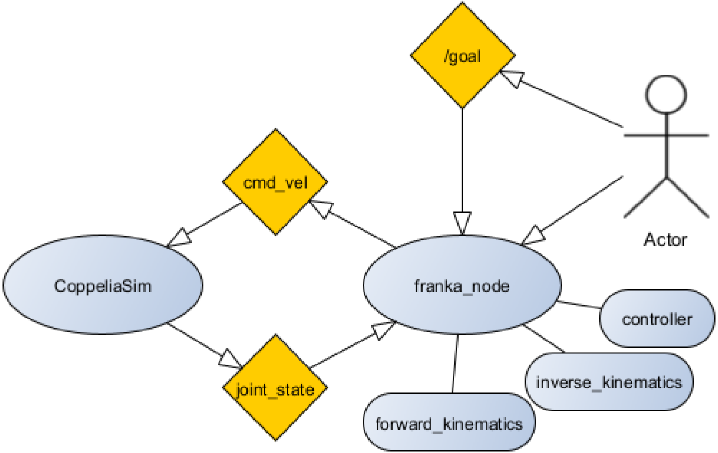
Live Demo

Overview of the ros-package

first_challange

- launch
  - first_assignment.launch
- scripts
  - franka_node.py
  - inverse_kinematics.py
  - forward_kinematics.py
  - controller.py
- package.xml
- CMakeList.txt

How does it work



The algorithmn

1. calculate Transformation-Matrices
2. calculate Total-Transformation-Matrix
   - extract rotation matrices
3. calculate Geometric Jacobian
   - calculate Z
   - calculate p
   - concat
4. calculate analytic Jacobian
5. calculate error rate
   - calculate f_q
6. update velocities

Calculate Transformation-Matrices

```
def calculate_t_matrix(a, alpha, d, theta):

    A = np.array([
        [(cos(theta)).evalf(), (-sin(theta)).evalf(), 0, 0],
        [(sin(theta)).evalf(), (cos(theta)).evalf(), 0, 0],
        [0, 0, 1, d],
        [0, 0, 0, 1]
    ])

    B = np.array([
        [1, 0, 0, a],
        [0, (cos(alpha)).evalf(), (-sin(alpha)).evalf(), 0],
        [0, (sin(alpha)).evalf(), (cos(alpha)).evalf(), 0],
        [0, 0, 0, 1]
    ])

    return A @ B
```

```python
def calculate_all_t_matrizes(a, alpha, d, theta):

    number_of_joints = len(theta)

    Ts = []

    for i in range(number_of_joints):
        t_matrix = calculate_t_matrix(a[i], alpha[i], d[i], theta[i])
        Ts.append(t_matrix)

    return Ts
```

Calculate Geometric Jacobian

```python
def calculate_geometric_jacobian(a, alpha, d, theta):

    Ts = forward_kinematics.calculate_incrementing_t_matrices(a, alpha, d, theta)
    total_T_matrix = forward_kinematics.calculate_total_t_matrix(a, alpha, d, theta)
    Zs = np.array([[0,0,1]])

    for matrix in Ts:
        Zs = np.append(Zs, matrix[:-1][:, 2])
    Zs = Zs.reshape(-1,3)

    Pe = total_T_matrix[:-1][:, 3]
    Ps = np.array([[0,0,0]])

    for matrix in Ts:
        Ps = np.append(Ps, matrix[:-1][:, 3])
    Ps = Ps.reshape(-1,3)
```

```python
upperrow = np.array([])
lowerrow = np.array([])

for i in range (len(Zs)):
    w = np.cross(Zs[i], (Pe-Ps[i]))
    upperrow = np.append(upperrow, w)
    lowerrow = np.append(lowerrow, Zs[i])

upperrow = upperrow.reshape(-1, 3)
lowerrow = lowerrow.reshape(-1, 3)

result = np.hstack((upperrow, lowerrow))

result = np.transpose(result)

return result
```

Calculate analytic Jacobian

```python
def calculate_analytic_jacobian_pseudo_invers(a, alpha, d, theta):

    analytic_jacobian = calculate_analytic_jacobian(a, alpha, d, theta)

    result = np.transpose(analytic_jacobian) @ np.linalg.pinv(analytic_jacobian @ np.transpose(analytic_jacobian))

    return result

def calculate_analytic_jacobian(a, alpha, d, theta):

    total_t_matrix = forward_kinematics.calculate_total_t_matrix(a, alpha, d, theta)
    jacobian_x = calculate_jacobian_x(total_t_matrix)
    geometric_jacobian = calculate_geometric_jacobian(a, alpha, d, theta)

    result = np.linalg.pinv(np.float64(jacobian_x)) @ np.float64(geometric_jacobian)

    return result
```

Calculate Euler-Angles

```
def calculate_euler_angles(total_t_matrix):

    phi = atan2(total_t_matrix[2][1], total_t_matrix[2][2])
    psi = atan2(total_t_matrix[1][0], total_t_matrix[0][0])
    theta  = atan2(-(total_t_matrix[2][0]), sp.sqrt((total_t_matrix[0][0] ** 2) + (total_t_matrix[1][0] ** 2)).evalf())

    return phi,psi,theta
```

Calculate End-Effektor-Position

```
def calculate_end_effector_position(total_t_matrix):

    x = total_t_matrix[:-1][0,-1]
    y = total_t_matrix[:-1][1,-1]
    z = total_t_matrix[:-1][2,-1]
    phi, psi, theta = calculate_euler_angles(total_t_matrix)

    result = [x,y,z,phi,psi,theta]

    return result
```

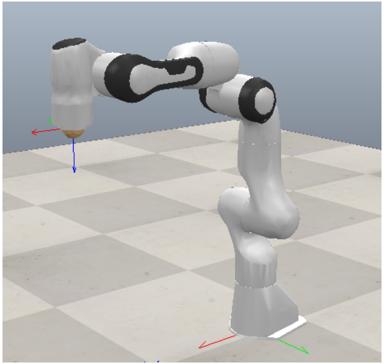
Update Velocities

```
while not rospy.is_shutdown():

    #actualize current theta/q
    theta = [current_q[0]+pi, current_q[1]-pi, current_q[2], current_q[3]+pi, current_q[4]-pi, current_q[5], current_q[6]]

    #forward kinematics getting endeffector position
    total_t_matrix = forward_kinematics.calculate_total_t_matrix(theta, d, a, alpha);
    Current_Endeffector_Position = forward_kinematics.calculate_end_effector_position(total_t_matrix);

    #calculating the error
    error = np.array(Current_Endeffector_Position) - np.array(Goal_Position)
    error = np.float64(error)

    #if below tolerance, sending stop signal to joints
    if (np.linalg.norm(error) < tolerance):
        ros_controller.moveToPosition(stop)
        continue

    #calculating the necassary velocity
    vel = - learning_rate * (inverse_kinematics.calculate_analytic_jacobian_pseudo_invers(theta, d, a, alpha) @ error)

    #oredering to move
    ros_controller.moveToPosition(vel)
    rate.sleep()
```

Problems we encountered

- Setup - how to start, where to begin?
- How to calculate Transformation-Matrices
- Velocity Mode vs Position Mode
- How to calculate error?
- catkin workspace

Its Demo Time



Thank you for your attention

Andreas Huber, Filip Dilber, Alexander Moser