Basic Pick and Place (part2)

-Robotic Manipulation-

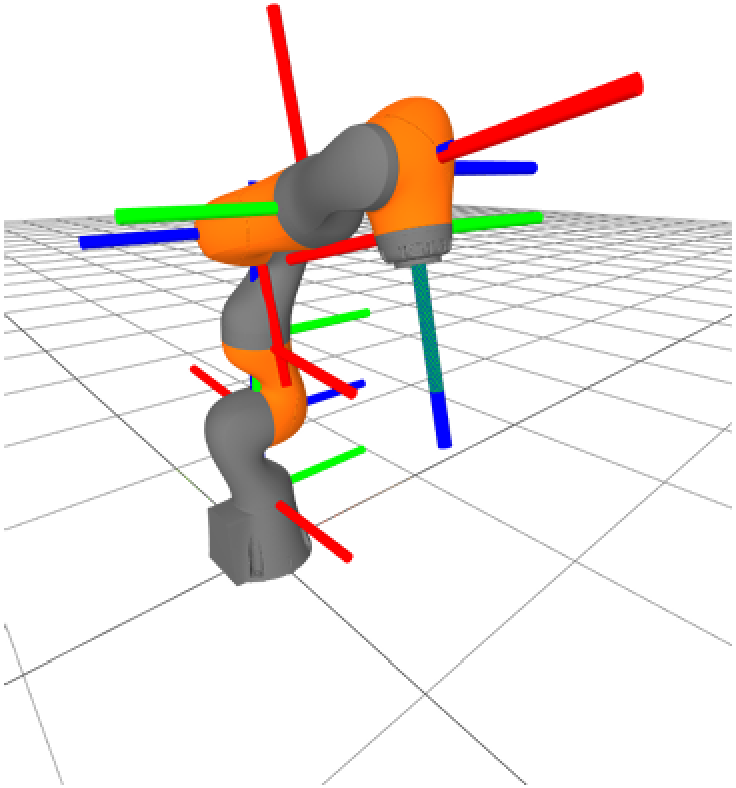
경희대학교

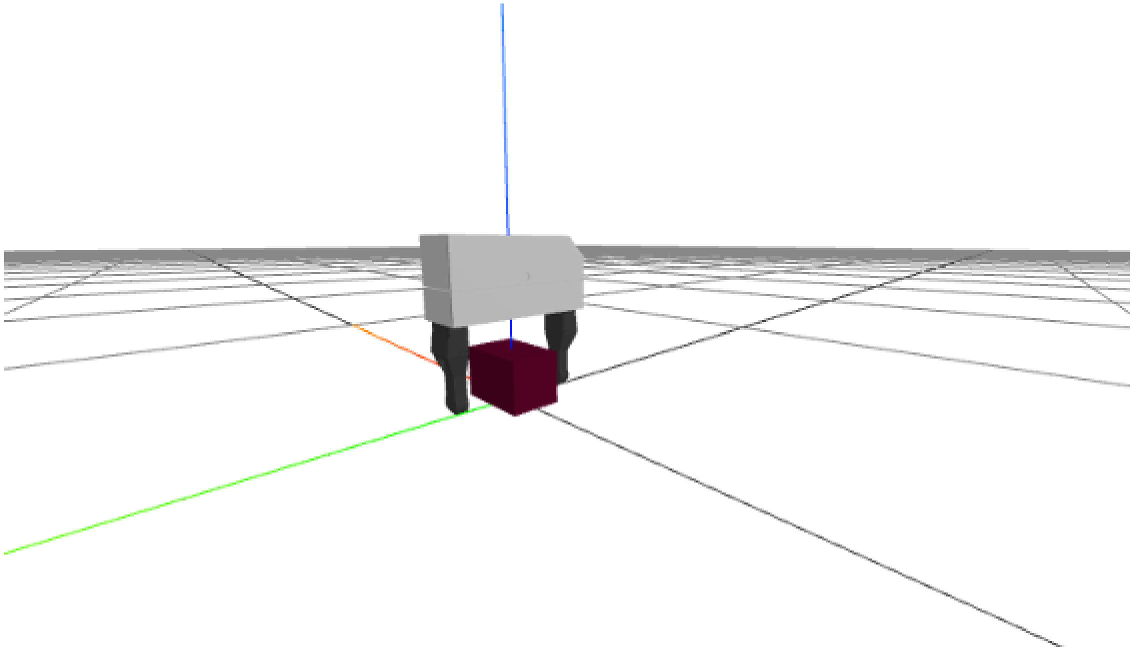
기계공학과 조교수
김상현

Basic Pick and Place

- https://manipulation.mit.edu/data/pick.html

Step 1: Kinematic Frames / Spatial Algebra

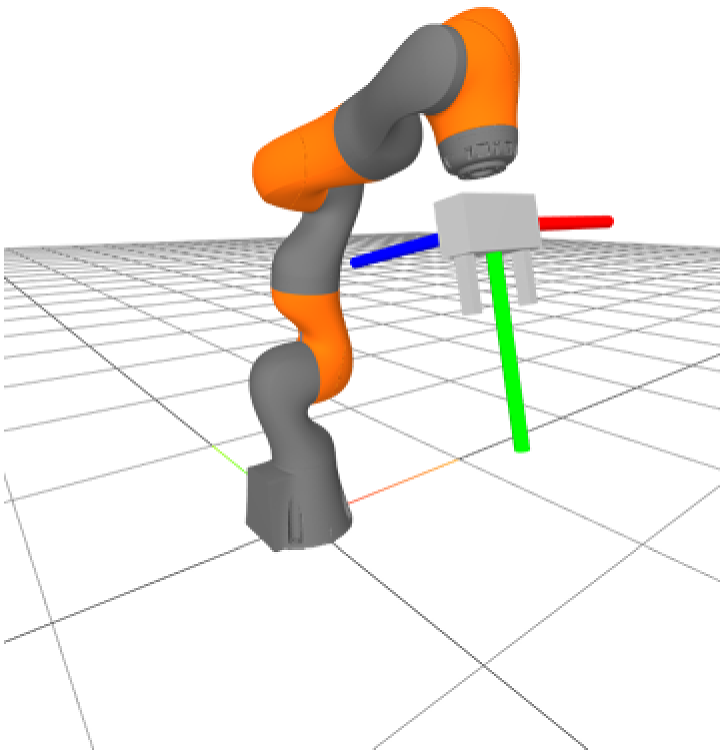


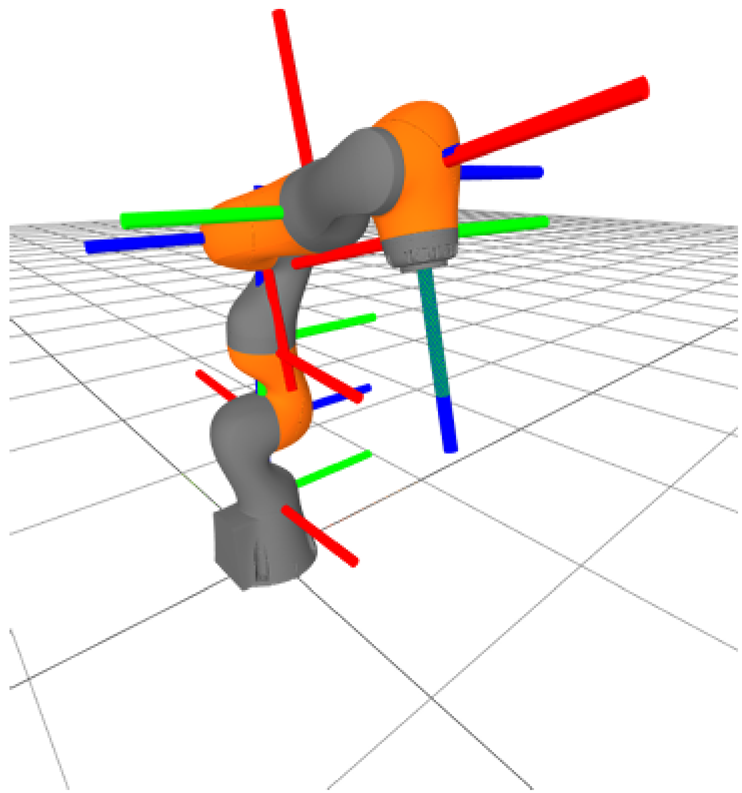


Step 2: Gripper Frame Plan "Sketch"

- https://manipulation.mit.edu/data/pick_sketch.html

Step 3: Forward kinematics of the iiwa + WSG





Step 4: Differential Inverse Kinematics

Kinematic Jacobian: ${}^{W}V^{G} = J^{G}(q)v$

spatial velocity

generalized velocity

V_WG → PseudoInverseController → iiwa_velocity_command

iiwa_position → PseudoInverseController

$$v = [J^{G}(q)]^{-1}\,{}^{W}V^{G}$$

Representations for 3D Rotation

- https://manipulation.csail.mit.edu/pick.html#3D_rotation

Representations for 3D Rotation

- Example: Gimbal Lock

Check yourself: A single "free" body

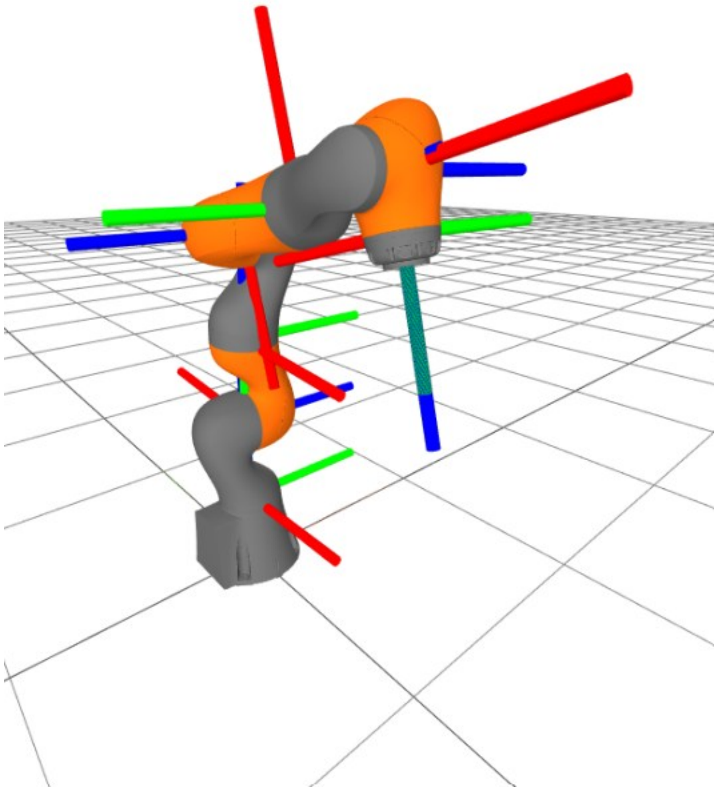
```python
plant = MultibodyPlant(time_step = 0.0)
Parser(plant).AddModelFromFile(FindResourceOrThrow(
        "drake/examples/manipulation_station/models/061_foam_brick.sdf"))
plant.Finalize()
brick = plant.GetBodyByName("base_link")

context = plant.CreateDefaultContext()
```

- What is q? `q = plant.GetPositions(context)`
- What is X^B? `X_B = plant.EvalBodyPoseInWorld(brick, context)`

$$X^B = f_{kin}^B(q)$$

Step 4: Differential Kinematics



$$X^B = f^B_{kin}(q)$$

$$dX^B = \frac{\partial f^B_{kin}(q)}{\partial q} dq$$

$$= \underbrace{J^B(q)}_{\text{Kinematic "Jacobian"}} dq$$

Spatial Velocity

$$\frac{d}{dt}\,{}^{A}X_{C}^{B} \equiv {}^{A}V_{C}^{B} = \begin{bmatrix} {}^{A}\omega_{C}^{B} \\ {}^{A}\mathbf{v}_{C}^{B} \end{bmatrix}$$

angular velocity

translational velocity

note the typesetting

- Q: How do we represent angular velocity, ω^{B}??

- A: Unlike 3D rotations, here 3 numbers are sufficient and efficient;
  so we use the same representation everywhere.

$$(\omega_x, \omega_y, \omega_z)$$

Spatial Velocity Algebra

- https://manipulation.csail.mit.edu/pick.html#jacobian

Check yourself: A single “free” body

```
plant = MultibodyPlant(time_step = 0.0)
Parser(plant).AddModelFromFile(FindResourceOrThrow(
          "drake/examples/manipulation_station/models/061_foam_brick.sdf"))
plant.Finalize()
brick = plant.GetBodyByName("base_link")

context = plant.CreateDefaultContext()
```

- What size is q?
- What size is v?

```
q = plant.GetPositions(context)

v = plant.GetVelocities(context)
```

- For any MultibodyPlant (not just a single body):

$$\dot{q} = N(q)v$$

```
v = plant.MapQDotToVelocity(context, qdot)

qdot = plant.MapVelocityToQDot(context, v)
```

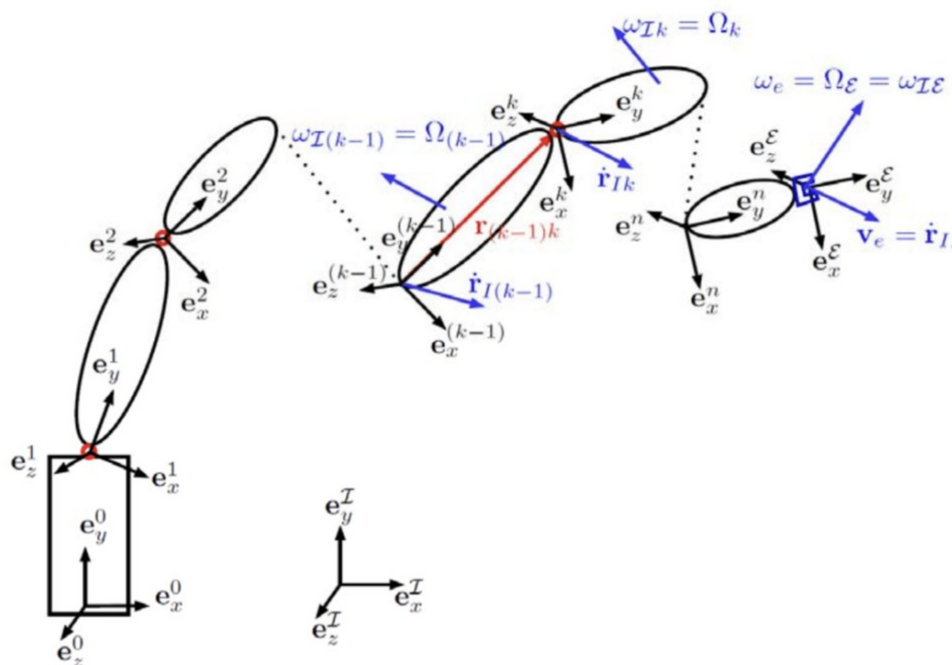
Geometric Jacobian Derivation

- Linear velocity

$$\dot{r}_{IE} = \underbrace{[n_1 \times r_{1(n+1)} \quad n_2 \times r_{2(n+1)} \quad \cdots \quad n_n \times r_{n(n+1)}]}_{J_{e0_P}} \begin{pmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \vdots \\ \dot{q}_n \end{pmatrix}$$

- Angular velocity

$$\omega_{\mathcal{IE}} = \sum_{i=1}^{n} n_i \dot{q}_i = \underbrace{[n_1 \quad n_2 \quad \cdots \quad n_n]}_{J_{e0_R}} \begin{pmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \vdots \\ \dot{q}_n \end{pmatrix}$$



- https://drake.mit.edu/doxygen_cxx/classdrake_1_1multibody_1_1_multibody_plant.html#a5f39a8f68fae3de53767a281503a3313

Kinematics

- *Forward kinematics* : joint positions ⇒ pose $q \Rightarrow X^B$
- *Inverse kinematics** : pose ⇒ joint positions $X^B \Rightarrow q$

- *Differential kinematics* :

  joint positions, velocities ⇒ spatial velocity $q, v \Rightarrow V^B$

- *Differential inverse kinematics* :

  spatial velocity, joint positions ⇒ joint velocities $V^B, q \Rightarrow v$

Moore-Penrose pseudo-inverse

- https://mathformachines.com/posts/least-squares-with-the-mp-inverse/

- https://manipulation.csail.mit.edu/data/two_link_singularities.html?height=400

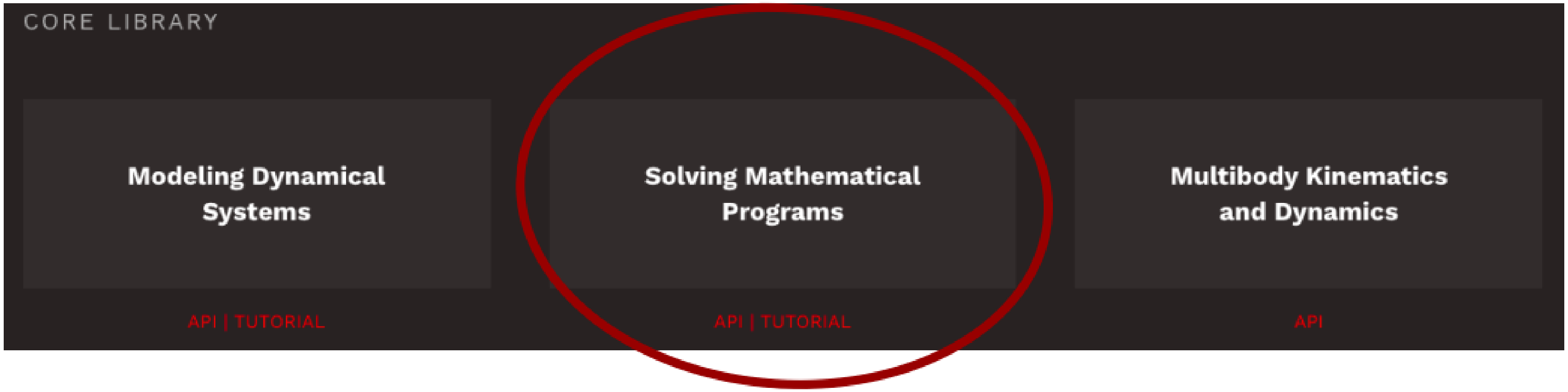
Our first Jacobian pseudo-inverse controller

$$v = [J^G(q)]^{-1}\,{}^W V^G$$

Our first Jacobian pseudo-inverse controller

```python
class PseudoInverseController(LeafSystem):
    def __init__(self, plant):
        LeafSystem.__init__(self)
        self.V_G_port = self.DeclareVectorInputPort("V_WG", 6)
        self.q_port = self.DeclareVectorInputPort("iiwa.position", 7
        self.DeclareVectorOutputPort("iiwa.velocity", 7, self.CalcOu

    def CalcOutput(self, context, output):
        V_G = self.V_G_port.Eval(context)
        q = self.q_port.Eval(context)
        self._plant.SetPositions(self._plant_context, self._iiwa, q)
        J_G = self._plant.CalcJacobianSpatialVelocity(
            self._plant_context,
            JacobianWrtVariable.kV,
            self._G,
            [0, 0, 0],
            self._W,
            self. W,
```




```
prog = MathematicalProgram()
x = prog.NewContinuousVariables(2)
prog.AddConstraint(x[0] + x[1] == 1)
prog.AddConstraint(x[0] <= x[1])
prog.AddCost(x[0] ** 2 + x[1] ** 2)
result = Solve(prog)
```

감사합니다

KHU