Τεχνολογίες Ευφυών Συστημάτων και Ρομποτική

RobotDART, URDF και Inverse Kinematics

Κωνσταντίνος Χατζηλυγερούδης - costashatz@upatras.gr

Τμήμα Μηχανικών Η/Υ και Πληροφορικής
Πανεπιστήμιο Πατρών

10 Μαρτίου 2023

Template made by **Παναγιώτης Παπαγιαννόπουλος**

Περιεχόμενα - Φροντιστήριο/Εργαστήριο

- **Ρομποτικοί Προσομοιωτές (Robotic Simulators)**
  - DART Simulator
  - robot_dart
- **Behavior Trees**
  - py_trees
  - BehaviorTree.CPP (συνοπτικά)
- **Βελτιστοποίηση Optimization**
  - ipopt
  - ifopt

Συγγράμματα

Προτεινόμενα:

- **Modern Robotics: Mechanics, Planning, and Control**, *Kevin M. Lynch and Frank C. Park*, 2017, Cambridge University Press. ebook

Από Εύδοξο:

- **Εισαγωγή στη Ρομποτική**, *J. Craig*, 2020, Τζιόλα
- **Πιθανοτική Ρομποτική**, *S. Thrun, W. Burgard, D. Fox*, 2011, Κλειδάριθμος
- **Ρομποτική**, *Δ. Εμίρης, Δ. Κουλουριώτης*, 2020, Τζιόλα

Διαμόρφωση Περιβάλλοντος

Για να παρακολουθήσετε επιτυχώς το μάθημα, θα πρέπει να:

- Έχετε ένα μηχάνημα (είτε physical είτε εικονικό) με διανομή Linux
  - Θα σας δώσουμε αναλυτικές οδηγίες για Ubuntu 20.04
  - Προτείνουμε επίσης ανεπιφύλακτα και Arch Linux
- Εξασκήσετε τις ικανότητές σας σε python (κυρίως) και C++
- Έχετε διάθεση για να γράψετε κώδικα, να διαβάσετε και να ψαχτείτε

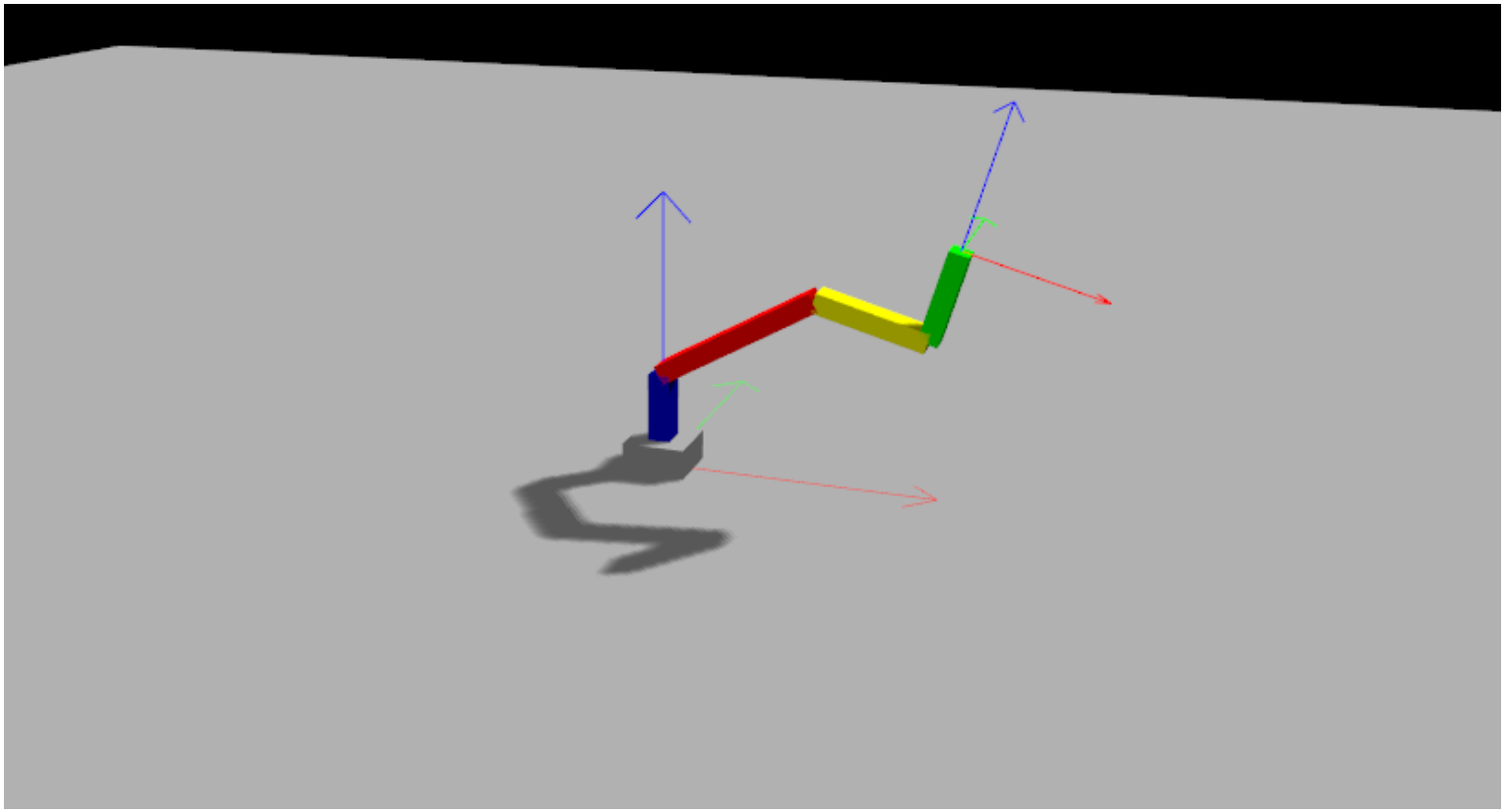
Συγγράμματα για Προγραμματισμό

Προτεινόμενα:

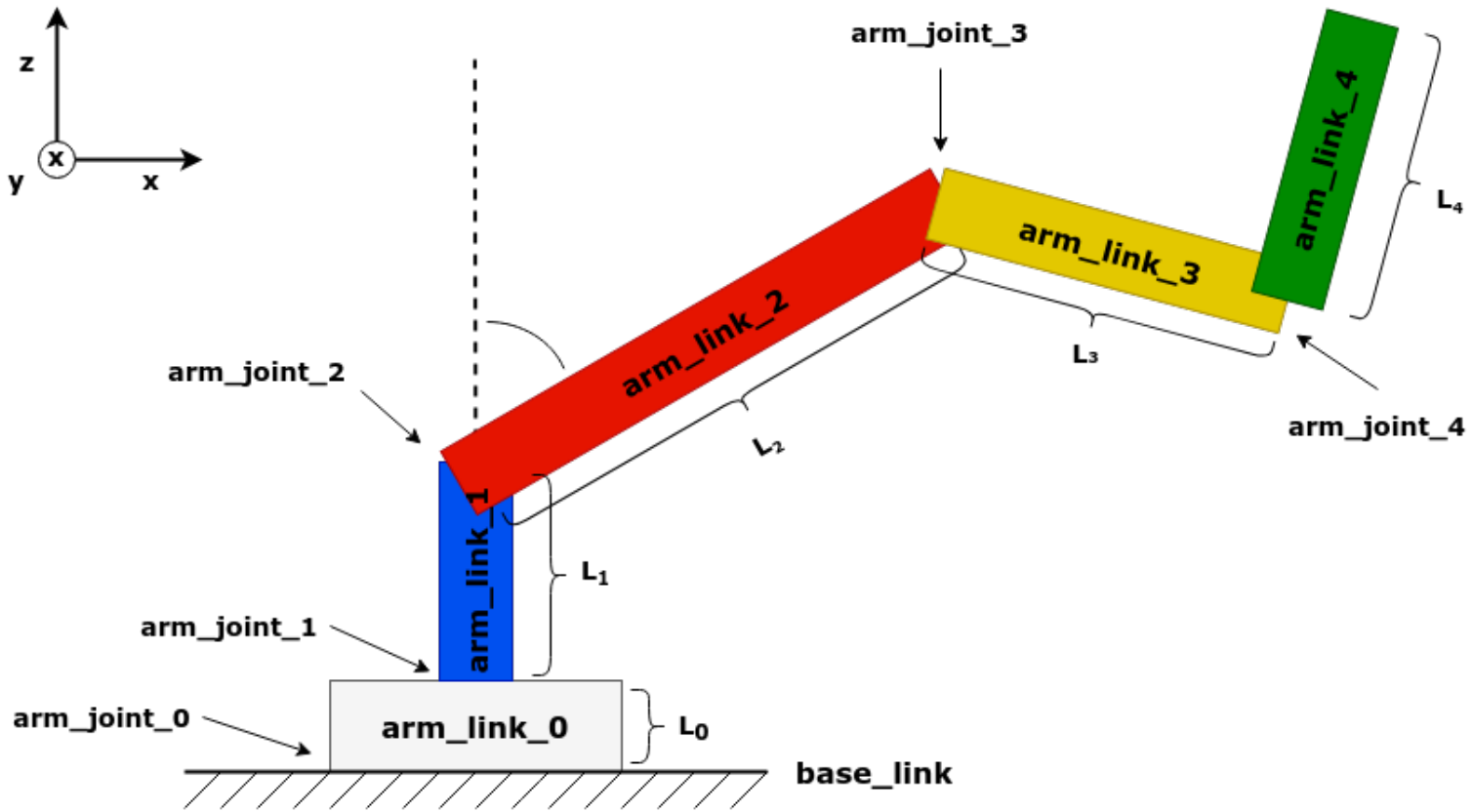
- **The Python Workbook**, *Ben Stephenson*, 2016, Springer International

- **Modern C++ Tutorial: C++11/14/17/20 On the Fly**, *Changkun Ou*, 2021. ebook

Παραδείγματα robot_dart (5)



Παραδείγματα robot_dart (5) - Σχεδιάγραμμα



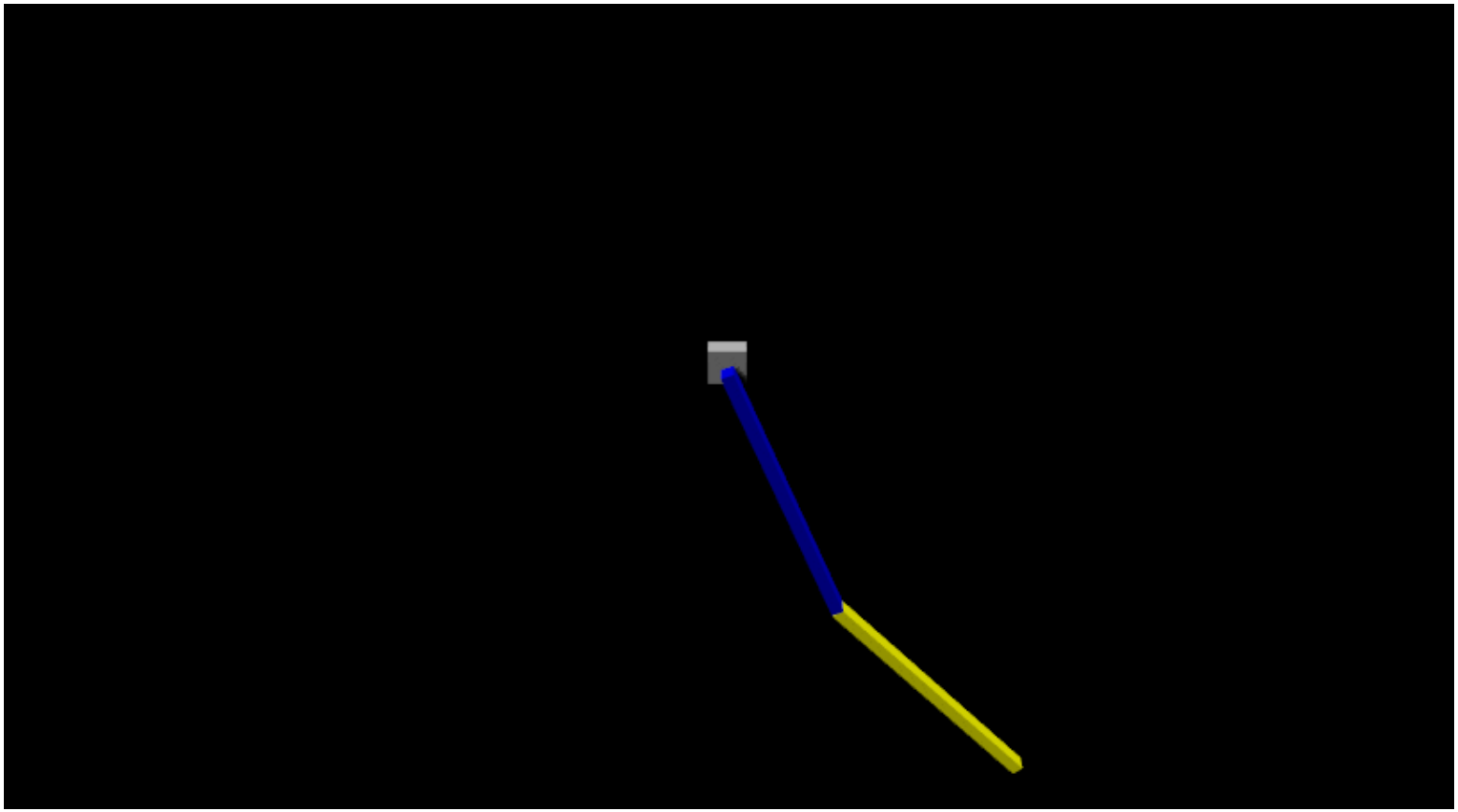
Παραδείγματα robot_dart (5) - Εξισώσεις

$$T_{01} = \begin{bmatrix} \cos\theta_0 & -\sin\theta_0 & 0 & 0 \\ \sin\theta_0 & \cos\theta_0 & 0 & 0 \\ 0 & 0 & 1 & L_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_{12} = \begin{bmatrix} \cos\theta_1 & 0 & \sin\theta_1 & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta_1 & 0 & \cos\theta_1 & L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{23} = \begin{bmatrix} \cos\theta_2 & 0 & \sin\theta_2 & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta_2 & 0 & \cos\theta_2 & L_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_{34} = \begin{bmatrix} \cos\theta_3 & 0 & \sin\theta_3 & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta_3 & 0 & \cos\theta_3 & L_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{45} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & L_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Παραδείγματα URDF (1)



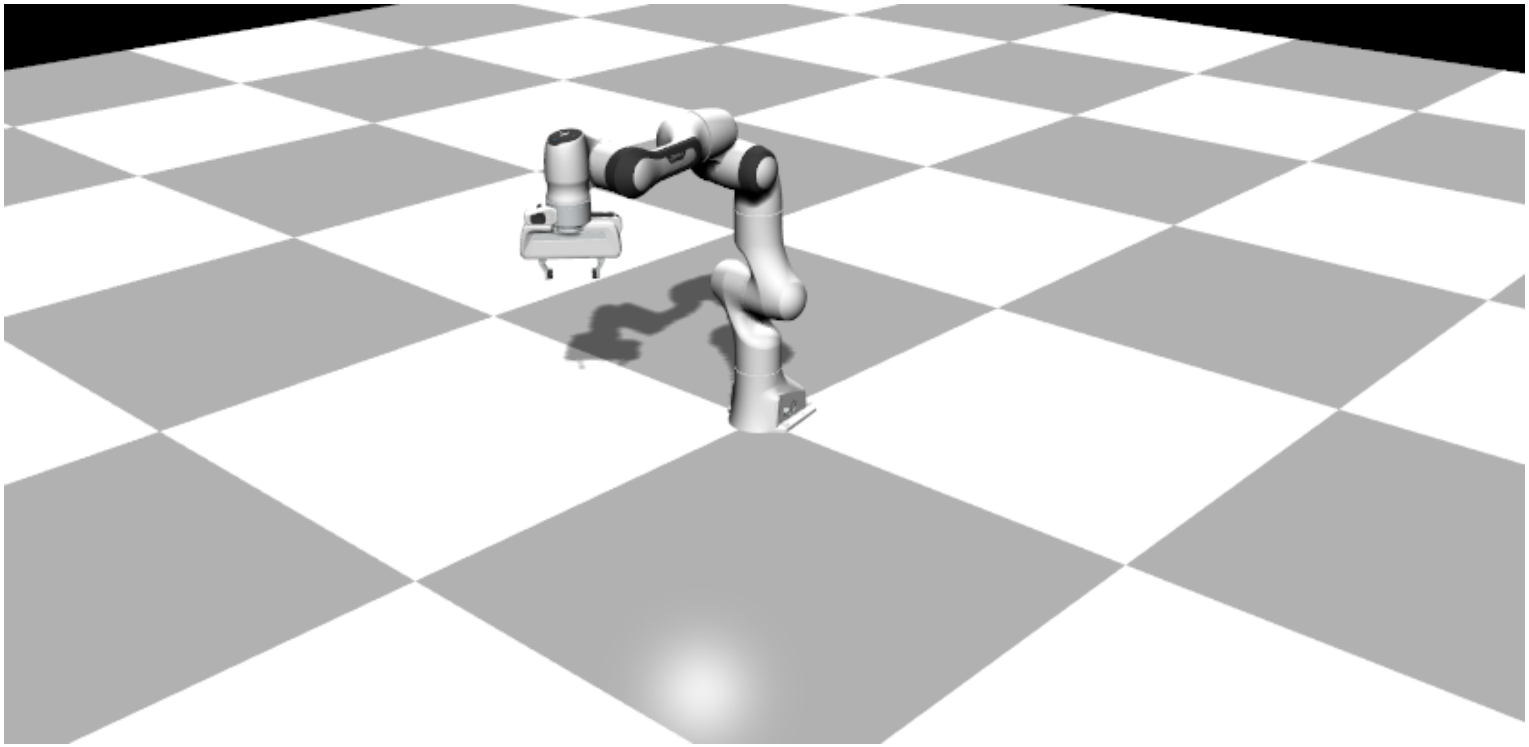
Παραδείγματα URDF (2) - Franka Panda

```
<link name="panda_link0">
    <visual>
        <geometry>
            <mesh filename="package://franka_description/meshes/visual/link0.dae" />
        </geometry>
    </visual>
    <collision>
        <geometry>
            <mesh filename="package://franka_description/meshes/collision/link0.stl" />
        </geometry>
    </collision>
    <inertial>
        <origin rpy="0 0 0" xyz="0 0 0" />
        <mass value="3.06" />
        <inertia ixx="0.3" ixy="0.0" ixz="0.0" iyy="0.3" iyz="0.0" izz="0.3" />
    </inertial>
</link>
```

C++ κώδικας:

```
std::vector<std::pair<std::string, std::string>> packages = {{"franka_description", "franka/franka_description"}};
auto robot = std::make_shared<robot_dart::Robot>("franka/franka.urdf", packages);
```

Παραδείγματα URDF (2) - Franka Panda



Ψευδο-αντίστροφοι πίνακες

- Αντίστροφοι Πίνακες (Inverse Matrix): $A^{-1}A = I$
- Ο αντίστροφος ορίζεται μόνο αν $A \in \mathbb{R}^{N \times N}$ (και όχι πάντα!)
- Οι Ιακωβιανοί πίνακες (Jacobians), δεν είναι τετραγωνικοί!
- Έστω $J \in \mathbb{R}^{m \times n}$, ο ψευδο-αντίστροφος πίνακας $J^{\dagger}$ ορίζεται ως:
  - $J^{\dagger} = J^T(JJ^T)^{-1},\ \text{αν}\ n > m, (J^{\dagger}J = I)$
  - $J^{\dagger} = (J^TJ)^{-1}J^T,\ \text{αν}\ n < m, (JJ^{\dagger} = I)$
  - Αν $n == m$, υπάρχει ο αντίστροφος!
- Damped Pseudoinverse:
  - $J^{\dagger} = J^T(JJ^T + \lambda^2 I)^{-1},\ \text{αν}\ n > m, (J^{\dagger}J = I)$
  - $J^{\dagger} = (J^TJ + \lambda^2 I)^{-1}J^T,\ \text{αν}\ n < m, (JJ^{\dagger} = I)$
  - $\lambda \in \mathbb{R}, \lambda > 0$

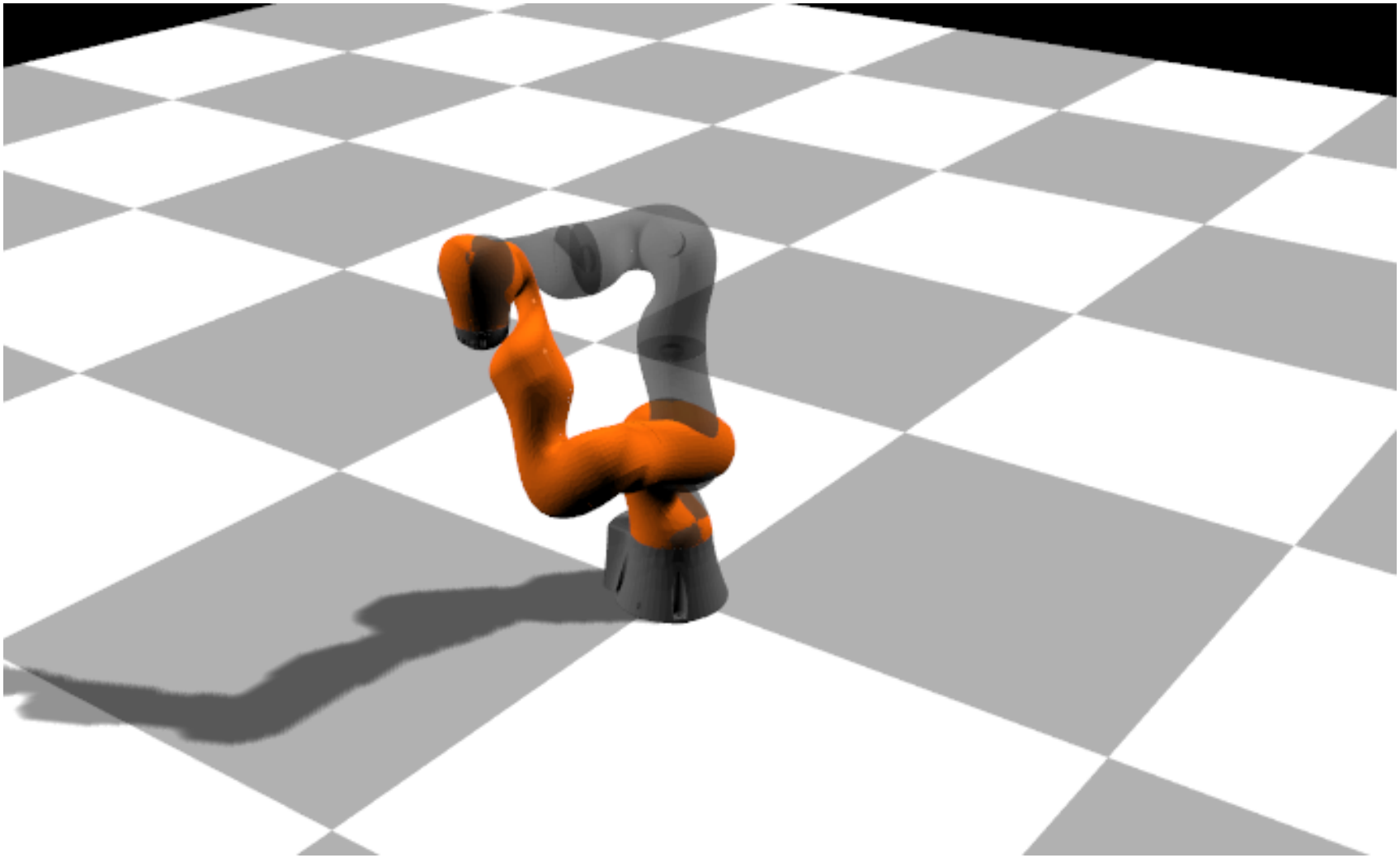
Inverse Kinematics ως Βελτιστοποίηση (1)

Αν έχουμε ένα μητρώο μετασχηματισμού για την θέση/προσανατολισμό του end-effector;
Έστω T_{wd} η θέση/προσανατολισμός στόχος, έχουμε τον εξής αλγόριθμο:

1. $\theta_i = \theta_0, i = 0$
2. $[\mathcal{V}_b] = \log(T_{wb}^{-1}(\theta_i) T_{wd})$, $T_{wb}(\theta)$ δίνει τα forward kinematics
3. $\Delta\theta = J_b^{\dagger} \mathcal{V}_b$, J_b είναι το body jacobian
4. $\theta_{i+1} = \theta_i + \Delta\theta$
5. Αν $||\mathcal{V}_b|| \approx 0$, τέλος, αλλιώς $i = i + 1$ και πίσω στο βήμα 2

Αντίστοιχα μπορούμε να κάνουμε τους υπολογισμούς στο world frame, με $[\mathcal{V}_w] = [Ad_{T_{wb}}]\mathcal{V}_b$ και χρησιμοποιώντας τον world jacobian, J_w.

Inverse Kinematics ως Βελτιστοποίηση (2)



Inverse Kinematics ως Βελτιστοποίηση (3)

Η μέθοδος με τον ψευδο-αντίστροφο έχει μερικά προβλήματα:

- Πρέπει το θ_0 να είναι κοντά σε καλή τιμή
- "Σπάει" όταν είμαστε σε singularities
  - όταν χάνει βαθμό/rank ο Ιακωβιανός πίνακας
  - όταν δηλαδή χάνουμε βαθμούς ελευθερίας στον end-effector
- Οι τελικές τιμές θ μπορεί να μην υπακούουν τα όρια του ρομπότ

Εναλλακτική μέθοδος:

- Ελαχιστοποίηση του λάθους
- $[\mathcal{V}_b] = \log(T_{wb}^{-1}(\theta_i)\, T_{wd})$ μας δίνει το σφάλμα!
- Θέλουμε να γίνει όσο το δυνατόν μικρότερο
- Άρα ελαχιστοποιούμε το $||\mathcal{V}_b||$ με όποιον αλγόριθμο βελτιστοποίησης θέλουμε
- Μπορούμε να βάλουμε όρια στην βελτιστοποίηση πιο εύκολα

Inverse Kinematics ως Βελτιστοποίηση (4)