Institut für Anthropomatik und Robotik (IAR)

High Performance Humanoid Technologies (H^2T)

Prof. Dr.-Ing. T. Asfour

Roboterpraktikum
Summer Term 2022

Exercise 6:

Constrained Inverse Kinematics

Advisor:

Jianfeng Gao

Abstract: *This exercise aims at the implementation of methods for solving inverse kinematics (IK) problems. Starting from the unconstrained IK problem we proceed towards solving IK problems while satisfying a set of additional constraints. The exercises are implemented in C++ using the Simox robotic toolbox.*

Contents

1 Introduction

In a general sense, a robot is a combination of connected joints. This *kinematic structure* includes one or more chains of joints that hierarchically influence each other. These *kinematic chains* start from a *root joint* which is not influenced by any other joint and end with an *end effector*. The end effector is the final point of a kinematic chain and is influenced by all joints in the chain. Fig. 1 depicts an exemplary kinematic chains in the kinematic structure of the humanoid robot ARMAR-III.

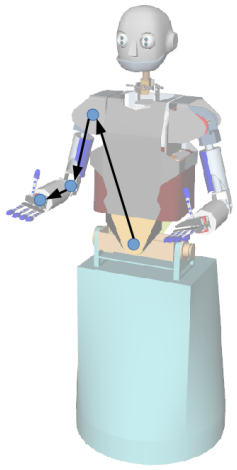


Figure 1: Exemplary kinematic chain in the kinematic structure of the humanoid robot ARMAR-III.

The first question that naturally arises when regarding robots as a combination of kinematic chains is:

> *Given the current joint configuration, which position and orientation does the end effector have?*

This problem is called *Forward Kinematics* (FK) and can be solved straightforwardly on the basis of the kinematic equations of the robot.
While computing forward kinematics is relatively easy, the inverse problem is usually much harder. The underlying question of the problem of *Inverse Kinematics* (IK) is:

> *How do the joint angles have to be set in order to realize a desired position and orientation of the end effector?*

If the robot has less degrees of freedom, i.e. less joints, than the workspace, it might be possible to find a closed-form solution for the IK problem. In the case of humanoid robotics however, the number of joints is usually much larger than the number of degrees of freedom in the workspace. In these cases the kinematic equations cannot be inverted, yielding a general optimization problem.
Several approaches for solving the IK problem exist and will be discussed in Sec. 2.3. Implementing some of these techniques is part of this exercise. However, solving an IK problem is usually not sufficient for controlling a humanoid robot. Several additional constraints have to be satisfied besides moving an end effector to a target pose, e.g.:

- The robot must to be in balance (see Sec. 2.4).
- The robot must not collide with obstacles.
- The joints must not be moved past their individual limits.

As the desired end effector pose can also be regarded as one of the constraints to satisfy, the question of *constrained IK* evolves to:

> *How do the joint angles have to be set in order to satisfy a chosen set of constraints.*

In this exercise you will get a practical insight into the solution of constrained IK problems. Sec. 2 will explain the theory necessary in order to understand and solve the exercises you find in Sec. 3.

2 Fundamentals

This section creates the foundation for the following practical exercises. It is meant to be read thoroughly and questions concerning the contents of this section will be asked. You can also refer to [2] for another introduction into the topic.

2.1 Definitions

$SE(3)$ The *Special Euclidean Group* is the group of all rotations and translations in $\mathbb{R}^3$. These transformations are usually described in terms of homogeneous matrices.

2.2 Forward Kinematics

As stated in the introduction, *forward kinematics* (FK), describes the problem of determining the end effector pose $\mathbf{p} \in SE(3)$ based on a given joint configuration $\theta \in \mathbb{R}^n$. For solving the forward kinematics problem, consider a robot to be a concatenation of joints $j_1, \ldots, j_n$ with limbs in between. Each joint has a local coordinate frame and a transformation matrix ${}^{i-1}\mathbf{T}_i(\theta_i) \in SE(3)$ that maps from the frame of joint i to the frame of joint $i-1$, depending on the joint angle θ_i. These matrices ${}^{i-1}\mathbf{T}_i(\theta_i)$ are known as *Denavit-Hartenberg matrices*.

Computing the forward kinematics is equivalent to transforming the origin of the end effector coordinate frame to the base coordinate frame. Assuming the end effector to be the k-th joint in the kinematic chain, this can be done by concatenating the Denavit-Hartenberg matrices up to joint k:

$$\mathbf{p} = {}^{0}\mathbf{T}_k(\theta) = \prod_{i=1}^{k} {}^{i-1}\mathbf{T}_i(\theta_i) \tag{1}$$

As soon as the kinematic structure of the robot is known, e.g. due to a robot model, computing forward kinematics solutions becomes as simple as multiplying a list of matrices. As usual, problems become hard when being looked at the other way around.

2.3 Inverse Kinematics

Solving the inverse kinematics (IK) problem, i.e. determining a suitable set of joint angles θ for which the end effector pose $\mathbf{p}(\theta)$ equals a target pose $\mathbf{p}_{\text{goal}} \in SE(3)$ is equivalent to solving Eq. 1 for θ with given $\mathbf{x}$. As the joint angle vector $\theta \in \mathbb{R}^n$ usually has a much larger dimension than the six dimensional workspace, the equation system has fewer equations than unknowns: it is underdetermined. This means that the IK problem can have infinitely many solutions as soon as the kinematic chain contains more than 6 joints. An example for such a situation is a human arm grasping an object with a fixed hand pose. Although the end effector pose is fixed, humans can usually still move the elbow yielding infinitely many solutions to the grasping problem.

Solving the Inverse Kinematics

The IK problem yields an underdetermined equation system with which we would be fine if it was a *linear equation system*. However, unfortunately, the general form of the Denavit-Hartenberg matrices that construct the equation system looks as follows:

$$ {}^{n-1}\mathbf{T}_n = \begin{pmatrix} \cos\theta_n & -\sin\theta_n \cos\alpha_n & \sin\theta_n \sin\alpha_n & a_n \cos\theta_n \\ \sin\theta_n & \cos\theta_n \cos\alpha_n & -\cos\theta_n \sin\alpha_n & a_n \sin\theta_n \\ 0 & \sin\alpha_n & \cos\alpha_n & d_n \\ 0 & 0 & 0 & 1 \end{pmatrix} . \quad (2) $$

This means that the IK problem, i.e. solving Eq. 1 for θ, is a highly non-linear system. There are some approaches for solving non-linear equation systems, e.g. Newton's Method, but these methods usually require the derivatives to be explicitely known. Although we could algebraically compute the derivative of ${}^0\mathbf{T}_k(\theta)$, i.e. the *Jacobian*, it would only be valid for one specific kinematic chain of one specific robot. As in robotics we usually deal with many different kinematic chains on many different robots, more general strategies have to be considered.

Approximating Jacobians

The Jacobian $\mathbf{J}(\theta)$ is the derivative of ${}^0\mathbf{T}_k(\theta)$, which enables us to incorporate Newton's Method or simple gradient descent strategies for solving the IK

problem:

$$\mathbf{J}(\theta) = \begin{pmatrix} \frac{\partial f_1}{\partial \theta_1}(\theta) & \frac{\partial f_1}{\partial \theta_2}(\theta) & \dots & \frac{\partial f_1}{\partial \theta_n}(\theta) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial \theta_1}(\theta) & \frac{\partial f_m}{\partial \theta_2}(\theta) & \dots & \frac{\partial f_m}{\partial \theta_n}(\theta) \end{pmatrix} \tag{3}$$

As stated above, computing the actual derivatives of ${}^0\mathbf{T}_k(\theta)$, i.e. computing the actual Jacobian $\mathbf{J}(\theta)$ is not feasible. However, we can easily compute a linear approximation of the Jacobian that is valid within a small radius around the current joint configuration θ:

$$\frac{\partial f_i}{\partial \theta_j} \approx \frac{f_i(\theta_i + \delta) - f_i(\theta)}{\delta} \tag{4}$$

Using this Jacobian approximation, we can derive a relation between small joint angle pertubations ($\Delta\theta$) and their respective changes to the end effector pose ($\Delta\mathbf{x}$):

$$\mathbf{J} \cdot \dot{\theta} = \dot{\mathbf{x}} \quad \Rightarrow \quad \mathbf{J} \cdot \Delta\theta = \Delta\mathbf{x} \tag{5}$$

Iterative Gradient Descent

We have computed a linear approximation of the Jacobian $\mathbf{J}(\theta)$, which is valid within a certain radius around the current joint configuration θ. We can now take a step towards the workspace goal $\Delta\mathbf{x} = \delta \cdot (\mathbf{x}_{\text{goal}} - \mathbf{x}_{\text{current}})$ and solve:

$$\mathbf{J} \cdot \Delta\theta = \Delta\mathbf{x} \tag{6}$$

for $\Delta\theta$. As $\mathbf{J} \in \mathbb{R}^{n \times 6}$ is a non-squared matrix, we use the *Moore-Penrose Pseudoinverse* $\mathbf{J}^{\#} = \mathbf{J}^T(\mathbf{J}\mathbf{J}^T)^{-1}$ in order to compute the best solution in terms of least squares:

$$\Delta\theta = \mathbf{J}^{\#} \cdot \Delta\mathbf{x} \tag{7}$$

In order to solve the full IK problem, consecutive steps of gradient descent have to be performed as shown in Alg. 1. This algorithm will be implemented in Exercise 1 (Section 3).

Algorithm 1: Iterative Gradient Descent

input : Initial joint configuration $\theta \in \mathbb{R}^n$
input : Workspace tolerance ε
input : Displacement factor δ
output: IK solution $\theta \in \mathbb{R}^n$

```
while NotStalledOrFailed() do
    Δx ← getDisplacement(θ)
    if ‖Δx‖ < ε then
        return θ
    else
        J ← getJacobian(θ)
        θ ← θ + J^# · δ · Δx
```

2.4 Constraints

Being able to solve the IK problem for individual constraints on end effector pose is nice, but not sufficient. A usual real-world IK problem consists of several constraining conditions, one of which being the desired end effector pose. In this section we are going to have a look on how to formalize different types of constraints in a way Alg. 1 can handle.

End Effector Constraints

Specifying a distinct target pose for the end effector is the most constraining type of end effector pose constraint. It is a special case of the *Task Space Region Constraint* [1], which constraints the end effector pose to a transformed interval in workspace.
A Task Space Region (TSR) is defined through the matrix $\mathbf{B}^w \in \mathbb{R}^{6\times 2}$, constraining position and orientation of the end effector in each workspace dimension (x, y, z, roll, pitch and yaw):

$$\mathbf{B}^w = \begin{pmatrix} x_{\min} & x_{\max} \\ y_{\min} & y_{\max} \\ z_{\min} & z_{\max} \\ \psi_{\min} & \psi_{\max} \\ \theta_{\min} & \theta_{\max} \\ \phi_{\min} & \phi_{\max} \end{pmatrix} \tag{8}$$

In addition the TSR can be transformed by a matrix $\mathbf{T}_w^0 \in SE(3)$ and an end effector offset $\mathbf{T}_e^w \in SE(3)$ is respected. The only requirement in order to make TSR constraints applicable in terms of Alg. 1 is the computation of a displacement vector $\Delta\mathbf{x}$ from the current end effector pose to the TSR, which is a straight forward projection to the implied workspace interval.

Stability

One very important constraint to consider when computing IK solutions is *static stability*, i.e. restricting the solution space to the set of configurations that are statically stable. Static stability means that the robot will not fall over when having achieved the desired configuration, while neglecting all dynamics. The fundamental information for static stability is the robot's center of mass x_{com}:

$$x_{com} = \frac{1}{\sum_{i=1}^{N} m_i} \sum_{i=1}^{N} m_i \cdot x_{com_i} \tag{9}$$

which is projected to the ground plane:

$$\Delta x = x_{goal} - \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \cdot x_{com} \tag{10}$$

A robot configuration is statically stable when the projection of the robot's center of mass lies within the *support polygon*, spanned by the contacts between the robot and the ground plane (see Figure 2). For moving the center of mass we can use *COG Jacobians* [3]:

$$J = \frac{1}{\sum_{i=1}^{N} m_i} \sum_{i=1}^{N} m_i \cdot J_{l_i}(x_{com_i}) \tag{11}$$

2.5 Simultaneous Satisfaction of Multiple Constraints

There are two elementary strategies for satisfying multiple kinematic constraints simultaneously: Stacking of Jacobians and the nullspace projection.

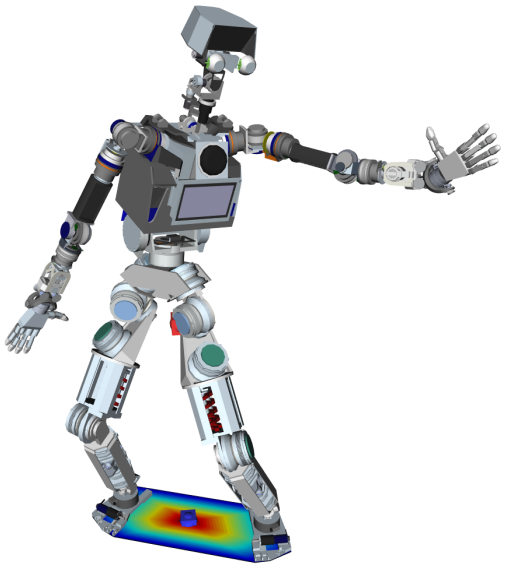


Figure 2: The support polygon of ARMAR-4 together with the center of mass x_{com} (red box) and its projection (blue box). The displayed robot configuration is in perfect balance.

Stack of Jacobians

The kinematic equation $\mathbf{J} \cdot \dot{\theta} = \dot{\mathbf{x}}$ that we solved for $\dot{\theta}$ in the previous sections is already an underdetermined equation system. One approach for encoding multiple constraints in the system is to simply add them to the equation system and solve them as if it was one single constraint. For that we need to stack the respective Jacobians $\mathbf{J}_1, \ldots, \mathbf{J}_N$ and error vectors $\Delta\mathbf{x}_1, \ldots, \Delta\mathbf{x}_N$:

$$\begin{bmatrix} \mathbf{J}_1 \\ \vdots \\ \mathbf{J}_N \end{bmatrix} \cdot \dot{\theta} = \begin{bmatrix} \Delta\mathbf{x}_1 \\ \vdots \\ \Delta\mathbf{x}_N \end{bmatrix} \tag{12}$$

Using this formulation we can again use Alg. 1 for solving multiple constraints at once. You will implement this strategy in Exercise 3.

Nullspace Projection

One comfortable property of the Moore-Penrose Pseudoinverse $\mathbf{J}^{\#}$ is that the Matrix $(\mathbf{I} - \mathbf{J}^{\#}\mathbf{J})$ performs a projection onto the nullspace of $\mathbf{J}$. The nullspace

of $\mathbf{J}$ is the space of joint velocities $\dot{\theta}_{\text{null}}$ that $\mathbf{J}$ maps to zero, i.e. $\mathbf{J} \cdot \dot{\theta}_{\text{null}} = 0$. Figuratively speaking, these joints do not influence the end effector in a way that is important to the imposed constraint. We can therefore perform a second gradient descent towards another constraint in the nullspace of $\mathbf{J}$ using only the leftover joints of the primary constraint. This is done by altering the rule of gradient descent in Alg. 1 to:

$$\theta \leftarrow \theta + \mathbf{J}_1^{\#} \cdot \delta_1 \cdot \Delta\mathbf{x}_1 + (I - \mathbf{J}_1^{\#}\mathbf{J}_1) \cdot \mathbf{J}_2^{\#} \cdot \delta_2 \cdot \Delta\mathbf{x}_2 \tag{13}$$

You will implement this strategy in Exercise 4.

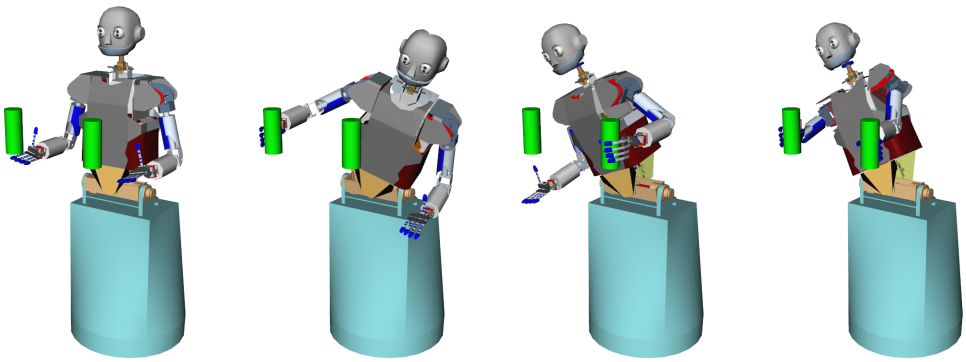
3 Exercises

Exercise 0: Theory (Homework)

Read and understand the theoretical backgrounds for constrained inverse kinematics outlined in Sec. 1 and Sec. 2. Try to answer the following questions:

1. What is the problem of *inverse kinematics* and why is it hard?
2. Explain the function of the Jacobian **J**.
3. How can we satisfy more than one constraint at once?

For the following practical exercises you will be provided a lab PC with the exercise sources already installed. Your instructor will give you a short introduction into building and running the code. Figure 3 depicts the initial demo scene together with possible solutions to the following exercises.



(a) (b) (c) (d)

Figure 3: The initial scene (a) and possible solutions to the exercises (b)-(c).

Exercise 1: Gradient Descent

Implement the gradient descent method (see Alg. 1) for one constraint in the code frame given in `MySingleConstraintIK.cpp`. You will find all necessary information regarding Simox or Eigen either as comments in the source code or in the coding tutorial of this lab.

Exercise 2: End-Effector Target Pose

Define the end-effector target pose for grasping the second object using the left hand. The code needs to be entered in `exercise2GetTSRBounds` in `ConstrainedIKWindow.cpp`.

Exercise 3: Constrained IK by Jacobian Stacking

Implement the Jacobian stacking method for solving multiple kinematic constraints simultaneously discussed in Sec. 2.5. Use the code frame provided in the function `solveStackedIkStep` in `MyDualConstraintIK.cpp`.

Exercise 4: Constrained IK by Nullspace Projection

Implement the nullspace projection method for solving multiple kinematic constraints simultaneously discussed in Sec. 2.5. Use the code frame provided in the function `solveNullspaceIkStep` in `MyDualConstraintIK.cpp`.

Compare the two approaches implemented in Ex. 3 and Ex. 4:

1. Do you see differences in the computed IK solutions?
2. Can you explain what you observe?

Exercise 5: Custom Constraint

During the previous exercises you should have noticed that reaching for distant objects sometimes risks the robot's static stability. Implement a stability constraint that attempts to maintain balance. Use the code frame provided in the functions `getJacobianMatrix` and `getError` in `MyCustomConstraint.cpp`.

References

[1] Dmitry Berenson, Siddhartha Srinivasa, and James Kuffner. Task space regions: A framework for pose-constrained manipulation planning. *International Journal of Robotics Research (IJRR)*, 30(12):1435 – 1460, October 2011.

[2] Samuel R Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. *IEEE Journal of Robotics and Automation*, 17:1–19, 2004.

[3] Tomomichi Sugihara, Yoshihiko Nakamura, and Hirochika Inoue. Realtime Humanoid Motion Generation through ZMP Manipulation based on Inverted Pendulum Control. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1404–1409, 2002.