Humanoid Robotics

Inverse Kinematics and Whole-Body Motion Planning

Maren Bennewitz
Goal of this Chapter

- How to calculate joint angles in order to reach a desired end-effector pose
- How to efficiently solve complex, high-dimensional motion planning problems, including whole-body motions
- How to determine the optimal stance pose to reach a desired end-effector pose
Motivation

- Whole-body motion to reach a desired goal configuration
- Planning for object manipulation
- Needed: sequence of robot configurations (=joint angle trajectories)

grabbing an object

opening a drawer
Example Planning Problems

- Two plans for a pick-and-place task:

- Hard planning problem due to local minima in the search space:
Recap: Forward Kinematics (FK)

- FK computes the **end-effector pose given the current joint encoder readings**
- Using a chain of transformation matrices (see Ch. 5)
- Example: Transformation $\mathcal{F}_E^B(q)$ from the left end-effector frame to the robot’s torso frame given the encoder readings $q$
Inverse Kinematics (IK)

- IK computes the **joint angle values so that the end-effector reaches a desired pose**
- Inverse of the forward kinematics problem
  - FK: $e = \mathcal{P}(q)$
  - IK: $q = \mathcal{P}^{-1}(e)$
- IK is challenging and cannot be as easily computed as FK
- It might be that there exist several possible solutions, or there may be no solution at all
Inverse Kinematics (IK)

- Many different approaches to solving IK problems exist
- Analytical methods: closed-form solution (e.g., for legs as in previous lecture)
- Numerical methods: iteratively calculate a sequence of configurations that approach target pose
Analytical IK Methods

- Closed-form solution (using trigonometry, geometry)
- Computes all IK solutions, determines whether or not a solution exists
- Once the equations are derived, solutions are very fast to compute
- No need to define solution parameters or initial guesses
- However, often difficult to define, must be derived independently for robots with different kinematic structures
Numerical IK Methods

- Given a start configuration, iteratively calculate a sequence of configurations to reach end-effector pose
- Use the Jacobian
- Converge to a solution
- Usually much slower but also more general
Inverse Kinematics Solvers

- **Analytical solver** (fast, calculates all possible solutions)
  - IKFast:
    http://openrave.org/docs/latest_stable/openravepy/ikfast

- **Numerical methods**
  - Kinematics and Dynamics Library (KDL)
    http://wiki.ros.org/kdl
  - trac_ik
    https://bitbucket.org/traclabs/trac_ik/src/master/
  - bio_ik
    https://github.com/TAMS-Group/bio_ik
Inverse Kinematics: Example

- Consider a simple 2D robot arm with two 1-DOF rotational joints
- Given a desired end-effector pose $e$
- Compute joint angles $q_1$ and $q_2$

$e = (e_x, e_y)$

Diagram: A 2D robot arm with two joints $q_1$ and $q_2$, and an end-effector at $(e_x, e_y)$. The arm is represented as a line with one joint at the base and another at the end, connected by two joints. The end-effector is shown as a red dot at the end of the arm.
Numerical Approach Using the Jacobian: Example

If we increased $q_1$ by a small amount, what would happen to $e$?

$e = (e_x, e_y)$
Numerical Approach Using the Jacobian: Example

If we increased $q_2$ by a small amount, what would happen to $e$?

\[ e = (e_x, e_y) \]
Numerical Approach Using the Jacobian

- Jacobian matrix for the simple example

\[
J(e, q) = \begin{pmatrix}
\frac{\partial e_x}{\partial q_1} & \frac{\partial e_x}{\partial q_2} \\
\frac{\partial e_y}{\partial q_1} & \frac{\partial e_y}{\partial q_2}
\end{pmatrix}
\]

- The Jacobian defines how each component of e changes w.r.t. joint angle changes.

- For any given vector of joint values, we can compute the components of the Jacobian.
Numerical Approach Using the Jacobian

- Usually, the Jacobian will be an \( 6 \times N \) matrix where \( N \) is the number of joints.
- The Jacobian can be computed based on the equations of FK.
- See exercise.
Numerical Approach Using the Jacobian

- Given a desired incremental change in the end-effector configuration, we can compute the corresponding incremental change of \( \mathbf{q} \):

\[
J \Delta \mathbf{q} = \Delta \mathbf{e}
\]

\[
\Delta \mathbf{q} = J^{-1} \Delta \mathbf{e}
\]

- As \( J \) cannot be inverted in the general case, it is replaced by the pseudoinverse or by the transpose in practice
Numerical Approach Using the Jacobian

- Forward kinematics is a nonlinear function
- Thus, we have an approximation that is only valid near the current configuration
- Until the end-effector is close to the desired pose, repeat:
  - Compute the Jacobian
  - Take a small step towards the goal
End-Effector Goal and Step Size

- Let $e$ represent the current end-effector pose and $g$ represent its desired goal pose.
- Choose a value for $\Delta e$ that will move $e$ closer to $g$, theoretically:

$$\Delta e = g - e$$

- The nonlinearity prevents the end-effector to reach the goal exactly.
- To avoid oscillation, take a smaller step:

$$\Delta e = \alpha(g - e), \quad 0 \leq \alpha \leq 1$$
Basic Jacobian IK Technique

while (\( e \) is too far from \( g \)) {
    Compute \( J(e, q) \) for the current configuration \( q \)
    Compute \( J^{-1} \)
    \( \Delta e = \alpha(g - e) \)  // choose a step to take
    \( \Delta q = J^{-1}\Delta e \)  // compute required change in joints
    \( q = q + \Delta q \)  // apply change to joints
    Compute resulting \( e \)  // by FK
}
Limitations of the IK-based Approach

- For local motion generation problems, IK-based methods can be applied
- Numerical optimization methods, however, bear the risk of being trapped in local minima
- IK generates unpredictable joint configurations
- For more complex problems, e.g., for collision-free motions in narrow environments, planning methods have to be applied
Whole-Body Motion Planning

- Find a path in a high-dimensional configuration space
- Consider constraints such as avoidance of joint limits, self- and obstacle collisions, and balance
- Complete search algorithms are not tractable
- Apply a randomized, sampling-based approach to find a valid sequence of configurations
- Implementation of sampling-based planners
  - https://ompl.kavrakilab.org/planners.html
  - https://moveit.ros.org/
Rapidly Exploring Random Trees (RRTs)

- General idea: Explore the configuration space by expanding incrementally from an initial configuration.
- The explored space corresponds to a tree rooted at the initial configuration.
- Basic principle: Sample configuration and compute local connection to nearest neighbor.

![Diagram showing exploration process from 45 iterations to 2345 iterations.](image)
RRTs: General Algorithm

Given: Configuration space $C$ and initial configuration $q_0$

Algorithm 1: RRT

1. $G$.init($q_0$)
2. repeat
3. $q_{rand} \rightarrow \text{RANDOM\_CONFIG}(C)$
4. $q_{near} \leftarrow \text{NEAREST}(G, q_{rand})$
5. $G$.add_edge($q_{near}, q_{rand}$)
6. until condition
RRTs: General Algorithm

Given: Configuration space $C$ and initial configuration $q_0$

Algorithm 1: RRT

1. $G$.init($q_0$)
2. repeat
3. $q_{\text{rand}} \rightarrow \text{RANDOM\_CONFIG}(C)$
4. $q_{\text{near}} \leftarrow \text{NEAREST}(G, q_{\text{rand}})$
5. $G$.add_edge($q_{\text{near}}, q_{\text{rand}}$)
6. until condition

Find closest vertex in $G$ using a distance function

$\rho : C \times C \rightarrow [0, \infty)$
RRTs: General Algorithm

Given: Configuration space $C$ and initial configuration $q_0$

**Algorithm 1: RRT**

$$
\begin{align*}
1 & \quad G.\text{init}(q_0) \\
2 & \quad \text{repeat} \\
3 & \quad \quad q_{\text{rand}} \rightarrow \text{RANDOM\_CONFIG}(C) \\
4 & \quad \quad q_{\text{near}} \leftarrow \text{NEAREST}(G, q_{\text{rand}}) \\
5 & \quad \quad G.\text{add\_edge}(q_{\text{near}}, q_{\text{rand}}) \\
6 & \quad \text{until condition}
\end{align*}
$$

Connect $q_{\text{rand}}$ with $q_{\text{near}}$ using a **local planner**:
- No collision: add edge
RRTs: General Algorithm

Given: Configuration space $C$ and initial configuration $q_0$

**Algorithm 1: RRT**

1. $G$.init($q_0$)
2. repeat
   3. $q_{rand} \rightarrow \text{RANDOM\_CONFIG}(C)$
   4. $q_{near} \leftarrow \text{NEAREST}(G, q_{rand})$
   5. $G$.add_edge($q_{near}, q_{rand}$)
3. until condition

Connect $q_{rand}$ with $q_{near}$ using a **local planner**:

- No collision: add edge
- Collision: new vertex is $q_s$, as close as possible to $C_{obs}$
RRTs to Reach a Given Goal Configuration

- Extend the tree (i.e., generate $q_{new}$) with a fixed step size towards $q_{rand}$
- Terminate when $q_{new}$ is close to the desired $q_{goal}$
RI 16-735, Howie Choset with slides from James Kuffner

Path Planning with RRTs

(Rapidly-Exploring Random Trees)

\[
\text{BUILD}_\text{RRT}\ (q_{\text{init}})\ {\text{for} \ k = 1 \ to \ K \ do} \\
\text{q}_{\text{rand}} = \text{RANDOM_CONFIG}() \\
\text{EXTEND}\ (T, \ \text{q}_{\text{rand}}) \\
\text{q}_{\text{near}} \ \text{q}_{\text{new}} \\
\text{q}_{\text{new}} \ \text{q}_{\text{init}} \\
\text{q}_{\text{rand}} \\
\text{The algorithm terminates when } q_{\text{new}} \text{ is near the goal}
\]

[RRTs: Tree Extension with a Small, Fixed Step Size]

[Kuffner&Lavalle, ICRA '00]
Bias Towards the Goal

- During tree expansion, pick the goal instead of a random node with some probability (5-10%) 
- Why not picking the goal at each iteration? 
- Avoiding running into local minima (due to obstacles or other constraints) instead of exploring the space
Bidirectional RRTs

- Some problems require more effective methods: bidirectional search
- Grow **two** RRTs, one from $q_I$ and one from $q_G$
- In every other step, try to extend each tree towards $q_{new}$ of the other tree
RRT-Connect: Basic Concept

- Grow two trees: from start and end node (start and goal configurations of the robot)
- Pick a random configuration: $q_{\text{rand}}$
- Find the nearest node in one tree: $q_{\text{near}}$
- Extend the tree from the nearest node by taking a step towards the random node to get $q_{\text{new}}$
- Extend the other tree towards that $q_{\text{new}}$ from nearest node in the tree
- Return the solution path when the distance between $q_{\text{new}}$ and the nearest node in the second tree is close enough
RRT-Connect: Example Path

Return path connecting start and goal

$q_{init}$

$q_{goal}$
Extend Function

Returns

- **Trapped**: Not possible to extend the tree due to collisions or constraints

- **Extended**: Performed a step from $q_{near}$ towards $q_{rand}$, generated $q_{new}$

- **Reached**: Trees connected, path found
RRT-Connect

RRT\_CONNECT \((q_{init}, q_{goal})\) \{
\begin{align*}
T_a &. \text{init} (q_{init}); \quad T_b &. \text{init} (q_{goal}); \\
\text{for } k = 1 \text{ to } K \text{ do} \\
q_{rand} & = \text{RANDOM\_CONFIG}(); \\
\text{if not } (\text{EXTEND}(T_a, q_{rand}) = \text{Trapped}) & \text{ then} \\
\text{if } (\text{EXTEND}(T_b, q_{new}) = \text{Reached}) & \text{ then} \\
& \text{ Return PATH}(T_a, T_b); \\
& \text{SWAP}(T_a, T_b); \\
& \text{Return Failure;}
\end{align*}
\}

Instead of switching, use \(T_a\) as smaller tree. This helped James a lot.
RRTs – Properties (1)

- Easy to implement
- Fast
- Good balance between greedy search and exploration
- Produce non-optimal paths: solutions are typically jagged and may be overly long
- Post-processing such as smoothing is necessary
- Generated paths are not repeatable and unpredictable
- Rely on a distance metric (e.g., Euclidean)
RRTs – Properties (2)

- Probability of finding a solution if one exists approaches 1 (probabilistic completeness)
- Unknown rate of convergence
- When there is no solution (path is blocked due to obstacles or other constraints), the planner may run forever
- To avoid endless runtime, the search is stopped after a certain number of iterations
Considering Constraints for Humanoid Motion Planning

- When randomly sampling configurations, most of them will not be valid since they cause the robot to lose its balance.
- Use a set of predetermined statically stable double support configurations from which to sample $q_{rand}$.
- In the extend function: Check $q_{new}$ for joint limits, self-collision, collision with obstacles, and whether it is statically stable.
Collision Checking

- FCL library for collision checks
  https://github.com/flexible-collision-library/fcl

- Check the mesh model of the robot for self-collisions and collisions with the environment
RRT-Connect: Considering Constraints

- Apply RRT-Connect
- Smooth path after a solution is found
Plan Execution: Pick and Place
Plan Execution: Grabbing Into a Cabinet
RRT-Connect: Parameters

- Database of 463 statically stable double support configurations, generated within 10,000 iterations
- Low probability of generating valid configurations, when sampling completely at random during the search
- Maximum number of iterations $K$ in RRT-Connect: 3,000
- Step size for generating the new configuration during the extension: $5^\circ$ for each joint
Example Results: 100 Planning Trials

- Expanded nodes upper / lower shelf: $19.84 \pm 30.06 / 1164.89 \pm 98.99$

- Unsuccessful planning attempts possible, depending on the chosen parameters
Stance Selection

- How to actually determine the goal configuration?
- Goal: Find the best robot pose for a given grasping pose

source: T. Asfour
Spatial Distribution of the Reachable End-Effector Poses

- Reachability map as representation of the robot’s reachable workspace
- Stored as a voxel grid
- Generated offline, only once
- Represents possible end-effector poses and quality information
Reachability Map (RM)

- Constructed by systematic **sampling joint configurations** of a kinematic chain
- Apply **FK** to determine the **corresponding voxel** containing the end-effector pose
Reachability Map (RM)

- Configurations are added to the RM if they are statically-stable and self-collision free
- Result: Representation of reachability, each voxel contains configurations and a corresponding quality measure
- Generating the RM is time-consuming, but needs to be done only once offline
Sampling of Configurations

- Discrete steps through the range of the joint angles of the kinematic chain
- Consider a serial chain: joints between the right foot and the gripper link
Generation of Double Support Configurations

- Using active-passive link decomposition and IK
- Given the hip pose and the desired foot pose, we can solve IK for the passive chain
Measurement of Manipulability

- Penalize configurations with limited maneuverability
- Consider:
  Distance to singular configurations and joint limits, self-distance, ...

red=low
green=high
Singular Configurations

- Certain EE movements are not possible
- Small desired changes in EE poses lead to large joint angle changes
Inversion of the RM

- Invert the precomputed reachable workspace: inverse reachability map (IRM)
- Iterate through the voxels of the RM
- **Invert the FK transform** for each configuration stored in a voxel to get the **pose of the foot wrt the EE frame**
- Determine the voxel in the IRM **containing the foot pose**
- Store configurations and manipulability measures from the RM in the corresponding IRM voxels
Inversion of the RM

```plaintext
while v ← GET_VOXEL(RM) do
    nc ← GET_NUM_CONFIGS(v)
    for i = 1 to nc do
        (qc, qSWL, w) ← GET_CONFIG_DATA(v, i)
        ptcp ← COMPUTE_TCP_POSE(qc)  // end-effector pose via FK
        pbase ← (ptcp)^-1  // inverse transform to get pose of foot wrt EE frame
        idx ← FIND_EE_VOXEL(pbase)  // determine voxel of foot
        IRM ← ADD_CONF_TO_VOXEL(idx, qc, qSWL, w)
    end
end
```

The IRM represents the valid stance poses relative to the end-effector pose.
Inverse Reachability Map (IRM)

- The IRM represents the set of **potential stance poses** relative to the EE frame
- Allows for selecting an optimal stance pose for a given grasping target
- Computed once offline
- Queried online

Cross section through the IRM showing potential feet locations:
- red = low
- green = high
Determining the Optimal Stance Pose Given a Grasp Pose

- Given a desired 6D end-effector pose with transform $F_{grasp}$
- How to determine the optimal stance pose?
Determining the Optimal Stance Pose Given a Grasp Pose

- Transform the IRM and determine valid configurations of the feet on the ground
- Align the origin of the IRM with the grasp frame $F_{\text{grasp}}$ to get the transformed IRM $tIRM$
- Intersect $tIRM$ with the floor plane $F$:
  $$IRM_{\text{floor}} = tIRM \cap F$$
- Remove unfeasible configurations from $IRM_{\text{floor}}$ to get $IRM_{\text{stance}}$
Determining the Optimal Stance Pose: Example

Select the optimal stance pose from the voxel with the highest manipulability measure.
IRMs for Planning Whole-Body Motions

- Use the IRM to determine the optimal stance pose to reach the desired grasp pose
- Let the robot walk to that stance location
- Determine the whole-body configuration to reach the desired grasp pose (IK might be necessary since discrete 3D representation of the workspace in the IRM)
- Plan a whole-body motion to that goal configuration with RRT-connect
IRMs for Reaching a 6D View Pose in the Environment

- Given: 6D camera pose
- Wanted: Stance location with whole-body configuration
RM for Camera Poses

Evaluation function considers

- Weight distribution between left and right foot (stability)
- Time to reach the desired robot configuration from standard walking posture
- Power consumption to reach and hold pose

red=low
green=high
Efficient Implementation of the IRM

- Note: Representation of the IRM as a sparse 6D structure and the intersection of two representations is not efficient.
- Make use of the assumption that the robot can only stand on horizontal planes.
- The roll and pitch angle of the feet relative to the camera frame as well as the vertical distance can be directly derived.
- Represent the IRM as database, indexed with roll, pitch, and z.
- Database returns a list of candidate stance poses to reach a desired 6D camera pose.
Summary (1)

- IK computes the joint angles so that the end-effector reaches a desired goal pose.
- Several approaches for IK exist (analytical/numerical).
- Basic Jacobian IK technique iteratively adapts the joint angles to reach the end-effector goal pose.
- Motion planning computes a full trajectory from the initial to the goal configuration and consider constraints.
- Sampling-based algorithms such as RRTs can solve complex, high-dimensional motion planning problems, including whole-body motion planning.
Summary (2)

- RRTs are efficient and probabilistically complete, but yield non-optimal, non-repeatable, and unpredictable paths
- Several extensions exist, e.g., anytime RRTs
- Also approaches that combine RRTs with local Jacobian control methods have been proposed
- IRMs determine the optimal stance pose for desired EE poses, computed once offline
- The probability of successfully reaching a grasp target is increased
Literature (1)

- Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least methods
  S.R. Buss,
  University of California, 2009

- RRT-Connect: An Efficient Approach to Single-Query Path Planning
  J. Kuffner and S. LaValle,
  Proc. of the IEEE International Conference on Robotics & Automation (ICRA), 2000

- Whole-Body Motion Planning for Manipulation of Articulated Objects
  F. Burget, Armin Hornung, and M. Bennewitz,
  Proc. of the IEEE International Conference on Robotics & Automation (ICRA), 2013
Literature (2)

- Stance Selection for Humanoid Grasping Tasks by Inverse Reachability Maps
  F. Burget and M. Bennewitz,
  Proc. of the IEEE International Conference on Robotics & Automation (ICRA), 2015

- Robot Placement based on Reachability Inversion
  N. Vahrenkamp, T. Asfour, and R. Dillmann,
  Proc. of the IEEE International Conference on Robotics & Automation (ICRA), 2013

- Efficient Coverage of 3D Environments with Humanoid Robots Using Inverse Reachability Maps
  S. Oßwald, P. Karkowski, and M. Bennewitz,
  Proc. of the IEEE-RAS International Conference on Humanoid Robots (Humanoids), 2017
Acknowledgment

- Parts of the slides rely on presentations of Steve Rotenberg, Kai Arras, Howie Choset, Felix Burget, and Stefan Oßwald