Humanoid Robotics

Inverse Kinematics and Whole-Body Motion Planning

Maren Bennewitz
Motivation

- Plan a sequence of configurations (vector of joint angle values) that let the robot move from its current configuration to a desired goal configuration
- High-dimensional search space according to the degrees of freedom of the humanoid
- Several constraints have to be considered such as joint limits, avoidance of self- and obstacle collisions, and balance
Example Planning Problems

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

- Hard planning problem due to local minima in search space:
Recap: Forward Kinematics

- FK computes the end-effector pose given the current joint encoder readings
- Using transformation matrices (rotational and translational components)
- Consider linear kinematic chains between a root (e.g., the torso) and an end-effector or a foot
- Example: Transformation $\mathcal{F}_{B}^{E}(q)$ from the left end-effector frame $E$ to the robot’s torso frame $B$ given the encoder readings $q$ (and possibly learned offsets)
Inverse Kinematics (IK)

- IK computes the **joint angle values that will cause the end-effector to reach a desired goal state** (3D/6D position)
- Inverse of the forward kinematics problem
- FK: \( e = \mathcal{F}(q) \)
- IK: \( q = \mathcal{F}^{-1}(e) \)
- 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)

- Even if a solution exists, it may require complex computations to find it.
- Many different approaches to solving IK problems exist.
  - **Analytical methods**: Closed-form solution, directly invert the forward kinematics equations, use trig/geometry/algebra.
  - **Numerical methods**: Use approximation and iteration to converge to a solution, usually more expensive but also more general.
Inverse Kinematics Solver

- **IKFast**: Analytically IK, very fast, calculates all possible solutions
  http://openrave.org/docs/latest_stable/openravepy/ikfast/

- **Kinematics and Dynamics Library (KDL)**: Included in ROS, contains several numerical methods for IK
  http://wiki.ros.org/kdl
Inverse Kinematics: Example

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

$e = (e_x, e_y)$
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: Example

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

- In general, the Jacobian will be an 3xN or 6xN matrix where N is the number of joints.
- For each joint, analyze how the end-effector would change if the joint position changed.
- The Jacobian can be computed based on the equation of FK.
Numerical Approach Using the Jacobian

- Given a desired incremental change in the end-effector configuration, we can compute an appropriate incremental change in joint DOFs:

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

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

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

- Forward kinematics is a nonlinear function (as it involves sin’s and cos’s of the input variables)
- Thus, we have an approximation that is only valid near the current configuration
- We must repeat the process of computing the Jacobian and then taking a small step towards the goal until the end-effector is close to the desired pose
End-Effector Goal, Step Size

- Let $e$ represent the current end-effector pose, and $g$ represent the goal pose we want the end-effector to reach.
- Choose a value for $\Delta e$ that will move $e$ closer to $g$, start with:
  \[ \Delta e = g - e \]
- Note that the nonlinearity prevents that the end-effector exactly reaches the goal, but it gets closer.
- For safety, take smaller steps:
  \[ \Delta e = \alpha(g - e), \, 0 \leq \alpha \leq 1 \]
Basic Jacobian IK Technique

while (e is too far from g) {
    Compute $J(e, q)$ for the current config. $q$
    Compute $J^{-1}$
    $\Delta e = \alpha (g - e)$ \hspace{1cm} // choose a step to take
    $\Delta q = J^{-1} \Delta e$ \hspace{1cm} // compute change in joints
    $q = q + \Delta q$ \hspace{1cm} // apply change to joints
    Compute resulting $e$
    // apply FK compute new pose of end-effector
}
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.

- For more complex problems requiring collision-free motions in narrow environments, planning methods have to be applied.
Whole-Body Motion Planning

- Find a path through a **high-dimensional configuration space** (>20 dimensions)
- 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
Rapidly Exploring Random Trees (RRTs)

- Aggressively probe and explore the configuration space by expanding incrementally from an initial configuration.
- The explored territory is represented by a tree rooted at the initial configuration.
RRTs – General Principle of Constructing the Tree

```
BUILD_RRT (q_init) {
    T.init(q_init);
    for k = 1 to K do
        q_rand = RANDOM_CONFIG();
        EXTEND(T, q_rand);
    }
```

The algorithm terminates by checking if \( q_{new} \) is near the goal.
Bias Towards Goal

- When expanding, with some probability (5-10%) pick the goal instead of a random node
- Why not picking the goal every time?
- This will waste much effort in running into local minima (due to obstacles or other constraints) instead of exploring the space
RRT-Connect – Basic Concept

- Build trees from both start and end nodes (start and end configuration)
- Pick a random configuration: $q_{rand}$
- Find the nearest node in one tree: $q_{near}$
- Extend the tree from the nearest node by a step towards the random node:
- Extend the other tree towards $q_{new}$ from nearest node in that tree
- Return the solution path when the distance between $q_{new}$ and the nearest node in second tree is close enough
RRT-Connect – Example Path

\[ q_{\text{init}} \rightarrow \ldots \rightarrow q_{\text{goal}} \]
Extend Function

Returns

- Trapped: Not possible to extend the tree due to collisions/constraints
- Extended: Generated a step from $q_{near}$ towards $q_{rand}$
- Reached: Trees connected, path found
RRT-Connect

RRT_CONNECT \((q_{init}, q_{goal})\)  

\(T_a.init(q_{init});  \quad T_b.init(q_{goal});\)

for \(k = 1\) to \(K\) do 

\(q_{rand} = \text{RANDOM\_CONFIG}();\)

if not (EXTEND\((T_a, q_{rand}) = \text{Trapped}\)) then

if (EXTEND\((T_b, q_{new}) = \text{Reached}\)) then

Return \text{PATH}(T_a, T_b);

SWAP\((T_a, T_b)\);

Return Failure;

}
RRTs – Properties (1)

- Easy to implement
- Fast
- Produce non-optimal paths: Solutions are typically jagged and may be overly long
- Post-processing such as smoothing smoothing is necessary: Connect non-adjacent nodes along the path with a local planner
- Generated paths are not repeatable and unpredictable
- Rely on a distance metric (e.g., Euclidean)
RRTs – Properties (2)

- **Probabilistic completeness:**
  The probability of finding a solution if one exists approaches one

- However, when there is no solution (path is blocked due to 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 precomputed statically stable double support configurations from which we sample $q_{rand}$.
- Check $q_{new}$ for joint limits, self-collision, collision with obstacles, and whether it is statically stable within the extend function.
Collision Checking

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

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

- Apply RRT-Connect
- Smooth path after a solution is found (trees connected)
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
- Success rate of only 4.63%: Low probability of generating valid configurations, when the configurations space is sampled 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: 0.1
Example Results
(100 Planning Trials)

- Planning time upper / lower shelf: 0.09±0.27s / 10.44±0.83
- Expanded nodes upper / lower shelf: 19.84±30.06 / 1164.89±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

valid goal configurations for the same grasp
Spatial Distribution of the Reachable End-Effector Poses

- Representation of the robot’s reachable workspace
- Data structure generated in an offline step: reachability map (voxel grid)
- Represents all possible end-effector poses and quality information (manipulability)
Reachability Map (RM)

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

- Constructed by **sampling joint configurations** from a kinematic chain
- Apply **FK** to determine the **corresponding voxel** containing the end-effector pose
- Configurations are added to the RM if they are statically-stable and self-collision free
- Result: Reachability representation, each voxel contains configurations and quality measure
- Generating the RM is time-consuming, but done only **once offline**
Configuration Sampling

- Stepping though the range of the kinematic chain’s joints
- Serial chain: joints between the right foot and the gripper link
- Larger step width for the upper body joints to keep the reachability map representation sparse
- Smaller step width for the lower body joints to increase the probability of achieving a double support configuration
Generation of Double Support Configurations

Double support generation via active-passive link decomposition and IK:

- Given the hip and the desired swing foot pose expressed in the support foot frame, we can determine the pose for the swing foot relative to the hip frame.
- Afterwards: Solve IK for the swing leg chain.
Measurement of Manipulability

- Penalize configurations with limited maneuverability
- Consider:
  Distance to singular configurations and joint limits, self-distance, and distance to obstacles
Inversion of the RM

- Invert the precomputed reachable workspace: inverse reachability map (IRM)
- **Iterate through the voxels** of the RM
- Compute the **inverse transform** for each configuration stored in a voxel:
  - Inverse of the end-effector transform (obtained through FK) to get the pose of the support foot wrt the end-effector 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

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

- The IRM represents the set of **potential stance poses** relative to the end-effector pose
- Allows for selecting an optimal stance pose in 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 the transform $F_{\text{grasp}}$
- How to determine the optimal stance pose?
Determining the Optimal Stance Pose Given a Grasp Pose

- The IRM needs to be transformed and valid configurations of the feet on the ground have to be determined.
- Transform the IRM voxel centroids according to $F_{\text{grasp}}$ to get $tIRM$.
- 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.
Summary (1)

- IK computes the joint angle values so that the end-effector reaches a desired goal
- Several analytical/numerical approaches
- Basic Jacobian IK technique iteratively adapts the joint angles to reach the goal
- Motion planning: Computes global plan from the initial to the goal configuration
- RRTs are efficient and probabilistically complete, but yield non-optimal, not repeatable, and unpredictable paths
Summary (2)

- RRTs have solved previously unsolved problems and have become the preferred choice for many practical problems
- Several extensions exist, e.g., anytime RRTs
- Also approaches that combine RRTs with local Jacobian control methods have been proposed
- Efficiently determine the optimal stance pose for a given grasping pose using an IRM that is computed once offline
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

- Whole-Body Motion Planning for Manipulation of Articulated Objects
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
Acknowledgment

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