UR3e Reachability Logic and Reasoning

Robot Reach Analysis

Nathan Antonietti
Made using Gemini

1 Purpose

This document explains the reachability method used in the project, centered on `URBasic/reachability.py` and the notebook workflow in `my_simulation/demos/duck_position.ipynb`.

The goal is to decide whether a target tool pose is reachable while respecting:

- inverse kinematics feasibility,
- optional joint limits,
- optional link-vs-obstacle collision checks along the path,
- orientation ambiguity around a surface normal.

2 Function Contract

The key function is `is_reachable(...) -> bool`.

It returns `true` if *at least one* sampled orientation produces a valid and collision-free solution, and `false` otherwise.

Main inputs:

- `robot_control`: URBasic control object (must expose current joint/TCP and IK methods).
- `tcp6d`: target pose interpreted as
  - position: (x, y, z),
  - direction vector: (n_x, n_y, n_z) used as tool approach axis.
- optional `joint_limits`: six $(\min, \max)$ intervals in radians.
- optional `obstacles`: sphere or axis-aligned box primitives.
- `roll_samples`: number of rotations sampled around the approach axis.
- `include_opposite_normal`: whether to also test $-n$.

3 Core Reasoning

3.1 Normalize Target Geometry

The target position and normal are extracted and normalized. If the normal magnitude is near zero, the target is invalid.

3.2 Build Orientation Candidates

A base orientation is generated so that tool z aligns with the target normal. Then roll rotations are sampled around that axis:

$\varphi_k = 2\pi \frac{k}{N}$, for $k = 0, 1, ..., N-1$.

with `N = roll_samples` (or only `0` if `N = 1`).

If `include_opposite_normal = true`, both n and $-n$ are tested. This helps when the same contact point can be approached from the opposite side.

3.3 3) Inverse Kinematics Feasibility

For each orientation candidate:

- compute target TCP pose,
- call IK (`get_inverse_kin`), optionally seeded by current joints,
- reject if IK fails or output is malformed.

3.4 4) Joint-Limit Filtering

If limits are provided, each joint value q_i must satisfy:

q_i in [q_i,min, q_i,max] for i = 1..6.

Candidates outside limits are discarded.

3.5 5) Collision Check Along Interpolated Joint Path

When obstacles are provided, the algorithm linearly interpolates from current joints to target joints across `interpolation_steps`:

q(t) = q_start + t (q_goal - q_start), with t in [0, 1].

At each interpolation sample:

- UR3e joint positions are computed with DH transforms,
- each link segment is tested against obstacles,
- link thickness is approximated by per-link radii + clearance margin.

Obstacle models:

- sphere: segment-to-center distance check,
- box: sampled segment points inside expanded AABB.

If any interpolated configuration collides, that candidate is rejected.

3.6 6) Early Success Rule

As soon as one candidate is IK-valid, within limits, and collision-free, the function returns `true`. If all candidates fail, it returns `false`.

This design is conservative and practical: existence of one feasible approach is enough to classify a point as reachable.

4 Why Roll Sampling Matters

A single fixed orientation can falsely classify reachable points as unreachable. For 6-DOF manipulators, rotating around the approach axis can substantially change wrist configuration and avoid singular or constrained postures.

Therefore, the strategy is:

- fix position and approach direction,
- sample in-axis roll,
- accept any feasible result.

This approximates a local search over orientation redundancy with low complexity.

5 Notebook-Level Reachability Search (Object Translation)

In `duck_position.ipynb`, reachability is extended from one point to an entire mesh:

1. Sample random points on each triangle face.
2. Use face normals for local approach directions.
3. Evaluate each sample with `is_reachable`.

4. If some samples fail, estimate a translation direction from failed points toward robot base.
5. Move object by a bounded step and iterate.

Heuristic direction:
- for each failed point p_j, build vector toward base $b - p_j$,
- normalize valid vectors,
- average them to get movement direction.

This drives the object toward regions where failures are reduced while keeping updates simple and stable.

6 Practical Tuning Guidelines
- Increase `roll_samples` for harder orientations (cost increases linearly).
- Keep `interpolation_steps` high enough when obstacles are tight.
- Use realistic `link_radii` and `clearance` to avoid optimistic collision results.
- If many points fail similarly, object translation can improve global reachability before fine path planning.

7 Limitations
- Collision check uses simplified link geometry (capsule-like approximation), not full meshes.
- Box collision uses point sampling on segments, so very thin collisions can be missed if sampling is coarse.
- Reachability is binary per sampled orientation, not an optimization over manipulability.

8 Summary
The implemented reachability logic is an existence test over orientation samples with optional safety constraints:

- IK success,
- joint-limit compliance,
- collision-free interpolated motion.

The notebook then scales this pointwise test to whole-object feasibility using face sampling and iterative translation from failure patterns. Together, this provides a practical and explainable workflow for UR3e pose and object placement assessment.