Duck Reachability Analysis

Cone Search Optimization & Specifications

Nathan Antonietti
Made using Gemini

---

1 Overview

Determine which points on a duck surface are reachable by the UR3e robot while maintaining drawing orientation constraints. This analysis decomposes the problem, identifies constraints, explores optimizations, and establishes specifications.

2 Problem Components

- **Geometric Reachability**: Which Cartesian points can the TCP reach (workspace + kinematics)
- **Orientation Feasibility**: Valid tool orientations aligned with surface normals (±25° cone)
- **Collision Avoidance**: Self-collision and obstacle detection with safety margins
- **Surface Sampling**: Uniform point distribution with normal extraction from STL mesh

3 Constraints

Technological:

- Max reach: 0.85 m; Joint limits: $\pm\pi$ rad; Up to 8 IK solutions per pose
- Collision detection: 8000+ checks for 1000 points × 8 solutions
- Orientation tolerance: ±25° cone search (DRAWING_ANGLE config)

Environmental:

- Workspace bounds: $Y \leq -0.15$ m, $Z \in [0.05, 0.50]$ m, radial ≤ 0.85 m
- Duck: 2000 mm^2 surface, millimeter-scale STL (0.001 scaling)
- Non-accessible: flat base ($Z < 1$ mm), hidden back ($Y < 0$)

Operational:

- Tool must align with surface normal ±25°
- Zero self-collisions; 0.01 m obstacle clearance
- Real-time path planning for interactive use

4 Functional Requirements

Requirement	Description
Surface Sampling	STL mesh → point cloud with normals (1–5 points/cm², ±5° accuracy, filter base & hidden regions)
Transformation	Duck placement → homogeneous transform matrix (±2 mm translation, ±2° rotation)
IK + Cone Search	Target TCP → valid joint config (max_angle=25°, cone_step=5°, prefer qnear)
Collision Check	Joint config → valid/invalid + reason (self-collision, obstacle, Z-min constraints)
Reachability Map	Batch process 100–5000 points, report success rate & per-point failures
Visualization	Interactive 3D with reachability overlay (green=reachable, red=unreachable)

5 Performance Targets

Metric	Target	Rationale
Single-point validation	$< 50ms$	Real-time interaction
100-point batch	$< 10s$	Quick testing
1000-point full scan	$< 2\,\text{min}$	Practical analysis
IK success rate	$> 95\%$	Few false negatives
Collision accuracy	100% @ 0.01 m	Safety critical

Table 1: Performance Targets

6 Non-Functional Requirements

- **Code Quality**: Return (success: bool, reason: str, metadata: dict); comprehensive logging; type hints
- **Configurability**: `max_cone_angle`, `cone_step`, collision margins, workspace bounds, sample density
- **Debugging**: Detailed rejection reasons; PyBullet GUI option; exportable reachability JSON
- **Robustness**: Handle missing STL files, PyBullet disconnections, variable IK solvers, bounds checks

7 API Contract

```python
ok, q, reason, tcp_adjusted = checker.validate_tcp(
    robot=RobotControl, tcp=TCP6D, qnear=Optional[list],
    margin=float, check_obstacle=bool, orientation_search=bool,
    max_cone_angle=float, cone_step=float
) → Tuple[bool, Optional[list], str, TCP6D]
```

Returns: Joint angles (if ok), diagnostic reason, optionally adjusted TCP

8 Success Criteria

1. Accuracy: ≥95% agreement with real-world validation
2. Performance: 1000-point scan in < 90 seconds
3. Usability: Clear visualization of feasible drawing strokes
4. Reliability: Zero collisions during executed trajectories
5. Debuggability: Clear logging and visualization for failures

9 Conclusion

The UR3e duck-drawing reachability problem spans kinematics, collision detection, orientation planning, and surface geometry. By decomposing the problem systematically and implementing the specified algorithms, we build a robust validation system enabling safe, efficient collaborative robot drawing.