

Collision-Aware Trajectory Planning for Passenger Boarding Bridges in Airport Apron

Abishai Asir A. Thederajan¹, Cheng Yee Low^{1*}, Jingye Yee², Ching Theng Koh¹, Ramhuzaini Abd Rahman¹, Tobias Seidenberg²

¹Faculty of Mechanical and Manufacturing Engineering, Universiti Tun Hussein Onn Malaysia, Parit Raja, Johor, Malaysia

²Fraunhofer Institute for Mechatronic Systems Design IEM, Paderborn, Germany

ARTICLE INFO

Article history:

Received 28 May 2026

Revised 19 August 2026

Accepted 01 September 2026

Published 15 September 2026

Keywords:

PBB

Autonomous docking

Trajectory planning

ROS 2

MoveIt 2

DOI:

10.24191/jmeche.v23i3.13743

ABSTRACT

Passenger Boarding Bridges (PBBs) are the primary physical interface between terminal buildings and aircraft, yet current semi-automated systems execute pre-programmed trajectory sequences designed for static environments and lack real-time obstacle awareness. This makes them unsuitable for aprons, where ground support vehicles, cargo equipment, and personnel continuously occupy the docking corridor. This study presents and validates a collision-aware trajectory planning framework for autonomous PBB docking, implemented in Robot Operating System 2 (ROS 2) and MoveIt 2. Rather than committing to a single fixed path, the framework uses the MoveIt Task Constructor (MTC) to evaluate four topologically distinct docking routes in parallel and autonomously select the fastest collision-free trajectory, falling back automatically when the primary route is obstructed. The framework was validated in kinematic simulation over 80 runs, comparing RRTConnect and RRT* across unobstructed and obstructed scenarios, with obstacles registered directly as collision objects. A planner returning a trajectory proved an insufficient safety criterion: under obstruction, RRTConnect returned a trajectory in every run, but 70% violated joint constraints or penetrated obstacle geometry at execution-time validation, whereas RRT* produced no such failures across 40 runs and selected the correct fallback strategy in all 20 obstructed runs. RRT* required roughly 10 times more planning time than RRTConnect (21.3 s versus 1.7 s) yet yielded an identical execution time in the unobstructed scenario (96.3 s). These results indicate that asymptotically optimal planning and execution-time trajectory validation are both necessary for safe autonomous PBB docking under the tested single-obstacle configuration.

^{1*} Corresponding author. E-mail address: cylow@uthm.edu.my
<https://doi.org/10.24191/jmeche.v23i3.13743>

Introduction

Passenger boarding bridges (PBBs) play an important role in modern commercial aviation terminals. They are the enclosed walkway passengers use to move between the terminal and the aircraft. The global aviation network faces unprecedented operational pressure as forecast passenger volumes approach 22.3 billion annually by 2053, representing a compound annual growth rate of 3% from 2024 levels, with corresponding aircraft movements projected to reach 176 million within the same period (Airports Council International [ACI] World, 2025). More flights mean less time available at each gate. Docking the bridge quickly therefore matters since every extra minute at the gate reduces the number of aircraft an airport can handle in a day. Airport aprons are a dynamic environment where multiple tightly scheduled ground support activities, such as aircraft positioning, cargo handling, refuelling, and passenger boarding, must occur simultaneously (Sunaryo & Basaria, 2024; Wani et al., 2025). On the other hand, PBB systems at technologically advanced hubs have integrated automation technologies that enhance docking precision and operational throughput; however, these systems predominantly rely on pre-programmed trajectory sequences executed within controlled and predictable environments. Such rigid control architectures fail under high-traffic operational conditions where efficiency depends on the seamless integration of robotic systems into dynamic apron workflows (Joo et al., 2024).

Despite technological advancements in docking precision and accuracy, modern PBBs are fundamentally limited by the inability to monitor dynamic apron conditions in real time. Existing systems lack sensor-driven obstacle avoidance capabilities necessary to respond to the continuous movement of ground support equipment, personnel, and variably positioned aircraft (Sunaryo & Basaria, 2024; Joo et al., 2024). In practice this means slower docking, real safety risks, and compromised system reliability. The root problem is caused by the absence of reactive kinematic architectures capable of autonomously adapting trajectory planning based on real-time sensor input. Recent work on automated planning for unmanned ground vehicles within apron environments demonstrates the operational complexity of collision-aware path generation in this domain (Yuan et al., 2025). Sampling-based motion planning algorithms, particularly variants of the Rapidly-exploring Random Tree (RRT) family, have been proven to function well in high-dimensional constrained configuration spaces relevant to robotic manipulation (Karaman & Frazzoli, 2011; Kuffner & LaValle, 2000; LaValle, 1998; Shaarawy et al., 2026). However, their application to PBB kinematics within the Robot Operating System 2 (ROS 2) and MoveIt 2 computational frameworks remains unexplored in the published literature. Several previous studies have looked at related parts of this issue, but none have combined them into a single trajectory planning system. The human-automation task delegation framework for PBB operations was examined by Joo et al. (2024), who found that operator workload and trust were obstacles to autonomy but did not address the underlying trajectory planning or collision avoidance architecture. A systematic review of apron automation trends by Sunaryo & Basaria (2024) revealed that collision-aware path planning is still an issue in the industry. Although these researchers did not integrate with a reactive motion planner, Wani et al. (2025) and Zhang et al. (2024) showed object identification and localisation capabilities for apron environments, providing a perception front-end required for autonomous docking. An autonomous vehicle's kinematic structure, whether holonomic or differentially driven, differs significantly from that of a PBB, which has a branched non-holonomic undercarriage geometrically coupled to a four degrees of freedom (DOF) serial chain that conventional inverse kinematics solvers cannot resolve. Yuan et al. (2025) demonstrated trajectory planning for autonomous ground vehicles within apron environments. Although RRT* (Karaman & Frazzoli, 2011) and RRTConnect (Kuffner & LaValle, 2000) have shown good performance in high-dimensional constrained configuration spaces for robotic manipulation in the trajectory planning literature, their behaviour within the particular joint constraints, non-holonomic coupling, and safety-critical tolerance requirements of a PBB kinematic structure within ROS 2 and MoveIt 2 is yet completely unidentified. This is not an issue of gradual technological improvement, but rather a fundamental gap in the literature. As a result, this study is motivated by the following research question: can a collision-aware, multi-path

trajectory planning architecture, implemented in ROS 2 and MoveIt 2, enable autonomous PBB docking in both unobstructed and obstacle-present apron environments, and what are the minimal algorithmic requirements for structurally safe trajectory execution?

This research project addresses this gap through the design, implementation, and simulation-based validation of a collision-aware trajectory planning system for autonomous PBB docking operations using ROS 2 Jazzy and MoveIt 2 frameworks (Macenski et al., 2022; Chitta, 2016). The architecture uses the MoveIt Task Constructor (MTC) (Gorner et al., 2019) to plan several docking routes at the same time and select the fastest one that is free of collisions. The routes are structurally different from one another: they move the joints in a different order, rather than taking small detours around the same path. If the chosen route is blocked, the planner falls back to the next fastest one. In the wider FastGate architecture, obstacle positions are supplied by Light Detection and Ranging (LiDAR) point clouds fused with stereoscopic computer vision (Zhang et al., 2024). In this study those positions are passed to the planner directly, so that planner behaviour can be isolated from perception error. A custom non-holonomic kinematic controller resolves the geometrically coupled undercarriage joints that standard inverse kinematics solvers cannot handle, enabling physically consistent bridge motion throughout the docking sequence (Yee et al., 2025). The system is validated within an RViz simulation environment through comparative benchmarking of two Open Motion Planning Library (OMPL) planners, which are RRTConnect (Kuffner & LaValle, 2000) and RRT* (Karaman & Frazzoli, 2011), across unobstructed and obstacle-present scenarios, with each returned trajectory verified against joint constraints and collision geometry at execution time, and planning time, execution time, and peak jerk evaluated as supporting metrics. This simulation-first approach provides a risk-free validation environment and empirical grounding prior to physical deployment (Koenig & Howard, 2004). The study is limited to kinematic simulation on purpose. Motor dynamics, sensor noise, and physical hardware are left out so that any difference in performance can be attributed to the planner and not to the implementation. This gives a measured baseline to compare against once the system moves to hardware.

Methodology

This paper presents a simulation-based methodology for the development and validation of an autonomous Passenger Boarding Bridge (PBB) docking system, integrating multi-modal sensor fusion with a hierarchical dual closed-loop motion planning and control architecture (Spong et al., 2020). Building on the microservice architecture established in the FastGate project (Seidenberg et al., 2024), this study focuses on the simulation-based validation of the MoveIt Task Constructor trajectory planning module, representing one of the final steps toward a fully operational end-to-end autonomous PBB docking system.

Hardware configurations

Fig. 1 illustrates the physical sensor configuration deployed on the airport apron, adapted from the sensor layout presented by Wani et al. (2025). Three complementary sensors form the perception front-end. The Blickfeld Cube 1 3D LiDAR (yellow, Fig. 1) generates dense point cloud data (PCD) covering the full docking area from bridge tip to aircraft fuselage. The Lupus Surveillance Camera (red), co-located on the LiDAR unit, captures Red-Green-Blue (RGB) colour imagery of the same field of view, which is fused with the PCD in the Sensor Fusion Microservice to produce a colourised RGB-PCD, improving downstream detection robustness under varying apron lighting (Wani et al., 2025). The ZED X Stereo Camera (green) is bridge-mounted at the cabin apex, providing a fixed egocentric depth reference that translates with the bridge during docking, enabling precise computation of the relative door position in the final approach phase.

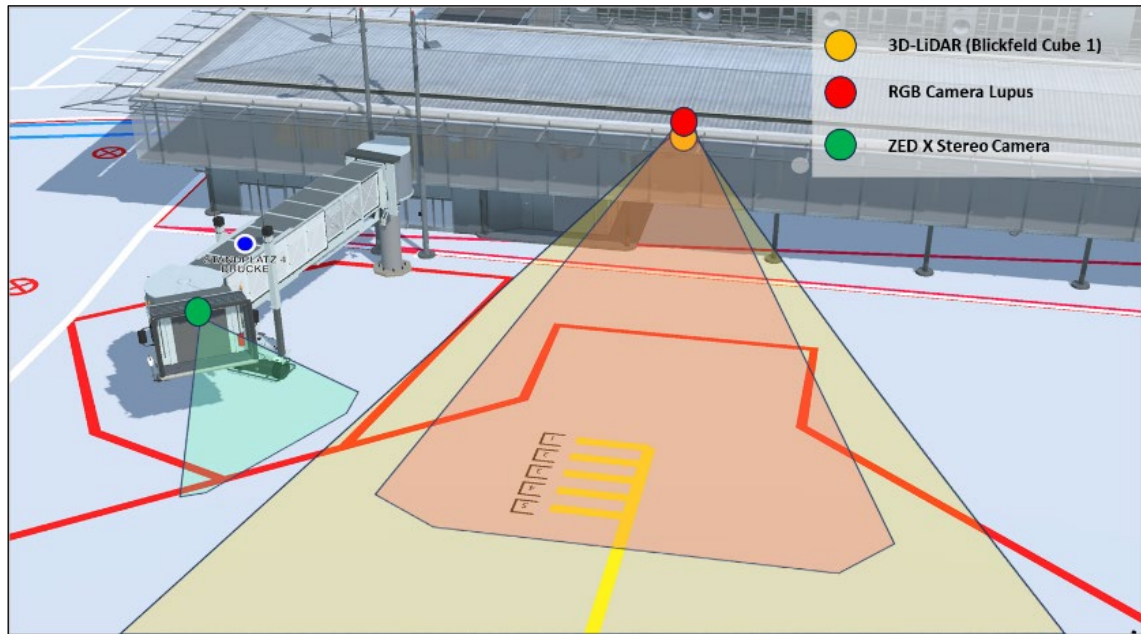


Fig. 1. Sensor placement and coverage on the airport apron. Adapted from Wani et al. (2025).

Kinematic modelling

The PBB is modelled as a branched kinematic structure with a primary four-DOF serial chain and a geometrically coupled undercarriage branch. The four controllable joints are the rotunda rotation ($\theta_1, revolute$), tunnel elevation ($\theta_2, revolute$), tunnel prismatic extension ($L_3, prismatic$), and cabin rotation ($\theta_4, revolute$) (Yee et al., 2025), and are summarised in Table 1. The world frame is anchored at the rotunda pivot axis at apron level ($Z = 0$). Although the PBB possesses four controllable DOFs within the six-dimensional task space, pitch and roll are mechanically constrained by the apron geometry, reducing the problem to four free variables (X, Y, Z, yaw). Consequently, a unique joint configuration exists for any reachable door position with no kinematic redundancy.

Table 1. Passenger boarding bridge joint parameters

Joint	Type	Physical Motion	Min	Max	Coupled
θ_1	Revolute	Rotunda rotation	-1.57	1.57	No
θ_2	Revolute	Tunnel elevation	-0.07	0.17	Drives L_3, θ_{steer}
L_3	Prismatic	Tunnel extension	-1	7.5	Drives L_3, θ_{steer}
θ_4	Revolute	Cabin rotation	-1.57	1.57	No
L_{Z^*}	Prismatic	Wheel vertical ext.	-	-	$f(\theta_2, L_3)$
θ_{steer^*}	Revolute	Wheel steering	-	-	$f(\theta_2, L_3)$

Rather than classical Denavit-Hartenberg (DH) parameterisation, forward kinematics are expressed from Unified Robot Description Format (URDF) joint origins, each specifying an XYZ translation and RPY rotation extracted from SolidWorks Computer-Aided Design (CAD) via *sw2urdf* exporter (Brawner, 2021). The end-effector transform is given in Equation 1:

$$T_4^0 = T_1^0(\theta_1) \cdot T_2^1(\theta_2) \cdot T_3^2(L_3) \cdot T_4^3(\theta_4) \quad (1)$$

where each T_{i+1}^i is derived from the corresponding URDF joint origin. MoveIt 2's Kinematics and Dynamics Library (KDL) solver constructs the kinematic tree natively from these transforms, bypassing manual DH extraction for this geometrically unique structure.

The mechanical model that was exported to URDF automatically defines joint constraint extraction, transform (TF) tree construction, and inertia computation from SolidWorks material assignments. Full-fidelity STL meshes were used for collision checking; the base link was simplified to a primitive cylinder to reduce Flexible Collision Library (FCL) overhead. Kinematic validation in RViz2 required three corrections: introduction of a dummy *base_footprint* link to resolve a KDL root-link inertia failure; tuning of prismatic joint velocity limits to eliminate a Time 0.0 Time-Optimal Trajectory Generation (TOTG) singularity; and manual Semantic Robot Description Format (SRDF) configuration to suppress false-positive self-collisions between adjacent tunnel links.

System architecture

Fig. 2 presents the overall system architecture, implementing a dual closed-loop control structure (Spong et al., 2020). The inner loop closes at the joint level via position encoders, providing fast actuator feedback. The outer loop closes at the Cartesian task level via Forward Kinematics, continuously re-referencing the perception pipeline against the bridge's actual pose to compensate for drift or aircraft movement during docking.

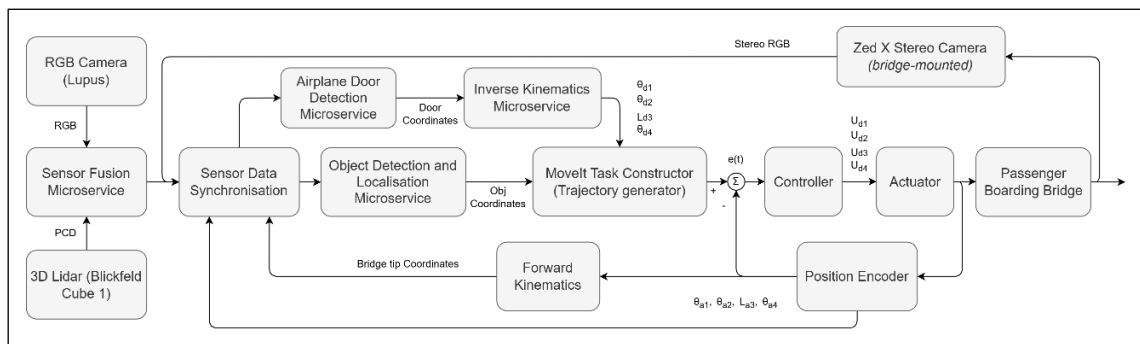


Fig. 2. Dual closed-loop architecture of the autonomous passenger boarding bridge docking system.

The flow of data is as follows: the Sensor Fusion Microservice combines the Lupus RGB stream and Blickfeld PCD into a colourised RGB-PCD, communicated between all microservices via Apache Kafka (Kreps et al., 2011). The Sensor Data Synchronisation module temporally aligns all sensor streams and ingests bridge tip coordinates from Forward Kinematics (FK) to anchor perception outputs to the real-world bridge pose. The Door Detection and Object Detection Microservices localise the aircraft door and apron obstacles respectively. The Inverse Kinematics Microservice converts door coordinates into desired joint states $(\theta_{d1}, \theta_{d2}, L_{d3}, \theta_{d4})$, which together with detected obstacle coordinates are passed to the MoveIt Task Constructor for collision-aware trajectory generation. The trajectory is executed by the joint Controller and Actuators, with Position Encoders feeding actual joint states $(\theta_{a1}, \theta_{a2}, L_{a3}, \theta_{a4})$ back to both the error junction (inner loop) and the Forward Kinematics module (outer loop).

Simulation environment

The simulation is built on ROS 2 Jazzy Jalisco and MoveIt 2. To separate benchmarking findings from physics solver variability, a purely kinematic simulation approach is used, which is in line with accepted trajectory planning benchmarking practices (Astorquia et al., 2025). The actuator dynamics are purposefully left out for this research scope. The simulated scene comprises the PBB URDF model; obstacle geometry is injected directly as MoveIt 2 collision objects, and door coordinates are supplied relative to the rotunda base frame.

Joint actuation uses the *ros2_control* framework with the *mock_components/GenericSystem* hardware plugin under pure position control. Two *JointTrajectoryController* instances are configured: *bridge_controller* ($\theta_1, \theta_2, L_3, \theta_4$) and *wheel_controller* (coupled undercarriage joints). MoveIt 2 generates trajectories via TOTG at 100 Hz; collision checking uses the FCL. A custom Python diagnostic node subscribes to */joint_states* and */rosout*, exporting five metrics per run to Comma-Separated Values (CSV) format as shown in Table 2. Each of the 80 runs was manually triggered within the simulation environment, with per-run metrics captured and exported automatically to eliminate transcription error.

Table 2. Validation metrics evaluated per benchmarking run

Metric	Significance	Units
Planning Success Rate & Strategy Selection	Validates path generation and MTC strategy selection	%
Peak Jerk	Passenger comfort and structural safety criterion	rad/s ³
Computation Time	Planning latency relative to turnaround scheduling	seconds
Execution Time	Docking duration	seconds
Fallback Activation Rate	Validates the MTC Alternatives container's autonomous fault-tolerance	%

Trajectory planning with MoveIt task constructor

The trajectory planning uses the MoveIt Task Constructor (MTC) framework (Gorner et al., 2019), utilising a parallel evaluation strategy. An Alternatives container evaluates four *SerialContainer* instances (Options 0-3) concurrently, each representing a topologically distinct docking approach as shown in Fig. 3. The container discards any option whose trajectory intersects a registered collision object and selects the survivor with the lowest *cost*: *TrajectoryDuration* score. This provides inherent fault tolerance; if the optimal path is obstructed, the system degrades automatically to the next fastest collision-free alternative.

Option 0 (Simultaneous Auto-Dock) interpolates θ_1, L_3, θ_2 , and θ_4 concurrently after pre-steering the undercarriage along the non-holonomic steering vector, producing the shortest execution time by overlapping all degrees of freedom in a single 3D arc manoeuvre. Option 1 (Rotate-then-Extend) sequentially completes rotunda rotation (θ_1) to the target bearing before initiating tunnel extension (L_3), decoupling the swing arc from the linear approach to reduce lateral sweep in corridor-constrained environments. Execution time is intermediate between Options 0 and 2. Option 2 (Deep Sweep) executes a fully sequential four-phase motion: (i) tunnel extension to full reach at flat elevation, (ii) undercarriage steering to final approach bearing, (iii) cabin alignment (θ_4), and (iv) final elevation (θ_2) to door height. By completing each joint group independently before the next begins, this strategy presents the smallest swept volume per phase and provides the greatest geometric clearance from obstacles occupying the central docking corridor. Lastly Option 3 (Vertical Bypass) elevates to maximum structural altitude ($\theta_2 = 0.12$ rad) before initiating any lateral motion, traverses at full height over any obstacle occupying the approach envelope, then descends to door level in the final stage. This strategy trades the longest execution time for maximum obstacle clearance height. All stages use the OMPL, with the planner identity set at runtime as a ROS 2 node parameter. The planning time budget is 15 s per stage; OMPL path simplification

is applied before TOTG time-parameterisation against *joint_limits.yaml* bounds. Table 3 summarises the specific MTC stage sequence executed by each option.

Table 3. MoveIt Task Constructor stage sequence for each docking option.

Option	Strategy	MTC Stage Sequence
0	Simultaneous Auto-Dock	Pre-Steer Wheels → Execute 3D Arc → Reset Steering
1	Rotate & Align → Push	Ensure Flat → Wheel to 90 → Rotate & Align Cabin → Wheel to 0 → Push & Lift
2	Deep Sweep	Ensure Flat → Extend Flat → Wheel to 90 → Rotate & Align Cabin → Wheel to 0 → Final Elevate
3	Vertical Bypass (Hop)	Elevate Max → Extend at Max → Wheel to 90 → Rotate & Align Elevated → Wheel to 0 → Lower to Target

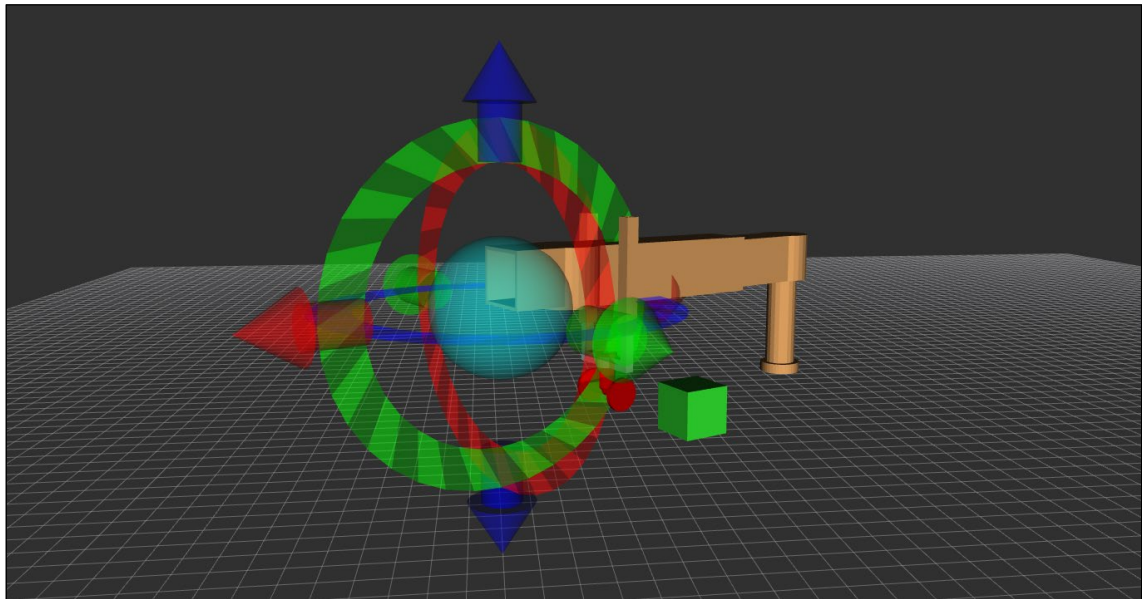


Fig. 3. RViz2 planning scene prior to planning.

Three joint path constraints as shown in Table 4 maintain structural integrity. RRTConnect's bidirectional greedy expansion randomly samples constraint volume extremes, producing trajectories that satisfy bounds at sampled waypoints but violate them during TOTG interpolation. RRT* iteratively rewires its search tree to minimise path cost, eliminating this interpolation violation at the cost of increased computation time, a trade-off that is the primary focus of the validation study.

Table 4. Joint path constraint definitions. All constraints applied to Tunnel1_Joint (elevation)

Constraint	Centre	Tolerance	Effective Range	Applied to
t1_lock	0.0 rad	± 0.02 rad	$[-0.02, 0.02]$ rad	Flat rotation & extension stages
t1_nofly_lock	0.05 rad	± 0.06 rad	$[-0.01, 0.11]$ rad	Execute 3D arc & final approach
t1_hop_lock	0.0 rad	± 0.15 rad	$[-0.15, 0.15]$ rad	Vertical Bypass stages only

Validation

System validation, which verifies that the MTC design and constraints direct the PBB to the docking destination without collision or constraint violation, and algorithmic benchmarking, which compares RRTConnect and RRT* across efficiency and trajectory safety parameters, are the two stages of validation. Two scenarios are assessed: Scenario A (Unobstructed), which establishes baseline performance, and Scenario B (Static Obstruction), in which the ideal path is blocked by a $2.0 \times 2.0 \times 2.0$ m collision cube. Continuous metrics are presented as $\mu \pm \sigma$, with 20 independent runs per planner per scenario (a total of 80), as shown in Table 5.

Table 5. Experimental matrix

Scenario	Objective	RRTConnect	RRT*
A - Unobstructed	Baseline efficiency and trajectory smoothness	20	20
B - Static Obstruction	MTC fallback selection and obstacle resilience	20	20

Within the benchmarking stage, each run was validated in two steps. First, planning success was recorded when MTC returned a trajectory. Second, the selected trajectory was verified at execution time, the TOTG-interpolated path was re-sampled at 100 Hz and checked against the Table 4 joint constraints and the collision geometry via FCL. Each run was then classified using an adapted confusion-matrix terminology, Pass Positive, where the planner returns a trajectory and the trajectory is valid; Fail Positive, where the planner returns a trajectory, but execution-time validation finds a violation; Fail Negative, where the planner fails despite a valid path existing; and Pass Negative, where the planner correctly fails because no valid path existed. Pass Positive is the only safe, deployable outcome; Fail Positive is the most hazardous since the unsafe trajectory would execute without warning, and no Pass Negative outcomes occurred across this study's 80 runs.

Trajectory smoothness is quantified as peak joint-space jerk via discrete third-order numerical differentiation of *joint_states* across all six active joints, as given in Equation 2:

$$j_i = (a_i - a_{i-1}) / \Delta t_i, \quad a_i = (v_i - v_{i-1}) / \Delta t_i, \quad v_i = (q_i - q_{i-1}) / \Delta t_i \quad (2)$$

where $\Delta t_i = t_i - t_{i-1}$ is the measured inter-sample interval (nominally 0.01 s at 100 Hz). Planning computation time is measured via a high-resolution wall-clock timer in the C++ sequencer node encapsulating the blocking *task.plan()* call, providing sub-millisecond accuracy independent of topic polling jitter.

Results

This section presents the benchmarking results for RRTConnect and RRT* across Scenario A (unobstructed) and Scenario B (static obstruction), based on 20 independent runs per condition (80 total). All continuous metrics are reported as mean $\pm$ standard deviation and visualised; full per-run data are summarised in Table 6.

Table 6. Benchmarking results for RRTConnect and RRT* across Scenario A and Scenario B

Metric	RRTConnect		RRT*	
	Scenario A	Scenario B	Scenario A	Scenario B
Trajectory Returned (%)	95%	100%	100%	100%
Pass Positive	19	6	20	20
Fail Positive	0	14	0	0
Fail Negative	1	0	0	0
Winning strategy	Option 0	Option 0 (13×) / Option 2 (7×)	Option 0 (20×)	Option 2 (20×)
Planning Time, $\mu \pm \sigma$ (s)	1.66 ± 0.38	1.76 ± 0.59	21.31 ± 0.56	18.27 ± 0.73
Execution Time, $\mu \pm \sigma$ (s)	96.28 ± 0.02	125.72 ± 13.79	96.29 ± 0.02	141.45 ± 0.01
Peak Jerk, $\mu \pm \sigma$ (rad/s ³)	3.56 ± 0.40	3.82 ± 1.03	3.85 ± 0.57	3.56 ± 0.28
Fallback Rate (%)	0%	35%	0%	100%

Trajectory safety

Scenario A evaluates docking in an unobstructed environment shown in Fig. 4. RRTConnect returned a trajectory in 19 of 20 runs (95%), with one run failing to provide a valid trajectory, whereas RRT* returned a trajectory in all 20 runs (100%). This single Fail Negative indicates that RRTConnect can occasionally fail to return a trajectory even in unobstructed settings. A valid path existed, since RRT* solved the identical configuration in all 20 runs. In Scenario B, both planners returned a trajectory in all 20 runs. However, this metric masked a substantial execution failure for RRTConnect.

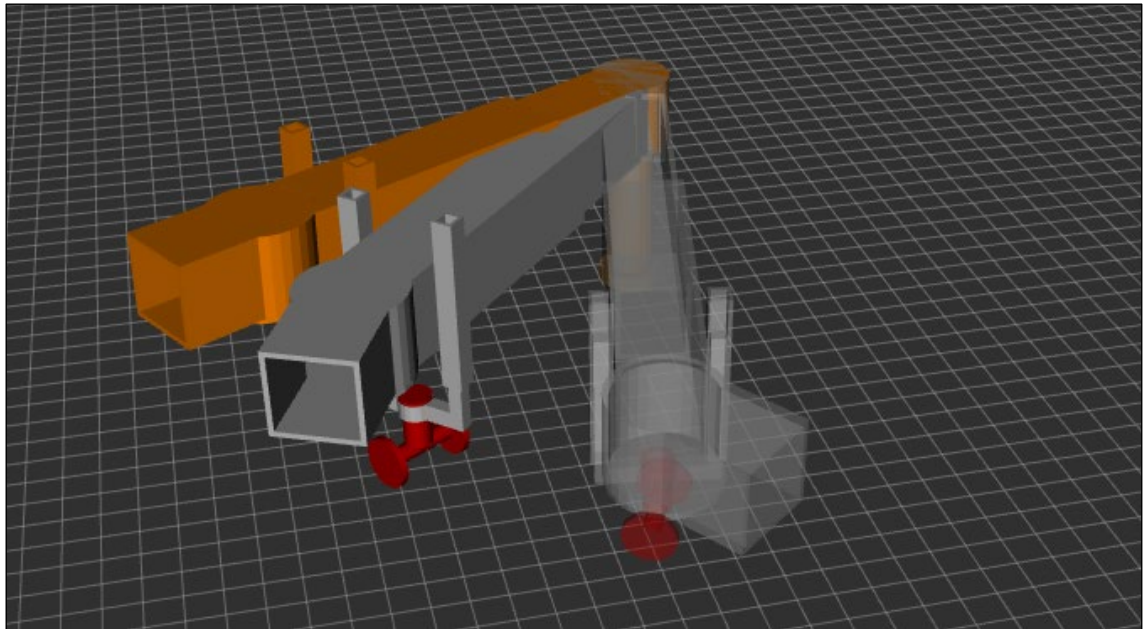


Fig. 4. RViz2 planning scene for Scenario A (unobstructed).

This study's primary safety finding relates to the Fail Positive outcome, the most operationally dangerous class of failure because the planner produces an invalid trajectory that would be carried out, driving the bridge into the obstacle or beyond its structural limits. Fail Positive outcomes are trajectories

that the planner declared collision-free but that broke structural constraints or penetrated obstacle geometry or both during execution. In Scenario A, neither planner produced Fail Positive outcomes, confirming that both can generate structurally valid trajectories in an unobstructed environment. However, the introduction of a static obstacle in Scenario B revealed a fundamental incompatibility between RRTConnect and constrained obstacle environments: 14 of 20 runs (70%) were classified as Fail Positive, with the planner returning a trajectory while the execution validator detected constraint violations or obstacle penetration. RRT* produced zero Fail Positive outcomes across all 40 runs in both scenarios, as shown in the confusion matrices in Fig. 5.

This failure is not caused by a planner misconfiguration but due to a fundamental property of bidirectional greedy expansion. RRTConnect rapidly connects sampled configurations without iterative path refinement, producing trajectories that satisfy constraints at discrete waypoints but violate them during TOTG interpolation. Under obstacle conditions, this sampling behaviour is compounded: in 13 of 20 Scenario B runs, RRTConnect selected Option 0 (Simultaneous Auto-Dock) despite the obstacle blocking that path, failing to identify the collision until the execution validator intervened. RRT*, by contrast, iteratively rewires its search tree to minimise path cost, producing trajectories that remain within constraint boundaries throughout interpolation and consistently identify the correct fallback strategy when the primary route was obstructed.

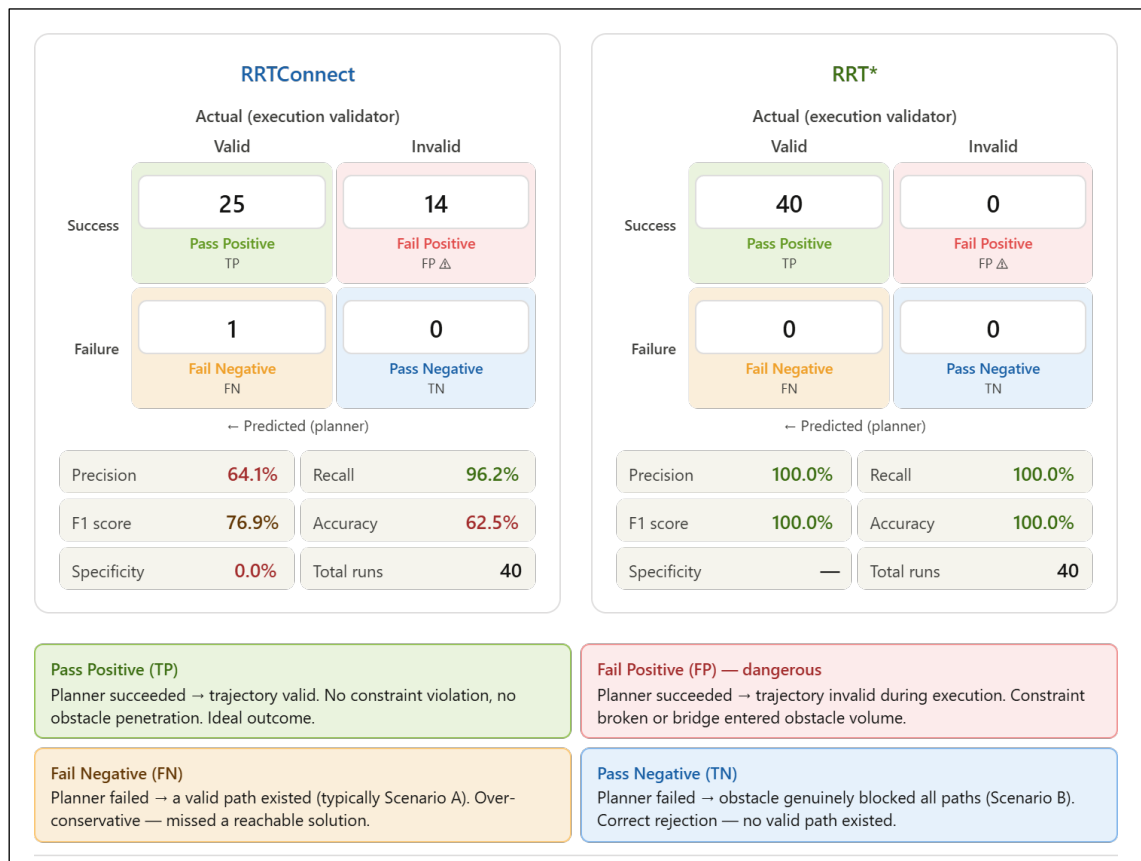


Fig. 5. Confusion matrices of planner outcome against execution validation for RRTConnect (left) and RRT* (right).

MoveIt task constructor fallback behaviour

In Scenario B, the collision object obstructed the Simultaneous Auto-Dock (Option 0) as shown in Fig. 6. Under RRT*, the MTC Alternatives container autonomously discarded Option 0 and selected Option 2 (The Deep Sweep) in all 20 runs, achieving a 100% fallback activation rate with zero Fail Positive outcomes. This confirms that RRT*'s path optimisation is capable of correctly identifying the absence of a valid collision-free path for Option 0 and selecting the next lowest-cost alternative without any manual intervention. Execution time increased from 96.29 ± 0.02 s (Scenario A) to 141.45 ± 0.01 s (Scenario B), reflecting the longer route of The Deep Sweep strategy relative to the direct Simultaneous Auto-Dock.

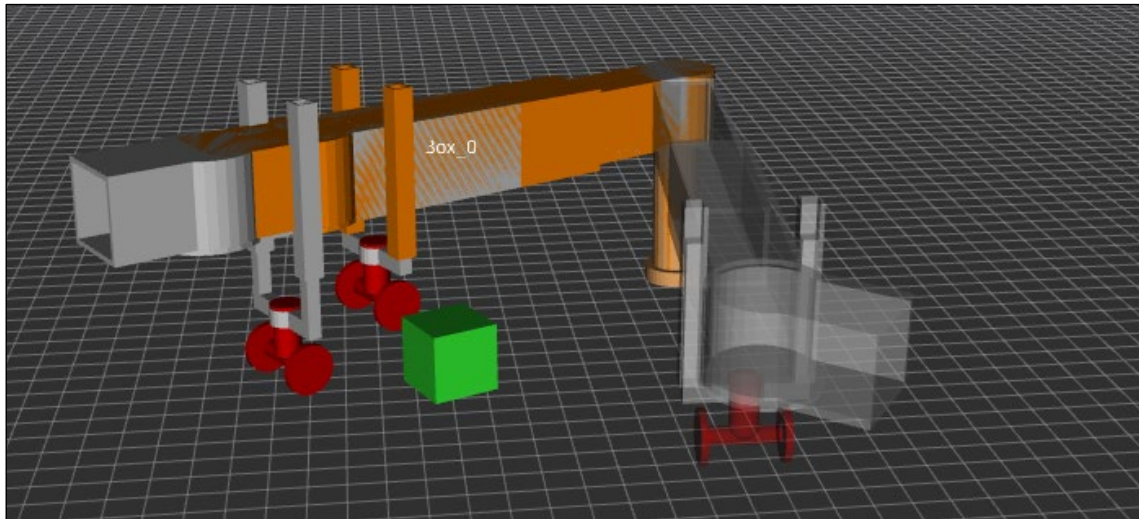


Fig. 6. RViz2 planning scene for Scenario B (obstructed).

Under RRTConnect, fallback behaviour was inconsistent: only 7 of 20 runs activated the fallback (selecting Option 2), while the remaining 13 runs proceeded with Option 0 despite the obstacle, producing 13 of the 14 observed Fail Positive outcomes. This confirms that RRTConnect's failure mode is specifically its inability to reliably detect collision-invalidated paths under constrained conditions, rather than a general planning failure. The single Fail Positive in the 7 fallback-activated runs suggests that Option 2 also produces constraint violations in a subset of RRTConnect executions, indicating that the greedy sampling problem is not fully resolved even when the correct strategy is selected.

Fig. 7 shows the planning computation time distribution for both planners across both scenarios. RRTConnect demonstrated a mean planning time of 1.66 ± 0.38 s in Scenario A and 1.76 ± 0.59 s in Scenario B, statistically comparable across conditions, confirming that obstacle presence has negligible effect on RRTConnect's planning latency. This is consistent with its greedy termination behaviour: RRTConnect stops as soon as a connecting path is found, irrespective of whether that path is geometrically optimal or execution safe.

RRT* required 21.31 ± 0.56 s in Scenario A and 18.27 ± 0.73 s in Scenario B, approximately $12.8 \times$ and $10.4 \times$ the planning time of RRTConnect respectively. The low standard deviation in both conditions indicates that planning duration is dominated by deterministic tree-rewiring cost rather than stochastic sampling variability. Notably, planning in Scenario B was faster than in Scenario A despite the presence of the obstacle. This is attributable to the MTC parallel evaluation structure, in Scenario A all four docking options are collision-free, and each undergoes full asymptotically optimal optimisation, whereas

in Scenario B the collision object invalidates two of the four options, which terminate early and remove their optimisation cost from the total planning budget. The obstacle therefore prunes the search rather than enlarging it.

Simulated execution time, shown in Fig. 8, reflects the actual docking duration independent of planning computation. In Scenario A, both planners produced near-identical execution times, 96.28 ± 0.02 s for RRTConnect and 96.29 ± 0.02 s for RRT*, confirming that planning time does not propagate into docking duration and that both planners converge on the same time-optimal Option 0 trajectory. The extremely low standard deviation (< 0.1 s) in Scenario A reflects that with fixed start and goal positions and no obstacle to route around, TOTG duration is governed by the joint limits and the total joint displacement rather than by the shape of the sampled path. Once an obstacle is introduced, path shape determines path length, and the stochastic component of RRTConnect's sampling becomes visible in the execution time.

In Scenario B, execution time increased to 125.72 ± 13.79 s for RRTConnect and 141.45 ± 0.01 s for RRT*. The high variance in the RRTConnect result has two sources. The first is inconsistent strategy selection: 13 runs executed Option 0 and 7 executed Option 2, producing a bimodal distribution. The second is that RRTConnect samples a different path on every run, so neither strategy settles on a fixed duration. Option 0 runs averaged 119.59 ± 12.00 s (range 101.20 - 139.50 s) and Option 2 runs 137.10 ± 9.07 s (range 117.10-141.45 s). Both are slower and far less repeatable than the unobstructed Option 0 trajectory at 96.28 ± 0.02 s, because the obstacle forces detour whose length depends on where the planner happened to sample. RRT* Scenario B shows near-zero variance ($\sigma = 0.01$ s), reflecting its consistent selection of Option 2 across all 20 runs.

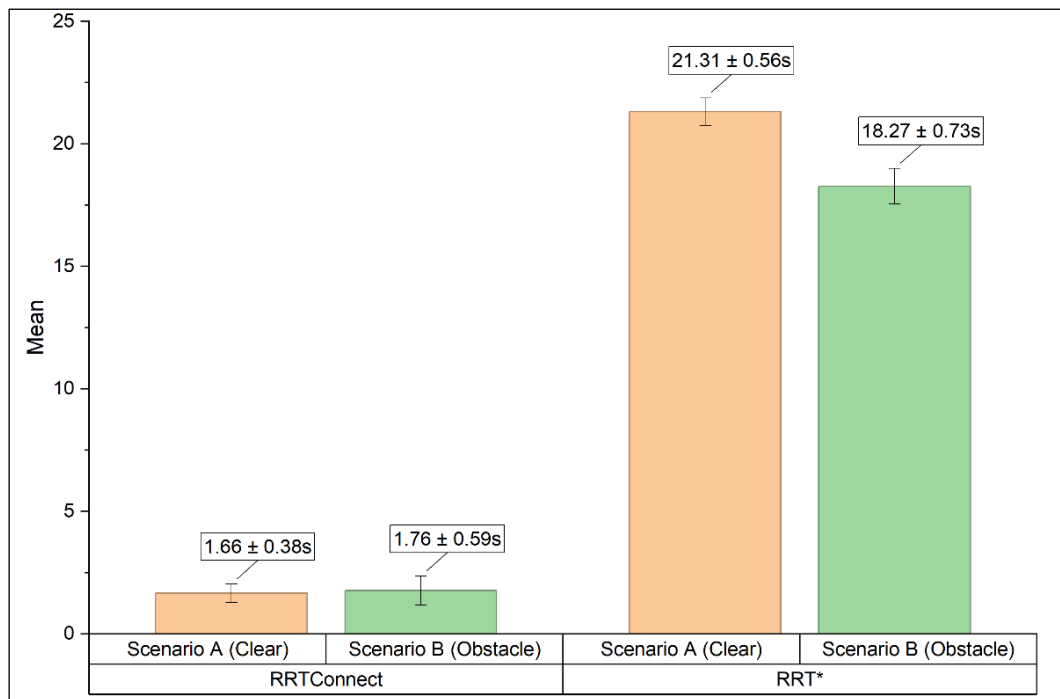


Fig. 7. Mean planning computation time (s) for RRTConnect and RRT* across Scenarios A and B (n = 20 per condition).

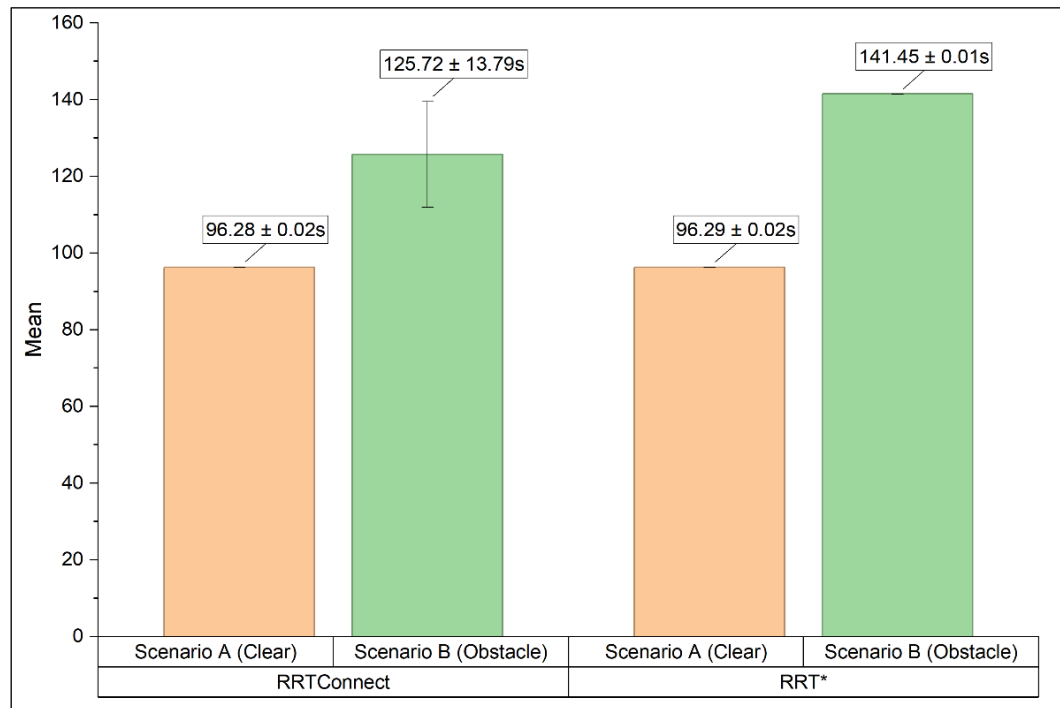


Fig. 8. Mean execution time(s) for RRTConnect and RRT* across Scenarios A and B (n = 20 per condition).

The distinction between Option 0 and Option 2 is further illustrated by the joint position trajectories recorded by the Python diagnostic node, shown in Fig. 9 and Fig. 10. In Fig. 9, the RRTConnect Scenario A sequence produces the characteristic simultaneous motion profile of Option 0 where *Tunnel2_Joint* extends gradually while Rotunda, Cabin, and Wheel joints all rise concurrently across the full execution window, reflecting the 3D arc manoeuvre in which all degrees of freedom are interpolated in parallel. While in Fig. 10, the RRT* Scenario B sequence produces a fundamentally different sequential profile corresponding to Option 2, where *Tunnel2_Joint* reaches full extension within the first 8 seconds, after which *Wheel1_Joint* steers and *Cabin_Joint* rotates gradually across the mid-phase, before *Wheel_Joint* rises in the final stage to complete the elevation sub-sequence. The topological difference between the two trajectories is immediately visible since the MTC architecture does not simply reroute the original path around the obstacle but selects a structurally distinct docking strategy whose motion primitives are physically incompatible with the obstructed route. This confirms that the fallback mechanism operates at the strategy level, not the waypoint level.

Within the Option 2 group, five of the seven runs fell between 141.25 s and 141.45 s, matching the deterministic RRT* Scenario B duration of 141.45 s. This indicates that when RRTConnect does find a clean Deep Sweep path, it converges on the same trajectory RRT* selects. The two shorter runs, at 135.75 s and 117.10 s, are the exceptions, and the 117.10 s run is the single Fail Positive recorded among the fallback-activated runs. A shorter execution time under RRTConnect therefore indicates a path that cut across the constraint boundary rather than a better route.

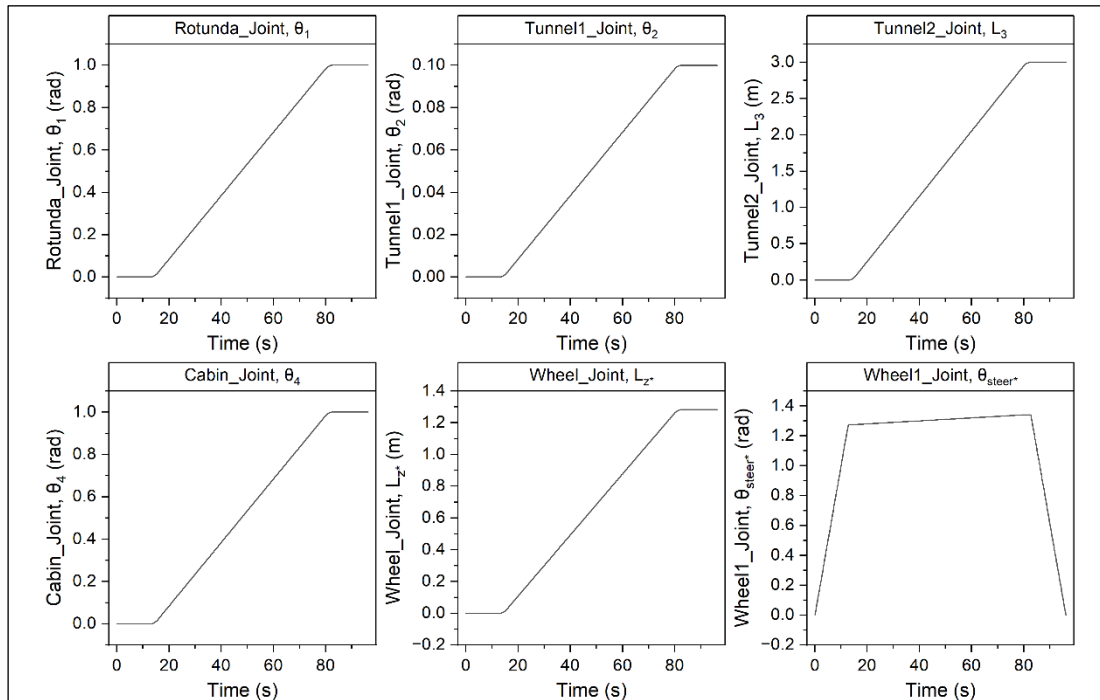


Fig. 9. Joint position trajectories for a representative RRTConnect run in Scenario A (Option 0 profile).

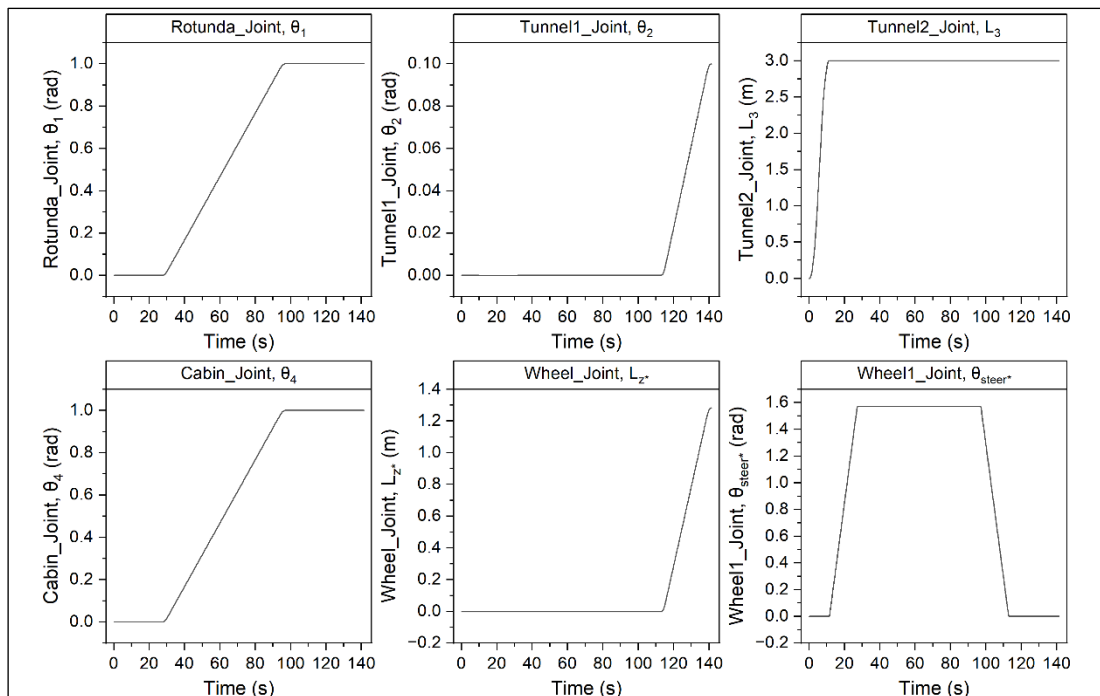


Fig. 10. Joint position trajectories for a representative RRT* run in Scenario B (Option 2 profile).

Taken together, the five metrics establish a consistent picture where RRT* is slower to plan but produces structurally valid, deterministic trajectories in both scenarios, while RRTConnect is fast but unreliable under obstacle conditions. The safety implications of this trade-off are explored in the Discussion.

Discussion

The MTC architecture performs collision-aware trajectory planning for autonomous PBB docking, but only when paired with a planner whose collision reporting is reliable. Under RRT*, the container discarded the obstructed route and selected the correct fallback in all 20 runs. Under RRTConnect, it selected the obstructed route in 13 of 20 runs, meaning the fault-tolerance mechanism failed at the architectural level rather than merely producing poor individual trajectories. This distinction is not a matter of degree; RRTConnect achieved a 70% Fail Positive rate in Scenario B, meaning that in most obstacle-present runs, it would have executed a dangerous trajectory in a deployed system. Within the single static obstruction tested, RRT* is therefore the appropriate planner for this architecture, and RRTConnect's greedy termination is incompatible with the constrained docking corridor. The combined precision of 64.1% across the 39 trajectories RRTConnect returned, contrasted with RRT*'s 100% precision, quantifies this incompatibility directly. The Fail Positive class is the critical outcome for safety critical PBB systems because it represents undetected failure, the scenario most likely to result in physical damage to the aircraft or bridge structure. Both planners produced comparable peak jerk values, $3.56 - 3.82 \text{ rad/s}^3$ for RRTConnect and $3.56 - 3.85 \text{ rad/s}^3$ for RRT*, indicating that trajectory smoothness is not the differentiating factor between planners. The critical distinction lies in execution validity rather than motion profile quality (Ali et al., 2025).

The planning time disparity between RRTConnect (1.66 – 1.76 s) and RRT* (18.27 – 21.31 s) is the only measurable disadvantage of the safer planner and requires explicit contextualisation. Standard airport gate turnaround procedures allocate several minutes between aircraft arrival at the gate and the commencement of passenger boarding (Kierzkowski et al., 2025), providing operational headroom that substantially exceeds the observed RRT* planning latency. Critically, planning computation occurs prior to any physical bridge movement and therefore does not extend the docking duration, which is governed exclusively by execution time. Since both planners produce comparable execution times in Scenario A (96.28 s versus 96.29 s), the planning time difference is operationally invisible to turnaround scheduling. Notably, planning in Scenario B (18.27 s) was faster than in Scenario A (21.31 s), since the obstacle invalidates two of the four MTC docking options early, removing their optimisation cost rather than enlarging the search space; this again occurs entirely before bridge movement begins. This speed-optimality trade-off between greedy and asymptotically optimal sampling-based planners is consistent with findings in comparative motion planning benchmarking studies, where optimal planners consistently incur higher computation costs relative to greedy alternatives across constrained configuration spaces (Astorquia et al., 2025).

The MTC parallel evaluation architecture performed as designed across all 40 RRT* runs. In Scenario B, the Alternatives container autonomously selected Option 2 (The Deep Sweep) in every run without exception, demonstrating that the fault-tolerance mechanism operates reliably when paired with a planner capable of correctly identifying collision-free paths. The consistent execution time of $141.45 \pm 0.01 \text{ s}$ across all RRT* Scenario B runs reflects the deterministic nature of the selected fallback strategy, with low variance confirming stable constraint adherence throughout. This validates the central architectural contribution of this study: a multi-path evaluation framework that handles apron obstacle variability without requiring manual replanning, obstacle classification, or operator oversight.

This study carries limitations inherent to kinematic simulation. The absence of rigid-body dynamics, actuator nonlinearities, and sensor noise means that reported values represent lower bounds on real-world performance. The obstacle evaluated in Scenario B is static; dynamic obstacles such as moving ground vehicles would require integration with a real-time perception pipeline, which lies outside the scope of this simulation phase. Transition to physics-enabled simulation using the *gazebo_ros2_control* plugin with full inertial properties, followed by physical hardware validation, constitutes the immediate next step toward operational deployment.

Conclusion

This study investigated collision-aware trajectory planning for autonomous passenger boarding bridge docking using MoveIt Task Constructor in ROS 2 Jazzy and MoveIt 2, while benchmarking RRTConnect against RRT* across 80 runs in unobstructed and obstructed scenarios. The results showed that a planner returning a trajectory is not evidence that the trajectory is safe to execute. Under obstruction, RRTConnect returned a trajectory in every run, yet 14 out of 20 (70%) violated joint constraints or penetrated the obstacle. RRT* discarded the obstructed route and selected the Deep Sweep alternative in all 20 obstructed runs, with execution time rising from 96.29 ± 0.02 s to 141.45 ± 0.01 s at near-zero variance ($\sigma = 0.01$ s) against RRTConnect's 13.79 s. Although RRT* required 18.27 – 21.31 s of planning time against 1.66–1.76 s for RRTConnect, this latency is incurred entirely before bridge movement and remains well within standard gate turnaround headroom, which is governed by execution time alone. Overall, the findings confirm that an asymptotically optimal planner paired with execution-time trajectory validation provides a viable basis for safe autonomous PBB docking under the single static obstruction tested. Physics-enabled simulation and hardware trials are still required to establish transferability to field conditions.

Acknowledgements/ Funding

This research was supported by Universiti Tun Hussein Onn Malaysia (UTHM) through the Geran Penyelidikan Pascasiswazah (GPPS) (Vot J195). Furthermore, the authors convey their appreciation to Fraunhofer IEM for providing the resources and opportunity to collaborate on this research. The authors further acknowledge the Hochschule Heilbronn (HHN) and German Academic Exchange Service (DAAD) for its support through the Integrated International Degree Programmes with Double Degrees.

Conflict of Interest Statement

The authors acknowledge that this research was undertaken in the absence of any self-benefits, commercial or financial conflicts and declare the absence of conflicts of interest with the funders.

Authors' Contributions

The authors confirm their contribution to the paper as follows: study conception and design: Abishai Asir A. Thederajan, Tobias Seidenberg, Ramhuzaini Bin Abd Rahman, Ching Theng Koh; data collection: Abishai Asir A. Thederajan; analysis and interpretation of results: Abishai Asir A. Thederajan, Cheng Yee Low, Jingye Yee; draft manuscript preparation: Abishai Asir A. Thederajan, Cheng Yee Low. All authors reviewed the results and approved the final version of the manuscript.

<https://doi.org/10.24191/jmeche.v23i3.13743>

Declaration of Generative AI in the Writing Process

The first author used Claude (Anthropic) to test and troubleshoot portions of the ROS 2 and MoveIt 2 implementation code while preparing this study. After using this tool, the first author examined and edited the text as needed, and he accepts full responsibility for the publication's contents.

Data Availability

The benchmark data from the 80 simulation runs and the ROS 2/MoveIt 2 implementation code developed during this study are accessible from the corresponding author on reasonable request.

Ethics Statement

The authors declare that this research did not involve human or animal subjects. All experimental procedures were performed following the institutional Safety, Health, and Environmental (HSE) protocols of Universiti Tun Hussein Onn Malaysia (UTHM).

Reference

- Airports Council International (ACI) World. (2025, February 26). ACI World projects 22.3 billion passengers by 2053. <https://aci.aero/2025/02/26/aci-world-projects-22-3-billion-passengers-by-2053/>
- Ali, A. A., Salem, F. B., & Mohammed, J. A. K. (2025). Design and implementation of jerk-controlled elevator systems using S-curve motion profiles. *International Journal of Power Electronics and Drive Systems (IJPEDS)*, 16(2), 780-793. <https://doi.org/10.11591/ijpeds.v16.i2.pp780-793>
- Astorquia, I. F., Villate-Castillo, G., Tellaeche, A., & Vazquez, J. I. (2025). Comparative benchmark of sampling-based and DRL motion planning methods for industrial robotic arms. *Sensors*, 25(17), 5282. <https://doi.org/10.3390/s25175282>
- Brawner, S. (2021). SolidWorks to URDF exporter [Computer software]. GitHub. https://github.com/ros/solidworks_urdf_exporter
- Chitta, S. (2016). MoveIt!: an introduction. In A. Koubaa (Ed.), *robot operating system (ROS)* (pp. 3–27). Springer. https://doi.org/10.1007/978-3-319-26054-9_1
- Gorner, M., Haschke, R., Ritter, H., & Zhang, J. (2019). MoveIt! Task constructor for task-level motion planning. 2019 International Conference on Robotics and Automation (pp. 190–196). IEEE Publisher. <https://doi.org/10.1109/ICRA.2019.8793898>
- Joo, J., Gomez-Beldarrain, G., Kim, E., & Verma, H. (2024, June 23). Transition towards automatic passenger boarding bridge: themes of task delegation for autonomous airport operations. *Design Research Society Biennial Conference Series 2024* (pp. 1-19). DRS Digital Library. <https://doi.org/10.21606/drs.2024.872>
- Karaman, S., & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7), 846–894. <https://doi.org/10.1177/0278364911406761>

- Kierzkowski, A., Ryczyński, J., Kisiel, T., Mardeusz, E., & Prentkovskis, O. (2025). Modelling the sustainable development of the ground handling process using the PERT-COST method. *Sustainability*, 17(24), 11278. <https://doi.org/10.3390/su172411278>
- Koenig, N., & Howard, A. (2004). Design and use paradigms for Gazebo, an open-source multi-robot simulator. 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 2149–2154). IEEE Publisher. <https://doi.org/10.1109/IROS.2004.1389727>
- Kreps, J., Narkhede, N., & Rao, J. (2011). Kafka: a distributed messaging system for log processing. Proceedings of the 6th International Workshop on Networking Meets Databases (NetDB) (pp. 1-7). ACM Publisher.
- Kuffner, J. J., & LaValle, S. M. (2000). RRT-connect: an efficient approach to single-query path planning. Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation (pp. 995–1001). IEEE Publisher. <https://doi.org/10.1109/ROBOT.2000.844730>
- LaValle, S. (1998). Rapidly-exploring random trees: a new tool for path planning (Technical Report No. 9811). Department of Computer Science, Iowa State University.
- Macenski, S., Foote, T., Gerkey, B., Lalancette, C., & Woodall, W. (2022). Robot operating system 2: design, architecture, and uses in the wild. *Science Robotics*, 7(66), eabm6074. <https://doi.org/10.1126/scirobotics.abm6074>
- Seidenberg, T., Stöhr, B., Eckertz, D., Jagnow, J., & Dumitrescu, R. (2024). HD mapping as enabler for autonomous systems on airports. Proceedings of the 2024 9th International Conference on Information Systems Engineering (pp. 10–15). ACM Digital Library. <https://doi.org/10.1145/3711954.3711967>
- Shaarawy, A., Rastegarpanah, A., & Stolkin, R. (2026). Task-aware motion planning in constrained environments using GMM-informed RRT planners. *Robotics and Computer-Integrated Manufacturing*, 97, 103095. <https://doi.org/10.1016/j.rcim.2025.103095>
- Spong, M. W., Hutchinson, S., & Vidyasagar, M. (2020). Robot modeling and control. (2nd Ed.). John Wiley & Sons.
- Sunaryo, E. R., & Basaria, F. T. (2024). Trends of automation in airport apron area: a systematic literature review. 2024 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS) (pp. 364–369). IEEE Publisher. <https://doi.org/10.1109/ICIMCIS63449.2024.10956779>
- Wani, B., Yee, J., Eckertz, D., Seidenberg, T., & Dumitrescu, R. (2025). Real-time object detection and localization for airport aprons. Seventeenth International Conference on Digital Image Processing (ICDIP 2025) (p. 137091Y). SPIE Digital Library. <https://doi.org/10.1117/12.3072877>
- Yee, J., Hermelingmeier, D., Thederajan, A. A. A., Low, C. Y., Gossen, A., & Dumitrescu, R. (2025). System architecture and analytical inverse kinematics for autonomous docking of passenger boarding bridges. 2025 IEEE International Symposium on Systems Engineering (ISSE) (pp. 1–8). IEEE Publisher. <https://doi.org/10.1109/ISSE65546.2025.11370093>
- Yuan, D., Zhong, Y., Zhu, X., Chen, Y., Jin, Y., Du, X., Tang, K., & Huang, T. (2025). Trajectory planning for unmanned vehicles on airport apron under aircraft–vehicle–airfield collaboration. *Sensors*, 25(1), 71. <https://doi.org/10.3390/s25010071>
- Zhang, T., Zhang, Z., & Zhu, X. (2024). Detection and control framework for unpiloted ground support equipment within the aircraft stand. *Sensors*, 24(1), 205. <https://doi.org/10.3390/s24010205>



© 2026 by the authors. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY-NC-SA) license (<https://creativecommons.org/licenses/by-nc-sa/4.0/>)