IFARLAB — Intelligent Factory and Robotics Laboratory

Eskişehir Osmangazi University

Multi-Robot Inspection of a Bus Chassis

Autonomous viewpoint planning and cooperative 3D coverage with two robot arms

Table 1: Scenario at a glance.

Item	Value
Task	Full-surface 3D inspection of a bus chassis frame
Robots	UR10e on a Festo linear axis + Kawasaki RS005L on a rail-guided AGV
Sensors	Two SICK Visionary-T Mini time-of-flight cameras, one per wrist
Autonomy	Viewpoints are computed from the CAD model — no taught positions
Measured coverage	87.1 % of the reference voxel map on the real cell (81.2 % with one arm)
Cycle time	UR10e 4.5 min, Kawasaki 8.7 min, executed in parallel
Maturity	Runs end-to-end on the physical cell; coverage is reach-limited, not sensor-limited

1. The industrial problem

Large welded structures such as a bus chassis are inspected today by moving a hand-held scanner around the part, or by a robot programme in which every measurement position has been taught by hand. Both approaches scale badly: a new part variant means re-teaching, and nobody can state afterwards which fraction of the surface was actually measured.

This use case removes the teaching step. The system takes the CAD mesh of the part, decides by itself where a camera has to stand to see every reachable surface, distributes those positions between two robots that approach the part from opposite sides, executes the tour, and finishes with a quantitative coverage report: which part of the structure was measured, and which was not.

2. Cell configuration

Table 2: Hardware and its role.

Component	Specification	Role in this scenario
UR10e	Collaborative 6-axis arm, mounted on a Festo linear axis (prismatic joint, 0.05–1.95 m travel along the cell Y axis)	Covers the near face and the upper region of the chassis
Kawasaki RS005L	6-axis industrial arm carried on an AGV that travels on a floor rail (0–2.0 m, commanded at 0.05 m/s)	Covers the far face, which the UR10e can never reach
SICK Visionary-T Mini	Time-of-flight depth camera, 70° × 60° field of view, working range used 0.2–1.8 m	One per wrist; provides the point cloud at every stop
Inspection target	Bus chassis CAD mesh: 121 956 triangles, 31.97 m² surface area	Both the planning input and the reference for the coverage report
Common frame	Single URDF, single world frame, one MoveIt planning scene	Both arms and both linear bases are planned as one kinematic system

The two linear bases are not separate navigation systems: they are prismatic joints inside the planning chain, so each arm is planned as a seven-degree-of-freedom system. This is what allows the planner to trade arm posture against base position when it searches for a reachable camera pose.

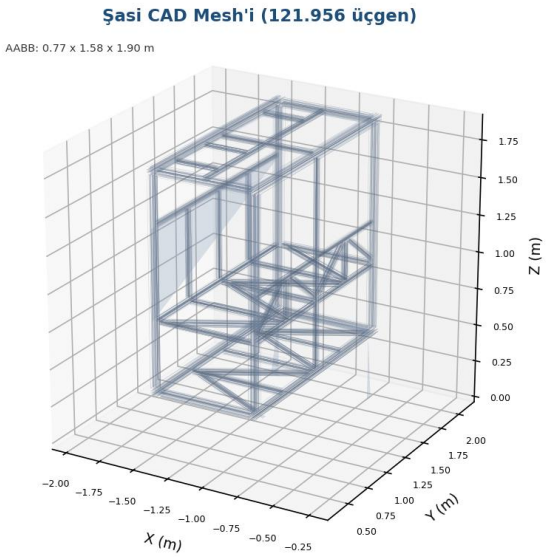


Figure 1: The inspection target as the planner sees it — the chassis CAD mesh in the cell world frame.

3. How the system works

Planning and execution are deliberately separated. Planning is geometric and needs no robot; execution replays a validated plan. The planner is started once and then triggered by a service call, so the expensive computation runs when the operator asks for it rather than at node start-up.

Table 3: The nine pipeline steps.

Step	What happens
1 — Surface sampling	The CAD mesh is sampled into target points; down-facing surfaces are dropped
2 — Candidate generation	Along each surface normal, candidate camera poses are produced at four standoff distances (0.3 / 0.6 / 1.0 / 1.4 m) and three tilts (−15° / 0° / +15°)
3 — Visibility model	A coverage matrix is built from the camera field of view, its working range, the incidence angle limit and ray occlusion against the mesh
4 — Scene synchronisation	The planning scene used at run time (floor plane, link padding of 4 cm) is applied before reachability is tested, so plan-time and run-time worlds agree
5 — Reachability	Every candidate is tested with inverse kinematics for both arms, with collision checking enabled and with each arm’s own workspace restriction
6 — Selection and allocation	A greedy set-cover picks the candidate that adds the most new surface and assigns it to an arm that can reach it; the tail of low-gain viewpoints is cut off deliberately
7 — Tour construction	Each arm’s stops are ordered into a monotone sweep; inverse-kinematics seeds are re-chained and whole 2π joint turns are removed
8 — Execution	Two independent processes drive the two arms simultaneously, each replaying cached trajectories and capturing a point cloud at every stop
9 — Reporting	The clouds are merged into an occupancy map and compared with a CAD-derived reference map to produce per-part coverage

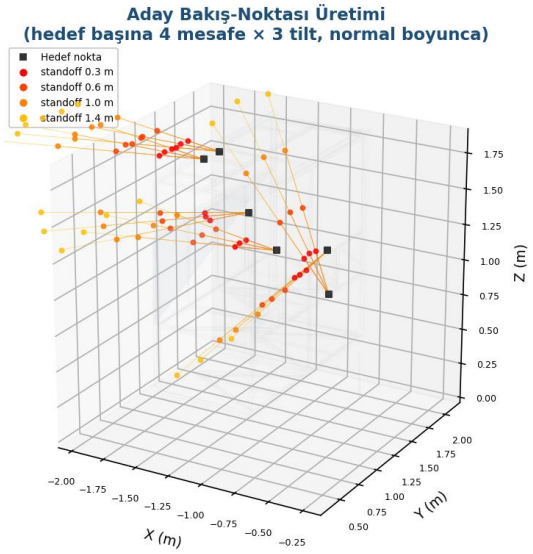


Figure 2: Step 2 — candidate camera poses generated along the surface normals of the part.

4. Software architecture

The software is organised as two ROS 2 packages: a single-arm planner that carries the geometric core, and a multi-robot package that adds the second arm, the allocation step and the two execution nodes. Nothing is forked — the multi-robot package depends on the single-arm one, so an improvement to the geometry benefits both.

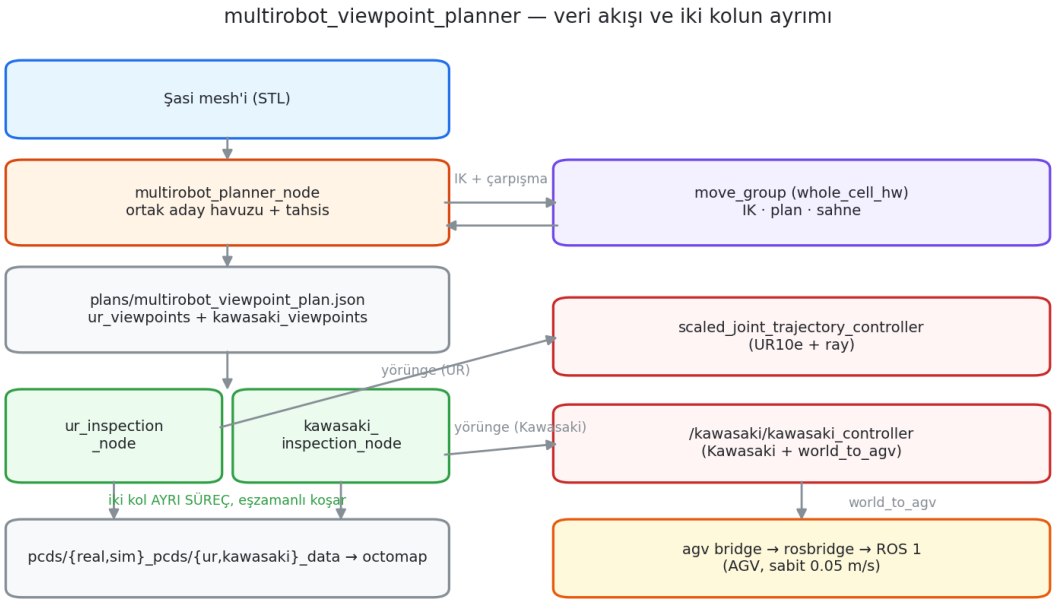


Figure 3: Data flow: planning happens in one node, execution in two separate processes, with the AGV as a slow platform behind the Kawasaki controller.

Table 4: Nodes and the ROS 2 interface each one uses.

Node	ROS 2 interface	Function
multirobot_planner_node	Service (Trigger) ~/plan; service clients /compute_ik, /apply_planning_scene	Runs the planning pipeline and writes the plan file
robot_allocator	Library	Greedy coverage selection and assignment of viewpoints to arms

ur_inspection_node	Action client (trajectory execution), point-cloud subscription, TF	Drives the UR10e through its stop list and captures clouds
kawasaki_inspection_node	Action client (FollowJointTrajectory), point-cloud subscription, TF	Drives the Kawasaki/AGV through its stop list
viewpoint_visualizer	MarkerArray publisher	Shows the plan in RViz for operator review

All three ROS 2 communication patterns appear in one application, each where it belongs: a service for the short request whose answer is awaited, an action for the minutes-long motion that must remain cancellable, and topics for the continuously streaming sensor data.

5. Measured results

Every number below was measured on the physical cell or computed from the plan files; none is an estimate.

Table 5: Single-arm baseline against the two-arm system, same part and same metric.

Metric	One arm (UR10e)	Two arms	Gain
Viewpoints in the plan	26	36 (25 UR10e + 11 Kawasaki)	—
Occupancy-map coverage, real robots	81.2 % (22 915 / 28 221 voxels)	87.1 % (24 588 / 28 221 voxels)	+5.9 points
Occupancy-map coverage, simulation	85.5 %	96.3 %	+10.8 points
Motion time (cached trajectories)	4.4 min	4.5 min (UR) and 8.7 min (Kawasaki), in parallel	Run paced by the slower arm

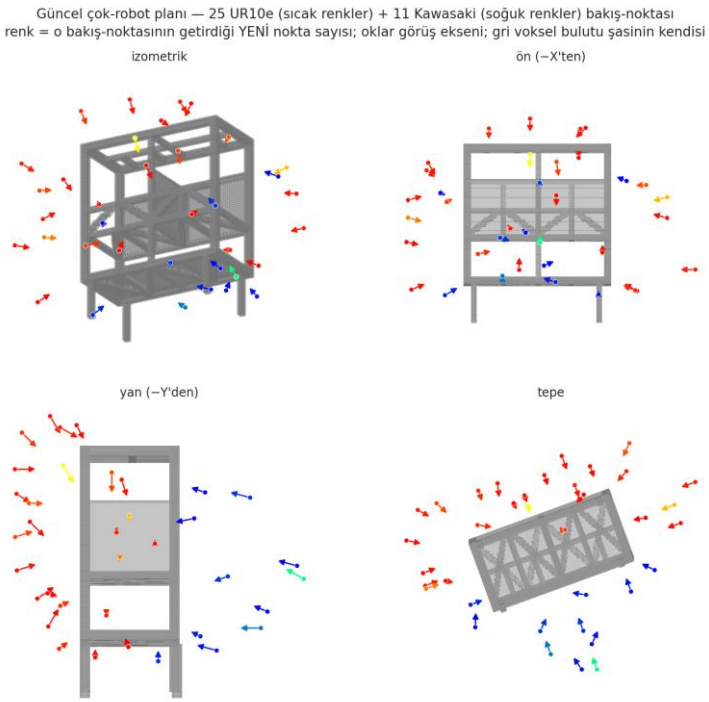


Figure 4: The executed plan in four views: 25 UR10e viewpoints (warm colours) and 11 Kawasaki viewpoints (cool colours). The top view shows that the two sets never overlap in X — the second arm is covering surfaces the first one cannot see.

The second arm is not redundancy. The two viewpoint sets occupy disjoint regions of the cell: the UR10e works between X = −1.31 m and −0.22 m, the Kawasaki between −2.47 m and −1.33 m. Without the second arm those surfaces are not merely measured more slowly — they are not measured at all.

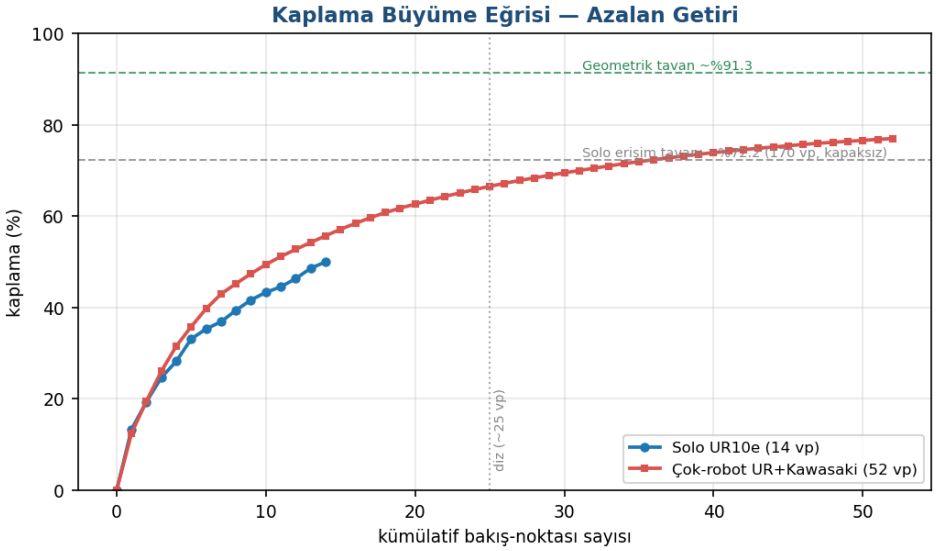


Figure 5: Coverage growth per selected viewpoint. The two-arm run passes the ceiling a single arm can reach; the dashed lines mark the geometric ceiling of the part and the single-arm ceiling.

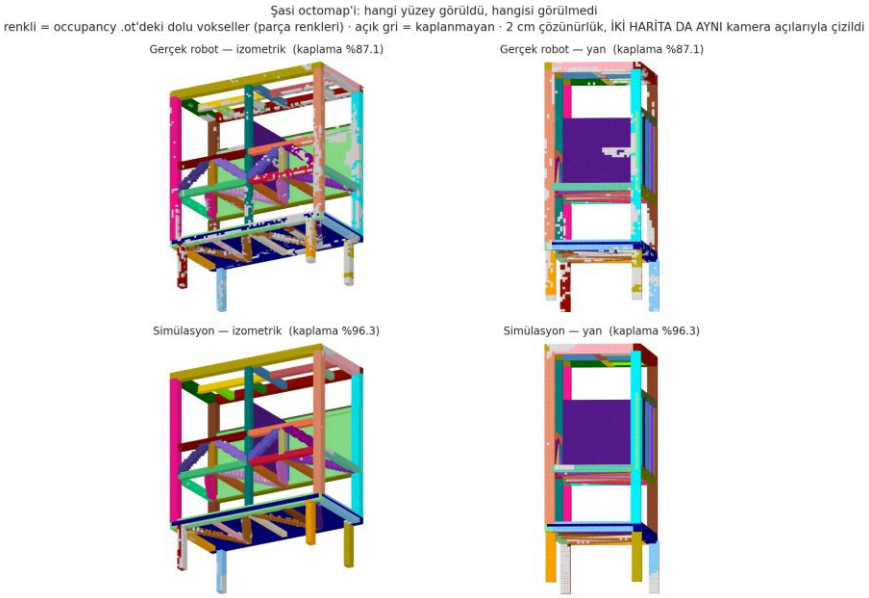


Figure 6: Occupancy map built from the captured clouds: coloured voxels were measured, grey voxels were not. Left and right are drawn from identical camera angles for the real cell and the simulated twin.

6. Engineering findings worth reporting

- Coverage is limited by reach, not by the sensor. About 98 % of the inverse-kinematics queries during selection fail with no solution; switching collision checking off moves the final coverage by 0.2 points. Adding a second arm is therefore the effective lever, not a better camera.
- Plan-time and run-time worlds must be identical. The execution scene — floor plane and 4 cm link padding — is applied before reachability is evaluated. When the two differed, failures appeared mid-tour rather than during planning, which is far harder to diagnose.
- Trajectories are recorded once and replayed. A sampling-based planner produces a different path every time; caching the validated path makes a run repeatable. The cache is invalidated explicitly after any change to the collision model.
- Removing whole joint turns shortened the tour by 35 % in joint-space travel (UR10e 10 860° → 6 969°, Kawasaki 4 495° → 3 009°) without changing a single camera pose.

- Planner choice is measurable: on the Kawasaki group, RRTConnect with ten attempts reduced joint travel by 22 % against a single attempt, while four other OMPL planners solved none of the 24 benchmark queries.

7. Current limitations

The following are stated plainly because they define what the system can be used for today.

- The configured coverage target of 98 % is not reached: the planner model reports about 70 % and the measured occupancy map 87 %. The binding constraint is arm reach around a part of this size.
- The two executors run in separate processes and share one planning scene; there is no explicit cross-arm collision avoidance. It is safe today because the two viewpoint sets are spatially disjoint, and a plan that violated that would need a different execution strategy.
- Real-cell coverage is 9.2 points below simulation, concentrated in the top rails and in the lower rails at AGV deck level — a reach and calibration effect, not a sensor defect.
- The AGV is commanded through a bridge to a legacy controller and moves open-loop at 0.05 m/s; arrival is confirmed by the executor, not by the controller.
- The second SICK camera currently shares a network segment with the UR real-time channel, which is why it is disabled in the default bring-up until the network is separated.

8. Running the scenario

The scenario is also reachable from the cell dashboard as "Multi-Robot Inspection".

```
# 1) Plan (no robot required) — the node waits for a service call
ros2 launch multirobot_viewpoint_planner multirobot_planning.launch.py
ros2 service call /multirobot_planner_node/plan std_srvs/srv/Trigger {}

# 2) Execute on the physical cell, both arms in parallel
ros2 launch multirobot_viewpoint_planner multirobot_inspection.launch.py only_sim:=false

# Single-arm reference runs (same plan file, for comparison)
ros2 launch multirobot_viewpoint_planner multirobot_inspection.launch.py only_ur:=true
ros2 launch multirobot_viewpoint_planner multirobot_inspection.launch.py only_kawasaki:=true
```

Reproducibility. Both the plan file and the trajectory cache are stored with the planning scene they were produced under, so a documented inspection run can be repeated later and compared against the original coverage report.