Nick Clark · Published April 25, 2026 · PDF

How to Stop a Robot From Attempting a Task Beyond Its Physical Precision

If your robot commits to a grasp, a fastener drive, or a fine placement it physically cannot hit, you learn about it only after the actuator stalls or the part is scrapped. This guide teaches an architecture that decides whether a motor objective can structurally exist on the robot before any motion plan is built, and refuses the objective when the robot's precision, degrees of freedom, force, or reach fall short. The approach is disclosed in United States Patent Application 19/647,395 and belongs to the Capability Awareness inventive step. It is a design you implement yourself, not a shipping library.

What You Are Building

You are building a gate that sits in front of your robot's motion planner. Before the robot commits to a motor objective, such as grasping an object, driving a fastener, or placing a part to a tight tolerance, the gate answers one question: can an executable form of this objective exist on this robot right now? If the robot's manipulators lack the degrees of freedom to orient the tool, if the actuator cannot produce the required force or torque, if the target is outside the reach envelope, or if the fine-placement tolerance is below the robot's motor precision, the gate resolves the objective as structurally impossible and refuses it before any trajectory is planned.

This is for anyone deploying manipulators, mobile platforms, or surgical and assembly robots where a physically infeasible attempt is expensive or dangerous: a stalled actuator, a scrapped workpiece, a collision at the edge of the workspace, or a precision task the hardware was never going to land. The developer who searches for how to stop a robot from attempting a task beyond its physical precision usually wants exactly this: a decision made up front, not an error caught after the fact.

The architecture described here is disclosed in United States Patent Application 19/647,395 as part of the Capability Awareness inventive step. This guide teaches the design. You implement it against your own hardware and control stack.

Why the Obvious Approaches Fall Short

The common way to handle infeasibility is to plan first and fail later. The robot receives a task, the motion planner attempts to synthesize a trajectory, and if the inverse kinematics do not converge, the force command saturates, or the goal is unreachable, the attempt returns an error. This works, but it does the expensive work first and discovers the impossibility second. It also collapses distinct failures into one generic error: a task the robot can never do on this hardware looks the same as a task it could do once the arm cools down or the battery recharges.

A second common approach is a static capability sheet: a manually maintained list of what the robot can do, checked at dispatch. The spec identifies the structural weakness here directly. Static registries do not capture the temporal dynamics of capability; a robot's affordances change as battery charge depletes, actuator temperatures rise, and sensors degrade. A sheet written at commissioning time does not know the arm is currently at a thermal limit.

A third pattern is a resource-availability check: enough compute, enough memory, enough bandwidth. The spec is explicit that resource availability is a necessary but insufficient condition for capability. A node, or a robot, may have ample compute and

still lack the structural affordance, the reach, the degree of freedom, the force capacity, required to produce an executable form of the objective. Precision is exactly this kind of structural affordance, not a resource you can free up by waiting.

The structural gap in all three: none of them treats feasibility as its own computation that runs before planning and produces an auditable, determinate result. That is what this architecture adds.

The Architecture

The central idea disclosed in the spec is that capability is a first-class computational state, evaluated before any executable process is constructed. Capability is defined there not as a metric, score, or probability but as a computed determination of whether an executable form of a given objective can exist on a given substrate. For a robot, the substrate is the physical body and its motor system.

The capability envelope. Each substrate advertises a capability envelope: a structured description of its affordances. For embodied systems the spec extends this beyond compute and memory to physical affordances, listing the degrees of freedom available to the manipulators, the force and torque limits of the actuators, the reach envelope of the arms, the locomotion capabilities of the mobility platform, the sensory modalities of the sensor suite, and the power budget for sustained operation. Where the objective involves fine manual work, motor precision, the ability to perform fine-grained tasks, is one of the physical affordance dimensions the framework evaluates.

Critically, the envelope is a living object. The spec states the capability envelope is a dynamic data object updated as the substrate's characteristics change, and that the system does not rely on stale or statically configured capability information. For a robot this means the envelope tracks present battery charge, actuator temperature, and sensor health, not commissioning-time nominals.

Dimension-by-dimension matching. A motor objective carries physical requirements, and those requirements are matched against the physical envelope in the same formal manner used for computational capability. The spec frames the matching as concrete questions: does the manipulator have the degrees of freedom required to orient the tool? Does the actuator have the force capacity required to drive the fastener? Does the mobility platform have the ground clearance and traction to traverse the terrain? Does the sensor suite include the modality required to detect the feature? A precision requirement joins this set: does the robot's motor precision meet the objective's tolerance?

Each dimension resolves to one of three outcomes described in the spec: satisfied, meaning the envelope meets or exceeds the requirement; unsatisfied, meaning it falls short in a way that cannot be resolved by deferral or reconfiguration; or conditionally satisfiable, meaning it falls short now but could be brought into satisfaction through temporal deferral, reconfiguration, or decomposition.

Aggregate determination. The per-dimension results compose, per the spec's composition rule, into one of four outcomes. All dimensions satisfied gives structurally possible. One or more unsatisfied with no conditional path gives structurally impossible. One or more conditionally satisfiable within a bounded horizon gives structurally deferred. Unsatisfied here but satisfied on a known alternative substrate gives rerouted. The spec is emphatic that none of these is an error, a timeout, or a default; each is a valid computational result. A precision shortfall that no waiting can fix lands in structurally impossible, and the objective is refused rather than attempted.

Separation from authorization. The spec enforces a strict separation between whether an operation can structurally exist and whether it is permitted. These are computed in architecturally separate subsystems with no bidirectional dependency and combined only at a joint evaluation gate. The illustrative surgical example makes the point: an operator may be governance-authorized to perform a procedure yet biologically incapable at the moment because assessed motor precision falls below the

procedure's requirement, and the system's response is to defer or reroute. For a robot the same logic holds: being cleared to run a task does not make the arm capable of the tolerance.

Physical temporal dynamics. The spec extends temporal executability forecasting to embodied systems by projecting physical state forward. Its worked case: a motor objective requiring sustained high-torque actuation may be immediately executable but become temporally impossible as actuator temperatures approach thermal limits, and the forecast detects the impending collapse and defers or reroutes before the limit is reached. This is what separates a task the robot can never do from one it can do only in a future window.

Uncertainty for physical systems. The spec notes that physical state estimation is inherently uncertain, sensors are noisy, actuator performance degrades non-linearly, terrain is partially observable, and that uncertainty bounds on physical dimensions are typically wider than on computational ones. It responds by applying wider confidence intervals to physical forecasts and more conservative execution-synthesis thresholds for motor objectives. Precision gating should inherit this conservatism: near the edge of the precision envelope, the correct move is to refuse or defer, not to gamble.

Auditable record. Every determination produces a structured record: the evaluated substrate, the extracted requirements, the retrieved envelope, the per-dimension match results, the aggregate determination, and the uncertainty bounds. A refusal is therefore explainable after the fact, which matters when a supervisor asks why the robot declined a task.

How to Approach the Build

1. **Model the physical envelope as a live object.** Define a structured descriptor with per-dimension value ranges and comparison operators, following the spec's dimensions: degrees of freedom, force and torque limits, reach envelope,

locomotion, sensory modalities, power budget, and motor precision. Wire it to live telemetry so battery charge, actuator temperature, and sensor health update it continuously. Do not seed it from a static datasheet and leave it.

2. **Give each motor objective a formal requirements vector.** For every task type, specify the minimum physical characteristics required, including the tolerance the task must hit. The gate compares this vector against the envelope; if you cannot state a task's precision requirement numerically, the gate cannot evaluate it.
3. **Implement dimension-by-dimension matching with three outcomes.** For each required dimension emit satisfied, unsatisfied, or conditionally satisfiable, per the spec. Precision is a first-class dimension here, compared like force or reach.

An illustrative interface sketch, faithful to the spec's structure and not a shipping API:

```
# Illustrative only. You implement this against your own stack.
determination = evaluate_capability(
    objective_requirements,   # includes required precision / tolerance
    physical_envelope,        # live: DOF, force, reach, locomotion,
                              #       motor precision, power, sensors
    temporal_uncertainty_state
)
# determination.aggregate in
#   { POSSIBLE, IMPOSSIBLE, DEFERRED, REROUTED }
```

4. **Compose to the four aggregate outcomes and route on them.** Map possible to planning, impossible to refusal, deferred to a temporal trigger, rerouted to an alternative robot or cell. A precision shortfall with no conditional path must land in impossible and be refused.
5. **Run the gate before the motion planner.** Enforce the ordering the spec enforces: determine whether an executable form can exist, and only on an affirmative determination proceed to synthesis. This is the whole point, no infeasible

plan is ever built.

6. **Add physical temporal forecasting.** Project actuator temperature, battery charge, and sensor drift forward and distinguish immediate executability, deferred executability, and temporal impossibility, so a task blocked only by a thermal window defers instead of being permanently refused.
7. **Apply wider margins near the precision edge and keep the record.** Follow the spec's conservative posture for physical uncertainty, and persist the structured determination record for every refusal so decisions are auditable.
8. **Keep the gate independent of authorization.** Compute feasibility in its own subsystem and combine with permission only at the joint gate, so an authorized-but-incapable task is still refused.

What This Does Not Give You

This is an architecture, not a drop-in library. There is no package to install and no SDK to import; the interface sketch above is illustrative and you implement the real thing against your own controllers, kinematics, and telemetry. The approach is disclosed in a patent filing. It has not been presented here as a shipping product, and this guide reports no benchmarks, accuracy figures, or performance guarantees, because the spec states none and inventing them would be dishonest.

The gate is only as good as your models. It compares a requirements vector against an envelope; if your envelope does not faithfully represent the robot's true motor precision, or your objective does not state its real tolerance, the determination will be wrong in the same direction as your model error. The spec itself notes that physical estimates carry wide uncertainty, which is why the conservative margins matter and why this gate reduces, rather than eliminates, infeasible attempts. It also does not plan motion, calibrate your hardware, or measure precision for you; those remain your responsibility. And it governs whether a task can structurally exist, not whether it is wise or permitted, permission is a separate determination by design.

Disclosure Scope

The architecture described in this guide, gating a robot's motor objectives against a real-time physical capability envelope, including motor precision, and refusing objectives that fall outside that envelope, is disclosed in United States Patent Application 19/647,395. This guide is educational. It explains an approach a developer can build and does not constitute a warranty, a performance guarantee, or an offer of software. Every mechanism described is traceable to the filed disclosure; where the disclosure is silent, this guide makes no claim.

Capability Awareness (/capability-awareness)

All 40 steps → (/inventive-steps)

Know what you can do before you try.

Chapter 6 (/patents/19-647395/chapters/capability)

PRIMARY TECHNICAL DISCLOSURE

- Capability-, Time-, and Uncertainty-Aware Execution in Autonomous Computational Networks (/articles/capability-time-and-uncertainty-aware-execution-in-autonomous-computational-networks)

SECONDARY TECHNICAL

- Capability as First-Class Computational State (/articles/capability-awareness/first-class-state)
- Capability Envelope for Substrates (/articles/capability-awareness/capability-envelope)
- Temporal Executability Forecasting (/articles/capability-awareness/temporal-forecasting)
- Uncertainty as First-Class Propagated Variable (/articles/capability-awareness/uncertainty-propagation)
- Capability Envelope Negotiation (/articles/capability-awareness/envelope-negotiation)
- Capability Genealogy Tracking (/articles/capability-awareness/genealogy-tracking)
- Biological Capability Extension (/articles/capability-awareness/biological-extension)
- Network-Level Capability Pressure (/articles/capability-awareness/network-pressure)
- Capability-Permission Distinction (/articles/capability-awareness/permission-distinction)

- Capability-Native Computation (/articles/capability-awareness/native-computation)
- Execution Synthesis (/articles/capability-awareness/execution-synthesis)
- Agent Behavior Under Constraints (/articles/capability-awareness/constrained-behavior)
- Predictive Network Planning Under Capability Pressure (/articles/capability-awareness/predictive-planning)
- Multi-Agent Contention Resolution (/articles/capability-awareness/contention-resolution)
- Capability Robustness Mechanisms (/articles/capability-awareness/robustness-mechanisms)
- Capability-Modulated Discovery Traversal (/articles/capability-awareness/discovery-constraint)
- Capability as Confidence Input (/articles/capability-awareness/confidence-input)
- Embodied Capability Envelopes (/articles/capability-awareness/embodied-envelopes)
- Substrate Resource Negotiation (/articles/capability-awareness/resource-negotiation)
- Place-Level Capability Envelope (/articles/capability-awareness/place-level-capability)
- Observation Staleness and TTL Governance (/articles/capability-awareness/observation-staleness-ttl)

APPLICATIONS · GENERAL

- Robotic Capability Assessment Before Commitment (/articles/capability-awareness/robotic-assessment)
- Edge Computing Resource Governance Through Capability Envelopes (/articles/capability-awareness/edge-resource-governance)
- Capability-Aware Surgical Robots: Refusing Procedures Beyond Calibrated Precision (/articles/capability-awareness/surgical-robotics)
- Capability-Aware Agricultural Robots: Terrain and Field-Condition Safety (/articles/capability-awareness/agricultural-robotics)
- Capability-Aware Autonomous Mining Equipment: Refusing Operations When Conditions Exceed the Safe Envelope (/articles/capability-awareness/mining-operations)
- Weather-Aware Autonomy for Offshore Energy Platforms (/articles/capability-awareness/offshore-energy)
- Capability Awareness for Warehouse Logistics Robotics (/articles/capability-awareness/warehouse-logistics)
- Capability Awareness for Construction Robotics (/articles/capability-awareness/construction-robotics)
- CORS and NTRIP RTK Without a Centralized Reference Network: Fleet-Emergent Precision Positioning (/articles/capability-awareness/cors-rtk-replacement)

- Self-Calibrating Autonomous Fleets: Precision Positioning Without Fixed Reference Infrastructure (/articles/capability-awareness/fleet-self-calibration)

APPLICATIONS · SPECIFIC

- Tesla FSD vs Capability-Aware Autonomy: What a Capability Envelope Adds (/articles/capability-awareness/tesla-fsd)
- John Deere Autonomous Tractors vs Capability-Governed Field Autonomy (/articles/capability-awareness/john-deere)
- KUKA Alternative: Capability-Aware Industrial Robots Beyond Static Parameters (/articles/capability-awareness/kuka)
- FANUC vs Capability-Aware Robot Execution (/articles/capability-awareness/fanuc)
- Universal Robots and Capability-Aware Cobots: Beyond Force Limiting (/articles/capability-awareness/universal-robots)
- ABB Robotics vs Capability-Aware Robot Execution (/articles/capability-awareness/abb-robotics)
- Yaskawa Motoman vs Capability-Aware Robot Execution (/articles/capability-awareness/yaskawa)
- Doosan Robotics Alternative: Governed Cobots With Capability Self-Knowledge (/articles/capability-awareness/doosan-robotics)
- Does Agility Robotics Digit Have Capability Awareness? (/articles/capability-awareness/agility-robotics)
- Figure AI Alternative: Governed Humanoid Execution With Capability Awareness (/articles/capability-awareness/figure-ai)
- Trimble RTK vs Capability-Aware Positioning Execution (/articles/capability-awareness/trimble-rtk)
- Hexagon SmartNet vs Capability-Aware Instrument Fleets (/articles/capability-awareness/hexagon-survey)
- Boston Dynamics (Spot / Atlas) vs a capability-envelope executability gate: how autonomy decides whether an action can structurally exist (/articles/capability-awareness/boston-dynamics)

Capability Awareness overview → (/capability-awareness)