LQR as a *Nav2 Controller Plugin* for TurtleBot3 Path Tracking

A C++ Linear Quadratic Regulator integrated into the ROS 2 navigation stack, benchmarked against DWB and MPPI across a 3×2 controller–planner matrix — averaged over 3 independent trials per configuration.

Tommy Oh · Ansh Aggarwal · Daniel Senteu · JunHang Wu

Simon Fraser University | {koa18, aaa275, dms26, jwa337}@sfu.ca

SFU

CMPT 419 · SPRING 2026

COURSE PROJECT · NAV2 · GAZEBO · ROS 2 HUMBLE

6/6	25.5s	0.052m	20 Hz
CONFIGS REACHED GOAL	LQR TIME-TO-GOAL	LQR MEAN CTE	CONTROL RATE

01 Abstract

LQR sits squarely between DWB and MPPI — near DWB's speed, approaching MPPI's smoothness — with *zero* gain tuning.

We implement a **Linear Quadratic Regulator** path-tracking controller as a C++ Nav2 plugin for ROS 2 Humble, deployed on a simulated TurtleBot3 Burger in Gazebo. The controller linearizes unicycle dynamics in body-frame coordinates, solves the Discrete Algebraic Riccati Equation (DARE) once at startup for a steady-state gain matrix, and produces smooth velocity commands at 20 Hz.

We benchmark LQR against **DWB** (Dynamic Window Approach) and **MPPI** (Model Predictive Path Integral) across two planners (NavFn and Smac 2D), yielding a 3×2 comparison matrix. Each configuration was run **3 times** and metrics averaged. All six configurations reached the goal in every trial (18/18).

HEADLINE RESULT

Averaged over **3 trials per configuration** (18/18 reached goal): LQR 25.5 s · 0.052 m CTE · dv/dt 0.19 — within 11% of DWB on speed, indistinguishable on accuracy, and 40–55% smoother on linear and angular commands.

02 System Architecture

The LQR controller integrates into the Nav2 stack as a `pluginlib` plugin implementing `nav2_core::Controller`. Implementation is split to keep pure mathematics separate from ROS 2 glue.

Global Planner · NavFn / Smac 2D
→ nav_msgs/Path

↓

LQR Controller Plugin · 20 Hz
computeVelocityCommands(pose, vel, goal_checker)

↓

/cmd_vel → TurtleBot3 Burger
v ∈ [0, 0.22] m/s · ω ∈ [−2.84, 2.84] rad/s

01	findClosestPoint	**02**	findLookaheadPoint · 0.5 m
03	computePathHeading	**04**	computeBodyFrameError
05	δu = −K · e	**06**	clamp(v, ω)

03 Method — LQR in Body Frame

TurtleBot3 Burger is modeled as a unicycle. Tracking error is projected into the path's local frame so the LQR cost separately penalizes lateral and longitudinal deviation. With $v_{\text{ref}} = 0.22$ m/s and $\omega_{\text{ref}} = 0$ constant, the linearized system (A_d, B_d) is constant — so DARE is solved **once at startup**, not every control cycle.

CONTROL LAW · GAIN · COST

$$\delta\mathbf{u} = -K\,\mathbf{e}, \qquad K = \left(R + B_d^\top P B_d\right)^{-1} B_d^\top P A_d$$

$$Q = \mathrm{diag}(1.0,\ 3.0,\ 1.0), \qquad R = \mathrm{diag}(1.0,\ 0.5)$$

$q_{\text{lat}} = 3.0$ prioritizes tight lateral path-tracking. These are the original spec gains — **no tuning was performed**.

04 Benchmark Results

Identical Gazebo world (`turtlebot3_world`), start $(-2.0, -0.5)$, goal $(2.0, 0.5)$. Each configuration was run **3 times**; metrics reported as mean ± std from rosbag odometry at ≈20 Hz.

3 × 2 Controller × Planner Matrix · n = 3

ORANGE = LQR · BOLD = BEST

CONFIGURATION	MEAN CTE	MAX CTE	STD CTE	DV/DT	DΩ/DT	TIME (S)	✓
LQR + NavFn	0.051	0.233	0.066	0.187	0.946	25.52	3/3
LQR + Smac 2D	0.054	0.205	0.059	0.183	0.478	25.51	3/3
DWB + NavFn	0.049	0.257	0.077	0.300	1.158	23.92	3/3
DWB + Smac 2D	0.048	0.199	0.059	0.311	1.062	**22.93**	3/3
MPPI + NavFn	**0.012**	**0.055**	**0.011**	**0.086**	**0.169**	37.65	3/3
MPPI + Smac 2D	0.081	0.285	0.077	0.097	0.267	36.48	3/3

LQR TIME-TO-GOAL
25.5 s
Within 11% of DWB's fastest (22.9 s) — and 32% faster than MPPI (36.5 s).

LQR MEAN CTE
0.052 m
Within 8% of DWB (0.048 m). MPPI + NavFn wins accuracy at 0.012 m.

LQR SMOOTHNESS
0.19
dv/dt — **40% smoother** than DWB (0.30); MPPI leads at 0.09.

ANGULAR SMOOTHNESS
0.48
LQR + Smac dω/dt — **55% smoother** than DWB + Smac (1.06).

05 6-Combo Benchmark

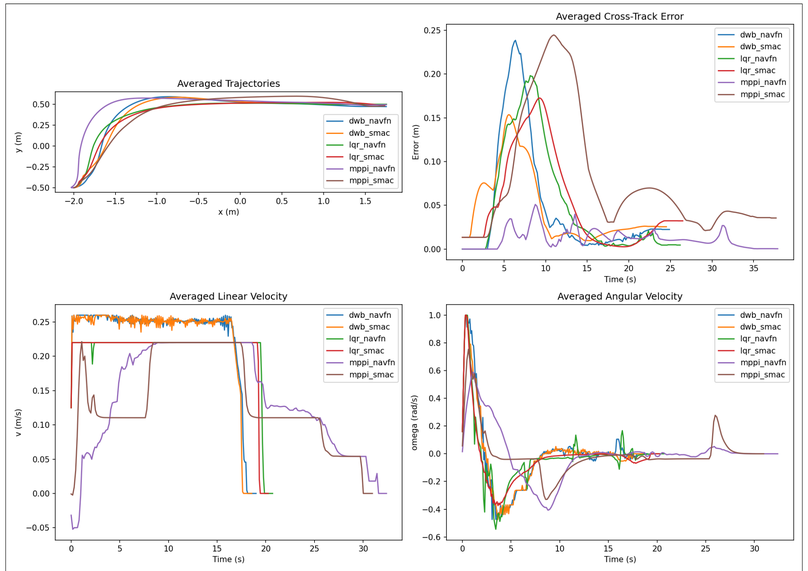

FIG. 1 Averaged results across 3 trials per configuration. **TOP-LEFT:** trajectory overlay — MPPI + NavFn (purple) cuts the corner most aggressively. **TOP-RIGHT:** cross-track error — MPPI + NavFn stays below 0.06 m throughout; DWB and LQR peak near 0.20–0.25 m during the turn. **BOTTOM:** linear and angular velocity — DWB saturates at the 0.26 m/s cap with oscillations; LQR holds a clean 0.22 m/s plateau; MPPI is smoothest but slowest.

06 Discussion

Three distinct profiles. The 3×2 matrix reveals a clean separation: **DWB** is fastest (22.9–23.9 s) but jerkiest; **MPPI + NavFn** is most accurate (0.012 m CTE) and smoothest, but slowest (37.7 s); **LQR** sits between, trading ~2.6 s for a 40–55% smoother command profile.

LQR vs DWB. Mean CTE is statistically indistinguishable (0.052 vs 0.048 m). LQR pays ~11% in time-to-goal but delivers markedly smoother linear and angular commands — important for battery life and actuator wear on real hardware.

LQR vs MPPI. MPPI + NavFn dominates accuracy and smoothness but takes 32% longer. MPPI + Smac 2D loses this advantage (CTE 0.081 m) — the path Smac produces interacts poorly with MPPI's sampled rollouts.

Planner effects. LQR is **planner-invariant** (NavFn 25.52 s vs Smac 25.51 s). DWB and MPPI are both sensitive to planner choice — Smac helps DWB marginally, hurts MPPI substantially.

Course relevance. The Dubins car from A1/A2, linearized in A3, is the model we deploy; DARE is the infinite-horizon formulation from *Underactuated Robotics* Ch. 8.

3 trials per configuration
Tight std (≤0.003 m CTE) but wider Monte Carlo would strengthen claims.

Simulation only
Real hardware would need re-tuning for slip, latency, and sensor noise.

Fixed start/goal pair
Longer routes and obstacle-dense scenes remain untested.

Constant reference
LQR gains are optimal at $v_{\text{ref}} = 0.22$; varying speeds would need gain scheduling.

07 Conclusions

Averaged over 3 trials per configuration, our LQR plugin reaches the goal in 25.5 s with a mean CTE of 0.052 m — indistinguishable from DWB on accuracy, within 11% on speed, and 40–55% smoother on linear and angular commands. MPPI + NavFn remains the accuracy leader (0.012 m CTE) at the cost of 32% longer runtime. The result reframes LQR not as a toy textbook algorithm but as a **balanced middle path** between reactive windowing and stochastic sampling — a competitive, production-ready Nav2 choice when **command smoothness and predictability** matter.