Learning-Based vs Classical Optimal Control on the Quanser QBot 3

Camille A. Campbell • Electrical and Computer Engineering, Auburn University • Md Hasanuzzaman Chowdhury Joy, Bosan Lian/ SURE Program • Summer 2026

CLASSICAL | LQR + FEEDFORWARD *vs* **LEARNED | NEURAL NETWORK (CEM)**

1. Motivation

A robot following a path needs a rule that turns tracking error into wheel commands. Classical optimal control gives one answer: linearize the error dynamics, solve a Riccati equation, and the resulting gain is optimal for that linearization. Reinforcement learning gives another: let a neural network find the rule by trial and error, without handing it a model.

Most published comparisons favor learning. A 2025 study of quadrotor tracking argues those comparisons are usually unfair by accident [2]. The learned controller typically receives three things the classical one does not: the task's objective function, training data drawn from the test task, and feedforward knowledge of the upcoming reference. Correct those asymmetries and the reported gap mostly closes.

This project tests that claim on the Quanser Qbot 3. Three controllers, one task, one cost. The third is the point: an LQR whose design weights are tuned on the task with the same optimizer and the same budget the neural network receives.

2. Platform: Quanser QBot 3

The QBot 3 is a differential-drive robot utilizing a Raspberry Pi 4 and MATLAB/Simulink via Quanser QUARC for real-time control, equipped with comprehensive odometry and an Intel RealSense camera. All results are derived from a dynamic simulator strictly calibrated to the robot's measured parameters: 38mm wheels, a 235 mm wheelbase, 3.5 kg mass, 0.15 s motor lag, and a 15 rad/s wheel limit. Simulation was a practical necessity rather than a stylistic choice; optimizing five seeds across two methods required roughly 320 total hours of driving (~32 hours per controller)—a continuous workload that the physical hardware's battery could not sustain.

3. Model & Task

The robot is modeled as a unicycle, exact for a differential drive [1]. State x = [p_x, p_y, θ]; inputs are wheel speeds ωL, ωR, combining into forward speed v and yaw rate ω:

$$v = \frac{r(\omega_R + \omega_L)}{2} \quad \omega = \frac{r(\omega_R - \omega_L)}{L}$$

$$\dot{p}_x = v\cos\theta \quad \dot{p}_y = v\sin\theta \quad \dot{\theta} = \omega$$

Error is expressed in the robot's body frame: e_x ahead or behind, e_y off to the side, e_θ in heading. This is the only signal any controller sees. The task is a 1 m circle at 20 s per lap, starting 15 cm off the path so convergence is exercised. Constant reference speed and curvature keep the linearization time-invariant, so one gain covers the loop.

4. Classical: Two LQRs

Linearizing the error dynamics gives $\dot{x} = A\,x + B\,u$. LQR solves the algebraic Riccati equation for P and sets $K = R^{-1}B^{T}P$. By defining these variables, we can automatically solve these complex equations using just one line of MATLAB: lqr(A,B,Qd,Rd). The control law adds feedforward:

$$u = \begin{pmatrix} v_{ref} \\ \omega_{ref} \end{pmatrix} - Ke$$

Two versions were built.

LQR-untuned takes the obvious choice: Qd, Rd equal to the weights that define the evaluation cost. This is what most papers use as their baseline.

LQR-tuned treats Qd, Rd as five free parameters and optimizes them on the dynamic plant, using the same optimizer and the same 3,840-episode budget given to the neural network. This is the symmetric baseline [2] argues for and that most comparisons leave out.

5. Learned: Neural Network

Neural Network Architecture:

- Replaces the classical -K e control law with a two-layer network.
- Configured with 16 hidden units, tanh activation, and 98 total parameters.
- Maintains identical inputs, outputs, and feedforward structures to the LQR.

Training Methodology:

- Utilizes the Cross-Entropy Method [3] for optimization.
- Evaluates 64 random parameter sets per iteration, selecting the top eight to update the search distribution.
- Completes 60 iterations, totaling 3,840 training episodes.

Optimization Strategy:

- The network learns directly from the dynamic plant (including motor lag and saturation) without knowledge of system matrices A, B, or K.
- Both LQR and Neural Network controllers operate with an identical optimizer and budget.
- The primary difference lies in parameterization: LQR tunes 5 numbers, whereas the network tunes 98.

6. Fair-Comparison Protocol

- **Fair Comparison:** Following previous studies [2], everything outside of the controllers including the goals, limits, and processing budget was kept exactly the same.
- **The Plant:** The simulated robot was programmed with real-world physics, not just idealized math. It included a 3.5 kg mass, rotational momentum, friction, tire slip, and a 0.15-second delay before the motors responded to commands.
- **Robustness Testing:** To test reliability, the controllers were forced to navigate 40 varied versions of the robot. These variations featured randomized changes to the robot's weight, wheel size, tire slip, and motor lag.
- **Training Seeds:** Every result is the average of five independent runs. This ensures that a successful test was the result of a genuinely good controller, rather than just a lucky random starting point.

7. Results & Discussion

	LQR untuned	LQR tuned	Neural network
Cost J	17.02	**7.27 ±0.00**	9.02 ±0.30
RMSE (cm)	6.63	**5.12 ±0.00**	5.47 ±0.15
RMSE perturbed	34.2 ±49.1	32.6 ±0.3	33.2 ±0.5
Free parameters	5	**5**	98
Training episodes	0	3,840	3,840

The Baseline Illusion: The neural network shows a 47% improvement over *untuned* LQR—a common but misleading benchmark.
Tuning > Architecture: Given equal optimization budgets, *tuned* LQR actually outperforms the neural network by 19%.
Parameter Efficiency: LQR-Tuned achieves superior results using just 5 parameters, compared to the network's 98.
Perfect Consistency: Across five seeds, LQR-Tuned hit the exact optimal score every time, whereas network performance fluctuated.
Hardware Robustness: Untuned LQR crashed on severe hardware perturbations; both tuned controllers failed equally only under extreme mismatches.

8. Conclusions & Limitations

Conclusion Machine learning only outperformed an untuned baseline, failing to beat finely tuned classical optimal control. Given an identical optimization budget, the Linear Quadratic Regulator (LQR) matched or exceeded the neural network across all metrics. LQR achieved this using just 5 parameters versus 98, offering perfect run-to-run reproducibility and mathematical stability guarantees that neural networks inherently lack.

Limitations & Future Work Future work will transition these simulations to physical hardware using QUARC. Current limitations include evaluating a single trajectory, excluding real-world sensor noise, and utilizing the Cross-Entropy Method, which may favor LQR's low parameter count over gradient-based methods like PPO. Although both tuned controllers failed under extreme hardware perturbations, identifying these physical limits establishes a valuable robustness baseline.

9. References

[1] Cornejo, J., Magallanes, J., Denegri, E., Canahuire, R. Trajectory Tracking Control of a Differential Wheeled Mobile Robot: A Polar Coordinates Control and LQR Comparison. IEEE INTERCON, pp. 1–4, 2018.

[2] Kunapuli, P., Welde, J., Jayaraman, D., Kumar, V. Leveling the Playing Field: Carefully Comparing Classical and Learned Controllers for Quadrotor Trajectory Tracking. arXiv:2506.17832, 2025.

[3] Rubinstein, R.Y., Kroese, D.P. The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning. Springer, 2004.

RESULTS: same cost, same observation, same feedforward, same optimizer, same budget. The only difference is the controller.