A Comparison of Reinforcement Learning and Optimal Control Techniques to Control a Non-Holonomic Vehicle

Blake Edwards [1] Shruti Garg [2] Erin Menezes [3]

Abstract

Autonomous driving control strategies have traditionally relied on classical optimal control techniques that solve a trajectory optimization problem using an explicit vehicle model. In recent years, however, the field has seen significant progress with learning-based controllers, particularly imitation and reinforcement learning (RL), which learn policies directly from data and interaction with the environment, rather than relying on explicit models. In this project, we compare these two paradigms by implementing three control strategies: two based on classical optimal control and one on reinforcement learning, applied to a vehicle navigating a sequence of waypoints. Using a 2D, non-linear, non-holonomic kinematic model, we evaluate a minimum-time controller, a nonlinear quadratic regulator, and an RL-based policy. All approaches successfully reached all waypoints, with the RL method achieving the fastest completion time (79s); however, this came at the cost of increased control effort, likely due to high amounts of path jitter.

The code for this project can be found here: https://github.com/blakete/MIT-18.065-Final-Project

1. Introduction

Recent breakthroughs in autonomous driving have come from replacing traditional classical control algorithms with machine-learned policies. As AI's role in robotics continues to expand, it's essential for practitioners to not only learn how to deploy these models but also to understand their inner workings and recognize the scenarios in which they perform best—so they can be harnessed to their full potential.

Apart from the interesting control models presented with AI based autonomous driving, typical cars also use constrained dynamics that make the model generally difficult to work with. These vehicles are considered non-holonomic as the directions they can move is constrained depending on their previous velocity (heading direction). That is, there is a constraint on their movement that depends on the tangential space to their configuration space.

Various control methods have been applied to a non-holonomic vehicle model. From a traditional optimal control perspective, this constraint adds a layer of interesting complexity by moving the problem into necessarily non-convex regimes. That said, the system is differentially flat given the correct control inputs: heading and speed. While industry seems to be moving away from such an approach to self-driving, it offers intuition into dealing with non-holonomic constraints (that show up in more applications than cars) and other more complex optimal control settings.

Recently, Waymo has been proving in the industry that the self-driving technology is mature enough to be deployed, if one uses learning based methods. One of the recent papers that give insight into their approach to the problem suggests supplementing learning off of collected expert data aka humans driving cars (behavior cloning or imitation learning) with learning how to maximize a reward function in its environment (reinforcement learning). In this paper, we explore the efficacy of similar RL techniques (specifically a soft actor critic on policy learning model) and contrast it with optimal control solutions.

2. Problem Statement

In this project, our goal is to steer a non-holonomic vehicle around a pre-defined race course by passing through a sequence of N waypoints in minimum total time, while always respecting the vehicle's kinematic model and actuator limits. We also measure the total control effort to assess the trade-off between speed and energy.

To be precise, in our problem we are given:

- **Vehicle model:** a 2D bicycle-model

$$\dot{x} = f(x,u), \quad x = (x,\, y,\, \theta), \quad u = (u_s,\, u_\phi),$$

[1]Department of Aeronautics and Astronautics [2]Department of Electrical Engineering and Computer Science [3]Department of Mechanical Engineering. Correspondence to: <blakete, sgrg, emenezes@mit.edu>.

- **Waypoints:** an ordered list $\mathcal{W} = \{w_1, \ldots, w_N\}$ with capture radius r_c (see Figure 1),
- **Initial state:** $x(0) = x_0$,

Given these components, our two solution paradigms differ only in what we solve for:

- **Optimal Control:** For each waypoint segment $i = 1, \ldots, N$, we compute a time-varying control trajectory

$$u_i(t) = (u_s^i(t),\, u_\phi^i(t)), \quad t \in [0, T_i],$$

  and its switching time T_i.

- **Reinforcement Learning:** We learn a policy π that, at fixed intervals Δt, outputs control actions

$$u_k = \pi\big(x_{1:2}(k\Delta t) - w_i,\ \theta(k\Delta t)\big).$$

  Here Δt (and thus all switching instants $k\Delta t$) is held fixed, so the only decision variables are the controls $\{u_k\}$.

Based on the resulting control trajectories, we evaluate performance using two metrics:

$$T_N = \text{total race time},$$
$$E = \int_0^{T_N} \|u(t)\|_2^2\, dt = \int_0^{T_N} \big(u_s^2(t) + u_\phi^2(t)\big)\, dt,$$

where T_N is the time at which the final waypoint is captured, and E quantifies total control effort over the trajectory.

Solution Strategies

We contrast two high-level implementations of these paradigms:

1. Optimal Control Starting from the current state x, solve a time-optimal control problem (and an NLQR variant) to each waypoint in turn. For segment i, the solver returns

$$[T_i,\, X_i(\cdot),\, U_i(\cdot)],$$

which we execute over $[0, T_i]$, update

$$x \leftarrow X_i(T_i), \quad E \leftarrow E + \int_0^{T_i} \|U_i(t)\|_2^2\, dt, \quad t \leftarrow t + T_i,$$

and proceed to the next waypoint. The total race time T_N is the sum of all T_i. Please see Algorithm 1 for how the optimal control based solutions were used to perform the multi-waypoint race. Additionally, please see Sections 4.1 and 4.2 for full optimal control problem formulation treatments.

2. Reinforcement Learning We train a neural-network policy π to produce a control action every Δt, based on the relative position and heading to the current waypoint. During evaluation, we step π, apply u_k for Δt, update the state, and advance to the next waypoint once $\|x_{1:2} - w_i\| \leq r_c$. A timeout $T_{\max}$ prevents infinite loops. The total time T_N equals the number of steps times Δt. Please see Algorithm 2 for how we used our trained policy to execute the full multi-waypoint race. Additionally, please see Sections 5.3.1 for full problem RL problem formulation treatment.

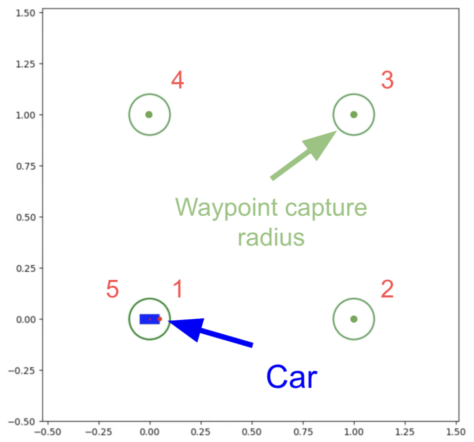

Figure 1. Race course with waypoints $w_1, \ldots, w_N$ to be sequentially captured within radius r_c.

3. Vehicle Kinematic Model

The underlying non-linear system kinematics and any observed outputs of a general dynamic system can be set up using a state space formulation as below:

$$\begin{aligned} \dot{x}(t) &= f\big(x(t), u(t), t\big), \quad x(0) = x_0, \\ y(t) &= h\big(x(t), u(t), t\big), \end{aligned} \tag{1}$$

where:

- $x(t) \in \mathbb{R}^n$ is the state of the system,
- $u(t) \in \mathbb{R}^m$ is the control input,
- $y(t) \in \mathbb{R}^\ell$ is the measured output,
- $f\colon \mathbb{R}^n \times \mathbb{R}^m \times [0, T] \to \mathbb{R}^n$ is the state-transition map,
- $h\colon \mathbb{R}^n \times \mathbb{R}^m \times [0, T] \to \mathbb{R}^\ell$ is the output map,
- $x_0 \in \mathbb{R}^n$ is the given initial state.

The car used in the paper consists of a rigid body frame with two front wheels and two back wheels. Only the back wheels are ”driven”, and provide forward movement to the car. The front wheels turn together to steer the car.

Typical four-wheel, back-wheel-driven cars present challenges for controllers as they operate in a two-dimensional space defined by three degrees of freedom, yet they can only control two degrees. However, as they are differentially flat under the choice of controls, they can reach any point in the two-dimensional space given some finite time.

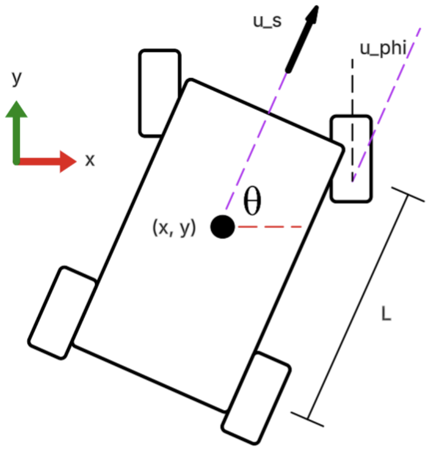

Figure 2. Diagram of the four-wheeled non-holonomic car

To plan motion, we first need a proper kinematic model. We base ours off the bicycle model (Taheri, 1990), which solves for the vehicle's velocities in the world frame using

$$\begin{aligned}\dot{x}(t) &= u_s(t)\cdot\cos(\theta(t))\\ \dot{y}(t) &= u_s(t)\cdot\sin(\theta(t))\\ \dot{\theta}(t) &= \frac{u_s(t)}{L}\cdot\tan(u_\phi(t))\end{aligned}\tag{2}$$

where (x, y, θ) denote the vehicle's pose in the world frame, u_s is the vehicle's driving speed (speed of back wheels), u_ϕ is the vehicle's heading (angle of front wheels), and L is the distance from the rear to front wheel.

We can then arrange these vehicle kinematics in the form of Eq. 1, as

$$\dot{x}(t) = \begin{bmatrix}\dot{x}(t)\\ \dot{y}(t)\\ \dot{\theta}(t)\end{bmatrix} = \begin{bmatrix} u_s(t)\,\cos\big(x_3(t)\big)\\ u_s(t)\,\sin\big(x_3(t)\big)\\ \frac{u_s(t)}{L}\,\tan\big(u_\phi(t)\big)\end{bmatrix},\tag{3}$$

where:

- $x(t)$ is the vehicle's x-coordinate,
- $y(t)$ is the vehicle's y-coordinate,
- $\theta(t)$ is the vehicle's heading angle,
- $u_s(t) \in \mathbb{R}$ is the commanded forward speed,
- $u_\phi(t) \in \mathbb{R}$ is the commanded steering angle,
- $L > 0$ is the distance between axles,

4. Optimal Control Formulation

Optimal control is a branch of control theory that uses optimization methods to solve control problems. The objective function is typically a function of state, control, and time that is integrated across the time frame of the problem, though there are optimal control problems like "minimum-time" formulations that fall outside this category. Additionally, model dynamics, path constraints, and control constraints are used to bound the optimization problem.

This section presents detailed implementations of two approaches to control the non-holonomic vehicle that vary only on their objective function: minimum-time and a Non-Linear-Quadratic-Regulator formulation.

4.1. Minimum Time Formulation

A minimum time objective function for an optimal control problem has a very simple form; the only objective is to minimize the final time T.

Thus, given $x_0 \in \mathbb{R}^3$ and a target way point $(x_{f,1}, x_{f,2})$, choose $x(t)$, $u(t) = [u_s, u_\phi]$ and T to

$$\begin{aligned}&\min_{x,u,T} \quad T\\ &\text{s.t.}\\ &\dot{x} = f(x,u) && \text{Dynamic Constraint,}\\ &|u| \le u_{\max} && \text{Control Limits,}\\ &T \ge 0 && \text{Time Limits,}\\ &x(0) = x_0 && \text{Initial State,}\\ &\|x(T) - x_f\|_2 \le 0.1 && \text{Final Captures Radius}\end{aligned}\tag{4}$$

This optimization problem minimizes the final time T while maintaining the vehicle dynamics and control limits. It also constrains the final time to be positive. The boundary conditions are that the initial state is given as x_0 and the final state must fall within a certain radius of the given target way-point x_f.

Lagrangian Lagrangian optimization is used to define the constrained optimization problem, which includes the objective function J and the sum of constraints multiplied by Lagrange multipliers p_i and v_i, which define the sensitivity of the objective to each constraint. Constraints are either equalities $\mathcal{E}$ or inequalities $\mathcal{I}$. We define the Lagrangian $\mathcal{L}$ as

$$\mathcal{L}[x,u,p,v] = J + \sum_{i\in\mathcal{E}} p_i\, c_i + \sum_{i\in\mathcal{I}} v_i\, c_i \tag{5}$$

$$= T + \int_0^T \Big[p^\top\big(\dot{x} - f(x,u)\big) + v_u^\top\big(|u| - u_{\max}\big)\Big]\,\mathrm{d}t + v_T\,(-T) \tag{6}$$

Karush-Kuhn-Tucker Conditions (KKT) The Karush-Kuhn-Tucker conditions define the necessary conditions for optimality. To use these conditions however, there are a few assumptions made.

- *Linear Independence Constraint Qualification (LICQ)*: This assumes the gradients of active inequality constraints and equality constraints are linearly independent. Because the gradient of the objective function is a linear combination of these active constraint gradients, the subspace that this gradient lies in has a rank equal to the number of active constraints.
- *Differentiability*: The objective function and the constraints must be continuously differentiable. This does not hold for the control constraints as they have absolute values, but because their sub-differentials contain zero, the first sub-differential KKT condition holds. This condition is similar to the first KKT condition, except that $0 \in \partial\mathcal{L}/\partial x$ rather than $0 = \partial\mathcal{L}/\partial x$.
- *Lipschitz*: The gradients of the objective and constraints are locally Lipschitz continuous, meaning that within the bounds of the problem, functions are ”L-smooth”.

The KKT conditions for Equation 4 are:

- *Primal feasibility (constraints are satisfied):*
$$\dot{x} - f(x,u) = 0\ ; \quad |u| - u_{\max} \leq 0\ ; \quad -T \leq 0$$
- *Dual feasibility (constraints on Lagrange multipliers):*
$$v_u(t) \geq 0\ ; \quad v_T(t) \geq 0$$
- *Stationarity ($\partial\mathcal{L}/\partial DV = 0$), where DV is a decision variable:*
$$\frac{\partial\mathcal{L}}{\partial x} = 0\ ; \quad \frac{\partial\mathcal{L}}{\partial u} = 0\ ; \quad \frac{\partial\mathcal{L}}{\partial T} = 0$$
- *Complementary Slackness:*
$$\begin{aligned} v_u^\top\left(|u| - u_{\max}\right) &= 0 \\ v_T \cdot (-T) &= 0 \\ p^\top\left(\dot{x}(t) - f(x,u)\right) &= 0 \end{aligned}$$

Pontryagin Maximum Principle (PMP) While the KKT conditions are useful for optimization where the optimization variable is a static value, optimal control involves optimizing over an entire function based on its dynamics. The Pontryagin Maximum Principle presents necessary conditions for optimality that can be derived from the KKT conditions.

We first define the Hamiltonian as:

$$\mathcal{H}[x,u,p,v_u] = p^\top f(x,u) - v_u^\top\big(|u| - u_{\max}\big) \tag{7}$$

which is derived from the Lagrangian as the function inside the integral. In optimal control optimization, it is used to find the necessary conditions for optimality with a dynamical system.

From the Stationarity KKT condition, we can determine that

$$\frac{\partial\mathcal{L}}{\partial x} = \int_0^T \big[-\dot{p}(t)^\top x(t) - \mathcal{H}_x(x,u,p,v)\big]dt = 0 \tag{8}$$

where $\mathcal{H}_x$ is the partial of the Hamiltonian with respect to x. This gives the first necessary condition for optimal control:

$$\dot{p}(t) = -\mathcal{H}_x(x,u,p,v)^\top \tag{9}$$

The other two PMP conditions are:

$$\dot{x}(t) = \mathcal{H}_p(x,u,p,v)^\top \tag{10}$$
$$\mathcal{H}_u(x,u,p,v) = 0 \tag{11}$$

which align with the Stationarity and Primal Feasibility KKT conditions.

4.2. Non-Linear Quadratic Regulator (NLQR) Based Formulation

The classical Linear Quadratic Regulator (LQR) for an optimal control problem creates a objective function that is quadratic with the state and the control. Here we keep the quadratic objective with a regularization term on control input. However, our dynamics are non-linear. Instead of linearizing the dynamics around a certain state, we solve the Nonlinear Optimal Control Problem (NOCP) using nonlinear programming (NLP).

Given $x_0 \in \mathbb{R}^3$ and a target way-point $(x_{f,1}, x_{f,2})$, choose $x(t)$, $u(t) = [u_s, u_\phi]$, and T to

$$\begin{aligned} \min_{x,\,u,\,T} \quad & J_{\text{NLQR}} = \frac{1}{2}x(T)^\top H x(T) \\ & + \frac{1}{2}\int_0^T \big[\,x(t)^\top Q\, x(t) + u(t)^\top R\, u(t)\big]\, dt \\ \text{s.t.} \quad & \\ & \dot{x} = f(x,u) \qquad \text{Dynamic Constraint,} \\ & |u| \leq u_{\max} \qquad \text{Control Limits,} \\ & x(0) = x_0 \qquad \text{Initial State,} \\ & \|x(T) - x_f\|_2 \leq 0.1 \qquad \text{Final Captures Radius} \end{aligned} \tag{12}$$

where $Q, H \in \mathbb{R}^{3\times3}$ and $R \in \mathbb{R}^{2\times2}$. Q and R denote weighing matrices that determine how important the cost on the state versus cost on the control is, and H is the weight on the final state.

Lagrangian We define the NLQR Lagrangian L as

$$\mathcal{L}[x, u, p, v] = J + \sum_{i\in\mathcal{E}} p_i\, c_i + \sum_{i\in\mathcal{I}} v_i\, c_i \tag{13}$$

$$\begin{aligned} &= \frac{1}{2}\, x(T)^\top H\, x(T) \qquad (14) \\ &+ \frac{1}{2}\int_0^T \left[x(t)^\top Q\, x(t) + u(t)^\top R\, u(t)\right] dt \\ &+ \int_0^T p^\top\left(\dot{x} - f(x,u)\right) + v_u^\top\left(|u| - u_{\max}\right) dt \end{aligned}$$

Karush-Kuhn-Tucker Conditions (KKT) The KKT conditions for the NLQR controller are:

- *Primal feasibility* (constraints are satisfied):

$$\dot{x} - f(x,u) = 0\ ; \quad |u| - u_{\max} \le 0\ ;$$

- *Dual feasibility*: (constraints on Lagrange multipliers):

$$v_u(t) \ge 0$$

- *Stationarity condition*: ($\partial\mathcal{L}/\partial DV = 0$), where DV is a decision variable:

$$\frac{\partial\mathcal{L}}{\partial x} = 0; \quad \frac{\partial\mathcal{L}}{\partial u} = 0$$

- *Complementary Slackness:*

$$v_u\left(|u| - u_{\max}\right) = 0\ ; \quad p^\top\left(\dot{x}(t) - f(x,u)\right) = 0$$

Pontryagin Maximum Principle (PMP) In the same manner as the minimum-time problem, we can define the Hamiltonian of the NLQR formulation to be the integrand of the Lagrangian:

$$\begin{aligned}\mathcal{H}[x, u, p, v_u] = p^\top f(x,u) - v_u^\top\left(|u| - u_{\max}\right) \qquad (15) \\ -\left(x(t)^\top Q\, x(t) + u(t)^\top R\, u(t)\right)\end{aligned}$$

Plugging in this Hamiltonian to the Lagrangian in Eq. 13, we can find the identical three PMP conditions from the KKT conditions as in the minimum-time implementation, as listed in Eqs. 9, 10, 11.

4.3. Numerical Method Implementation

Both the minimum-time and NLQR optimal controllers were implemented as NLPs in Python using the CasADi package, an open-source tool for nonlinear optimization (Andersson et al., 2019).

Both controllers were set up as discrete time problems, each with 100 discretization steps across the solution horizon. The kinematic constraints of the vehicle model were added to the optimization problem for every time step using discrete integration. The discrete integration method was initially Euler integration, yet after encountering incorrect car movements, it was decided that a more accurate integration method should be used. A fixed-step fourth order Runge-Kutta discrete integrator was implemented to propagate dynamics between states in the final version.

Additionally, initial implementations resulted in unideal paths between points, including chattering in the vehicle's heading, extreme accelerations, and massive detours from the path. These changes were made to help both controllers output smoother and more realistic control sequences:

- Limits on accelerations prevented the car from speeding up too quickly (unrealistic),
- Path constraints provided reasonable bounds on the world positions (x, y) of the car around the given waypoints,
- Guesses of the car states, times, and controls at each time step provided the CasADi optimization problem an idea of what its output should be,
- The cost for the minimum time implementation was revised to include a small factor that limited the heading value from getting too big. This greatly improved the heading chattering, and the optimized paths became much smoother,
- Forced only forward motion - though the car should be able to move forwards and backwards, without a constraint on backward motion, the car would find a path that involved driving backwards for long segments,
- World velocity constraints were added on the changes of x, y and θ between every time step
- Increasing the number of time steps did not work as the computer ran out of memory, so this was left at 100, though could be a useful change in the future.

5. Reinforcement Learning

Inspired by the Waymo paper on combining Imitation Learning and RL (Lu et al., 2023), we implement an actor critic formulation for our Reinforcement Learning problem: Proximal Policy Optimization. An on-policy actor-critic RL method that is "first-order," it relies only on the gradient of the objective instead of also considering Hessians or further landscape derivatives. OpenAI's PPO documentation emphasizes how it avoids "performance collapse" by "stepping too far" (OpenAI, 2018). Armed with the tools we learned in 18.065, we can put this more specifically as: PPO uses the proximal operator (Algorithm 3 line 17) to stay close to current policy when optimizing the objective function (reward function) at each update. Given the scope of this project, the simplified model, and the lack of custom data for our setting, we thought PPO would teach us very relevant things to the BC-SAC while having a better chance at good performance. Moreover, some of our implementation details push the PPO implementation closer to a traditional Soft Actor Critic (as used in (Lu et al., 2023)). In this section we will cover design decisions we made, and observations that popped out to us during this implementation that were inspired by our 18.065 learning.

5.1. Actor-Critic Formulation

A popular model for RL, an actor critic model trains two different neural networks. One of these is trained as the "actor," that is it learns the feature mapping from observations of the outside world to the control output or optimal action to be taken. The other network is trained as the "critic," that is it learns the feature mapping from an action given an environment to the expected return of taking said action.

For our implementation, for both the Actor and the Critic, we use an MLP with 1 deep layer and $tanh$ activations. The Actor network outputs a mean action and also learns a scaling for an associated log-standard deviation–a common approach to learning in a continuous action space (also used in (Lu et al., 2023)). During training, the actions taken by the agent are taken from the distribution represented by this mean and deviation which allows for exploration during learning. We use the Adam optimizer to train these networks, which is a more efficient variant of Stochastic Gradient Descent; it uses an adaptive step size but is still a first order optimizer.

We learned in class that first order methods such as gradient descent work better in practice to train Neural Networks as they are much cheaper than second order methods such as Newton's Method. PPO's predecessor Trust Region Policy Optimization (TRPO) (Schulman et al., 2017) relied on second order information to compute updates which resulted in slower updates and more demand on resources. PPO being first order has made it more compatible with the deep learning backbone of Actor-Critic models resulting in cheaper, faster updates and better results with more widespread use.

The training data for these networks comes from interactions between the agent and the environment. Specifically, for multiple updates the agent rolls out a trajectory given a random start state and the actor's current action policy. The Critic estimates the value of the state-action pairs in these rollouts. The agent collects ground truth data during the rollouts, such as states, actions, rewards, and next states. This data is collected into batch matrices and used to update the actor's policy and critic's value estimation as follows: the Actor's updates are guided by the critic's value estimates or advantages (i.e., how much better an action was compared to the expected value). The Critic is trained to minimize the difference between its predicted and actual returns.

5.2. State-Action Representation

The state and action representations are integral to any control learning implementation. In this problem, there is a natural representation for the control outputs (similar to the optimal control problem) which is to use speed and heading as the outputted controls. The state or what the agent observes from its environment is constituted by its position relative to goal, current heading and current speed. The relative position allows the agent to learn a start-point invariant global policy to go from any point to any end point in its space. This state space is a little different than the optimal control representation, but the usage of a neural network (capable of approximating complex nonlinear systems) allows this state to capture all relevant information needed.

5.3. Objective to Optimize

There are three terms in the objective or total Loss function the networks are being optimized on (Algorithm 3 line 18). The first term is the scaled and clipped returns of the (updated) policy. The returns in turn are based on the rewards obtained in each episode, and are expressed in a way to maximize these rewards. The second term is the square of difference between the old returns and the new returns (line 17). This second term was of interest to us because it makes the function resemble the proximal prox_h operator. Recalling that

$$\text{prox}_h = \arg\min_u h(u) + 0.5 * ||u - x||^2$$

it is obvious that in this case $h(u)$ is a function of V_ϕ, u is V_ϕ at the updated policy and x is the V_ϕ at the previous policy. The inclusion of this second term is what keeps PPO training stable by penalizing jumping far away from its current policy. The third term encourages exploration by trying to maximizing entropy (variability) of the policy.

5.3.1. Reward Shaping

A key part of writing the objective to optimize for what we want in a scenario then is to strategically shape the heuristic by which the agent attains rewards. Our reward function involves multiple parts all motivating different qualitative behaviors in the agent. They are as follows:

- *Time Penalty*: a small time penalty (-0.05) added for each time step during an episode. This becomes a more significant term if the length of episode is long, thus penalizing length of needlessly long episodes.
- *Capture Reward*: if the agent reaches within a capture radius of the target position, it is awarded with this

relatively heavy terminal reward that is scaled by t
distance from start to end point. (Thus the longer poi
it can connect, the higher its reward.)

  - *Quick capture Bonus* We also reward a quick
    capture using a term scaled by the proportion
    episode length left unused by the agent.

- *Transitional Reward*: if the agent hasn't reached ca
  ture radius yet, we still give it incremental rewards
  avoid a sparse reward structure under long horizo
  which can result in training inefficiency.

  - *Distance to Waypoint*: This term is the differen
    in distance to waypoint between this time step a
    the past–in essence giving a negative reward if t
    agent is moving away from target and a positi
    reward for moving towards the target.
  - *Heading Alignment*: This term is the speed tim
    cos of the angle the agent would need to turn
    face the target. This term encourages the agent to face the target as it is 0 at that angle, and -1 for facing opposite the target. Note that we only allow the agent positive speeds.

- *Speed Reward*: Lastly, we always reward high speed by giving a reward scaled by the speed divided by the maximum speed.

A visualization of what training environment looked like is in Figure 5 in the appendix. The training of the model took about 10 hours, and as seen in the training curves in Appendix D, the value loss curves decreases and converges which is the indicator in this case of the model training. As we used an annealing learning rate, we can also see the learning rate decay over time.

6. Results and Discussion

With three control algorithms implemented, we wanted to compare their performances against each other across a few key metrics. Some of these control methods have been directly optimized for specific metrics, like minimum time optimal control. These performance metrics are summarized in Table 1:

Table 1. Comparisons of Metrics across Three Control Methods

Metric	MinT	NLQR	RL
Waypoints Reached	4/4	4/4	4/4
Total Race Time	81.52	148.41	79.50
Avg. Time per Wpt.	20.38	37.10	19.83
Total Control Effort	100.77	101.74	141.64
Speed Ctrl. Effort	80.52	81.53	70.82
Steer Ctrl. Effort	20.25	20.21	70.82

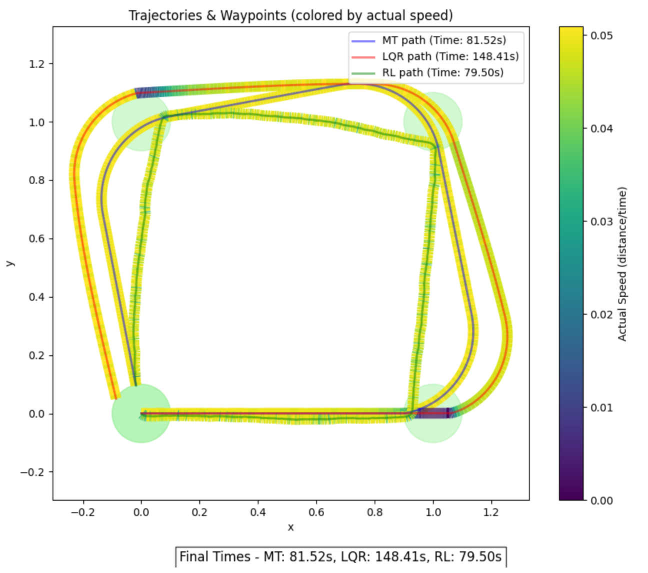

Figure 3. The trajectory for each control method: minimum time, NLQR, and RL.

6.1. Comparing Finish Time

In this race course example, we wanted to find the method that could accurately hit all the way points in the least amount of time. As in Figure 3, all control methods are able to hit the capture radius of each way point, though they accomplish the path in various amounts of time.

As expected, the minimum time optimal control implementation is faster than the non-linear quadratic regulator, as its cost function forces the min-time method to optimize for the smallest final time T, as in Eq. 4, while the NLQR method does not include the final time as something to minimize in its cost function. Furthermore, the reinforcement learning based controller beats out both optimal controls with a final time that is two seconds faster than the minimum time implementation.

These results can be better visualized in Appendix Figure 11 which shows the number of time step counts that the vehicle was at a certain speed for each control method.

We can see why the NLQR implementation was significantly slower than the minimum time and RL implementations. Since it was not optimized for decreasing the final time T, the vehicle spent quite a lot of time at very low speeds. This can also be seen in Figure 3, where the darker blue and purple shading represents slower speeds, which are only found in the NLQR trajectory. Interestingly, this "slow down" zone in the NLQR implementation is only found at two of the four way points, and it is unclear as to why this is happening. Perhaps an additional term to the cost function

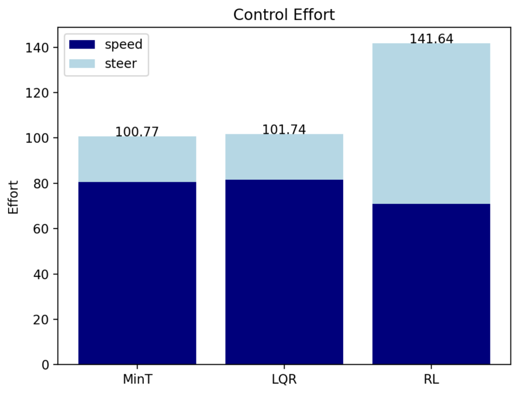

Figure 4. The total control effort used by each controller is divided into the speed effort and the steer effort.

whose goal is to minimize the final time would help reduce the vehicle's unnecessary time spent at a slower speed.

$$\begin{aligned}\min_{x,u,T} \quad & J_{\text{NLQR}} = T + \frac{1}{2}x(T)^\top H x(T) \\ & + \frac{1}{2}\int_0^T \left[x(t)^\top Q\, x(t)\right] dt \qquad (16) \\ & + \frac{1}{2}\int_0^T \left[u(t)^\top R\, u(t)\right] dt\end{aligned}$$

6.2. Comparing Control Effort

While speed is an important factor in car racing and self-driving cars, the control effort is the next more important metric to consider. The total control effort, which is the sum of the steering and speed efforts, is directly related to the energy used by the system. As in Table 1, the reinforcement learning approach used significantly more control effort than the optimal control efforts. Upon further analysis, and better viewed in Figure 4, we found that the steering effort for the RL method was equal to its speed effort, likely an artifact in the way it was set up. We qualitatively observed jittery motion, which was the most likely cause for high steer effort.

The optimal control methods presented much smoother trajectories that involved significantly less control effort as both had a small additional control cost penalty in their cost functions. This minimizer was included in the minimum time implementation after noticing similar jitter with that controller too. We found out later that jittery motion is a common observation in robot RL motion and it is common practice to add a smoothing term or to add noise to the data to prevent the model from overfitting to the objective landscape (in some sense).

Finally, it is interesting to note that even though the RL finished in the least amount of time, it used about ten units less of speed control effort, meaning that it was not the speed that helped it win the race, but instead the path that it took. As in Figure 3, it is evident that the trajectory that the RL car followed likely has a shorter length than the paths from the optimal controllers. This could be due to a constraint in both optimal controllers that limited the rate of turning ($\dot{\theta}$) as seen by the wide turns that both these trajectories include, but this constraint might not have been implemented in the RL method.

7. Conclusion

In this paper, we have presented a kinematic model of a non-holonomic vehicle and compared trajectories generated from three control methods. The vehicle model is based on the famous bike model (Taheri, 1990), that uses speed and steer to control the motion of the vehicle. We looked at three methods to control the vehicle: a minimum time optimal control method, a non-linear quadratic regulator (NLQR), and a reinforcement learning (RL) approach.

While comparing these the trajectories created from these three controllers, we found that all methods successfully hit all the way-points. As expected, the minimum time implementation had a lower final time than the NLQR implementation, but the RL method beat the minimum time approach by about two seconds. However, the RL method used about 40% more effort than both the optimal controllers.

It is likely that the RL method with smoothing terms or better training practices might outperform the optimal control methods. However, in the scope of this project, the minimum time is likely the ”winner”.

7.1. Future Work

While we focused on just reinforcement learning in this work, industry deployments use a mix of reinforcement learning and learning from expert data. One line of future work would be to use the traditional optimization methods presented here to generate ”expert” data. This would obviously incentivize the RL model to output similar paths as those found by the minimum time optimization or NLQR. It would be interesting to examine the difference in training time required for the car to learn the appropriate controls. Another important future work would be to do ablation studies on the different components of the rewards. We have many different kinds of rewards, resulting in complex incentives for the learned model. It would be interesting to omit components one at a time to understand the relevance and effect each term has upon the paths the car takes.

Software and Data

https://github.com/blakete/18.065-Final-Project/tree/main

References

Andersson, J. A. E., Gillis, J., Horn, G., Rawlings, J. B., and Diehl, M. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019. doi: 10.1007/s12532-018-0139-4.

Lu, Y., Fu, J., Tucker, G., Pan, X., Bronstein, E., Roelofs, R., Sapp, B., White, B., Faust, A., Whiteson, S., Anguelov, D., and Levine, S. Imitation is not enough: Robustifying imitation with reinforcement learning for challenging driving scenarios, 2023. URL `https://arxiv.org/abs/2212.11419`.

OpenAI. Spinning up in deep rl: Proximal policy optimization. `https://spinningup.openai.com/en/latest/algorithms/ppo.html`, 2018. Accessed: 2025-04-30.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. Trust region policy optimization, 2017. URL `https://arxiv.org/abs/1502.05477`.

Taheri, S. *An Investigation and Design of Slip-Control Braking Systems Integrated with Four-Wheel Steering*. PhD thesis, Clemson University, Clemson, SC, 1990. URL `https://search.worldcat.org/oclc/24119633`. Ph.D. thesis.

A. Optimal Control & RL Race Execution Pseudocode

Algorithm 1 Optimal Control Execution

```
1: Initialize $t \leftarrow 0$, $E \leftarrow 0$, $x \leftarrow x_0$
2: for $i = 1$ to $N$ do
3:     $[T_i, X_i(\cdot), U_i(\cdot)] \leftarrow \mathsf{SolveOptimalControl}(f, x, w_i, r_c)$
4:     Execute $U_i$ on $[0, T_i]$, set $x \leftarrow X_i(T_i)$
5:     $E \leftarrow E + \int_0^{T_i} \|U_i(t)\|_2^2 \, dt$
6:     $t \leftarrow t + T_i$
7: end for
8: return trajectory, $T_N = t$, $E$
```

Algorithm 2 Reinforcement Learning Execution

```
1: Initialize $t \leftarrow 0$, $x \leftarrow x_0$, $i \leftarrow 1$
2: while $i \leq N$ and $t < T_{\max}$ do
3:     $u \leftarrow \pi\big(x_{1:2} - w_i,\ \theta\big)$
4:     Execute $u$ for $\Delta t$, update $x \leftarrow x + \int_0^{\Delta t} f(x, u) \, dt$
5:     $t \leftarrow t + \Delta t$
6:     if $\|x_{1:2} - w_i\|_2 \leq r_c$ then
7:         $i \leftarrow i + 1$
8:     end if
9: end while
10: return trajectory, $T_N = t$
```

B. PPO Pseudocode

Algorithm 3 Proximal Policy Optimization (PPO)

```
 1: Initialize Policy $\pi_\theta$, value function $V_\phi$
 2: for num updates do
 3:     for $t_i \in [1, T]$ do
 4:         $a_{t_i} = \pi_\theta(s_{t_{i-1}}, a_{t_{i-1}})$
 5:         Observe $r_{t_i}$ and $s_{t_i}$
 6:         Store $s_{t_{i-1}}$, $a_{t_i}$, $r_{t_i}$, and $V_\phi(s_{t_i})$
 7:     end for
 8:     Trajectory $\tau = \{(s_t, a_t, r_t)\}_{t=1}^{T}$
 9:     for update epochs do
10:         $\pi_{\theta_{\text{old}}} = \pi_\theta$
11:         $\mathcal{B}_i = \{(s_t, a_t, r_t)\}_{t=i}^{i+M-1}, \quad \text{for} \quad i \in \{1, M, 2M, \dots\}$
12:         These next few functions are calculated for the entire minibatch $\mathcal{B}_i$
13:         Returns $R_t(\mathcal{B}_i) = \sum_{l=0}^{\infty} \gamma^l r_{t+l}$
14:         Advantages
                $$\hat{A}_t(\mathcal{B}_i) = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}$$
            where $\delta_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$
15:         $k_\theta^t = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$
16:         $L_{\text{policy}} = \max\left(k_\theta^t \hat{A}_t,\ \text{clip}(k_\theta^t, 1-\epsilon, 1+\epsilon)\hat{A}_t\right)$
17:         $L_{\text{ValueFunc}} = \frac{1}{2}\left(V_\phi(s_t) - R_t\right)^2$
18:         Total loss: $L = -L_{\text{policy}} + 0.5 * L_{\text{ValueFunc}} - c_e * \text{Entropy}$
19:         Update $\pi_\theta$, $V_\phi$ using Adam to minimize $L$
20:     end for
21: end for
```

C. RL Environment

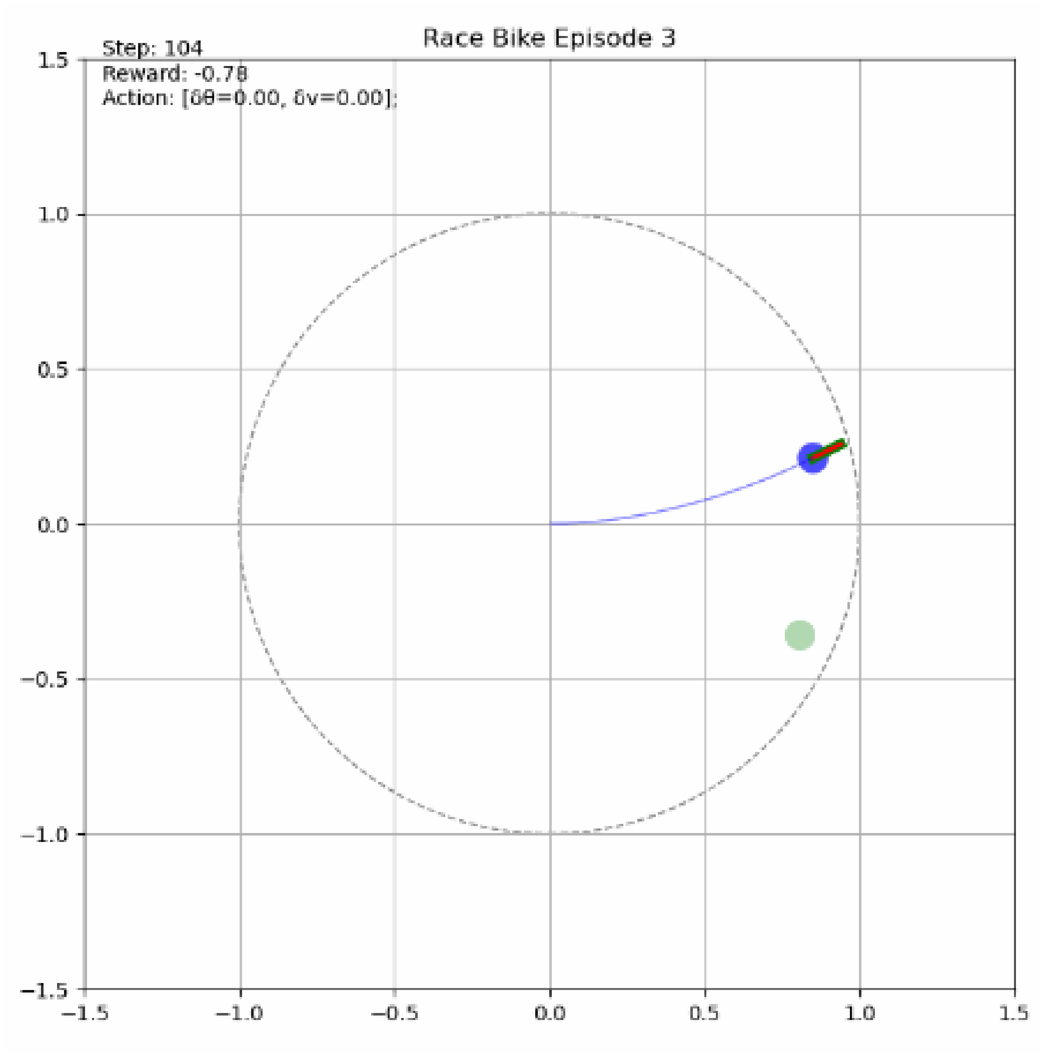

Figure 5. An example of the training environment. The blue dot is a car following a random policy and the light green dot is the goal. Note that it is accruing a negative reward for moving away from the goal. We randomly generate the green goal anywhere within the dashed circle radius for training.

D. RL Training Curves

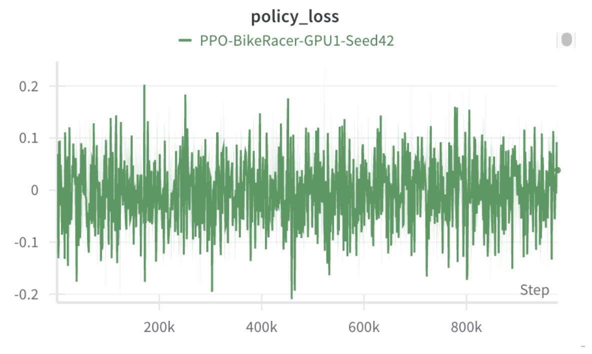

Figure 6. Policy Loss over time stayed very noisy, and centered around 0.

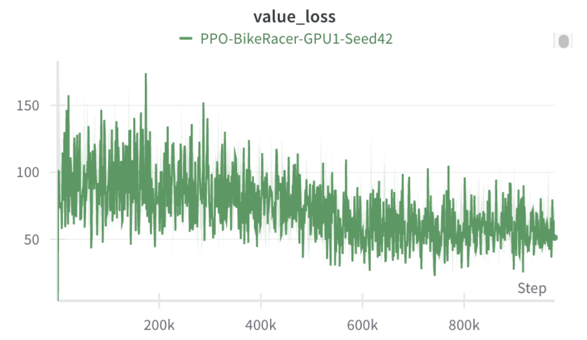

Figure 7. The Critic Value loss generally decreases over time

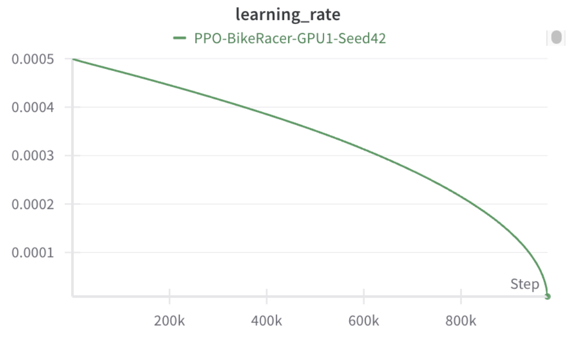

Figure 8. We used an annealing learning rate i.e. the learning decayed over time as shown here

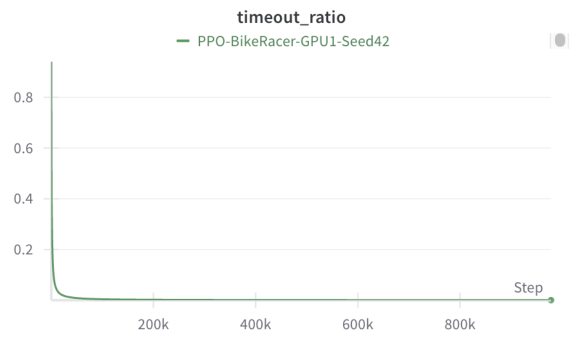

Figure 9. Timeout Ratio drops within the first 100k iterations, showing that the model learns very quick to start capturing waypoints and not just waiting for episode to run to max length.

Figure 10. Key graphs from training the PPO model. Note that we got best performing models were observed to be from the 50 to 100 epoch range. Models from later epochs became overfit to the reward objective.

E. Speed Histograms

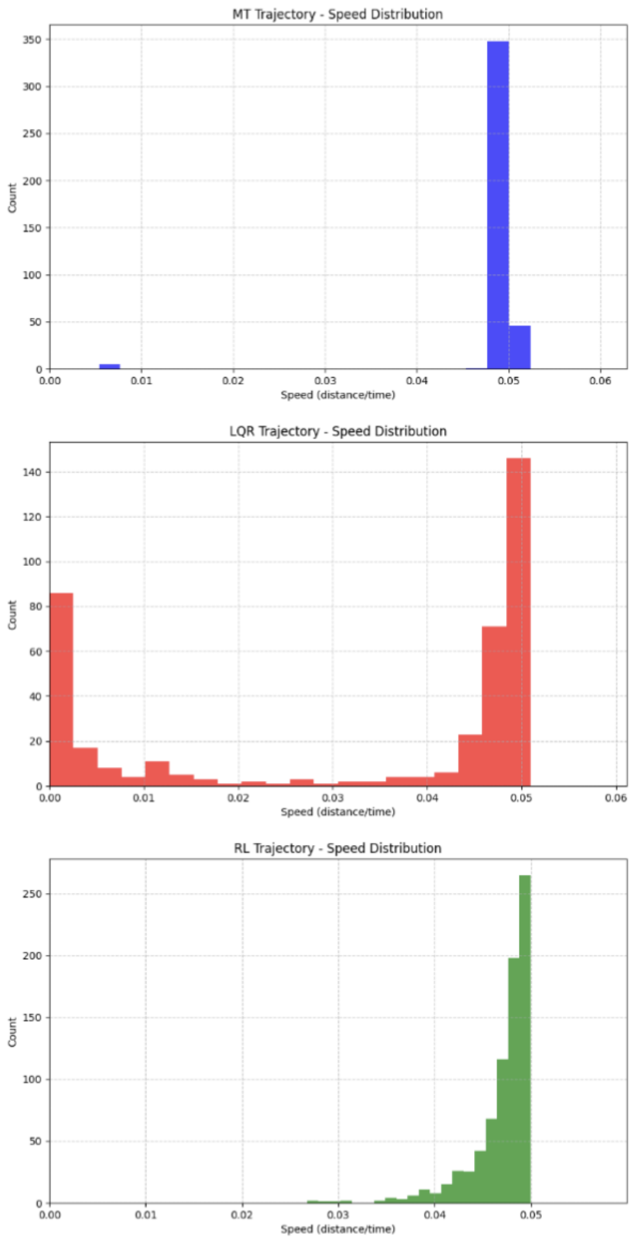

Figure 11. Histograms of the Speed of the vehicle for each method