Optimal Kinodynamic Motion Planning in Known Environments Using LQR-RRT*

Alvin James Bacani
Electrical and Systems Engineering
Washington University in St. Louis
St. Louis, Missouri
a.bacani@wustl.edu

Gabe Herman
Electrical and Systems Engineering
Washington University in St. Louis
St. Louis, Missouri
g.m.herman@wustl.edu

***Abstract*—In this project, we implement and evaluate a kinodynamic motion planner for a car-like robot navigating in 2D environments with known circular obstacles. Our approach is based on LQR-RRT*, a sampling-based planning algorithm that incorporates linear-quadratic regulation into the RRT* framework to generate dynamically feasible and cost-efficient trajectories. The robot's nonlinear dynamics include position, orientation, velocity, and curvature, and our planner locally linearizes these dynamics to compute control inputs that steer toward sampled states. We apply the method in four environments of increasing complexity, generating feasible trajectories and comparing the resulting control costs. Results show that LQR-RRT* consistently produces smooth, collision-free paths that respect the system's dynamics and improve over time toward lower-cost solutions. We also comment on the algorithm's completeness and discuss its practical performance in the context of kinodynamic planning.**

I. Introduction

Planning dynamically feasible trajectories for systems with complex motion constraints is a fundamental problem in robotics. Unlike geometric planning, kinodynamic planning must account not only for obstacle avoidance, but also for the system's dynamics, including velocity, acceleration, and other physical constraints. In this project, we focus on planning trajectories for a car-like robot with nonlinear dynamics, operating in a bounded 2D environment populated with known circular obstacles.

The task involves two main objectives. First, we aim to generate collision-free trajectories that are dynamically feasible given the robot's motion model. Second, we seek to improve those trajectories by minimizing the total control effort required to follow them. To address this, we implemented LQR-RRT*, an extension of the RRT* algorithm that incorporates local linearization and optimal control via linear-quadratic regulation. This method allows us to automatically derive meaningful distance metrics and extension heuristics that better reflect the system's dynamics, improving both the feasibility and efficiency of the planned paths.

We tested our planner across multiple environments with varying obstacle configurations and evaluated its performance based on both trajectory quality and control cost. This report describes the system model, our implementation of LQR-RRT*, experimental results, and a discussion on the planner's effectiveness and limitations.

II. Problem Formulation

We consider the problem of planning dynamically feasible, collision-free trajectories for a car-like robot navigating in a 2D environment with known circular obstacles. The robot's state is defined as

$$x = \begin{bmatrix} x \\ y \\ \theta \\ v \\ \kappa \end{bmatrix}$$

where x and y are the position coordinates in meters, θ is the orientation in radians, v is the linear speed in m/s, and κ is the path curvature in m^{-1}. The control input is defined as

$$u = \begin{bmatrix} u_v \\ u_\kappa \end{bmatrix}$$

where u_v and u_κ represent the rate of change of speed and curvature, respectively.

The nonlinear dynamics governing the robot's motion are:

$$\begin{aligned} \dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \dot{\theta} &= v\kappa \\ \dot{v} &= u_v \\ \dot{\kappa} &= u_\kappa \end{aligned}$$

The robot starts at the initial state:

$$x_0 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

and must reach a goal region defined by $x = 9$, $y = 9$, with no specific requirements on the remaining state variables.

The allowable state space is constrained as follows:

$$\begin{aligned} x, y &\in [0, 10] \\ \theta &\in [-\pi, \pi] \\ v &\in [0, 2] \\ \kappa &\in [-0.5, 0.5] \end{aligned}$$

The objective has two parts:

1) **Feasibility:** Find a trajectory $\rho : [0,1] \to X$ and control signal $\pi : [0,1] \to U$ such that the robot avoids all obstacles and obeys the system dynamics.
2) **Optimality:** Among all feasible trajectories, find one that minimizes the total control effort:

$$c(T) = \int_0^T u_t^T R u_t \, dt, \quad \text{where } R = \begin{bmatrix} 0.1 & 0 \\ 0 & 1 \end{bmatrix}$$

The environment contains static, known, circular obstacles, and we consider four predefined scenarios of increasing complexity. The planner must ensure the robot reaches the goal region without collisions while satisfying the dynamic constraints.

III. Planning Algorithm

To address the kinodynamic planning problem, we implemented the LQR-RRT* algorithm, a variant of RRT* that incorporates local linearization and optimal control through Linear Quadratic Regulation (LQR). This method improves upon standard RRT-based planners by using control-aware distance metrics and steering functions, enabling better exploration and more efficient convergence in systems with complex dynamics.

A. Overview of LQR-RRT*

LQR-RRT* follows the general structure of RRT*: it incrementally builds a tree of feasible trajectories through random sampling and local connection of states. However, instead of using simple geometric distances and straight-line connections, LQR-RRT* linearizes the nonlinear dynamics around each node and computes both the cost-to-go and control policy using LQR. This results in more informed exploration and more reliable trajectory extensions.

At each iteration, the algorithm performs the following steps:

1) **Sampling:** A random state x_{rand} is sampled uniformly from the state space.
2) **Nearest Neighbor:** The nearest node in the existing tree is found using an LQR-based cost-to-go metric:

$$x_{\text{nearest}} = \arg\min_{v \in V} (v - x_{\text{rand}})^T S (v - x_{\text{rand}})$$

   where S is the solution to the continuous-time algebraic Riccati equation (CARE) computed during linearization.
3) **Steering:** A new state x_{new} is generated by simulating the system forward from x_{nearest} under the locally optimal LQR policy for a short horizon.
4) **Neighborhood Search:** A set of nearby nodes is selected using the LQR cost metric.
5) **Parent Selection:** Among nearby nodes, the one that connects to x_{new} with the lowest total cost is selected as its parent.
6) **Attempt to connect to Goal:** After extending the graph, attempt to connect x_{new} directly to the goal state with an LQR trajectory.
7) **Collision Check:** The resulting trajectory is checked for collisions with known obstacles. If collision-free, x_{new} is added to the tree.
8) **Rewiring:** The algorithm attempts to reconnect neighboring nodes through x_{new} if doing so improves their cost-to-come.

B. LQR Linearization and Control

To perform local planning, we linearize the system dynamics around a state x_0 with zero control input:

$$\dot{\bar{x}} = A(x_0)\bar{x} + B(x_0)\bar{u}$$

where $\bar{x} = x - x_0$, $\bar{u} = u - 0$, and the matrices A and B are Jacobians of the dynamics:

$$A(x_0) = \left.\frac{\partial f}{\partial x}\right|_{x_0}, \quad B(x_0) = \left.\frac{\partial f}{\partial u}\right|_{x_0}$$

The continuous-time algebraic Riccati equation is then solved:

$$A^T S + SA - SBR^{-1}B^T S + Q = 0$$

to compute the optimal feedback gain:

$$K = R^{-1}B^T S$$

The Q matrix above is the penalty on state error, and was set to $\rho \cdot diag\,(1,1,0,0,0)$ with ρ a tunable parameter, indicating that we only penalize the position (x, y) in the control design. This policy $u = -K\bar{x}$ is applied to steer the system during tree expansion.

IV. Implementation

Our implementation of the LQR-RRT planner was done in Python using NumPy and SciPy for matrix operations and integration. The overall planner was structured to follow the standard RRT loop, with custom modules for system dynamics, LQR linearization, collision checking, and tree management.

A. Dynamics and Linearization

The car-like robot's nonlinear dynamics were implemented based on the equations provided in the project description. At each iteration, the linearized A and B matrices were computed and the Hautus test performed to check that the pair was controllable. These matrices were passed to the standard `scipy.linalg.solve_continuous_are` function to solve the Riccati equation and compute the LQR gain and cost matrices.

B. Tree Expansion and Steering

Tree nodes were stored as Python objects containing the state vector, parent pointer, cost-to-come. When a sample was drawn, we computed the LQR-based cost-to-come from all existing nodes and selected the node which minized the LQR cost-to-come. A new state was then generated by applying the local LQR policy for a short time horizon using Euler integration.

C. Collision Checking

Obstacles were modeled as circles and defined using center coordinates and radii. Collision checking tested whether any intermediate states along a trajectory came within an obstacle's radius. We assumed a point-mass robot for simplicity, which made collision detection efficient and scalable.

D. Cost Computation and Rewiring

Each new trajectory was evaluated using the cost function.

$$c(T) = \int_0^T u_t^T R u_t \, dt$$

This was approximated using trapezoidal integration over the sampled control sequence. During rewiring, we evaluated whether connecting through the new node would reduce the total cost for nearby nodes and updated the tree accordingly.

E. Goal Region and Termination

The planner terminated when a state entered the goal region (defined as a ball centered at $[9, 9]$ with a small radius). Throughout the process, we tracked the best solution found and its associated cost for reporting and analysis.

V. Results

We tested our LQR-RRT planner in four $10\,\text{m} \times 10\,\text{m}$ environments with static circular obstacles. Each scenario presented increasing levels of complexity, with tighter passages and more constrained goal regions. For each environment, we generated a dynamically feasible, collision-free trajectory and then refined it to reduce control effort using the LQR-based steering and cost-aware rewiring.

A. Feasible Trajectories (Problem 1)

In the first phase, the planner focused on feasibility—finding any path that connects the start state to the goal region while satisfying the system's nonlinear dynamics. Across all four environments, LQR-RRT successfully produced feasible trajectories. We observed that:

- In open environments, the planner converged quickly with relatively short trees.
- In cluttered or narrow-passage scenarios, the algorithm required more iterations to find valid connections due to the steering constraints and dynamics.
- The LQR-based steering function helped avoid local dead-ends by guiding extensions in dynamically consistent directions.

B. Cost-Optimized Trajectories (Problem 2)

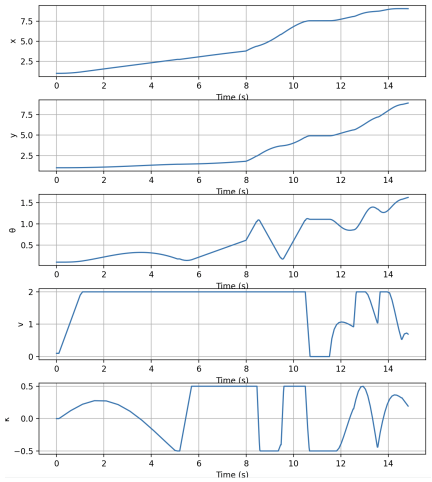
After obtaining a feasible path, we continued running the planner to improve the solution by minimizing the total control effort:

$$c(T) = \int_0^T u_t^T R u_t \, dt$$

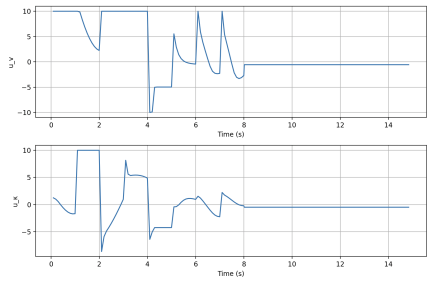
where $R = \begin{bmatrix} 0.1 & 0 \\ 0 & 1 \end{bmatrix}$. The cost function penalized curvature changes more heavily than acceleration.

For each scenario, we recorded the number of iterations to reach the goal and the control cost of the optimal path.

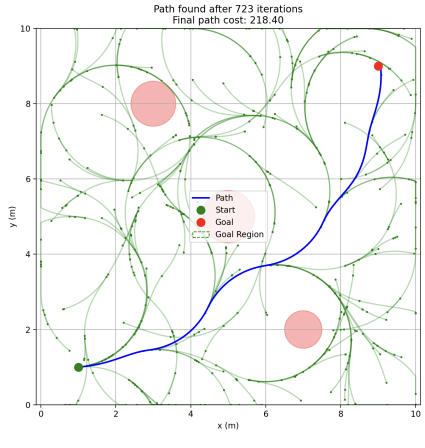
- In open spaces, paths smoothed out to reduce unnecessary steering, resulting in lower curvature rates and more direct motion.
- In tight environments, the optimized paths traded off longer durations for lower control effort, often using smoother, multi-turn arcs rather than sharp maneuvers.



(a) State variables x, y, θ, v, and κ over time for Environment 1

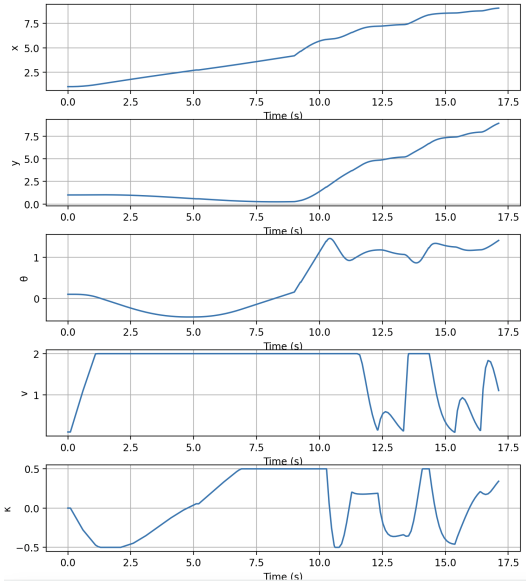


(b) Control inputs u_v and u_κ over time for Environment 1

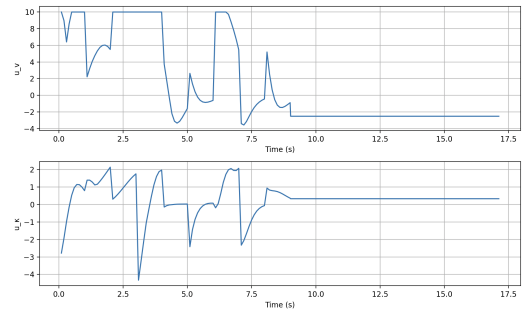


(c) Path found in Environment 1 using RRT, with final cost 218.40

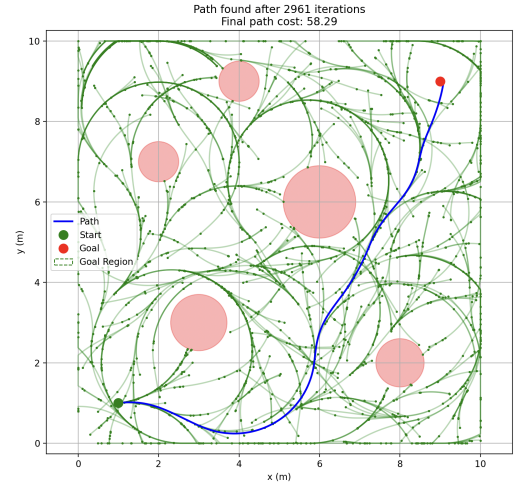
Fig. 1: Environment 1 results.



(a) State variables x, y, θ, v, and κ over time for Environment 2

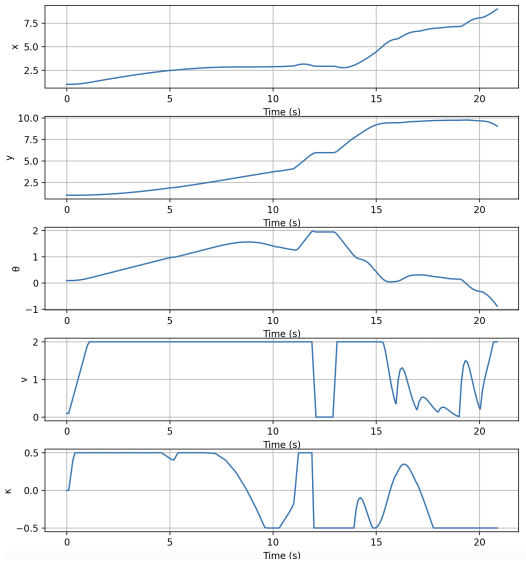


(b) Control inputs u_v and u_κ over time for Environment 2

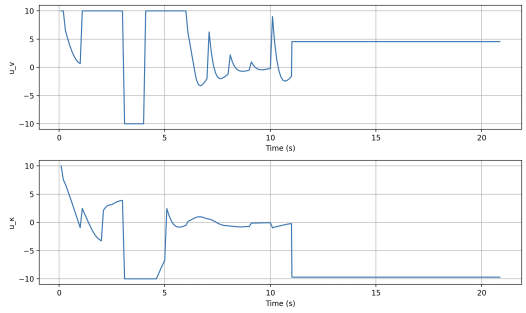


(c) Path found in Environment 2 using RRT, with final cost 58.29

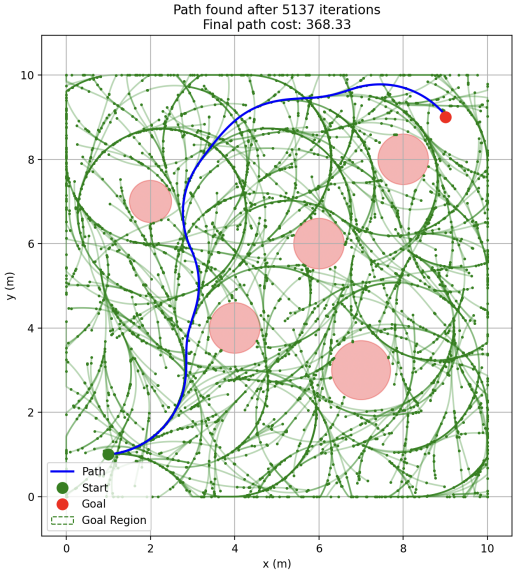
Fig. 2: Environment 2 results.



(a) State variables x, y, θ, v, and κ over time for Environment 3

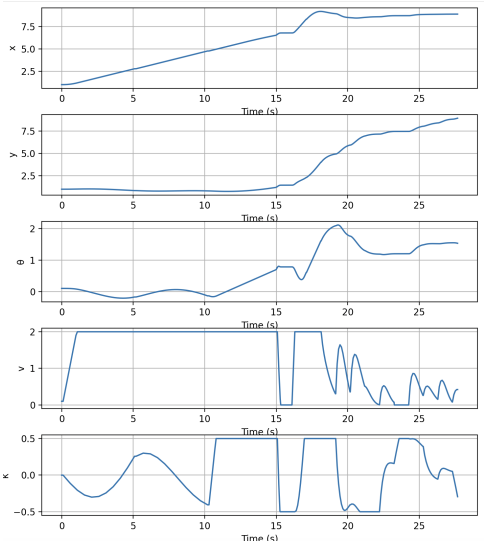


(b) Control inputs u_v and u_κ over time for Environment 3

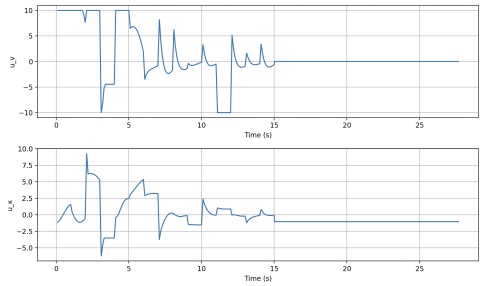


(c) Path found in Environment 3 using RRT, with final cost 368.33

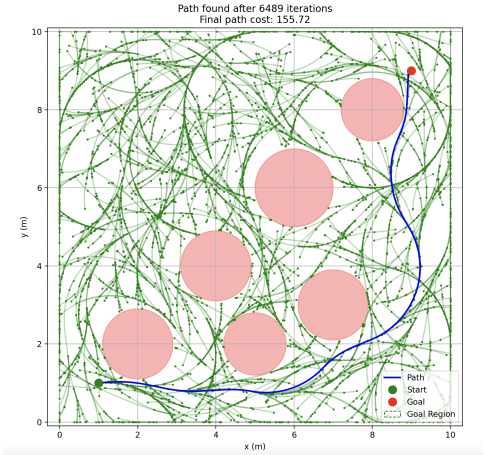
Fig. 3: Environment 3 results.



(a) State variables x, y, θ, v, and κ over time for Environment 4



(b) Control inputs u_v and u_κ over time for Environment 4



(c) Path found in Environment 4 using RRT, with final cost 155.72

Fig. 4: Environment 4 results.

Table I summarizes the final control costs for each scenario.

Environment	Final Cost
1	218.40
2	58.29
3	368.33
4	155.72

TABLE I: Control cost of best path found after optimization in each environment.

VI. Analysis

The performance of LQR-RRT in our experiments reflects both the strengths and trade-offs of sampling-based kinodynamic planning. Below, we evaluate the algorithm in terms of completeness, optimality, efficiency, and sensitivity to problem parameters.

A. Completeness

LQR-RRT inherits the probabilistic completeness property of the original RRT algorithm. That is, given enough time and samples, it will find a feasible trajectory if one exists. In all four environments, the planner consistently found feasible solutions, confirming this property. However, in more constrained environments, the algorithm required significantly more iterations due to the reduced volume of reachable space from each node under the system's dynamics. The number of iterations to completion also varied significantly from run to run. Due to long runtimes, it was not feasible to collect enough data to characterize the variance of the number of iterations; however, over only a few runs, the number of iterations could vary by a factor of 10.

B. Optimality

A key advantage of RRT and its variants is asymptotic optimality—the guarantee that, over time, the optimal (minimum cost) path is achieved. By incorporating LQR-based heuristics, LQR-RRT maintains the optimality of RRT while allowing for systems with differential constraints, and furthermore can achieve locally optimal trajectories with respect to a given cost function. This is ultimately the drawback, or at least limitation, of the LQR-based method. The algorithm guarantees that the control cost is minimized among the possible paths in the tree; however, the optimality of the actual trajectories generated is entirely dependent on the integration time-step, the frequency of linearization, and the chosen LQR penalty matrices. In our implementation, the LQR gains stray from the optimal for most of the path planning. To change this would require much more frequent linearization, or simply using another method to handle nonlinear dynamics.

C. Efficiency and Scalability

Although LQR-RRT* performed well in our 5-dimensional state space, the overall planning time was sensitive to step size, tree size, and the complexity of the environment. Solving Riccati equations and integrating dynamics at each node added overhead compared to simpler planners. In practice, this makes tuning important—step size, horizon length, and cost weights all affect the balance between fast exploration and solution quality.

D. Summary

LQR-RRT* offers a principled and effective solution to kinodynamic motion planning for nonlinear systems. Its ability to produce dynamically consistent, smooth, and cost-efficient paths—without the need for hand-designed heuristics—makes it a strong candidate for planning tasks involving car-like robots and similar systems. Future improvements could include adaptive sampling, or belief-space extensions.

$$Q = \rho \cdot \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{1}$$

REFERENCES

[1] A. Perez, R. Platt Jr., G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez, "LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, St. Paul, MN, USA, May 2012, pp. 2537–2542.