Sampling-based Planning Algorithms Applied to Bicycles

Steve Levine

Abstract— **In this paper, I explore the application of various random sampling based algorithms to dynamic models of bicycles. Starting with relatively simple 3-dimensional state space models of bicycle and working up to a more realistic 7-dimensional model that incorporates skidding, I examine RRT, RG-RRT, RRT*, and LQR-RRT*.**

I. Introduction

Having a modest obsession for anything bicycle-related, I thought it would be quite exciting to try and apply various planning techniques to my favorite two-wheeled vehicles. Specifically, my aim was to use algorithms to generate adventurous and aggressive trajectories for interesting dynamical models for bicycles. Having some previous knowledge of RRT and RRT* for holonomic systems, I thought it would be interesting to apply them and some of their variants to problems with rich dynamical constraints.

II. Bicycle Models

This project examines planning for various bicycle models restricted to the 2D plane. We begin with the Dubin's vehicle, a 3-dimensional state space vehicle that moves at a constant velocity and can be steered. We then progress to a more realistic, non-slipping bicycle that takes into account the bicycle's wheel base and differential constraints associated with steering. Next, we move to a seven-dimensional bicycle model which is the most realistic and incorporates skidding.

A. Dubin's Vehicle

The Dubin's vehicle is a popular dynamical model and can be thought of as a simple bicycle. The dubins vehicle moves around in a two dimensional plane at constant velocity v and is parameterized by its position and angle. The dynamics are

$$\begin{aligned} \dot{x} &= vcos(\theta) \\ \dot{y} &= vsin(\theta) \\ \dot{\theta} &= u \end{aligned} \tag{1}$$

where u is the control input to the system. It is possible to restrict the turning radius of the vehicle to ρ by imposing the actuation limit $|u| \leq \frac{v}{\rho}$.

The control-limited Dubin's vehicle is perhaps the most simplistic nonholonomic system (no motion is instaneously possible in the direction of $\theta + \pi/2$), also making it perhaps the simplest model of a bicycle.

B. Non-slipping Bicycle

A more realistic model of a bicycle is presented in this section. This model follows a similar spirit to the Dubin's vehicle in that differential constraints prevent motion in certain directions. Specifically, the non-slipping bicycle models wheel in which lateral motion (parallel to the axis) is not permitted.

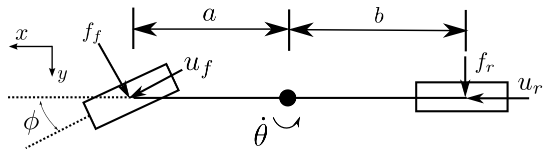

Fig. 1. Parameters of the bicycle model.

The Non-slipping bicycle model can be thought of two wheels separated by a fixed wheel base w, in which a force u_f is applied from the rear wheel and the steering angle of the front wheel with respect to the bicycle frame ϕ is directly controllable. Taking into account the wheelbase and differential constraints of both wheels simultaneously, a turning radius can be computed for any ϕ. The dynamics of the non-slipping bicycle can be written as (derivation omitted):

$$\begin{aligned} \dot{x} &= vcos(\theta) \\ \dot{y} &= vsin(\theta) \\ \dot{\theta} &= \frac{v}{w}\tan\phi \\ \dot{v} &= \frac{u_r}{m} \end{aligned} \tag{2}$$

This bicycle model is very useful, and is quite applicable in regimes where the wheels have excellent traction and do not skid.

C. Slipping Bicycle

In this section, I will present a model for a slipping bicycle, which is a slighly modified version of system known as the Bicycle Model. This model has a seven-dimensional state space and is the most complicated model examined in this project. Unlike the previously-described non-slipping bicycle that imposed a differential constraint that the tires could not move laterally, the slipping bicycle is capable of moving in all directions. There is however a (usually large) lateral force on the bicycle wheel that is modeled as proportional to the sine of slip angle of the wheel. This is what keeps the wheels in line and encourages (but does not require) the bicycle to to travel in the direction the wheel is facing instead of laterally.

The slip angle α of any wheel is defined as the angle between its forward and lateral velocities (see Figure 2). A free-body diagram for the Bicycle Model is presented in Figure 1 showing parameters for the model. Please also note

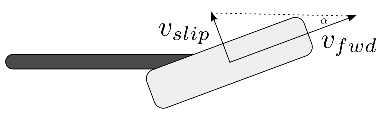

Fig. 2. The slip angle α of a wheel is the angle between it's slip and forward velocities. Note that when $v_{slip} = 0$, or namely the bicycle is not slipping at all, $\alpha = 0$.

that the bicycle has slipping angles α_{front} and α_{rear} for the front and rear wheels. We model the lateral forces as being proportional to the sine of the slip angles, or namely

$$\begin{aligned} f_f &= C \sin \alpha_{front} \\ f_r &= C \sin \alpha_{rear} \end{aligned} \tag{3}$$

where C is a large and negative constant. Note that in the standard Bicycle model, these lateral forces are set to be proportional to the slip angles themselves, not to the sine of these angles. While this approximation works well for small slip angles since the taylor expansion of $\sin \alpha \approx \alpha$, I found empirically that this model tends to break down and cause unrealistic simulation results when adventurous trajectories that feature sharp steering are computed. By making the lateral forces proportional the sines of the slip angles, there will be no lateral forces should the bike wheel be rotated 180 degrees.

Using the parameters noted above and this slipping model, we can compute the dynamics of the system:

$$\begin{aligned} m\ddot{x} &= u_r \cos\theta + u_f \cos(\theta + \phi) - f_r \sin\theta - f_f \sin(\theta + \phi) \\ m\ddot{y} &= u_r \cos\theta + u_f \cos(\theta + \phi) + f_r \cos\theta + f_f \cos(\theta + \phi) \\ I\ddot{\theta} &= -bf_r + af_f \cos\phi + au_f \sin\phi \\ \dot{\phi} &= u_\phi \end{aligned} \tag{4}$$

We can compute the slip angles in order to compute f_f and f_r as follows:

$$\alpha = \tan^{-1}\left(\frac{v_{slip}}{v_{fwd}}\right) \tag{5}$$

We will now derive the slip angle α_{front} for the front wheel. We can derive the kinematic position x_f and y_f with respect to the ground frame for the front wheel as

$$\begin{aligned} x_f &= x + a\cos\theta \\ y_f &= y + a\sin\theta \end{aligned}$$

and the velocities as

$$\begin{aligned} \dot{x}_f &= \dot{x} - a\dot{\theta}\sin\theta \\ \dot{y}_f &= \dot{y} + a\dot{\theta}\cos\theta \end{aligned}$$

Let $\hat{b}_{slip}$ be a unit vector in the direction of v_{slip} and $\hat{b}_{fwd}$ be a unit vector in the direction of v_{fwd}. Since the forward direction for the wheel is in the direction of $\theta + \phi$, we can write

$$\hat{b}_{fwd} = \begin{bmatrix} \cos(\theta+\phi) \\ \sin(\theta+\phi) \end{bmatrix}, \quad \hat{b}_{slip} = \begin{bmatrix} -\sin(\theta+\phi) \\ \cos(\theta+\phi) \end{bmatrix} \tag{6}$$

This will allow us to compute v_{fwd} using some trigonometric identities as

$$\begin{aligned} v_{fwd} &= \begin{bmatrix} \dot{x}_f \\ \dot{y}_f \end{bmatrix} \cdot \hat{b}_{fwd} \\ &= (\cos\theta\cos\phi - \sin\theta\sin\phi)(\dot{x} - a\dot{\theta}\sin\theta) \\ &\quad + (\sin\theta\cos\phi + \sin\phi\cos\theta)(\dot{y} + a\dot{\theta}cos\theta) \\ &= a\dot{\theta}\sin\phi + \dot{x}\cos(\theta+\phi) + \dot{y}\sin(\theta+\phi) \end{aligned}$$

and v_{slip} similarly as

$$\begin{aligned} v_{slip} &= \begin{bmatrix} \dot{x}_f \\ \dot{y}_f \end{bmatrix} \cdot \hat{b}_{slip} \\ &= a\dot{\theta} - \dot{x}\sin(\theta+\phi) + \dot{y}\cos(\theta+\phi) \end{aligned}$$

Now with v_{fwd} and v_{slip} derived, we are poised to solve for α_{front} using Equation 5:

$$\alpha_{front} = \tan^{-1}\left(\frac{-\dot{x}\sin(\theta+\phi) + \dot{y}\cos(\theta+\phi) + a\dot{\theta}}{\dot{x}\cos(\theta+\phi) + \dot{y}\sin(\theta+\phi) + a\dot{\theta}\sin\phi}\right) \tag{7}$$

Using similar techniques we can solve for the slightly simpler α_{rear} as

$$\alpha_{rear} = \tan^{-1}\left(\frac{-\dot{x}\sin\theta + \dot{y}\cos\theta - b\dot{\theta}}{\dot{x}\cos\theta + \dot{y}\sin\theta}\right) \tag{8}$$

This model as presented has a seven-dimensional state space (state variables are $x, y, \theta, \dot{x}, \dot{y}, \dot{\theta}$, and ϕ). Note that in the common Bicycle Model, the state space is 6-dimensional because ϕ is modeled as directly controllable. Here, for continuity, we model that we $\dot{\phi}$ is controllable via u_ϕ.

III. Planning Algorithms

In this section, I will present the various planning algorithms that I use to plan bicycle trajectories. My focus is specifically on randomized sampling-based planning algorithms - namely RRT, RG-RRT, RRT* and LQR-RRT*.

A. *RRT*

The RRT, or Rapidly-exploring Random Tree, is a well-known randomized sampling-based planning algorithm. Its fundamental approach is to grow a tree through a (potentially high-dimensional) state space that connects a start initial configuration x_{init} to a goal region $\mathcal{G}$ while avoiding obstacle regions. Pseudocode for the RRT algorithm is shown in Algorithm 1. A random sample x_{rand} is generated in state space from some probability distribution (usually uniform). The closest vertex $x_{nearest}$ in the tree to this sample is computed using some distance metric (often Euclidean). A new sample point x_{new} is then generated that extends the tree in the direction of the random sample. If the resulting trajectory connecting $x_{nearest}$ and x_{new} is collision-free, x_{new} is added to the tree. This relatively-simple algorithm has performed quite well in practice, in part due to it's so-called Voronoi bias that encourages sampling of yet-unexplored regions of state space.

Algorithm 1: RRT

```
begin
    $V = \{x_{init}\}, E = \{\}$
    while $iterations \leq N$ do
        $x_{rand} \longleftarrow$ SAMPLE()
        $x_{nearest} \longleftarrow$ NEAREST($V, x_{rand}$)
        $x_{new} \longleftarrow$ STEER($x_{nearest}, x_{rand}$)
        if OBSTACLEFREE($x_{nearest}, x_{new}$) then
            $V = V \cup x_{new}$
            $E = E \cup (x_{nearest}, x_{new})$
        end
    end
end
```

*B. RRT**

The RRT algorithm has no notion of cost or optimality. In fact, RRT often generates strange and unexpected paths, sometimes with loops or other suboptimal characteristics. RRT's are thus in practice often run repeatedly online with branch and bound techniques to minimize cost in an attempt to improve paths. While this does indeed work and has been proven on many real systems, such as MIT's entry into the DGC, it can be proven that repeated iterations of RRT will converge to a suboptimal path with probability 1 as the number of vertices approaches ∞ [4].

The RRT* algorithm, which was only developed within the past few years, seeks to ameliorate this drawback with RRT's by providing probabilistically optimal random sampling-based motion planning with the same asymptotic performance as RRT [4]. RRT* is in spirit very similar to RRT, but includes several important modifications. Namely, RRT* selects minimal parents for new vertices and additionally actively rewires progressively smaller regions of the tree using a Dijkstra-like heuristic. It can be proven that with these extensions, RRT* will converge to the optimal solution as defined by some cost function with probability 1 as the number of nodes in the tree approaches ∞. Put another way, RRT* is probabilistically optimal. It additionally works quite well in practice for a number of systems, especially those that are holonomic. In Figure 3, my RRT* implementation is run on a holonomic system with an obstacle.

C. RRT and RRT for Dynamical Systems*

RRT and RRT* have worked well in practice for a number of low and high-dimensional systems. RRT's can also be extended to work with systems that have differential constraints, such as non-holonomic vehicles or systems with dynamic constraints. In these situations, the STEER() function is modified to obey the differential constraints. The performance of RRT and RRT* for these types of systems is heavily-dependent on both the distance metric used in the NEAREST() function and the extension technique

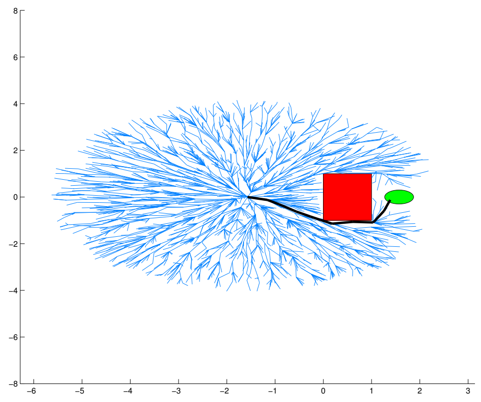

Fig. 3. RRT* running on a holonomic planning problem. Note that the vertices visually appear to bend in the optimal direction towards the goal. The oval shape is due to branch and bound.

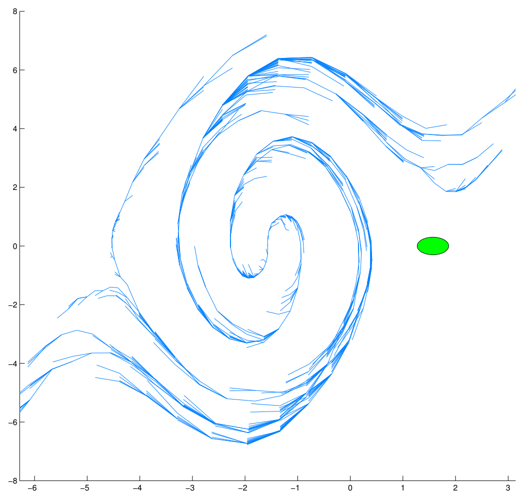

Fig. 4. A standard RRT running on the inverted pendulum problem, before reaching the goal. Note the large number of redundant vertices that are very close to each other.

used in STEER(). While Euclidean distance works well for holonomic systems, it typically performs poorly for systems with dynamics and results in repeated expansions of the same node (see Figure 4). This is because two points in state space that are close together as measured by a Euclidean metric may actually be quite far apart in time, or as measured by some other cost function. This results in RRT trees that are excessively dense in some areas, needlessly slowing down computation.

A number of approaches for improving RRT's performance in planning for dynamical systems have been proposed. Among them are techniques for forcing the RRT to explore regions of state space with extension restrictions, as well as techniques for improving the distance and extension

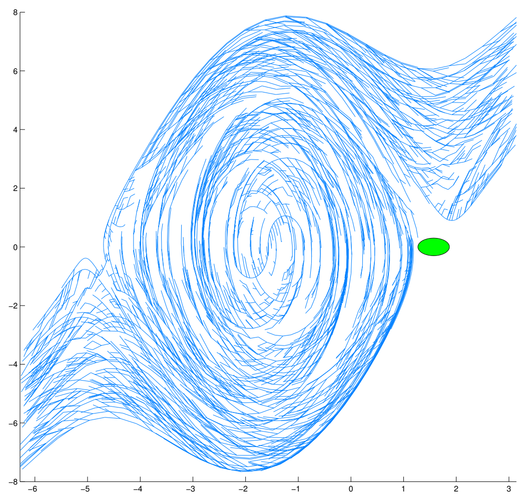

Fig. 5. An RG-RRT for the inverted pendulum problem. Note that the graph explores state space much better than standard RRT for this problem with no reachability constraints. In this instance, no goal bias sampling was used, resulting in a large tree. Had aggressive goal bias sampling been used, RG-RRT would likely have found a solution much quicker without constructing such a large tree.

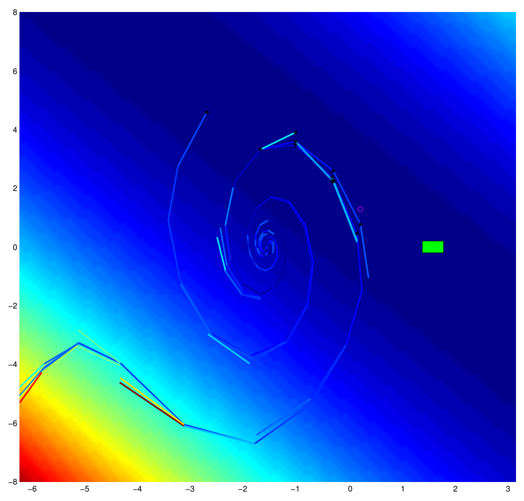

Fig. 6. A visualization of the LQR-RRT* algorithm running on the pendulum problem. The tree in the middle has edges color-coded by cost. The background color gradient represents the optimal cost-to-go for the linearized dynamics. The $x_{nearest}$ nodes, denoted by black asterisks, all fall in the lower well of the cost function. A Euclidean distance metric would have caused different $x_{nearest}$ nodes to be selected.

heuristics used. I examine both of these approaches in this project. Specifically, I implement the Reachabilty-Guided-RRT (RG-RRT) [7], as well as the recent LQR-RRT* [6] algorithm that uses locally linearizations at points in the tree for improved distance and extension heuristics.

D. RG-RRT

The reachability-guided RRT, or RG-RRT, is a technique used to restrict the expansion of an RRT [7]. The technique works by augmenting RRT's with an additional type of vertex, called reachable nodes. These vertices represent states reachable by applying control extrema; intuitively, the entire "reachable" set of states from some other state. In RG-RRT, new nodes are only added to the tree if they are closest to one of the reachable nodes as distinct from one of the regular nodes of the tree - thus favoring expansion into unexplored regions of state space via rejection sampling. The goal here is to minimize the number of redundant vertices in an RRT while using a less-than-perfect distance metric that would otherwise encourage such redundancy in a standard RRT. By keeping the tree small, the size of all other operations in RRT remain fast - thereby hopefully minimizing the time to find a solution.

Figure 5 demonstrates a tree built with RG-RRT for the inverted pendulum problem. Compare this with Figure 4 - the difference is extremely noticeable. RG-RRT does a much better job of expanding into yet unexplored regions of state space.

E. LQR-RRT*

Unlike RG-RRT, which seeks to restrict the expansion of the tree in already-explored areas, LQR-RRT* takes a different approach in that it attempts to improve the distance and extension heuristics of RRT and RRT* [6]. While it is sometimes possible to devise domain-specific functions for these requisites, such an approach does not generalize and is not easily applied to complicated dynamical systems. LQR-RRT* proposes to automatically derive reasonable distance and extension heuristics using locally optimal LQR controllers. LQR-RRT* is a very new algorithm, presented at ICRA '12.

LQR-RRT* builds on ideas proposed in [2]. Nodes in the tree, which represent points in state space, are locally linearized. An optimal LQR controller is than computed about these linearized dynamics, which can be done efficiently by solving an algebraic Ricatti Equation. This optimal controller provides a cost-to-go matrix S such that $J(\overline{x}) = \overline{x}^T S \overline{x}$ as well as an optimal control policy $\overline{u} = -K\overline{x}$, where $\overline{x} = x - x_0$ is the difference to the linearized node x_0, and similar for $\overline{u}$. It is important to note that these control policies only work well in regimes close to x_0. However, as the tree grows larger and more densely covers the state space, the average distance to random nodes will decrease, causing the linearizations to grow more and more accurate with respect to these random nodes.

This approach is illustrated in Figure 6, which shows a snapshot of the algorithm when selecting a group of nodes near the currently linearized state. The optimal cost function for the linearized dynamics is used to select the vertices that minimize $J(\overline{x})$, which for highly constrained dynamic systems does a much better job than the Euclidean distance metric.

One other point to note that is that, since LQR-RRT* uses

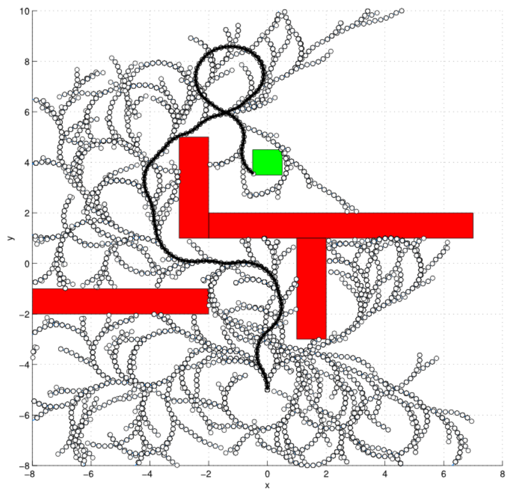
Fig. 7. Standard RRT solving the Dubins vehicle planning problem with obstacles. Note that due to the lack of any notion of optimality, the vehicle takes a rather adventurous and loopy trajectory near the top.

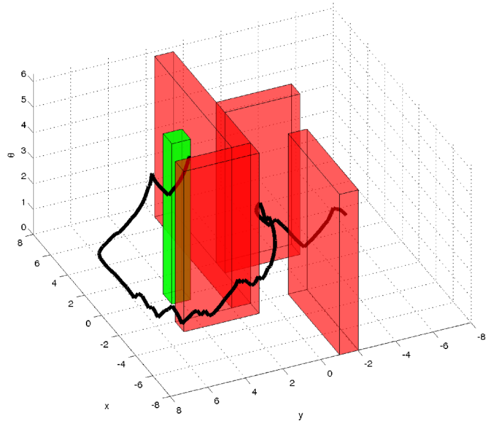
Fig. 8. A 3D visualization of the Dubin's vehicle reaching the goal.

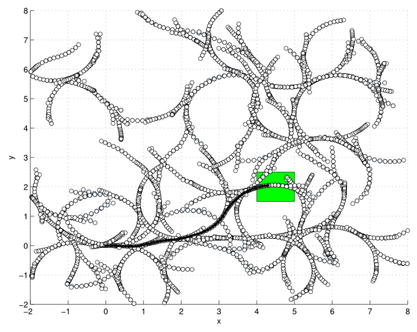
Fig. 9. RRT planning for the no-slip bicycle, with no obstacles.

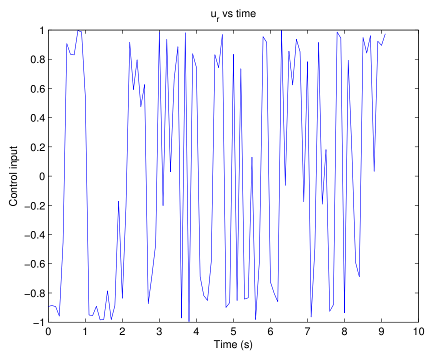

Fig. 10. Control inputs for the rear tire u_r for the no-slip bicycle. RRT's lack of any sense of optimality can result in unintuitive control inputs.

an LQR policy about a non-fixed point, the affine terms of the dynamics may come into play and threaten the control policy. As such, there is active research towards using an AQR-based heuristic for distances and extensions as an alternative to LQR.

IV. Results

In this section, we present the results of running the above-mentioned planning algorithms on the different bicycle models.

A. RRT

The standard RRT faired well for a number of dynamical systems, despite its oftentimes inaccurate distance metric. In all of the trials, a Euclidean distance metric was used. Figure 7 shows RRT planning for the Dubin's vehicle with many obstacles and a heavily restricted turning radius. RRT was successfully able to find a path. The tree however contains many redundant nodes, and oftentimes RRT leaves some areas of state space unexplored. Figure 8 shows a similar RRT solution to the Dubin's planning problem visualized in 3D, where the z-axis represents θ.

Next, RRT was used to plan for the more complicated non-slipping bicycle. The results without obstacles can be seen in Figure 9. The RRT was indeed able to generate a plan for this bicycle, though it took considerably longer - likely due to the increased dimensionality of the state space. This example also illustrates the suboptimal nature of RRT's. If we examine the control tape for a solution to the non-slip bicycle as in Figure 10, we see very spurious and seemingly random control application.

Finally, the RRT was also successfully able to plan paths for the slipping bicycle system. Results were similar in quality to the paths found by RG-RRT, which are described in the next section.

B. RG-RRT

RG-RRT shared many of the performance characteristics of RRT. Namely, it returned paths that often had suboptimal features. However, RG-RRT was able to successfully return paths much more consistently in these examples than RRT, and with greater coverage of state space. For example, when planning for the slipping bicycle, the standard RRT algorithm

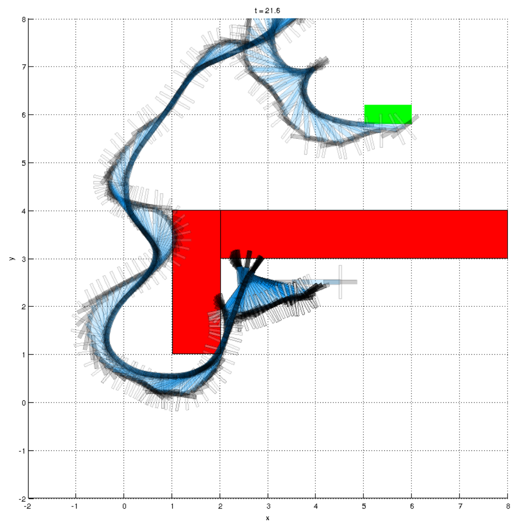

Fig. 11. A solution to the 7-DOF bicycle planning program solved using RG-RRT. The bicycle at different positions are superimposed over time. The blue blur represents the bicycle's frame, the gray bars are the handebars, and the black tire outlines can be seen clearly. In this example, the bicycle started near the corner of the red obstacles, and was tasked with driving around them with a goal region. This solution is for $C = 20$.

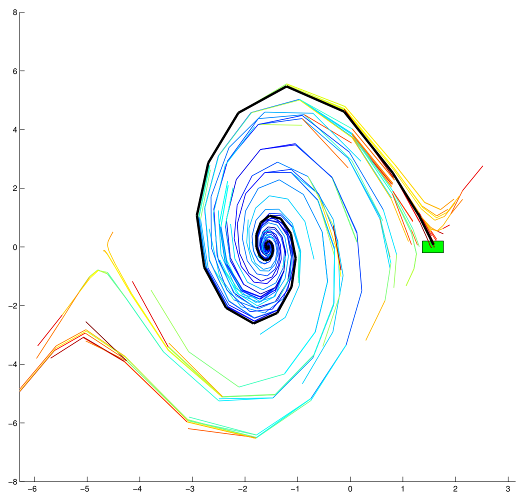

Fig. 12. Final output of the LQR-RRT* algorithm working on the inverted pendulum problem. The path found is shown in black. This case represents $R = 50$ with Q equal to the identity, thereby heavily penalizing control effort. Note the swing stages used. Branch and bound was also used to prune the search tree to minimize the number of vertices.

would fail to return a solution by 40,000 tree nodes roughly one third of the time, whereas RG-RRT would nearly always return a solution.

RG-RRT was able to successfully plan for the slipping bicycle model. An example solution trajectory is shown in Figure 11. The quality of solutions was quite comparable to that of a normal RRT, which makes sense given their shared heritage.

While RG-RRT would return a solution more consistently than RRT, I found in my experimentation that it often took more time to return a solution. This appeared to be due to the sometimes high rate of rejection sampling used in RG-RRT. In fact, I also found that without substantial goal bias (10%), RG-RRT would take significantly longer to find a solution if the path towards the goal happened to be closer to existing nodes in the tree as opposed to reachable vertices.

C. RRT*

As noted earlier, RRT* performance in kinodynamic planning is heavily dependent on the choice for the STEER() and DIST() metrics. I did limited testing of RRT* with dynamic constraints, but abandoned this approach in favor of pursuing LQR-RRT* early on in this project. I imagined that this would yield all of the benefits of RRT* but with the additional tool of automagically generating extension and distance metrics based on LQR policies.

D. LQR-RRT*

Although I was able to complete a working implementation of LQR-RRT*, I was surprised to discover that it was not applicable to some of the bicycle models presented earlier. Specifically, I tried running LQR-RRT* on the Dubin's vehicle model as well as the Slipping Bicycle Model, and MATLAB's LQR command failed to find optimal feedback agains in many circumstances, citing that certain poles of the system may not be controllable. I believe this is because the Dubin's vehicle is nonholonomic, making it difficult for a linearization to stabilize the system (the vehicle will never be able to move in certain directions, according to the linearization). This result was surprising to me, as I had not previously considered the ways in which LQR could fail.

In addition to not working properly on the Dubin's vehicle, LQR could not at times find controllers for the more complicated slipping bicycle, despite the fact that it can move in more directions than the Dubin's vehicle. Interestingly however, only certain configurations of the slipping bicycle resulted in LQR failure, whereas all configurations of the Dubin's vehicle were unsolvable. Specifically, configurations where the velocities of the bicycle's wheels with respect to the ground were 0 caused issues for LQR. This makes sense, as the bicycle cannot move laterally in such configurations.

Since the LQR requisite of LQR-RRT* failed for these bicycle-like systems, I was not able to successfully evaluate LQR-RRT* on them. Despite this however, I was able to successfully demonstrate LQR-RRT* on systems in which LQR worked reliably, such as the inverted pendulum. A sample solution for the swing-up task is shown in Figure 12. Note the smooth swingup is void of disturbances and visible suboptimal characteristics.

V. Conclusions, Lessons, and Ideas

Implementing and comparing all of these sampling based algorithms was an interesting experience. Some of my more interesting findings are listed below:

- In practice, RG-RRT is more reliable than RRT at finding a solution. However, in my experience, it also generally takes longer, in part due to high rates of rejections while sampling at some points.
- LQR policies can not always be computed, namely for nonholonomic systems! This means that LQR-RRT* may not always be possible.
- Planning high-precision, smooth paths with a small dt is quite difficult. As $dt \rightarrow 0$, the average number of vertices for any trajectory to the goal increases by approximately $1/dt$. An RRT has many such partial paths, and hence the size of the tree is exponential in $1/dt$.

In may be useful for future research to try and find other useful distance and extension heuristics in addition to LQR. In particular, I can envision several ideas based on AQR, trajectory optimization, or even a form of a relaxed Lypapunov function (perhaps computed via SOS techniques). Any of these may yield suitable, automatically-generated heuristics for RRT* in support of probabilistically optimal kinodynamic planning.

VI. Implementation

I have implemented all of the algorithms noted in the results in this paper from scratch for this project in MATLAB. Namely, please find working implementations of RRT, RG-RRT, RRT*, and LQR-RRT*, with branch and bound functionality (this amounts to over 2200 lines of MATLAB code). The code is designed to be generic and work with any dimensionality. Dynamics have been implemented for the inverted pendulum, Dubin's vehicle, non-slipping bicycle, and slipping bicycle. In cases where LQR-RRT* was used, code was also made to automatically generate the linearized jacobian matrices using the symbolic toolbox in MATLAB.

In order to optimize computational speed, I also used the MATLAB MEX compiler, which converts MATLAB to object code. For some problems, this resulted in a speed boost of up to 40X, and allowed me to construct larger more detailed RRT's with smaller dt increments.

In addition to the main algorithmic code, I also made several graphing and animation tools to visualize solutions in MATLAB. Please see the videos of selected bicycle solutions attached with this paper.

VII. Acknowledgements

I would like to thank Alejandro Perez for his remarkably useful, fast, and detailed advice in implementing the new LQR-RRT* algorithm - despite being quite busy and attending conferences! I'd also like to thank Joe Moore for his advice on RG-RRT speed and LQR failure.

References

[1] RW Allen, JP Chrstos, and TJ Rosenthal. A tire model for use with vehicle dynamics simulations on pavement and off-road surfaces. *Vehicle System Dynamics*, 27(S1):318–321, 1997.

[2] E. Glassman and R. Tedrake. A quadratic regulator-based heuristic for rapidly exploring state space. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 5021–5028. IEEE, 2010.

[3] S. Karaman and E. Frazzoli. Incremental sampling-based algorithms for optimal motion planning. In *Proc. Robotics: Science and Systems*, 2010.

[4] S. Karaman and E. Frazzoli. Optimal kinodynamic motion planning using incremental sampling-based methods. In *Decision and Control (CDC), 2010 49th IEEE Conference on*, pages 7681–7687. IEEE, 2010.

[5] S.M. LaValle and J.J. Kuffner Jr. Rapidly-exploring random trees: Progress and prospects. 2000.

[6] A. Perez, R. Platt, G. Konidaris, L. Kaelbling, and T. Lozano-Perez. LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012.

[7] A. Shkolnik, M. Walter, and R. Tedrake. Reachability-guided sampling for planning under differential constraints. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 2859–2865. IEEE, 2009.