i(ll-conditioned)LQR for Off-Road Trajectory Optimization?

Matthew Sivaprakasam
Carnegie Mellon University
Pittsburgh, U.S.A
msivapra@andrew.cmu.edu

Nayana Suvarna
Carnegie Mellon University
Pittsburgh, U.S.A
nsuvarna@andrew.cmu.edu

Thomas Detlefsen
Carnegie Mellon University
Pittsburgh, U.S.A
tdetlefs@andrew.cmu.edu

***Abstract*—Model Predictive Path Integral (MPPI) control is the standard for off-road autonomy due to its sampling-based nature. Its processing can be parallelized using a GPU and it can be applied to non-differentiable costmaps which is useful when dealing with costmaps generated by neural networks. However, this method comes with drawbacks since it requires a GPU to run in real-time, it requires the constant rollout of actions, and can be highly stochastic. In this work, we explore using iterative Linear Quadratic Regulator (iLQR) control in the context of off-road autonomy, addressing the challenges of gradients and hessians from a neural-network based costmap, and latency and systems without GPU.**

***Index Terms*—Model Predictive Control (MPC), Iterative Linear Quadratic Regulator (iLQR), Model Predictive Path Integral Control (MPPI)**

I. Introduction

Navigation in off-road terrain presents many challenges due to the unstructured and irregular nature of the environment. Vehicles have to adapt to mud, ditches, steep slopes, height irregularities and other conditions that aren't present in standard on-road driving scenarios. They have to quickly and effectively adapt to these conditions to ensure the safety of passengers as well as minimize damage to the vehicle.

State of the art approaches rely on Model Predictive Path Integral (MPPI) to efficiently plan in off-road environments. This process consists of sampling random actions, feeding them into a dynamics model to get a set of trajectories, and then picking the trajectory with the lowest cost. Although effective, MPPI requires a GPU to process a large set of trajectories in real time. This can make it infeasible to run on platforms with limited compute or in scenarios where the set of possible trajectories can't easily be modeled.

In this work, we propose a method for using iterative Linear Quadratic Regulator (iLQR) for trajectory optimization in off-road environments. We use it to optimize the lowest cost trajectory from a pre-computed library of trajectories generated using the kinematic bicycle model. The novelty in our approach lies behind using imperfect costmaps generated from inverse reinforcement learning.

- A framework for utilizing noisy costmaps for iLQR
- A real-time implementation of iLQR that uses CPU alone

This paper is organized as follows: Section II reviews related works. Section III outlines our proposed iLQR method and the various components of our system. Section IV steps through our qualitative and quantitative results. Section V provides our conclusion and touches upon future work.

Fig. 1: Autonomous All-Terrain Vehicle Platform used for data collection and deployment

II. Related Work

We can split the related works into three categories: the way the environment is represented through costmaps, existing approaches using MPPI, and existing approaches using iLQR.

A. Costmaps

Navigating in off-road environments requires an accurate representation of the environment that encodes traversability. Often, this is represented through a binary occupancy grid where areas with obstacles such as terrain height, curvature, and roughness [5], [6] are marked as occupied while remaining cells are marked as free. Although these approaches can provide compact representations, they lose information on the degree of traversability for regions spanning the map.

Representing the terrain instead as a costmap allows for greater degree of granularity and enables higher-level reasoning over which regions of the map are more travesable over others. This can be done through a controller or local and global planner. Costmaps are based on a variety of measurements from different sensors over multiple time steps. It can be difficult to incorporate these observations using traditional methods, instead learning based methods can be

used. S. Treist et al [2] presents an inverse reinforcement learning method to generate costmaps under uncertainty. We use the costmaps generated from this method as the underlying costmaps for our method.

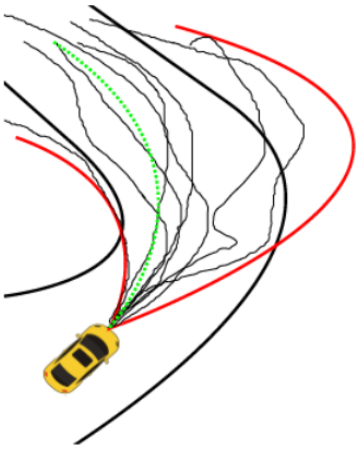

Fig. 2: Simple example MPPI problem for a wheeled vehicle, where randomly-sampled actions are rolled out and the best one that stays within the road is chosen.

B. Model Predictive Path Integral (MPPI)

MPPI is the standard approach for control on off-road autonomous vehicles. This is due to the sampling-based nature which allows MPPI to select the lowest cost trajectory based on non-differentiable environment representations. These can include methods such as a costmap generated by a neural networks [9]. [7] uses MPPI for an aggressive driving task with dynamics from neural networks. In [8] MPPI is used to incorporate traversability constraints based on rollouts to the elevation map.

Although effective, MPPI often requires a GPU and a dense trajectory library in practice in order to achieve real-time results. As a result, it is not as effective on edge devices with limited compute. This is our primary motivation for exploring using iLQR instead.

C. Iterative Linear Quadratic Control (iLQR)

iLQR is a common control method for many applications from biological movement systems [11] to UAV transport [12], but it is not often used for off-road autonomous driving because it requires the gradient and hessian of the costmap in order to perform the backward pass. Since costmaps can be generated by neural networks, they may not be differentiable and as a result cannot be used for iLQR directly.

In order to use costmaps with iLQR, the costmap can be modified to be differentiable. [10] presents a method that modifies the costmap to represent dangerous terrains through a combination of gaussian distributions. While this does make the costmap differentiable, it effectively creates a binary costmap and loses the intermediate information about traversability.

III. Method

A. Overview

Our goal is to safely navigate through complex off-road terrain by producing trajectories whose costs have been optimized over a costmap in order to traverse challenging terrain and avoid unsafe obstacles. The costmap is produced using inverse reinforcement learning. We initialize the iLQR process with the trajectory from a pre-computed trajectory library that has the lowest navigation cost on the costmap. For simplicity in this experiment, navigation "cost" comes directly from the costmap rather than also including costs derived from distance-to-goal and other common costs.

B. Learned Costmap

We use the inverse reinforcement learning process described in [2], through which a neural network is trained to observe geometric feature maps and predict a costmap that guides a planner to match previous expert demonstrations. The training data comes from TartanDrive 2.0 [13]. As shown in the work by Triest et. al., the costmap is expressive enough to use with gradient-free planners like MPPI to safely navigate difficult terrain. In this work we explore its viability in a trajectory-optimization approach that requires informative gradients and hessians that aren't necessarily guaranteed to be present in the output of a neural network.

C. Autonomy Platform

The costmaps we used for our method was generated through sensor data from a Yamaha Viking All-Terrain Vehicle (ATV) platform as seen in Fig. 1. The sensor payload on the vehicle includes three lidars, a stereo camera, a GNSS system, wheel encoders, and shock travel sensors. The data is collected at a 255 acre test site in Monroeville, Pennsylvania.

D. Vehicle Model

We use a kinematic bicycle model (KBM) to predict the robot's future states given an initial state and sequence of actions. Although it is a rather simple model, it has been shown to be sufficient in several prior works that use MPPI for off-road autonomous driving [2]. As an additional advantage, its simplicity makes it easy to differentiate at given states.

Our state space is $X = [x, y, \theta]$ and control space is $U = [v, \delta]$, where x, y are coordinates in 2D space, v is velocity, θ is heading angle, δ is steering angle, and L is the wheelbase. The state transition for the model is as follows:

$$\begin{aligned} \dot{x} &= v\cos(\theta) \\ \dot{y} &= v\sin(\theta) \\ \dot{\theta} &= v\frac{\tan\delta}{L} \end{aligned} \tag{1}$$

We also show the jacobian with respect to the state space and control space:

$$\frac{\partial f}{\partial x} = \begin{bmatrix} 0 & 0 & -vsin\theta \\ 0 & 0 & vcos\theta \\ 0 & 0 & 0 \end{bmatrix} \tag{2}$$

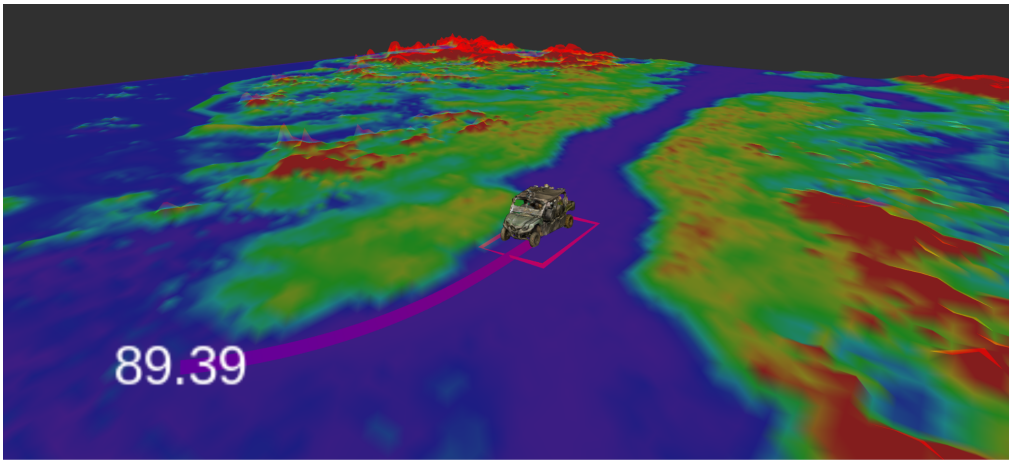

Fig. 3: Simulated view of the autonomous navigation stack. The ATV navigates by traversing low-cost regions according to a costmap (purple is low cost and red is hight cost).

$$\frac{\partial f}{\partial u} = \begin{bmatrix} cos\theta & 0 \\ sin\theta & 0 \\ \frac{\tan(\delta)}{L} & v\frac{sec^2\delta}{L} \end{bmatrix} \tag{3}$$

E. Trajectory Initialization

Because of the expressive nature of the costmap, it is possible (and in fact highly likely) for several local minima to be present. For example if provided an initial trajectory that goes through tall grass, an optimizer might find the smoothest part of the tall grass and ignore the more-traversable trail that goes in a different direction. To circumvent this issue, we initialize with the best trajectory from a trajectory library computed offline. We generate this by sequentially sampling inputs consisting of velocities ranging from 0 to 6m/s and steering rates from 0 to .3 radians/s. We then compute a trajectory for each sample by rolling it out through the kinematic bicycle model for 50 timesteps, where each timestep is 0.1 seconds. To compute the cost for a trajectory, we find the corresponding cost in the costmap for each point, and sum all the costs together. The selected trajectory at the end of this stage is the one with the lowest total cost with respect to the costmap. Since the trajectory library covers a wide area in front of the robot, it can be used to determine find the approximately optimal region of the costmap which in turn provides a sufficient initialization for the next stage to optimize to a good solution as shown in 5.

F. Pre-Computed Gradients and Hessians

The costmap remains static within the optimization process (we optimize at each timestep), and the cost values come directly from the x, y location on the map and rely on no other information. This means that not only is differentiating the costmap with respect to the state simple, but it can also be easily pre-computed for every point on the map in parallel simply by taking differentiating as many times as needed in the x, y directions. Then, during the different stages of the optimization process, those gradients are readily available simply by indexing into the differentiated map at a given state.

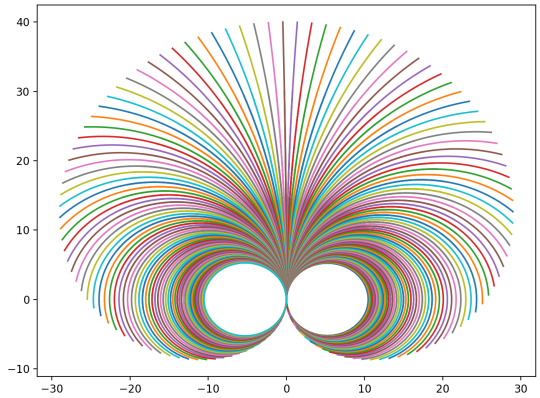

Fig. 4: Trajectory library used for planning

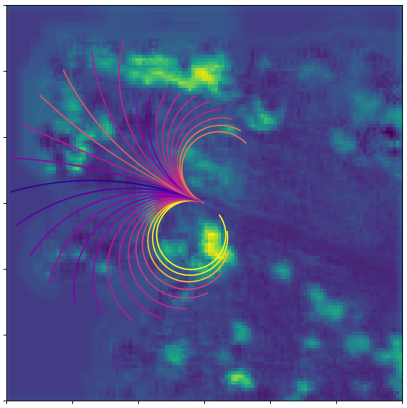

Fig. 5: As an initialization for the iLQR process, we provide the lowest cost trajectory (blue is lowest cost, yellow is highest) according to the costmap.

Since the predicted costmap is the output of a neural network, the gradients aren't always guaranteed to suitable for

optimization. As shown in 6, challenges such as noise, distinct boundaries, and low resolution can be present. To alleviate this, we apply a Gaussian Blur to the entire costmap in the hope that the smoothing affect provides bettery gradients for the optimizer to follow 7.

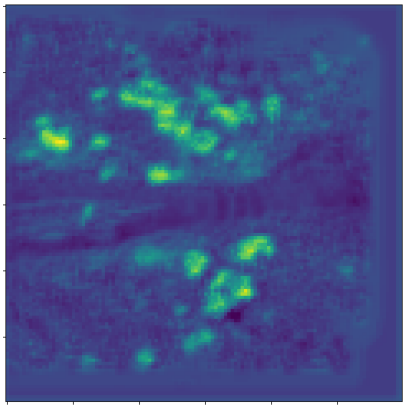

Fig. 6: Example costmap, produced using inverse reinforcement learning, to be used in optimization

G. iLQR on Learned Costmaps

1) Overview: We use an iterative Linear Quadratic Regulator (iLQR) to optimize the initial trajectory, following the standard process of integrating backwards from the end to calculate cost-to-go and its derivatives using dynamic programming in order to determine how to adjust the control inputs (backward pass). We then roll out the new control inputs, performing a line search to determine how much to adjust without overshooting (forward pass).

2) Backward Pass: We follow the procedure outlined in Algorithm 1 to perform the backward pass, where $Xref$ and $Uref$ are the initial trajectory and corresponding control inputs, and X and U contain the final outputs at the end of the optimization process. We employ the following cost function, where $costmap(x)$ denotes the cost value from the costmap at point x and Q_c is it's weight relative to the other weights Q, R:

$$\begin{aligned} J =& \frac{1}{2}(x_N - x_{\text{ref},N})^T Q_N (x_N - x_{\text{ref},N}) \\ &+ \frac{1}{2} Q_c costmap(x_N)^2 \\ &+ \sum_{k=0}^{N-1} \frac{1}{2}(x_k - x_{\text{ref},k})^T Q_k (x_k - x_{\text{ref},k}) ... \\ &+ \frac{1}{2}(u_k - u_{\text{ref},k})^T R_k (u_k - u_{\text{ref},k}) ... \\ &+ \frac{1}{2} Q_c costmap(x_k)^2 \end{aligned} \tag{4}$$

Since the costmap cost is solely dependent on state and doesn't affect the control gradients/hessians $\frac{d^2J}{du^2}, \frac{dJ}{du}$, they can be calculated as expected by differentiating the cost function and ignoring the costmap component. For calculating the state gradients/hessians, the costmap is differentiated and indexed as described in section IV.E . An example of the gradients and hessians of the costmaps are shown in Figs. 7, 8.

Algorithm 1 Pseudocode for iLQR Backward Pass

$P = \text{zeros}([N, \text{nx}, \text{nx}])$
$p = \text{zeros}([N, \text{nx}, 1])$
$d = \text{zeros}([N-1, \text{nu}, 1])$
$K = \text{zeros}([N-1, \text{nu}, \text{nx}])$
$\frac{d^2J}{dx^2}, \frac{dJ}{dx} \leftarrow \text{terminal\_cost\_expansion}(X[-1], Xref[-1])$
$P[-1] \leftarrow \frac{d^2J}{dx^2}$
$p[-1] \leftarrow \frac{dJ}{dx}$
$dJ \leftarrow 0.0$
for k from $N-2$ downto 0 **do**
  $\frac{d^2J}{dx^2}, \frac{dJ}{dx}, \frac{d^2J}{du^2}, \frac{dJ}{du} \leftarrow \text{stage\_cost\_expansion}(X[k], U[k], Xref[k], Uref[k], k)$
  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial u} \leftarrow \text{dynamics\_jacobian}(X[k], U[k])$
  $g_x \leftarrow \frac{dJ}{dx} + \frac{\partial f}{\partial x}^T \cdot p[k+1]$
  $g_u \leftarrow \frac{dJ}{du} + \frac{\partial f}{\partial u}^T \cdot p[k+1]$
  $G_{xx} \leftarrow \frac{d^2J}{dx^2} + \frac{\partial f}{\partial x}^T \cdot P[k+1] \cdot \frac{\partial f}{\partial x}$
  $G_{uu} \leftarrow \frac{d^2J}{du^2} + \frac{\partial f}{\partial u}^T \cdot P[k+1] \cdot \frac{\partial f}{\partial u}$
  $G_{xu} \leftarrow \frac{\partial f}{\partial x}^T \cdot P[k+1] \cdot \frac{\partial f}{\partial u}$
  $G_{ux} \leftarrow \frac{\partial f}{\partial u}^T \cdot P[k+1] \cdot \frac{\partial f}{\partial x}$
  $d[k] \leftarrow \text{pinv}(G_{uu}) \cdot g_u$
  $K[k] \leftarrow \text{pinv}(G_{uu}) \cdot G_{ux}$
  $p[k] \leftarrow g_x - K[k]^T \cdot gu + K[k]^T \cdot G_{uu} \cdot d[k] - G_{xu} \cdot d[k]$
  $P[k] \leftarrow G_{xx} + K[k]^T \cdot G_{uu} \cdot K[k] - G_{xu} \cdot K[k] - K[k]^T \cdot G_{ux}$
  $dJ \leftarrow dJ + g_u^T \cdot d[k]$
end for

3) Forward Pass: In the forward pass, we use the d, K matrices produced in the backward pass to modify the current control inputs. Starting with a term $\alpha = 1.0$, we weight the adjustment by α. If the cost of the new trajectory is not greater than the previous, then we halve α and try again, iterating this process until we achieve a better trajectory.

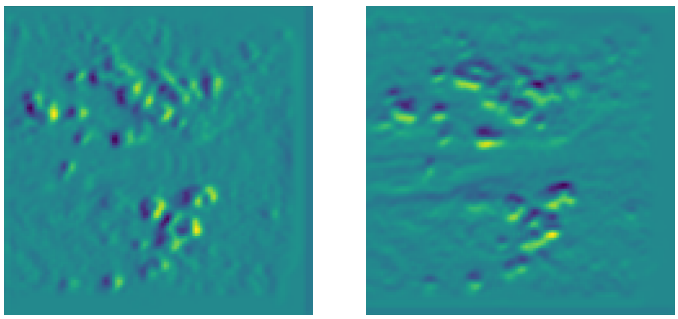

Fig. 7: An example of the x and y gradients after applying a Gaussian blur to the costmap

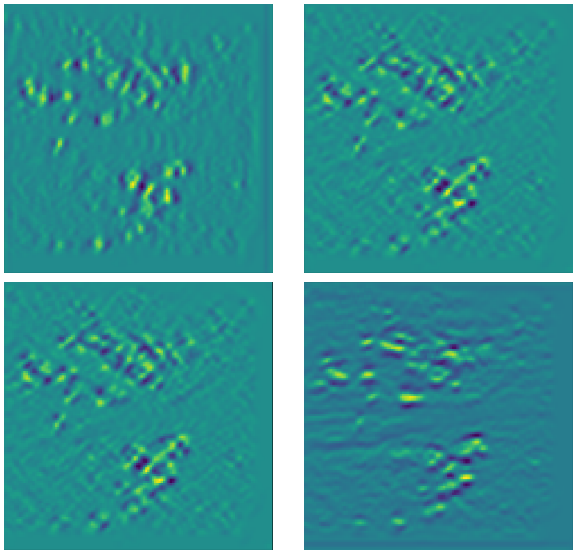

Fig. 8: An example of the hessian after applying a Gaussian blur to the costmap

IV. Results

For our experiments, we follow the formulation described in the methods section, selecting a trajectory from the trajectory library and using that as an initialization for iLQR. We allow iLQR to run for up to 10 iterations unless it converges earlier. Internally, the forward pass is allowed to iterate 15 times. For the costmap blurring, we use a kernel size of 5, which corresponds to a 2.5m square. We found that in practice, although the costmap is the main cost being optimized for, it was still necessary to include a small cost weight for tracking the original initial trajectory states. The final cost weights we chose were $Q = .0001, R = 0, Q_f = .3, Q_c = 1.5$.

Qualitatively, we find that this method can find a better path, and quantitatively we see a decrease in the overall cost of the trajectory on the costmap after optimization. To demonstrate more dramatic effects, we also tested a poor initialization by choosing the worst trajectory from the library and optimizing on that instead. As shown in 9, even despite a poor reference trajectory the optimizer is able to converge to a path that better avoids the high-cost regions.

In order to get our implementation to work in real-time at a rate comparable to our existing MPPI controller (>10Hz), we re-wrote our prototyped functions in Numba in order to enable just-in-time (JIT) compilation. The latency reduction achieved through this approach can be shown in Table 1. This increase in speed is impactful enough that we expect to be able to integrate more complex pieces (for example, sophisticated vehicle models) without losing real-time capability.

	Forward Pass	**Forward Pass Numba**	**Backward Pass**	**Backward Pass Numba**
Worst Case Speed (s)	0.27	0.004	0.04	0.0005

V. Conclusions and Future work

In this work, we explored a framework for utilizing noisy costmaps for iLQR on off-road autonomous vehicles that can run in real-time. Our method can be run on real-time on a CPU alone and does not require a dense trajectory library to create expressive paths around high-cost regions on a costmap.

In the future, we plan to make improvements to this method by improving the line search using the Wolfe conditions. When testing our implementation, we noticed that in some cases with the forward pass we observed weird behavior when following our current approach to determine step length. Using the Wolfe conditions could help determine step length more effectively.

We also plan to use a modified version of KBM that uses throttle and steering rate for more realistic dynamics. The current control input formulation of velocity and steering rate is not realistic for our vehicle. The input that controls velocity in this case is throttle, which has a non-linear mapping to velocity. We will experiment with using more sophisticated vehicle models that predict these processes more effectively, and maybe even explore using learned dynamics models. Since these models are not as easily differentiable, we would also explore using sampling methods to approximate their derivatives, such as in [3].

Finally, we plan to deploy this work on the Yamaha Viking and evaluate it against MPPI. This entails setting up a number courses at our test site that challenge various potential shortcomings of the two methods to see which one consistently performs better. We also need to come up with a fair way of evaluating them since one relies heavily on GPU and the other relies on CPU. We at least plan on keeping both implementations in the same language but we will consult more experts in this area in order to set up this evaluation in a sound manner.

Acknowledgment

This report was developed for Dr. Zachary Manchester's class Optimal Control and Reinforcement Learning. We would like to thank Dr. Manchester and his teaching assistants for their help during the development of this work. We would also like to thank Joshua Spisak for his advice when developing the idea for this work, and Samuel Triest for providing the learned costmaps on which we based our work.

References

[1] Y. Ding "Simple Understanding of Kinematic Bicycle Model"

[2] S. Triest, M. Guaman Castro, P. Maheshwari, M. Sivaprakasam, W. Wang, S. Scherer "Learning Risk-Aware Costmaps via Inverse Reinforcement Learning for Off-Road Navigation" IEEE International Conference on Robotics and Automation, May 2023

[3] Z. Manchester, S. Kuindersma "Derivative-Free Trajectory Optimization with Unscented Dynamic Programming" IEEE Conference on Decision and Control, Dec. 2016

[4] B. Plancher, Z. Manchester, S. Kuindersma "Constrained Unscented Dynamic Programming" IEEE/RSJ International Conference on Intelligent Robots and Systems, Sept. 2017

[5] P. Krusi, P. Furgale, M. Bosse, and R. Siegwart, "Driving on point clouds: Motion planning, trajectory optimization, and terrain assessment in generic nonplanar environments," Journal of Field Robotics, vol. 34, no. 5, pp. 940–984, 2017.

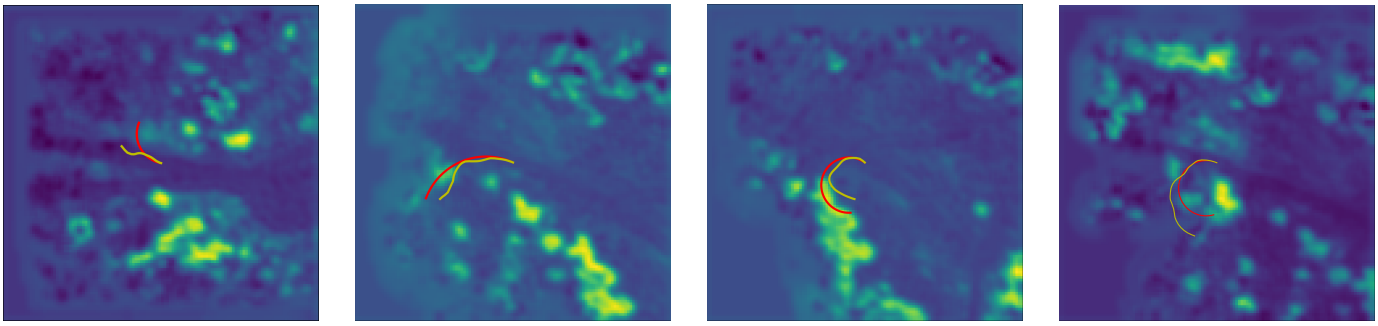

Fig. 9: Four examples of paths generated (yellow) based off the worst cost trajectory (red). The generated paths avoid the high cost regions (green-yellow) while still attempting to reach the final position.

[6] P. Fankhauser, M. Bjelonic, C. D. Bellicoso et al., "Robust rough-terrain locomotion with a quadrupedal robot," in 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2018, pp. 5761–5768.

[7] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, E. A. Theodorou, "Information Theoretic MPC for Model-Based Reinforcement Learning" in 2017 IEEE International Conference on Robotics and Automation (ICRA)

[8] T. Han, A. Liu, A. Li, A. Spitzer, G. Shi, B. Boots "Model Predictive Control for Aggressive Driving Over Uneven Terrain" Nov. 2023

[9] X. Cai1, M. Everett, L. Sharma1, P. R. Osteen, and J. P. How1 "Probabilistic Traversability Model for Risk-Aware Motion Planning in Off-Road Environments" Jul. 2023

[10] D. Fan , A. Agha-mohammadi, E. A. Theodorou "Learning Risk-Aware Costmaps for Traversability in Challenging Environments" in 2022 IEEE Robotics and Automation Letters. IEEE, 2022, pp. 279-286.

[11] W. Li, E. Todorov "Iterative Linear Quadratic Regulator Design for Nonlinear Biological Movement Systems" 1st International Conference on Informatics in Control, Automation and Robotics

[12] Y. Alothman, D. Gu "Quadrotor Transporting Cable-Suspended Load using iterative Linear Quadratic Regulator (iLQR) Optimal Control" in 2016 8th Computer Science and Electronic Engineering (CEEC)

[13] M. Sivaprakasam, P. Maheshwari, M. Guaman Castro, S. Triest, M. Nye, S. Willits, A. Saba, W. Wang, and S. Scherer "TartanDrive 2.0: More Modalities and Better Infrastructure to Further Self-Supervised Learning Research in Off-Road Driving Tasks" Feb. 2024