Generative CoLearn

Steering and cost prediction with generative adversarial nets in kinodynamic RRT

by

Nick Tsutsunava

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Friday October 5, 2018 at 14:00.

Student number:	4085396	
Project duration:	November 15, 2017 – September 15, 2018	
Thesis committee:	Prof. dr. ir. M. Wisse,	TU Delft, supervisor
	Dr. ing. J. Kober,	TU Delft
	Ir. T.D. de Bruin,	TU Delft
	Ir. W.J. Wolfslag,	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Generative CoLearn: steering and cost prediction with generative adversarial nets in kinodynamic RRT

Nick Tsutsunava*, Wouter J. Wolfslag†, Carlos Hernández Corbato*, M. Bharatheesha*, Martijn Wisse*

***Abstract*— Kinodynamic planning is motion planning in state space and aims to satisfy kinematic and dynamic constraints. To reduce its computational cost, a popular approach is to use sampling based methods such as RRT with off-line machine learning for estimating the steering cost and inputs. However, scalability and robustness are still open challenges in these type of Learning-RRT algorithms. We propose the use of generative adversarial networks (GAN) for learning of the steering cost and inputs. Furthermore, a novel data generation method is introduced, which is easy to learn and, in terms of parameter count, scales linearly to higher degrees of freedom. In our experiments, we show that the GAN has excellent generalisation capabilities, resulting in a considerable improvement in performance compared to the state-of-the-art. Consequently, we show that our method can scale to a planar arm and is robust to data dimensionality.**

I. Introduction

In robotics and motion planning, generating a trajectory that satisfies both kinematic and dynamic constraints is regarded as a computationally heavy problem [1]. Solving the required *two-point boundary value problem* (2PBVP) makes implementation increasingly prohibitive for real-time applications where computation time is crucial. Kinodynamic planning, path planning in state space, and sampling based methods, i.e., Rapidly-exploring Random Trees (RRT), are believed to be the most viable solution in overcoming this computational burden.

In kinodynamic planning, the RRT algorithm expands a tree of nodes in state-space, randomly exploring permissible states until the goal is reached, up to a desired tolerance. Since the Euclidean distance between nodes in state space is not appropriate, kinodynamic RRT algorithms typically require a metric that also takes the system dynamics into account. Due to the computational burden, this distance is approximated with pseudo-metrics, which generally underutilise the non-linear dynamics of the system, resulting in inefficient state space exploration [2], [3]. Another standing challenge is efficiently *steering* the robot from one state to another. This involves finding a *reachable* set of trajectories, i.e. dynamically feasible, after which the steering inputs are computed for the trajectory with the lowest *cost*.

Recently, promising methods in the literature for kinodynamic planning propose offline machine learning for improved approximation of the steering input, distance metric and reachable set, resulting in faster online planning [2], [4], [5]. These types of Learning-RRT algorithms use datasets that contain short segments of optimal trajectories and corresponding costs and steering inputs. The two main open questions for this approach is how to generate the dataset, and how to learn the steering input from the dataset, and doing so for a broad class of robots. In this paper, we address both questions within the context of RRT-CoLearn, the state-of-the-art algorithm proposed in [5].

The primary challenge in RRT-CoLearn is scaling from a pendulum to a system with more degrees of freedom, which was difficult due to the following issues.

- The k-nearest neighbour (KNN) algorithm used for online prediction stores the entire dataset in a tree. This results in poor generalisation for small datasets and slow computation as the dimensionality of data increases.
- KNN is not appropriate for extrapolation and can only predict meaningful results for queries that lie within some *reachable* boundary. This boundary parameter requires manual tuning.
- The generated data contains overlapping trajectories with different costs and steering inputs. A cleaning algorithm reduces the amount of such trajectories and improves the predictions of the learning algorithm. This cleaning algorithm requires careful tuning and reduces the amount of data to learn from.

We propose Generative CoLearn, an extension to RRT-CoLearn, which leverages a novel data generation method and new developments in machine learning that enable us to train a deep generative model on the data. We hypothesise that replacing the deterministic KNN algorithm with a generative model will result in improved generalisation and scalability, faster convergence and stable run-time performance. Additionally, we eliminate the necessity for data cleaning. To demonstrate our approach, we plan a path in state space for two systems: a pendulum and a 2-DoF planar arm. Due the high dimensional parameter space of the planar arm, we perform the majority of our analysis on the pendulum. In our method, outlined in Algorithm 1, we tackle the machine learning aspect to improve generalisation and robustness to data dimensionality and its parameterisation. This paper presents the following contributions.

- We propose a more general data sampling strategy that can be used for all fully-actuated serial chain robots, in a time optimal control setting with input bounds.
- We propose the use of the GAN framework for learning the steering costs and inputs, resulting in improved generalisation and robustness.

* Authors are with the Faculty of 3mE, Delft University of Technology. nicktsutsunava@gmail.com, {c.h.corbato,m.bharatheesha,m.wisse}@tudelft.nl

†Author is with the School of Informatics, University of Edinburgh. wouter.wolfslag@ed.ac.uk

- We show that we can effectively use the discriminator of the GAN as a classifier for trajectory feasibility, replacing the reachable bound, which scales poorly with system size.

The paper is organised as follows. In Section II, we present the data sampling strategy used in Generative CoLearn. In Section III we discuss our reasoning for using GAN and its implementation. Our evaluation metrics are presented in Section IV with the experimental method in Section V. The results and a discussion with future work are presented in sections, VI and VII, respectively. Finally, we conclude our findings in section VIII.

II. Data Generation

A main step in RRT-CoLearn algorithms is to generate optimal trajectories using the optimal equations of motions obtained via the maximum principle of Pontryagin [6]. The dataset is generated by randomly sampling the initial state and costate, and integrating the equations of motion. For the type of problem under study, the initial state and costate must satisfy a constraint. An ad-hoc approach to solving that constraint for the pendulum swing up was used in [5]. This solution was complex, even for this simple system. As a result, the scalability of this method is not obvious.

In this section, a sampling procedure that results in feasible initial state-costate combinations is presented. This sampling procedure is valid for a large class of problems: time optimal control of fully actuated serial chain robotic manipulators.

For the class of fully actuated open chain robots, the equations of motions are represented by the following differential equation:

$$M(q)\ddot{q} = C(q,\dot{q}) + G(q) + \tau \tag{1}$$

Where q are the generalised coordinates, M is a positive definite mass matrix, C represents the convective acceleration terms, G the forces due to potential energy and finally τ are the motor torques. Note that friction losses are neglected.

The optimal equations of motions are found by first specifying the cost function. For time optimal control, the cost J is simply the final time, which can be expressed as:

$$J = \int 1 \mathrm{d}t \tag{2}$$

The next step is to define the Hamiltonian, which is the sum of the integrand of the cost function, J, and the inner product of the costate with the state derivative. Here we split the costate into two parts λ, which is multiplied with velocity, and μ which is multiplied with the acceleration. As a result, we obtain the Hamiltonian

$$\mathcal{H} = 1 + \lambda^\top \dot{q} + \mu^\top M^{-1}(q)(C(q,\dot{q}) + G(q) + \tau) \tag{3}$$

where Equation 1 was used to find the derivative of velocity.

The optimal input is found by minimising the Hamiltonian subject to the input constraints. In our case, each motor torque is bounded by a constant: $|\tau_i| \leq \hat{\tau}_i$, where i is the index of each motor, and $\hat{\tau}_i$ is the torque bound for that motor. This results in the optimal torque: $\tau^* = -\text{sign}(M^{-\top}\mu) \odot \hat{\tau}$, where $\odot$ signifies the element-wise product, and $\hat{\tau}$ is the vector of bounds on the motor torques.

We obtain the optimal Hamiltonian by filling in the optimal motor torque in the Hamiltonian:

$$\mathcal{H}^* = 1 + \lambda^\top \dot{q} + \mu^\top M^{-1}(q)(C(q,\dot{q}) + G(q)) - |M^{-\top}\mu|_E \odot \hat{\tau} \tag{4}$$

From this optimal Hamiltonian, the optimal equations of motion are derived by taking appropriate derivatives, as in:

$$\ddot{q} = \frac{\partial \mathcal{H}}{\partial \mu}, \quad \dot{\lambda} = -\frac{\partial \mathcal{H}}{\partial q}, \quad \dot{\mu} = -\frac{\partial \mathcal{H}}{\partial \dot{q}} \tag{5}$$

To generate the data for Generative CoLearn, these optimal equations of motion are numerically integrated, starting from randomly sampled initial states and costates. For free final time, but otherwise time-independent problems, the value of the optimal Hamiltonian is constrained to be 0 at $t = 0$. This effectively is a constraint on the initial state-costate combination.

For sampling initial states and costates while taking into account that constraint, the following approach is proposed. First, sample the initial states uniformly over their domain (q_0 and $\dot{q}_0$). Second, sample a costate uniformly over the unit sphere, resulting in $\hat{\lambda}_0$ and $\hat{\mu}_0$. The costate used in simulation will be a positive constant (α) times that unit costate vector. By restricting α to be positive, the optimal Hamiltonian is linear in α, meaning we can solve the constraint:

$$\alpha = \frac{-1}{\mathcal{H}^*(q_0, \dot{q}_0, \hat{\lambda}_0, \hat{\mu}_0) - 1} \tag{6}$$

The value computed for α might not be positive, meaning it is invalid. We use rejection sampling to sample $\hat{\lambda}_0$ and $\hat{\mu}_0$ such that a valid α is obtained. Due to the structure of the optimal Hamiltonian, if a sampled initial costate is invalid, its negative must be valid. This is due to fact the only term in Equation 4 that is non-linear in the costates is always negative. As a result, at most half of the sampled costates are expected to be rejected, making rejection sampling an effective approach.

Note that the state equations will remain unchanged when multiplying the costate with a positive constant α. The costate equations of motion will simply scale when multiplying the costate with α. As a result, the state equations will behave exactly the same over time, when multiplying the initial costate by a positive factor. We therefore sample the initial costate while verifying a positive α as described above, and directly use that costate.

The time-optimal control problem results in a bang-bang type torque signal, with the sign of the torque signal the opposite of that of the costate μ. Due to this bang-bang nature, the state trajectory for simulations with a different costate (μ and λ) is the same, at least until μ has switched sign. When unbiasedly sampling costates from the complete unit sphere, many sample trajectories will not contain a switching μ. As a result, the final states of the trajectory will be lumped together, as shown in Fig 1. The data of this

figure is generated for a pendulum swing up, with the initial state at the origin.

To more evenly distribute the data, we bias the costate sampling to only include the regions of the unit sphere that will cause μ to switch in time. The region from which to sample the costate is computed by approximating the time derivative of μ (Eq. 5) as $\dot{\mu} = -\hat{\lambda}_0$. For a given $\hat{\lambda}_0$ and final time, this differential equation provides bounds for values of $\hat{\mu}_0$ that will likely contain a switch. Due to the bounds being derived from an approximation, we symmetrically adjust the bounds, such that the region they contain is doubled in size. As seen in Fig. 1, the resulting final states are more evenly spread out.

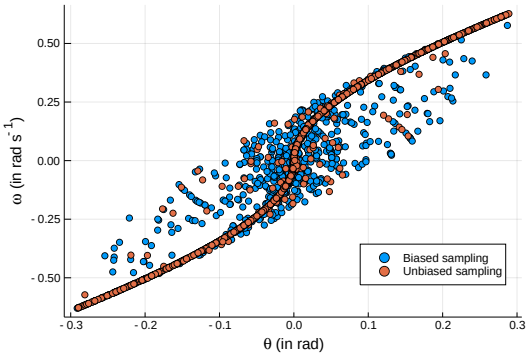


Fig. 1: Biasing the costate sampling results in an improved state space coverage.

III. Machine Learning

Deep generative models have become popular in unsupervised learning. These models attempt to capture the probability distribution of given data in order to generate new but similar samples. Currently, the two most popular frameworks are the variational autoencoder and generative adversarial networks (GAN) [7], [8]. In this research, the GAN in particular seems more suitable as it does not make assumptions on the latent model of the data and has superior generation quality [9]. Aside from image generation, GANs have seen applications in medical data analysis, robotics and autonomous driving [10]–[12].

Learning in the unsupervised GAN framework consists of two loss functions, typically implemented with neural nets, playing a *minimax* game. The generator $G(z)$ samples a latent vector z and aims to fool the discriminator $D(x)$ with generated samples. In turn, the discriminator learns to discern real samples x from generated samples. The feedback provided by the discriminator improves the generated samples. Another approach is to use these type of generative models in a supervised manner in conjunction with real continuous data, where the model is conditioned on some auxiliary information [13]. In our case, we use the costates and steering cost as target data and the trajectories as conditioning labels for the model. At run-time we can query the generative model with trajectories to predict the costates and cost. We outline our reasoning for using GAN in learning-based kinodynamic planning.

- Based on promising results in literature on image generation with GANs, it is reasonable to assume that the GAN can resolve the non-linearities in our dataset.
- The GAN implicitly learns a continuous probability distribution from which it can sample the target data. This makes it more suitable for problems with multiple solutions, resulting in improved generalisation [14].
- The discriminator can be used in run-time to classify which trajectories are feasible, eliminating the need of the reachable bound.
- Neural network time complexity is not dependent on data dimensionality making it much faster than a tree lookup for large amounts of data.

The original formulation of the GAN is notoriously difficult to optimise [15]. Recently, a new adversarial objective was proposed by implementing a least squares loss on the networks, resulting in stable training dynamics and slower saturation [16]. We combine this least squares objective with the conditioning approach proposed by [13] and formalise the conditional networks with:

$$\begin{aligned} \min_D V(D) = \frac{1}{2}\mathbb{E}_{x\sim p_{data}(x)}[(D(x|y)-1)^2] \\ + \frac{1}{2}\mathbb{E}_{z\sim p_z(z)}[D(G(z|y)))^2] \\ \min_G V(G) = \frac{1}{2}\mathbb{E}_{z\sim p_z(z)}[(D(G(z|y))-1)^2] \end{aligned} \tag{7}$$

By concatenating the label y to the input of the generator and the discriminator during training, both networks are conditioned on the extra information provided. At run-time we query the networks with the same concatenation step. As a result, the trained discriminator is able to classify the feasibility of the generated costates and trajectories with $D(G(z|y)) > 0$ where 0 is the decision boundary.

IV. Metrics

We propose a set of metrics for analysing the performance of GAN and KNN in Generative CoLearn. For measuring the online RRT performance, 1) the steering error, 2) node count, 3) failure rate, 4) and planning time are used as the metrics, which are standard in similar literature [2], [5]. The steering error is used as a measure for accuracy and is computed as the mean squared error (MSE) between the target and final state. It gives insight to the robustness and generalisation capacity of the learning algorithm when queried with unseen data at run-time. This error affects the node count, which is therefore an indirect measure of the model performance. For example, a high node count results from poor costate predictions leading to the final state constantly diverging from the target state. This predominantly occurs near the goal when the required trajectory is extremely short, resulting in a probability of the algorithm getting stuck. Therefore, we terminate the planning and regard it as a failure once a critical amount of nodes is reached. Finally, the planning time is dependent on the amount of nodes but is also affected by external factors such as implementation efficiency, model sampling speed and available computing power. We note

that these experiments were run on a shared machine with a suboptimal implementation. Therefore, the planning time between the planar arm and pendulum is not comparable.

Good generalisation is typically achieved if the learning algorithm is able to generate samples with the same features as the ground truth. One of the main features is that the costates are sampled from the unit sphere. Therefore we require the norm of the costates to approach unity. In addition, if the learning algorithm has been conditioned well, the error of the states should be low relative to their domain. This can be tested by generating costates from a test set and running a forward simulation to observe the error between true and final states. Therefore, we validate the models with 1) the proximity of generated costates to the unit sphere measured as the norm of the generated costates, 2) and given a test set, the mean squared error between target and final states, for each state θ and ω.

V. Experiments

A. Data Generation and Learning

In this section we follow the approach as outlined in Algorithm 1 and implement the pendulum and planar arm with Eq. (1) for data generation. For simplicity, the masses, torque bounds and link lengths are set to unity. The pendulum task involves a swing-up for which we bound the torque to $\tau_{\text{max}} = 0.5$, instead. Similar to the pendulum dataset in [5], we generate 40000 samples by uniformly sampling an initial position, measured in rad, from $\theta \sim \mathcal{U}(\frac{-3\pi}{2}, \frac{\pi}{2})$ and the initial velocity, in rad s^{-1}, from $\omega \sim \mathcal{U}(-\pi, \pi)$. This dataset covers the state-space for a swing up from state equilibrium $(\theta, \omega) = (-\pi, 0)$ to the unstable equilibrium at $(0, 0)$. For the planar arm we simplify the problem to reduce the data size and plan a path from $(\theta_0, \theta_1, \omega_0, \omega_1) = (-\frac{\pi}{4}, 0, 0, 0)$ to $(\frac{\pi}{4}, 0, 0, 0)$. The initial states are sampled from $\theta \sim \mathcal{U}(-\frac{\pi}{2}, \frac{\pi}{2})$ and $\omega \sim \mathcal{U}(-1, 1)$ to generate 1 million points. Additionally, we generate 100,000 points at the start and goal where the initial states are sampled normally with $\sigma^2 = 0.1$. We improve the state space coverage by setting half of the initial states as final states and simulating backwards in time.

For testing and validation we use 20% of the dataset. We run 10 epochs where during each epoch we generate the data, fit the model and run the RRT algorithm 100 times. For the planar arm we generate the data only once across all epochs. The experiment either is complete if the Euclidean distance to the goal is within 0.15, or times out and restarts after 1000 nodes. The epoch is terminated after 200 RRT attempts, resulting in a 50% fail rate. The GAN is trained for 30000 epochs with a batch size of 100. The generator and discriminator are trained equally. The noise vector has size 32 and is sampled from $z \sim \mathcal{U}(-1, 1)$. The generator has 5 hidden ReLU layers with (32, 64, 128, 256, 512) units whereas the discriminator has 5 hidden Leaky ReLU layers with (512, 256, 128, 64, 32) units and batch normalisation for each hidden layer. For the planar arm both networks have 3 hidden layers with each 256 units. The output layer for the generator is split and constrained to the range of the data using appropriate activation functions (e.g. $\tanh$ for the costates and ReLU for the cost). The discriminator output has linear activation. Similar to RRT-CoLearn, we use the KNN algorithm [17] with $k = 3$ and the Euclidean distance metric. The leaf size is set to 10.

Algorithm 1 Generative Colearn

```
data ← generate_data()
G, D ← train_network(data)
node_tree ← x_start
while not goal() do
    x_random ← random_state()
    N_reachable ← empty()
    for i in node_tree do
        T ← {x_i, x_random}
        û, cost ← G(T)
        if D(û, T) > 0 then
            append(N_reachable, {x_random, û, cost})
        end if
    end for
    x_near, û ← min_cost(N_reachable)
    x_final ← simulate(x_near, û)
    append(node_tree, x_final)
end while
```

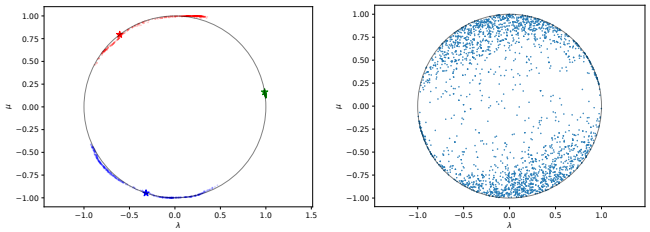
All the experiments are run in Ubuntu 16.04 on an Intel(R) Xeon(R) CPU E5-2697 v4 and an NVIDIA Titan Xp. The code for Generative CoLearn[1] is written in Python 3, Julia 0.6 and Tensorflow 1.8.

B. Reachability and Planning

In kinodynamic applications it is not trivial to expand the nearest node to a randomly sampled state as this problem involves solving a 2PBVP to determine which trajectories are dynamically feasible [18]. Similar to [5], this difficulty exposes itself via limitations of the generalisation capability of the learning algorithm. For non-parametric algorithms such as KNN, this generalisation is limited and depends on interpolation in Euclidean space. Therefore, queries that are not within some reachable bound δ_{max} to the data stored within KNN result in invalid predictions. Since our parametrisation for steering is different from RRT-CoLearn, we run a parameter sweep with 100 runs for each δ_{max} sampled from `linspace(0.1,1.0,10)` and choose the value resulting in the most balanced performance in terms of fail rate, steering error and node count. We find for GAN, $\delta_{\text{max}} = 0.4$ and for KNN, $\delta_{\text{max}} = 0.3$.

At run-time, we uniformly sample a random node within the bounds the data was generated in and build a matrix, T, of trajectories from the nodes in the tree. The matrix contains entries $(\theta_i, \omega_i, \theta_{\text{rand}}, \omega_{\text{rand}})$ for i nodes in the tree. For each trajectory in T, we find nearest neighbours in the dataset and store the nodes (θ_i, ω_i) in a set $N_{\text{reachable}}$ if the Euclidean distance is within δ_{max}. If an empty set is returned, a new random node is sampled. For the planar arm experiments we use the trained discriminator to find the reachable set. As shown in Algorithm 1, we feed the output of the generator into the discriminator along with the queried trajectory. We append the nodes in the tree to $N_{\text{reachable}}$

[1] https://github.com/ortix/generative-colearn



(a) GAN (b) KNN

Fig. 2: The GAN predicts costates on the correct half of the unit circle (2a). Stars indicate the true costates. Clustering of the costates (green star) occurs if μ switches sign during simulation. The KNN algorithm interpolates over the nearest neighbours resulting in predictions inside the unit circle (2b).

TABLE I: We show that cleaning of the dataset is not necessary as reducing the amount of overlapping trajectories has a detrimental effect on KNN.

d	0	0.1	0.2	0.3	0.4	0.5
Error	0.09	0.09	0.12	0.16	0.17	0.21
Nodes	156	166	270	327	447	549
Fail rate	40%	43%	32%	24%	14%	24%

if the corresponding trajectories result in a positive output from the discriminator. We justify this approach with the following reasons. First, using the trained discriminator as a classifier eliminates tuning the reachable bound. Second, and more critically, we eliminate the need to perform a nearest neighbour lookup, which is computationally expensive as the dataset grows.

With the reachable set of nodes, the cost and steering inputs for the corresponding trajectories is predicted. The predicted cost is clipped to a lower bound at 10^{-5} times the euclidean trajectory distance ensuring that 0 cost nodes in $N_{\text{reachable}}$ are not constantly expanded from. When the trajectory involves the goal state, the costates are sampled from a normal distribution with the mean as the predicted value and $\sigma^2 = \pi/2$, improving the algorithm performance near the goal. We only perform this sampling step for the pendulum experiments. Finally, the node with the lowest cost is expanded by running a forward simulation. The final state is then appended as a node to the tree. This process is repeated until the final state is within a Euclidean distance of 0.15 to the goal.

VI. Results

To evaluate[2] the learning performance, we initially feed GAN and KNN a set of trajectories from the test set and generate costates. From Fig. 2 we find that GAN generates costates from the unit circle whereas KNN's prediction lie within the circle as a result of averaging nearest neighbours. Additionally, we fit the norm of the costates to a normal distribution and show that GAN has a lower variance. Furthermore, GAN outperforms KNN with lower errors for the cost and the states after simulation with predicted costates. We summarise these results in Table II. A more thorough analysis for GAN with three queries reveals two distinct features in the predictions. First, the generated costates always have the correct sign for μ, which indicates that the algorithm is able to uncover the relationship between torque and trajectory. Second, some costates result in a close clustering to the ground-truth. This is due to longer trajectories with a switch in torque sign resulting in a one-to-one relationship between μ and λ. For shorter trajectories with no switch, any value for λ is valid. When KNN is queried in the same way, each predicted costate lies on the same location within the unit circle as a result of interpolation and deterministic nature of the algorithm. Consequently, costates generated from the entire test set are scattered within the unit circle as seen in Fig. 2b. The predicted costates that lie nearest to the ground truth on the unit circle are not necessarily the nearest in the dataset. These nearest-neighbours lie on different locations on the unit circle and are interpolated, resulting in the prediction to lie within the unit circle.

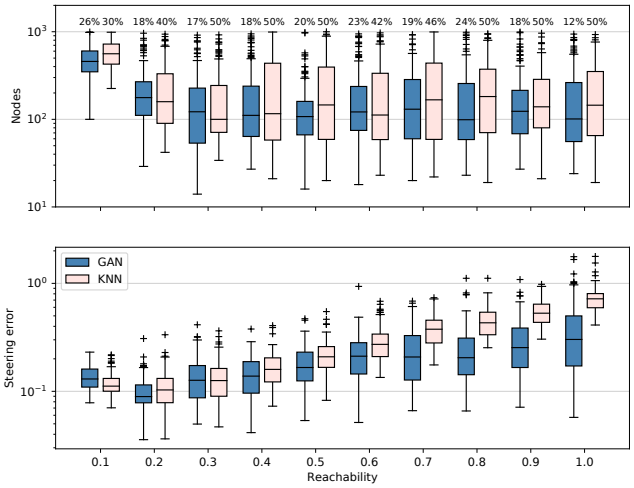


Fig. 3: Reachability has a profound effect on KNN while GAN is less sensitive. The fail rate is shown at the top.

To gain a better insight on the effect of the parameterisation of the costates, we investigate the necessity for cleaning. We follow the cleaning algorithm outlined in [5] with variations in the cleaning parameter d and find in Table I that cleaning has a detrimental effect on KNN. Therefore, we do not clean the dataset in our experiments. Furthermore we investigate the effect of reachability parameter $\delta_{\max}$. From Fig. 3 we see a profound effect on KNN. An increase in reachability is detrimental for KNN while GAN is less sensitive, confirming our hypothesis that GAN is robust and performs significantly better in generalisation. As opposed to KNN, GAN has a low fail rate with 0 terminations and a low variability in node count. The steering error does increase with an increase in $\delta_{\max}$ for both algorithms but does so at a slower rate for GAN and remains lower than KNN.

We run our planning algorithm for both the pendulum and

[2]We report all metrics as median values over all epochs.

TABLE II: The top rows show the results for the pendulum. The bottom row is for the planar arm results with only GAN.

	RRT Results				Model Validation			
	Error	Time	Nodes	Fail	Proximity	θ_E	ω_E	Cost$_E$
GAN	0.15	0.85 s	92	13%	$1.0 \pm .09$	.240	.713	.006
KNN	0.11	0.81 s	132	50%	$.85 \pm .50$	.250	.723	.012
2DOF	0.16	8.80 s	94	0.1%	$1.0 \pm .10$	.006	.049	.005

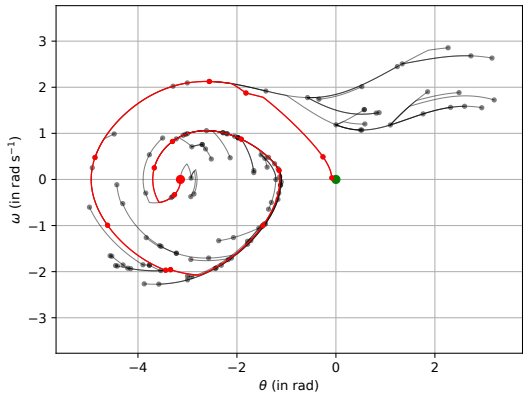


Fig. 4: This path generated by the GAN shows the natural evolution of the pendulum dynamics.

planar arm and summarise the results in Table II. For the pendulum, GAN outperforms KNN in terms of stability and node count. The steering error is slightly higher for GAN, caused by the relatively low state space coverage. However, this is a reasonable trade-off for the end performance. The planning time for both models is similar, yet depends strongly on data dimensionality for KNN. The prediction time for KNN and GAN is 5 ms and 1.1 ms, respectively. Finally, we show the path planned by GAN in Fig. 4, with the natural evolution of the dynamics of the pendulum in state space. This particular path requires 3 swings, where on the final swing the goal is reached. We clearly observe a switch in torque between the second and third node resulting in a rapid deceleration.

Finally, we test our approach on a planar arm. We disregard KNN for this experiment as the dataset size makes computation time impractical. Our algorithm is able to converge 99.9% of the time. The median node count is 94 with a steering error of 0.16. We note a prediction time of 1.1 ms, which is the same for the pendulum experiments. In terms of model validation, we observe extremely low errors for the final states after simulation and for cost prediction. We attribute the discrepancy in performance between the two systems to the following. First, we use a much larger dataset for the planar arm compared to the pendulum. This results in GAN generalising better. Second, the proposed task is relatively simple and does not take gravity into account. However, given enough state-space coverage in the dataset, it should be possible to perform more complex tasks.

VII. Discussion and Future Work

We have introduced a novel application of a GAN, which shows promising results for learning the steering cost and inputs in kinodynamic planning.

The proposed data-sampling strategy is general, in the sense that it is valid for all fully actuated serial chain robots in a time-optimal control setting with input bounds. The method also applies when including a squared torque cost term to the problem, which represents the energy consumption of the movement. The trick of multiplying a sampled unit length costate with a positive factor α still applies. The difference is that the Hamiltonian becomes a continuous piecewise polynomial with respect to α (Eq. 6). Because the polynomials are of degree one and two, the roots of the Hamiltonian can still be readily found. The property also remains in this case that at least half of the sampled costate will have a positive α for a root of the Hamiltonian. Care must be taken to multiply the sampled costate with α, as the resulting trajectory is not invariant to this multiplication when including a squared torque cost. Evaluating the effectiveness of the sampling method for higher degrees of freedom systems, and when including a squared torque term, is important for real-world applications of the techniques.

Supervised learning in deep generative models such as GANs is still an active field of research and no consensus exists on how to properly condition the networks on auxiliary information. Therefore, we propose several points for investigation. First, *minibatch discrimination* or *feature matching* can be used to improve the quality of the generated samples and discriminator classification performance, respectively [15]. Second, the data size requirement can be reduced by improving generalisation with structured and disentangled latent space. This has the benefit of enabling meaningful interpolation over a latent feature and can be achieved with an additional inference network or information maximisation between mutual latent variables [19], [20].

VIII. Conclusion

We presented Generative CoLearn, an extension to RRT-CoLearn using the GAN framework for predicting the required costates and steering cost in kinodynamic path planning. This approach solves two issues in the original RRT-CoLearn implementation. First, robustness and generalisation are improved. We show that GAN achieves a higher success rate and lower node count compared to KNN, attributed to the implicitly learned probability distribution from which the target data is sampled. Second, we demonstrate that our method scales robustly to a 2-DoF planar arm, which is impractical with KNN due to the large amount of data necessary. We show that the discriminator is a suitable classifier for feasible trajectories. The large amount of data necessary for complex systems is still a challenge. However, we believe deep generative models are good candidates to tackle this problem and can ultimately open new doors in kinodynamic RRT in terms of performance and scalability.

Acknowledgment

We gratefully acknowledge the support of NVIDIA Corporation for donating a Titan Xp GPU for this research.

REFERENCES

[1] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning," *J. ACM*, vol. 40, pp. 1048–1066, nov 1993.

[2] M. Bharatheesha, W. Caarls, W. J. Wolfslag, and M. Wisse, "Distance metric approximation for state-space RRTs using supervised learning," in *2014 IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, pp. 252–257, IEEE, sep 2014.

[3] T. Kunz and M. Stilman, "Probabilistically complete kinodynamic planning for robot manipulators with acceleration limits," in *2014 IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, pp. 3713–3719, IEEE, sep 2014.

[4] R. Allen and M. Pavone, "A Real-Time Framework for Kinodynamic Planning with Application to Quadrotor Obstacle Avoidance," in *AIAA Guid. Navig. Control Conf.*, (Reston, Virginia), American Institute of Aeronautics and Astronautics, jan 2016.

[5] W. J. Wolfslag, M. Bharatheesha, T. M. Moerland, and M. Wisse, "RRT-CoLearn: Towards Kinodynamic Planning Without Numerical Trajectory Optimization," *IEEE Robot. Autom. Lett.*, vol. 3, pp. 1655–1662, jul 2018.

[6] D. S. Naidu and S. Naidu, *Optimal Control Systems*. Boca Raton, FL, USA: CRC Press, Inc., 2002.

[7] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," *arXiv Prepr. arXiv1312.6114*, dec 2013.

[8] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Nets," in *Adv. Neural Inf. Process. Syst. 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 2672–2680, Curran Associates, Inc., 2014.

[9] M. Lučić, K. Kurach, M. Michalski, S. Gelly, and O. Bousquet, "Are GANs Created Equal? A Large-Scale Study," in *Adv. Neural Inf. Process. Syst.*, 2018.

[10] O. Mogren, "C-RNN-GAN: A continuous recurrent neural network with adversarial training," in *Constr. Mach. Learn. Work. NIPS 2016*, p. 1, 2016.

[11] Z. Erickson, S. Chernova, and C. C. Kemp, "Semi-Supervised Haptic Material Recognition for Robots using Generative Adversarial Networks," *PMLR*, pp. 157–166, jul 2017.

[12] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "DeepRoad: GAN-based Metamorphic Autonomous Driving System Testing," *CoRR*, vol. abs/1802.0, feb 2018.

[13] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv Prepr. arXiv1411.1784*, 2014.

[14] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative Adversarial Networks: An Overview," *IEEE Signal Process. Mag.*, vol. 35, pp. 53–65, jan 2018.

[15] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, and X. Chen, "Improved Techniques for Training GANs," in *Adv. Neural Inf. Process. Syst. 29* (D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, eds.), pp. 2234–2242, Curran Associates, Inc., 2016.

[16] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. P. Smolley, "Least Squares Generative Adversarial Networks," in *2017 IEEE Int. Conf. Comput. Vis.*, pp. 2813–2821, IEEE, oct 2017.

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine Learning in Python," *J. Mach. Learn. Res.*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[18] R. E. Allen, A. A. Clark, J. A. Starek, and M. Pavone, "A machine learning approach for real-time reachability analysis," in *2014 IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, pp. 2202–2208, IEEE, sep 2014.

[19] V. Dumoulin, I. Belghazi, B. Poole, O. Mastropietro, A. Lamb, M. Arjovsky, and A. Courville, "Adversarially Learned Inference," *CoRR*, jun 2016.

[20] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets," in *Adv. Neural Inf. Process. Syst. 29* (D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, eds.), pp. 2172–2180, Curran Associates, Inc., 2016.