Project Brainstorming

Devesh Nath

April 1, 2025

1 Whatd been done so far and what can we use

1.1 Neural Network Verification with Branch-and-Bound for General Nonlinearities

For general non linearities, imagine sine, over an interval the linear upper and lowe bounds can be conservative. What they do is, they have a function that describes how conservative the bounds are and split if it is too conservative and keep doing that till you reach a limit of branches. So on a sine activation function, if you have a long interval, the bounds can be really conservative but it you split the crests and troughs from the part thats mostly linear, you get tighter bounds from the split. Its a bit slower but gives very tight bounds.

1.2 Learning Complex Motion Plans using Neural ODEs with Safety and Stability Guarantees

Generate Neural ODE Trajectory. No guarantees on the ODEs generated trajectory itself but guarantees disturbance rejection using CLF and safety using CBF.

1.3 BAB-ND: LONG-HORIZON MOTION PLANNING WITH BRANCH-AND-BOUND AND NEURAL DYNAMICS

Using branch and bound to get the optimal control action. Decide an action space to search, sample, lowest sample is upper bound for optimal, split, call crown, whichever space has LB above the upper bound we selected, prune it.

1.4 RAIL: Reachability-Aided Imitation Learning for Safe Policy Execution

Train an imitation learning policy. Inflate the arms geometry and move the inflated arm set using the control by the NODE. Union all the inflated arm positions, thats your reachable set. If thats in colission, switch to backup planner and plan. Do in a MPC fashion.

1.5 Safe Diffuser

Diffusion model + Barrier functions. 3 Different ways of implementing this.

1.6 DDAT: DIffusion Policies enforcing dynamically feasible trajectories

Predict trajectories using diffusion. Project them onto the the reachable set of the dynamics for dynamic feasibility.

2 Immitation Learning, Multimodaliity, Neural ODE, Reachability, Safety

2.1 Problem

Immitation learning datasets have multimodal trajectory distributions (going fro the left or the right as an example). Methods like CROWN for reachable set propagations cannot handle such multimodality because the estimate gets too conservative at decision boundaries. Tthis problem can be solved by some sort of mode collapse where the trajectories collapse to a random mode or the closest mode to the initial state.

2.2 Contractive Policy

Learning provably contractive policies like [3] is a good idea because then trajectories can collapse to a mode and be provably contractive, being in favor of reachability analysis. However this paper might not generalize well to rich distributions because of the low expressieness of the REN structure. When I initially trained the neural ODE by integrating it whole and then doing MSE, I could visually see contractive behaviour like what is in the paper. But this paper puts it into a framework to prove that it is contractive. I will try to train it the contractive way again so it falls to a mode and see if CROWN works better.

2.3 One way of dealing with multimodality

Still, all of that assures that the NODE is contractive but still there will be decision boundaries at t0. One way to change that is to have some way of randomly selecting a mode to converge to. We con do something like clustering at the start of training to cluster all the given trajecoties into different modes. Some ways of doing that clustering can be any classical method (K Means, K Shift, Agglomerative Clustering, DBSCAN etc) or a ML way (VAE, VQ-VAE, and maybe some more ways I havent thought of) and represent the cluster's modes by some variable z. Once they are clustered, we can use that during training. During training we sample a z, add that as a param to the NODE, rollout the NODE and minimize the error to any trajectory in the mode z. During inference, we randomly sample z from our codebook of trajectories and all initial points in the state space go towards one distribution.

2.4 Learned Barrier Functions from depth info

Since our planner is working in a space with identity dynamics, the simple learned barrier function that I was proposing can give us safe plans, adapting to the depth cloud at test time. However, we would just have a plan but no dynamic feasibility guarantees or enforcement. If we can form a feasible optimization problem where we optimize for both the dynamic feasibility and satisfying the CBF then that would be awesome. Something like:

$$\begin{aligned} \text{minimize}_{\{u_t\}_{t=0}^{T}} \quad & \sum_{t=0}^{T} \|\dot{x}_t - v_\theta(x_t, t)\|^2 \\ \text{subject to} \quad & \dot{x}_t = f(x_t) + g(x_t)u_t \quad \text{(System dynamics)} \\ & \frac{\partial h(x_t)}{\partial x} \cdot \dot{x}_t + \alpha(h(x_t)) \geq 0 \quad \text{(CBF condition)} \\ & u_t \in \mathbb{U} \quad \text{(Control constraints)} \end{aligned}$$

2.5 Dynamically feasible Neural ODEs

Currently we our neural ODE is:

$$f_\theta(x, t) = \frac{dx}{dt}$$

We are training the neural ODE by fitting it to $x_{t+1} - x_t = \dot{x}$. Since all the trajectories we collect are already (since we are collecting them on the orignal system) dynamically feasible, the measure for how dynamically feasible our generated trajectory is just the model error. We can get concrete bounds on the error by using the trusted domain construction using the method in Glen's paper. But where would that be helpful?
Problems we can solve (at least those that I can think of):

1. If the trajectory was collected on another system then we would want to make our neural ODE dynamically feasible. Lets say we had a smaller robot arm to collect trajectories and a larger arm to replicate it on. Then finding a way to ensure dynamical feasibility makes sense.

2. $\|f(x, u) - g(x, u)\| < \epsilon$ where f is the real system and g is the neural ODE dynamics. Can we do something like $f(x, u) < g(x, u) + \epsilon$ where after the first step we add ϵ to $g(x, u)$ and get the interval (I would assume overapproximated) where the true dynamics lie. Lets say we also have a controller $u^\star(x)$ in the next step we pass the interval of x to $g(x, u)$ to get the reachable set and add ϵ to it to construct another interval? What would that be useful for? Propagating the probable interval where the true dynamics lie?

3. After meeting with Glen: It is also important to enforce dynamic feasibility outside of the data points the neural ODE has seen during training, and that's where a dynamics/physics-informed loss would come in. Dually train the NODE and the controller. At all intervals of x, evaluate the upper bound on $\|f(x) - g(x)u - NODE(x)\|$ to ensure bounded error across the state space. Use a contraction-based controller around $u^\star$, then use $x^\star$ and $u^\star$ to get the reachable set around the nominal trajectory.

3 Dynamically feasible CFM Trajectories

Neural ODE learned dynamics:

$$\dot{x} = f(x) + g(x)u$$

where $u \in \mathbb{U}$ which is the allowable set of controls.
CFM gives us statistically accurate trajectories so they are not dynamically feasible but look close to the target distribution.
We can formulate this into an optimization problem to project the trajectories to a feasible one.

$$\begin{aligned} \text{minimize}_{\{u_t\}_{t=0}^{T}} \quad & \sum_{t=0}^{T} \alpha \|x_{t+1} - x_{d,t+1}\|^2 + \beta \|u_t\|^2 \\ \text{subject to} \quad & x_{t+1} = x_t + dt * (f(x_t) + g(x_t)u_t) \\ & u_t \in \mathbb{U} \\ & x_0 = x_{init} \end{aligned}$$

$x_{d,t}$ is the desired trajectory, output of the CFM model after integration.

3.1 Barrier Functions

Since diffusion/CFM dynamics are similar, $\dot{x} = v_\theta(x,t)$ and very simple, we can treat the dynamics as $\dot{x} = u$. Because of this simplification we can use barrier functions that have the following properties:

- $h(x) \geq 0$ for all $x \in \mathbb{S}$ where $\mathbb{S}$ is the state space.
- $h(x) \leq 0$ for all $x \in \mathbb{S}^c$ where $\mathbb{S}^c$ is the set of states that are not allowed.
- $h(x)$ is continuously differentiable.
- $\frac{dh(x_i)}{dx}.v_{\theta,i}(x,t) + \alpha(h(x_i)) \geq 0$ is a relatively simple QP to solve because of the identity dynamics of CFM. Here t is the CFM time 0 -¿ 1 and i is the planning time.

Such a barrier function should be easily learnable straight from the input pointcloud data. We can split the FOV of the depth camera and split it into a voxel grid of N points. All the voxels that have objects can be labeled as **unsafe**. This includes everything we cant observe from the camera. Every other voxel can be labeled as **safe**.
We can then train an unsupervised model to learn the barrier function. All the voxels can be treated as intevrals to CROWN. The loss function can be formulated like this:

$$\mathcal{L} = \sum_{(\underline{x},\overline{x})\in\mathbb{S}} max(0, -E_l(h(x), \underline{x}, \overline{x})) + \sum_{(\underline{x},\overline{x})\in\mathbb{S}^c} -min(0, -E_u(h(x), \underline{x}, \overline{x}))$$

where $E_l(h(x), \underline{x}, \overline{x}) = \text{CROWN}(h(x), \underline{x}, \overline{x})[0]$ and $E_u(h(x), \underline{x}, \overline{x}) = \text{CROWN}(h(x), \underline{x}, \overline{x})[1]$ are the scalar lower and upper bounds of the intervals returned by CROWN.
This trains h(x) to be a barrier function from the observed data. Theres one thing we need to consider for this training to be feasible. Since we are evaluating boxes and enforcing them

to be positive and negative even when they are beside each other, their boundaries will be discontinuous. To mitigate this, we can create sort of a buffer between the safe and the unsafe regions where you can penalize the lipschitz continuity of the barrier function. This can be done by adding a term to the loss function that penalizes the Lipschitz continuity of the barrier function to ensure continuity and we would also have to use tanh or a continuous activation function that compatible with CROWN.

3.2 Guarantees on the learned dynamics

We can use Glen's papers method to get a bound on the error of the learned dynamics. We can incorporate that in the optimization problem. That is if we can do something like this $f(x, u) < g(x, u) + \epsilon$ where we know x_0. So at x_{t+1} we can get an interval $\underline{x_{t+1}} \leq x_{t+1} \leq \overline{x_{t+1}}$ where $\underline{x_{t+1}} = x_t + dt * (f(x_t) + g(x_t)u_t) - \epsilon$ and $\overline{x_{t+1}} = x_t + dt * (f(x_t) + g(x_t)u_t) + \epsilon$. The true state is in the interval.
Not sure how this is helpful but just putting it here for the sake of brainstorming.

4 References

1. Glen's paper on trusted domain construction for bounding errors in learned dynamics: *Glen et al., ”Trusted Domain Construction for Neural ODEs”, 2021.*

2. CROWN for reachable set propagation: *Zhang et al., ”Efficient Neural Network Verification with CROWN”, 2018.*

3. Contractive Dynamical Imitation Policies: *Richards et al., ”Contractive Dynamical Imitation Policies for Efficient Out-of-Sample Recovery”, 2020.*