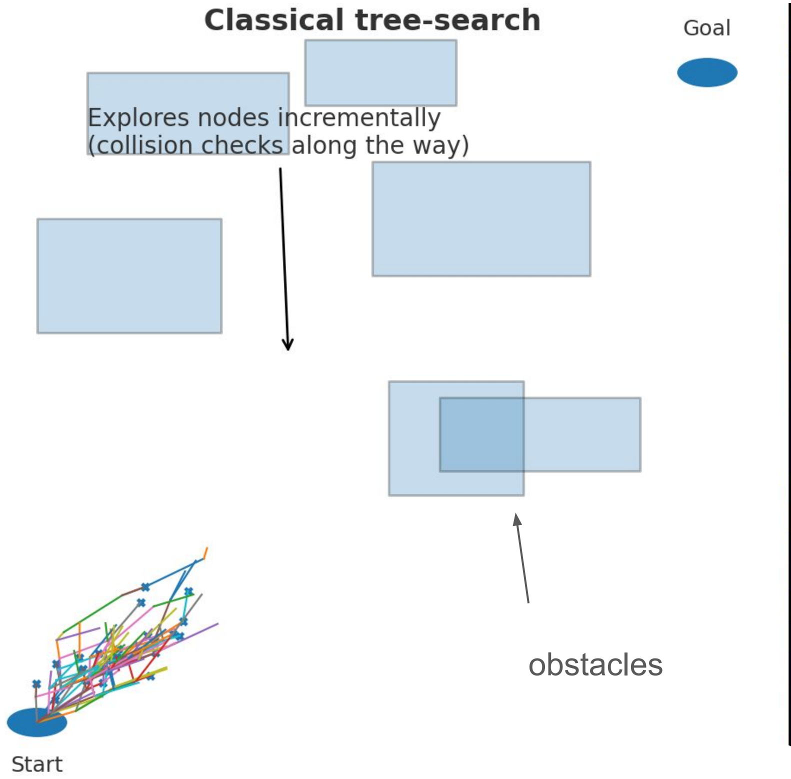
How fast can we plan it?

Sakib Chowdhury

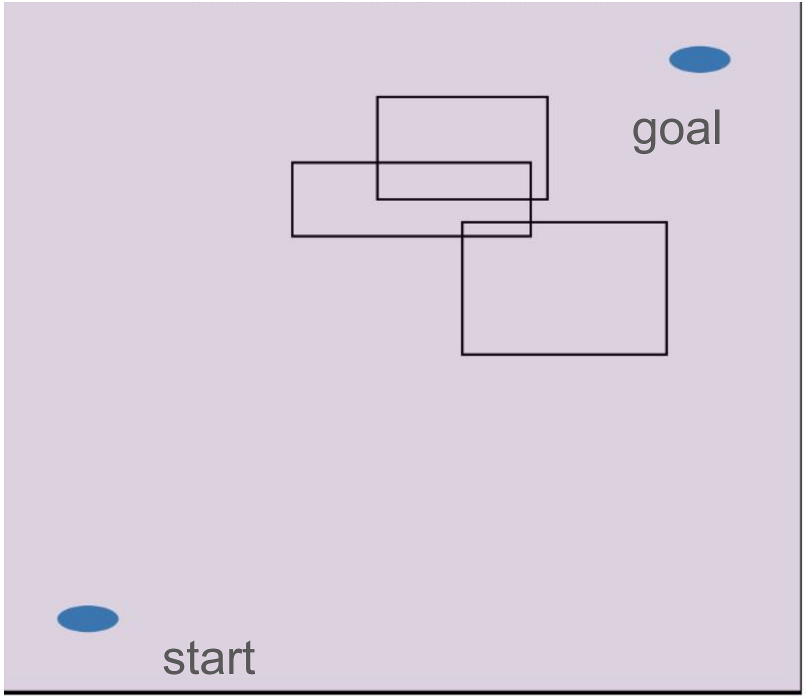
Motivation

- **Athletic intelligence is time-sensitive** — decisions must happen under tight reaction times.
- **Allocate more time to action, less to planning** — long planning delays cause missed opportunities.
- **Classical planners (tree-search) scale poorly** — bigger search spaces → longer planning times.
- **RRT-style planners explore incrementally** — extend nodes and collision-check; they can't "see" the whole scene at once.



Motivation

- **CNN (Convolutional Neural Networks)** based planners could view the entire environment as a matrix in one pass — less exploration overhead, faster decisions.
- **Supervised Imitation Learning** could learn the relation between trajectory and the environment matrix from expert data (in this case: a classical planner)



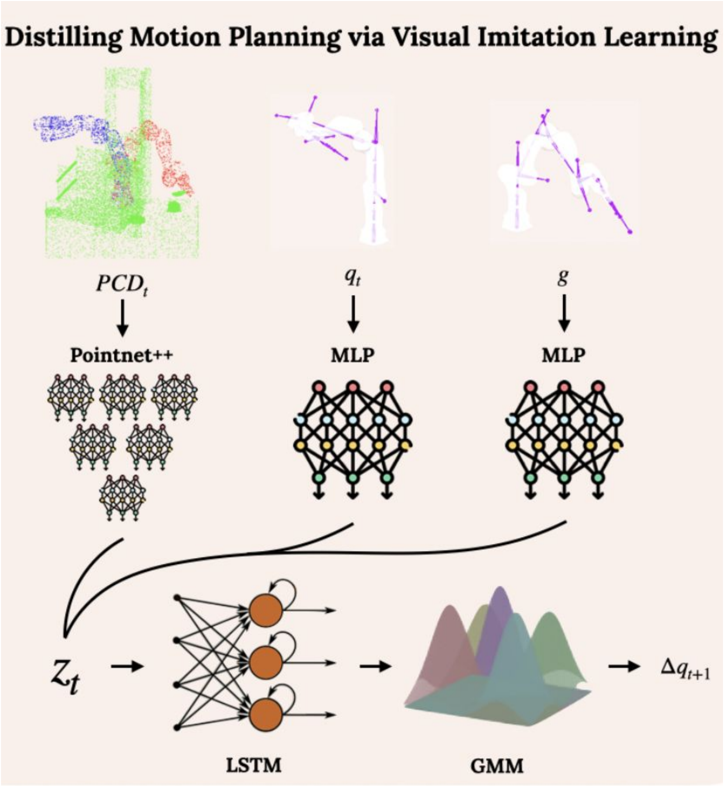
Recent work

***Neural MP: A Generalist Neural Motion Planner (Best Paper Award Winner)**

- Has shown that supervised imitation learning methods can learn from classical planners (e.g. AIT*)
- Uses point cloud as environment representation and PointNet++ to get a view of the environment.
- The use of PointNet++ shows that modeling the environment is possible with Neural Networks.

Limitations:

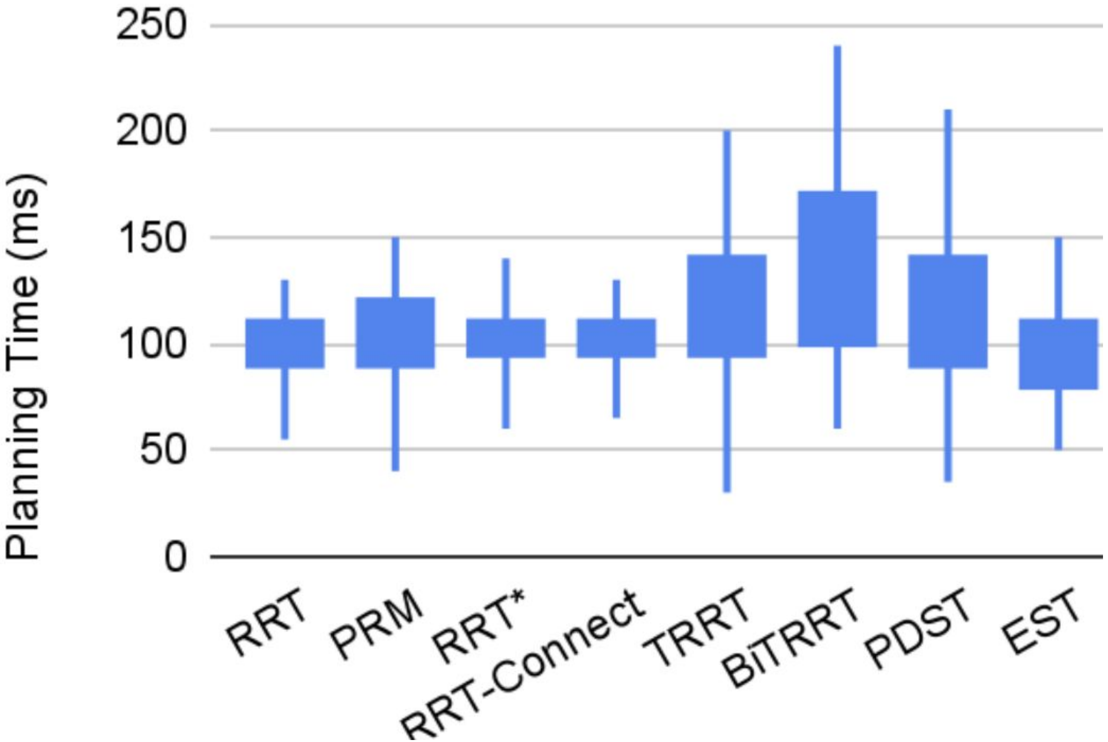
- Learns/Predicts the increment in joint positions at each time step (so still similar to incremental search of classical planners). Takes longer time to plan longer trajectories.



***https://mihdalal.github.io/neuralmotionplanner/**

Choosing fastest planner to generate expert data

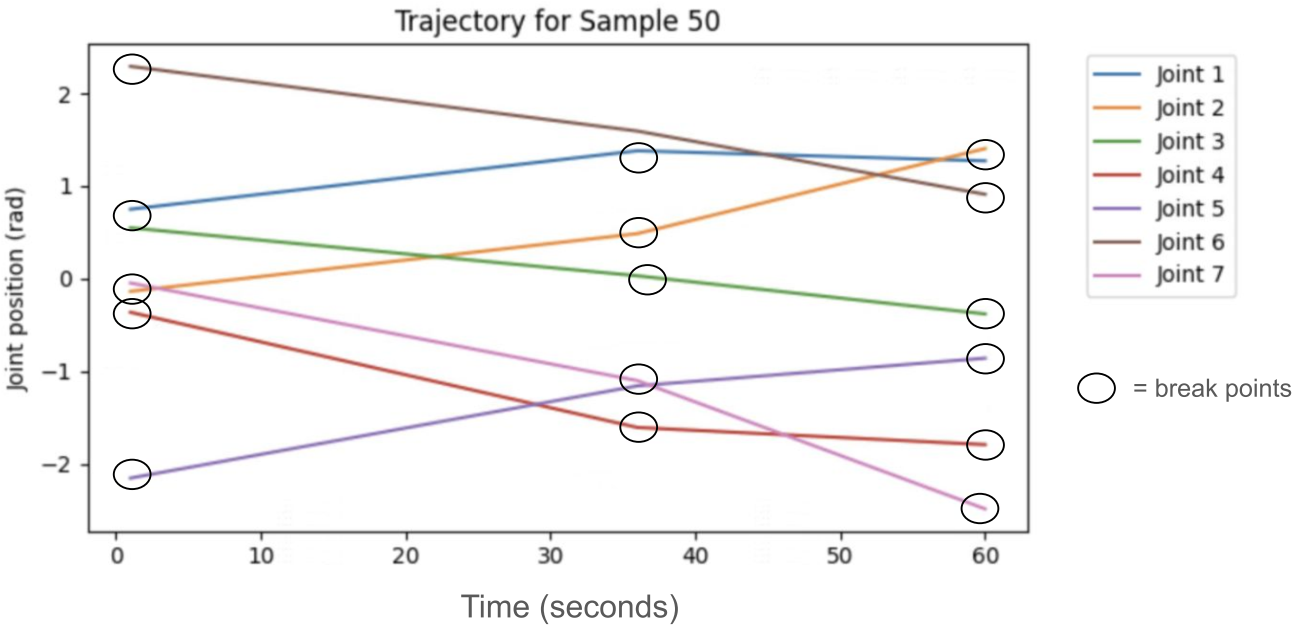
- “RRT Connect” shows the fastest planning behavior.
- We choose “RRT Connect” to generate expert data.



Typical planned motion of RRT connect

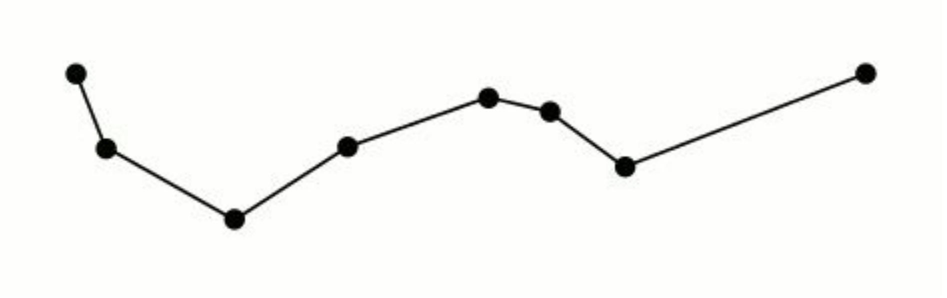
Observation:

- Planned motion is piecewise linear functions
- the break points are sufficient to reproduce the motion.



Ramer–Douglas–Peucker (RDP) Algorithm

- An algorithm that decimates a curve composed of line segments to a similar curve with fewer points.
- Simplifies a piecewise linear curve.
- We use RDP to simplify the RRT Connect motions.



https://en.wikipedia.org/wiki/Ramer%E2%80%93Douglas%E2%80%93Peucker_algorithm

Data Collection

Process:

- Randomly place obstacles in environment
- Randomly choose valid starting and goal states
- Use OMPL (Open Motion Planning Library) to plan motion using RRT-Connect
- Use RDP to simplify the motion representation
- Some paths are not feasible. Store them as infeasible motion.

Dataset Size:

- 100,000 samples
    - Training samples: 80, 000
    - Validation samples: 20, 000

Our Method

Context

- Start State (θ_s^j)
- Goal State (θ_g^j)
- Obstacle (size, 3D Cartesian position, orientation)

Occupancy Map

Our Planner
(Context + Map) -> trajectory

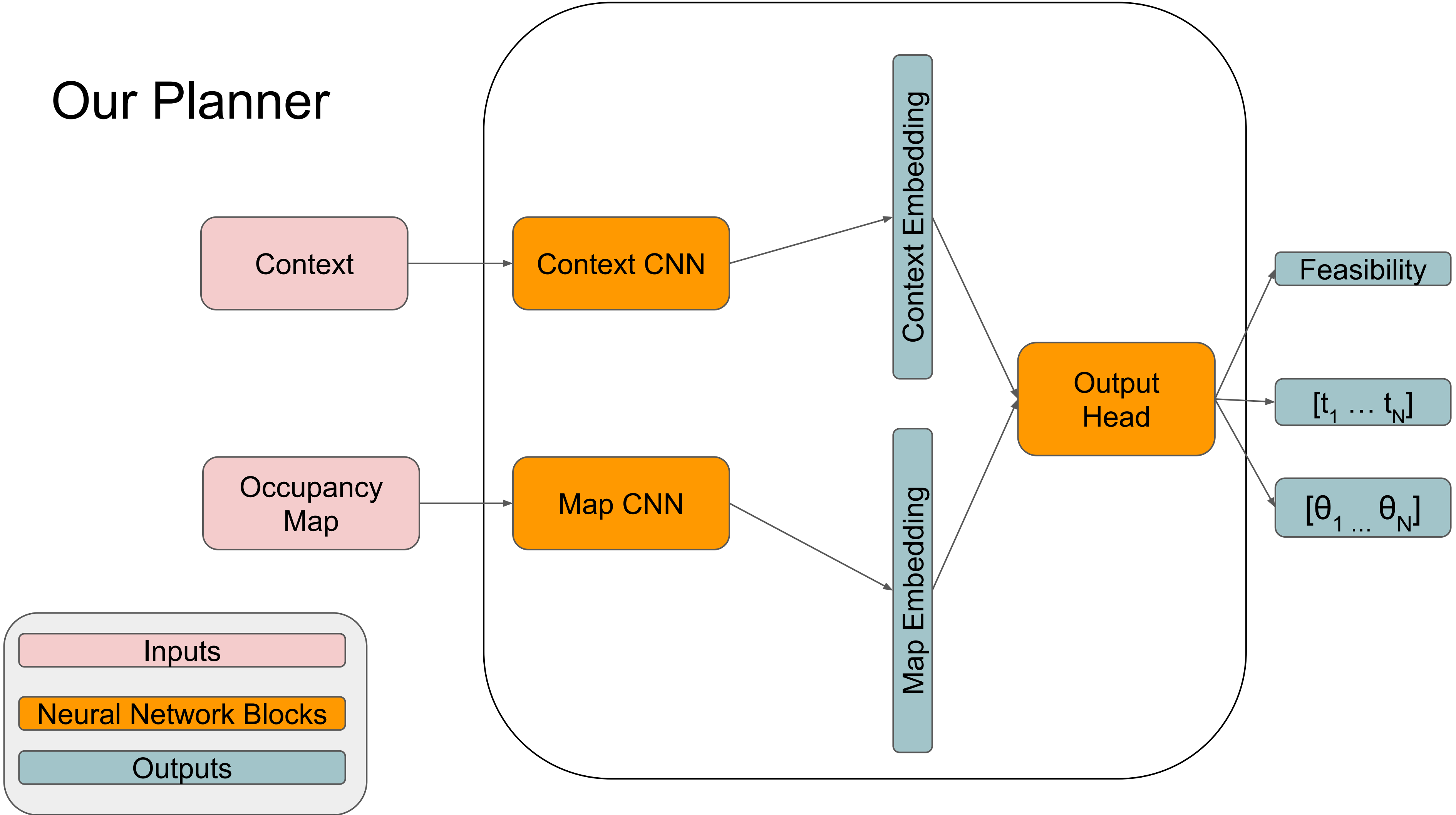
Motion planned in single inference (not incrementally). Means faster planning.

t = time
θ = joint position
j = joint indices of the robot.
For a 7 DOF franka emika panda robotic arm, $j \in \mathbb{N} = [1,7]$
ω = joint velocity = θ'

Full Motion
$\{(t_1, \theta^j_1), (t_2, \theta^j_2), (t_3, \theta^j_3), \dots (t_N, \theta^j_N)\}$,

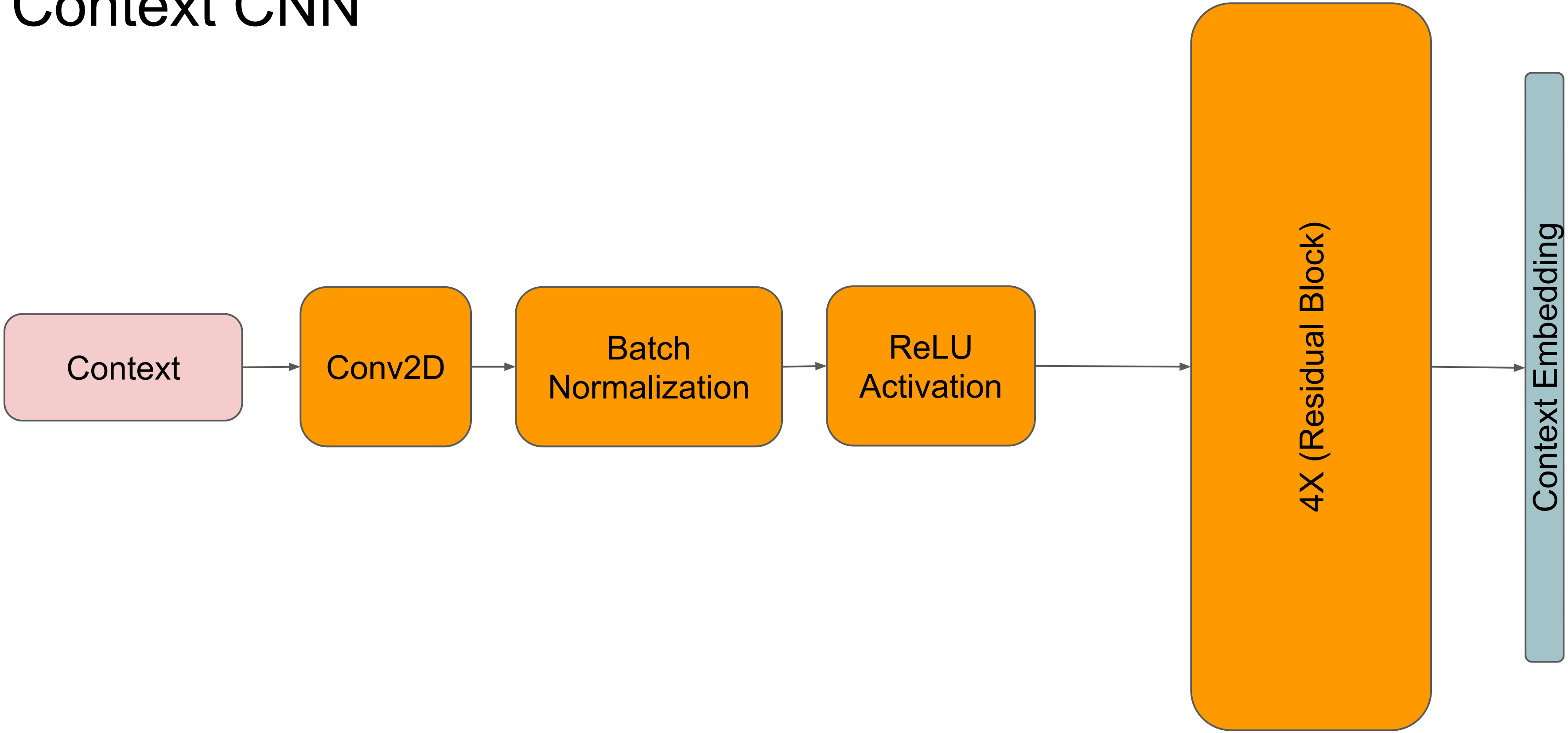
j = Joint index
N = maximum number of break-points

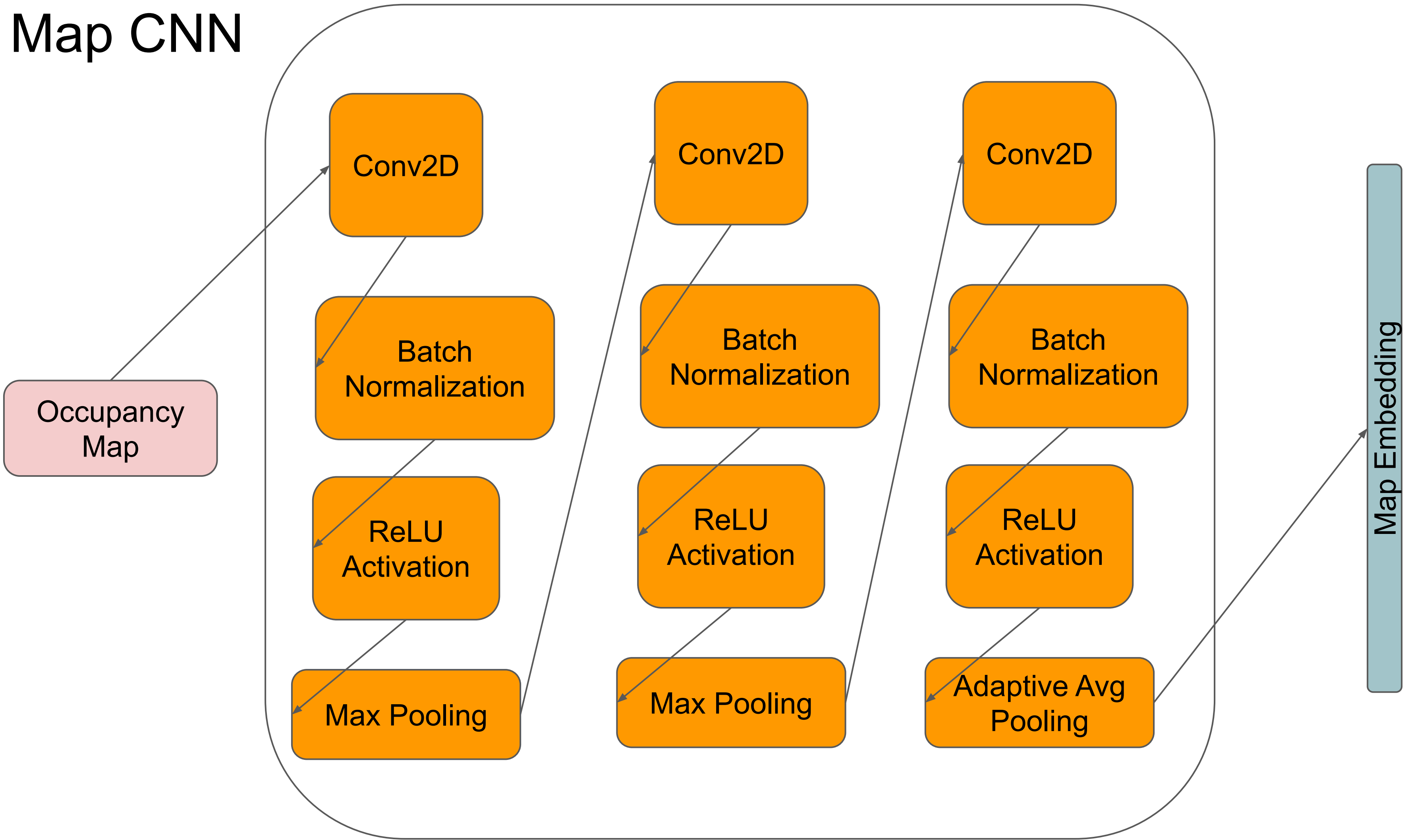
Our Planner



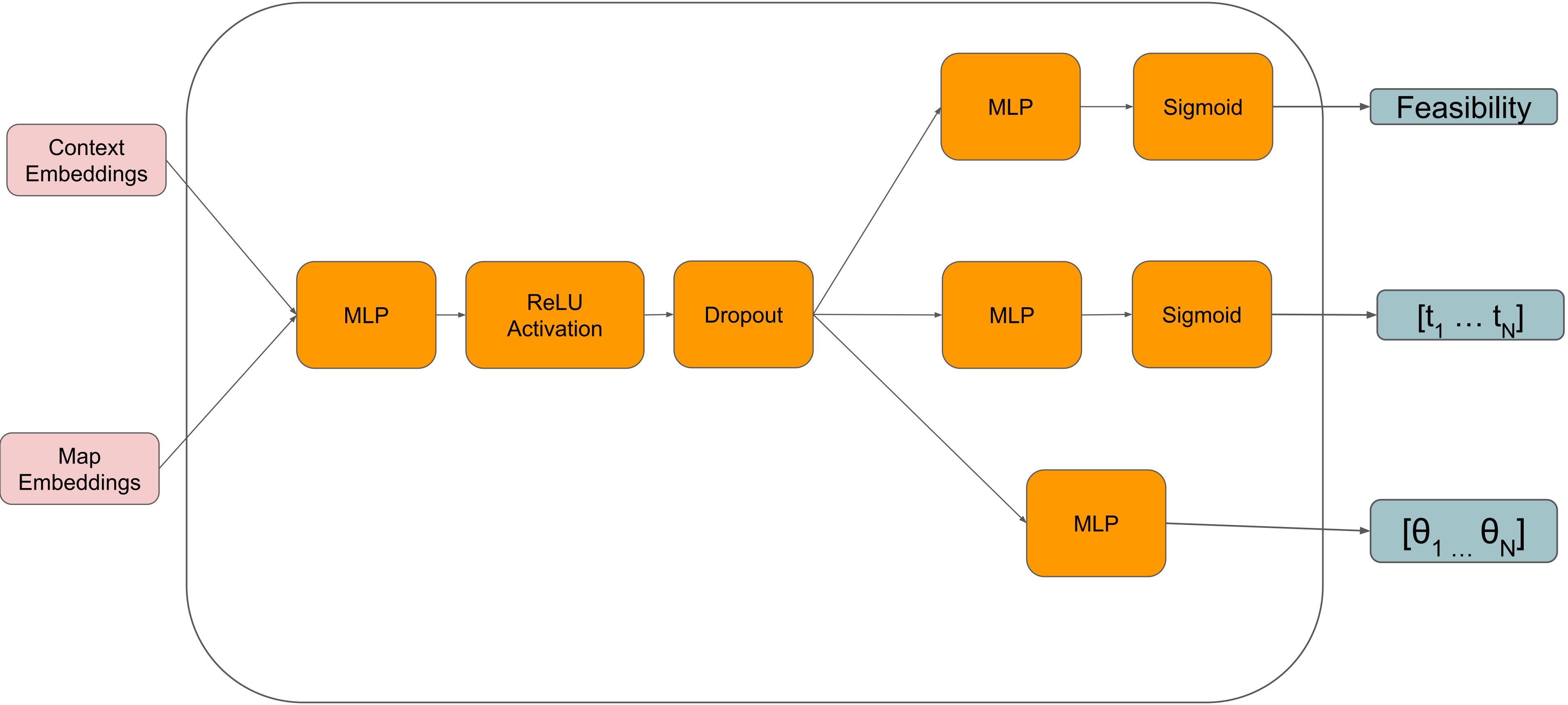
Context
Occupancy Map
Context CNN
Map CNN
Context Embedding
Map Embedding
Output Head
Feasibility
[t_1 … t_N]
[θ_1 … θ_N]
Inputs
Neural Network Blocks
Outputs

Context CNN


Context
Conv2D
Batch Normalization
ReLU Activation
4X (Residual Block)
Context Embedding


Map CNN
Occupancy Map
Conv2D
Batch Normalization
ReLU Activation
Max Pooling
Conv2D
Batch Normalization
ReLU Activation
Max Pooling
Conv2D
Batch Normalization
ReLU Activation
Adaptive Avg Pooling
Map Embedding

Output Head


Context Embeddings
Map Embeddings
MLP
ReLU Activation
Dropout
MLP
Sigmoid
Feasibility
MLP
Sigmoid
[t_1 … t_N]
MLP
[$θ_1$ … $θ_N$]

Loss Function

- **We took a weighted loss approach:**

$$loss_{\alpha} = w_f * loss_f + w_t * loss_t + w_{\theta} * loss_{\theta} + w_g * loss_g + w_{d\theta} * loss_{d\theta}$$

$loss_f$ = Binary Cross Entropy ($true_f$, $pred_f$) → error in feasibility

$loss_t$ = MSE ($true_t$, $pred_t$) → error in time values

$loss_{\theta}$ = MSE ($true_{\theta}$, $pred_{\theta}$) → error in θ values

$loss_g$ = MSE ($true_g$, $pred_g$) → error in the goal point (ensures the motion reaches goal)

d_{θ} = first derivative of θ

$loss_{d\theta}$ = MSE($true_{d\theta}$, $pred_{d\theta}$) → reduces abrupt transitions between breakpoints

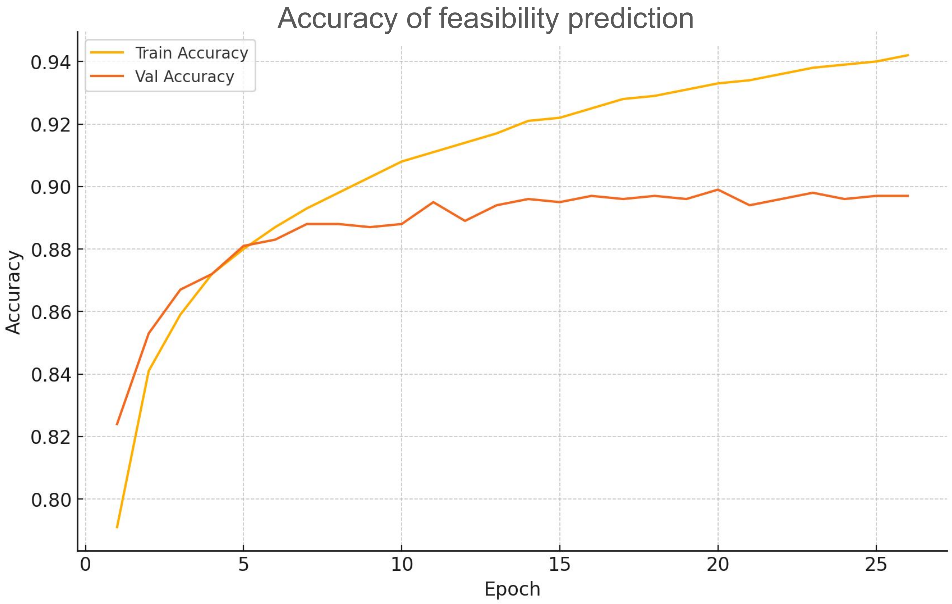
w = weight

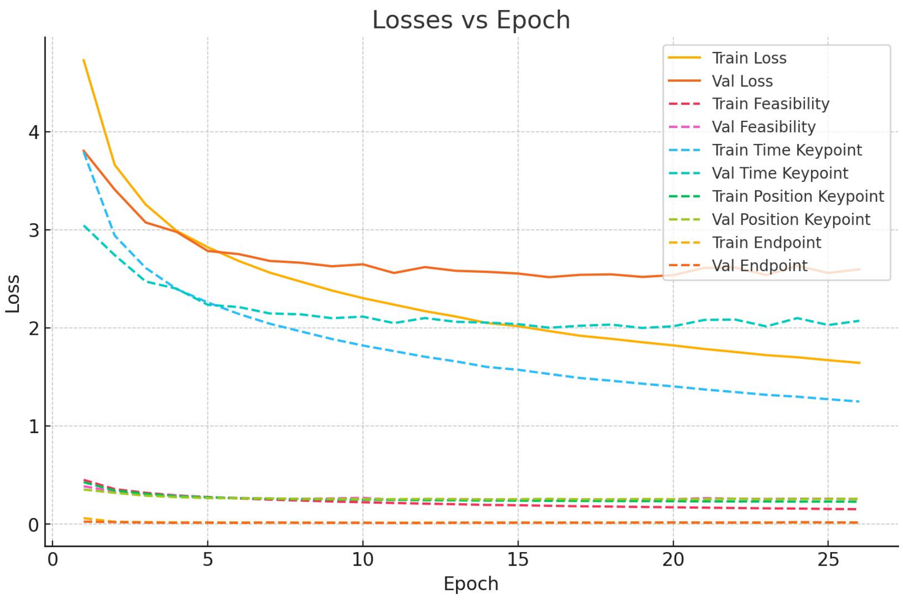
We use Adam optimizer to minimize the total $loss_{\alpha}$

$w_f, w_t, w_g = 1$

$w_{\theta}, w_{d\theta} = 5$

Training Performance



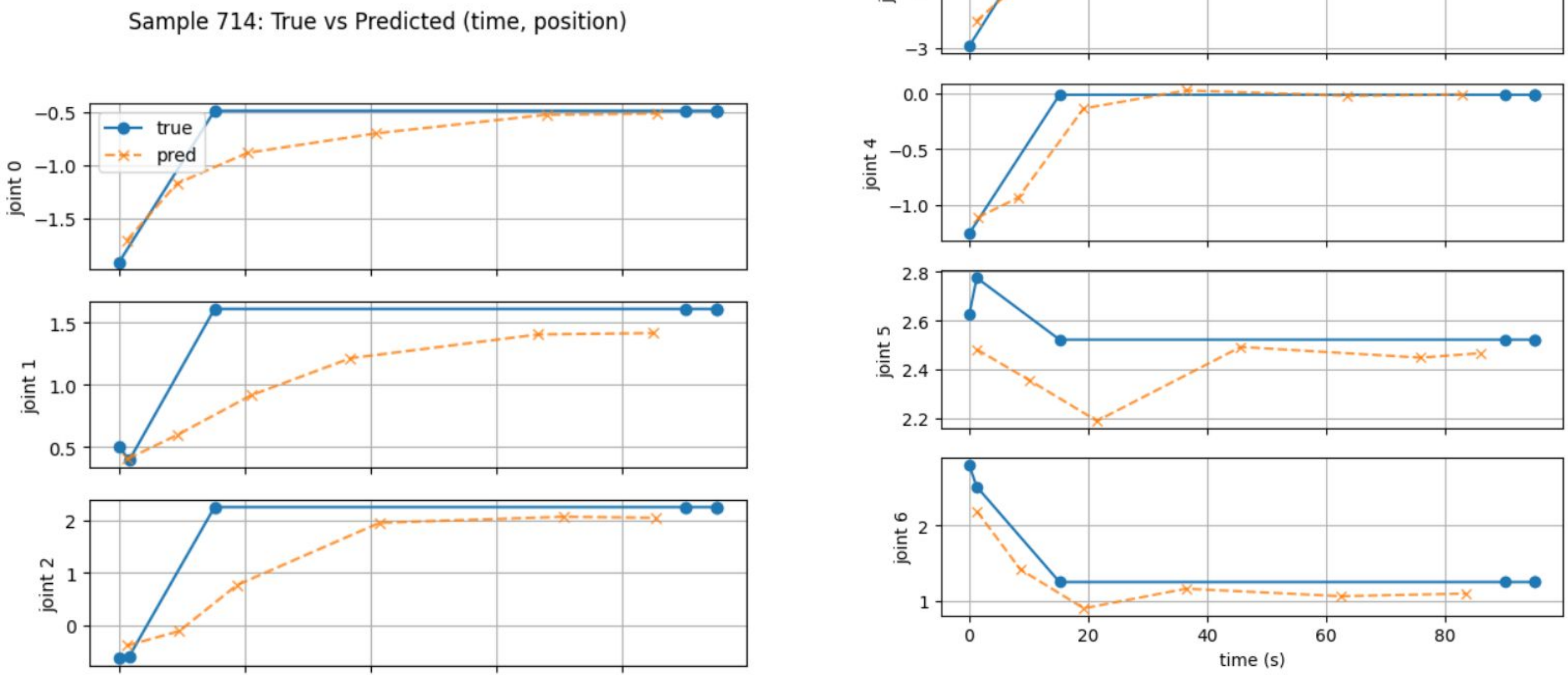


Trained on NVIDIA A6000 GPU

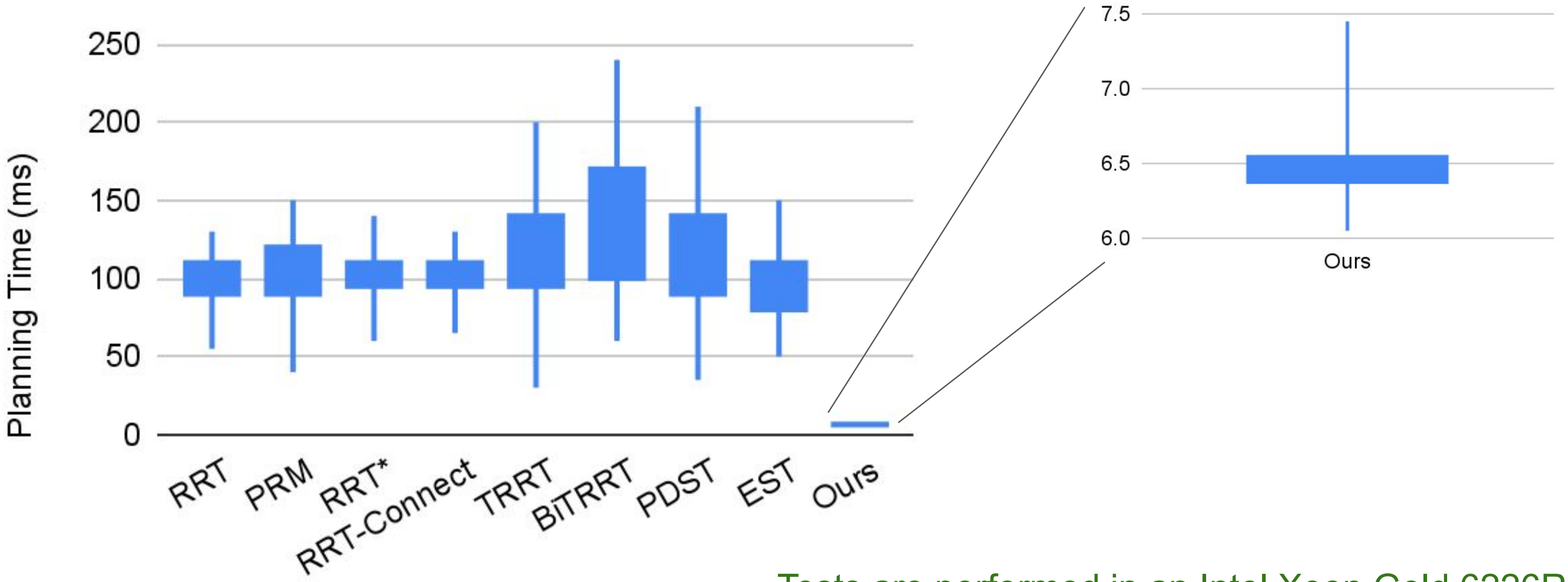
Finetuning with collision-aware loss

- The trained model collides with obstacles very often. To improve it, we finetune it with collision-aware loss for 5,000 episodes.
- Procedure:
  - Randomly choose samples from original training data
  - Execute the motion in a real simulator (Pybullet).
  - Check if collision happens.
  - Calculate $loss_{\beta}$ and optimize with Adam optimizer.
- $loss_{\beta} = loss_{\alpha} + w_c$ * collision_rate (we choose $w_c = 10$)
- collision_rate = C / T
- C = number of collisions in the episode
- T = number of simulation time steps in the episode
- Why not use collision-aware loss in the original training? ->
  - This loss is dependent on live simulation. takes longer time compute.
  - Using it in the previous training step will make the training much longer.

Sample Inference



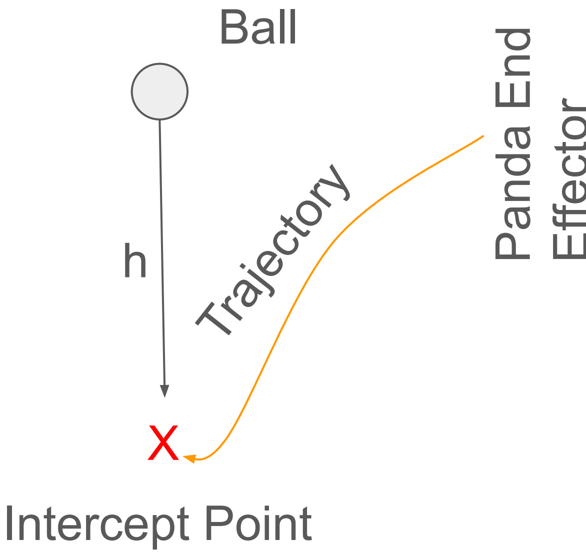
Planning Time Comparison



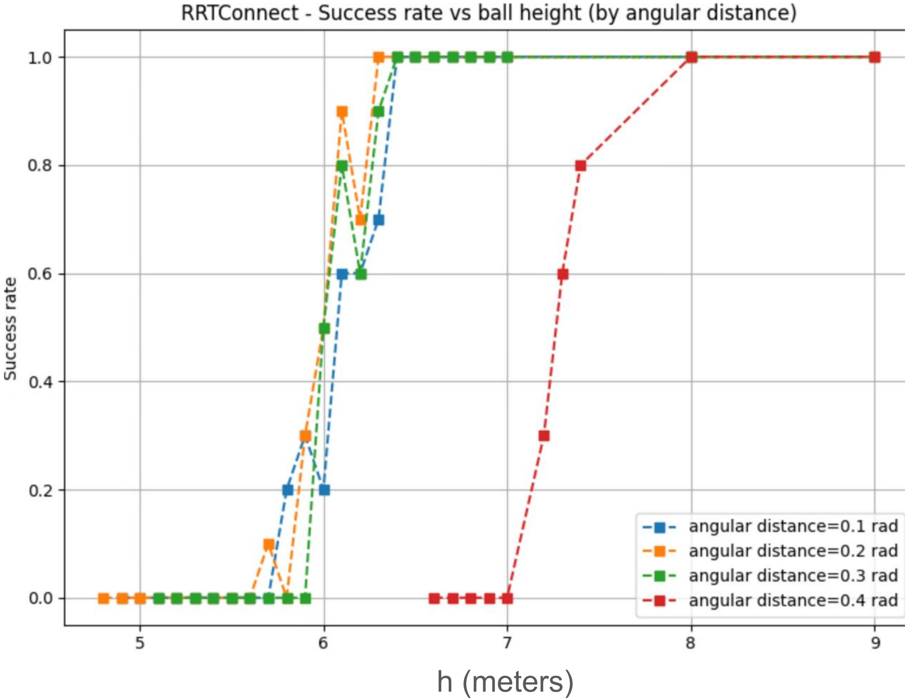
Tests are performed in an Intel Xeon Gold 6226R CPU

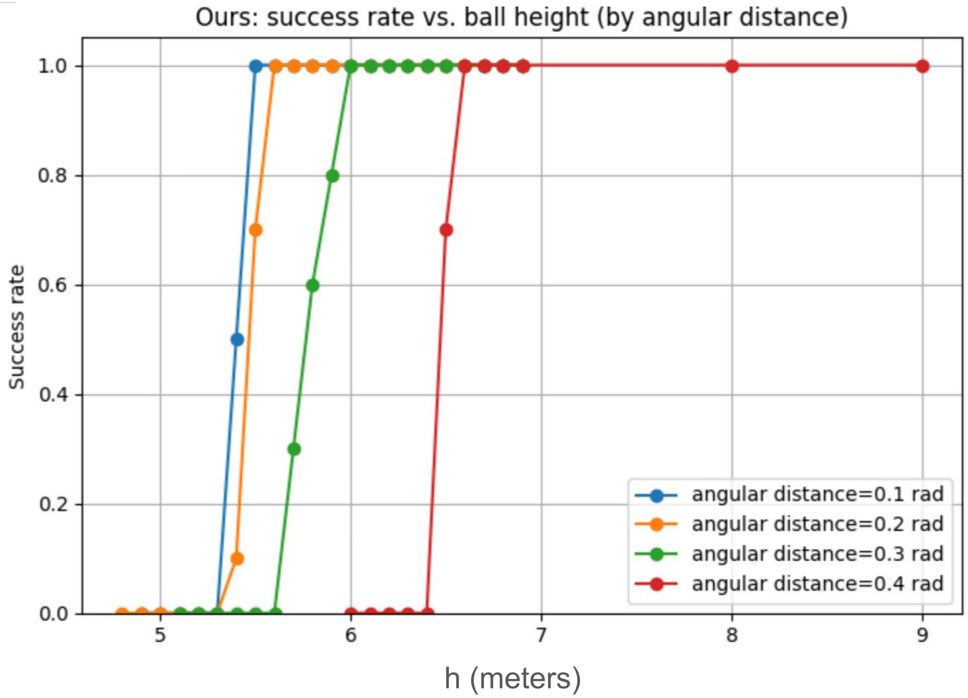
Ball Drop Test (in Simulation)

- We drop a ball from h height
- the ball takes t = sqrt(2*h/g) time to reach the intercept point
- Panda gets t amount of time to plan and execute the motion.
- The smallest "h" beyond which panda cannot intercept the ball, is "critical h".
- Slower planning means larger "critical h".
- Faster planning means smaller "critical h"
- We check successful interception rate to find the "critical h".



Results

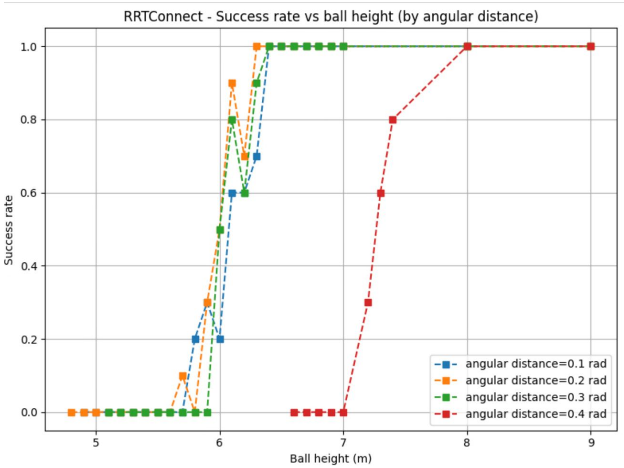


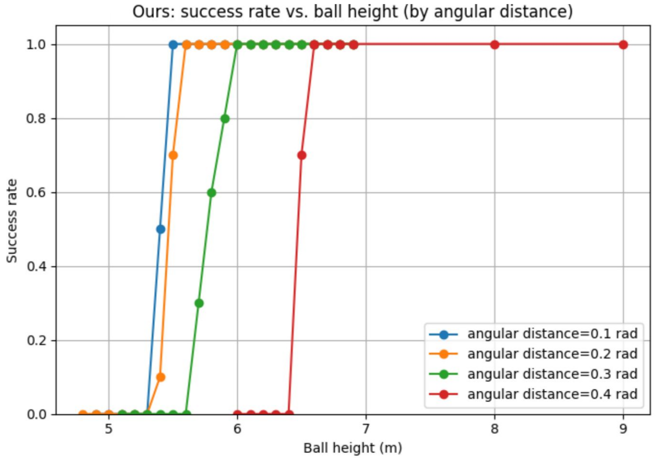


angular_distance is the distance between the start and intercept position for the panda robotic arm with respect to its base axis

Observation

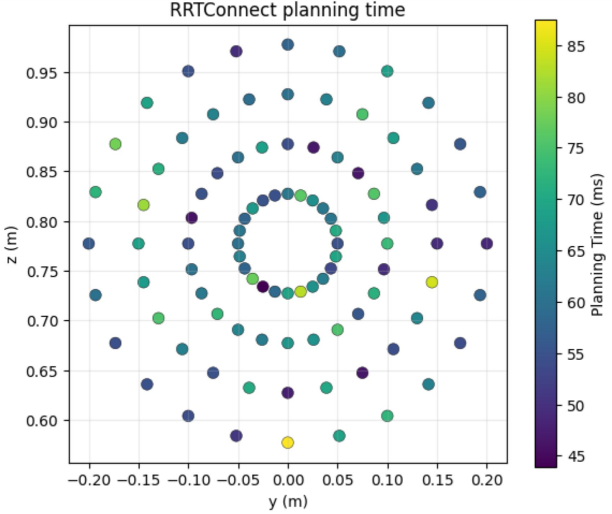
- RRT Connect has higher “critical h” (point beyond which panda cannot intercept the ball)
- Ours has smaller “critical h”.
- Our planning is much faster for time sensitive tasks compared to the fastest classical motion planner (in our list).
- Our curve is smoother and more reliable compared to RRT Connect

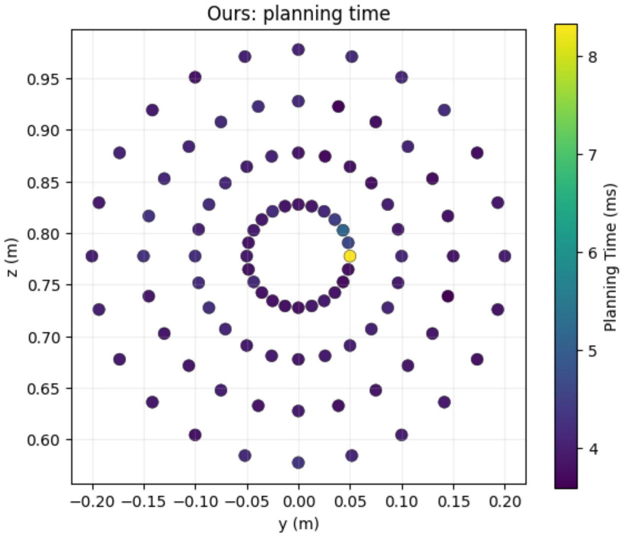




Spatial Dependence of Planning Time

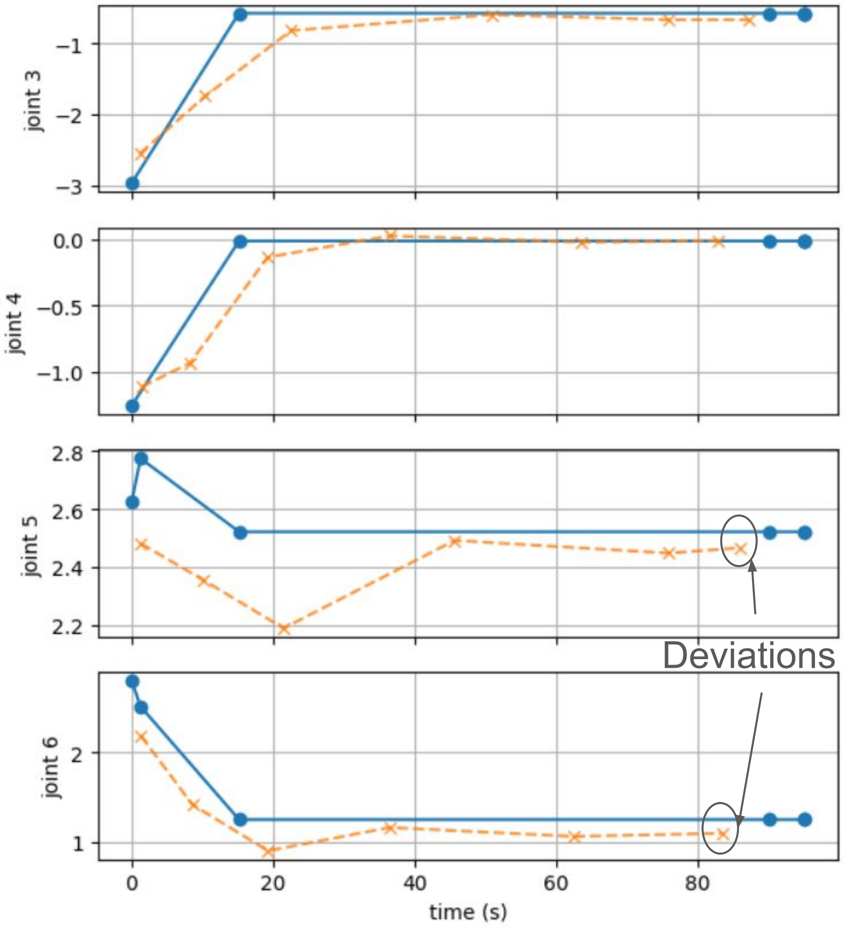
- The center of the circles is the start location of the end effector.
- Each point in the circle is goal location at the vertical plane around the home location
- RRT Connect planning time is inconsistent. Depends on the distance between start and goal state and number of obstacles in between.
- Ours is fairly consistent and shorter as our model looks into the environment at a single shot.





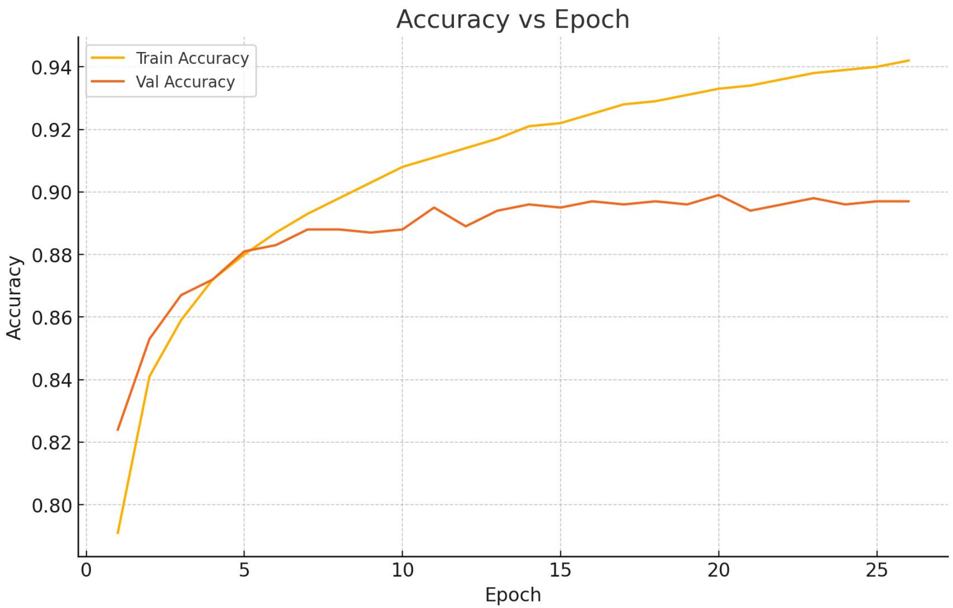
Limitations

- Our predicted motion does not land exactly at the goal state (small deviation is observed
- We also observe a small deviation from the starting location.
- In some critical scenarios, where the trajectory is very likely to hit an obstacle - our model predicts “feasible motion”.



Limitations

- The feasibility prediction is approximately 90% accurate.
- In 1 out of 10 cases, it inaccurately predicts whether a valid trajectory is possible between the start and the goal state.



Thanks

Kind and constructive suggestions are appreciated.