

ROSCon 2024

# A Fuzzy-Matching Trajectory Cache for MoveIt 2

**Brandon Ong**

[github.com/methylDragon](https://github.com/methylDragon)

SWE. Open Robotics @ Intrinsic

some stuff I've done:

core ROS dev

REP-2011

cloud robotics

cool memes, and boardgames

# The Problem

# The Problem

In an industrial context:

“How can we get **manipulator trajectories quickly and deterministically?**”

# The Problem

In an industrial context:

“How can we get **manipulator trajectories**  
**quickly** and **deterministically?**”

cycle-time → \$\$\$

# The Problem

In an industrial context:

“How can we get **manipulator trajectories**  
**quickly** and **deterministically**?”

cycle-time → \$\$\$

reliability,  
repeatability,  
safety

# The Problem

In an industrial context:

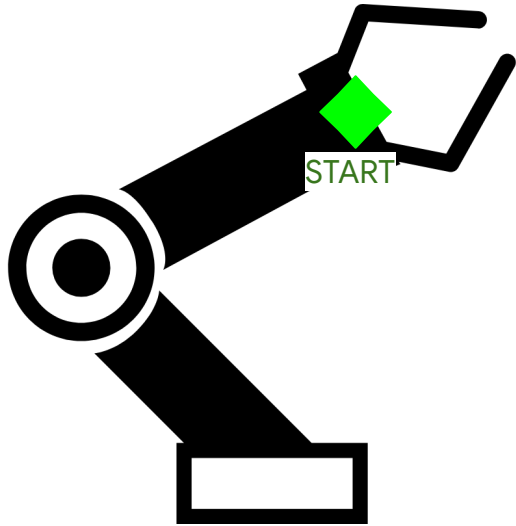
“How can we get **manipulator trajectories** **quickly** and **deterministically**?”

Currently to get this in MoveIt 2,  
you would have to **manually label and save pre-planned trajectories,**  
and then **manually reuse them**

**Very labor intensive!**

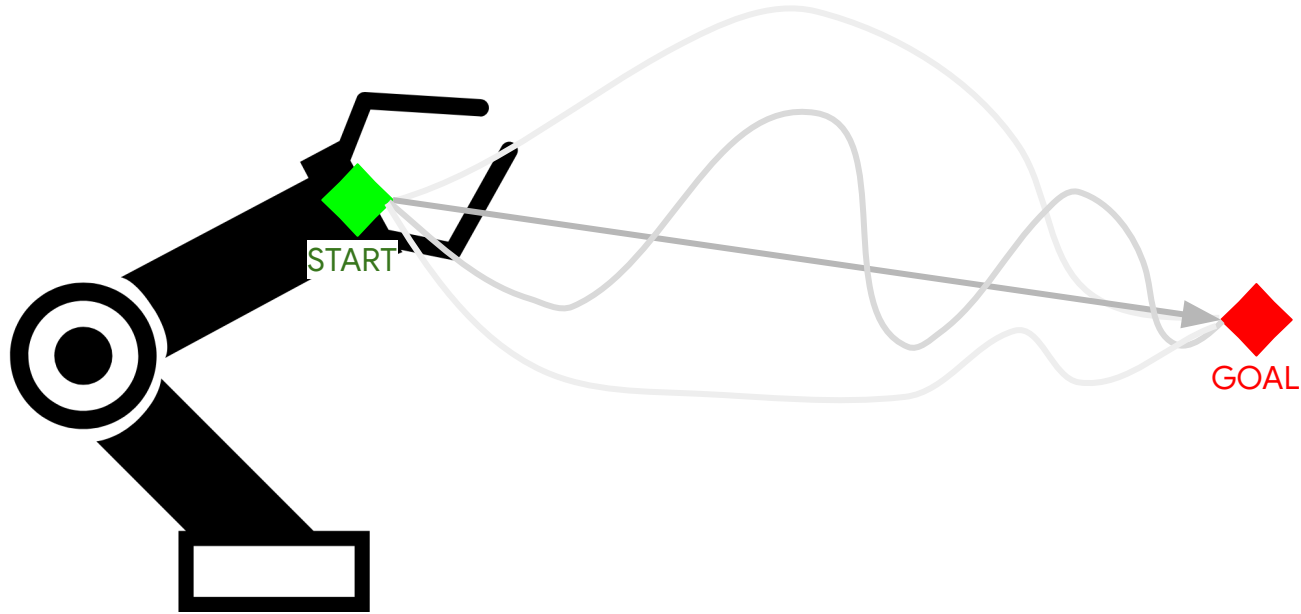
**Context**

For a given **manipulator**,  
There exists **multiple paths** from **start** to **goal**

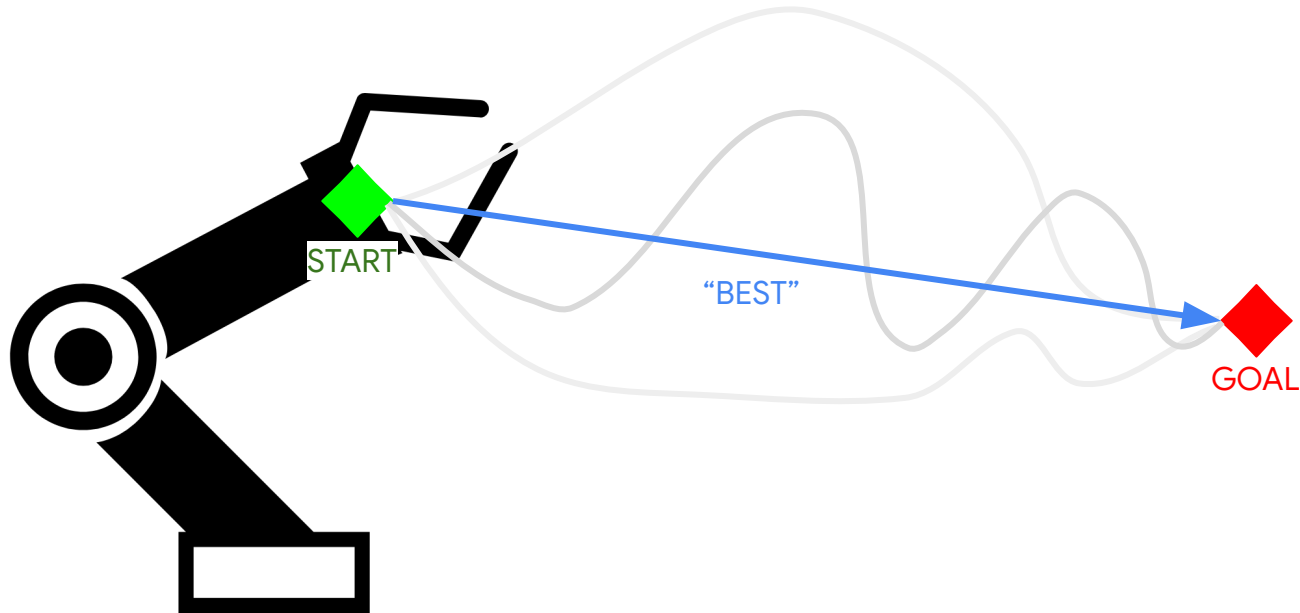




For a given **manipulator**,  
There exists **multiple paths** from **start** to **goal**



For a given **manipulator**,  
There exists **multiple paths** from **start** to **goal**  
**WITH VARYING DEGREES OF OPTIMALITY**



# And the two main planning strategies have opposing pros and cons

## Sampling-Based

## Optimization-Based

Fast



Slow

Random



Deterministic

Rarely finds “more optimal” paths



Finds “more optimal” paths

# A cache lets you reap all benefits

Sampling-Based

Optimization-Based

**Fast**



Slow

Random



**Deterministic**

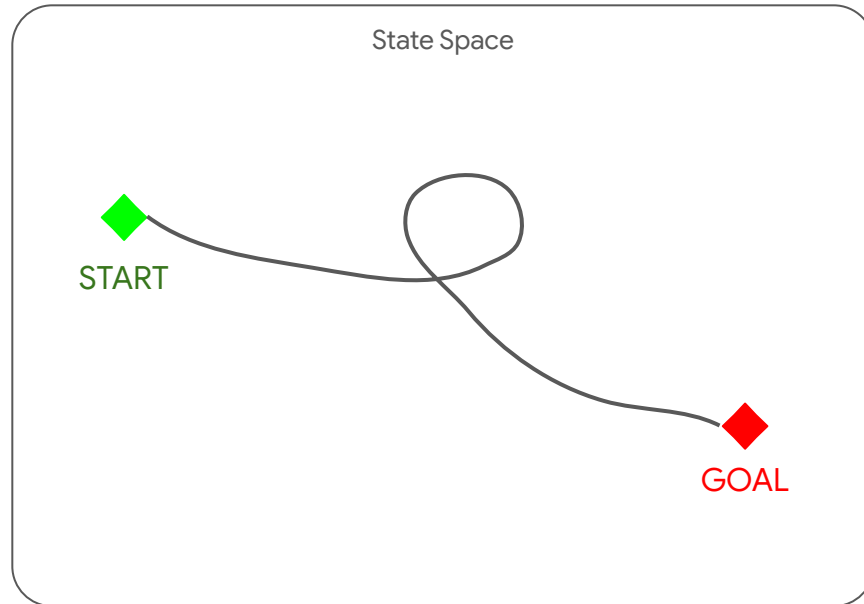
Rarely finds “more optimal” paths



**Finds “more optimal” paths**

# The Core Idea

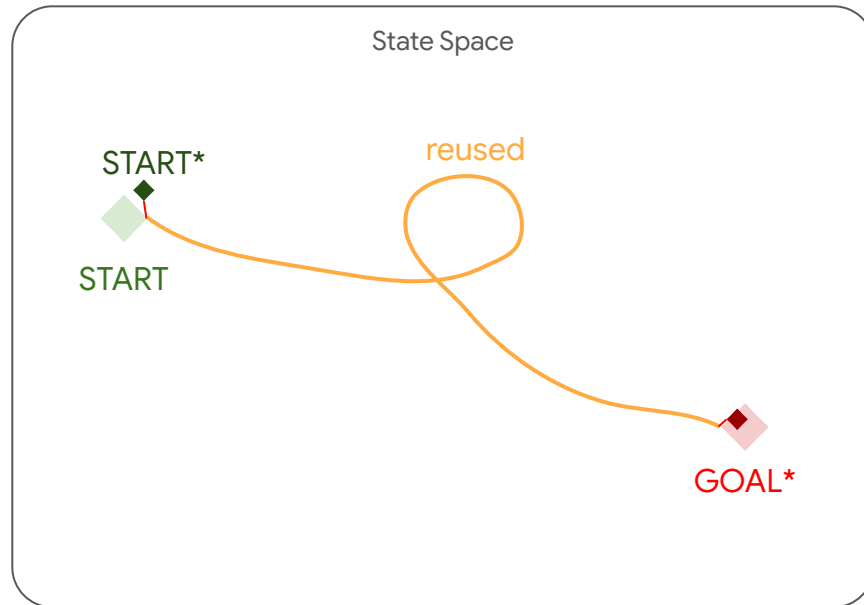
If you've already **cached** a path from some **start to goal**, under some **set of constraints**



# The Core Idea

If you've already **cached** a path from some **start to goal**, under some **set of constraints**

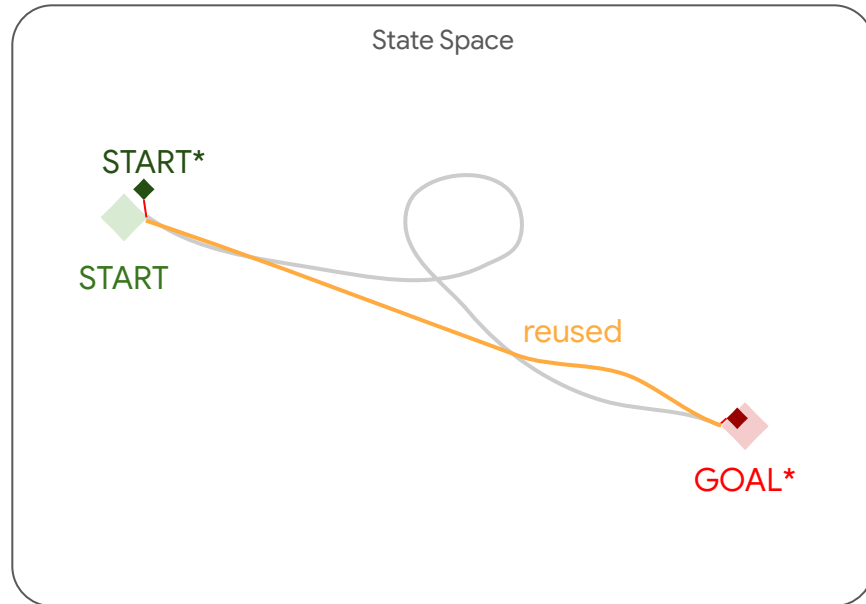
If faced with a “close-enough” scenario, just reuse the cached path



# Developing Idea

If you planned again, you might find a “better”, more optimal path

If the cache is aware of the metric, you could rank paths and use the best one



# A cache lets you reap all benefits

Sampling-Based

Optimization-Based

**Fast**



Slow

Random



**Deterministic**

Rarely finds “more optimal” paths



**Finds “more optimal” paths**



# A cache lets you reap all benefits

By skipping planning through plan reuse and ranking

Sampling-Based

Optimization-Based

**Fast**

Slow

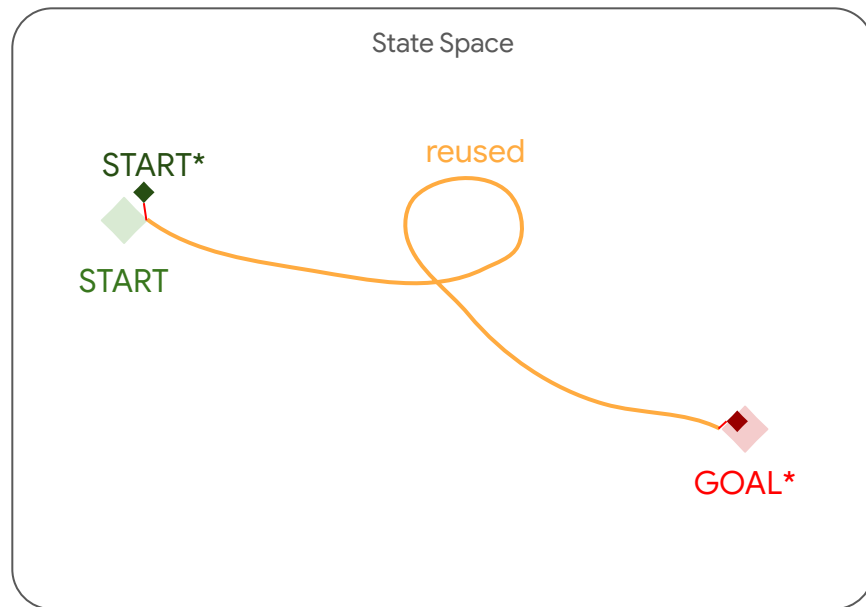
Random

**Deterministic**

Rarely finds “more optimal” paths

**Finds “more optimal” paths**

# A **fuzzy-matching** cache lets you reap all benefits and Allows you to automate matches of suitable trajectories



MoveIt: Is there a trajectory-cache?

Asked 7 years, 8 months ago Modified 7 years, 8 months ago Viewed 21 times

# The Fuzzy-Matching Trajectory Cache

finally after 7 years, it arrives

moveit/moveit2

# #2838 Implement fuzzy- matching Trajectory Cache 🔥



17 comments 52 reviews 8 files +3487 -0



methylDragon • May 18, 2024 27 commits



Available now in MoveIt2!

moveit/moveit2

## #2838 Implement fuzzy-matching Trajectory Cache 🔥



17 comments 52 reviews 8 files +3487 -0

methylDragon • May 18, 2024 27 commits



Available now in MoveIt2!

moveit/moveit2

## #2941 Fuzzy-matching Trajectory Cache Injectable Traits refactor...



6 comments 0 reviews 33 files +7078 -1658

methylDragon • July 31, 2024 51 commits



Pending refactor for extensibility

moveit/moveit2

## #2838 Implement fuzzy-matching Trajectory Cache 🔥



17 comments 52 reviews 8 files +3487 -0

methylDragon • May 18, 2024 27 commits



Available now in MoveIt2!

moveit/moveit2

## #2941 Fuzzy-matching Trajectory Cache Injectable Traits refactor...



6 comments 0 reviews 33 files +7078 -1658

methylDragon • July 31, 2024 51 commits



Pending refactor for extensibility

moveit/moveit2\_tutorials

## #940 Add Trajectory Cache Example For Refactor



0 comments 0 reviews 23 files +2354 -0

methylDragon • August 7, 2024 7 commits



With interactive demo!



# Good News!

The cache is available **RIGHT NOW**



# Better News!

It's based on work done for <https://github.com/osrf/nexus>

Which means it's been used with  
real SCARA and 6-DoF manipulators





# Incredible News!

Even in freespace planning, we've seen

**We saw 5%-99% reduction in planning time in  
production**



# Bananas News!

The cache works for motion plans and cartesian plans

And is usable with ANY planner and robot with  
single-DoF joints

**Quick “Demo”**

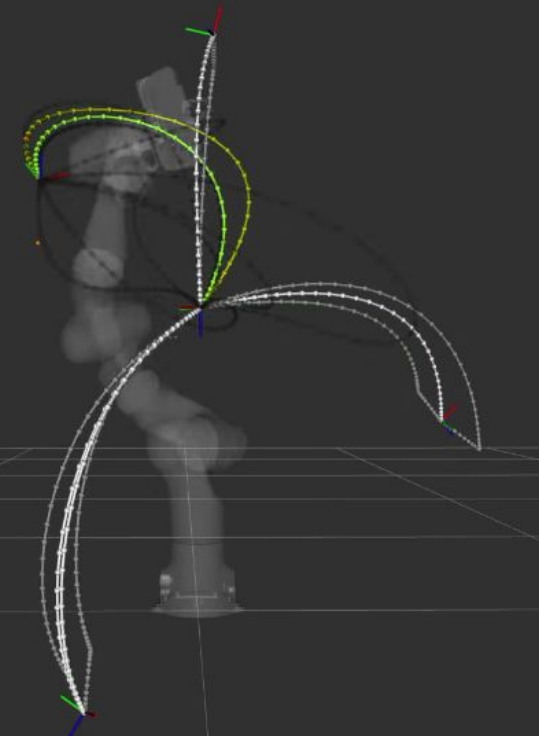
# The following are screenshots from the interactive tutorial demo

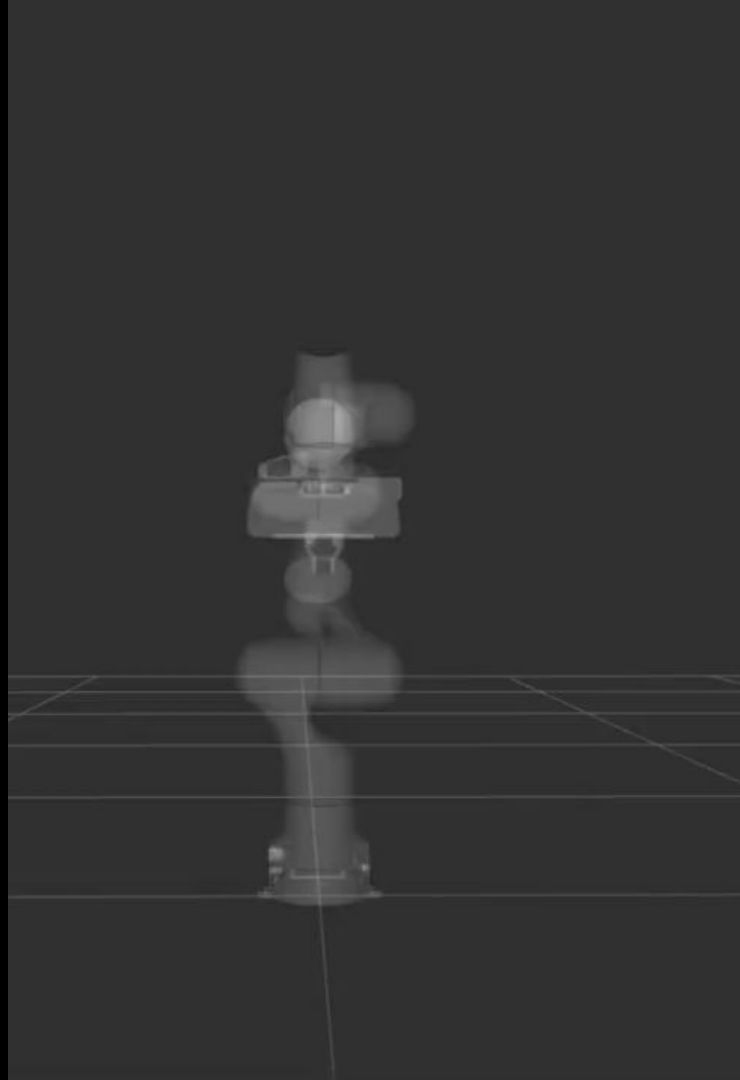
```
Demo_ExecuteWithCache(3/4)

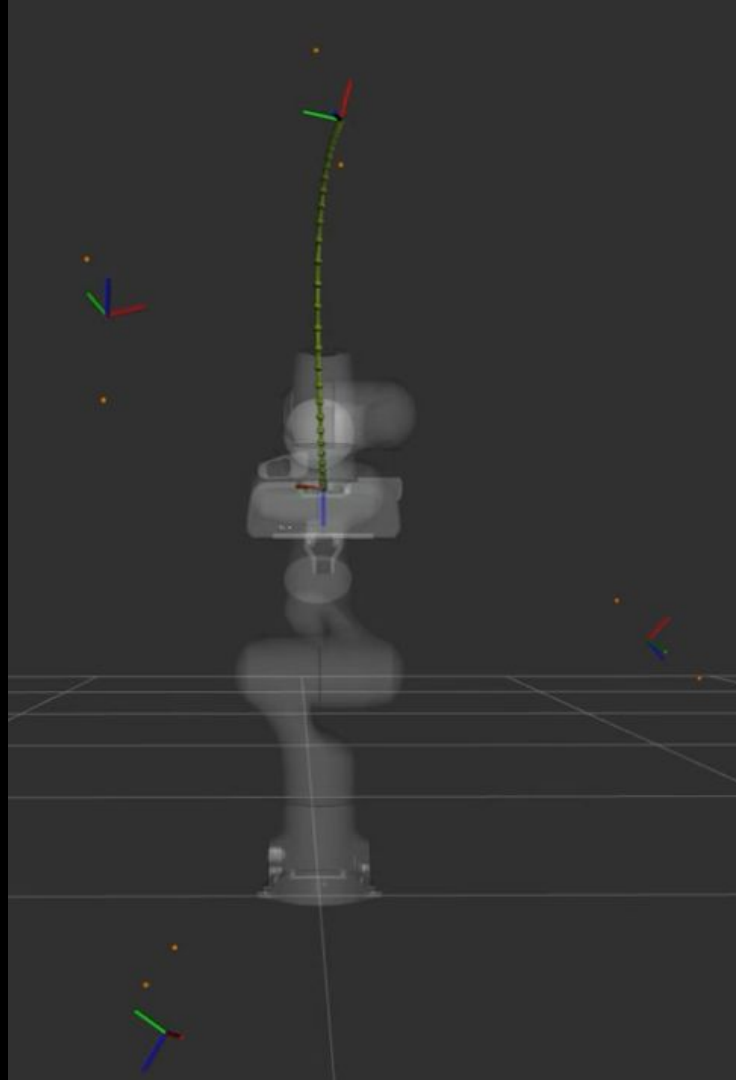
cached-motion-plans:      16
cached-cartesian-plans:   5
fetched-plan-planning-time: 0.023438
fetched-plan-fetch-time:  0.0101488

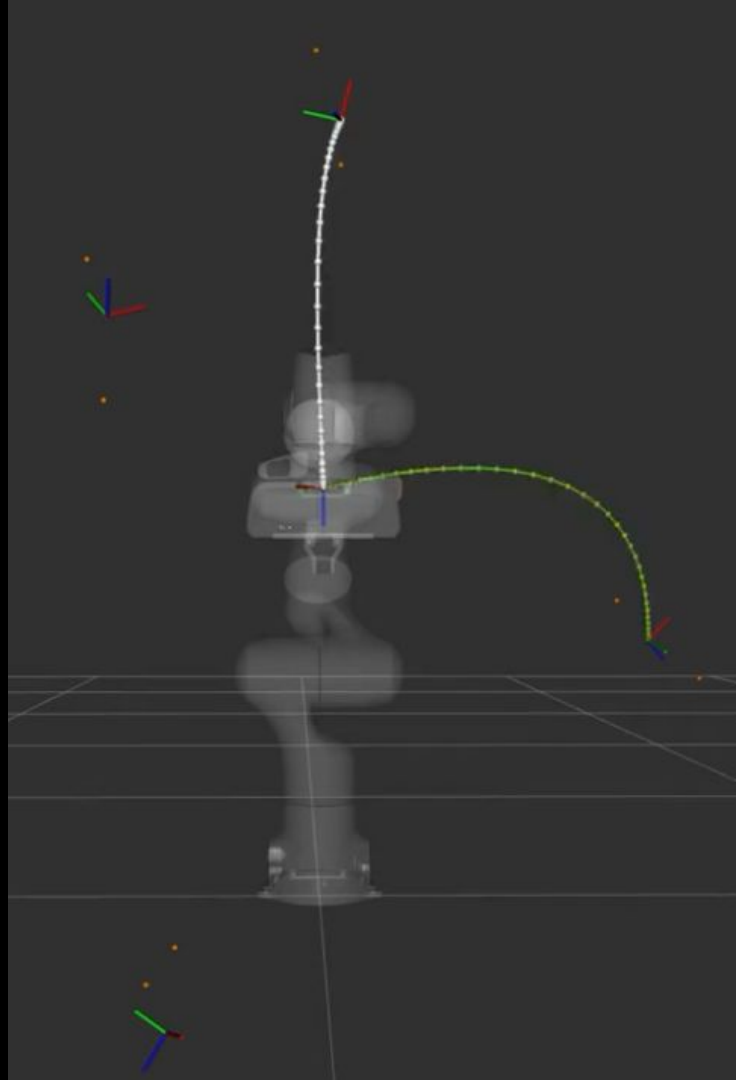
[[PARAMETERS]]
cache_db_host:           :memory:
start_tolerance:         0.025
goal_tolerance:          0.001
delete_worse_trajectories: true

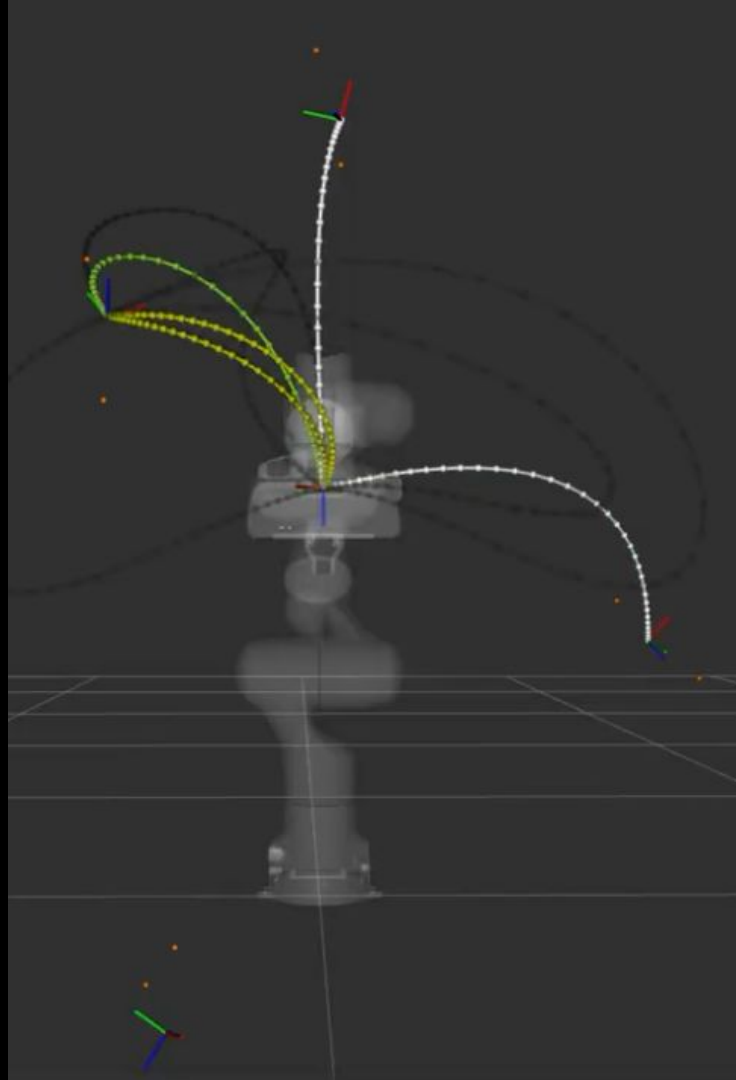
[[LEGEND]]
TRANSLUCENT:            planner_plans
GREY:                   all_cached_plans
WHITE:                  matchable_cached_plans
YELLOW:                 matched_cached_plans
GREEN:                  best_cached_plan
RED:                    diff_to_trajectory
```



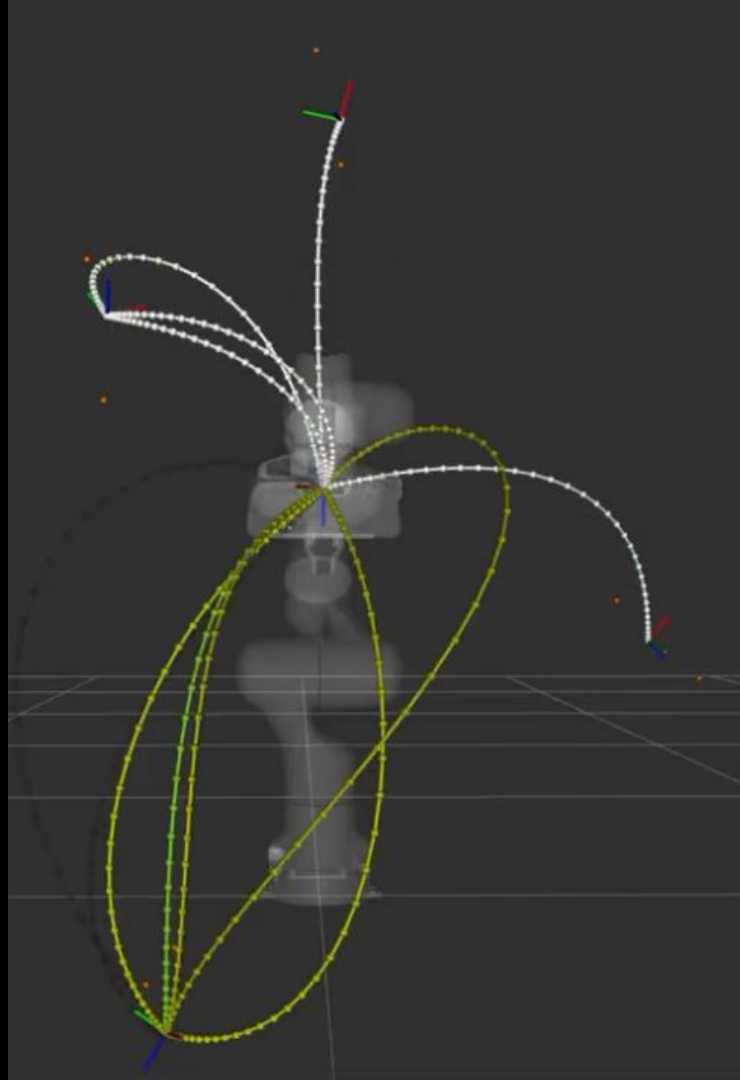








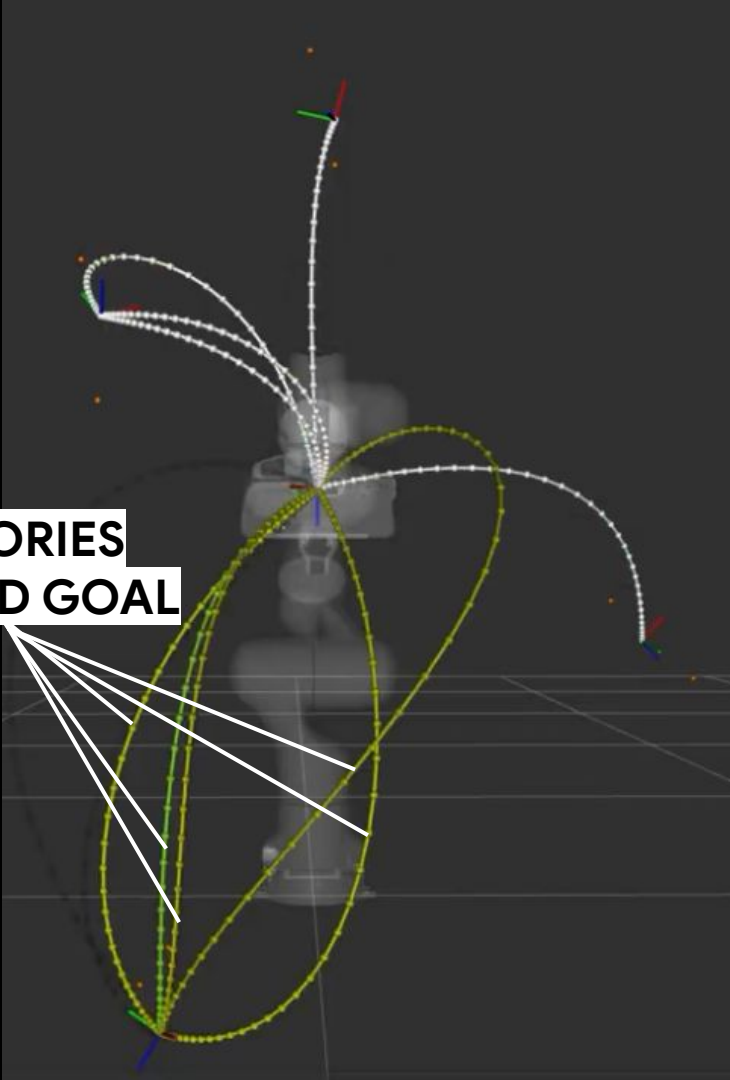






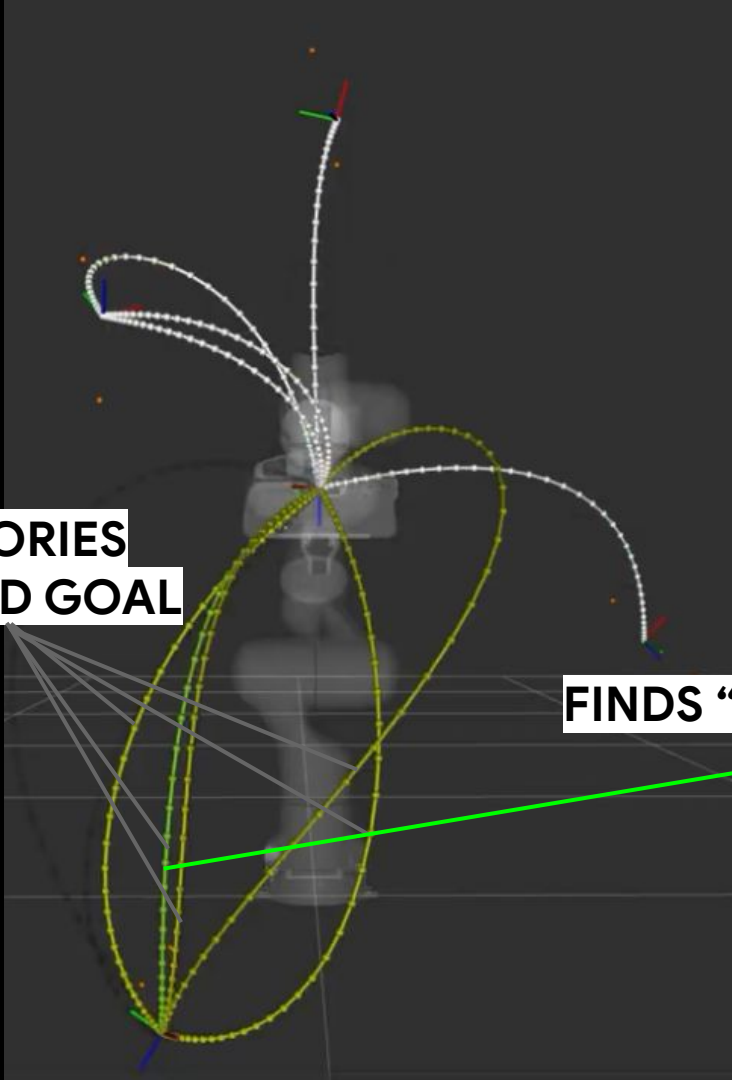
**CACHE POPULATION WORKS**

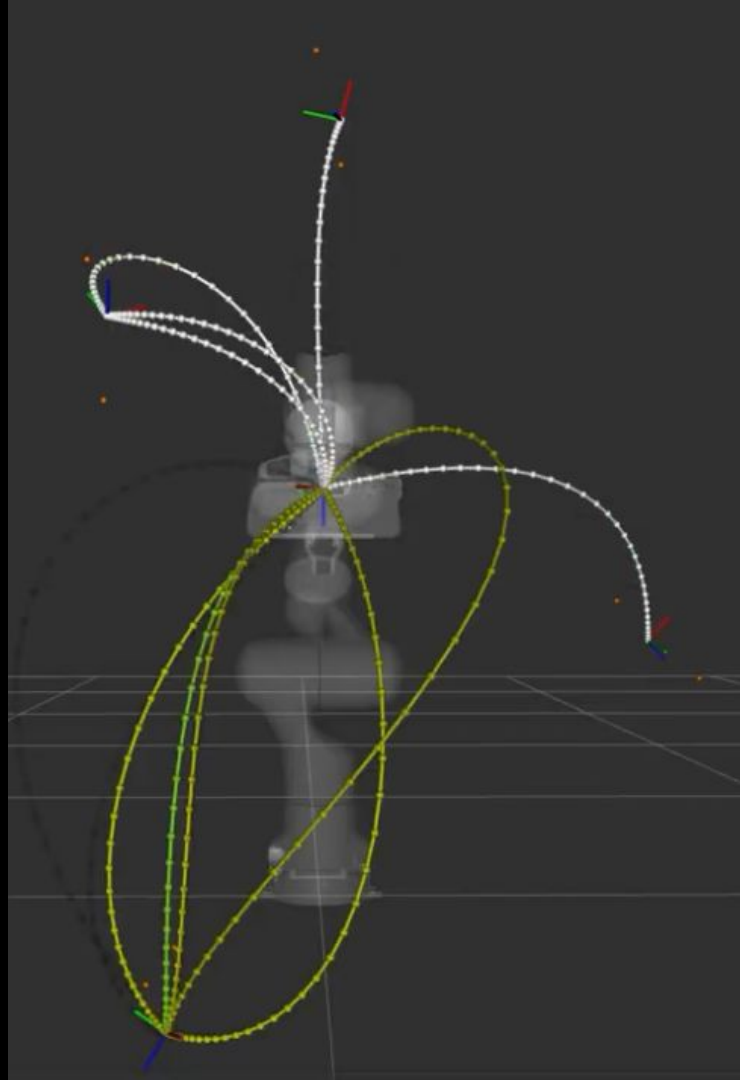
**FETCHES TRAJECTORIES  
MATCHING START AND GOAL**

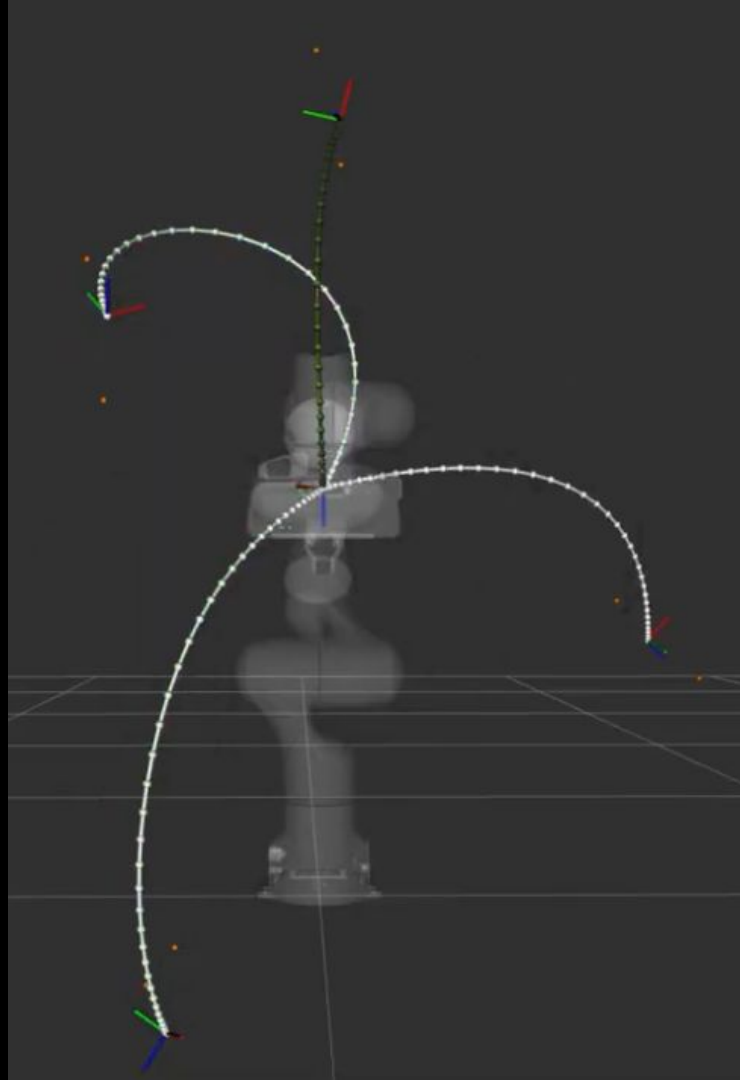


**FETCHES TRAJECTORIES  
MATCHING START AND GOAL**

**FINDS "BEST" TRAJECTORY**









**ALSO DOES PRUNING**  
(PRESERVING BEST)



**ALSO DOES PRUNING**  
**(PRESERVING BEST)**

**Over time the cache “learns”  
the best trajectory!**

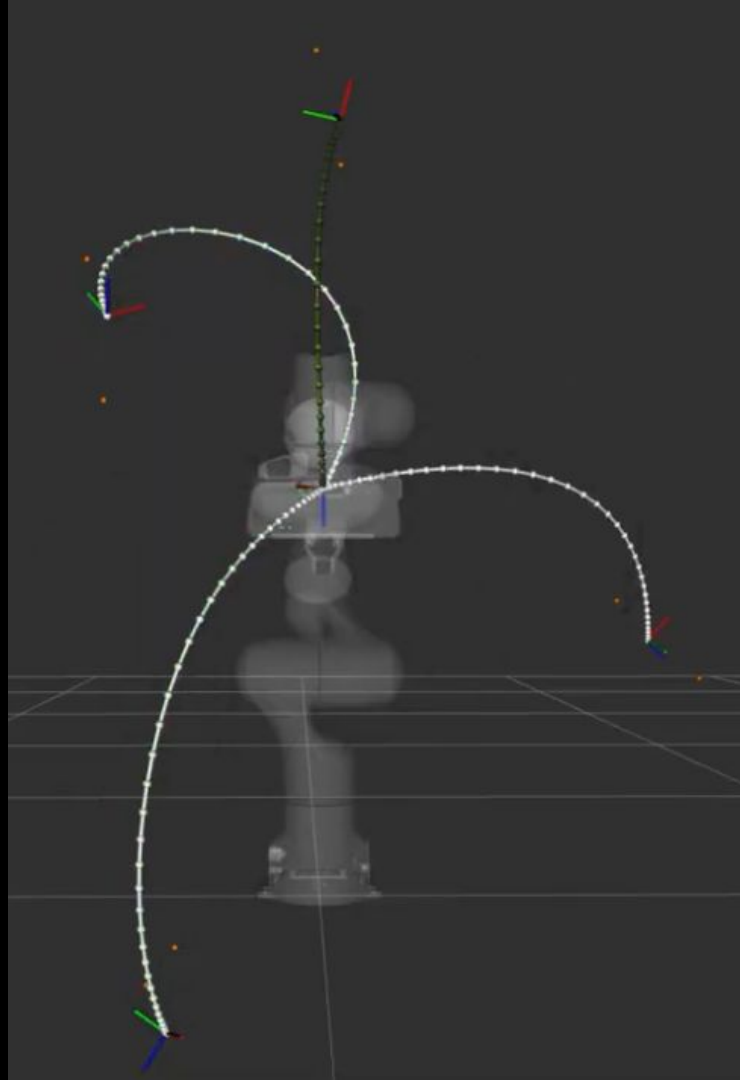


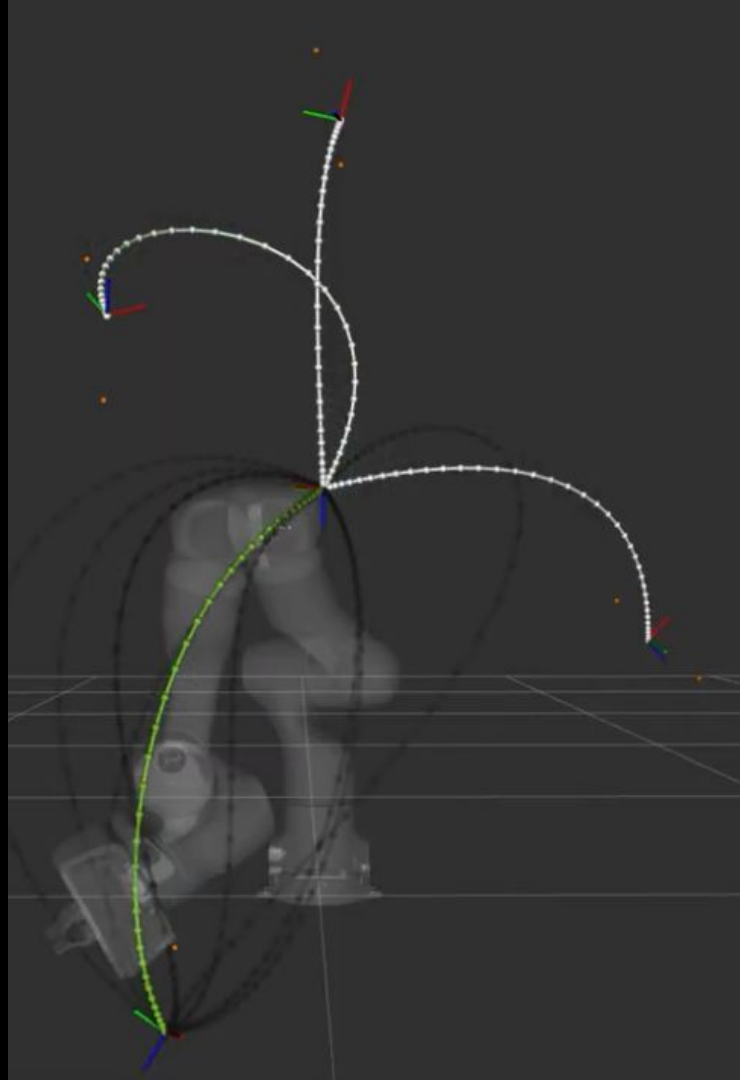


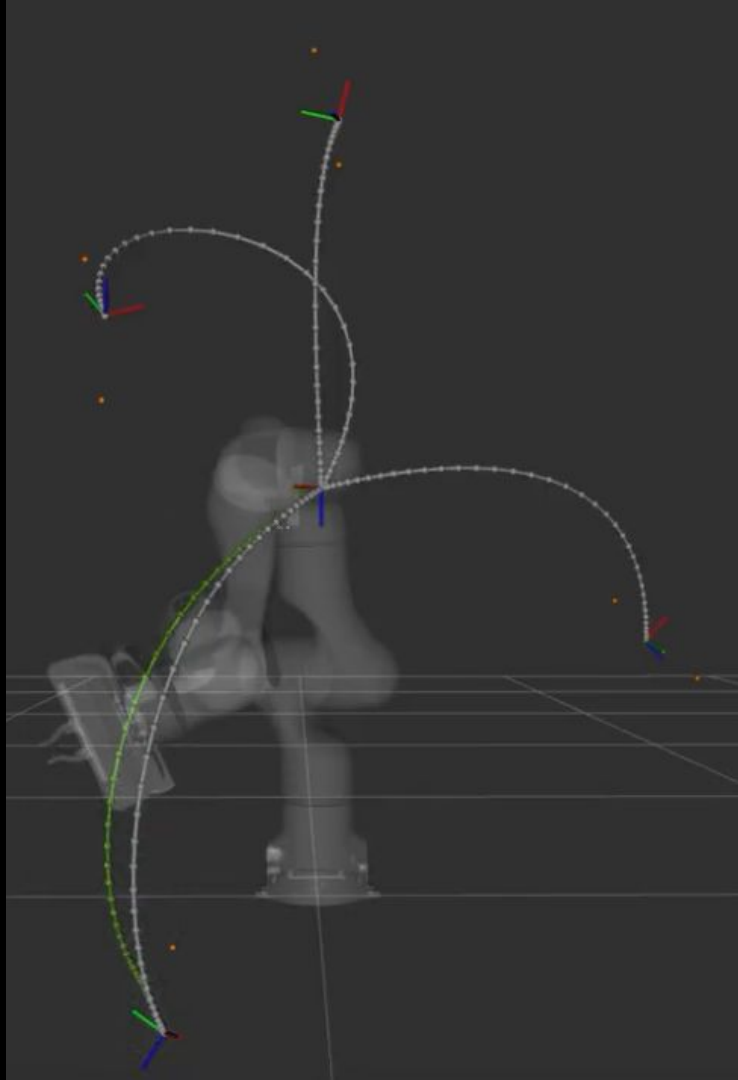
**ALSO DOES PRUNING**  
(PRESERVING BEST)

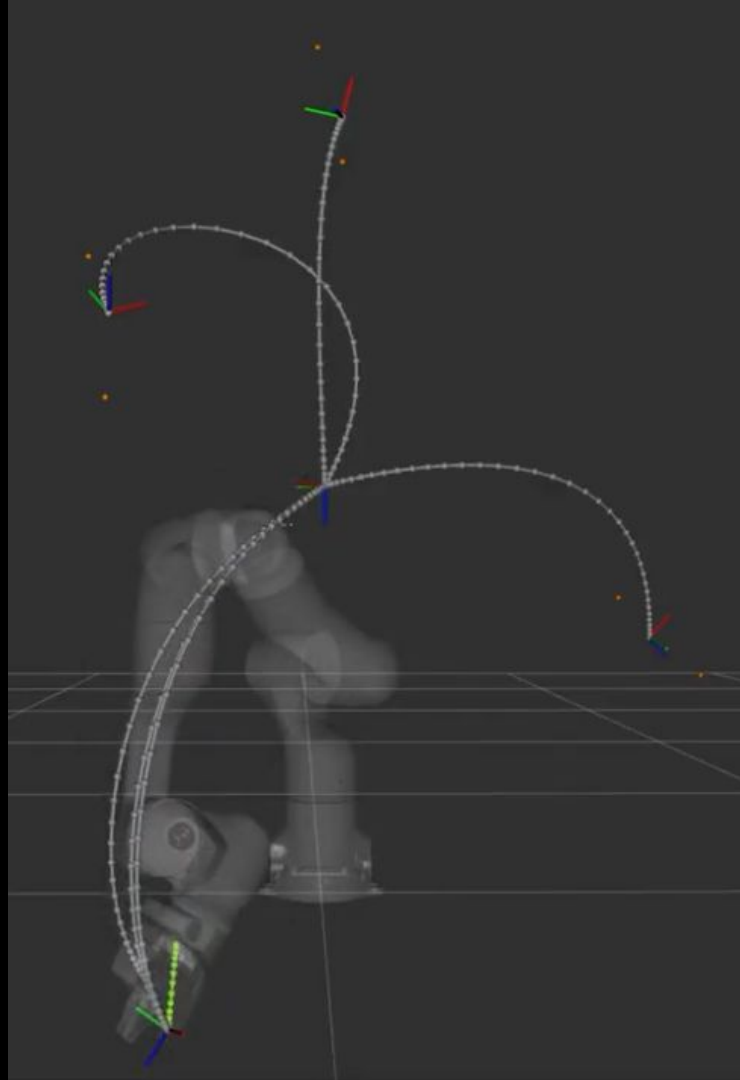
Over time the cache “learns”  
the best trajectory!

**THIS IS NOT DEEP LEARNING**

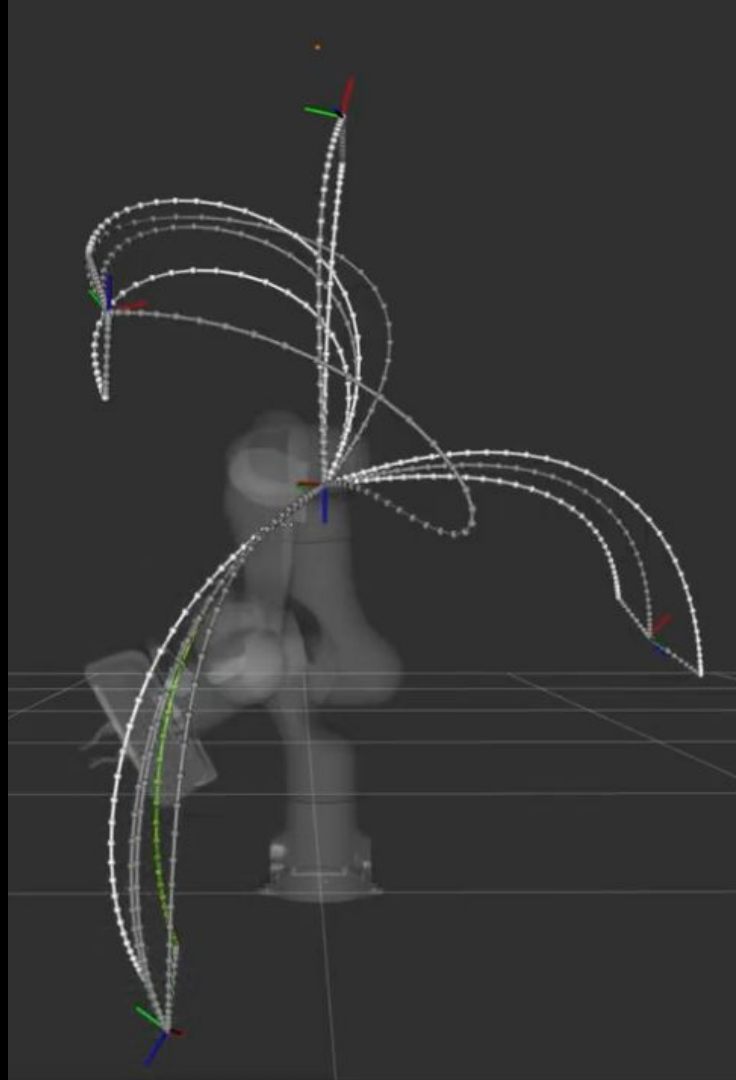








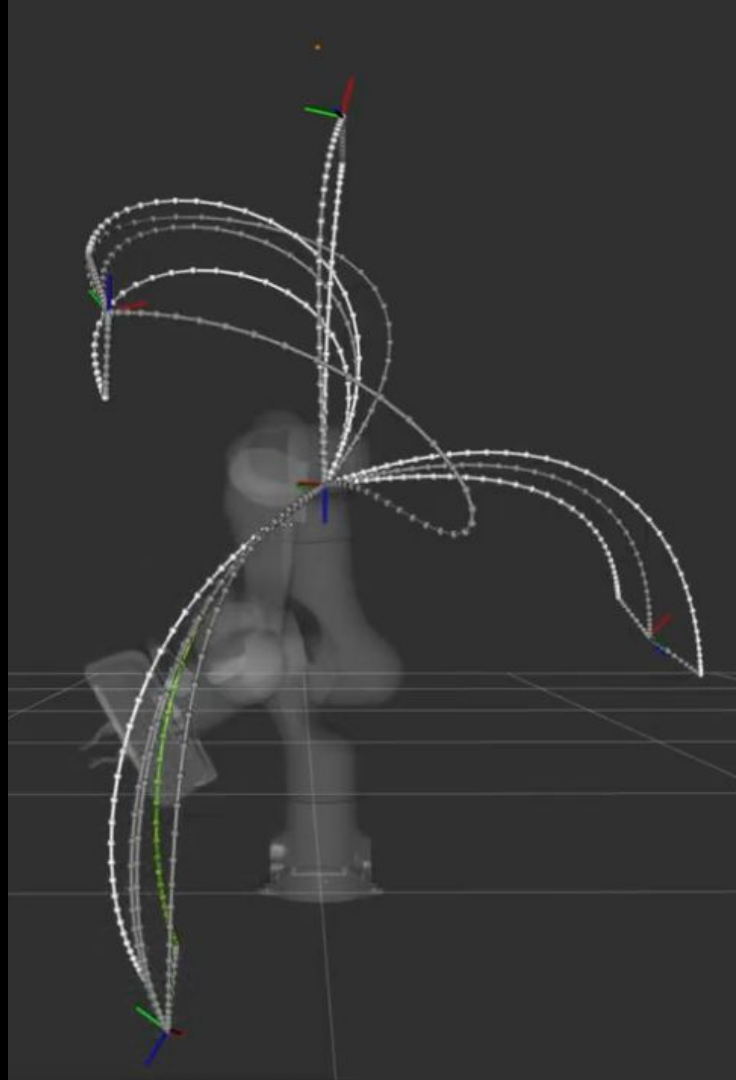
**SOME TIME LATER...**

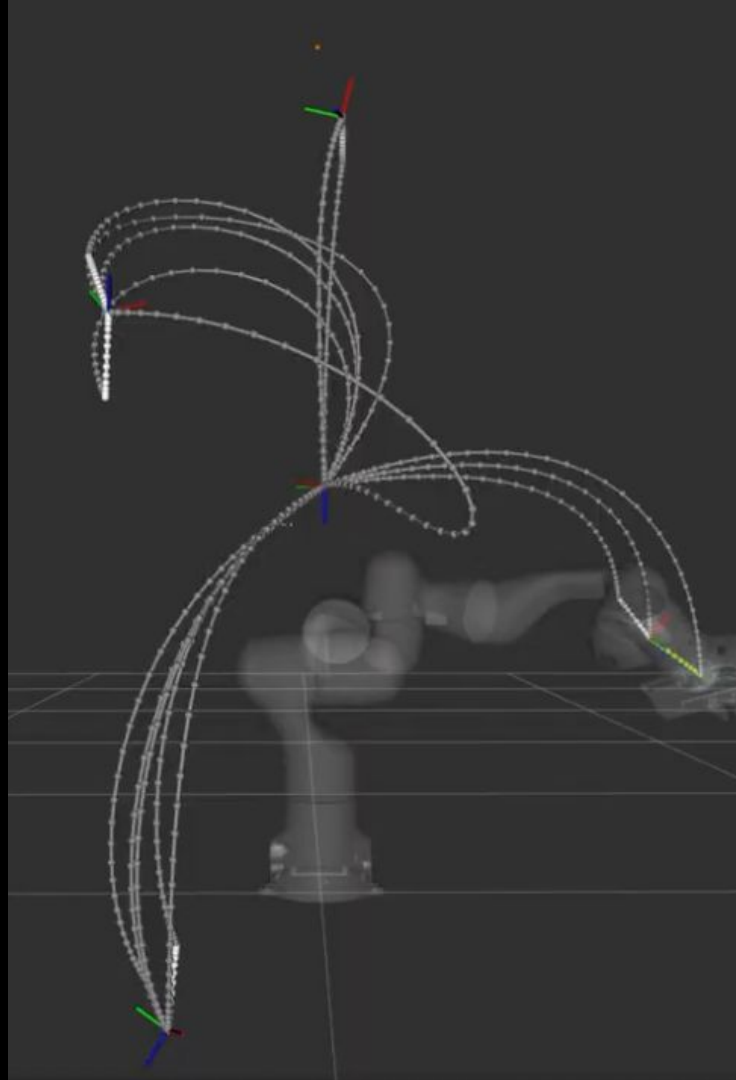




**FETCH AND EXECUTE BEST  
OK**



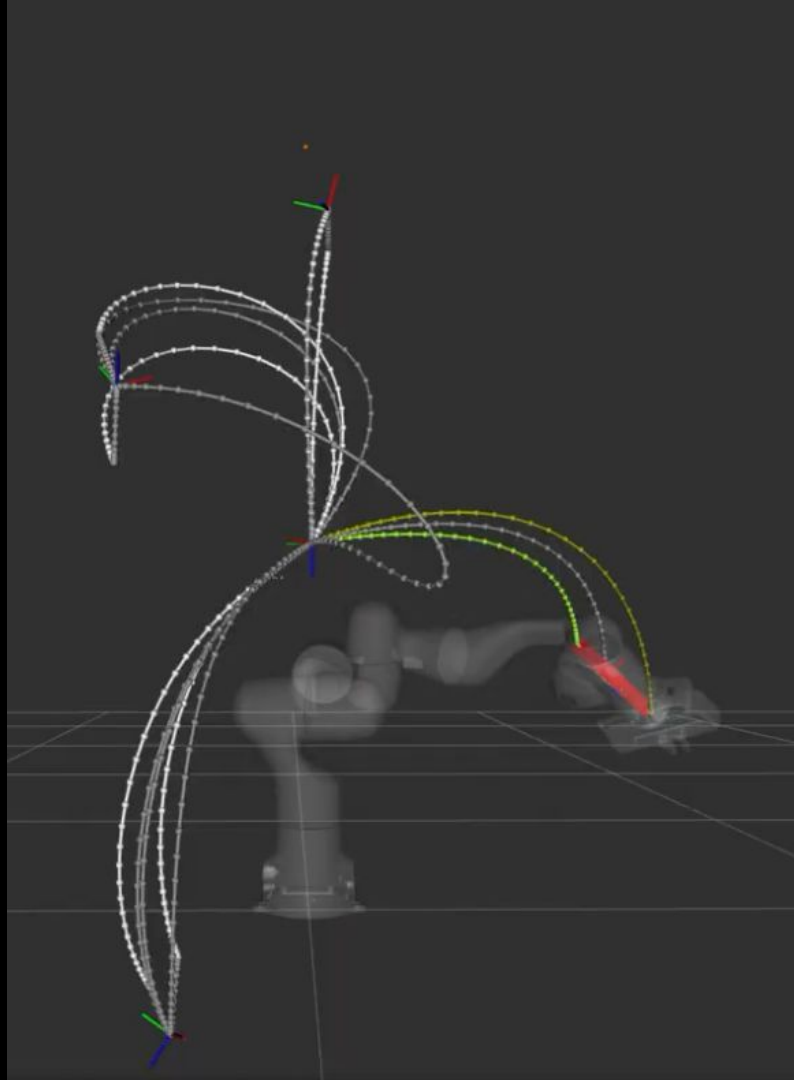






**CARTESIAN PLANS**

**OK**



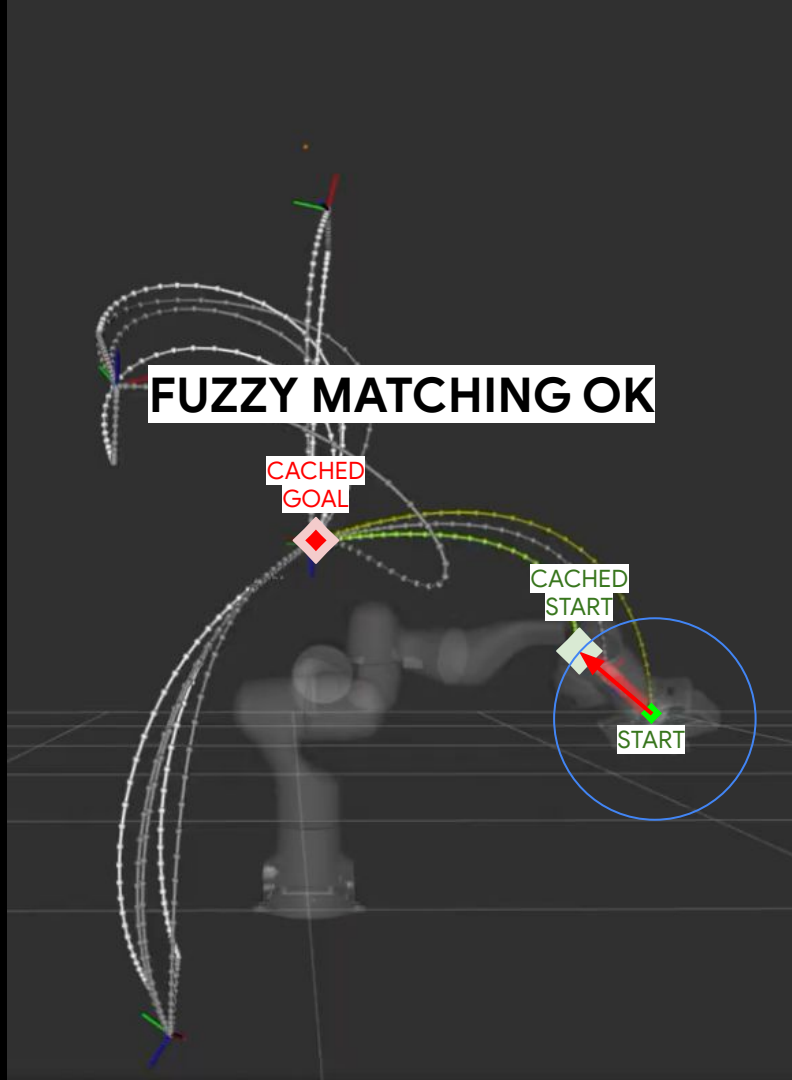


**FUZZY MATCHING OK**

CACHED  
GOAL

CACHED  
START

START



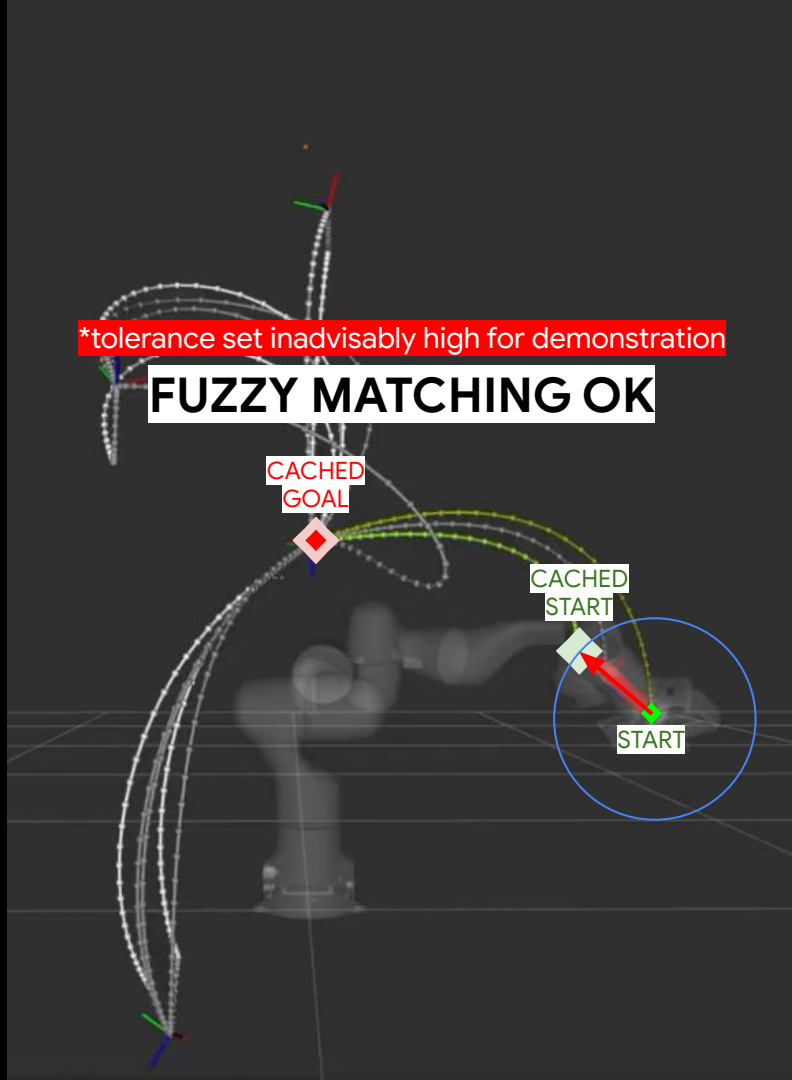
\*tolerance set inadvisably high for demonstration

## FUZZY MATCHING OK

CACHED  
GOAL

CACHED  
START

START





**DEMO DONE**



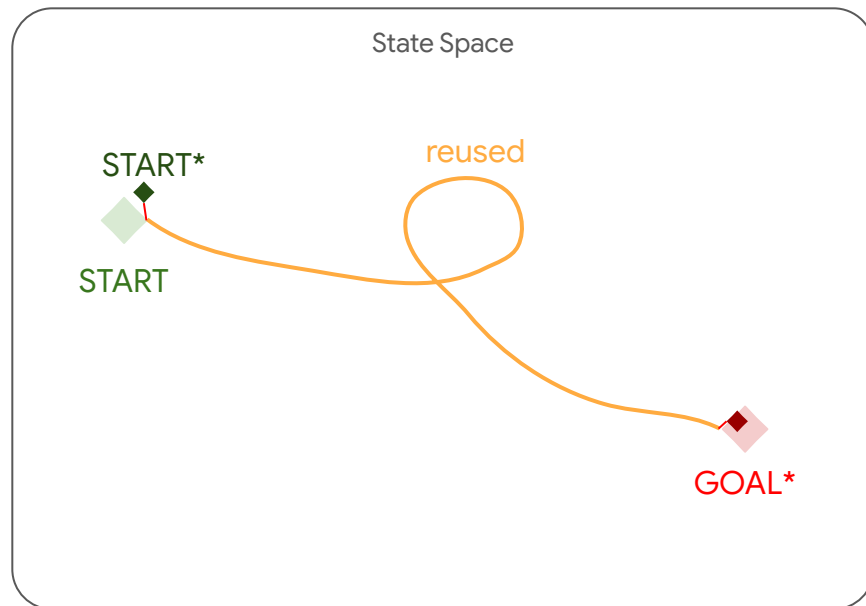


# How It Works

# Recalling The Core Idea

If you've already **cached** a path from some **start to goal**, under some **set of constraints**

If faced with a “close-enough” scenario, just reuse the cached path



# Recalling The Core Idea

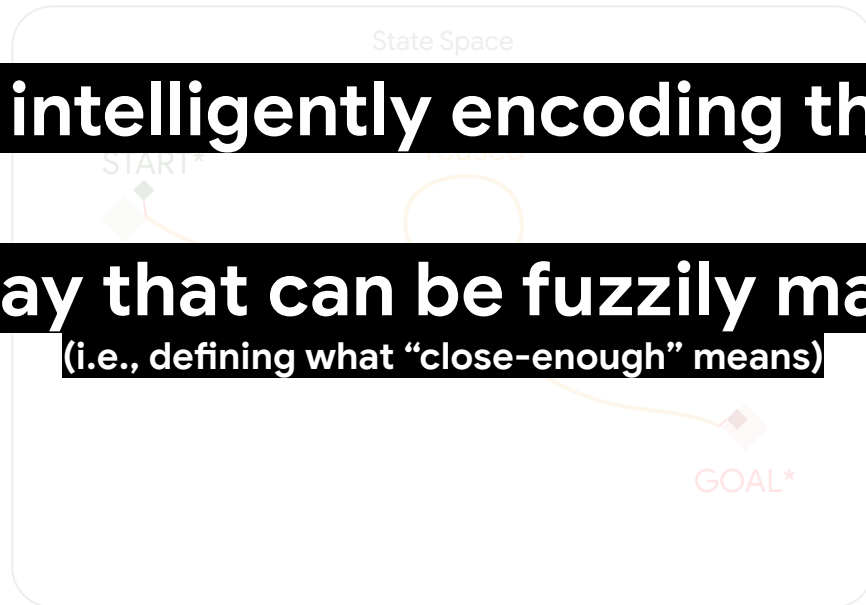
If you've already **cached** a path from some **start to goal**, under some **set of constraints**

If faced with a **“close-enough” scenario**, just reuse the cached path

**The key is intelligently encoding the scenario**

**In a way that can be fuzzily matched**

(i.e., defining what “close-enough” means)



# Encoding The Scenario

# Encoding The Scenario

We key the cache on:

1. **Workspace Features**
  - Move group name
  - Planning frame ID
  - Workspace limits

# Encoding The Scenario

We key the cache on:

## 1. **Workspace Features**

- Move group name
- Planning frame ID
- Workspace limits

## 2. **Robot Starting State Features**

- In configuration space (i.e., the joint states)!

# Encoding The Scenario

We key the cache on:

## 1. **Workspace Features**

- Move group name
- Planning frame ID
- Workspace limits

## 2. **Robot Starting State Features**

- In configuration space (i.e., the joint states)!

## 3. **Goal Constraints Features**

- Fetched from plan request message
- Considered:
  - Acceleration limits
  - Joint/pose goal
  - Etc.

And many more “features” of the scenario!

# Encoding The Scenario

We key the cache on:

## 1. **Workspace Features**

- Move group name
- Planning frame ID
- Workspace limits

## 2. **Robot Starting State Features**

- In configuration space (i.e., the joint states)!

**\*all poses canonicalized  
to robot base frame**

## 3. **Goal Constraints Features**

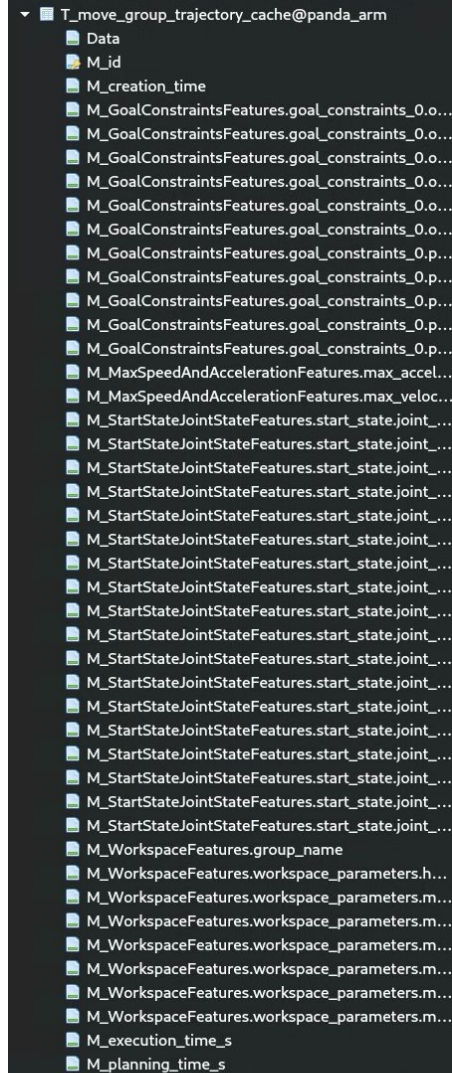
- Fetched from plan request message
- Considered:
  - Acceleration limits
  - Joint/pose goal
  - Etc.

And many more “features” of the scenario!



The cache is a  
**warehouse\_ros database**

All those features are inserted as  
a giant list of warehouse\_ros  
metadata annotations



# Fuzzily Fetching Plans

A cache entry is a valid match if it is:

1. **Close enough**
2. **Cached under “greater-than-or-equaly” strict constraints than the lookup request**

# Fuzzily Fetching Plans

A plan fetch is just a long list of warehouse\_ros DB lookup queries

# Fuzzily Fetching Plans

A plan fetch is just a long list of warehouse\_ros DB lookup queries

- **Exact** lookups for...
  - Joint names
  - Frame names
  - Robot names
  - Etc.

# Fuzzily Fetching Plans

A plan fetch is just a long list of warehouse\_ros DB lookup queries

- **Exact** lookups for...
  - Joint names
  - Frame names
  - Robot names
  - Etc.
- **Less-Than-Or-Equal / Greater-Than-Or-Equal** lookups for...
  - Acceleration Limits
  - Velocity Limits
  - Workspace Limits
  - Etc.

# Fuzzily Fetching Plans

A plan fetch is just a long list of warehouse\_ros DB lookup queries

- **Exact** lookups for...
  - Joint names
  - Frame names
  - Robot names
  - Etc.
- **Less-Than-Or-Equal / Greater-Than-Or-Equal** lookups for...
  - Acceleration Limits
  - Velocity Limits
  - Workspace Limits
  - Etc.
- **Range** lookups for...
  - Everything else

# Fuzzily Fetching Plans

A plan fetch is just a long list of warehouse\_ros DB lookup queries

\*match tolerance is independently adjustable for start and end constraints!

All the way to  $0+\epsilon$   
(floating-point exact-match)

# Using The Cache



# Insert and Fetch

```
// Insert (and prune)
```

```
cache->insertTrajectory(  
    move_group, robot_name, std::move(plan_req_msg), std::move(plan),  
    /*cache_insert_policy=*/BestSeenExecutionTimePolicy(),  
    /*prune_worse_trajectories=*/true,  
    /*additional_features=*/{});
```

```
// Fetch
```

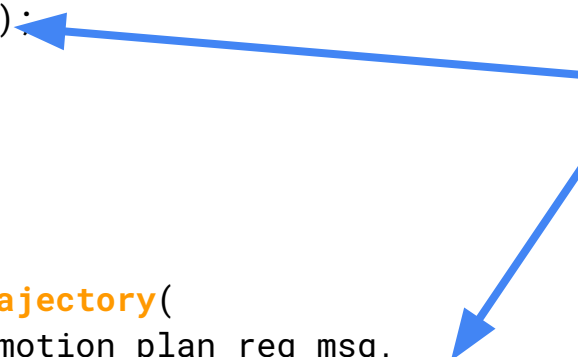
```
auto fetched_trajectory =  
    cache->fetchBestMatchingTrajectory(  
        move_group, robot_name, motion_plan_req_msg,  
        /*features=*/TrajectoryCache::getDefaultFeatures(start_tolerance, goal_tolerance),  
        /*sort_by=*/TrajectoryCache::getDefaultSortFeature(),  
        /*ascending=*/true);
```

# Insert and Fetch

```
// Insert (and prune)
```

```
cache->insertTrajectory(  
    move_group, robot_name, std::move(plan_req_msg), std::move(plan),  
    /*cache_insert_policy=*/BestSeenExecutionTimePolicy(),  
    /*prune_worse_trajectories=*/true,  
    /*additional_features=*/{}):
```

Inject your own cache  
feature extractors!

A blue callout box with white text is positioned to the right of the code. Two blue arrows originate from the box: one points to the `/*additional_features=*/{}` parameter in the `insertTrajectory` function call, and the other points to the `/*features=*/TrajectoryCache::getDefaultFeatures` parameter in the `fetchBestMatchingTrajectory` function call.

```
// Fetch
```


```
auto fetched_trajectory =  
    cache->fetchBestMatchingTrajectory(  
        move_group, robot_name, motion_plan_req_msg,  
        /*features=*/TrajectoryCache::getDefaultFeatures(start_tolerance, goal_tolerance),  
        /*sort_by=*/TrajectoryCache::getDefaultSortFeature(),  
        /*ascending=*/true);
```

# Insert and Fetch

```
// Insert (and prune)
```

```
cache->insertTrajectory(  
    move_group, robot_name, std::move(plan_req_msg), std::move(plan),  
    /*cache_insert_policy=*/BestSeenExecutionTimePolicy(),  
    /*prune_worse_trajectories=*/true,  
    /*additional_features=*/{});
```

Inject your own insert  
and prune policy!



```
// Fetch
```

```
auto fetched_trajectory =  
    cache->fetchBestMatchingTrajectory(  
        move_group, robot_name, motion_plan_req_msg,  
        /*features=*/TrajectoryCache::getDefaultFeatures(start_tolerance, goal_tolerance),  
        /*sort_by=*/TrajectoryCache::getDefaultSortFeature(),  
        /*ascending=*/true);
```

# Extending The Cache

You can implement and pass in your own:

- **Feature extractors**
  - (for encoding the scenario and fetching)
- **Cache insert policy**
  - (for pruning and insertion logic)
  - (also associates with a set of pre-baked feature extractors)

# Extending The Cache

You can implement and pass in your own:

- **Feature extractors**
  - (for encoding the scenario and fetching)
- **Cache insert policy**
  - (for pruning and insertion logic)
  - (also associates with a set of pre-baked feature extractors)

The default implementations support sorting and pruning by **execution time**.

# Extending The Cache

You can implement and pass in your own:

- **Feature extractors**
  - (for encoding the scenario and fetching)
- **Cache insert policy**
  - (for pruning and insertion logic)
  - (also associates with a set of pre-baked feature extractors)

The default implementations support sorting and pruning by **execution time**.

**The cache provides extension points for you to implement other functionality,** like sorting and pruning by **path length/minimum jerk/etc.** instead!

# Abstracting The Cache

**You can build on top of the cache!**

# Abstracting The Cache

**You can build on top of the cache!**

**Here is a starter idea, cache modes:**



# Abstracting The Cache

You can build on top of the cache!

Here is a starter idea, cache modes:

- **TrainingOverwrite**: Always plan, always insert, always prune
- **TrainingAppendOnly**: Always plan, always insert, never prune
- **ExecuteBestEffort**: Always fetch, only plan if fetch failed, never insert
- **ExecuteReadOnly**: Always fetch, never plan

# Abstracting The Cache

You can build on top of the cache!

Here is a starter idea, cache modes:

- **TrainingOverwrite**: Always plan, always insert, always prune
- **TrainingAppendOnly**: Always plan, always insert, never prune
- **ExecuteBestEffort**: Always fetch, only plan if fetch failed, never insert
- **ExecuteReadOnly**: Always fetch, never plan

You can see how such behaviors effectively model the "dev" and "deploy" phases of a robot deployment, and how they could be useful.

# Some Caveats

# Caveats

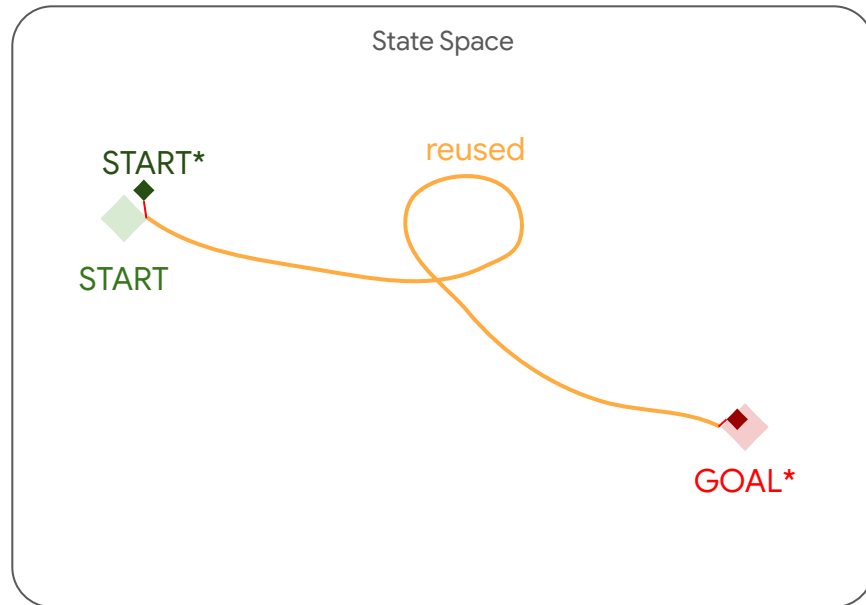
## Missing support for:

- Multi-DoF joints
- Constraint Regions

Contributions welcome for intelligent encoding strategies for those!

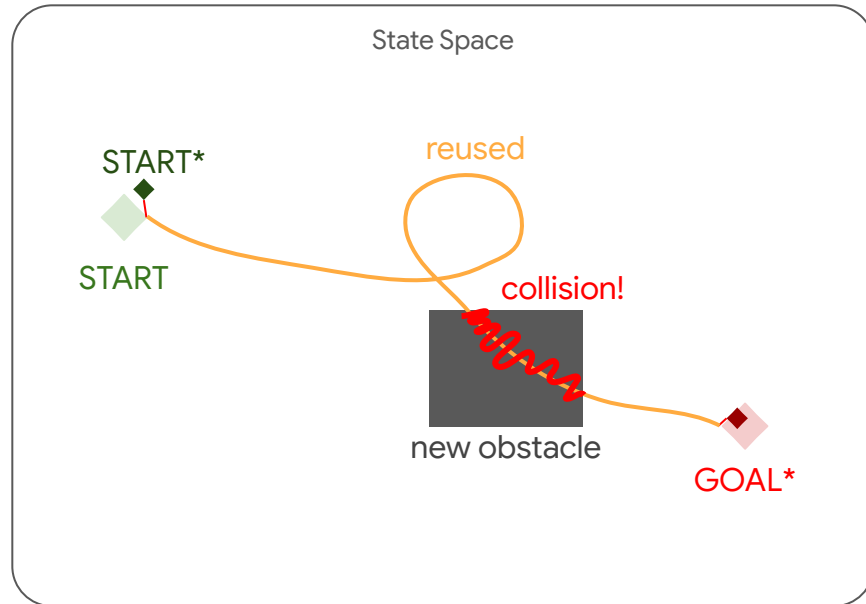
# Caveats

The cache **does not consider the planning scene**  
**(caching the scene is difficult)**



# Caveats

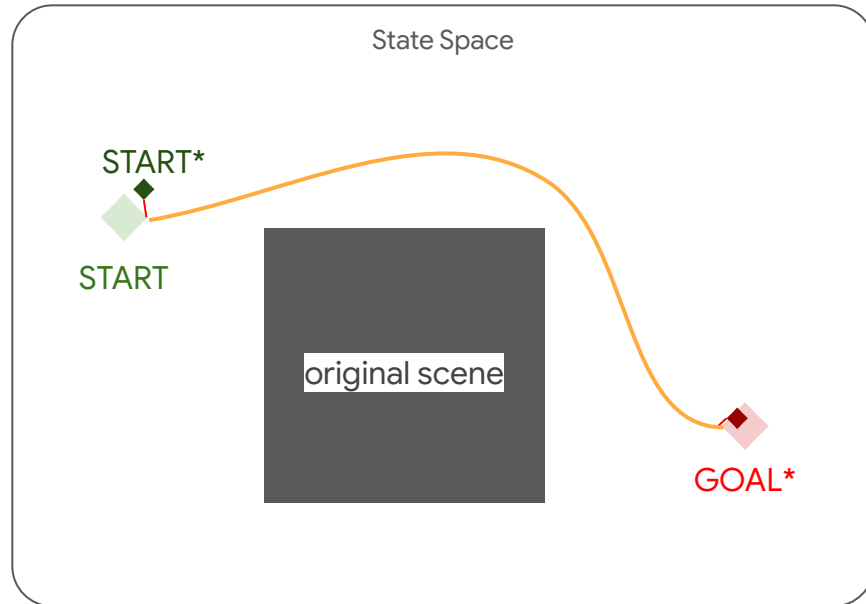
The cache **does not consider the planning scene**  
**If your scene changes, a fetch could result in a collision**



# Dealing With Collisions

# Dealing With Collisions

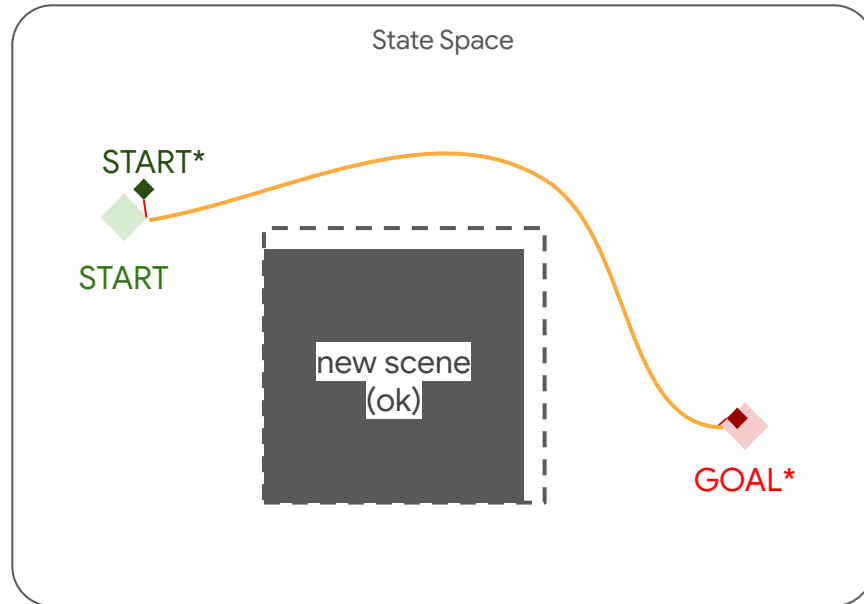
Keep your scene static, or strictly less obstructed





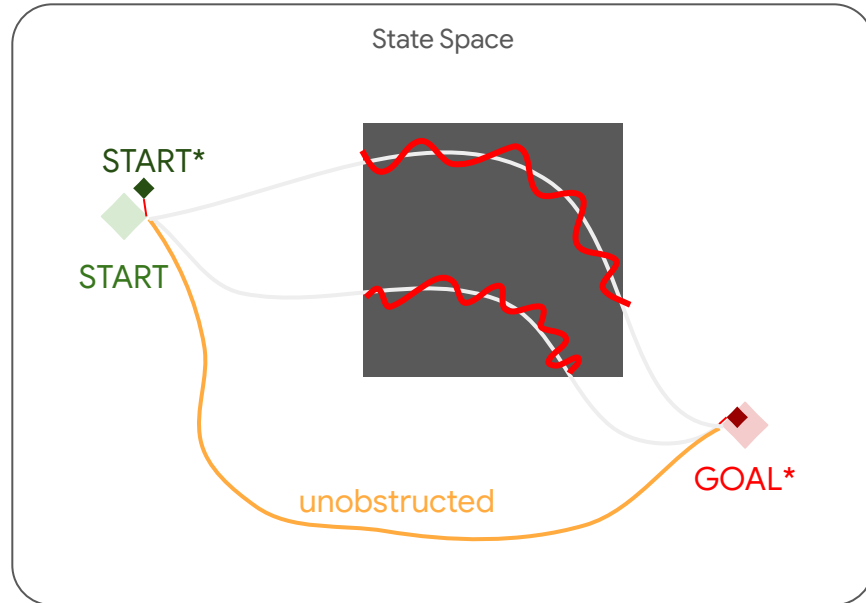
# Dealing With Collisions

Keep your scene static, or strictly less obstructed



# Dealing With Collisions

Or store and fetch multiple trajectories, and validate after fetch!  
(The cache lets you prune up to  $K$  trajectories for a given query)



Please cite the cache if you use it!

**CH3EERS!**

**question time.**

**Brandon Ong**

github.com/methylDragon  
SWE. Open Robotics @ Intrinsic

**Cache README**



github.com/  
moveit/moveit2/tree/main/moveit\_ros/trajectory\_cache