Sample-Based Motion Planning: Space, Compute, and Sampling Techniques

1 Abstract

Motion planning is an area of active research and sample-based methods provide a potentially efficient way of tackling high dimensional systems. This report will briefly survey a few general approaches within sample-based motion planning that aim to reduce space and compute resources, namely sparsity, motion primitives and importance sampling. In particular, methods from the survey papers range from shooting-based technique to 2-point boundary value problems with a gridded approach as well as incrementally improving trajectory via updating sample distributions. The report begins with a brief background, followed by reviews of surveyed papers, and finally follow up with a hybrid implementation integrating main ideas from these papers. Additionally, several parameters in the SST propagation algorithm are adjusted and studied. Through these, sample based methods enable a feasible way to compute a rough estimate of a feasible open loop control suitable for further optimization. Demonstration of the implementation in several scenarios that involve 2 point motion planning problems in planar and 3D domains will also be shown.

2 Background

Motion planning of dynamical system can be formulated as a transition system coupled with additional constraints of the system dynamics and the environment. Assuming an accurate model of the system, the state space, X, gives all possible instances of the system. The three dimensional space of the physical world as we know will be referred to as the configuration space, Q. Q is further partitioned into free Q_{free} and obstructed Q_{obs}(includes unavoidable obstacles due to inertia).

A general system of equation that describes a dynamical system is formulated as

$$\begin{aligned} &x \in X := \text{system state space} \\ &u \in U := \text{control space} \\ &\dot{x} = f(x,u) \\ &\dot{u} = g(x,u) \end{aligned} \tag{1}$$

. Reachability of the 2 point motion planning problem is formulated as finding a feasible control(out of possibly many)

$$\{u \mid x(\tau) = \int_0^\tau f(x(t),u(t))\ dt\} \tag{2}$$

$$x(0) = x_{start} \tag{3}$$

$$x(\tau) = x_{dest} \tag{4}$$

$$Q := \text{configuration space}$$

$$proj_Q(x(t)) \in Q_{free},\ t \in [0,\tau]$$

. Additionally, optimality can be further imposed as an addition constraint. One such form is

$$u^* = \underset{u(t)}{\operatorname{argmin}} \int_0^\tau cost(x(t), \dot{x(t)}, u(t), \dot{u(t)})\ dt \tag{5}$$

.

In general, the following assumptions are made for the applicability of the approaches in the surveyed papers:

- bounded change (Lipschitz) in cost function,
- bounded change (Lipschitz) in system states and control states,
- system is Small-Time Locally Accessible.

In the considered approaches in the surveyed papers, computing a propagation amounts to constructing a propagation tree for search consisting of discretized sequence of states and controls approximating the continuous system.

3 Models

The following system dynamics are used for simulation and benchmark.

3.1 Dubins Car

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \in X, u \in U \tag{6}$$
$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} Vcos(\theta) \\ Vsin(\theta) \\ u \end{bmatrix}$$
$$u \in [-40, 40] * 2\pi/180$$

.

3.2 Dubins Airplane

$$\begin{bmatrix} x \\ y \\ z \\ \theta \end{bmatrix} \in X, \begin{bmatrix} u \\ v \end{bmatrix} \in U \tag{7}$$
$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} Vcos(\theta) \\ Vsin(\theta) \\ v \\ u \end{bmatrix}$$
$$u \in [-40, 40] * 2\pi/180$$
$$v \in [-0.5, 0.5]$$

.

4 Metric

All demonstrations for the above models use a state space metric that is the square root of equal weighted linear combination of L2 squared positional displacement and squared rotational displacement (only 1 angle in the sample models), each normalized to the unit interval.

$$Dist_{ss}(a,b) = \left[\sum_{i=0}^{d} (a.pos_i - b.pos_i)^2 + \left(\frac{(a.theta - b.theta + 2\pi) \ mod \ 2\pi}{2\pi} \right)^2 \right]^{1/2}$$

$a.pos_i \in [0,1]$

$b.pos_i \in [0,1]$

$a.theta \in [0, 2\pi]$

$b.theta \in [0, 2\pi]$

The quality metric used for trajectory optimization in demonstrations is the minimization of total time duration of the trajectory. Then, (5) reduces to

$$u^* = \underset{u(t)}{\operatorname{argmin}} \int_0^\tau dt$$

, subject to other previously given constraints.

5 Environment

All environment samples are processed so that the physical space are normalized to be contained within [0,1] for each dimension (eg: 1x1 for 2D, 1x1x1 for 3D). Sample demonstrations are performed on randomly generated domains as well as structured maps taken from Dragon Age (available from `https://www.movingai.com/benchmarks/grids.html`).

6 Sparsity

Stable Sparse Rapidly Exploring Random Tree (SST) that is introduced by Yanbo Li, Zakary Littlefield, and Kostas E. Bekris is a sample-based planner that encourages sparseness by partitioning space into witness neighbourhood regions that is parameterized by some measure of closeness [4]. Given assumptions of Lipschitz continuity of system dynamics, cost function, Small-Time Locally Accessible of the system, and appropriate propagation parameters, this approach enables probabilistic guarantee of finding a trajectory that is within the parameterized neighbourhood if there is one.

Sparsity is enforced by the following:

- samples are considered to be within a neighbourhood if the sample is within some measure of closeness (δ_s) to the witness of the neighbourhood,
- each neighbourhood has one representative that is computed with respect to a given measure (such as propagation distance/time from x_{start}),
- a sample that is not a representative of any neighbourhood is marked as inactive,
- samples in the propagation tree that are leaves and inactive are eligible for pruning (samples that are not leaves and inactive cannot be pruned since there exists descendent samples that may be active),
- adding a sample that does not belong to any witness neighbourhood with the considered measure of closeness (δ_s), then that sample itself becomes a witness and this creates a new neighbourhood,
- adding a sample to an existing neighbourhood challenges the representative of that neighbourhood with respect to a quality metric (such as total propagation time from x_{start}.

Propagation mechanism is done via generating a sample in state space and querying the best existing sample within neighbourhood (δ_v) of the query with respect to a quality metric (such as total propagation time from x_{start}. If no sample is within δ_v, a nearest sample is used. The selected existing sample is used for Monte Carlo propagation by selecting a random propagation duration and control input within constraints of the system.

Given the described approach by the authors, SST has some interesting properties:

- witnesses are only added and not deleted; this allows precomputation of pseudo-randomly spaced witness nodes ahead of time if desired,
- sample generation, propagation, and propagation tree are computed in the state space of the system and ignores configuration space, however it needs projection onto configuration space for obstruction checking,
- nodes can be pruned from an inactive leaf towards the root of the propagation tree via local parent pointer of each node,
- parameter controlling the witness neighbourhood size ((δ_s) affects propagation tree sparseness which is related to the resolution of coverage in the configuration space,

- propagation delta has to be effectively larger than than some threshold to overcome neighbourhood size in order to explore unexplored regions,
- Monte Carlo propagation may lead to unuseful new sample due to sparsity constraints,

In the analysis of the algorithm, the authors use parameters δ_s (neighbourhood size of witness) and δ_v (neighbourhood size of selecting the best node with respect to some quality metric for propagation) in a series of steps in showing δ-robustness which is a probabilistic guarantee that a trajectory is generated going from a sample in a hypersphere to another hypersphere of neighbourhood as the number of iterations approaches infinity. This requires Lipschitz continuity of the system dynamics. Furthermore, Lipschitz continuity and monotonicity of the quality metric can affect generated trajectory classes. An important assumption is that Monte Carlo propagation has a non-zero probability of generation propagation from one hypersphere to another which must be taken into account when setting appropriate parameters for propagation.

Asymptotic near-optimality is shown by reducing δ to an arbitrary small amount, however this seems to limits the practical advantages of being a sparse approach in the first place. It is notable this idea can be incorporated into a practical implementation by dynamically changing parameters related to δ such as through annealing to quickly flood vast empty space at the start, however this can still be limiting for configuration spaces with critical path having multiple small bottleneck areas.

7 Motion Primitives

Sampling-based optimal kinodynamic planning with motion primitives by Sakcak, B and Bascetta, L and Ferretti, G and Prandini, M explores motion planning by a combination of precomputation of motion primitives and gridding to allow cheap lookup queries at runtime [5]. This approach requires Small-Time Locally Accessible of the system, and invariances of the system dynamics (with respect to translation and rotation in the demonstrated example), and transformations from the global space to the reference frame of the database lookup.

During the precomputation stage, a database is constructed on a small region of grids in which different combinations of control inputs are computed in order to achieve 2 point boundary value problem from the reference origin point to every other point in the grid. Thus, the price of computation via custom solvers for 2 point boundary value problem is paid up front. Subsequent planning in the larger space of the domain uses transformations in order to map global states to the reference frame of the database lookup. If such transformations exist, then the lookup procedure cheaply produces a potential propagation control input.

During runtime, tree rewiring is done to improve trajectory quality via triangle inequality comparison of transitivity between two nodes via a third node.

Given the described approach by the authors, motion primitives have the following properties:

- precomputation of motion primitives lookup can be done by minimizing a cost function (such as cost of control input magnitude) over 2 point boundary value problem,
- different primitives can be chained together since they obey invariance of staying on predetermined grid,
- database lookup effectively serves as encoding with additional opportunity for space requirement reduction from symmetric dynamics of the system,
- grid regularity eases lookup implementation, however the regularity may cause aliasing problems if resolution of the grid does not play well with problem specific configuration domain,
- guarantee of transition on uniform grid allows discrete algorithms to be applied to find a solution (eg: A^*),
- grid may need to be adjusted to allow grid point to coincide or near an acceptable neighbourhood of start and destination points,
- having finer grids impacts the number of connectible edges in the configuration domain ($O(\alpha^{2d})$, $\alpha :=$ resolution factor),
- collision checking is done after lookup query to check validity of the proposed trajectory.

Assuming Lipschitz continuity of cost function, asymptotic optimality of the algorithm is shown by setting arbitrarily small grid spacing. It is also notable that the quality of the trajectory is impacted by the precomputation stage of the motion primitive lookup. It is important that the precomputation step covers sufficient range of classes of trajectory or else a possibly smaller solution space is available for search at runtime. Considering only a finite number of trajectory classes are generated and available for lookup, it is apparent that the algorithm is incomplete for a given fixed size of grid resolution and arbitrarily hard configuration space.

8 Importance Sampling

Cross-entropy motion planning by Marin Kobilarov provides a general method for improving the quality of motion planning in a incremental way using estimation of rare events such that the model of the planner uses distributions that approach an optimal [3]. This can be applied globally or locally depending on the tradeoff of optimality and computation time of the problem at hand.

Proposed implementation of this method uses a given quality metric to rank samples and iteratively adapt the threshold of cutoff of quality based on qualities of observed samples. Practical implementation typically uses a certain quantile of samples (ranked with respect to the quality metric) for monotonically changing the quality threshold for the next iteration of optimization. This assumes that the rare event occurrence of obtaining good quality of trajectory path is not too low, which is reflected in the quantile parameter, such as 1% to 10%. Importantly, the flexibility of this framework allows use of wide range of distributions (such as Gaussian mixture model) to be used for sampling and allow any parameterizations that is used for generation of samples.

The general framework of importance sampling and use of cross entropy in the paper has the following characteristics.

- stochastic optimization,
- optimization that is not restricted to gradient based methods,
- allow use of mixture of large range of different distributions that can sufficiently be adapted to approximate the optimal distribution,
- iteratively changing sub-level-set of solutions by monotonically adjusting quality threshold used in evaluation of a rare event, assuming some constant quantile parameter for optimization ranking,
- has a similar flavour compared to Expectation Maximization and evolutionary approaches.

The authors of the paper demonstrated this framework by using RRT based variants (RRT,RRT*), parameterizing for forward chained end-to-end trajectory generation with heuristic estimate of cost-to-go from the tip of the trajectory to the goal and a Gaussian mixture model with 4 components. Parameters for trajectory generation is updated via Expectation Maximization. It is notable for certain distributions such as ones in the natural exponential family, analytical solutions exist and are suitable for parameter update. This optimization is done globally and reported results show that this is feasible for spaces with low obstruction counts.

9 Implementation

9.1 SST

The SST algorithm is implemented with adjustable parameters for experimentation. This forms the basis for additional features such as motion primitives and importance sampling.

In encouraging efficiency, Monte Carlo propagation is further clamped on possible propagation durations. Due to sparseness constraint of having one active node within a witness neighbourhood, propagation delta that is too small have higher chance in landing in the same witness region as the node, thus wasting iterations. Therefore it is reasonable to set sufficiently large propagation duration in order for higher chance of going to neighbouring spaces.

Below shows a comparison of observed metrics with different sparseness parameters for a given Dubins car scenario, going from green to red point, that records statistics leading up to finding a first feasible path. Total iterations includes iterations changed plus iterations unchanged (due to collision and witness neighbourhood contentions).

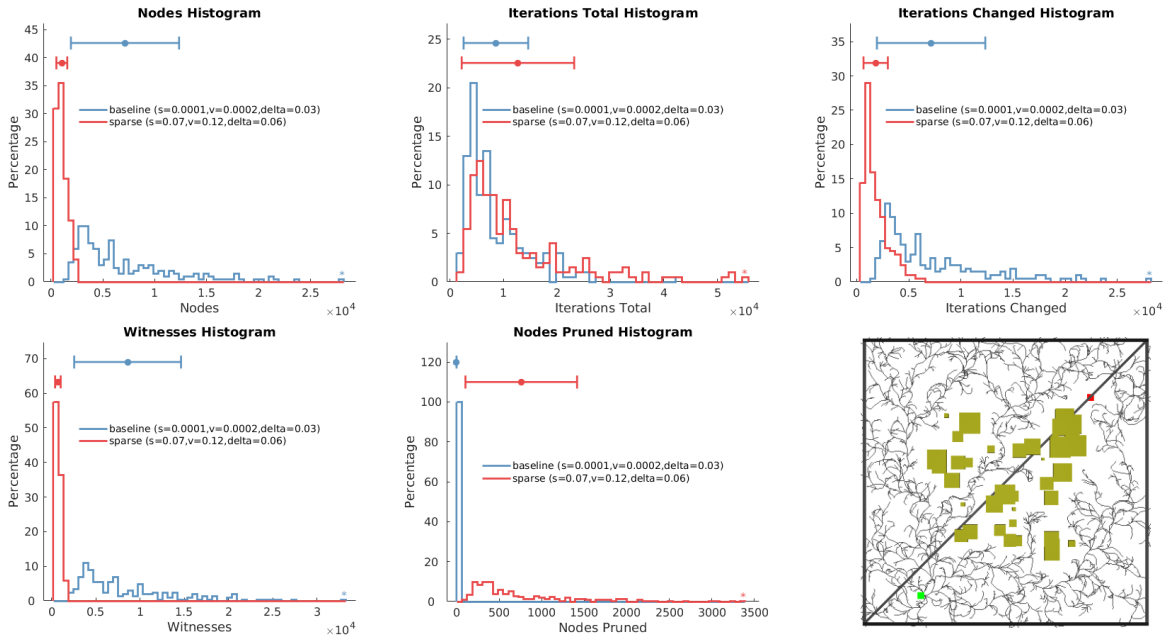

Figure 1: Start: pos=[0.2,0.1], theta=0. Goal: pos[0.8,0.8] +/- [0.01,0.01]. Baseline parameters: $\delta_s = 0.0001, \delta_v = 0.0002, t_{prop} = 0.03 * [0.05, 1]$. Sparse parameters: $\delta_s = 0.07, \delta_v = 0.12, t_{prop} = 0.06 * [0.05, 1]$. t_{prop} corresponds to Monte Carlo propagation duration constraints. Note that all sample scenarios have environment normalized within [0,1] for each dimension. 200 experiments are performed for each type.

9.2 Motion Primitive

Instead of gridding approach, motion primitive is stochastically stored and looked up. It is notable that a ML approach for learning "steering" function of the motion primitive might be possible, but given time constraints this is not considered.

Due to possible degenerate control of the system, it is generally not possible that the motion primitive will steer in a way that guarantees a solution especially when the cost metric is not sufficiently accurate and the current node being propagated is still far away from the goal. Thus, the heuristic that is used in implementation is to use motion primitives only when it is sufficiently close to the goal. Ideally the use of a motion primitive will snap right into the goal region thus completing the very last portion of a feasible trajectory. Below are some examples of "ideal" ways in which motion primitives can help propagate towards the goal in a map taken from Dragon Age.

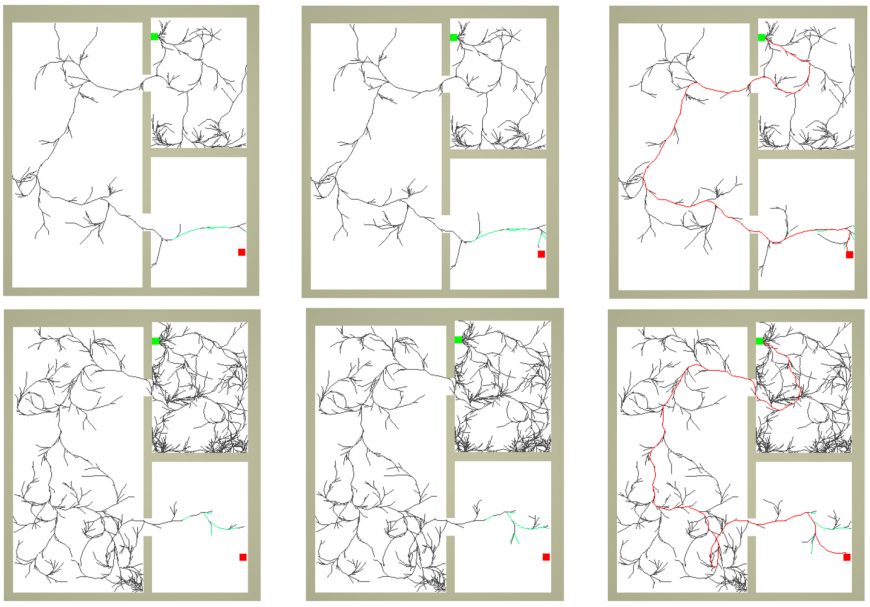

Figure 2: Top and bottom rows, 2 sample solutions with motion primitives and frontier expansion node selection(see Frontier Expansion Node Selection section) enabled. Propagation parameters set to $\delta_s = 0.01, \delta_v = 0.02, t_{prop} = 0.035 * [0.1, 1]$. Motion primitive triggering: within 0.25 L2 distance away from goal in configuration space with a probability of 50%, primitive lookup table capacity: 500, lookup filling after a propagation: fill if lookup table not full or else fill with 15% probability. Note that all sample scenarios have environment normalized within [0,1] for each dimension.

Below shows a comparison of observed metrics with motion primitives enabled and disabled for a Dubins car scenario that records statistics leading up to finding a first feasible path. In general, its effects is small due to constraining trigger condition to a small neighbourhood around the goal.

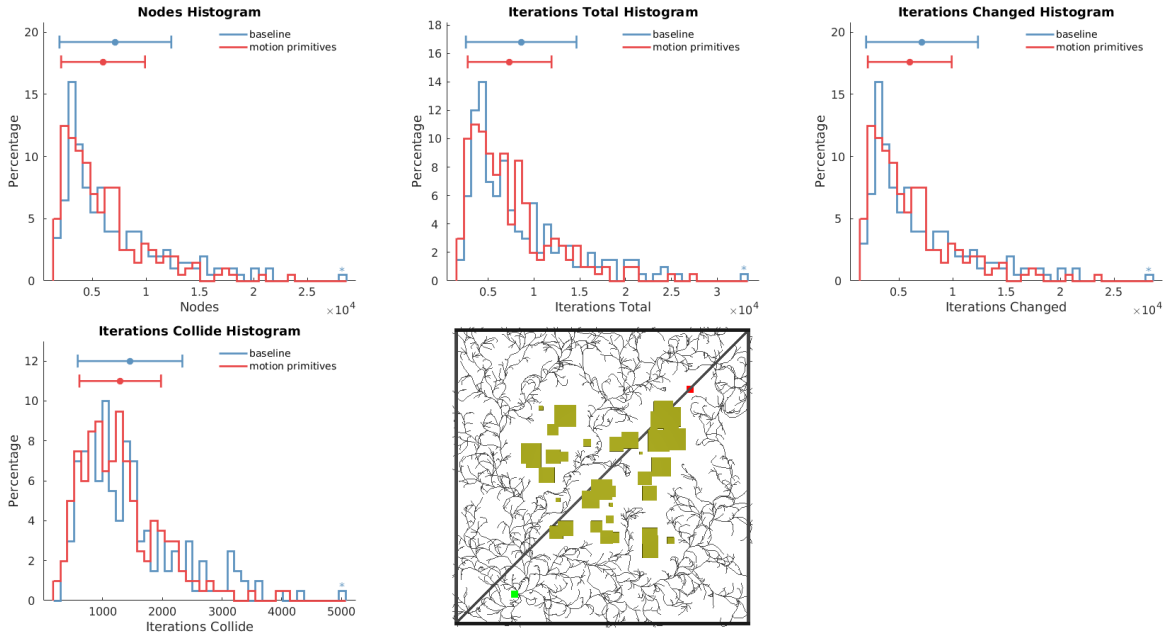

Figure 3: Start: pos=[0.2,0.1], theta=0. Goal: pos[0.8,0.8] +/- [0.01,0.01]. Common parameters: $\delta_s = 0.0001, \delta_v = 0.0002, t_{prop} = 0.03 * [0.05, 1]$. 200 experiments are performed for each type.

9.3 Approximate Nearest Neighbour Query

Nearest neighbour is optionally implemented with a approximate method that is suitable for large node counts (equivalently small witness neighbourhood size). Due to dynamical insertions and deletions of nodes, approximate nearest neighbour is implemented with a connectivity graph that tries to maintain a connected graph with each node having edges connecting to K neighbours such that K is proportional to logarithm of the current total node count. Additional rewiring is performed when nodes are detected with fewer valence connections than a current threshold.

Querying for nearest neighbours samples a number of nodes proportional to square root or logarithm (optional compile-time configurable) of the the existing node count to preliminary obtain a node with the nearest estimate distance. Neighbours of this node is further expanded in order to find nearby nodes with even better closeness metric. This is done recursively to find the best K nodes such that K is proportional to logarithm of the total node count. Fixed point convergence of the best K node set finishes the algorithm and an optimal node within the K candidates is selected as the nearest node.

In practice, the overhead of the approximate method is justified when a large number of nodes (equivalently small neighbourhood size) is expected to exist during search. Empirical observation on a personal laptop shows that linear nearest neighbour search is more effective for node count that is under a few thousand.

9.4 Stochastic Disturbance Injection

Additional disturbance for witness representative replacement is adaptively injected when the current propagation tree is not making enough progress in terms of rate of new witness neighbourhood discoveries. A sliding window keeping record of a brief history of new witness neighbourhood expansion enables probabilistic witness representative replacement when a trajectory is propagated within an existing witness neighbourhood. The reason for this is to allow a range of trajectory classes to be tried by temporally adjusting witness representatives such that the resulting change allows further propagation into new territories.

Contrasting with lack of disturbances, the propagation tree is vulnerable to become stuck in a certain structure without further progress in cluttered environment. This method is found to be quite effective in example scenarios of simple to moderate configuration space complexity. Due to the stochasticity of witness representative changes, this method allows effective exploration using larger neighbourhood tha would still give good change of finding feasible trajectories as time passes.

Below shows a comparison of observed metrics with witness disturbance enabled and disabled for a Dubins car scenario that records statistics leading up to finding a first feasible path. It is observable that the expected iterations to find a first feasible trajectory is slightly lowered and the expected iterations that results in some change is increased.

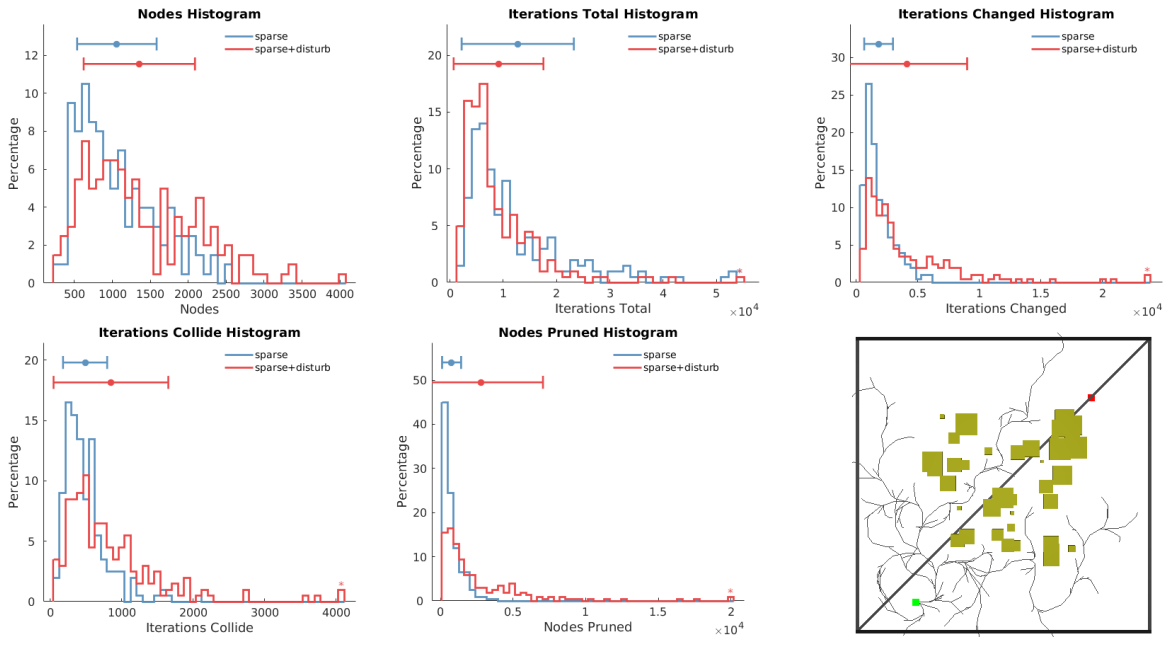

Figure 4: Start: pos=[0.2,0.1], theta=0. Goal: pos[0.8,0.8] +/- [0.01,0.01]. Common parameters: $\delta_s = 0.07, \delta_v = 0.12, t_{prop} = 0.06 * [0.05, 1]$. Disturbance is activated when new witness discovery rate is lower than 10% of recent iterations using a sliding window. 200 experiments are performed for each type.

Below shows a comparison of observed metrics with witness disturbance and motion primitves for a Dubins car scenario.

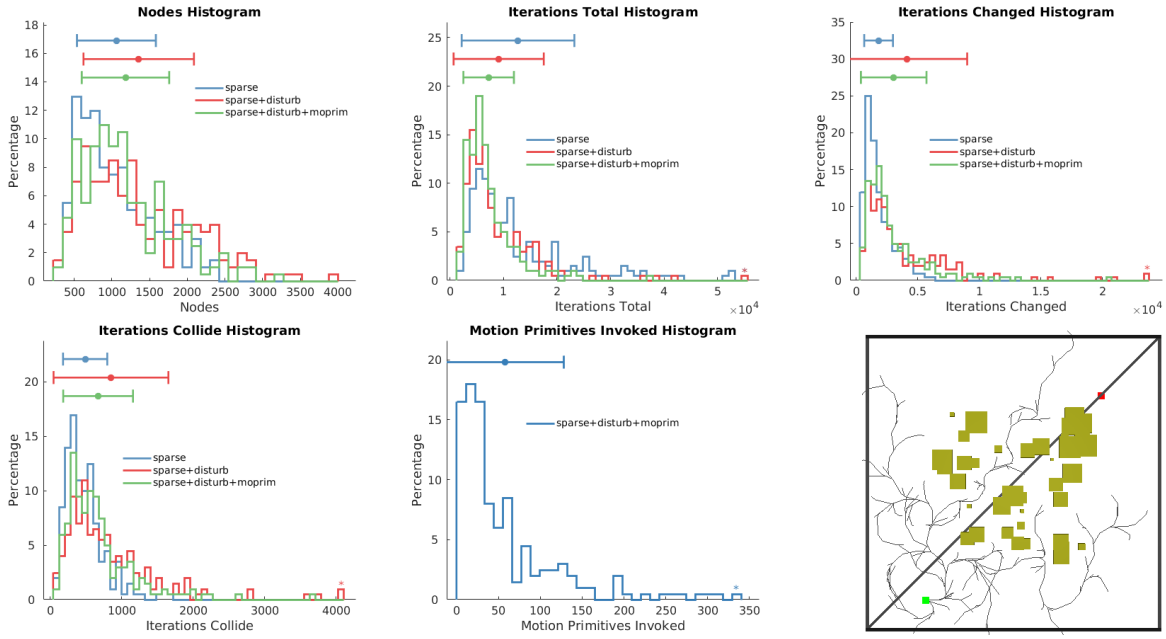

Figure 5: Start: pos=[0.2,0.1], theta=0. Goal: pos[0.8,0.8] +/- [0.01,0.01]. Common parameters: $\delta_s = 0.07, \delta_v = 0.12, t_{prop} = 0.06 * [0.05, 1]$. Disturbance is activated when new witness discovery rate is lower than 10% of recent iterations using a sliding window. 200 experiments are performed for each type.

Another approach that is similar in effect of altering existing propagation structure in hopes of further exploration is through neighbourhood size augmentation such as using a periodic function. This would allow propagation node to deposit in new smaller neighbourhood while still maintaining a level of sparseness due to periodic return to larger neighbourhood size. This approach is not implemented due to extra overhead for sweeping to clean up for nodes when neighbourhood parameter grows back up.

9.5 Importance Sampling

Global optimization is possible using importance sampling, however this might require an infeasible amount of computation for search in the early iterations of importance sampling. For fast local optimization, a first feasible trajectory that is discovered is bootstrapped for an initial sampling distribution that is optimized. The overal algorithm for trajectory optimization is outlined in *alg.* (1).

In the implementation, a sample is defined as a first feasible trajectory found. Rounds of importance sampling are performed with previous sampling distributions and expected to improve in quality with use of a provided fitness function and retaining an elite set of samples with respect to a current fitness cost. Fitness cost is updated using the cost of a sample at some specified quantile of the samples evaluated with respect to the fitness function. Multivariate Gaussian mixture is used for change of sampling distribution and the available parameterizations are thus mean and variance.

Algorithm 1: Trajectory Optimization

input : F(trajectory generator, SST in this case), S(quality metric, assume lower score is better), ϵ(quality convergence constant), ρ(percentile for quality narrowing)
output: quality score, desirable trajectory parameterization

begin
  $trajectory_{initial} \leftarrow$ F.generate()
  quality $\leftarrow \infty$
  G $\leftarrow$ GenerateGaussianComponents(trajectory.waypoints)
  repeat
    //generated n samples; a sample := a first feasible trajectory found by SST algo
    $X[1, .., n] \leftarrow$ (0..n).map(| _ | F.reset(); F.useSampleDistr(G); F.generate())
    $X_{filt} \leftarrow X$.filter(| x | S(x) $\leq$ quality)
    X_{filt}.sortDescendingBy(| x | S(x))
    i $\leftarrow$ floor(len(X_{filt}) * ρ)
    //take quality value of ρ percentile as threshold for next iteration
    $quality_t \leftarrow$ S(X_{filt}[i])
    Elite $\leftarrow X_{filt}$[i+1, ...]
    GS $\leftarrow$ Elite.iter().map(| x | GenerateGaussianComponents(x.waypoints))
    G $\leftarrow$ MergeAndAverage(GS) // update parameterization here(eg: μ), potentially use Cross-Entropy Clustering
  until $|quality - quality_t| \leq \epsilon$, *else* $quality \leftarrow quality_t$;
  return (quality, G.parameterization)

Using the sparseness of SST, all witness regions that are touched by the found feasible trajectory are converted into Gaussian mixture model with the number of components equal to the number of touched witness regions. A sample is defined in this implementation as a first feasible trajectory found and a number of these are generated afresh each time by executing the SST algorithm. Once a sufficient number of samples are generated, they are evaluated according to a quality function ranking the sample and importance sampling framework is applied to update the quality threshold and use samples of sufficient quality to update parameters for the next round of sample generation. The parameters in this implementation are the mean and variances of Gaussian components. For simplicity, variance is set to a constant proportional to witness neighbourhood size of the SST so the Gaussian components are isotropic. In sample generation, the SST algorithm is modified to select nodes for propagation using the Gaussian mixture model from the previous optimization iteration

and this is done in order to generate a prescribed number of samples for the current optimization iteration.

As simplification, only state space is used for importance sampling in the implementation, however this can be extended for (state space, control space) pair such as conditioning control space importance sampling for each state space region, which is in turn also importance sampled.

In this simplified implementation, Gaussian mixture model is generated anew for each new optimization iteration by using the witness neighbourhood of sampled quality trajectories. It may be more stable to incrementally adjust mixtures model of previous iteration with the current quality trajectory as suggested in Cross Entropy tutorial [1]. More sophisticated methods such as Cross-Entropy Clustering for automatic clustering and cluster redundancy elimination for mixtures can be used [2]. These are left for future investigations.

Rounds of importance sampling on the Dubins car in a relatively simple map taken from Dragon Age is shown below.

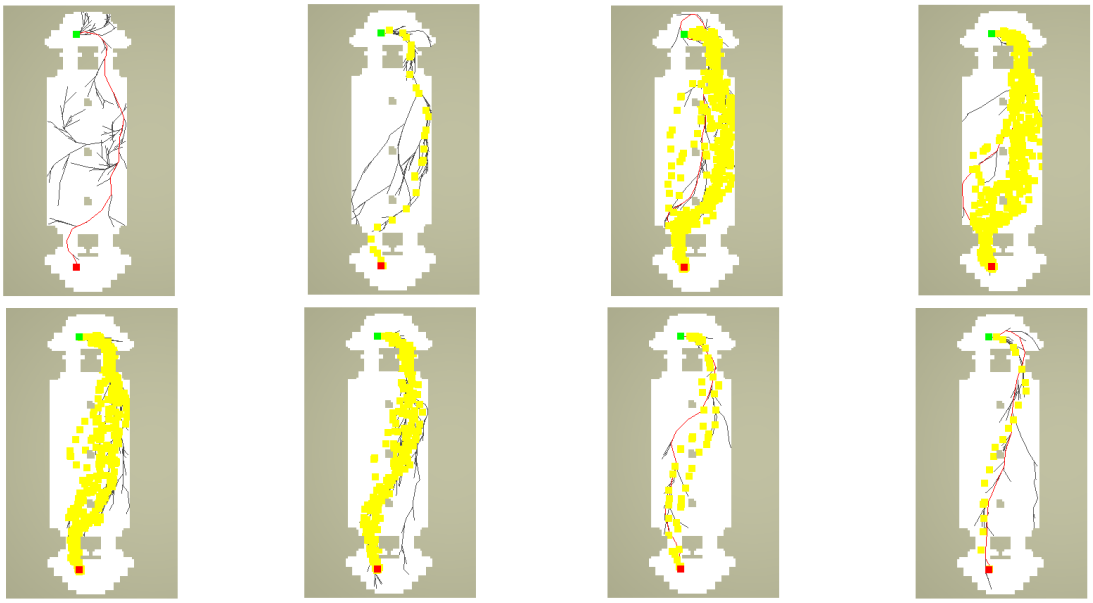

Figure 6: Start: pos=[0.2,0.1,0], theta=0. Goal: pos[0.8,0.8,0] +/- [0.01,0.01,0]. Optimization parameters: number of samples = 20, elite set percentile cutoff = 10%, fitness function = duration of travel, variance of Gaussian components set to a constant of $2\delta_s$ of the SST. $\delta_s = 0.004, \delta_v = 0.008, t_{prop} \in [0.025, 1.0] * 0.07$. Snapshots of initial feasible trajectory found, bootstrapped distribution components with positional means indicated by yellow points, subsequent importance sampling with mixture distributions at iterations: 1, 2, 4, 8, 12 ,16. Quality threshold scores for elite set at iterations: 1(1.008956), 2(0.9829887), 4(0.96761334), 8(0.9411696), 12(0.9142173), 16(0.8817812).

Rounds of importance sampling of the Dubins airplane model in a relatively obstruction-free 3D scenario is shown below.

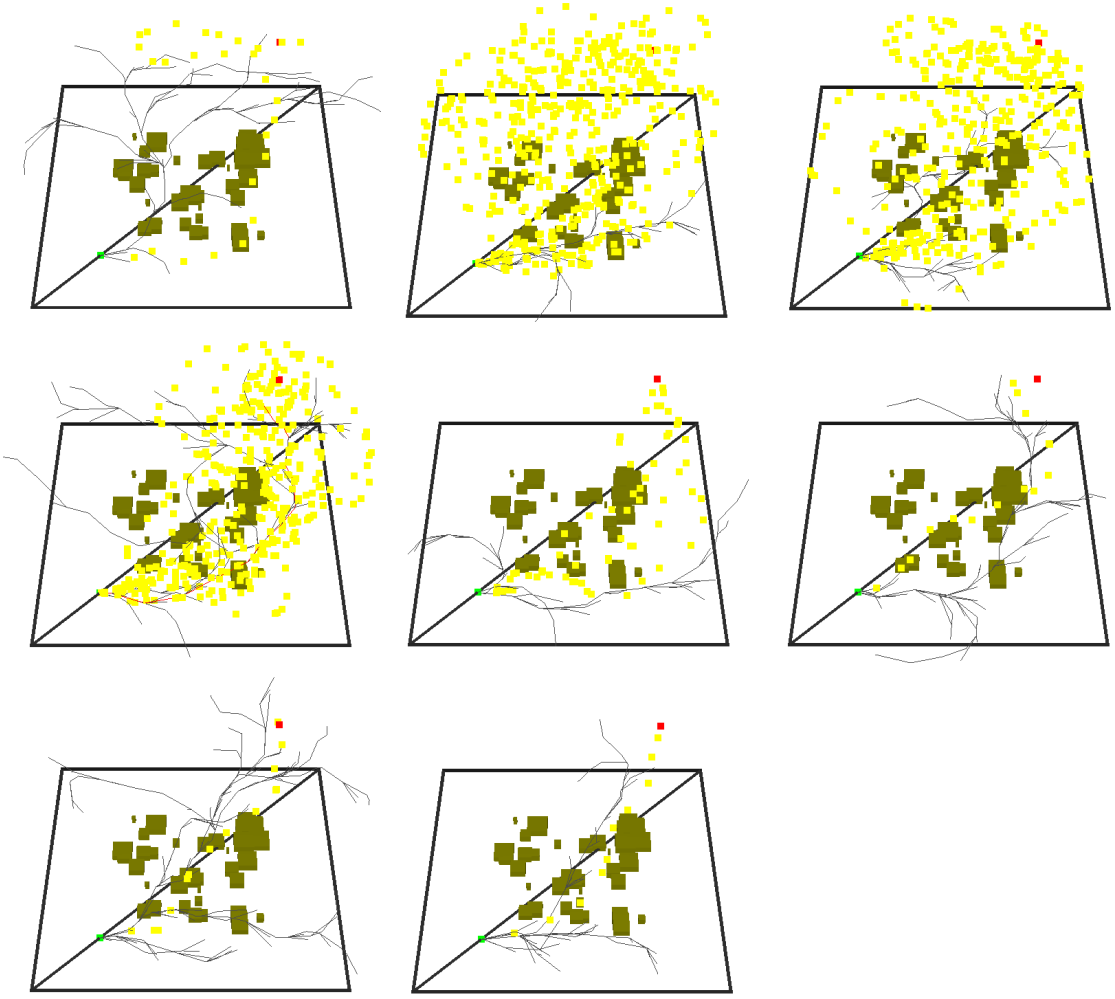

Figure 7: Start: pos=[0.2,0.2,0], theta=0. Goal: pos[0.8,0.8,0.4] +/- [0.075,0.075,0.075]. Frontier node selection with 50% activation is used (see next section) to encourage exploration. Optimization parameters: number of samples = 20, elite set percentile cutoff = 10%, fitness function = duration of travel, variance of Gaussian components set to $1\delta_s$ of the SST. $\delta_s = 0.075, \delta_v = 0.12, t_{prop} \in [0.05, 1.0] * 0.12$. Snapshots of bootstrapped distribution components with positional means indicated by yellow points, subsequent importance sampling with mixture distributions at iterations: 1, 2, 4, 6, 8, 12 ,16. Quality threshold scores for elite set at iterations: 1(2.9550786), 2(2.5031433), 4(2.359734), 6(1.3488877), 8(1.1966971), 12(1.053702), 16(0.95340395).

Rounds of importance sampling of the Dubins airplane model in a moderately obstructed 3D scenario is shown below.

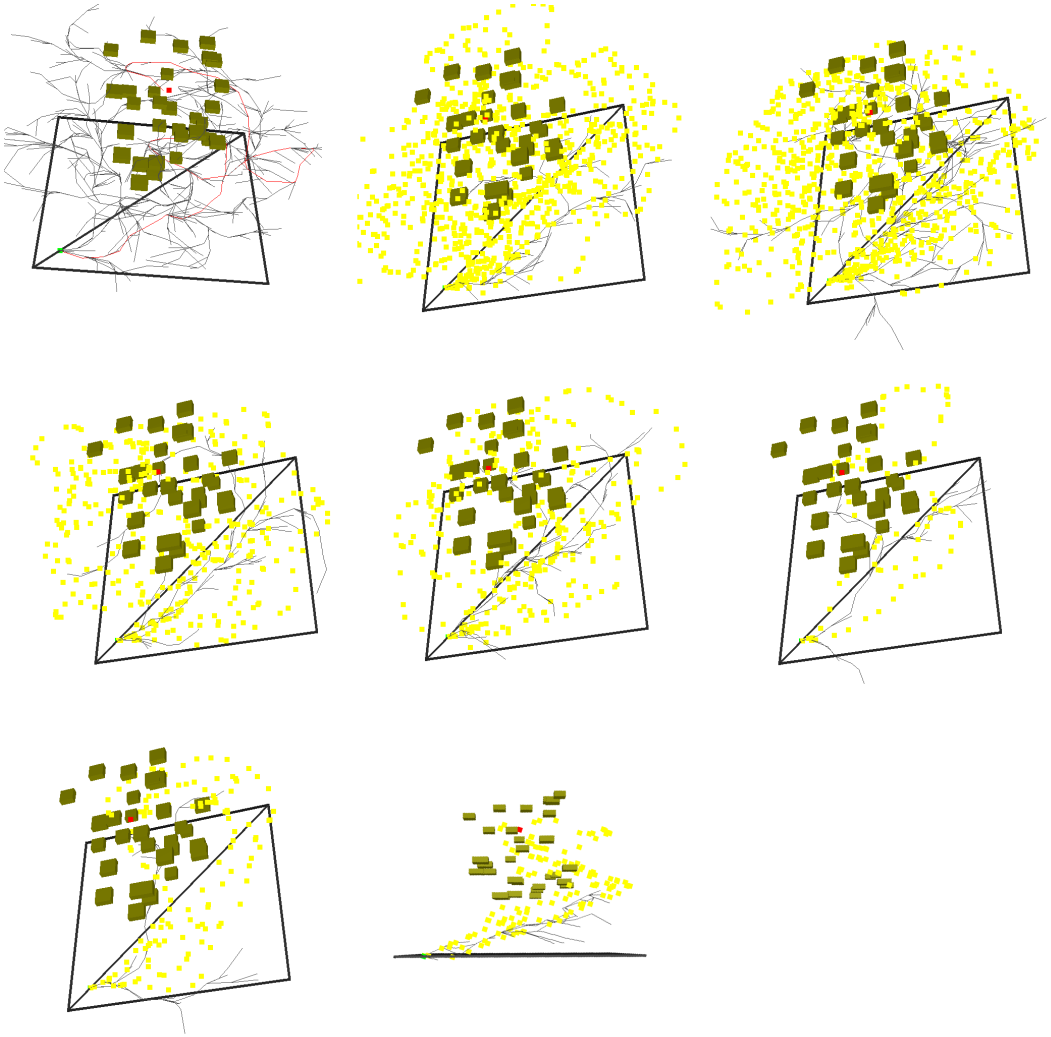

Figure 8: Start: pos=[0.1,0.1,0], theta=0. Goal: pos[0.5,0.5,0.6] +/- [0.075,0.075,0.075]. Frontier node selection with 50% activation is used (see next section) to encourage exploration. Optimization parameters: number of samples = 20, elite set percentile cutoff = 10%, fitness function = duration of travel, variance of Gaussian components set to $1\delta_s$ of the SST. $\delta_s = 0.075, \delta_v = 0.12, t_{prop} \in [0.05, 1.0] * 0.12$. Snapshots of initial feasible trajectory found, subsequent importance sampling with mixture distributions at iterations: 1, 2, 3, 4, 8, 12, and side view of scenario. Quality threshold scores for elite set at iterations: 1(4.366925), 2(4.0489664), 3(3.4321098), 4(3.2385962), 8(2.5732057), 12(2.14029).

Further optimization beyond 12 iterations is observed to have no changed, hence it is around the limit of optimization with the given system and optimization parameters. It is notable that the quality function being optimizated is not convex since the parameterization domain in state space is itself not convex due to certain arbitrary"subspaces" being occupied by the obstructions of the physical space as well as potentially degenerate control regions in a dynamical system.

10 Node Selection & Control Selection

10.1 Node Selection

It is desirable to choose nodes in frontiers for expansion in that they are closer to regions of unexplored space. The idea is to randomly sample multiple locations and find out which sample is the furthest away from already propagated region, then selecting a node that is close to the randomly selected state in hopes of filling space toward the unexplored region.

Local averaging using k-hop neighbourhood of a nearest node to a sample location is used to approximate a score of closeness to the sample. This is integrated with SST algorithm for selecting a node for Monte Carlo propagation. Several random states are selected and a nearest node for each of these are scored based on average distance of its neighbourhood within k hops to the corresponding randomly selected state. The node corresponding to the largest neighbourhood distance score to a randomly selected state is eligible for propagation. By using the existing graph from the approximate nearest neighbour algorithm, k-hop neighbourhood can be identified for a node.

10.2 Control Selection

The control parameter used for a node expansion can be altered by considering control that allow far propagation from a selected node. Multiple control sample points are considered and scored according to propagation length after they are considered to be obstruction free. The effects of node selection and control selection are observed below for a Dubins car scenario (where there is relatively low number of obstructions and no target goal to purely observe expansion properties) that records statistics in the first 4000 total iterations.

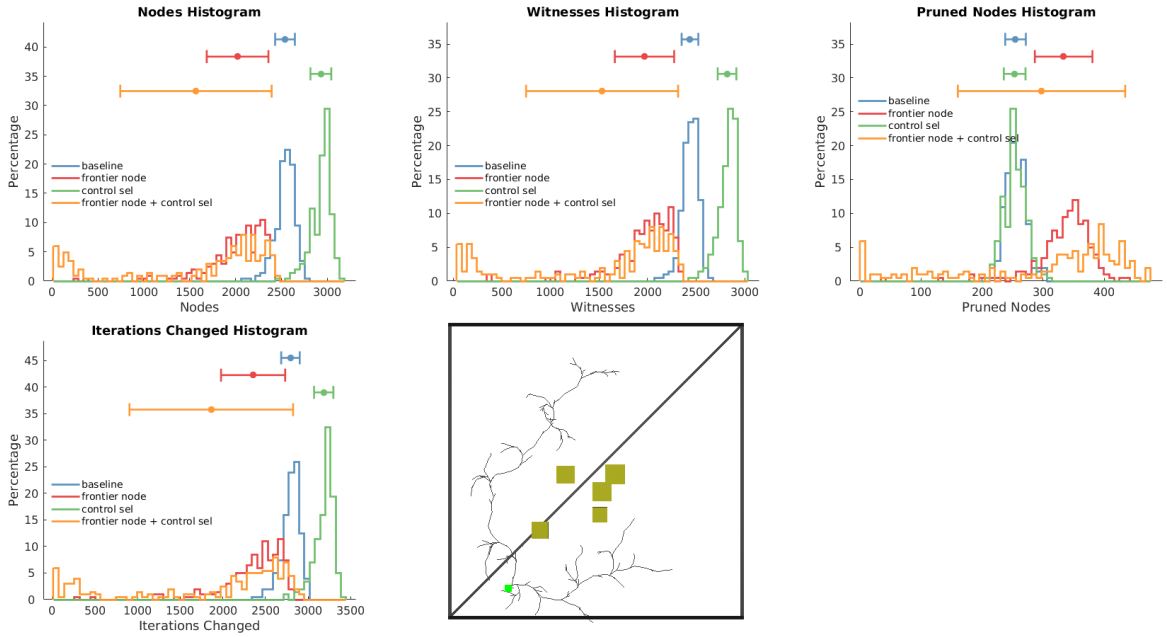

Figure 9: Parameters: $\delta_s = 0.02, \delta_v = 0.04, t_{prop} \in [0.1, 1.0] * 0.03$, total iterations = 4000. Total iterations = iterations changed + iterations no change (witness contention + collision). Number of samples for node selection: 10. Number of samples for control selection: 10. Goal is omitted to observe expansion properties.

It is notable that using frontier node expansion, the amount of expansion measured in number of witness nodes is lower than the baseline that does not use it. It is perhaps related to somehow selecting certain nodes repeatedly but failing to propagate it due to Monte Carlo propagation duration as this is evidence in the amount of increased node pruning. In the case of control selection, propagation is enouraged due to seleciton of a candidate with max propagate distance.

Use of these allows faster expansion in uncluttered space, however it restricts exploration due to lack of knowledge of obstructions. It can be observed that a combination of heuristics can cause adverse effects in the below scenario in which there are failures to expand to different partitions of the configuration space.

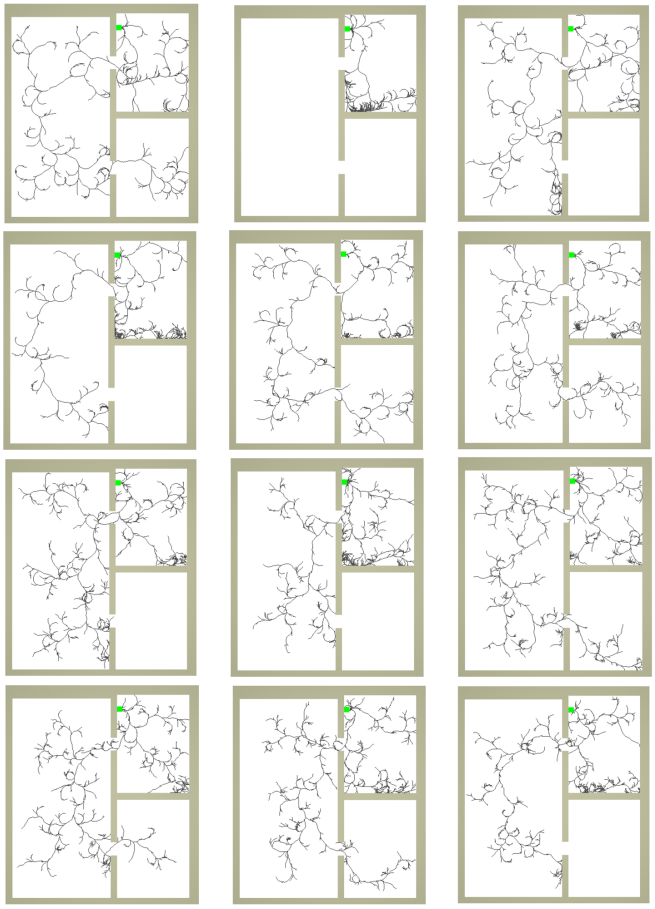

Figure 10: Top row: node selection and control selection applied all of the time. 2nd row: node selection(parameterized to be triggered 100% of time), control selection (parameterized to be triggered 50% of time). 3rd row: node selection (50%), control selection (50%). 4th row: node selection (50%), control selection (25%). Number of nodes for each run ≈ 3200. Propagation parameters set to $\delta_s = 0.003, \delta_v = 0.006, t_{prop} \in [0.0015, 0.015]$.

11 Main Algorithm

Below is an outline of the main SST algorithm implemented.

Algorithm 2: Main SST Algorithm

input : x_{start}, x_{goal}, X, U, δ_s, δ_v, $dist_{ss}$, Q, quality(assume lower is better), dynamics
output: feasible trajectory

```
1  begin
2      traj ← ∅
3      Tree_prop ← ({x_start}, {})
4      Witnesses ← {x_start}
5      repeat
6          x_rand ← selState(X)
7          xs ← getExistingWithinDist( Tree, x_rand, δ_v ) //optional: frontier node sampling
8          if xs = ∅
9              x ← getExistingNearest( Tree, x_rand )
10         else
11             x ← xs.getMinBy(| i | quality(i) )
12         //optional: motion primitive, control sample selection, defaults to Monte Carlo prop
13         u,t ← selPropParam(Q,X,U,x,t)
14         x_prop ← propagate( x, u, t, dynamics )
15         w ← getNodesWithinDist( Witnesses, x_prop, δ_s )
16         if project(x_prop) ∈ Q_free
17             if w = ∅
18                 Witnesses ← Witnesses ∪ w //sliding window to track witness expansion rate
19                 Tree_prop.extend( x, x_prop, u, t )
20                 markActive(x_prop)
21             else if w ≠ ∅
22                 x_repr ← w.representative
23                 if x_repr = ∅ ∨ quality(x_prop) ≤ quality(x_repr)
                       ∨ WitnessExpansionRateTooSlow
24                     markInactive(x_repr)
25                     Tree_prop.extend( x, x_prop, u, t )
26                     markActive(x_prop)
27                     w.representative ← x_prop
28                     x_prune ← x_repr
29                     while x_prune ≠ ∅ ∧ ¬isActive(x_prune) ∧ ¬hasDescendents(x_prune) ∧
                           ¬isRoot(x_prune)
30                         x_p ← x_prune.parent
31                         Tree_prop ← Tree_prop \ Edge_{x_p,x_prune}
32                         x_prune ← x_p
33                 if x_prop ∈ x_goal
34                     traj ← π_{x_start,u_1,...,u_n,x_prop}
35     until traj = ∅;
36     return traj
```

12 Summary

Several ideas that are relevant in sample based motion planning are explored in the literature and this is further realized in a hybrid implementation with benchmarks that empirically examine their effects on several quantified metrics in different scenarios.

In particular, there exist tradeoffs between sparseness efficiencies and the ability to explore in depth in complex configuration spaces. Motion primitives also provide some degree of sparseness by precomputing for a subset of the available solution space in a local grid and presenting these discretized primitives for discrete algorithms. However, this efficiency is counterbalanced by constraining to the solved canonical trajectory classes and may not provide a trajectory that is feasible or in sufficient quality even given infinite run-time compute resource. Finally, importance sampling enables flexible optimization for multi-extremal problems while allowing reparameterization techniques, however, efficiencies of this method depends on sample generation techniques in general and thus may take a lot of resource for sampling especially in the early optimization iterations.

Additional attempts of optimization are investigated. Disturbances are injected when new witness expansion rate falls below a threshold in order to stimulate propagation tree rewiring so that altered structure allows further movement in already explore spaces. Approximate nearest neighbour using graph based approach is implemented using authors approach, however it seems the overhead outweighs its benefits especially at low node counts which is the ideal regime that a sparse algorithm would like to be in.

Furthermore, biasing node selection for expansion is investigated by stochastically sampling multiple states and choosing a nearby node that is most likely to be near spaces that are not densely populated with existing nodes. This uses graph based approximate nearest neighbour to obtain a k-hop neighbourhood for distance summary to a target sample. Propagation length is investigated by selecting a propagation duration that tries to maximize propagation distance from a selected node. This approach seems to work well to cover large spaces but may fail to expand with varied trajectory classes in cluttered spaces. A combination of the node selection and propagation length are experimented and these have non-desirable effects especially in cluttered configuration spaces.

Other improvements such as automated clustering to Gaussian mixture model for trajectory optimization and better steering function prediction via machine learning techniques would be a great enhancement to the existing algorithms and these are topics for future exploration.

13 Code

Code implementation can be found at `https://github.com/clearlycloudy/sample_planning`.

14 References

[1] PT de Boer et al. "A Tutorial on the Cross-Entropy Method". In: *Annals of Operations Reesearch* (2005).

[2] P Spurek J Tabor. "Cross-entropy clustering". In: *Pattern Recognition* (2014).

[3] M Kobilarov. "Cross-entropy motion planning". In: *The International Journal of Robotics Research* (2012).

[4] Y Li, Z Littlefield, and KE Bekris. "Sparse Methods for Efficient Asymptotically Optimal Kinodynamic Planning". In: *Workshop on the Algorithmic Foundations of Robotics (WAFR)* (2014).

[5] B Sakcak et al. "Sampling-based optimal kinodynamic planning with motion primitives". In: *Autonomous Robots* (2018).