BadgerWorks Lectures

Sampling-based Path Planning Primer

Kostas Alexis

AUTONOMOUS ROBOTS LAB

Course Outline

- **A brief introduction**
  - What path planning is about and some basic notes
- **Collision-free Navigation through PRM, RRT and RRT***
  - An intro to some basic methods for collision-free motion planning
- **Unknown Area Exploration Path Planning**
  - Methods to explore unknown environments
- **Unknown Area Exploration Path Planning *under Uncertainty***
  - Methods to explore unknown environments while maintaining localizability
- **Integrated Task and Motion Planning**
  - And a bit of introduction on linear temporal logic
- **Thoughts for a Curiosity-aware Exploration planner**
  - Thoughts for a next path planning contribution

How do I plan my motion and actions?

What is Path Planning rougly?

- **Determining the robot path based on a set of goals and objectives, a set of robot constraints and subject to a representation and map of the environment.**

Trends in Robotics/Motion Planning

- **Classical Robotics (mid-70's)**
  - Exact models
  - No sensing necessary
- **Reactive Paradigm (mid-80's)**
  - No models
  - Relies heavily on good sensing
- **Hybrids (since 90's)**
  - Model-based at higher levels
  - Reactive at lower levels
- **Probabilistic Robotics (since mid-90's)**
  - Seamless integration of models and sensing
  - Inaccurate models, inaccurate sensors

Notes from G. Hager http://voronoi.sbp.ri.cmu.edu/~motion

Overview of Concepts

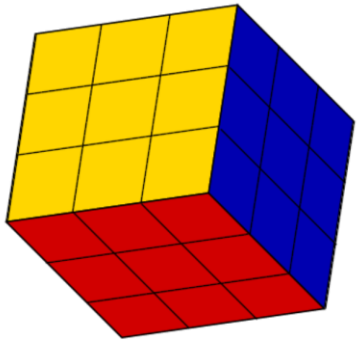
- Planning Tasks
  - Navigation
  - Coverage
  - Exploration
  - Target follow
  - Localization
  - Mapping

- Properties of the Robot
  - Degrees of Freedom
  - Non/Holonomic
  - Kinematic vs Dynamic

- Properties of the Environment
  - Static / Dynamic
  - Deterministic / Uncertain
  - Known / Unknown

- Algorithmic Properties
  - Optimality
  - Computational Cost
  - Completeness
    - Resolution completeness
    - Probabilistic completeness
  - Online vs Offline
  - Sensor-based or not
  - Feedback-based or not

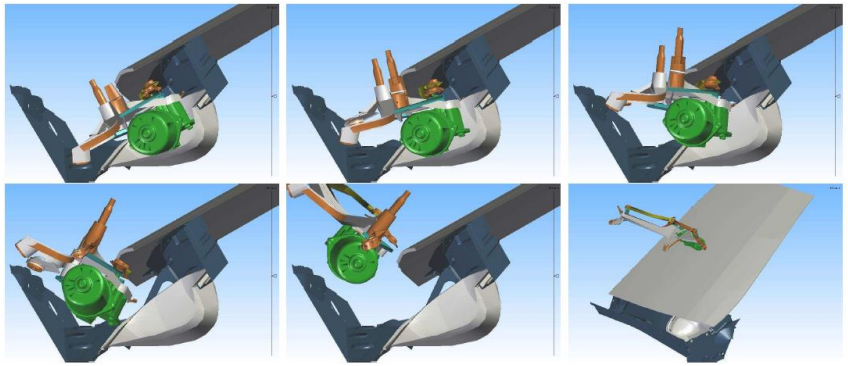
Indicative Examples

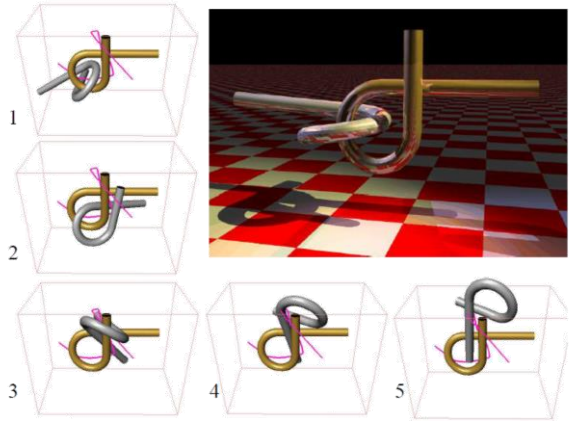


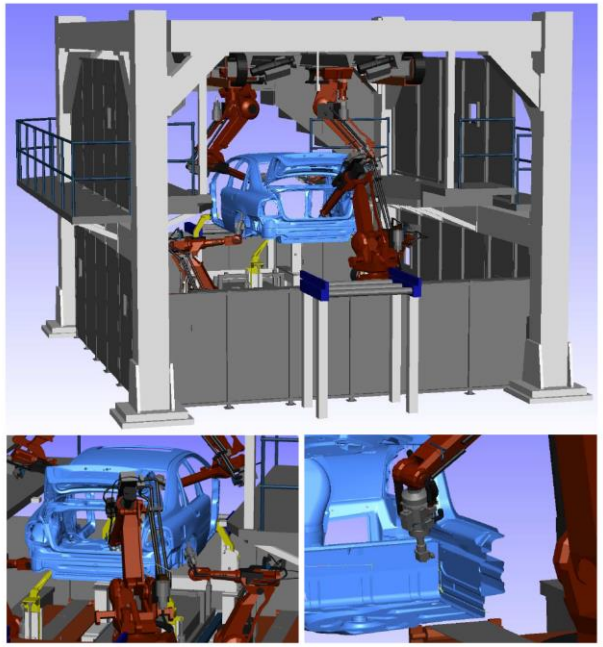
(a)

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

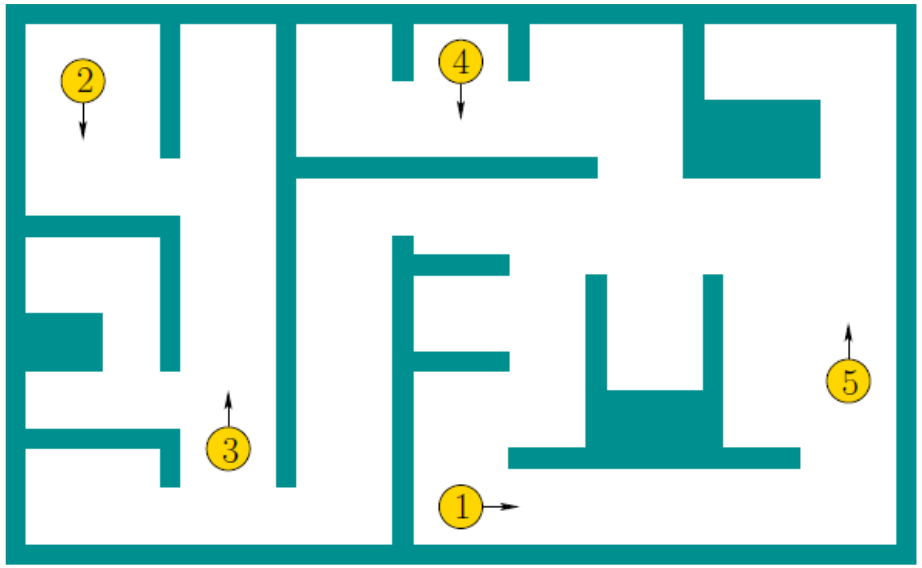
(b)

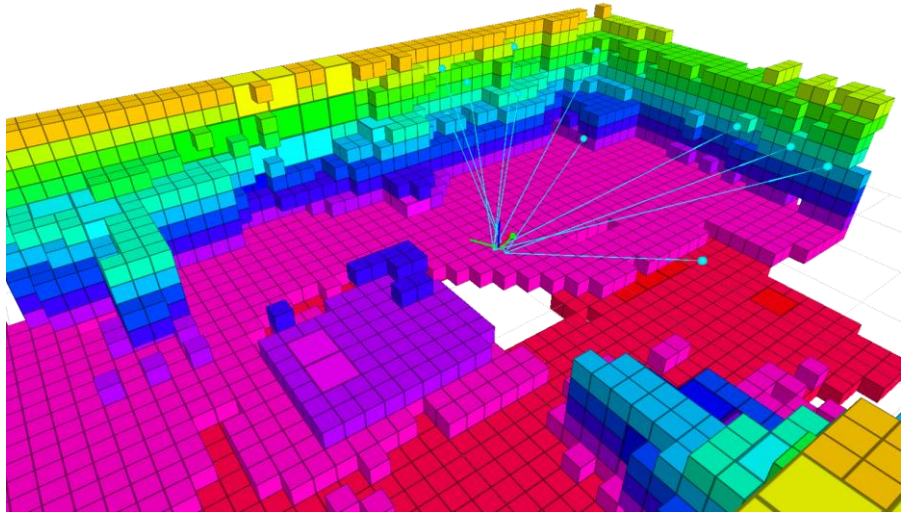


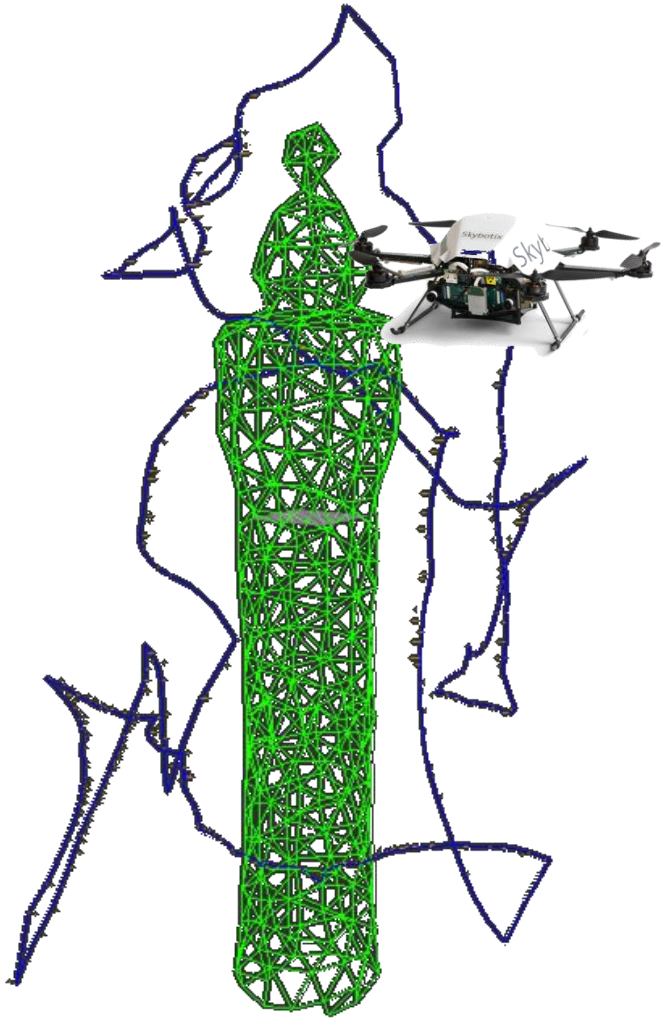




Example of a world (and a robot)



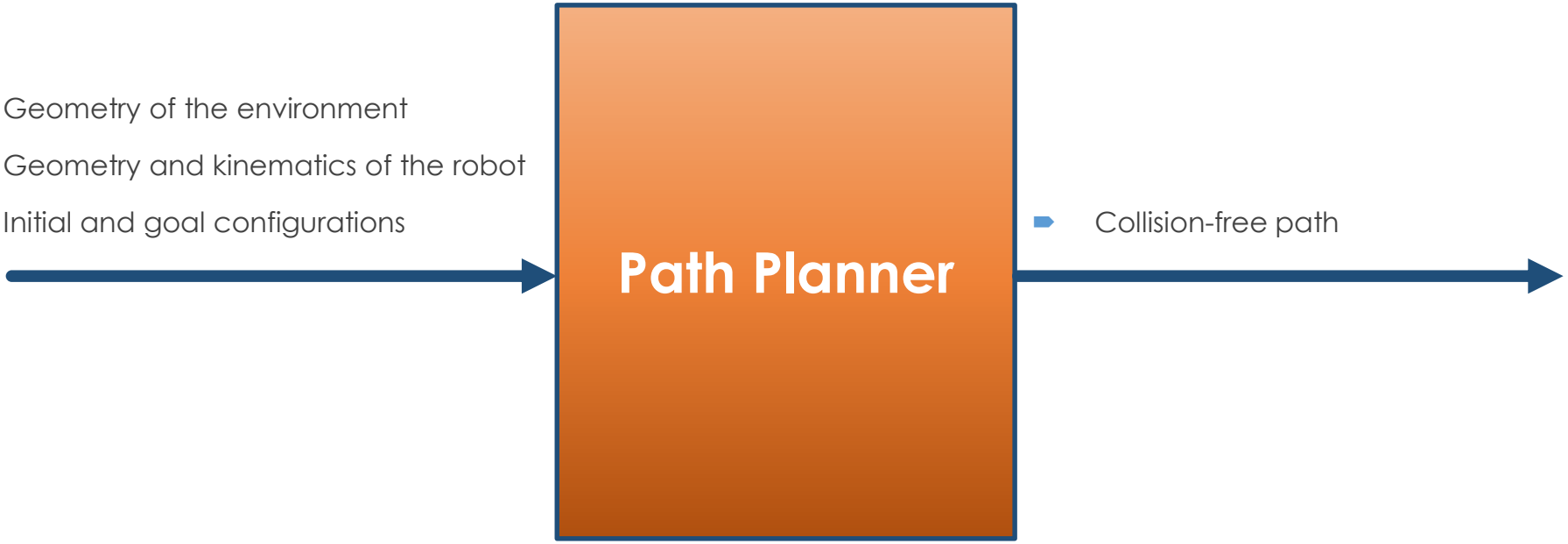




Fundamental Problem of Path Planning

- **Problem Statement:**
  - Compute a continuous sequence of collision-free robot configurations connecting the initial and goal configurations.

- Geometry of the environment
- Geometry and kinematics of the robot
- Initial and goal configurations



- Collision-free path

Fundamental Problem of Path Planning

- **Problem Statement:**
  - Compute a continuous sequence of collision-free robot configurations connecting the initial andgoal configurations.
- **Motion Planning Statement for collision-free navigation**
  - If W denotes the robot's workspace, and WO_i denotes the i-th obstacle, then the robot's free space, W_{free}, is defined as: $W_{free} = W - (\cup WO_i)$ and a path c is $c: [0,1] \rightarrow W_{free}$, where $c(0)$ is the starting configuration q_{start} and $c(1)$ is the goal configuration q_{goal}.

ETH zürich

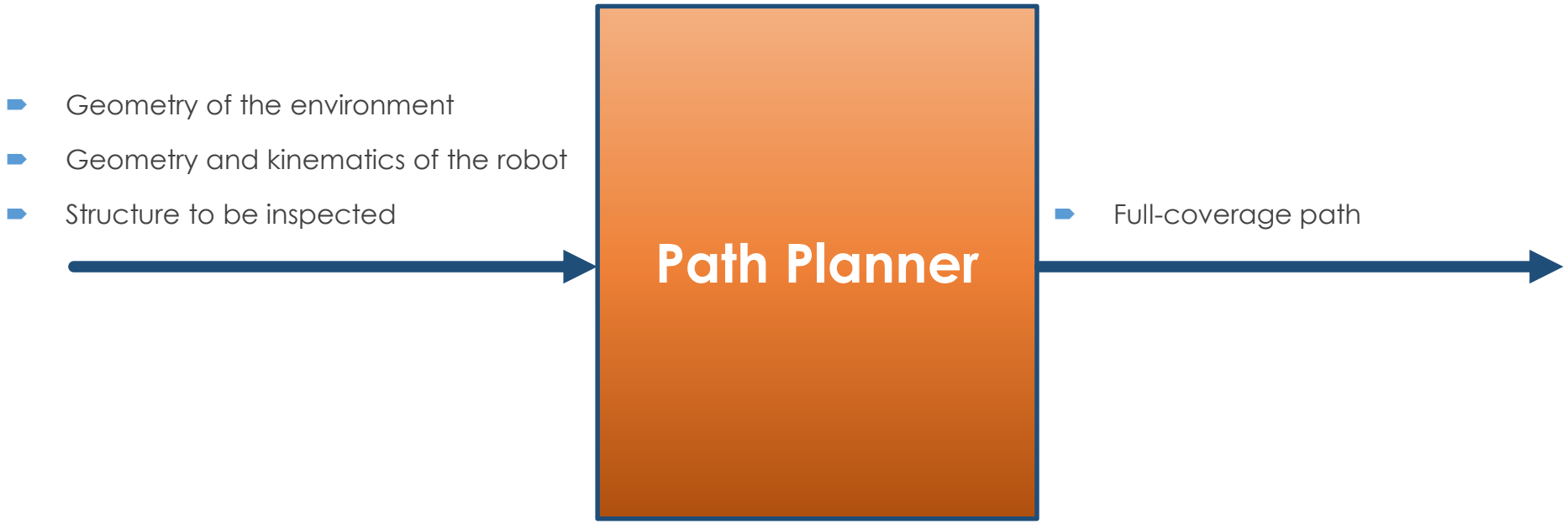
Autonomous Systems Lab

Continuous-Time Trajectory Optimization for Online UAV Replanning

Helen Oleynikova, Michael Burri, Zachary Taylor, Juan Nieto, Roland Siegwart and Enric Galceran

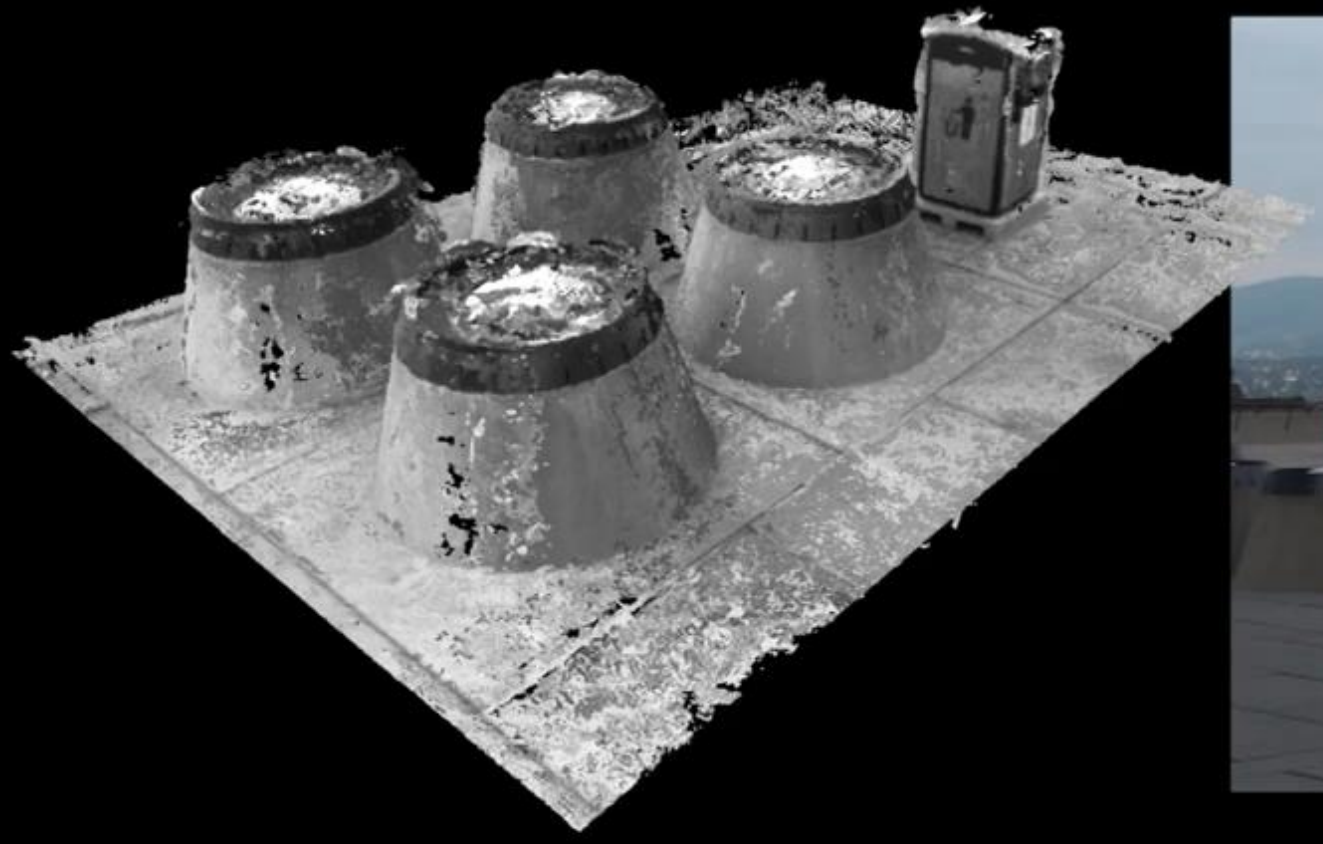
Coverage Path Planning Problem

- **Problem Statement:**
  - Consider a 3D structure to be inspected and a system with its dynamics and constraints and an integrated sensor, the limitations of which have to be respected. The 3D structure to be inspected is represented with a geometric form and the goal is to calculate a path that provides the set of camera viewpoints that ensure full coverage subject to the constraints of the robot and the environment.



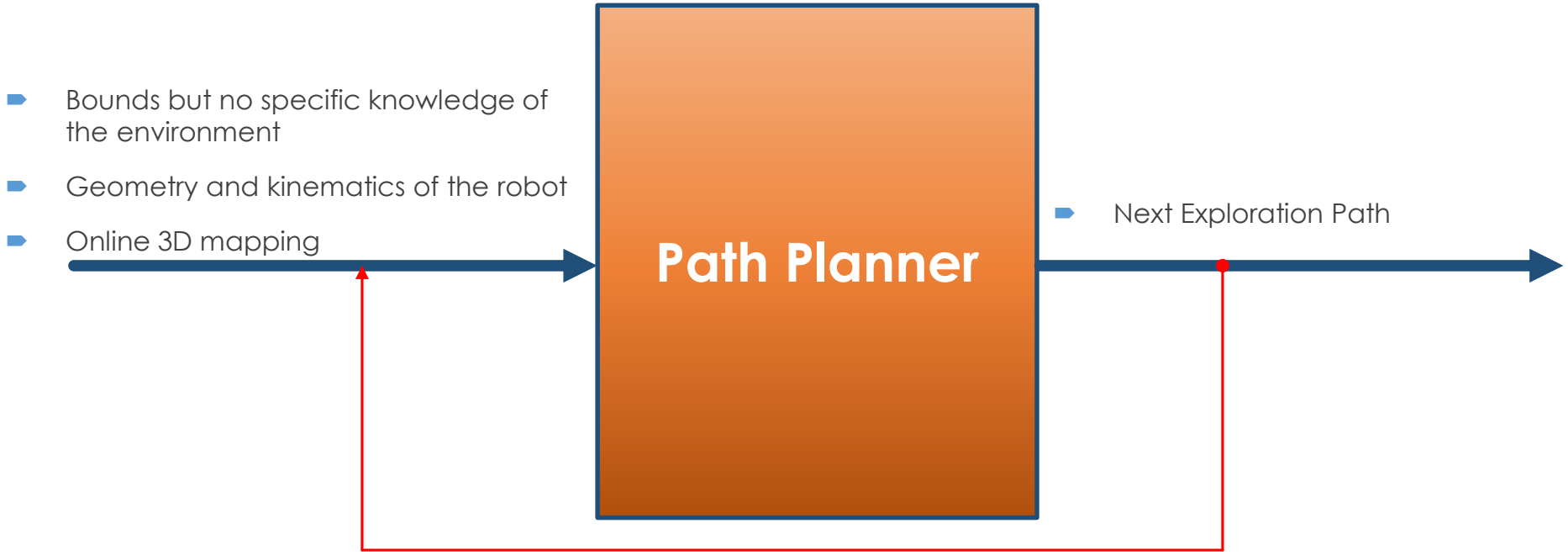
Three-dimensional Coverage Path Planning via Viewpoint Resampling and Tour Optimization using Aerial Robots

A. Bircher, K. Alexis, M. Kamel, M. Burri, P. Oettershagen, S. Omari, T. Mantel, R. Siegwart



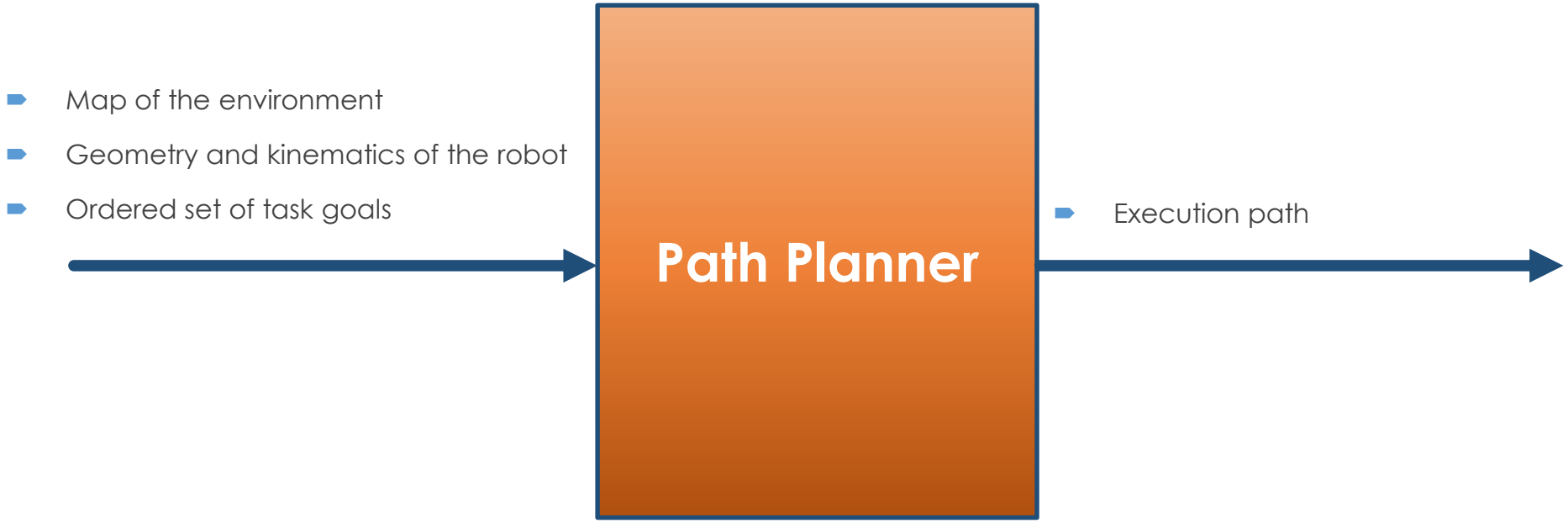
Exploration of Unknown Environments

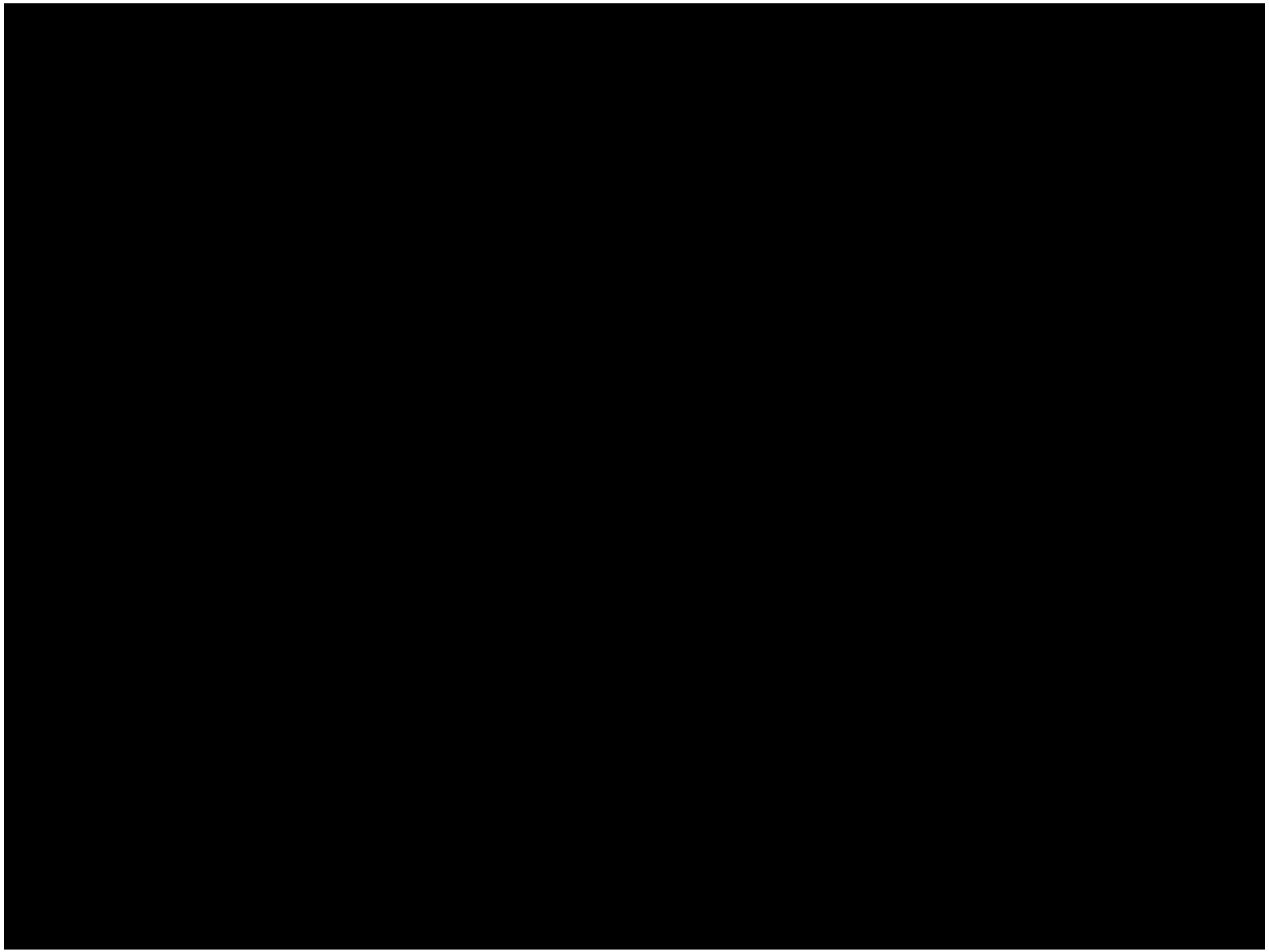
- **Problem Statement:**
  - Consider a 3D bounded space V unknown to the robot. The goal of the autonomous exploration planner is to determine which parts of the initially unmapped space are free V_{free} or occupied V_{occ} and essentially derive the 3D geometric model of the world.



Task and Motion Planning

- **Problem Statement:**
  - Execute a complex, multi-objective mission that contains an ordered set of tasks and implies the derivation of the plan per task.





BadgerWorks Lectures

Collision-free Motion Planning

Kostas Alexis

AUTONOMOUS ROBOTS LAB

The motion planning problem

- Consider a dynamical control system defined by an ODE of the form:

$$\frac{dx}{dt} = f(x, u), x(0) = x_{init} \ (1)$$

- Where is x the state, u is the control.
- Given an obstacle set X_{obs}, and a goal set X_{goal}, the objective of the motion planning problem is to find, if it exists, a control signal u such that the solution of (1) satisfies $x(t) \notin X_{obs}$ for all $t \in R^+$, and $x(t) \in X_{goal}$ for all $t > T$, for some finite $T \geq 0$. Return failure if no such control signal exists.

- Basic problem in robotics
- Provably hard: a basic version of it (the Generalized Piano Mover's problem) is known to be PSPACE-hard.

Motion planning in practice

- Many methods have been proposed to solve such problems in practical applications:
  - **Algebraic planners:** Explicit representation of obstacles. Use complicated algebra (visibility computations/projections) to find the path. Complete, but impractical.
  - **Discretization + graph search:** Analytic/grid-based methods do not scale well to high dimensions. Graph search methods (A*, D*, etc.) can be sensitive to graph size. Resolution complete.
  - **Potential fields/navigation functions:** Virtual attractive forces towards the goal, repulsive forces away from the obstacles. No completeness guarantees; unless "navigation functions" are available – very hard to compute in general.

- These algorithms achieve tractability by foregoing completeness altogether, or achieving weaker forms of it, e.g. resolution completeness.

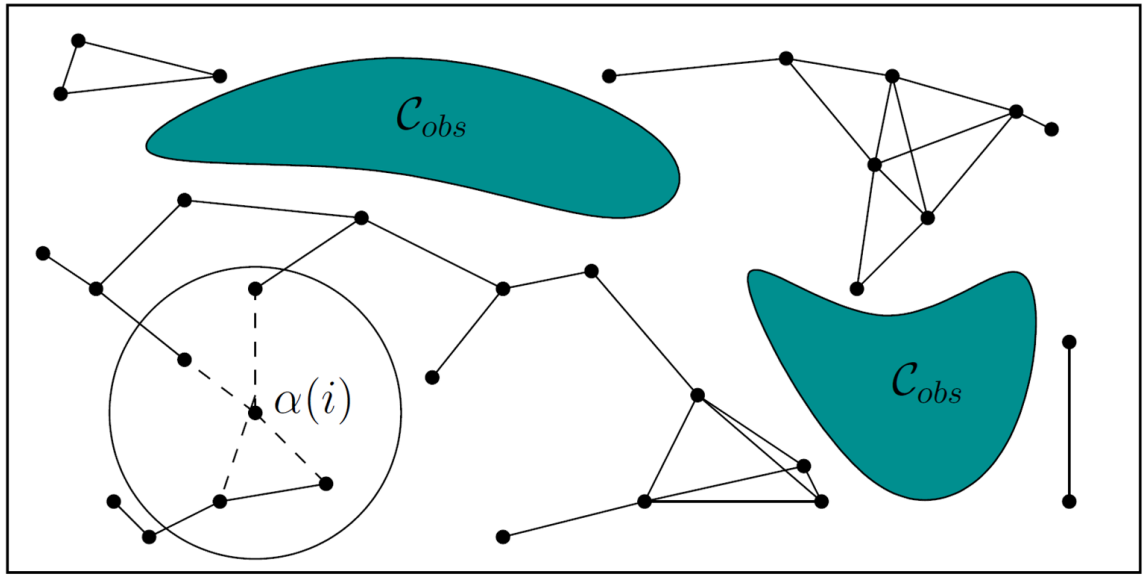
Sampling-based algorithms

- A proposed class of motion planning algorithms that has been very successful in practice is based on (batch or incremental) ***sampling*** methods: solutions are computed based on samples drawn from some distribution. Sampling algorithms retain some form of completeness, e.g., probabilistic or resolution completeness.
- **Incremental sampling** methods are particularly attractive:
  - Incremental sampling algorithms lend themselves easily to real-time, on-line implementation.
  - Applicable to very generic dynamical systems.
  - Do not require the explicit enumeration of constraints.
  - Adaptively multi-resolution methods (i.e. make your own grid as you go along, up to the necessary resolution).

Probabilistic RoadMaps (PRM)

- Introduced by Kavraki and Latombe in 1994.
- Mainly geared towards "multi-query" motion planning problems.
- **Idea:** build (offline) a graph (i.e., the roadmap) representing the "connectivity" of the environment – use this roadmap to find paths quickly at run-time.
- **Learning/pre-processing phase:**
  - Sample n points from $X_{free} = [0,1]^d \backslash X_{obs}$.
  - Try to connect these points using a fast "local planner" (e.g. ignore obstacles).
  - If connection successful (i.e. no collisions), add an edge between the points.
- **At run-time:**
  - Connect the start and end goal to the closest nodes in the roadmap.
  - Find a path on the roadmap.
- *First planner ever to demonstrate the ability to solve generic planning problems in > 4-5 dimensions!*

Probabilistic RoadMap example



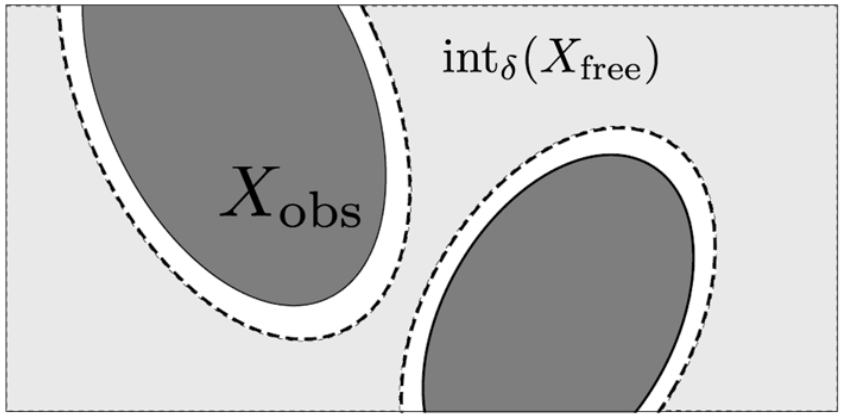
- **"Practical" algorithm:**
  - Incremental construction.
  - Connects points within a radius r, starting from "closest" ones.
  - Do not attempt to connect points that are already on the same connected component of the RPM.
- *What kind of properties does this algorithm have? Will it find a solution if there is one? Will that be an optimal solution? What is the complexity of the algorithm?*

Probabilistic Completeness

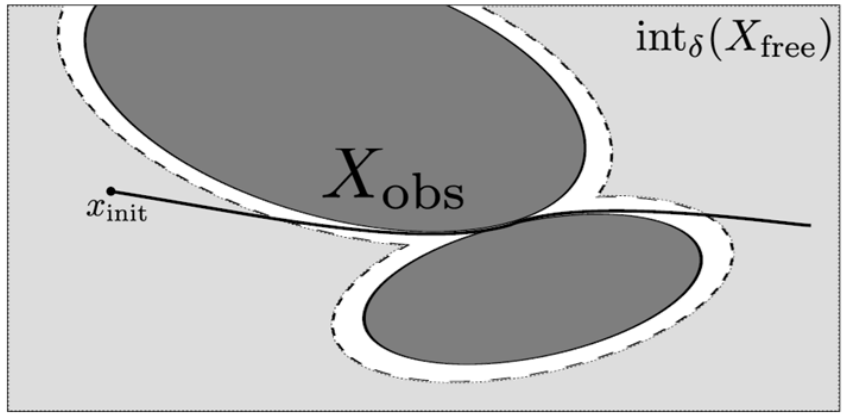
- Definition – Probabilistic Completeness:
- An algorithm ALG is probabilistically complete if, for any robustly feasible motion planning problem defined by $P = (X_{free}, x_{init}, X_{goal})$, then:

$$\lim_{N \to \infty} \Pr(ALG\ returns\ a\ solution\ P) = 1$$

- A "relaxed" notion of completeness
  - Applicable to motion planning problems with a robust solution. A robust solution remains a valid solution even when the obstacles are "dilated" by small small δ.



robust



NOT robust

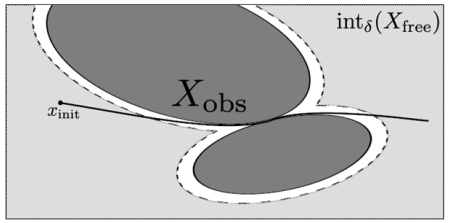
Asymptotic Optimality

- Definition – Asymptotic Optimality:
- An algorithm ALG is asymptotically optimal if, for any motion planning problem defined by $P = (X_{free}, x_{init}, X_{goal})$ and function c that admit a robust optimal solution with finite cost c^*,

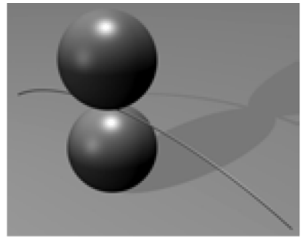
$$P\left(\left\{\lim_{i\to\infty} Y_i^{ALG} = c^*\right\}\right) = 1$$

- The function c associates to each path σ a non-negative $c(\sigma)$, e.g. $c(\sigma) = \int_\sigma \mathrm{X}(s)ds$
  - The definition is applicable to optimal motion planning problems with a robust optimal solution. A robust optimal solution is such that it can be obtained as a limit of robust (non-optimal) solutions.

NOT robust



robust



Complexity

- How can we measure complexity for an algorithm that does not necessarily terminate?
- Treat the number of samples as the "size of the input" (Everything else stays the same).
- Also, we analyze complexity per sample: how much effort (time/memory) is needed to process one sample.
- Useful for comparison of sampling-based algorithms.
- Cannot compare with deterministic, complete algorithms.

Simple PRM (sPRM)

sPRM Algorithm

$V \leftarrow \{x_{init}\} \cup \{SampleFree_i\}_{i=1,,\ldots,N-1}; E \leftarrow 0;$
foreach $v \in V$ **do**:
    $U \leftarrow Near(G = (V, E), v, r) \backslash \{v\};$
    foreach $u \in U$ **do**:
        if $CollisionFree(v, u)$ **then** $E \leftarrow E \cup \{(v, u)\}, (u, v)\}$
return $G = (V, E);$

- The simplified version of the PRM algorithm has been shown to be probabilistically complete.
- Moreover, the probability of success goes to 1 exponentially fast, if the environment satisfies "good visibility" conditions.
- New key concept: combinatorial complexity vs "visibility".

Operation Concept of PRM

- **Learning Phase:**
  - Initially empty graph
  - A configuration is randomly chosen
  - If this configuration lies in the free space – add
  - Repeat until N vertices are added
  - For each new configuration select k-closest neighbors
  - Local planner adds vertex q to q' – IF planner is successful then edge is added
- **Finding the path:**
  - Given starting vertrex ginit and end vertex qgoal
  - Find k-nearest neighbors of q_{init} and q_{goal} in roadmap, plan local path Δ
  - Roadmap graph may have disconnected components
  - Need to find connections from q_{init}, q_{goal} to same component
  - Once on roadmap, use Dijkstra

Remarks on PRM

- sPRM is probabilistically complete and asymptotically optimal.
- PRM is probabilistically complete but NOT asymptotically optimal.
- Complexity for N samples: $O(N^2)$.
- Practical complexity-reduction tricks:
  - k-nearest neighbors: connect to the k nearest neighbors. Complexity $O(NlogN)$. (Finding nearest neighbors takes $logN$ time.)
  - Bounded degree: connect at most k nearest neighbors among those within radius r.
  - Variable radius: change the connection radius r as a function of N. How?

Rapidly-exploring Random Trees

- Introduced by LaValle and Kuffner in 1998.
- Appropriate for single-query planning problems.
- **Idea:** build (*online*) a tree, exploring the region of the state space that can be reached from the initial condition.

- **At each step:** sample one point from X_{free}, and try to connect it to the closest vertex in the tree.
  - Very effective in practice but presents "Voronoi bias".

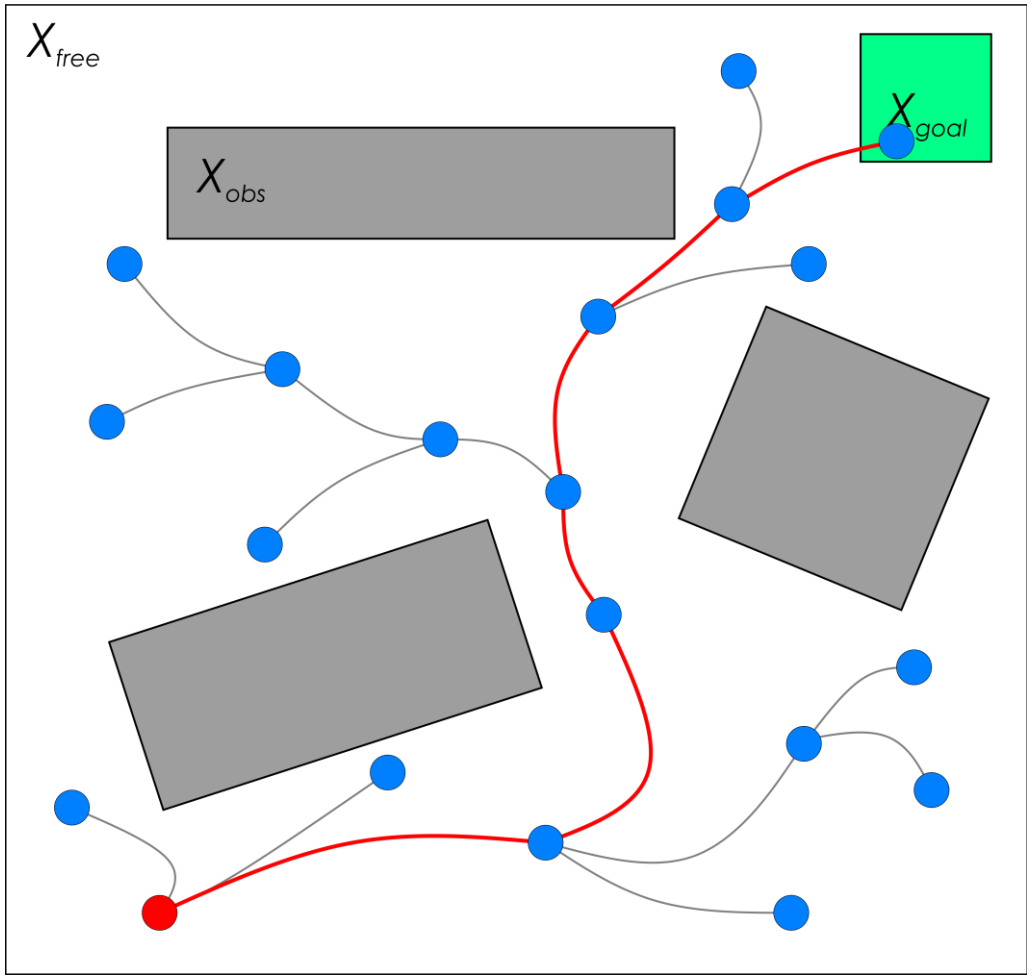
Rapidly-exploring Random Trees

RRT

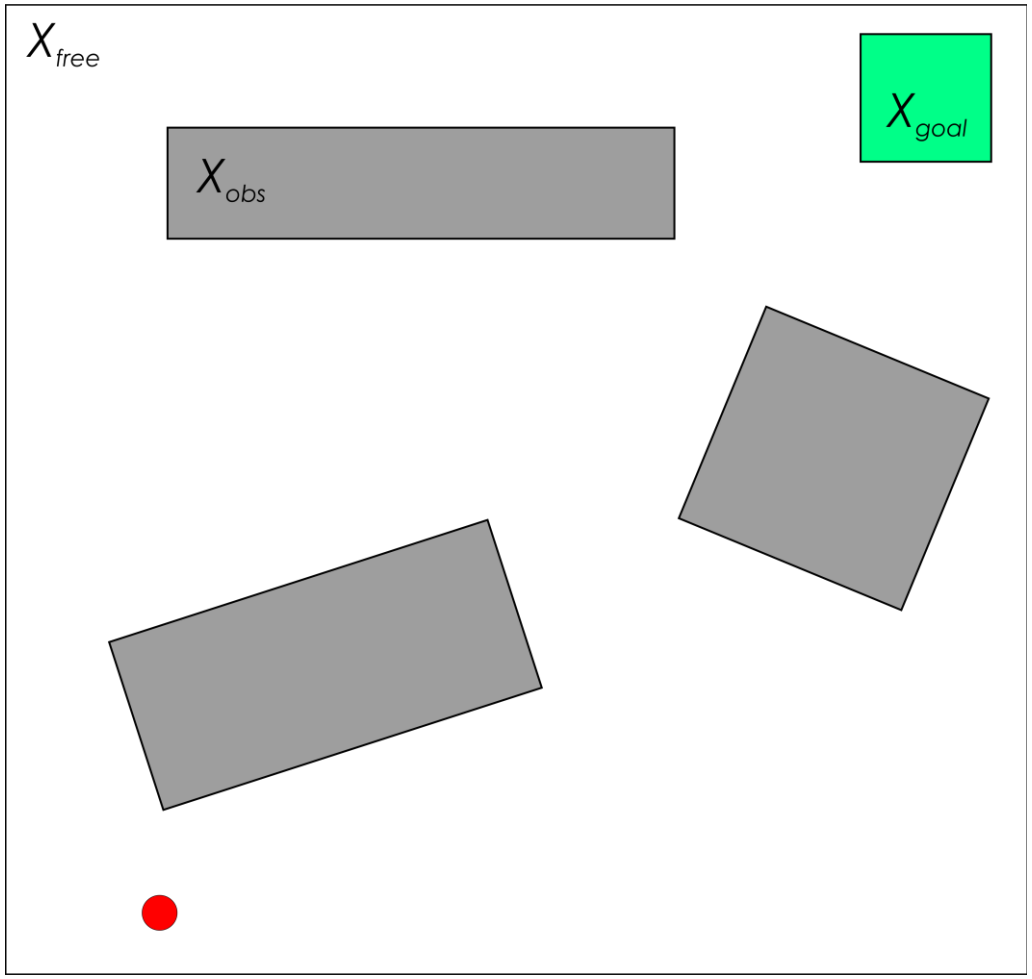
$V \leftarrow \{x_{init}\}; E \leftarrow 0;$
for i=1,…,N **do**:
    $x_{rand} \leftarrow SampleFree;$
    $x_{nearest} \leftarrow Nearest(G = (V, E), x_{rand});$
    $x_{new} \leftarrow Steer(x_{nearest}, x_{rand});$
    if $ObstacleFree(x_{nearest}, x_{new})$ **then**:
        $V \leftarrow V \cup \{x_{new}\}; E \leftarrow E \cup \{(x_{nearest}, x_{new})\};$
return $G = (V, E);$

- The RRT algorithm is **probabilistically complete**.
- The probability of success goes to 1 exponentially fast, if the environment satisfies certain "*good visibility*" conditions.

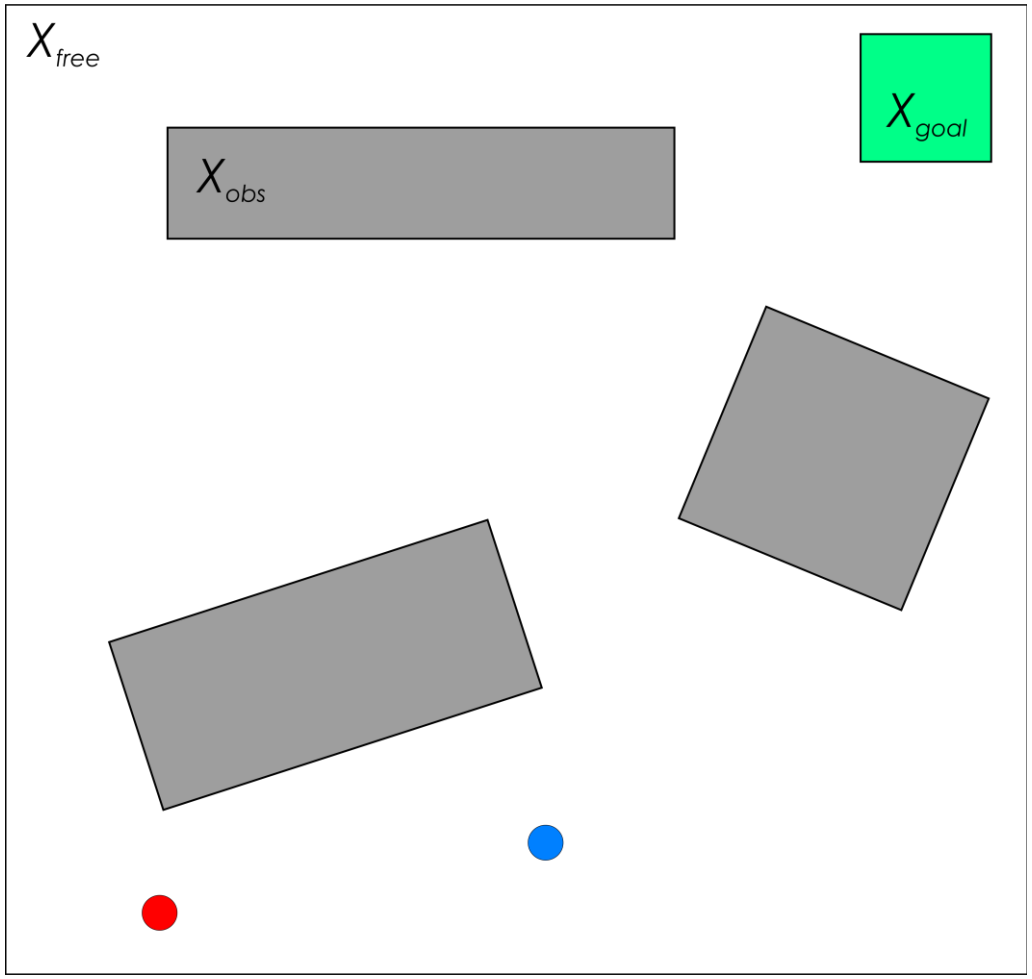
Rapidly-exploring Random Trees (RRTs)



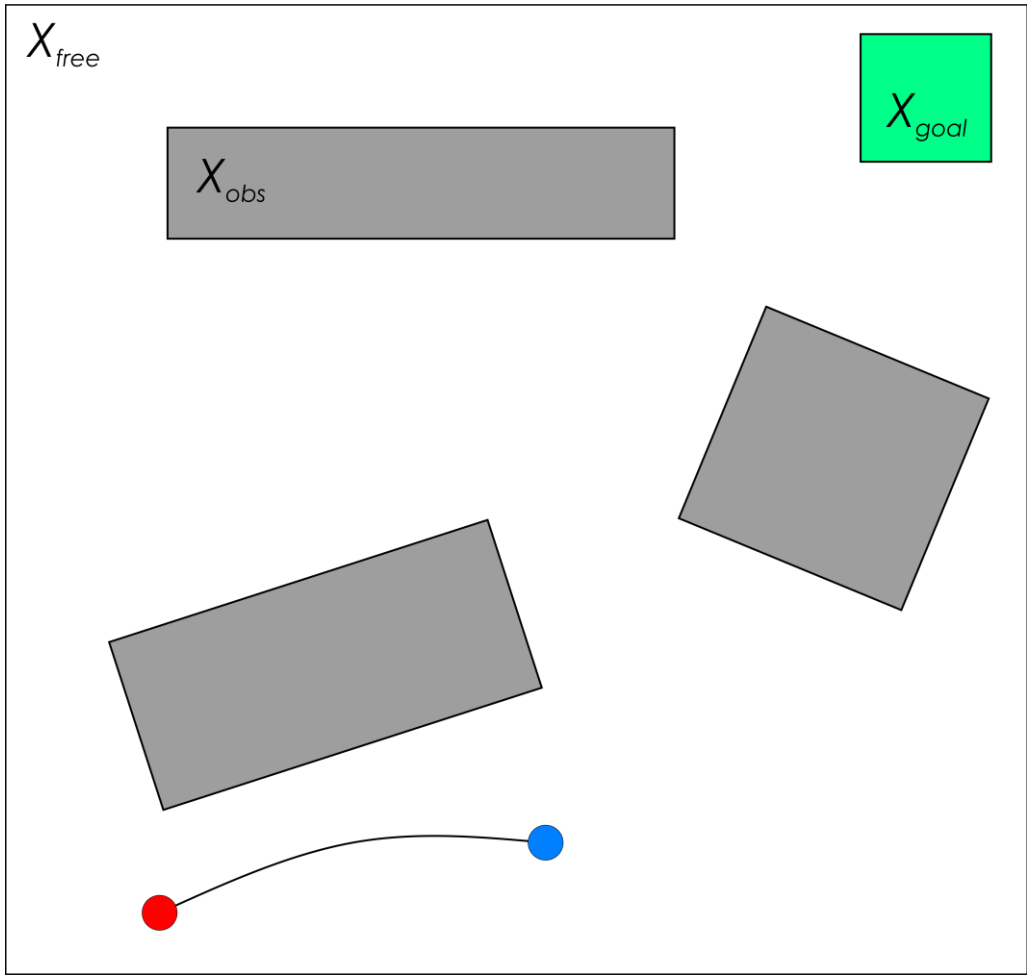
Rapidly-exploring Random Trees (RRTs)



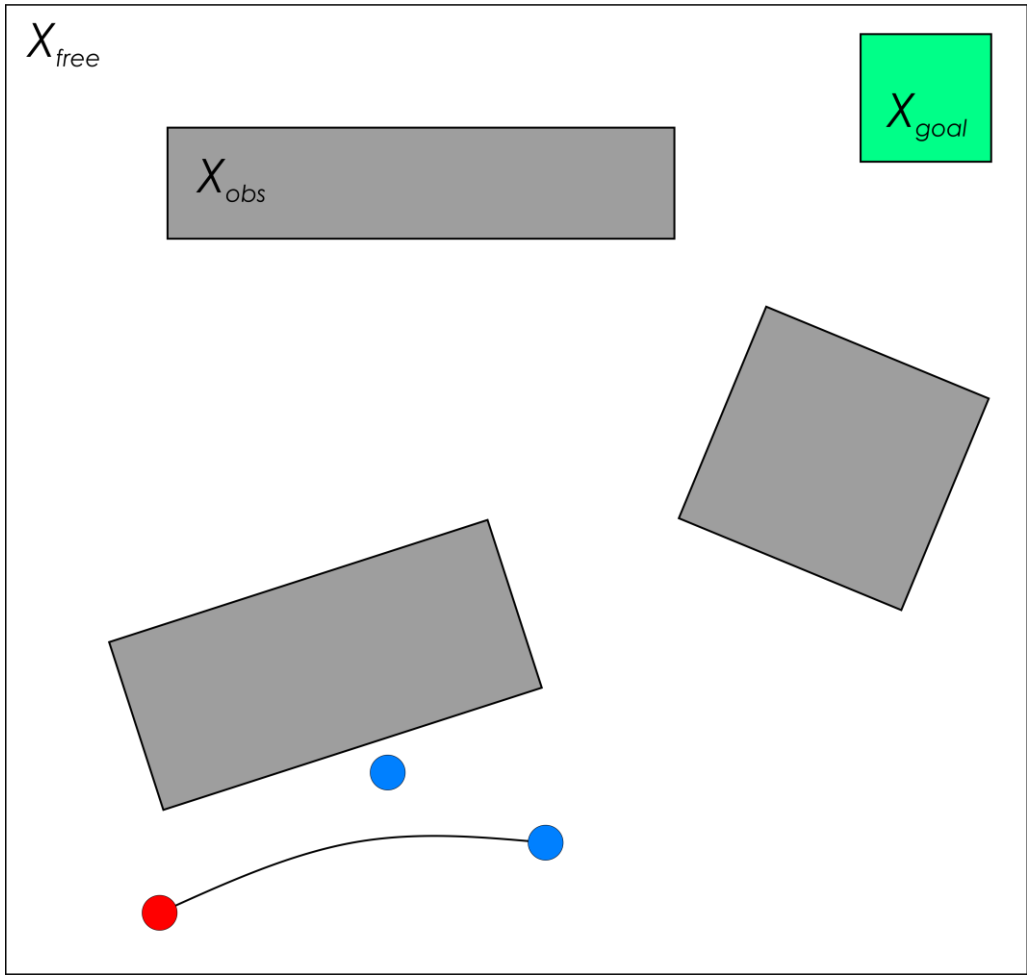
Rapidly-exploring Random Trees (RRTs)



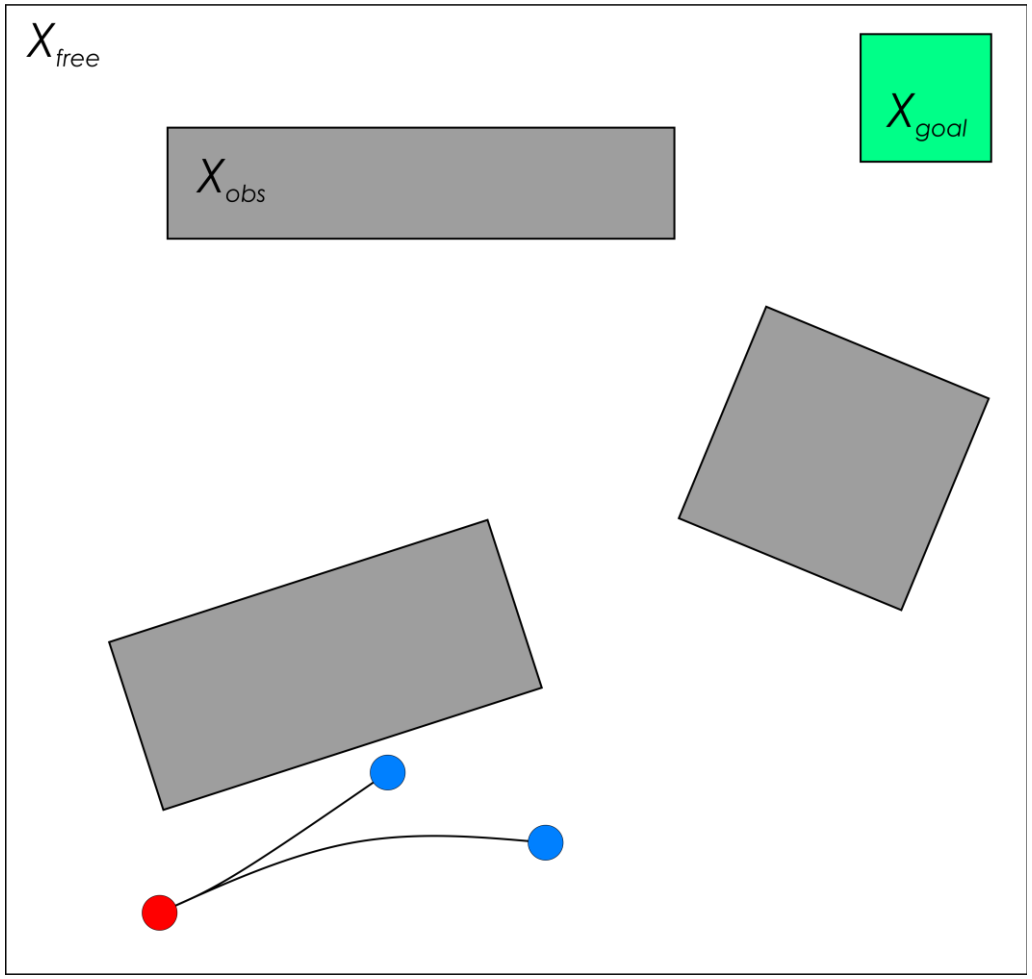
Rapidly-exploring Random Trees (RRTs)



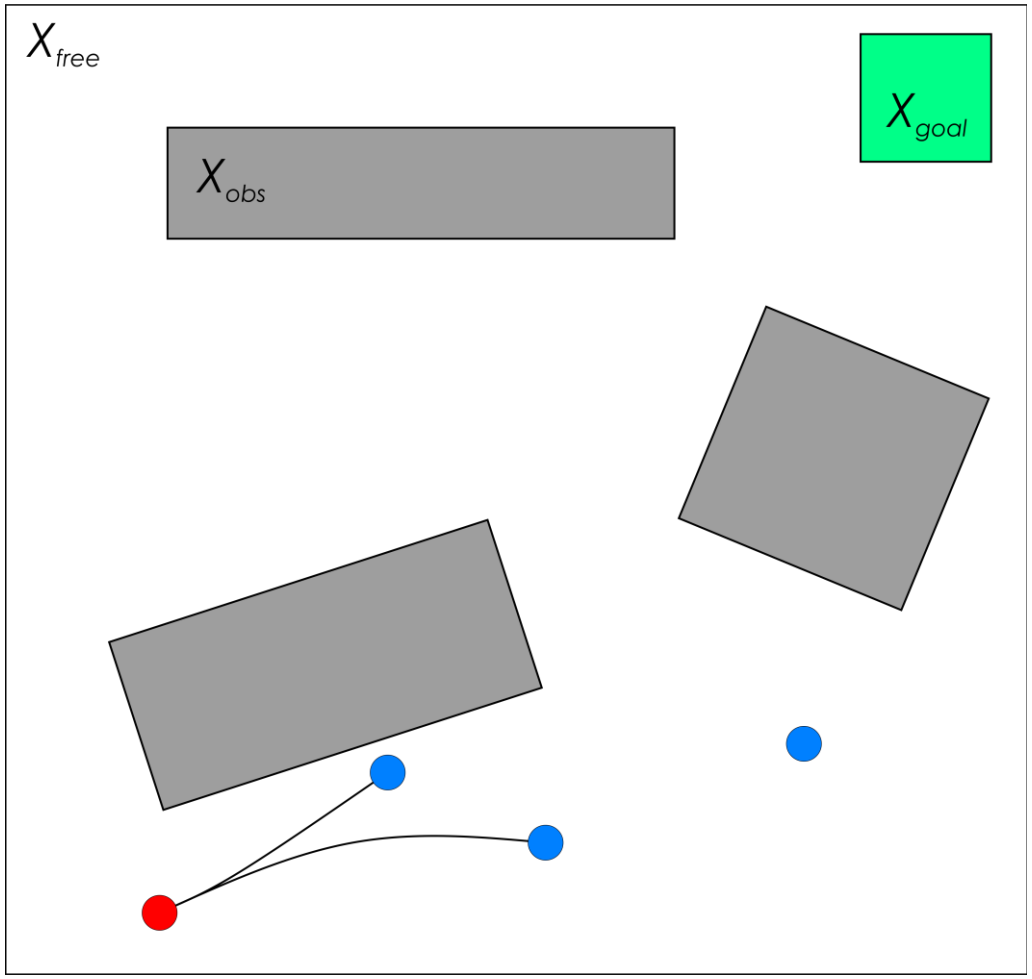
Rapidly-exploring Random Trees (RRTs)



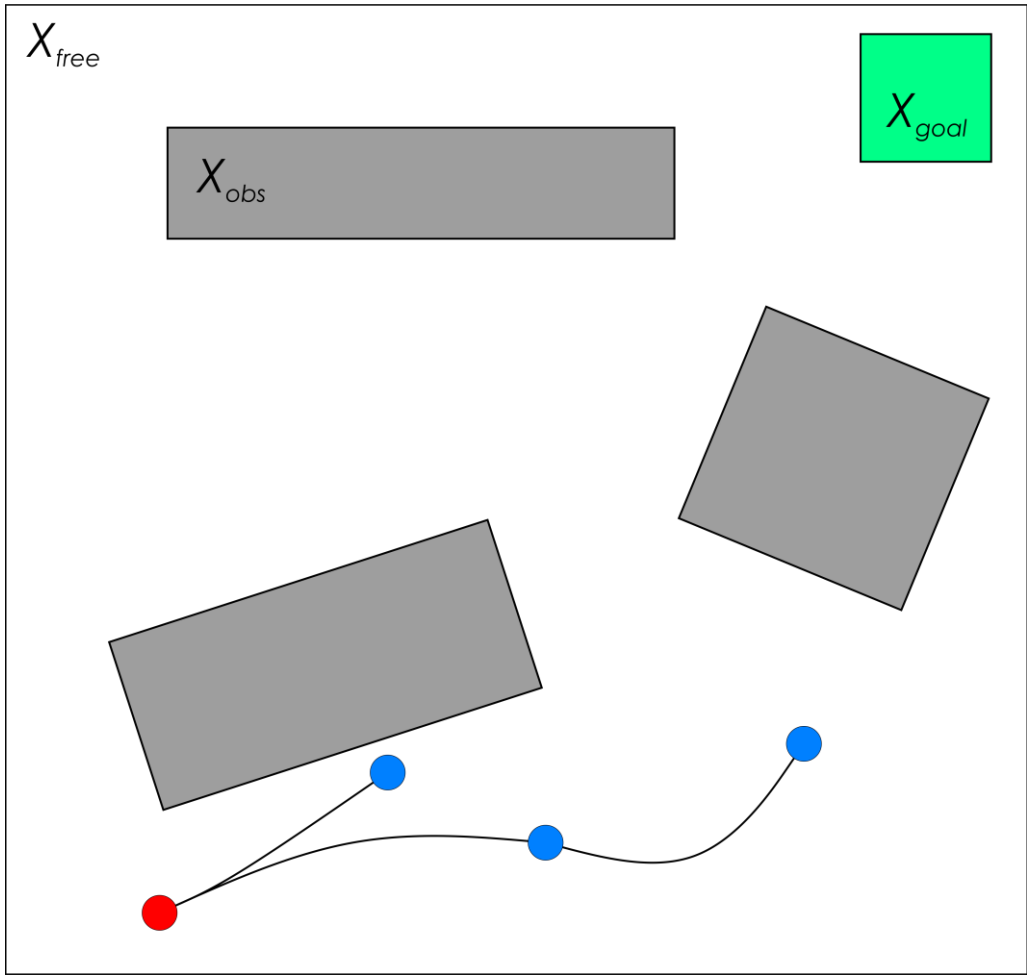
Rapidly-exploring Random Trees (RRTs)



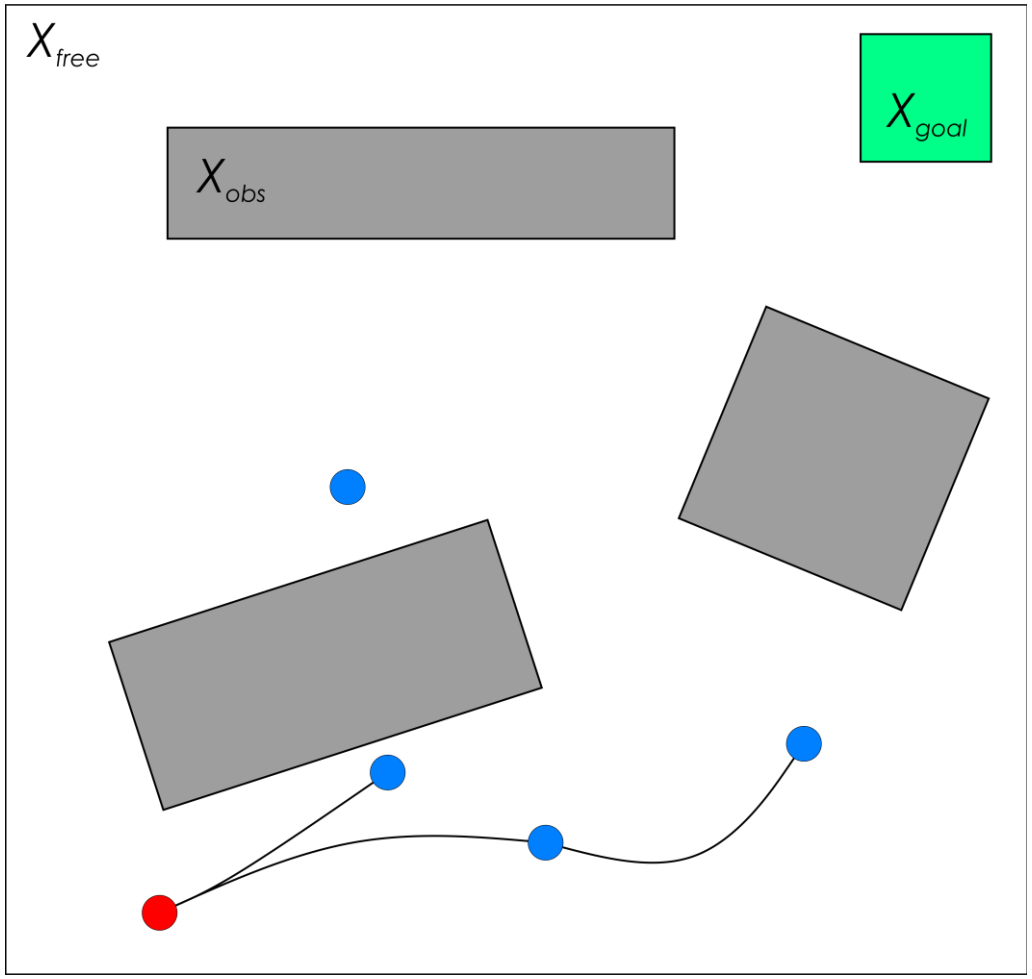
Rapidly-exploring Random Trees (RRTs)



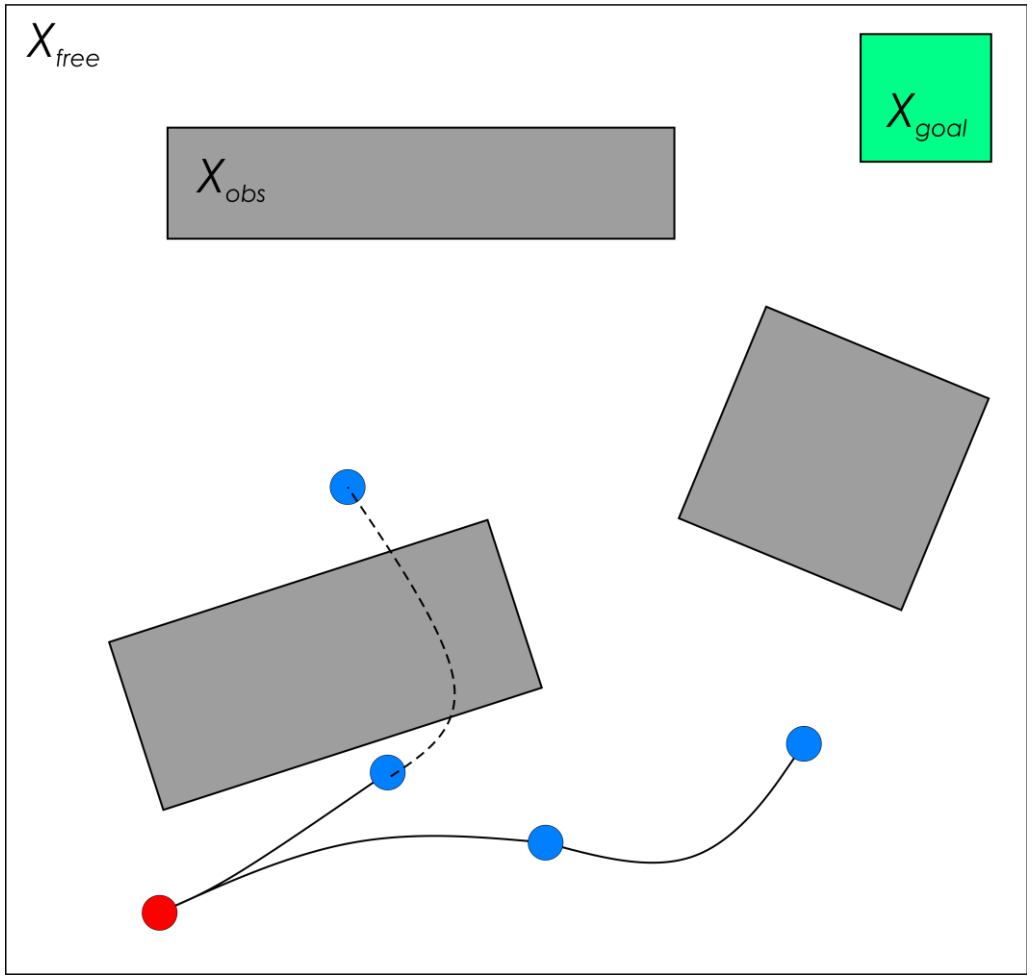
Rapidly-exploring Random Trees (RRTs)



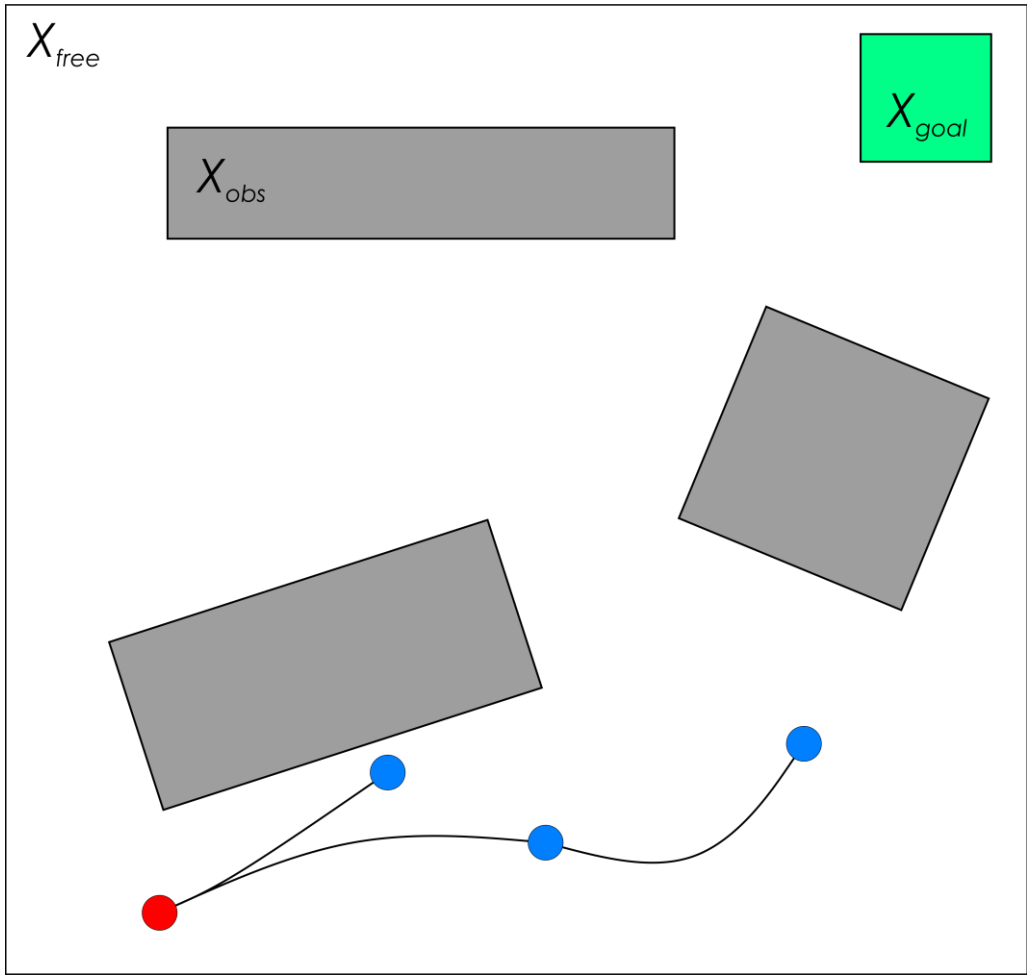
Rapidly-exploring Random Trees (RRTs)



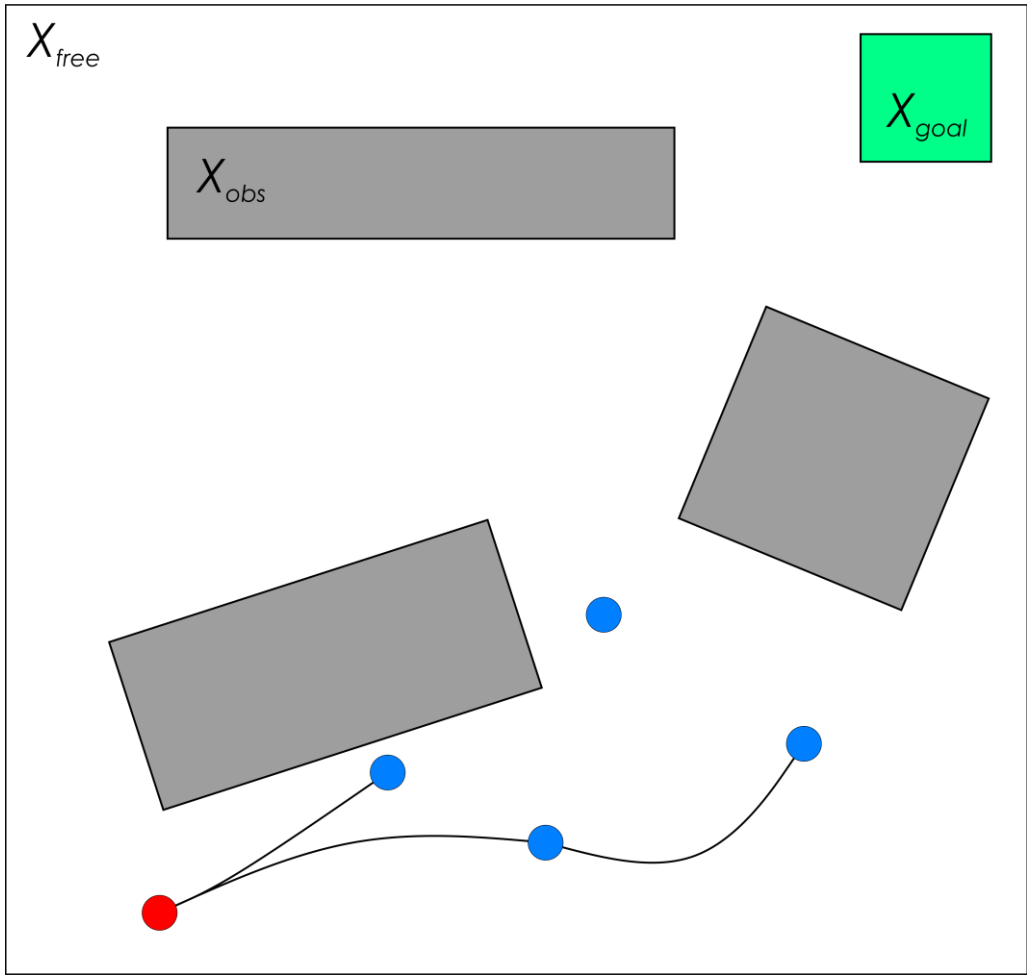
Rapidly-exploring Random Trees (RRTs)



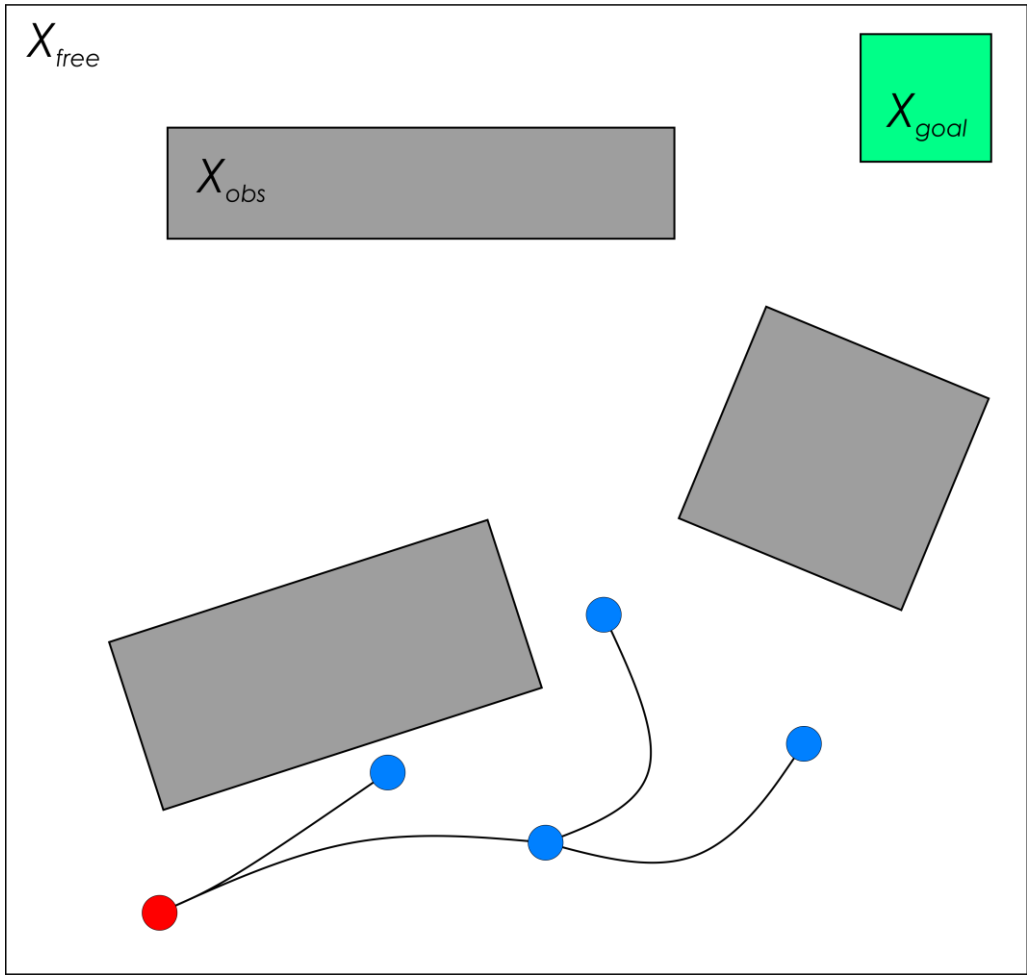
Rapidly-exploring Random Trees (RRTs)



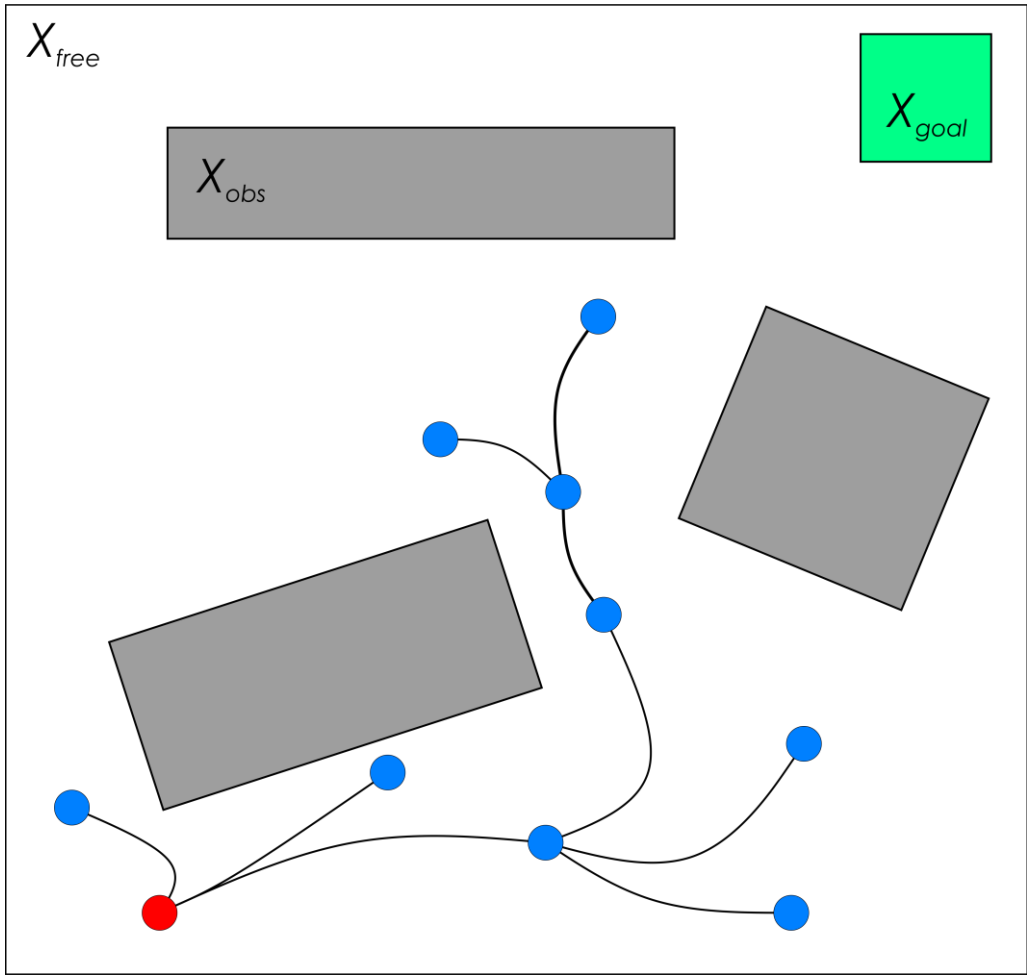
Rapidly-exploring Random Trees (RRTs)



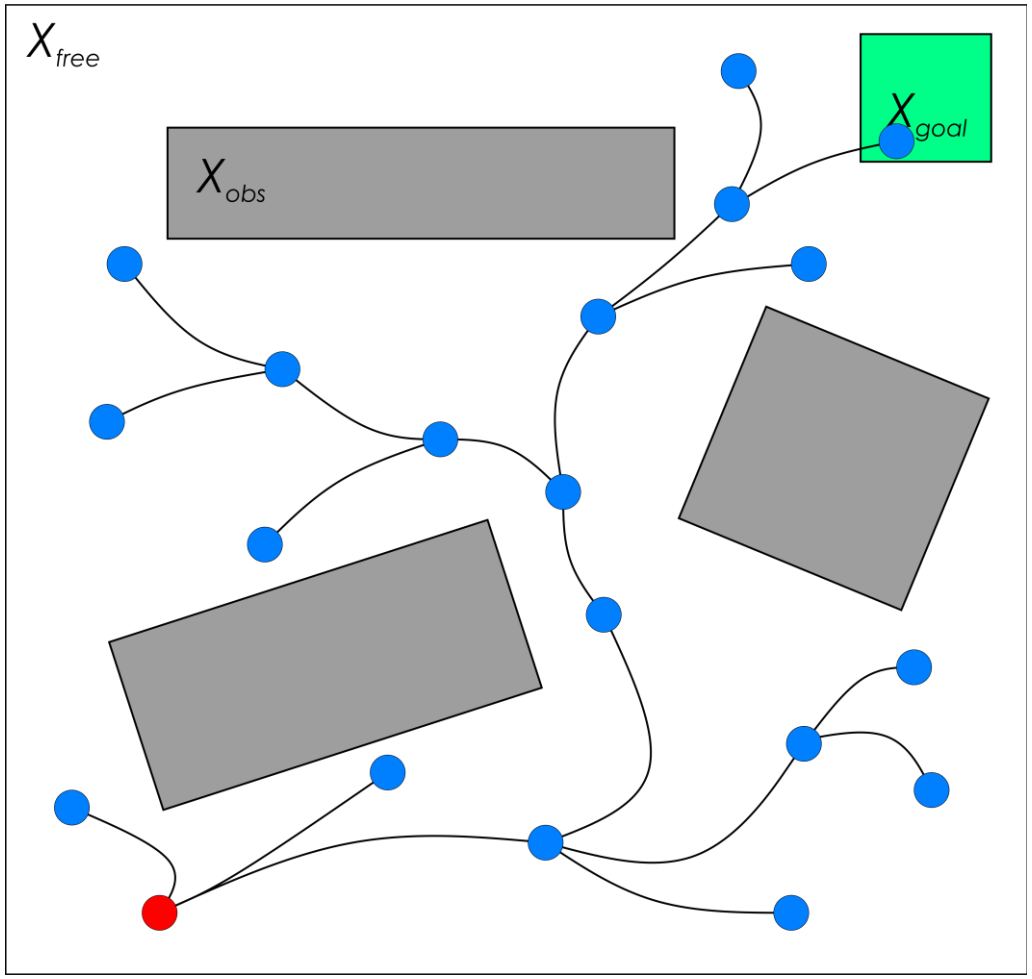
Rapidly-exploring Random Trees (RRTs)



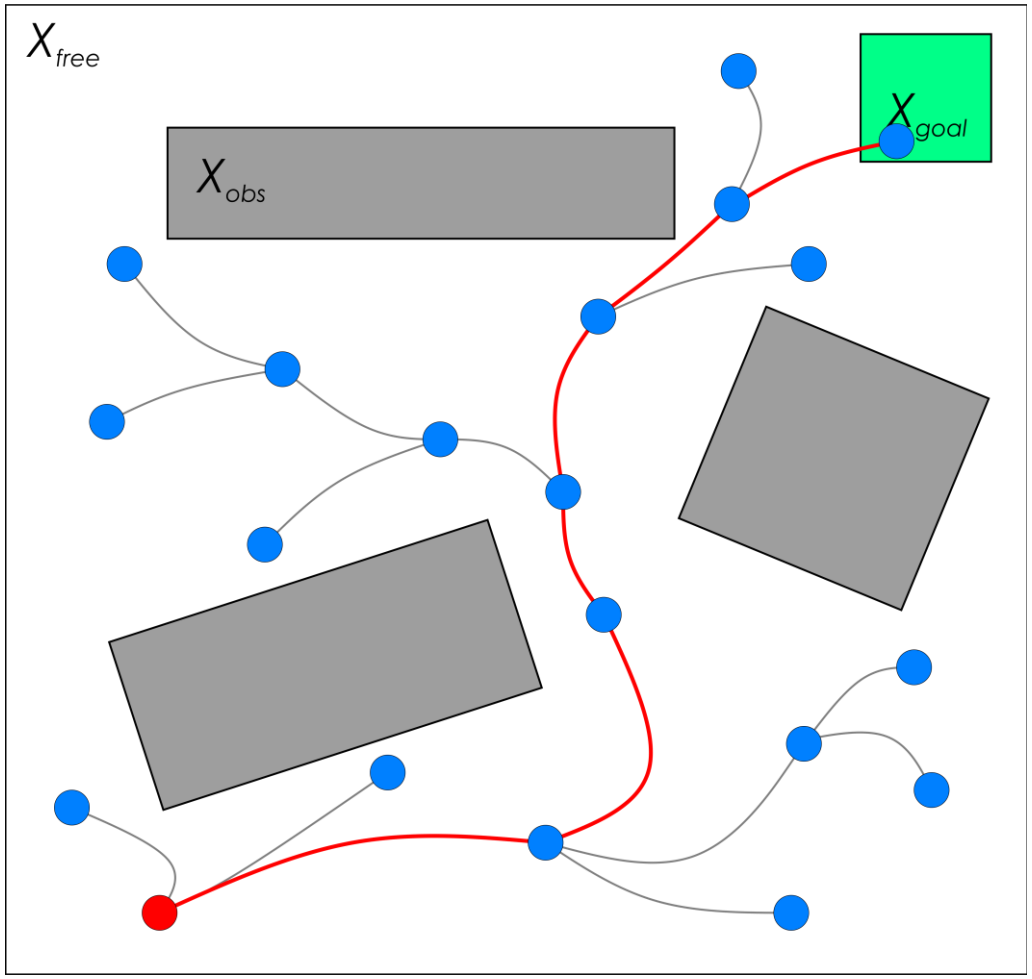
Rapidly-exploring Random Trees (RRTs)



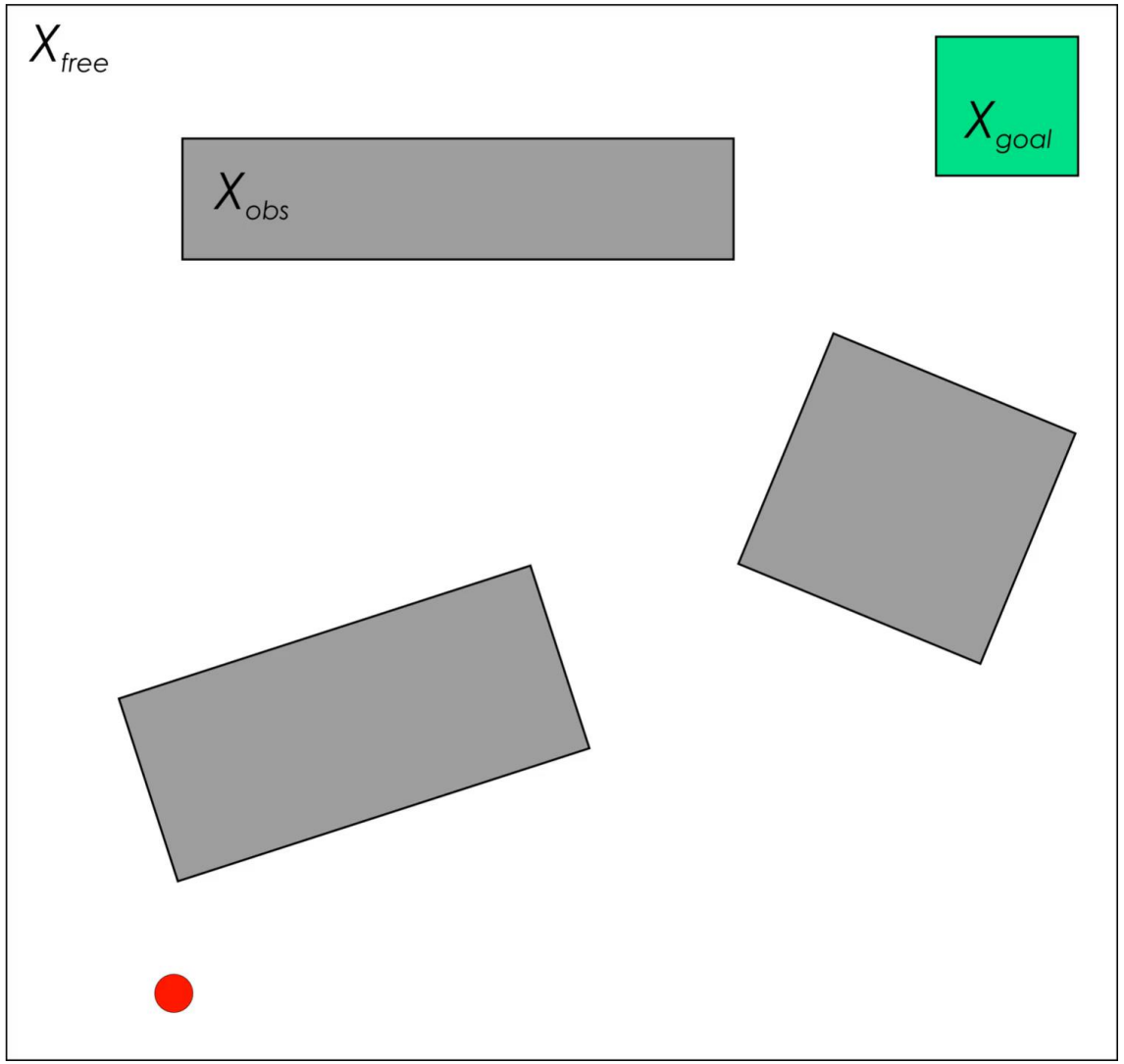
Rapidly-exploring Random Trees (RRTs)



Rapidly-exploring Random Trees (RRTs)



Rapidly-exploring Random Trees (RRTs)



Voronoi bias

Definition - Voronoi diagram:

Given n sites in d dimensions, the Voronoi diagram of the sites is a partition of $\mathbb{R}^d$ into regions, one region per site, such that all points in the interior of each region lie closer to that regions site than to any other site.

- Vertices of the RRT that are more "isolated" (e.g. in unexplored areas, or at the boundary of the explored area) have the larger Voronoi regions – and are more likely to be selected for extension.

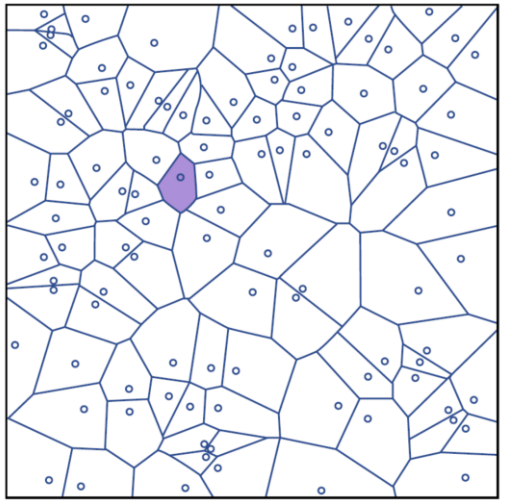


Image by MIT OpenCourseWare.

RRTs in action

- Great results for collision-avoidance of aerial robots.
- Integral component of several, higher-level algorithms (e.g. exploration and inspection).

Limitations of such incremental sampling methods

- No characterization of the quality (e.g. "cost") of the trajectories returned by the algorithm.

  Keep running the RRT even after the first solution has been obtained, for as long as possible (given the real-time constraints), hoping to find a better path than the one already available.

- *No systematic method for imposing temporal/logical constraints*, such as, e.g. the rules of the road, complicated mission objectives, ethical/ deontic code.

RRTs don't have the best behavior

- Let Y_n^{RRT} be the cost of the best path in the RRT at the end of iteration n
- It is easy to show that Y_n^{RRT} converges (to a random variable), i.e.:

$$\lim_{n\to\infty} Y_n^{RRT} = Y_\infty^{RRT}$$

- The random variable Y_∞^{RRT} is sampled from a distribution with zero mass at the optimum:

Theorem – Almost sure suboptimality of RRTs

If a set of sampled optimal paths has measure zero, the sampling distribution is absolutely continuous with positive density in X_{free} and $d \geq 2$, then the best path in the RRT converges to a sub-optimal solution almost surely, i.e.:

$$\Pr[Y_\infty^{RRT} > c^*] = 1$$

Some remarks on that negative result

- **Intuition:** RRT does not satisfy a necessary condition for asymptotic optimality, i.e., that the root node has infinitely many subtrees that extend at least a distance ϵ away from x_{init}.
- The RRT algorithm "traps" itself by disallowing new better paths to emerge.
- **Heuristics such as**
  - Running the RRT multiple times
  - Running multiple times concurrently
  - Deleting and rebuilding parts of the tree etc.

  Work better than the standard RRT, but cannot remove the sub-optimal behavior.

- **How can we do better?**

Rapidly-exploring Random Graphs (RRGs)

RRG Algorithm

$V \leftarrow \{x_{init}\}; E \leftarrow 0;$
for i=1,…,N **do**:
    $x_{rand} \leftarrow SampleFree;$
    $x_{nearest} \leftarrow Nearest(G = (V, E), x_{rand});$
    $x_{new} \leftarrow Steer(x_{nearest}, x_{rand});$
    if $ObstacleFree(x_{nearest}, x_{new})$ **then**:
        $X_{near} \leftarrow Near\left(G = (V, E), x_{new}, \min\{\gamma_{RRG}\left(\frac{\log(card\ V)}{card\ V}\right)^{1/d}, \eta\}\right);$
        $V \leftarrow V \cup \{x_{new}\}; E \leftarrow E \cup \{(x_{nearest}, x_{new}), (x_{nearest}, x_{new})\};$
        foreach $x_{near} \in X_{near}$ **do:**
            if $CollisionFree(x_{near}, x_{new})$ then $E \leftarrow E \cup \{(x_{near}, x_{new}), (x_{near}, x_{new})\};$
return $G = (V, E);$

- At each iteration, the RRG tries to connect to the new sample all vertices in a ball radius r_n centered at it. (Or simply default to the nearest one if such a ball is empty).
- In general, the *RRG builds graphs with cycles*.

Properties of RRGs

Theorem – Probabilistic completeness

Since $V_n^{RRG} = V_n^{RRT}$, for all n, it follows that RRG has the same completeness properties of RRT, i.e.

$$\Pr\left[V_n^{RRG} \cap X_{goal} = 0\right] = O(e^{-bn})$$

Theorem – Asymptotic optimality

If the $Near$ procedure returns all nodes in V within a ball of volume

$$Vol = \gamma \frac{\log n}{n}, \gamma > 2^d(1 + {}^1/_d),$$

Under some additional technical assumptions (e.g., on the sampling distribution, on the ϵ clearance of the optimal path, and on the continuity of the cost function), the best path in the RRG converges to an optimal solution almost surely, i.e.:

$$\Pr[Y_\infty^{RRG} = c^*] = 1$$

Computational complexity

- At each iteration, the RRG algorithm executes $O(\log n)$ extra calls to $ObstacleFree$ when compared to the RRT.
- However, the complexity of the $Nearest$ procedure is $\Omega(\log n)$. Achieved if using, e.g., a Balanced-Box Decomposition (BBD) Tree.

Theorem – Asymptotic (Relative) Complexity

There exists a constant $\beta \in \mathbb{R}_+$ such that

$$\lim_{i \to \infty} sup\, E\left[\frac{OPS_i^{RRG}}{OPS_i^{RRT}}\right] \leq \beta$$

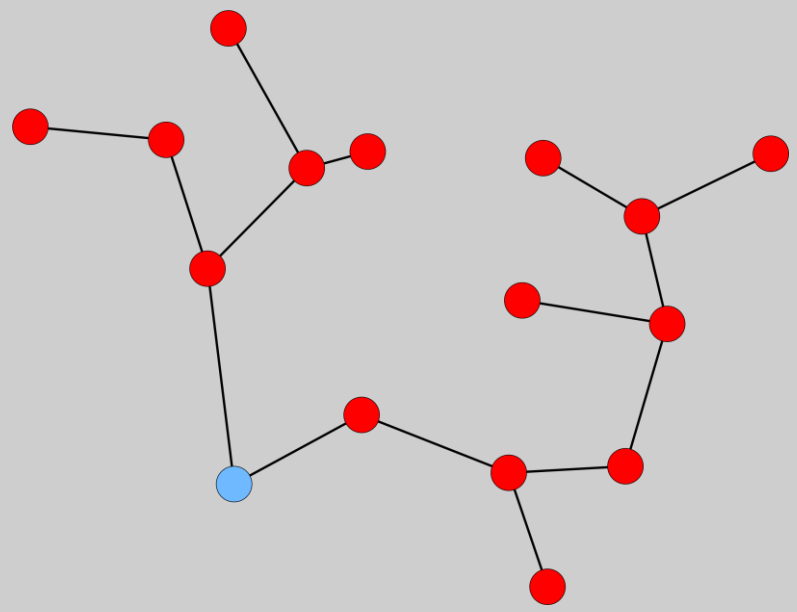
- In other words, the RRG algorithm has **not much more computational overhead** over RRT, and ensures asymptotic optimality.

RRT*: A tree version of the RRG

- RRT algorithm can account for nonholonomic dynamics and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the $Steer$ procedure has to be exact. *Exact steering methods are not available for general dynamic systems.*

RRT* Algorithm

- RRT* is a variant of RRG that essentially "rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRG and RRT.

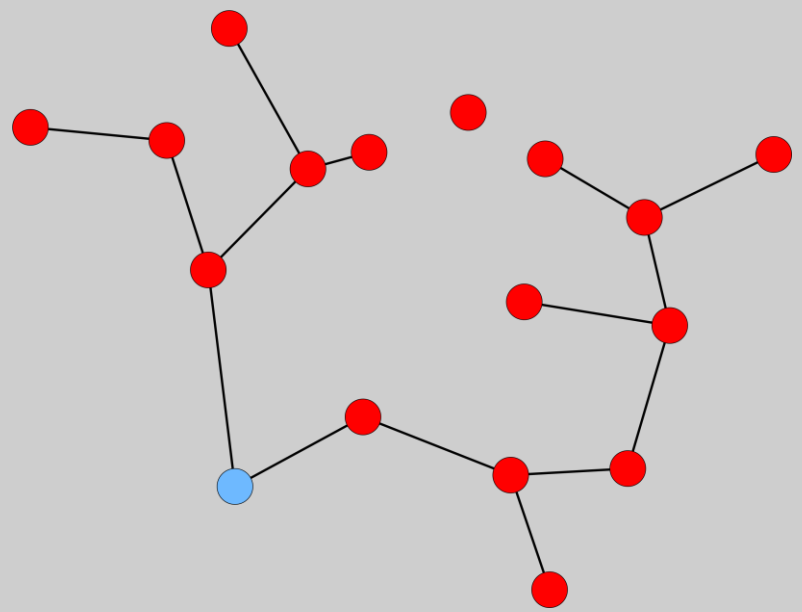


RRT*: A tree version of the RRG

- RRT algorithm can account for nonholonomic dynamics and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the $Steer$ procedure has to be exact. Exact steering methods are not available for general dynamic systems.

RRT* Algorithm

- RRT* is a variant of RRG that essentially "rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRG and RRT.

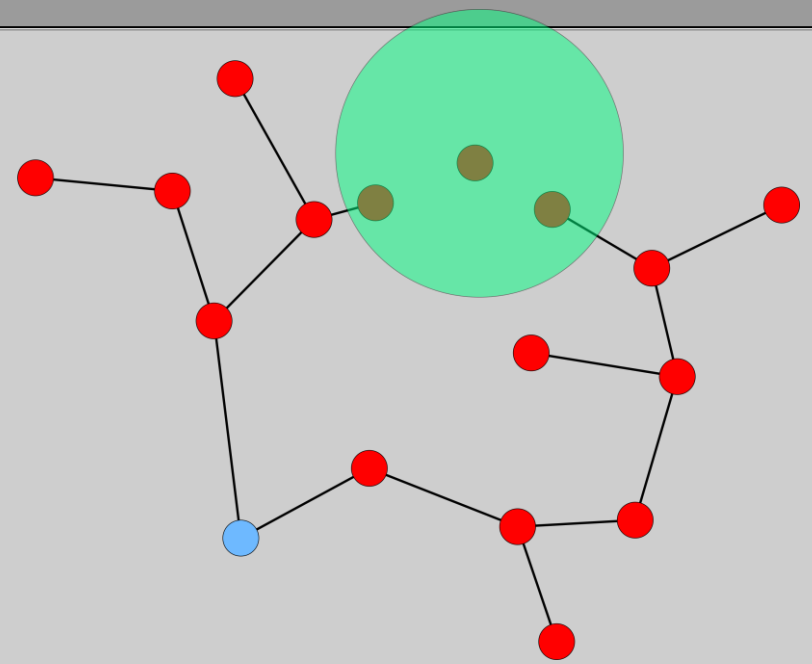


RRT*: A tree version of the RRG

- RRT algorithm can account for nonholonomic dynamics and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the $Steer$ procedure has to be exact. Exact steering methods are not available for general dynamic systems.

RRT* Algorithm

- RRT* is a variant of RRG that essentially "rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRG and RRT.

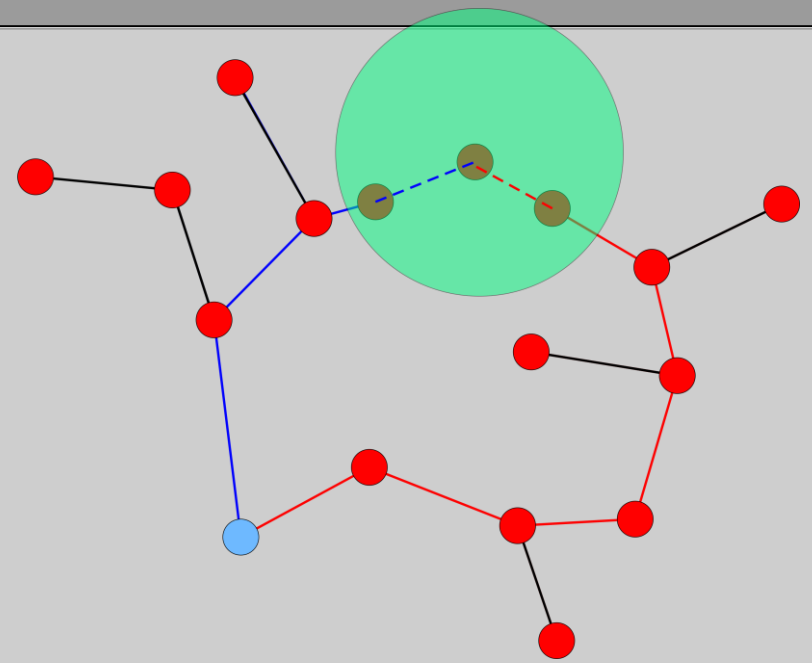


RRT*: A tree version of the RRG

- RRT algorithm can account for nonholonomic dynamics and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the $Steer$ procedure has to be exact. Exact steering methods are not available for general dynamic systems.

RRT* Algorithm

- RRT* is a variant of RRG that essentially "rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRG and RRT.

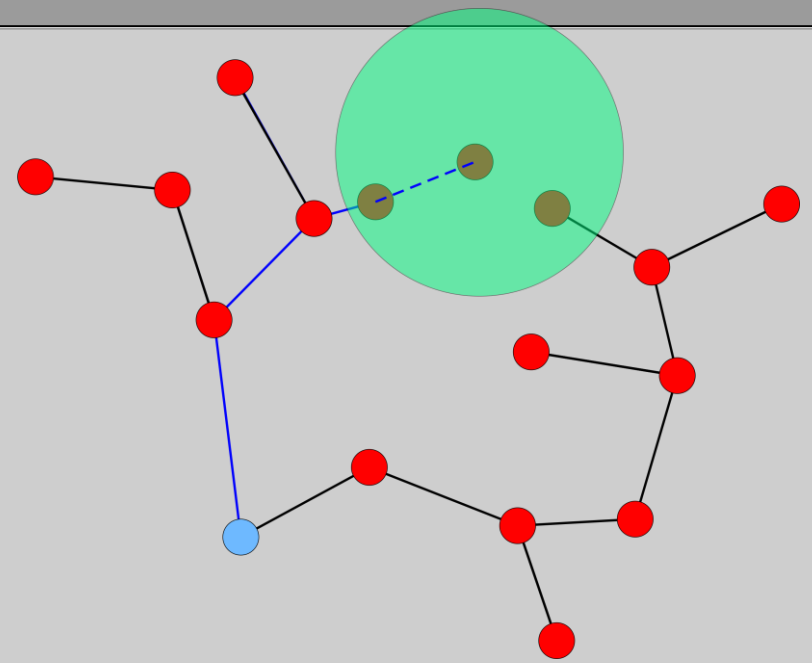


RRT*: A tree version of the RRG

- RRT algorithm can account for nonholonomic dynamics and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the $Steer$ procedure has to be exact. Exact steering methods are not available for general dynamic systems.

RRT* Algorithm

- RRT* is a variant of RRG that essentially "rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRG and RRT.

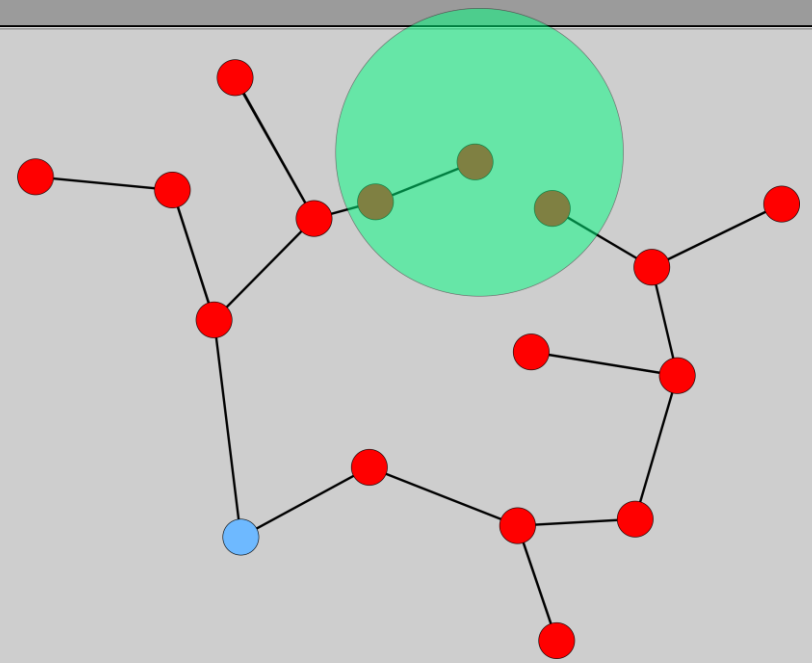


RRT*: A tree version of the RRG

- RRT algorithm can account for nonholonomic dynamics and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the $Steer$ procedure has to be exact. Exact steering methods are not available for general dynamic systems.

RRT* Algorithm

- RRT* is a variant of RRG that essentially "rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRG and RRT.

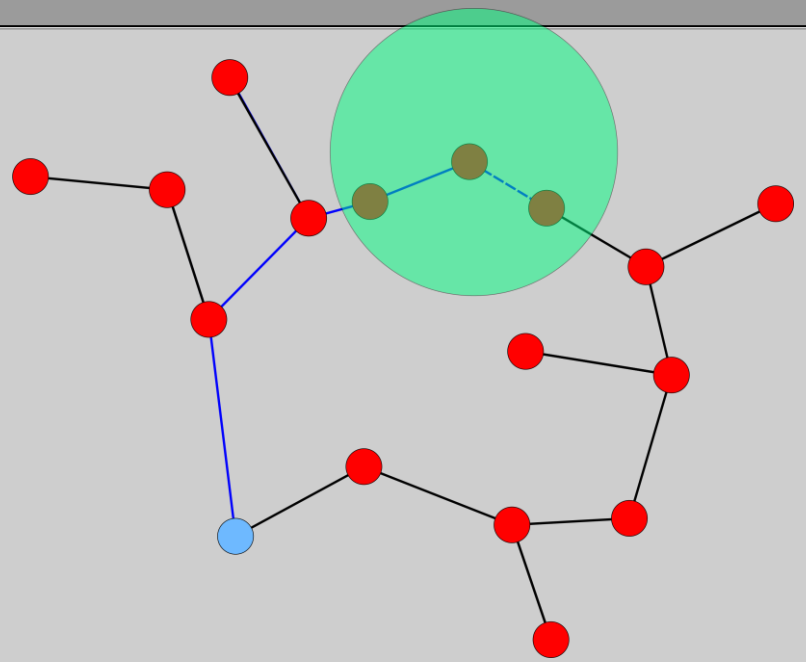


RRT*: A tree version of the RRG

- RRT algorithm can account for nonholonomic dynamics and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the $Steer$ procedure has to be exact. Exact steering methods are not available for general dynamic systems.

RRT* Algorithm

- RRT* is a variant of RRG that essentially "rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRG and RRT.

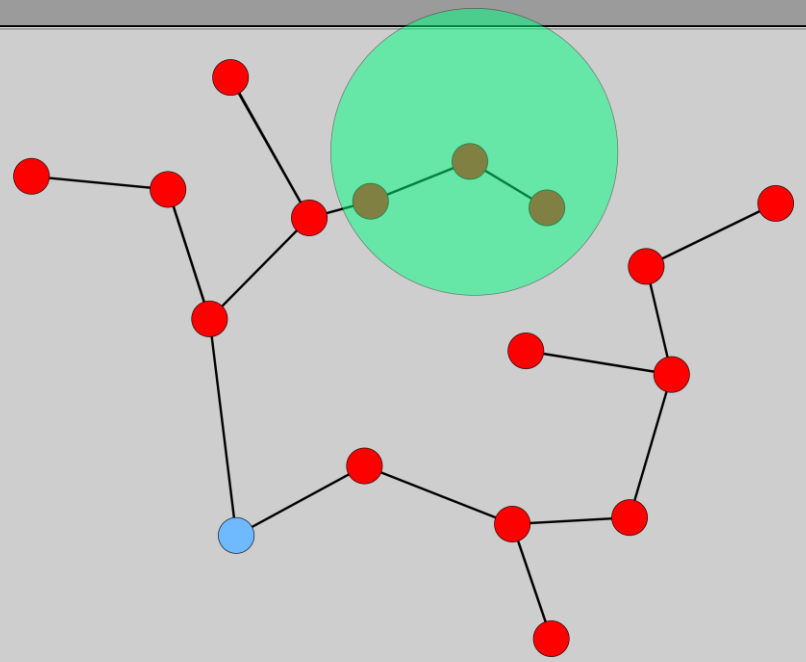


RRT*: A tree version of the RRG

- RRT algorithm can account for nonholonomic dynamics and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the $Steer$ procedure has to be exact. Exact steering methods are not available for general dynamic systems.

RRT* Algorithm

- RRT* is a variant of RRG that essentially "rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRG and RRT.

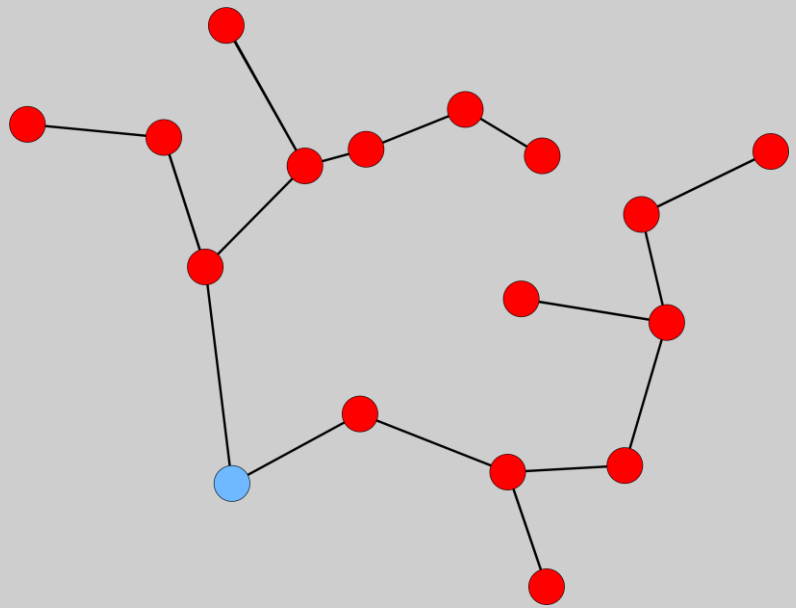


RRT*: A tree version of the RRG

- RRT algorithm can account for nonholonomic dynamics and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the $Steer$ procedure has to be exact. Exact steering methods are not available for general dynamic systems.

RRT* Algorithm

- RRT* is a variant of RRG that essentially "rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRG and RRT.

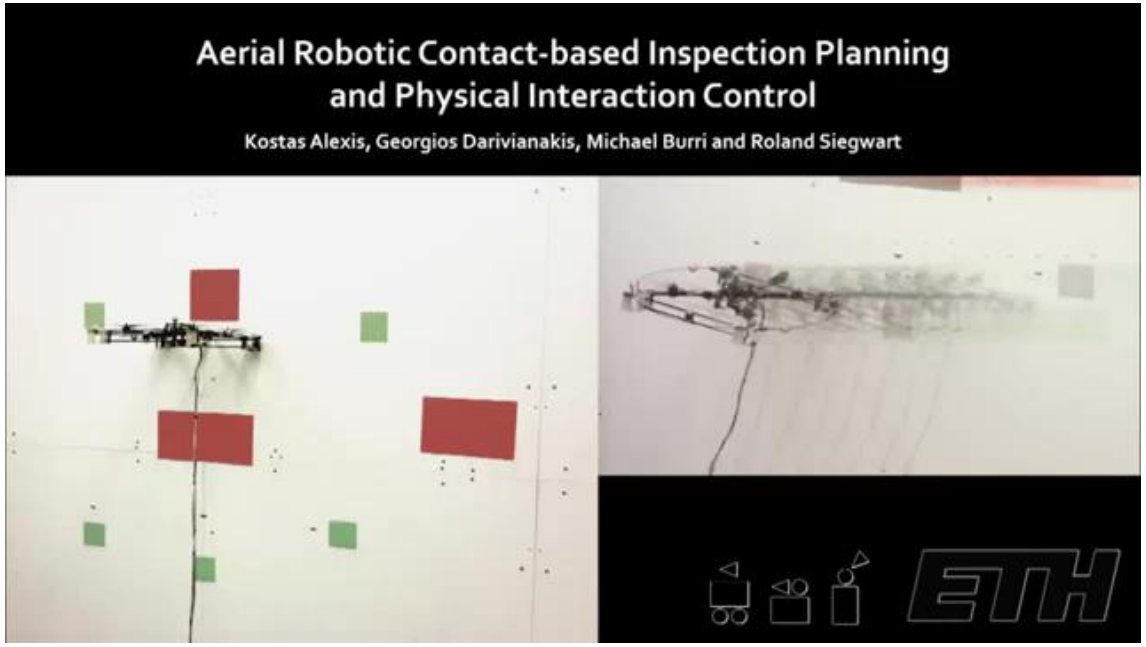


Rapidly-exploring Random Tree-star (RRT*)

RRT* Algorithm

$V \leftarrow \{x_{init}\}; E \leftarrow 0;$
for i=1,…,N **do**:
    $x_{rand} \leftarrow SampleFree;$
    $x_{nearest} \leftarrow Nearest(G = (V, E), x_{rand});$
    $x_{new} \leftarrow Steer(x_{nearest}, x_{rand});$
    if $ObstacleFree(x_{nearest}, x_{new})$ **then**:
        $X_{near} \leftarrow Near\left(G = (V, E), x_{new}, \min\{\gamma_{RRG}\left(\frac{\log(card\ V)}{card\ V}\right)^{1/d}, \eta\}\right);$
        $V \leftarrow V \cup \{x_{new}\};$
        $x_{min} \leftarrow x_{nearest};\ c_{\min} \leftarrow Cost(x_{nearest}) + c(Line(x_{nearest}, x_{new}))$
        foreach $x_{near} \in X_{near}$ **do:**
            if $CollisionFree(x_{near}, x_{new}) \wedge Cost(x_{near}) + c\big(Line(x_{near}, x_{new})\big) < Cost(x_{near})$ **then:**
                $x_{parent} \leftarrow Parent(x_{near});$
                $E \leftarrow E \backslash \{(x_{parent}, x_{near})\} \cup \{(x_{new}, x_{near})\};$
return $G = (V, E);$

RRT* in Action



BadgerWorks Lectures

Primer Series - Exploration Path Planning

Kostas Alexis

AUTONOMOUS ROBOTS LAB

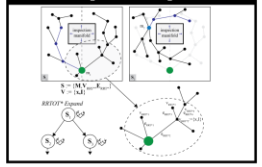
ARL 2016 Planning Ensemble for Mapping

- **Classification:**
  - Prior environmental knowledge?
  - Active perception and belief-space planning?
  - Possible human co-working?
  - Applicable to "any" robot configuration?
- **Robot evaluation:**
  - Multi-rotor UAV systems
  - Fixed-wing UAV systems
  - …not limited

Coverage Path Planning

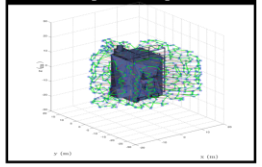
Exploration Path Planning

Optimal Coverage (RRTOT)

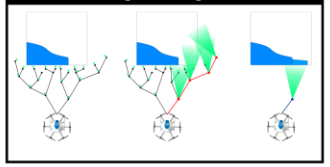


Efficient Coverage (SIP)

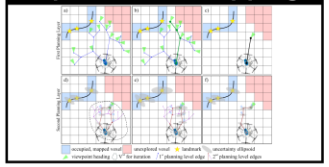
Uniform Coverage (UC3D)



Autonomous Exploration (NBVP)



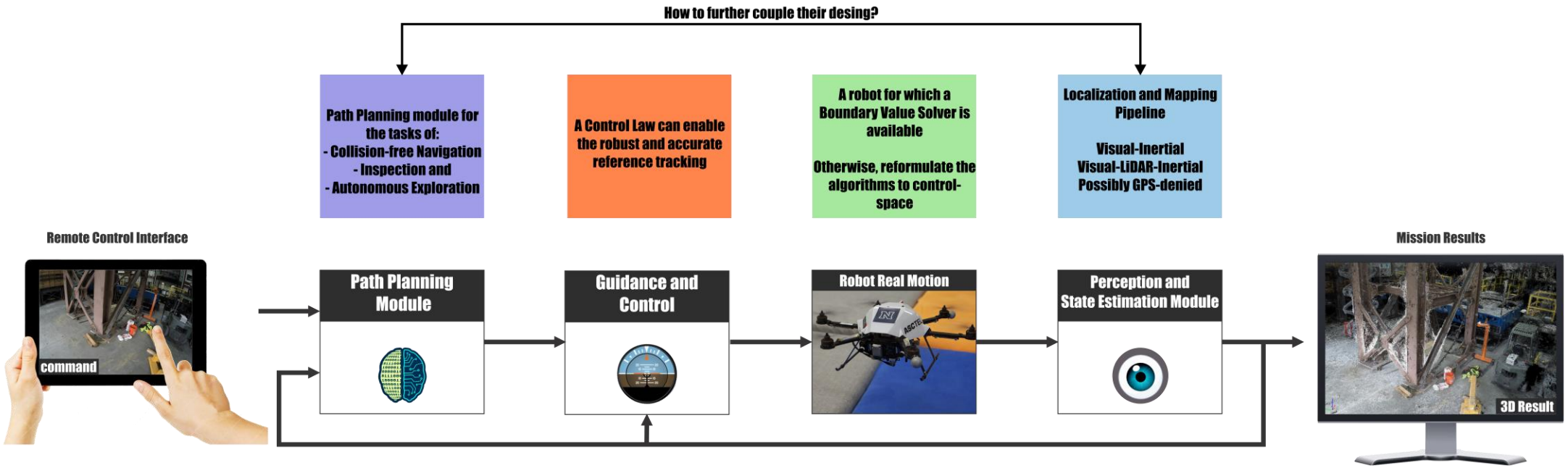
Uncertainty-aware Exploration & Mapping



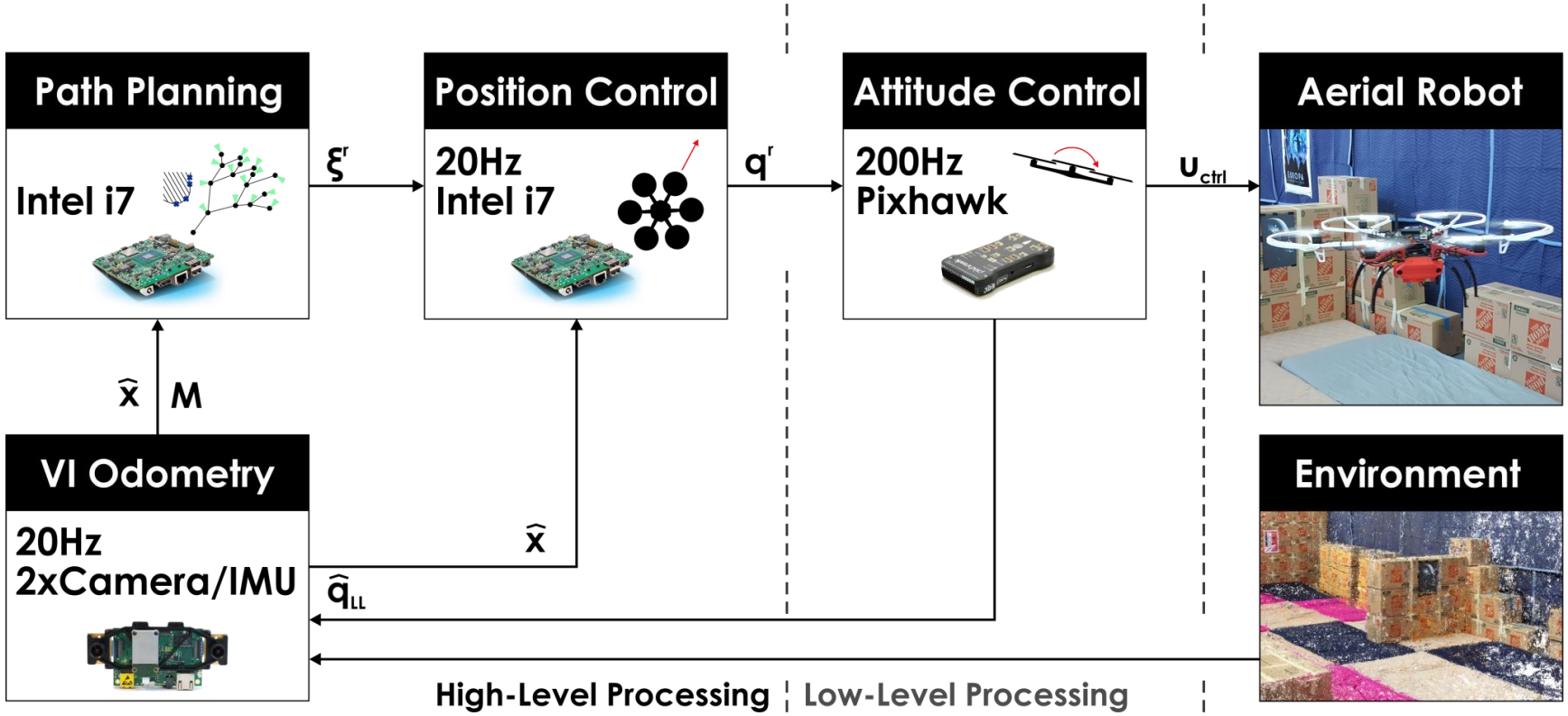
Augmented Reality Inspection

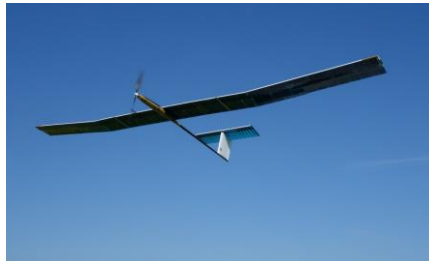
Augmented Reality-enhanced Inspection

Assumed Robot Configuration



Example Specific Robot Configuration





Proposed Planning Ensemble

- **Classification:**
  - **NO Prior environmental knowledge**
  - **Active perception and belief-space planning?**
  - **Possible human co-working?**
  - **Applicable to "any" robot configuration?**

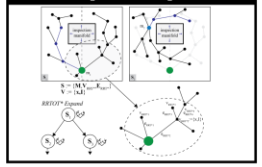
Focus of this presentation

- **Robot evaluation:**
  - **Multi-rotor UAV systems**
  - **Fixed-wing UAV systems**

Coverage Path Planning

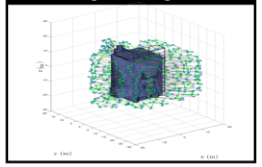
Exploration Path Planning

Optimal Coverage (RRTOT)

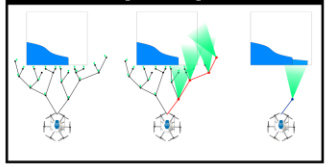


Efficient Coverage (SIP)

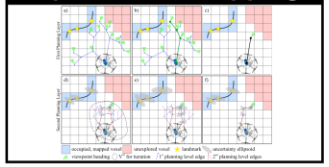
Uniform Coverage (UC3D)



Autonomous Exploration (NBVP)



Uncertainty-aware Exploration & Mapping



Augmented Reality Inspection

Augmented Reality-enhanced Inspection

The Exploration path planning problem

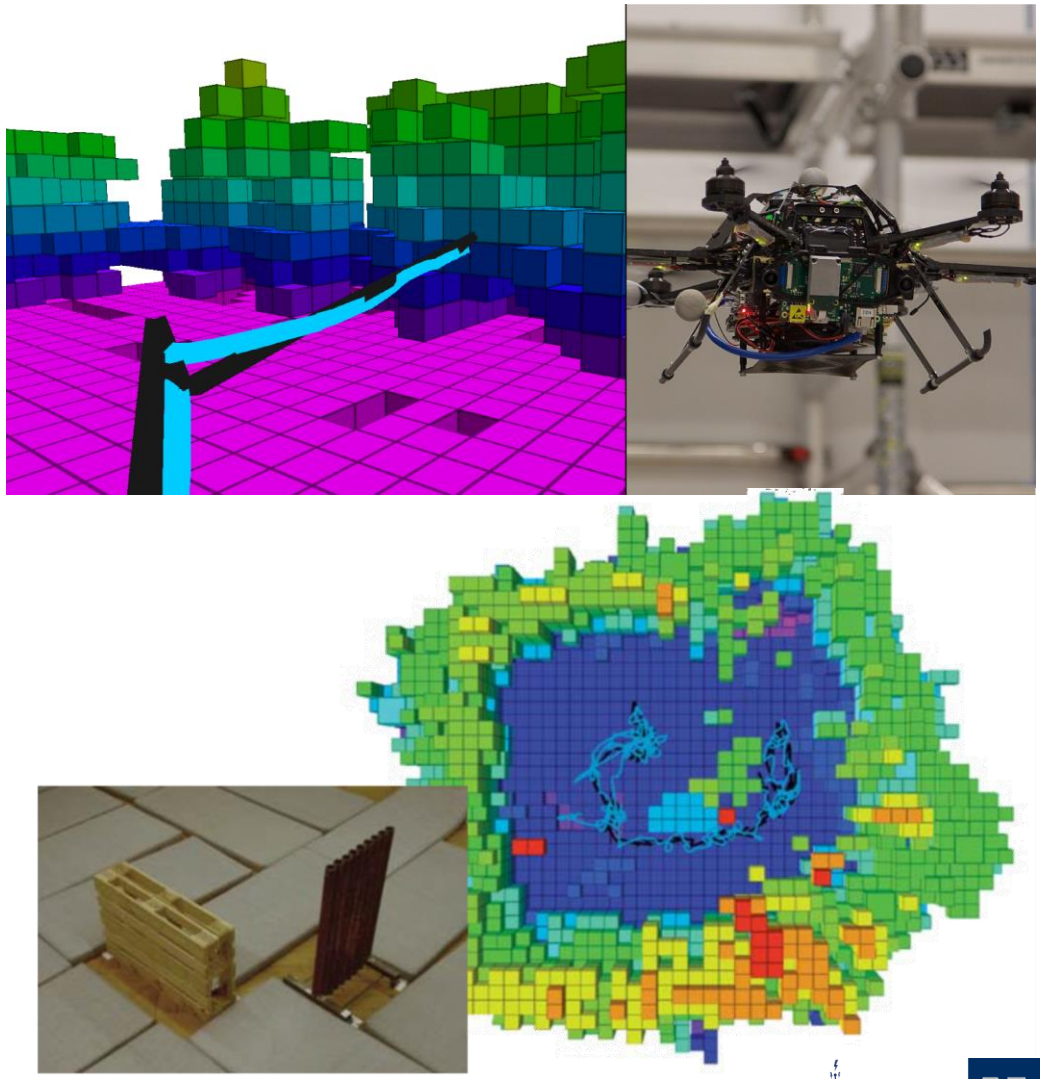
Problem Definition: Volumetric Exploration

The exploration path planning problem consists in **exploring a previously unknown bounded 3D space** $V \subset \mathbb{R}^3$. This is to determine which parts of the initially unmapped space $V_{unm} = V$ are free $V_{free} \subset V$ or occupied $V_{occ} \subset V$. The operation is subject to vehicle kinematic and dynamic constraints, localization uncertainty and limitations of the employed sensor system with which the space is explored.

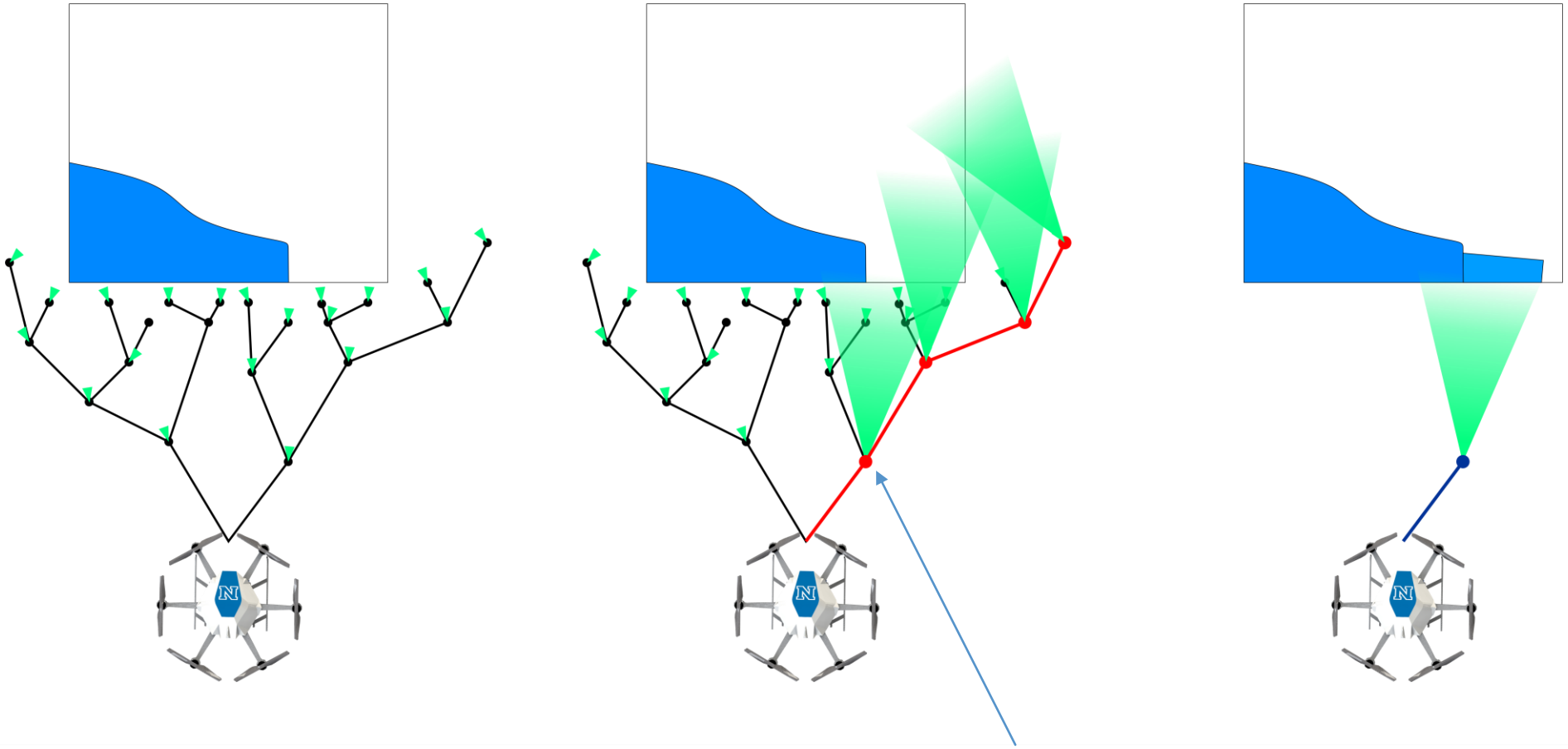
- As for most sensors the perception stops at surfaces, hollow spaces or narrow pockets can sometimes not be explored with a given setup. This residual space is denoted as V_{res}. The problem is considered to be fully solved when $V_{free} \cup V_{occ} = V \backslash V_{res}$.
- Due to the nature of the problem, a suitable path has to be computed online and in real-time, as free space to navigate is not known prior to its exploration.

Receding Horizon Next-Best-View Exploration

- **Goal:** Fast and complete exploration of unknown environments.
- Define **sequences of viewpoints based on vertices sampled using random trees**.
- Select the path with the best sequence of best views.
- Execute only the first step of this best exploration path.
- Update the map after each iteration.
- Repeat the whole process in a receding horizon fashion.



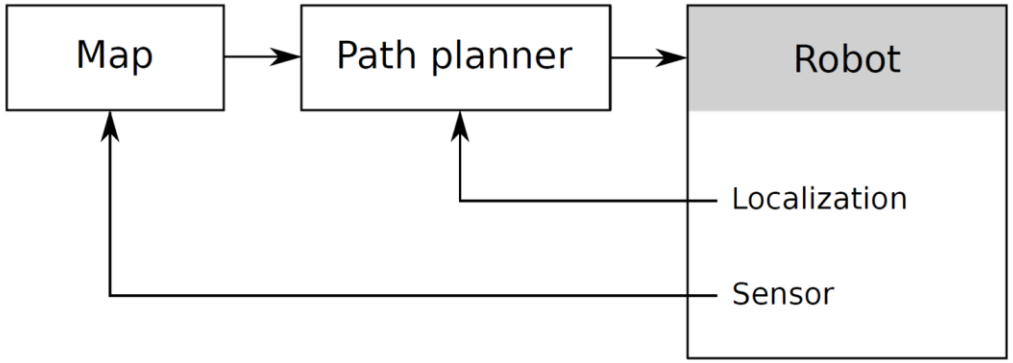
Exploration Planning (**`nbvplanner`**)



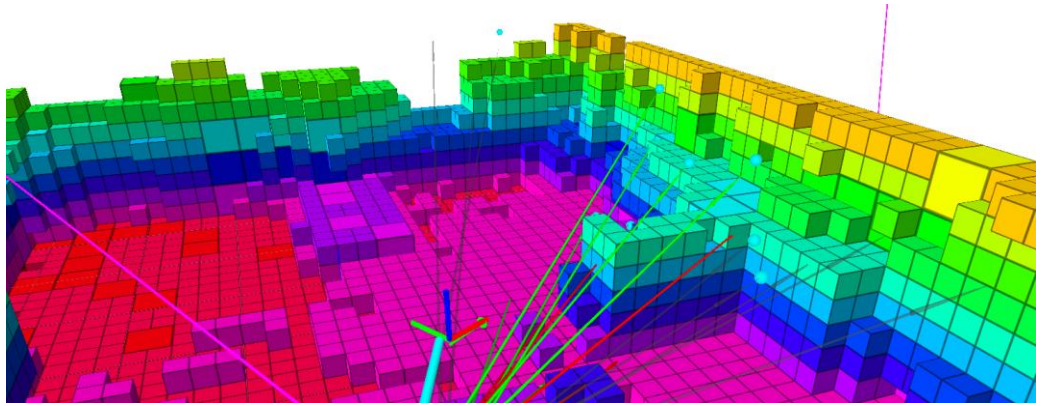
$$\mathbf{Gain}(n_k) = \mathbf{Gain}(n_{k-1}) + \mathbf{Visible}(\mathcal{M}, \xi_k) e^{-\lambda c(\sigma_{k-1}^k)}$$

Exploration Planning (`nbvplanner`)

- **Environment representation:** Occupancy Map dividing space V into $m \in M$ cubical volumes (voxels) that can be marked either as free, occupied or unmapped.
- Use of the `octomap` representation to enable computationally efficient access and search.
- Paths are planned only within the free space V_{free} and collision free point-to-point navigation is inherently supported.
- At each viewpoint/configuration of the environment ξ, the amount of space that is visible is computed as $Visible(M, \xi)$



The Receding Horizon Next-Best-View Exploration Planner relies on the real-time update of the 3D map of the environment.

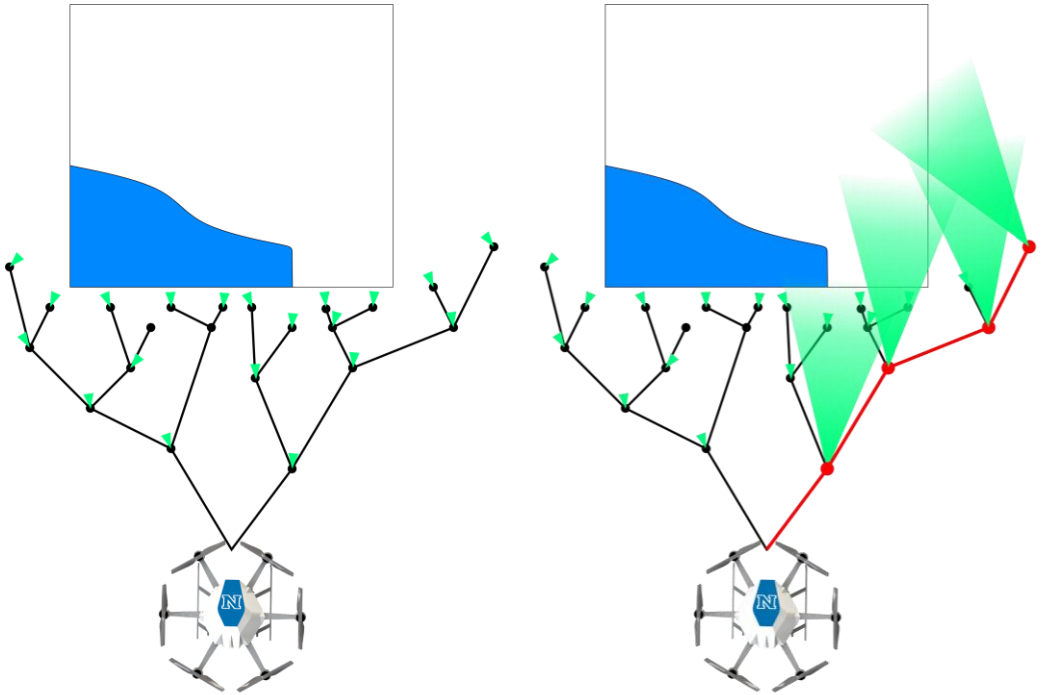


Exploration Planning (`nbvplanner`)

- **Tree-based exploration:** At every iteration, the `nbvplanner` spans a random tree of finite depth. Each vertex of the tree is annotated regarding the collected Information Gain – a metric of how much new space is going to be explored.

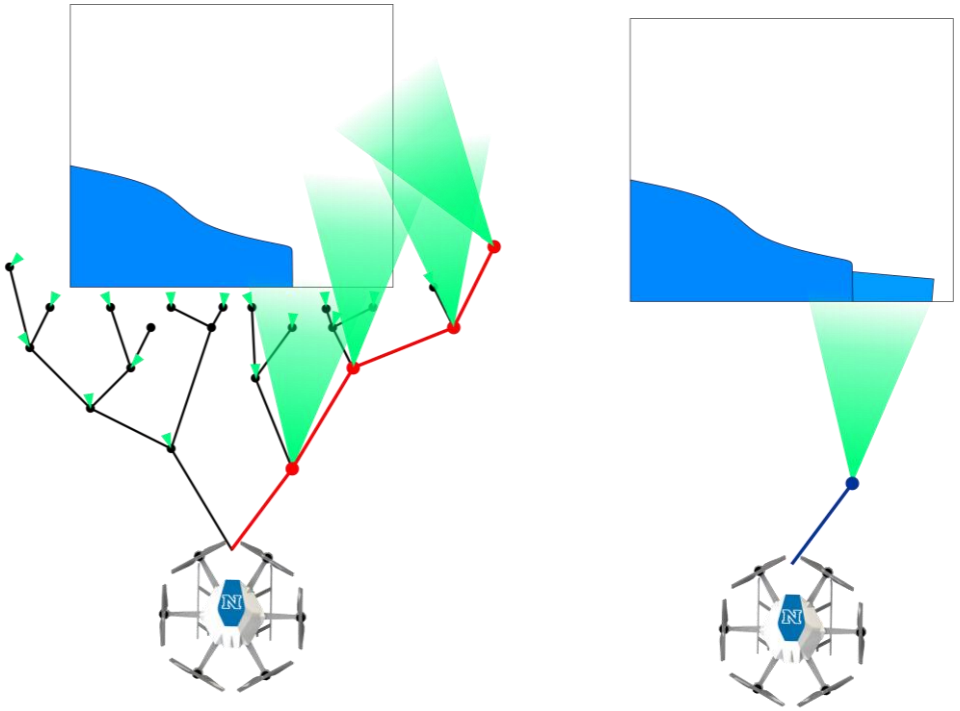
$$\mathbf{Gain}(n_k) = \mathbf{Gain}(n_{k-1}) + \mathbf{Visible}(\mathcal{M}, \xi_k) e^{-\lambda c(\sigma_{k-1}^k)}$$

- Within the sampled tree, evaluation regarding the path that overall leads to the highest information gain is conducted. This corresponds to the **best path** for the given iteration. It is a sequence of next-best-views as sampled based on the vertices of the spanned random tree.



Exploration Planning (`nbvplanner`)

- **Receding Horizon:** For the extracted best path of viewpoints, only the first viewpoint is actually executed.
- The system moves to the first viewpoint of the path of best viewpoints.
- The map is subsequently updated.
- Subsequently, the whole process is repeated within the next iteration. This gives rise to a receding horizon operation.



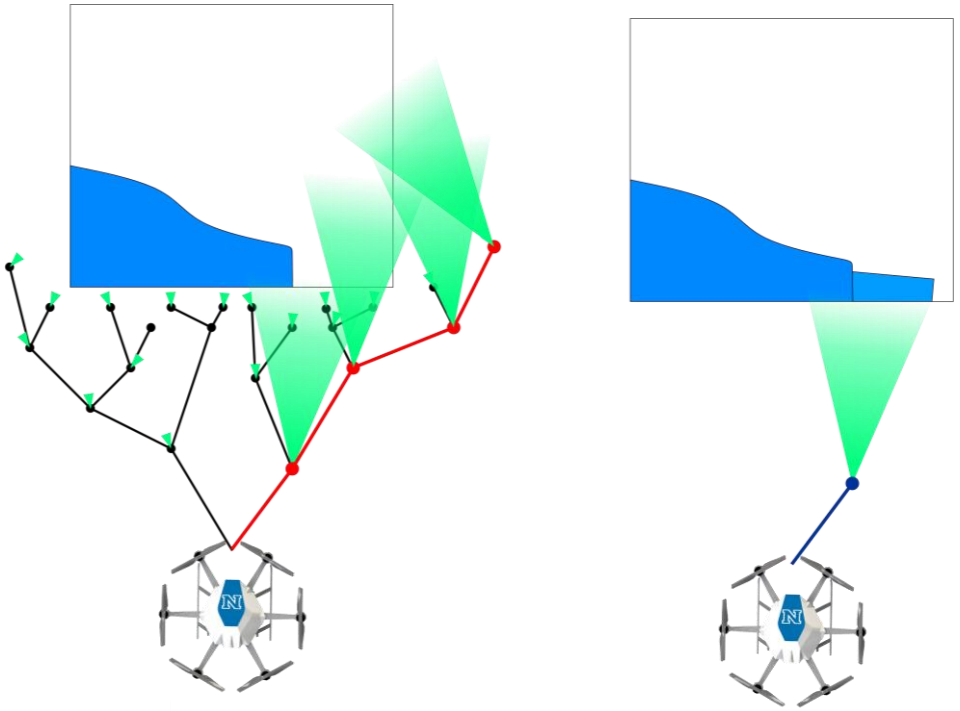
Exploration Planning (**nbvplanner**) Algorithm

nbvplanner Iterative Step

- $\xi_0 \leftarrow$ current vehicle configuration
- Initialize $\boldsymbol{T}$ with ξ_0 and, unless first planner call, also previous best branch
- $g_{best} \leftarrow 0$ // Set best gain to zero
- $n_{best} \leftarrow n_0(\xi_0)$ // Set best node to root
- $N_T \leftarrow$ Number of nodes in $\boldsymbol{T}$
- **while** $N_T < N_{max}$ or $g_{best} == 0$ **do**
  - Incrementally build $\boldsymbol{T}$ by adding $n_{new}(\xi_{new})$
  - $N_T \leftarrow N_T + 1$
  - **if** $Gain(n_{new}) > g_{best}$ **then**
    - $n_{best} \leftarrow n_{new}$
    - $g_{best} \leftarrow Gain(n_{new})$
  - **if** $N_T > N_{TOT}$ **then**
    - Terminate exploration
- $\sigma \leftarrow \boldsymbol{ExtractBestPathSegment}(n_{best})$
- Delete $\boldsymbol{T}$
- **return** σ

Exploration Planning (**`nbvplanner`**) Remarks

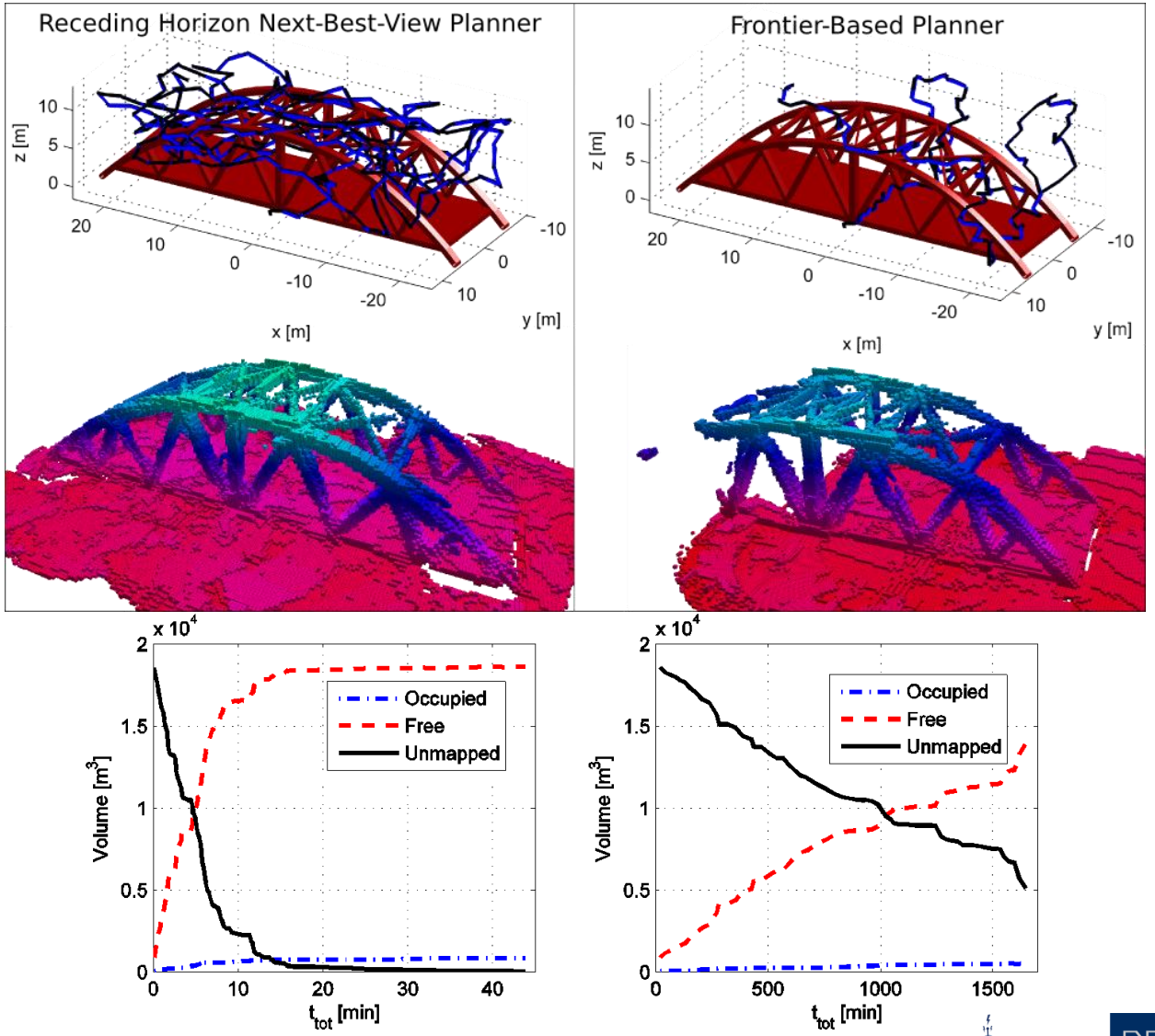
- **Inherently Collision-free:** As all paths of `nbvplanner` are selected along branches within RRT-based spanned trees, all paths are inherently collision-free.
- **Computational Cost:** nbvplanner has a thin structure and most of the computational cost is related with collision-checking functionalities. The formula that expresses the complexity of the algorithm takes the form:



$$\mathcal{O}(N_{\mathbb{T}} \log(N_{\mathbb{T}}) + N_{\mathbb{T}}/r^3 \log(V/r^3) + N_{\mathbb{T}}(d_{\max}^{\text{planner}}/r)^4 \log(V/r^3))$$

`nbvplanner` Evaluation (Simulation)

- **Simulation-based evaluation:** Explore a bridge.
- Comparison with Frontier-based exploration.



Extension to Surface Inspection

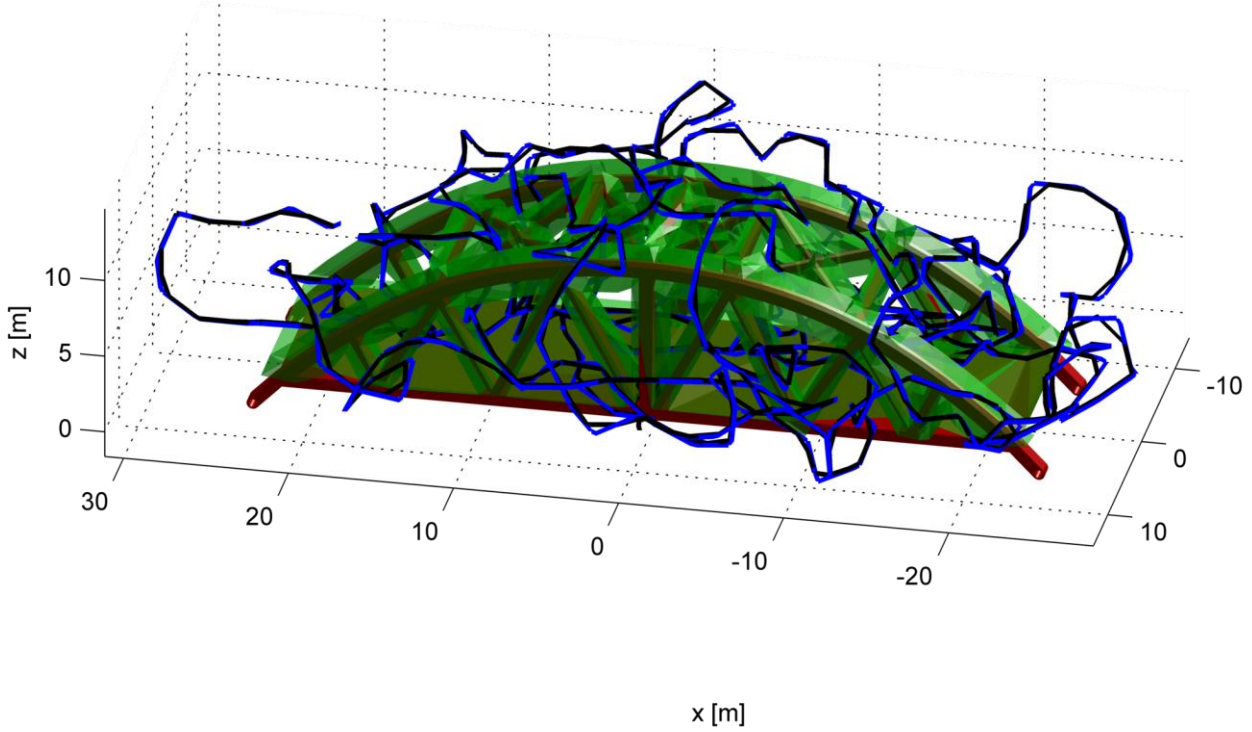
Problem Definition: Surface Inspection

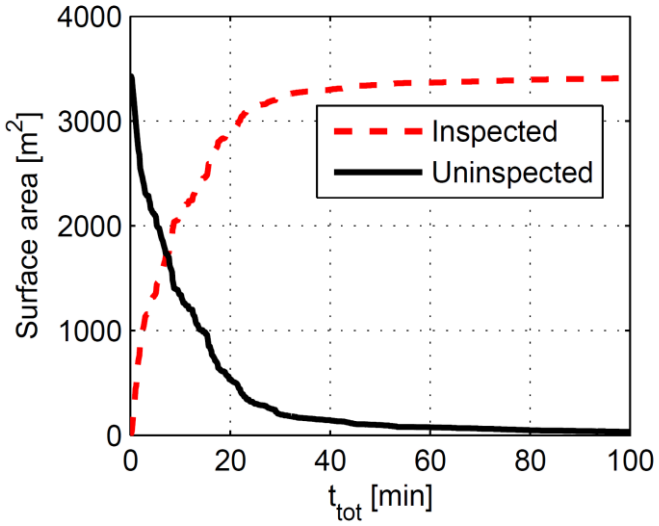
Given a surface S, find a collision free path σ starting at an initial configuration $\xi_{init} \in \Xi$ that leads to the inspection of the part S_{insp}, when being executed, such that there does not exist any collision free configuration from which any piece of $S \backslash S_{insp}$ could be inspected. Thus, $S_{insp} = S \backslash S_{res}$.

- Let $\overline{V}_s \subseteq \Xi$ be the set of all configurations from which the surface piece $s \subseteq S$ can be inspected. Then the residual surface is given as $S_{res} = \cup_{s \in S} (s | \overline{V}_s = 0)$

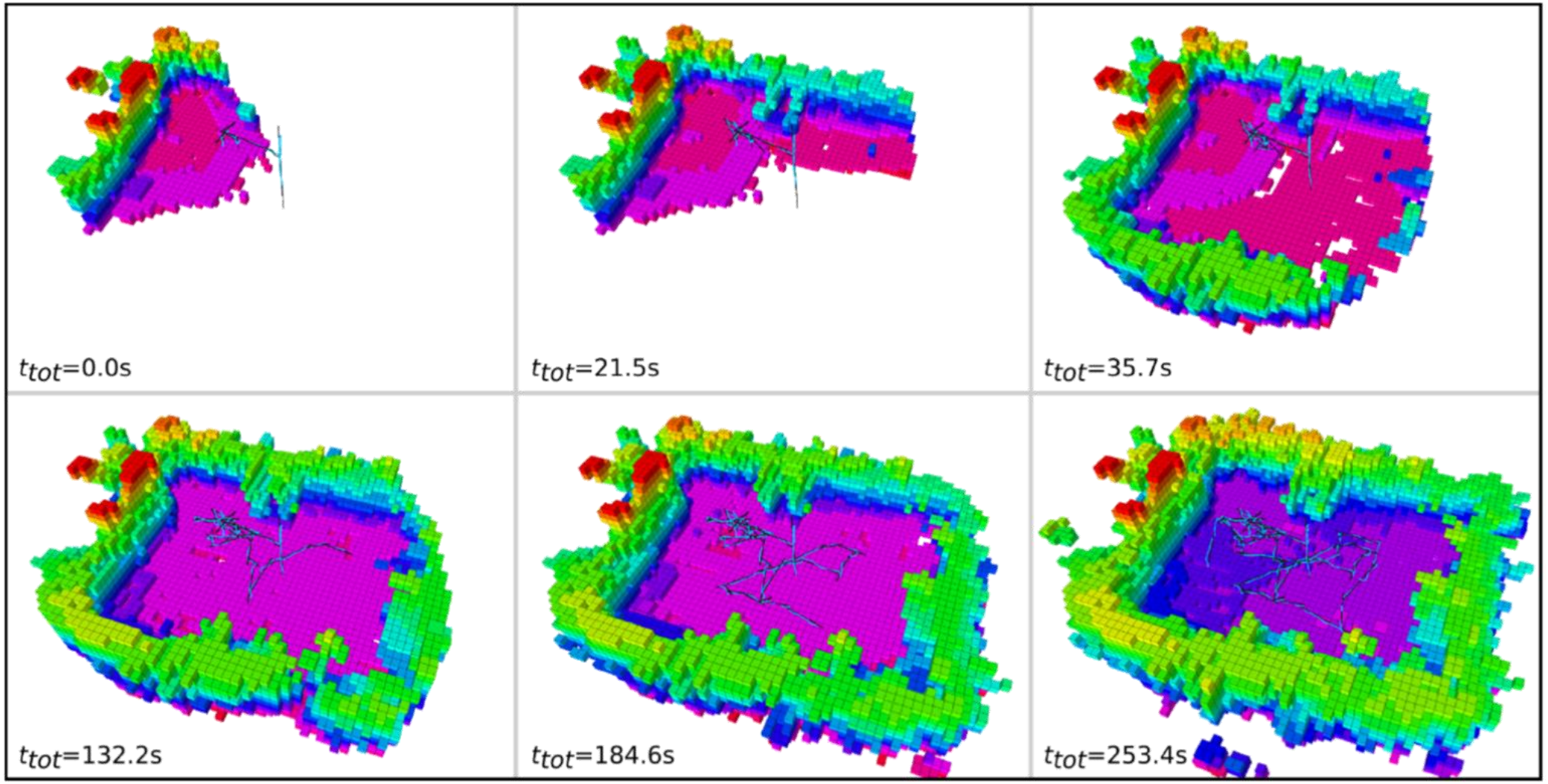
`nbvplanner` Evaluation (Simulation)

- **Extension to surface inspection:** The robot identifies trajectories that locally ensure maximum information gain regarding surface coverage.





nbvplanner Evaluation (Experiment)



BadgerWorks Lectures

Primer Series - Exploration Planning under Uncertainty

Kostas Alexis

AUTONOMOUS ROBOTS LAB

Proposed Planning Ensemble

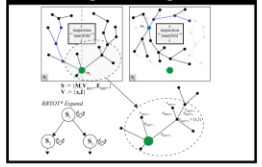
- **Classification:**
  - **NO Prior environmental knowledge**
  - **Active perception and belief-space planning**
  - **Possible human co-working?**
  - **Applicable to "any" robot configuration?**

- **Robot evaluation:**
  - **Multi-rotor UAV systems**
  - **Fixed-wing UAV systems**

Coverage Path Planning

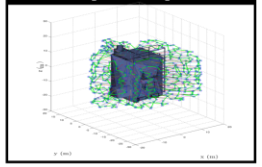
Exploration Path Planning

Optimal Coverage (RRTOT)

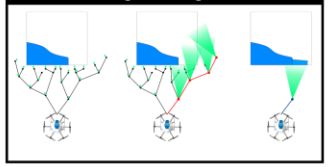


Efficient Coverage (SIP)

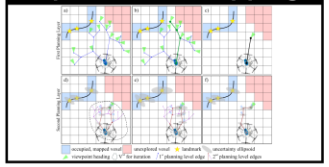
Uniform Coverage (UC3D)



Autonomous Exploration (NBVP)



Uncertainty-aware Exploration & Mapping



Augmented Reality Inspection

Augmented Reality-enhanced Inspection

Uncertainty-aware Exploration & Mapping

Problem Definition

The overall problem is that of **exploring an unknown bounded 3D volume** $V^E \subset \mathbb{R}^3$, **while aiming to minimize the localization and mapping uncertainty** as evaluated through a metric over the robot pose and landmarks **probabilistic belief**.

Problem 1: Volumetric Exploration

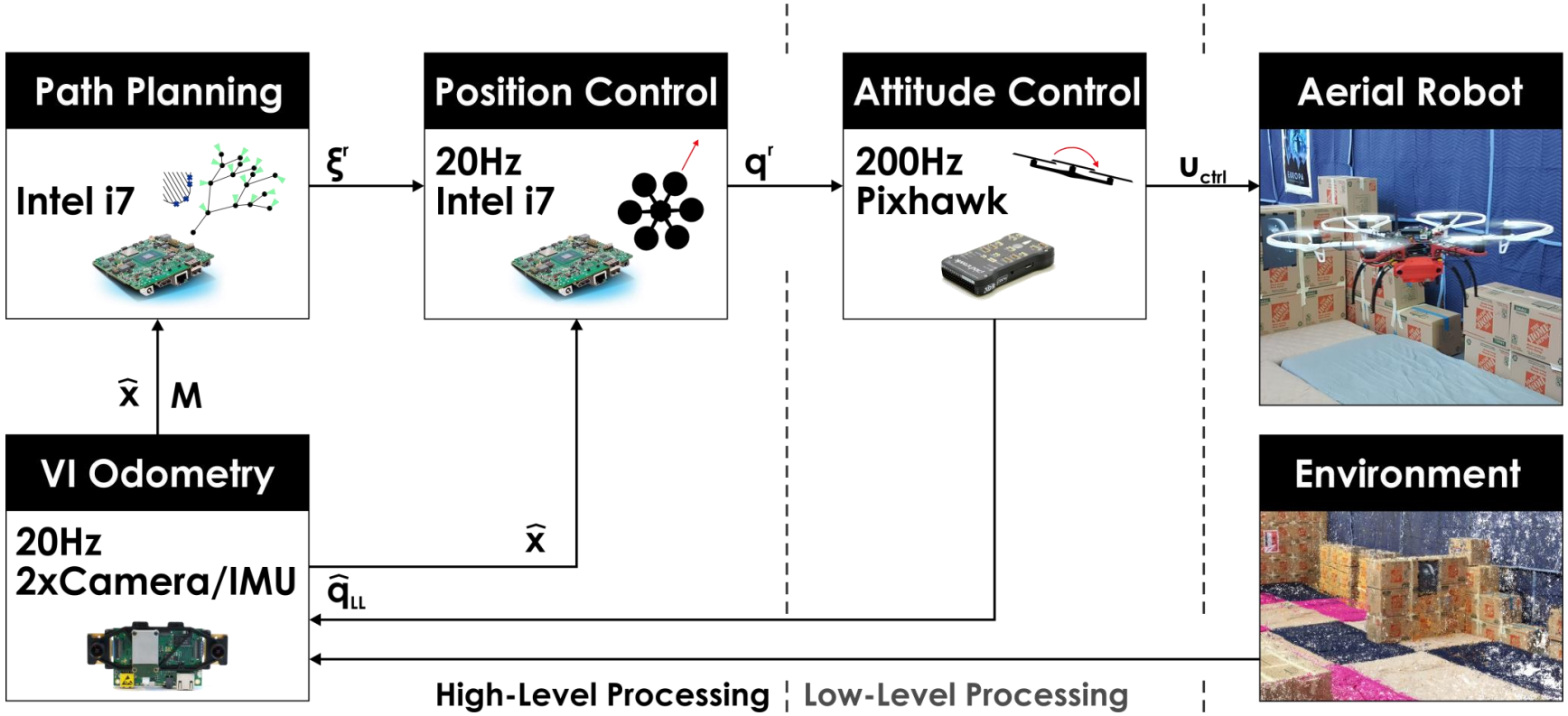
Given a bounded volume V^E, find a collision free path σ starting at an initial configuration $\xi_{init} \in \Xi$that leads to identifying the free and occupied parts V^E_{free} and V^E_{occ} when being executed, such that there does not exist any collision free configuration from which any piece of $V^E\{V^E_{free}, V^E_{occ}\}$ could be perceived.

Combined Problem

Problem 2: Belief Uncertainty-aware planning

Given a $V^M \subset V^E$, find a collision free path σ^M starting at an initial configuration $\xi_0 \in \Xi$ and ending in a configuration $\xi_{final} \in \Xi$ that aims to improve the robot's localization and mapping confidence by following paths of optimized expected robot pose and tracked landmarks covariance.

Recall Robot Configuration

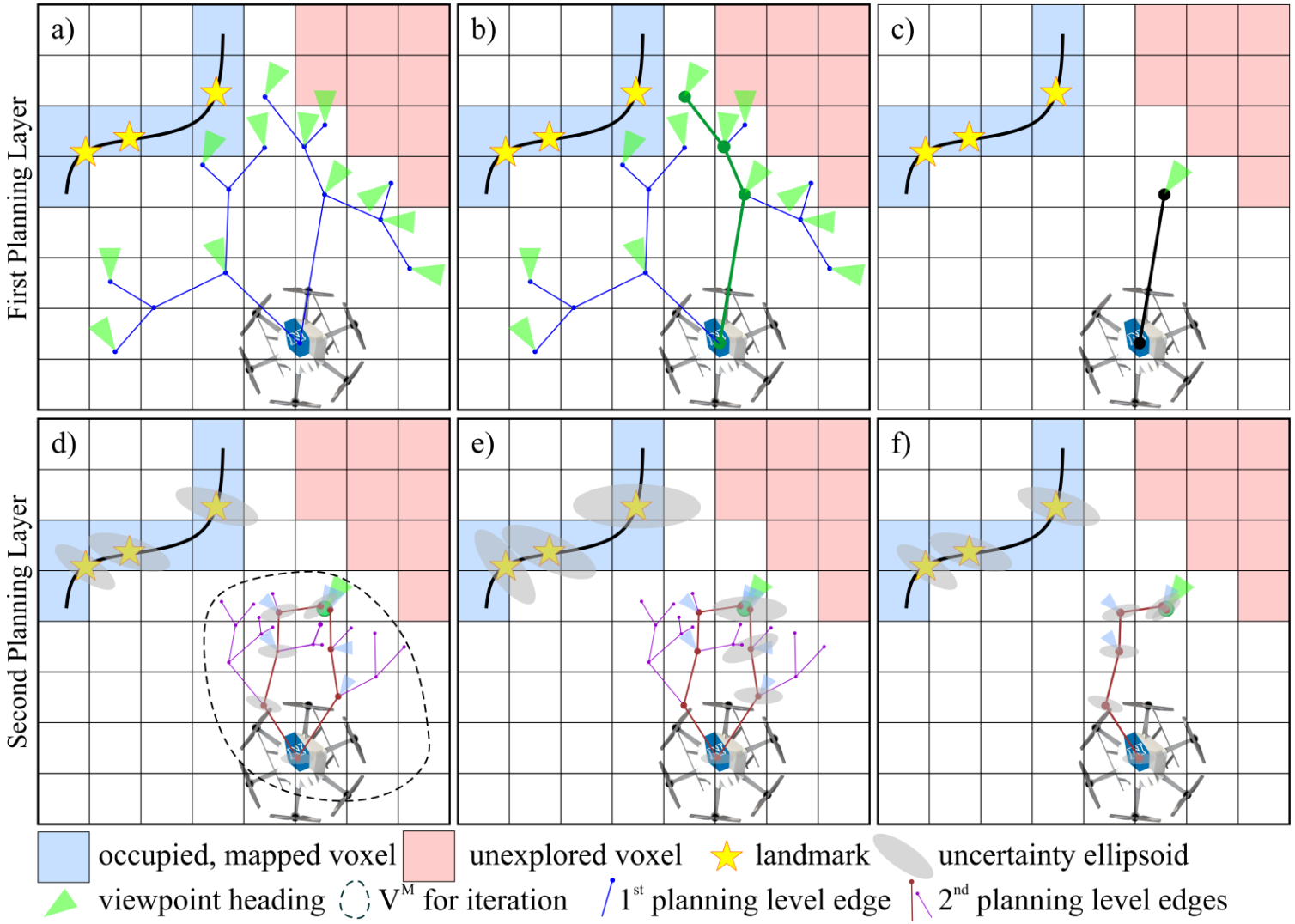


Uncertainty-aware Exploration & Mapping

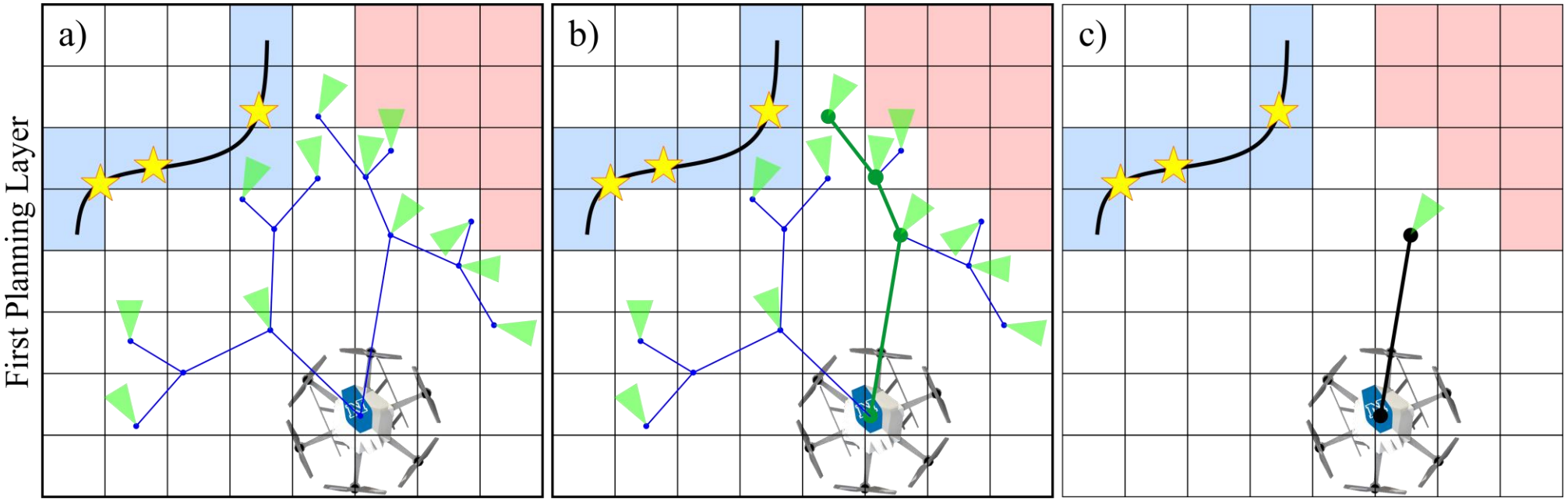
Receding Horizon Exploration and Mapping Planner (rhemplanner)

Two-levels Path Planning paradigm

Broader topic: Planning under uncertainty



`rhemplanner` - Exploration Step

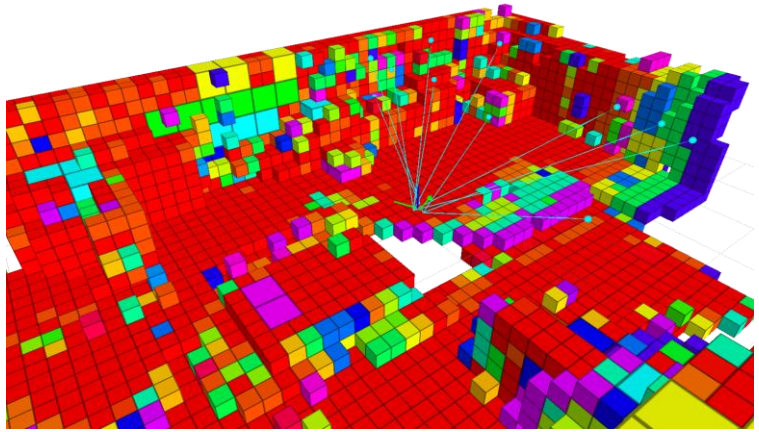


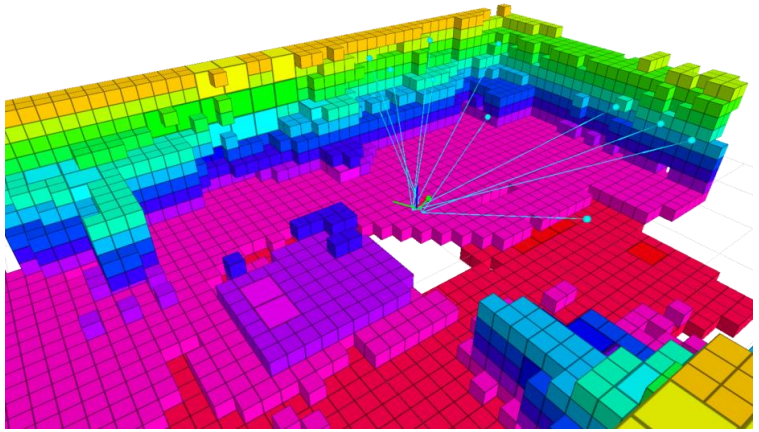
`rhemplanner` - Exploration Step

- **Exploration Gain** with **probabilistic reobservation**

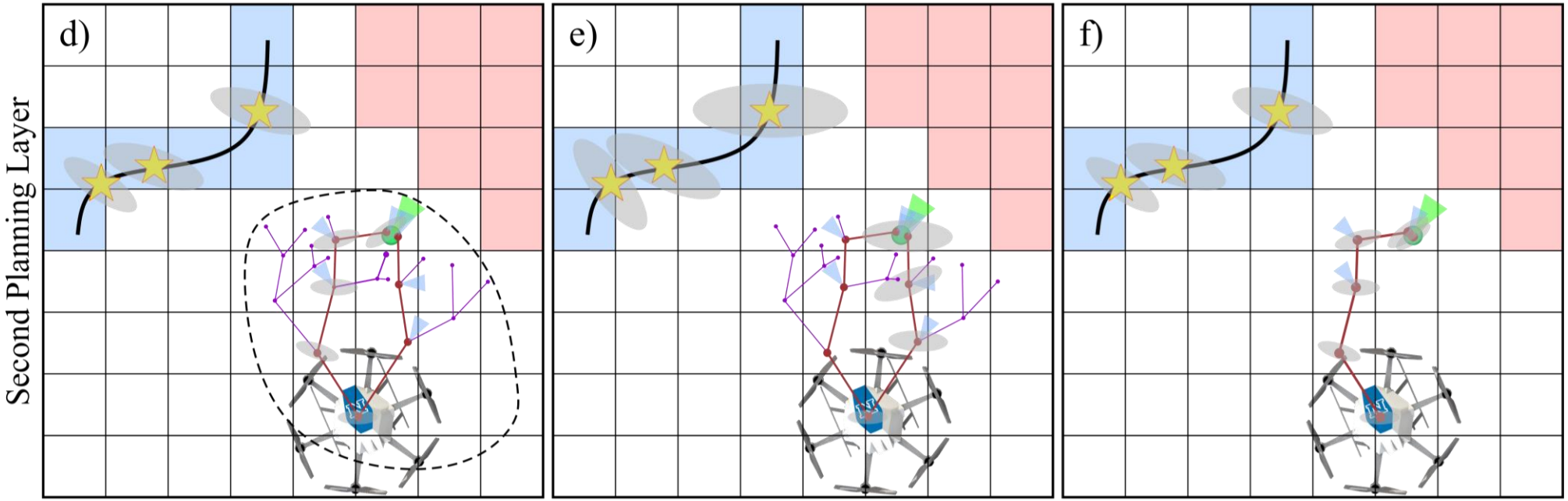
$$\mathbf{ExplorationGain}(n_k^E) = \mathbf{ExplorationGain}(n_{k-1}^E) + \mathbf{VisibleVolume}(\mathcal{M}, \xi_k)\exp(-\lambda c(\sigma_{k-1,k}^E)) + \mathbf{ReobservationGain}(\mathcal{M}, \mathcal{P}, \xi_k)\exp(-\lambda c(\sigma_{k-1,k}^E))$$

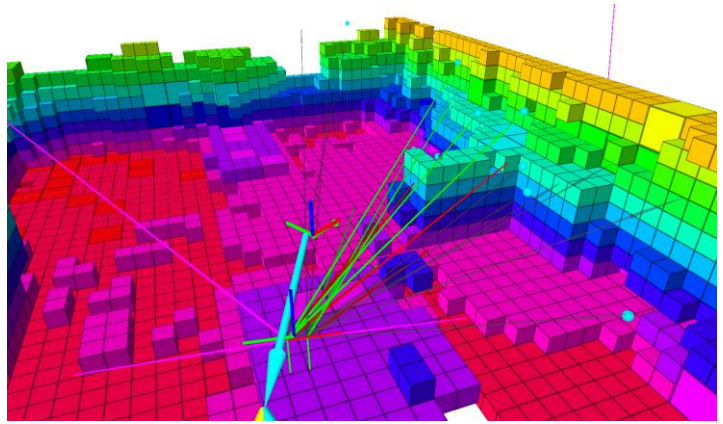
  - Aiming to maximize newly explored space and reobserve space with decreased confidence of being mapped as occupied.

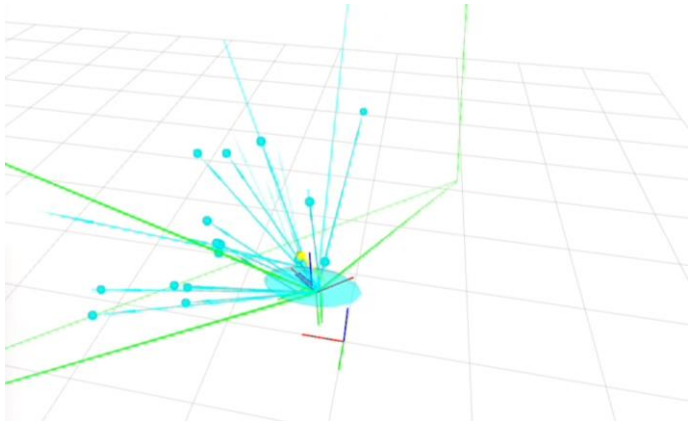




`rhemplanner` - Uncertainty-aware Step







`rhemplanner` - Uncertainty-aware Step

- The robot performs **onboard localization and mapping**
  - For the case of our experiments it performs visual-inertial localization
    - **Can be generalized to different cases given a filter-based approach**
  - The assumptions are:
    - Pose, features and their uncertainties are estimated
    - Dense, volumetric mapping takes place
- To get an estimate about its pose, it relies on tracking landmarks from its sensor systems. The system performs odometry in an EKF-fashion and the overall state of the filter is:

$$\mathbf{x} = [\underbrace{\overbrace{\mathbf{r}\ \mathbf{q}}^{\text{pose, } l_p}\ \boldsymbol{v}\ \mathbf{b}_f\ \mathbf{b}_\omega\ \mathbf{c}\ \mathbf{z}}_{\text{robot states, } l_s} \mid \underbrace{\boldsymbol{\mu}_0,\ \cdots\ \boldsymbol{\mu}_J\ \rho_0\ \cdots\ \rho_J}_{\text{features states, } l_f}]^T$$

`rhemplanner` - Uncertainty-aware Step

- **Belief Propagation:** in order to identify the paths that minimize the robot uncertainty, a mechanism to propagate the robot belief about its pose and the tracked features has to be established.

Equations adopted from Bloesch et al. "Robust Visual Inertial Odometry Using a Direct EKF-Based Approach"

State Propagation Step - Equations (3)

$$
\begin{aligned}
\dot{\mathbf{r}} &= -\hat{\boldsymbol{\omega}}^{\times}\mathbf{r} + \boldsymbol{v} + \mathbf{w}_r \\
\dot{\boldsymbol{v}} &= -\hat{\boldsymbol{\omega}}^{\times}\boldsymbol{v} + \hat{\mathbf{f}} + \mathbf{q}^{-1}(\mathbf{g}) \\
\dot{\mathbf{q}} &= -\mathbf{q}(\hat{\boldsymbol{\omega}}) \\
\dot{\mathbf{b}}_f &= \mathbf{w}_{bf} \\
\dot{\mathbf{b}}_\omega &= \mathbf{w}_{bw} \\
\dot{\mathbf{c}} &= \mathbf{w}_c \\
\dot{\mathbf{z}} &= \mathbf{w}_z \\
\dot{\boldsymbol{\mu}}_j &= \mathbf{N}^T(\boldsymbol{\mu}_j)\hat{\boldsymbol{\omega}}_{\mathcal{V}} - \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \mathbf{N}^T(\boldsymbol{\mu}_j)\frac{\hat{\boldsymbol{v}}_{\mathcal{V}}}{d(\rho_j)} + \mathbf{w}_{\mu,j} \\
\dot{\rho}_j &= -\boldsymbol{\mu}_j^T \hat{\boldsymbol{v}}_{\mathcal{V}}/d'(\rho_j) + w_{\rho,j}
\end{aligned}
$$

$$
\begin{aligned}
\hat{\mathbf{f}} &= \tilde{\mathbf{f}} - \mathbf{b}_f - \mathbf{w}_f \\
\hat{\boldsymbol{\omega}} &= \tilde{\boldsymbol{\omega}} - \mathbf{b}_\omega - \mathbf{w}_\omega \\
\hat{\boldsymbol{v}}_{\mathcal{V}} &= \mathbf{z}(\boldsymbol{v} + \hat{\boldsymbol{\omega}}^{\times}\mathbf{c}) \\
\hat{\boldsymbol{\omega}}_{\mathcal{V}} &= \mathbf{z}(\hat{\boldsymbol{\omega}})
\end{aligned}
$$

Filter Update Step - Equations (4)

$$
\begin{aligned}
\mathbf{y}_j &= \mathbf{b}_j(\boldsymbol{\pi}(\hat{\boldsymbol{\mu}}_j)) + \mathbf{n}_j \\
\mathbf{H}_j &= \mathbf{A}_j(\boldsymbol{\pi}(\hat{\boldsymbol{\mu}}_j))\frac{d\boldsymbol{\pi}}{d\boldsymbol{\mu}}(\hat{\boldsymbol{\mu}}_j)
\end{aligned}
$$

By stacking the above terms for all visible features, standard EKF update step is directly performed to derive the new estimate of the robot belief for its state and the tracked features.

Notation

$^{\times} \rightarrow$ skew symmetric matrix of a vector, $\tilde{\mathbf{f}} \rightarrow$ proper acceleration measurement, $\tilde{\boldsymbol{\omega}} \rightarrow$ rotational rate measurement, $\hat{\mathbf{f}} \rightarrow$ biased corrected acceleration, $\hat{\boldsymbol{\omega}} \rightarrow$ bias corrected rotational rate, $\mathbf{N}^T(\boldsymbol{\mu}) \rightarrow$ projection of a 3D vector onto the 2D tangent space around the bearing vector, $\mathbf{g} \rightarrow$ gravity vector, $\mathbf{w}_\star \rightarrow$ white Gaussian noise processes, $\boldsymbol{\pi}(\boldsymbol{\mu}) \rightarrow$ pixel coordinates of a feature, $\mathbf{b}_i(\boldsymbol{\pi}(\hat{\boldsymbol{\mu}}_j)) \rightarrow$ a 2D linear constraint for the jth feature which is predicted to be visible in the current frame with bearing vector $\hat{\boldsymbol{\mu}}_j$

`rhemplanner` - Uncertainty-aware Step

Propagate the robot's belief about its pose and the tracked landmarks: Prediction step

State Propagation Step

$$
\begin{aligned}
\dot{\mathbf{r}} &= -\hat{\boldsymbol{\omega}}^{\times}\mathbf{r} + \boldsymbol{\upsilon} + \mathbf{w}_r \\
\dot{\boldsymbol{\upsilon}} &= -\hat{\boldsymbol{\omega}}^{\times}\boldsymbol{\upsilon} + \hat{\mathbf{f}} + \mathbf{q}^{-1}(\mathbf{g}) \\
\dot{\mathbf{q}} &= -\mathbf{q}(\hat{\boldsymbol{\omega}}) \\
\dot{\mathbf{b}}_f &= \mathbf{w}_{bf} \\
\dot{\mathbf{b}}_\omega &= \mathbf{w}_{bw} \\
\dot{\mathbf{c}} &= \mathbf{w}_c \\
\dot{\mathbf{z}} &= \mathbf{w}_z \\
\dot{\boldsymbol{\mu}}_j &= \mathbf{N}^T(\boldsymbol{\mu}_j)\hat{\boldsymbol{\omega}}_{\mathcal{V}} - \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \mathbf{N}^T(\boldsymbol{\mu}_j)\frac{\hat{\boldsymbol{\upsilon}}_{\mathcal{V}}}{d(\rho_j)} + \mathbf{w}_{\mu,j} \\
\dot{\rho}_j &= -\boldsymbol{\mu}_j^T\hat{\boldsymbol{\upsilon}}_{\mathcal{V}}/d'(\rho_j) + w_{\rho,j}
\end{aligned}
$$

$$
\begin{aligned}
\hat{\mathbf{f}} &= \tilde{\mathbf{f}} - \mathbf{b}_f - \mathbf{w}_f \\
\hat{\boldsymbol{\omega}} &= \tilde{\boldsymbol{\omega}} - \mathbf{b}_\omega - \mathbf{w}_\omega \\
\hat{\boldsymbol{\upsilon}}_{\mathcal{V}} &= \mathbf{z}(\boldsymbol{\upsilon} + \hat{\boldsymbol{\omega}}^{\times}\mathbf{c}) \\
\hat{\boldsymbol{\omega}}_{\mathcal{V}} &= \mathbf{z}(\hat{\boldsymbol{\omega}})
\end{aligned}
$$

`rhemplanner` - Uncertainty-aware Step

Propagate the robot's belief about its pose and the tracked landmarks: Update step

Filter Update Step

$$\mathbf{y}_j = \mathbf{b}_j(\boldsymbol{\pi}(\hat{\boldsymbol{\mu}}_j)) + \mathbf{n}_j$$

$$\mathbf{H}_j = \mathbf{A}_j(\boldsymbol{\pi}(\hat{\boldsymbol{\mu}}_j))\frac{d\boldsymbol{\pi}}{d\boldsymbol{\mu}}(\hat{\boldsymbol{\mu}}_j)$$

By stacking the above terms for all **visible landmarks**, standard EKF update step is directly performed to derive the new estimate of the robot belief for its state and the tracked features.

- Identify which viewpoints are expected to be visible given the next pose and the known map

$$\begin{bmatrix} X_j \\ Y_j \\ Z_j \end{bmatrix} = \mathbf{R}_C^W \left(\mathbf{K}_I^{-1} \frac{1}{\rho_j} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \right) + \mathbf{T}_C^W$$

- Compute the propagated belief covariance matrix

$$\boldsymbol{\Sigma}_{t,l} = (\mathbf{I} - \mathbf{K}_t \mathbf{H}_t)\bar{\boldsymbol{\Sigma}}_{t,l}$$

`rhemplanner` - Uncertainty-aware Step

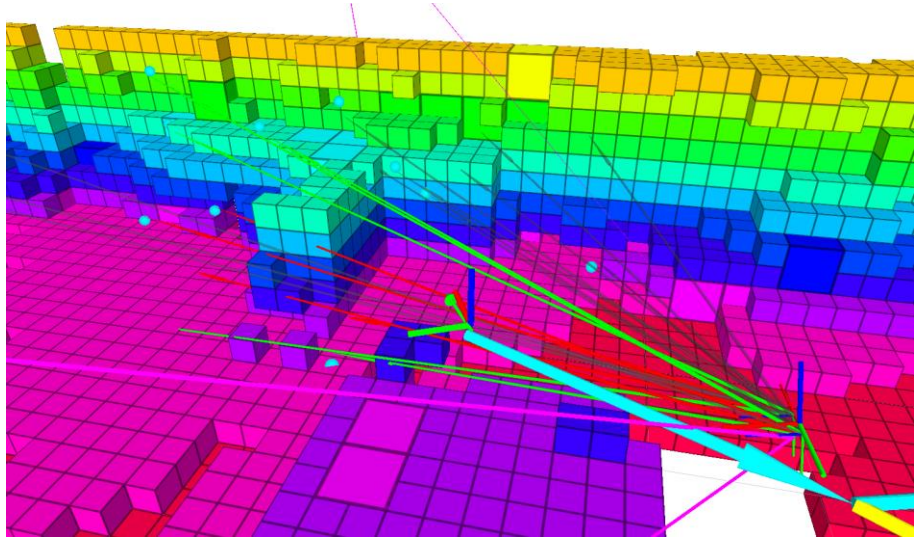
Propagate the robot's belief about its pose and the tracked landmarks: Update step

Filter Update Step

$$\mathbf{y}_j = \mathbf{b}_j(\boldsymbol{\pi}(\hat{\boldsymbol{\mu}}_j)) + \mathbf{n}_j$$

$$\mathbf{H}_j = \mathbf{A}_j(\boldsymbol{\pi}(\hat{\boldsymbol{\mu}}_j))\frac{d\boldsymbol{\pi}}{d\boldsymbol{\mu}}(\hat{\boldsymbol{\mu}}_j)$$

By stacking the above terms for all **visible landmarks**, standard EKF update step is directly performed to derive the new estimate of the robot belief for its state and the tracked features.

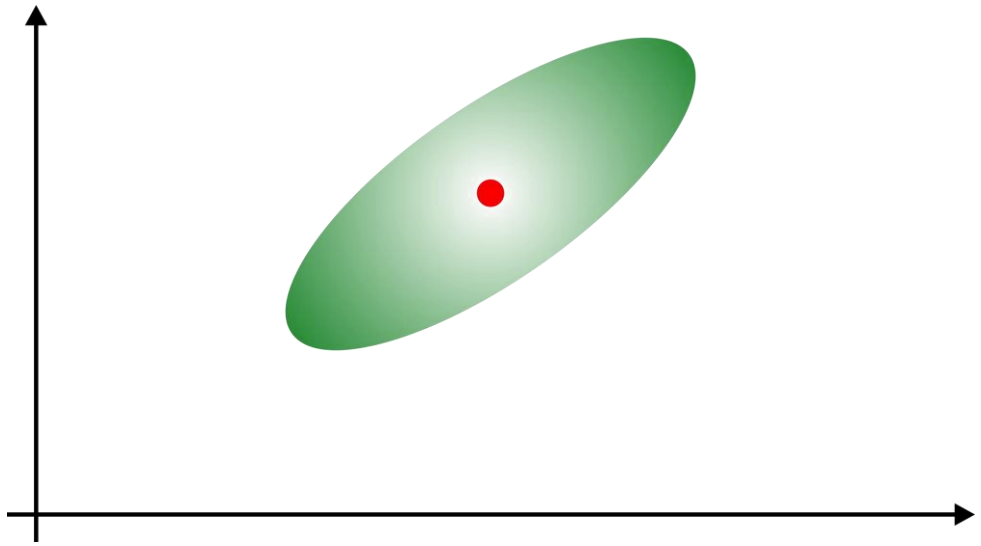


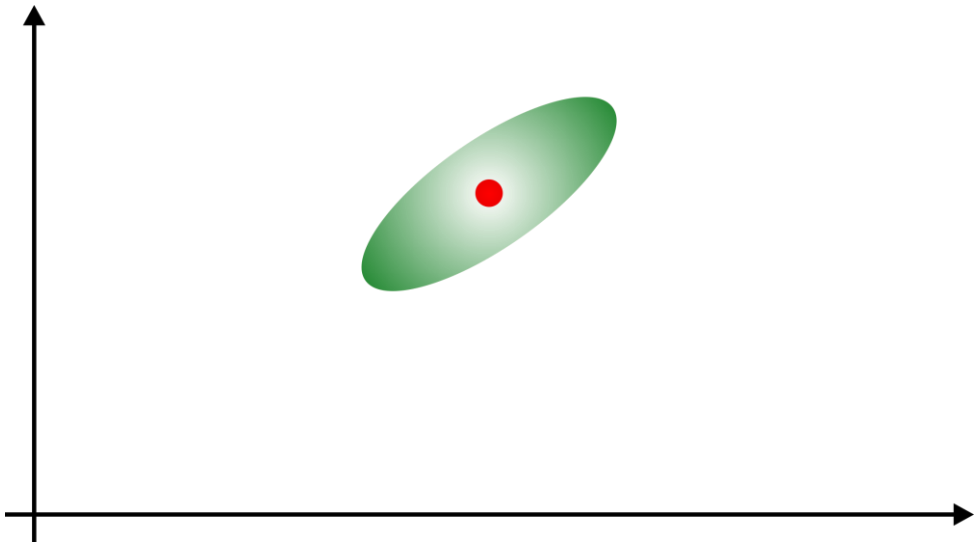
$$\begin{bmatrix} X_j \\ Y_j \\ Z_j \end{bmatrix} = \mathbf{R}_C^W \left(\mathbf{K}_I^{-1} \frac{1}{\rho_j} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \right) + \mathbf{T}_C^W$$

$$\boldsymbol{\Sigma}_{t,l} = (\mathbf{I} - \mathbf{K}_t \mathbf{H}_t)\bar{\boldsymbol{\Sigma}}_{t,l}$$

`rhemplanner` - Uncertainty-aware Step

- **Uncertainty optimization:** to be able to derive which path minimizes the robot uncertainty about its pose and the tracked landmarks, a metric of how small the covariance ellipsoid is has to be defined.
  - **What metric?**





`rhemplanner` - Uncertainty-aware Step

- **Uncertainty optimization:** to be able to derive which path minimizes the robot uncertainty about its pose and the tracked landmarks, a metric of how small the covariance ellipsoid is has to be defined.
  - **D-optimality metric:**

$$D_{opt}(\sigma^M) = \exp(\log([\det(\mathbf{\Sigma}_{p,f}(\sigma^M)]^{1/(l_p+l_f)}))$$

$$\mathbf{BeliefGain}(\sigma_\alpha^M) \quad = \quad D_{opt}(\sigma_\alpha^M)$$

Broadly: maximize the determinant of the information matrix X'X of the design. This criterion results in maximizing the differential Shannon information content of the parameter estimates.

rhemplanner Algorithm

First Planning Step

- $\xi_0 \leftarrow$ current vehicle configuration
- Initialize $\boldsymbol{T}^E$ with ξ_0
- $g^E_{best} \leftarrow 0$ // Set best exploration gain to zero
- $n_{best} \leftarrow n_0(\xi_0)$ // Set best best exploration node to root
- $N^E_T \leftarrow$ Number of nodes in $\boldsymbol{T}^E$
- **While** $N^E_T < N^E_{max}$ or $g^E_{best} == 0$ **do**
  - Incrementally build $\boldsymbol{T}^E$ by adding $n^E_{new}(\xi^E_{new})$
  - $N^E_T \leftarrow N^E_T + 1$
  - **if** Exploration$Gain(n^E_{new}) > g^E_{best}$ **then**
    - $n^E_{new} \leftarrow n^E_{new}$
    - $g^E_{best} \leftarrow$ Exploration$Gain(n^E_{new})$
  - **if** $N^E_T > N^E_{TOT}$ **then**
    - Terminate planning
- $\sigma^E_{RH}, n^E_{RH}, \xi_{RH} \leftarrow \boldsymbol{ExtractBestPathSegment}(n_{best})$
- $S_{\xi_{RH}} \leftarrow LocalSet(\xi_{RH})$

rhemplanner Algorithm

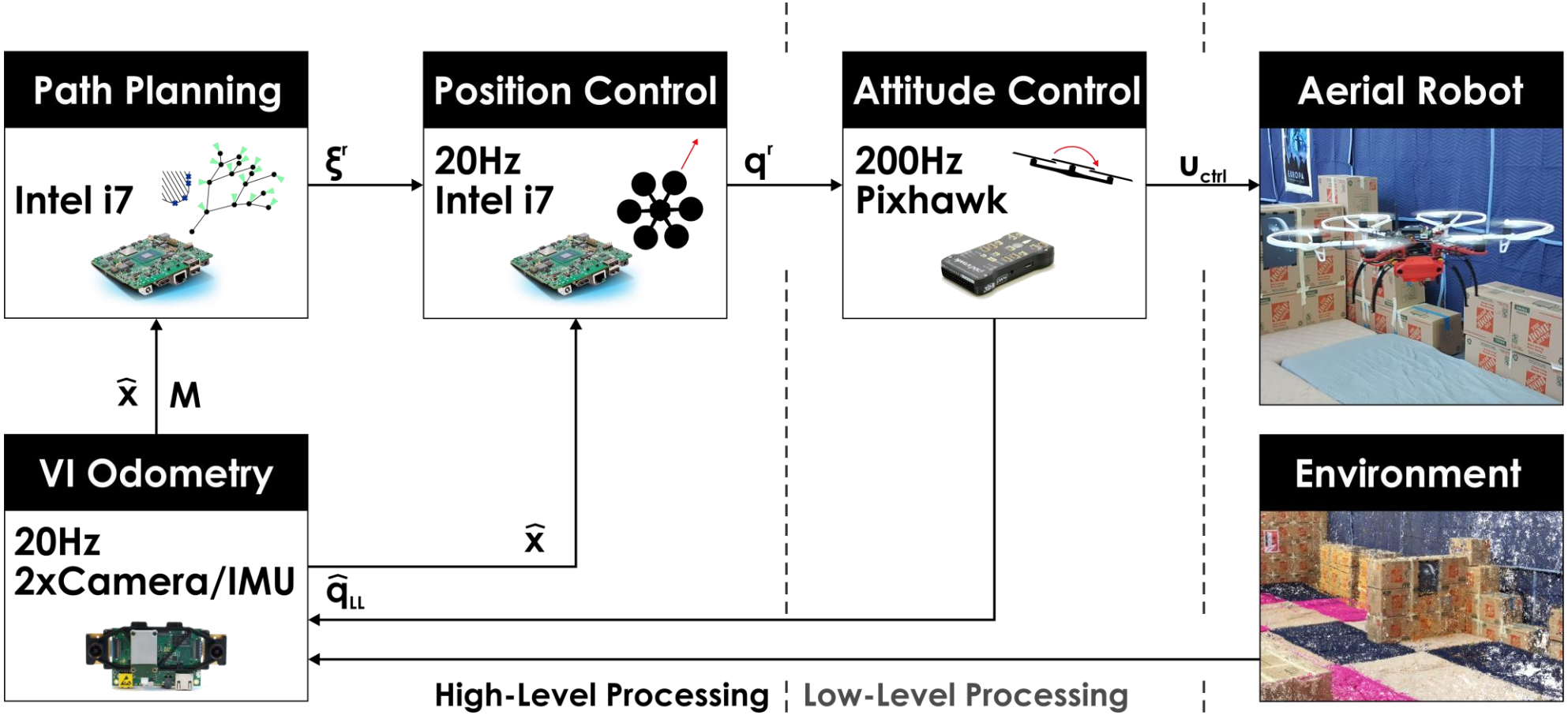
Second Planning Step

- Propagate robot belief along σ_{RH}^{E}
- $a \leftarrow 1$ // number of admissible paths
- $g_a^M \leftarrow BeliefGain(\sigma_{RH}^{E})$
- $g_{best}^M \leftarrow g_a^M$ // straight path belief gain
- $\sigma_{best}^M \leftarrow \sigma_{RH}^M$
- **while** $N_T^M < N_{max}^M$ or $V(T^M) \cap = \emptyset S_{\xi_{RH}}$ **do**
  - Incrementally build $\boldsymbol{T}^M$ by adding $n_{new}^M(\xi_{new})$
  - Propagate robot belief from current to planned vertex
  - **if** $\xi_{new} \in S_{\xi_{RH}}$ **then**
    - Add new vertex n_{new}^M at ξ_{RH} and connect
    - $a \leftarrow a + 1$
    - $\sigma_a^M \leftarrow ExtractBranch(n_{new}^M)$
    - $g_a^M \leftarrow BeliefGain(\sigma_a^M)$
    - **if** $g_a^M < g_{best}^M$ **then**
      - $\sigma^M \leftarrow \sigma_a^M$
      - $g_{best}^M \leftarrow g_a^M$
- **return** σ^M

`rhemplanner` Complexity Analysis

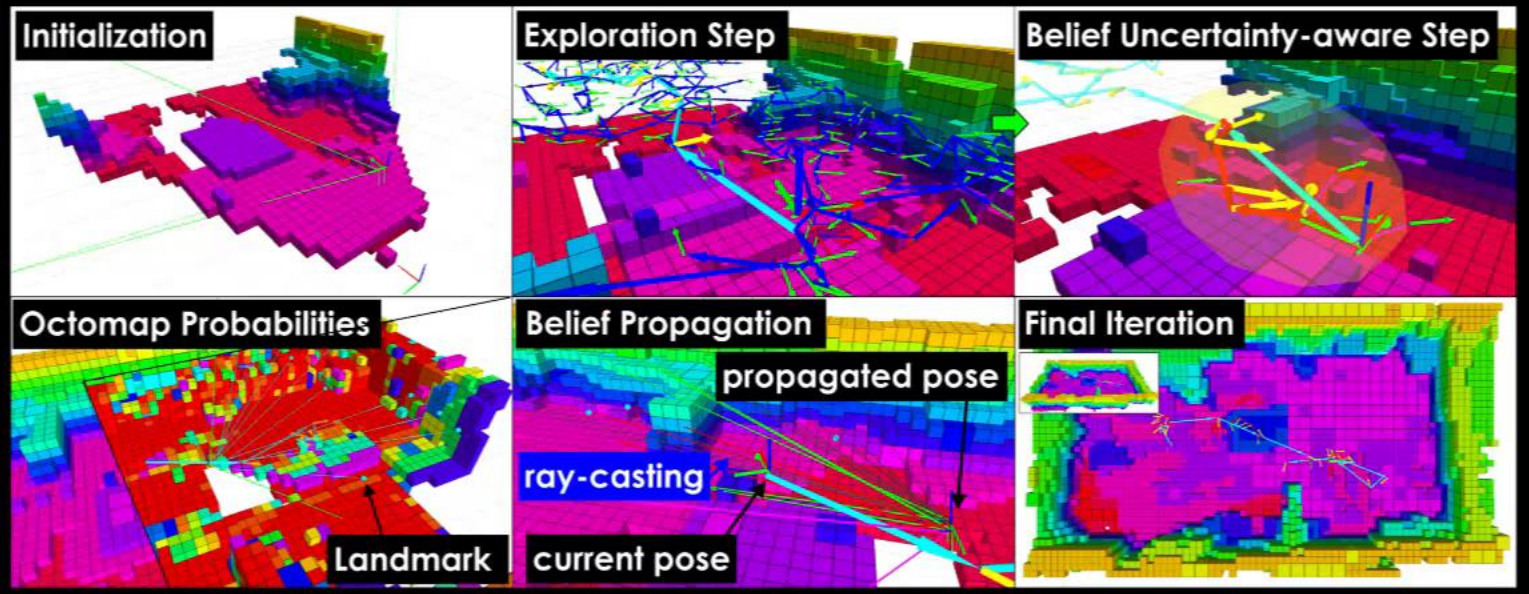
RRT construction	Collision checking
$\mathcal{O}(N_{\mathbb{T}}^{S}\log(N_{\mathbb{T}}^{S}))$, $S \rightarrow E, M$	$\mathcal{O}(N_{\mathbb{T}}^{S}/r^{3}\log(V^{E}/r^{3}))$, $S \rightarrow E, M$
1st planning level gain computation	
$\mathcal{O}(N_{\mathbb{T}}^{E}(d_{\max}^{\text{planner}}/r)^{4}\log(V^{E}/r^{3}))$	
2nd planning level gain computation	
$\mathcal{O}(N_{\mathbb{T}}^{M}(d_{\max}^{\text{sensor}}/r)^{4}\log(V^{E}/r^{3})l_{f} + n_{M}(l_{s}^{2.4} + l_{f}^{2}) + n_{M}(l_{p} + l_{f}))$	

`rhemplanner` Evaluation (Experimental)



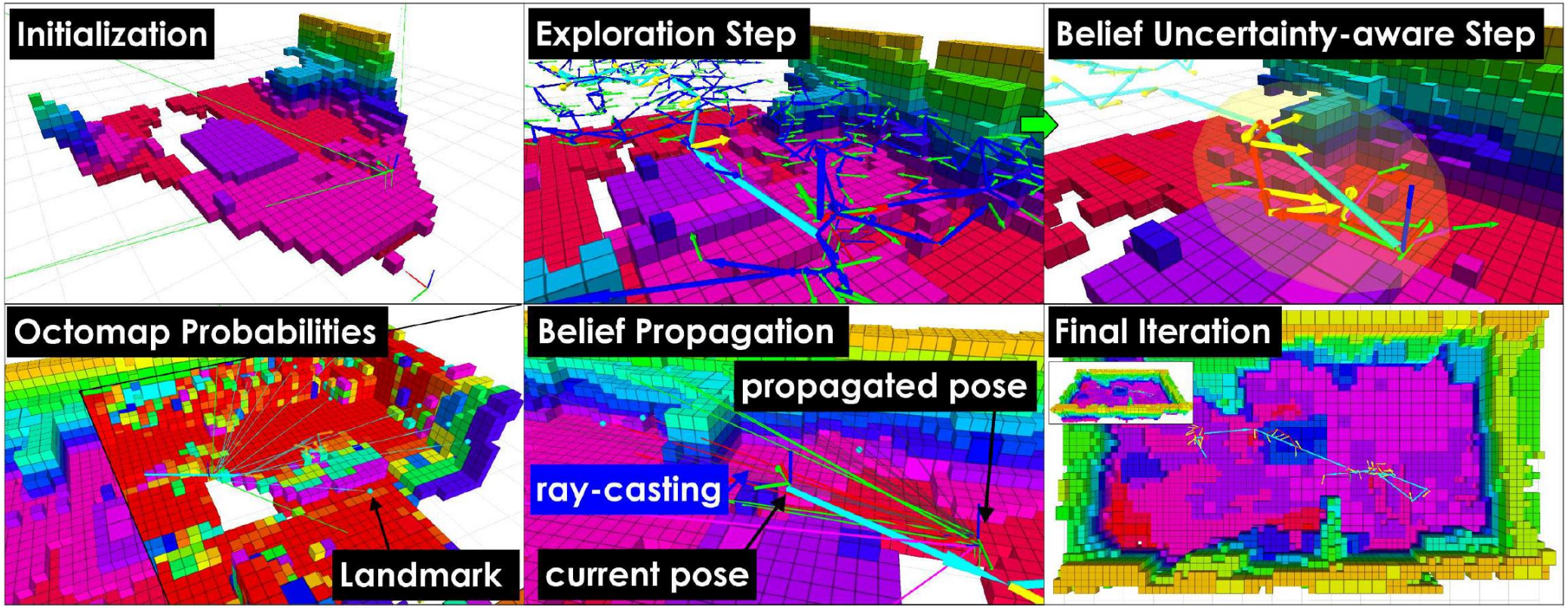
Uncertainty-aware Receding Horizon Exploration and Mapping using Aerial Robots

Christos Papachristos, Shehryar Khattak, Kostas Alexis



AUTONOMOUS ROBOTS LAB

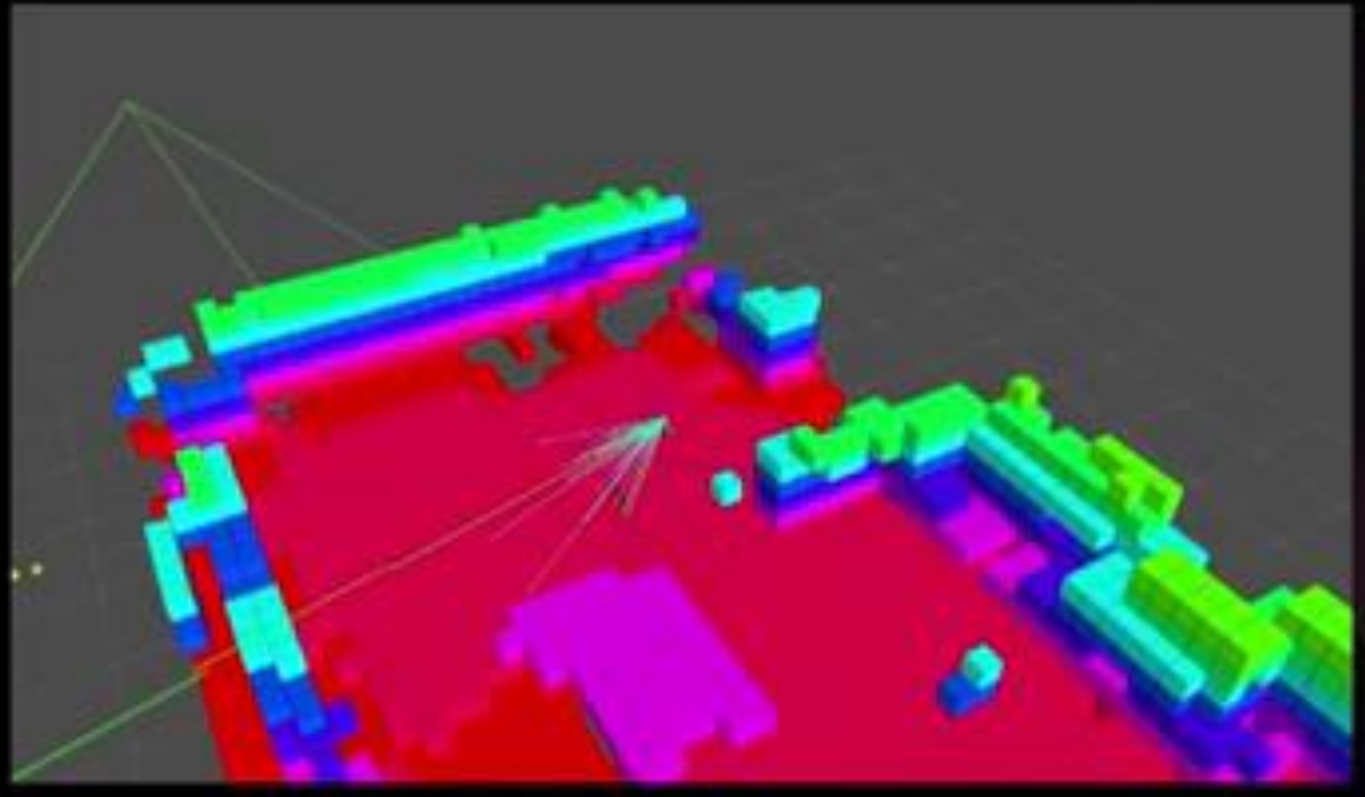
`rhemplanner` Evaluation (Experimental)



`rhemplanner` Evaluation (Experimental)

Autonomous Exploration in Visually-degraded Dark Environments using a NIR-IMU-Depth Sensor

C. Papachristos, S. Khattak, K. Alexis



AUTONOMOUS ROBOTS LAB

This material is based upon work supported by the Department of Energy under Award Number [DE-EM0004478]

BadgerWorks

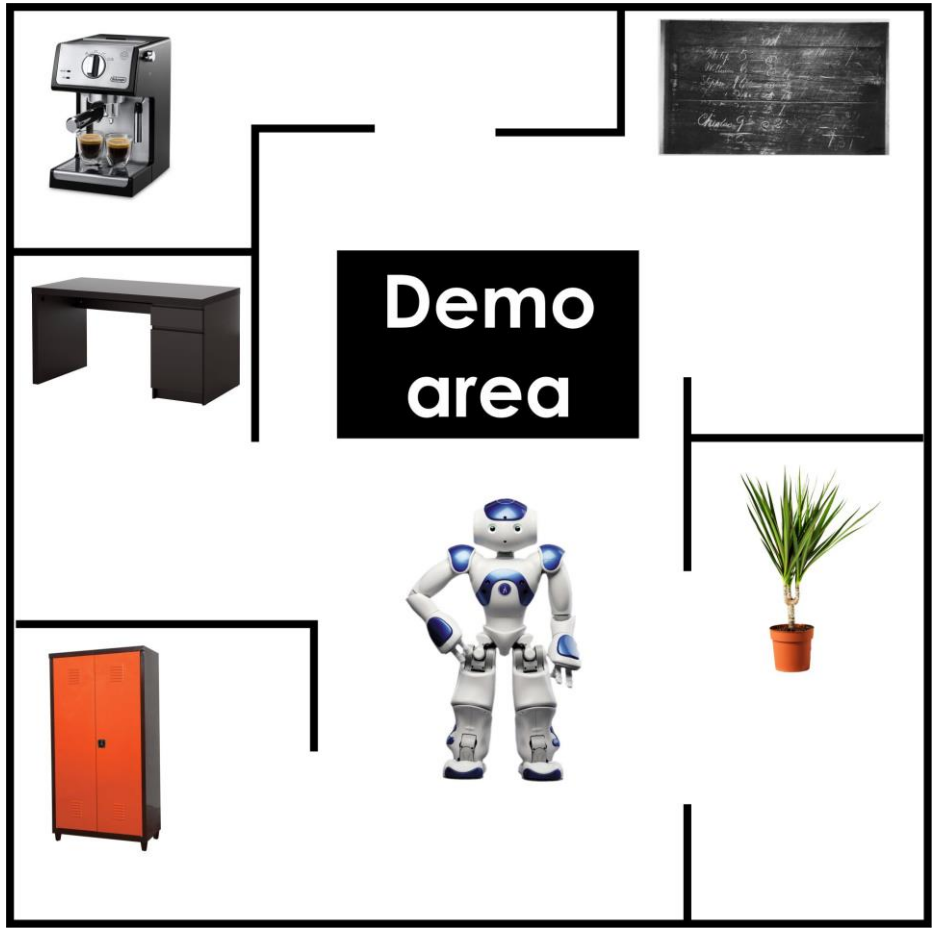
Primer Series – Integrated Task and Motion Planning

Kostas Alexis

AUTONOMOUS ROBOTS LAB

Move beyond "reach the goal" paradigm

- **Mission:** "Do the following in any order and always avoid the black demo area:
  - Water the plant.
  - Clean the blackboard.
  - Turn off the coffee maker.
  - Pick up vacuum cleaner from the supply room, then vacuum the orange room and dust its table"



Move beyond "reach the goal" paradigm

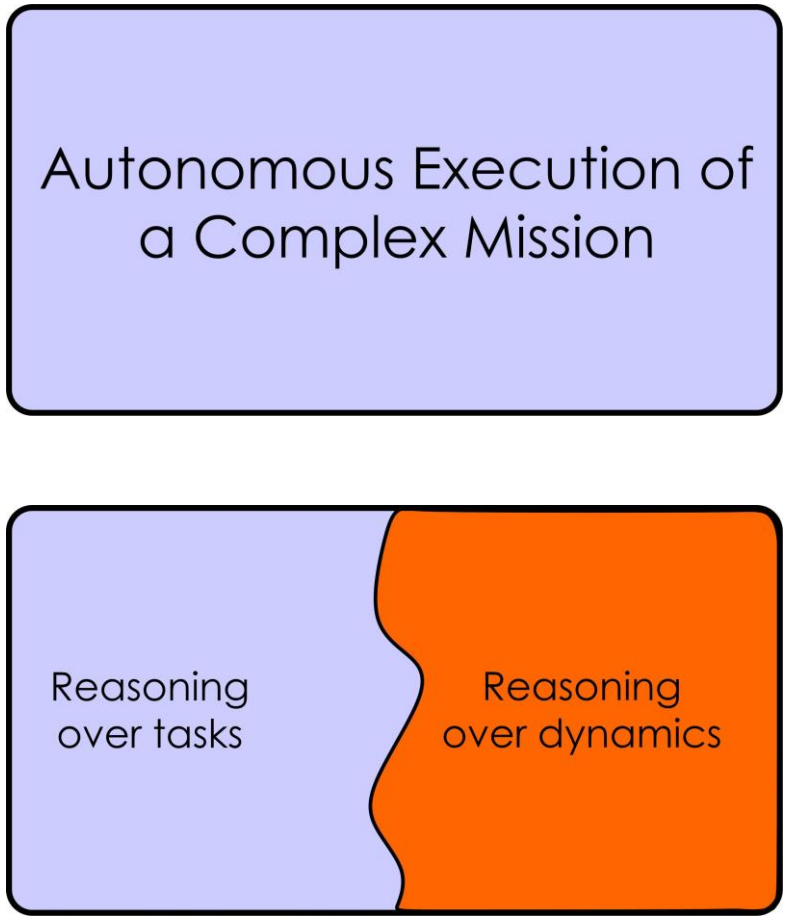
How can I plan to execute a complex mission

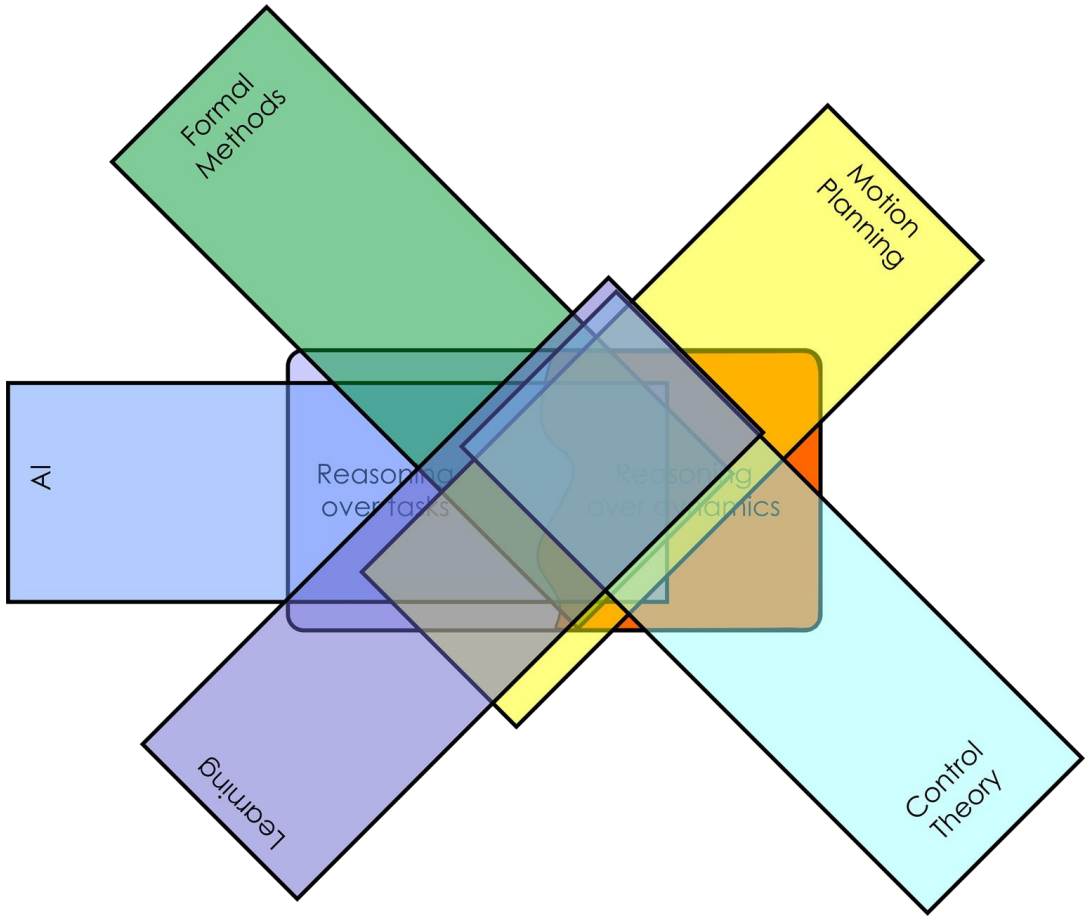
Move beyond "reach the goal" paradigm

Reason over tasks

Reason over dynamics

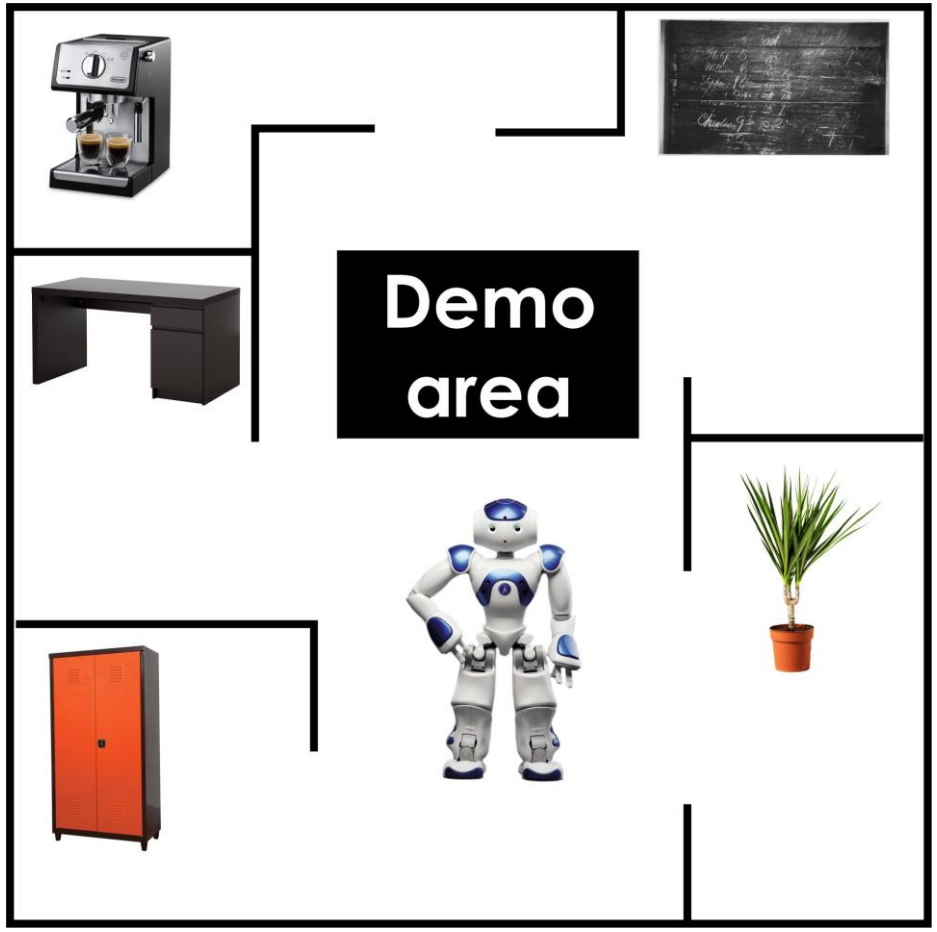
Move beyond "reach the goal" paradigm



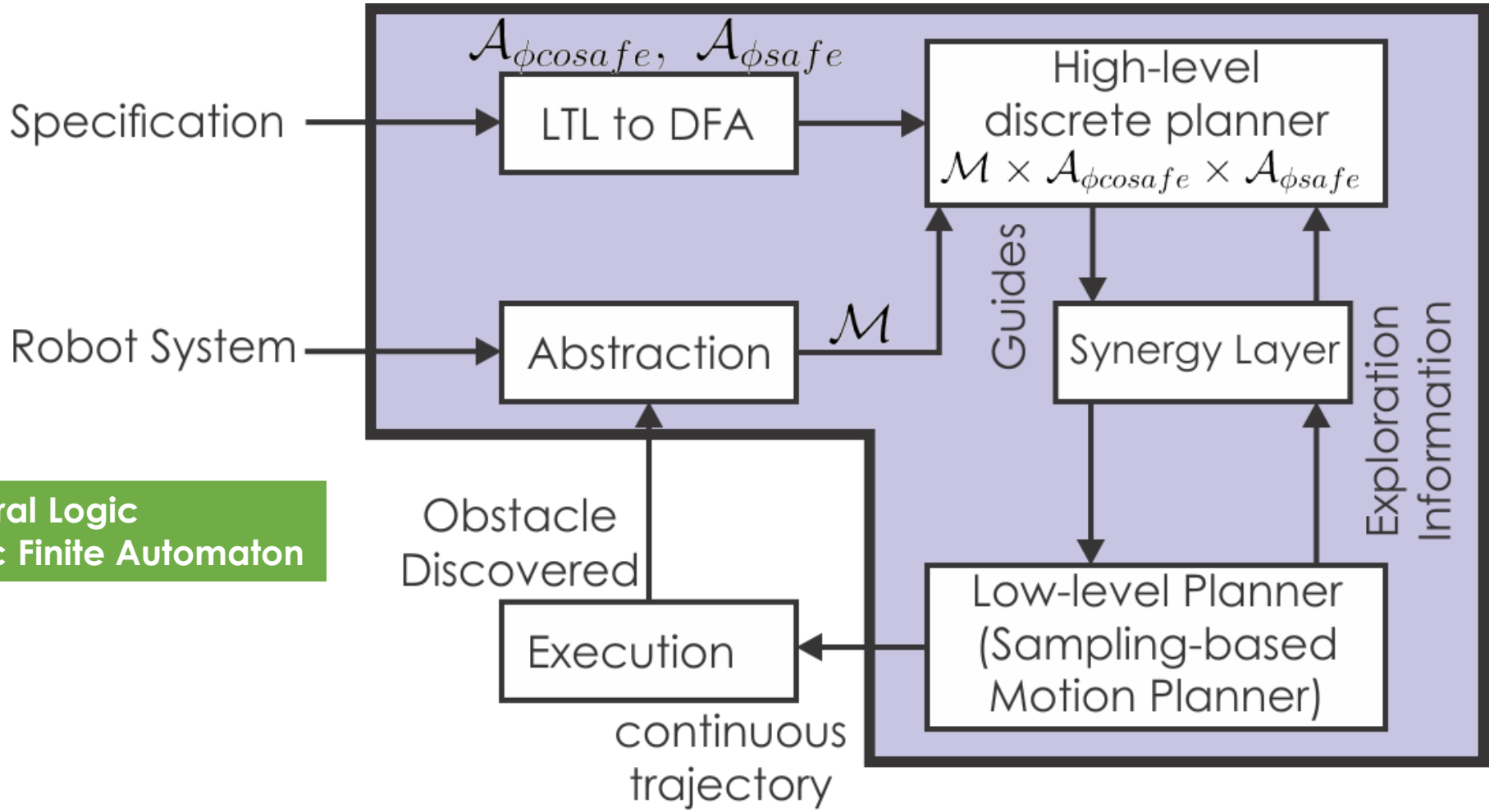


Move beyond "reach the goal" paradigm

- Ability to plan for general complex robotic systems using sampling-based motion planners.

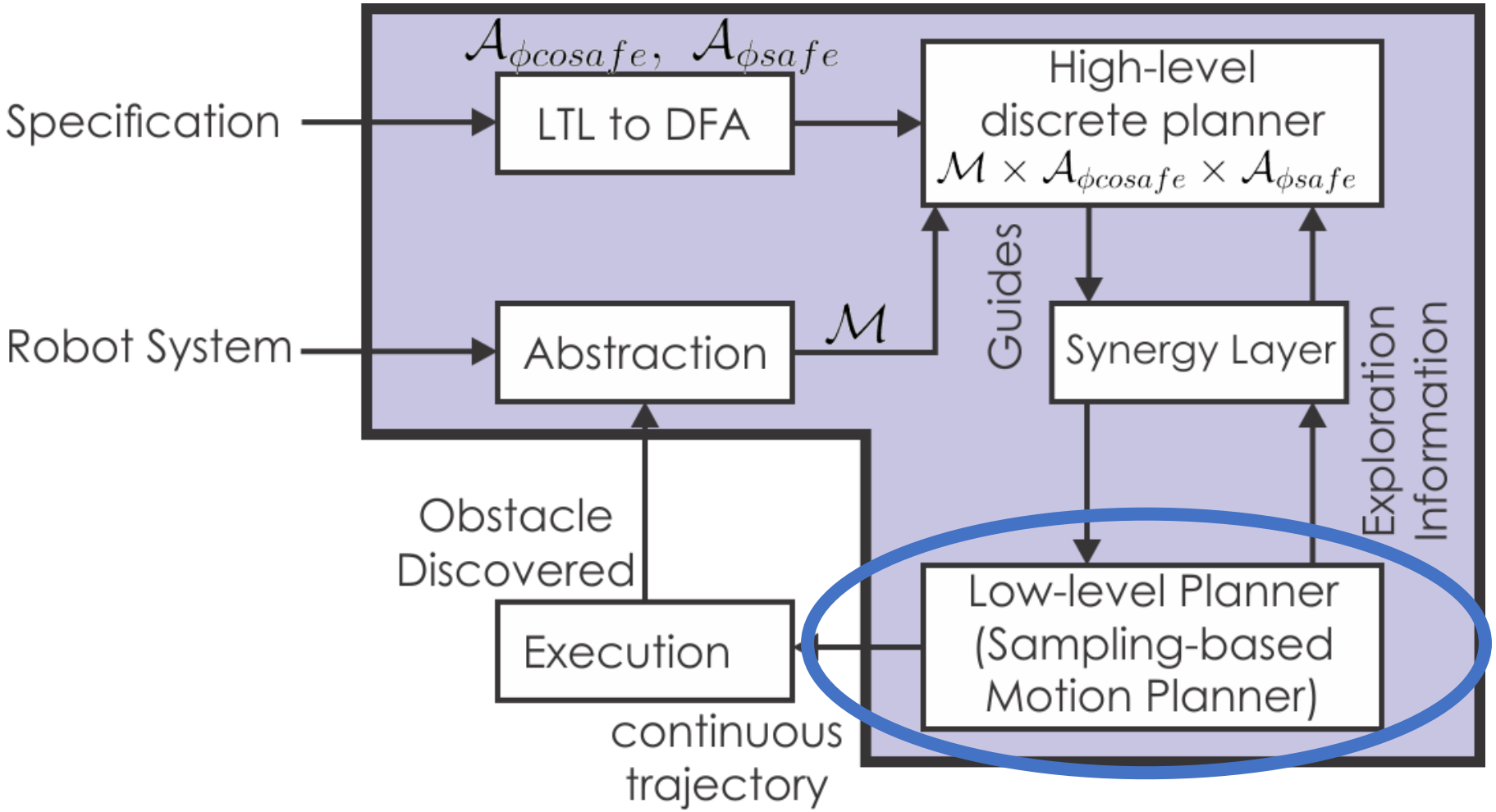


Solution employing Linear Temporal Logic

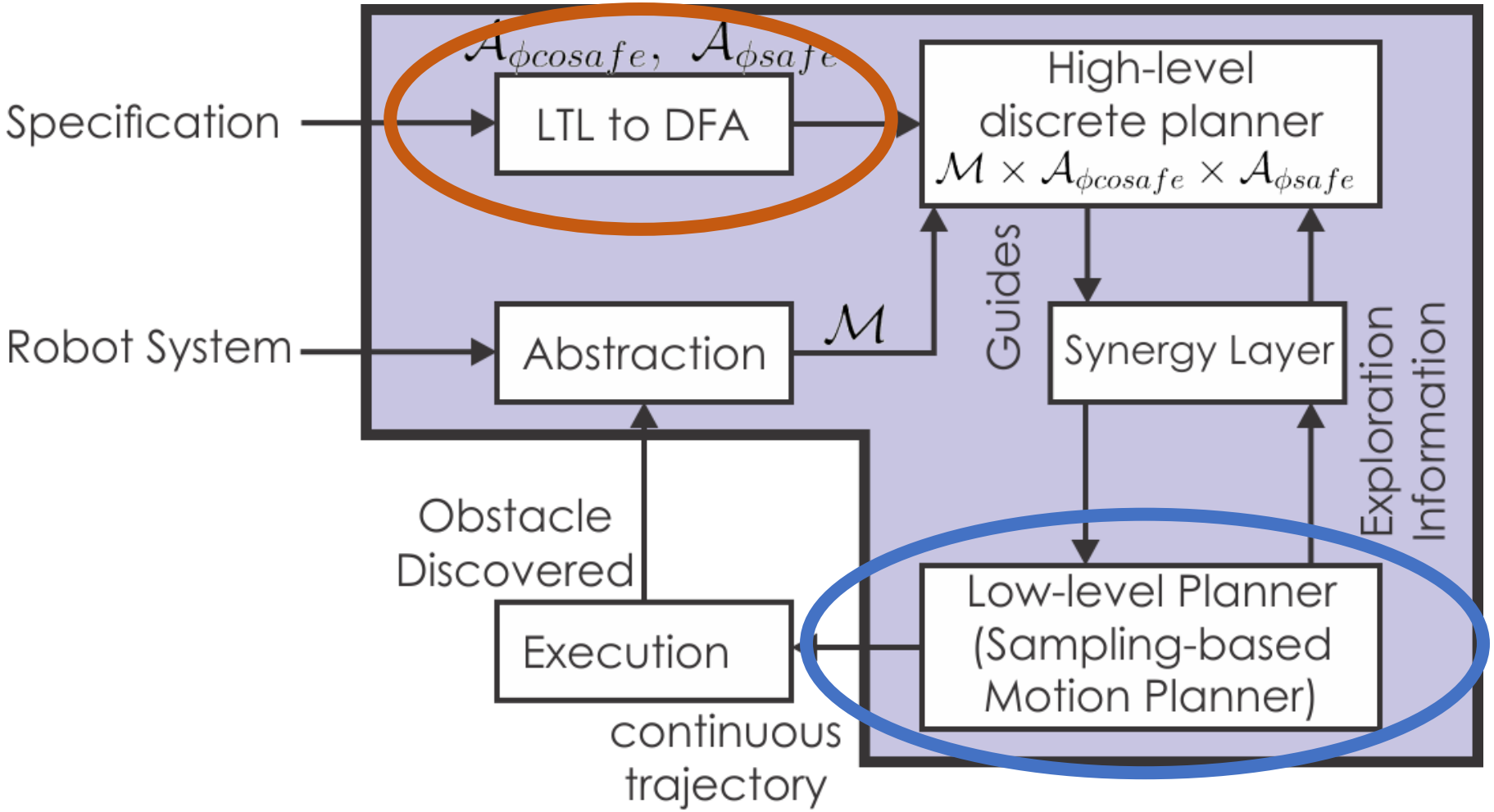


LTL: Linear Temporal Logic
DFA: Deterministic Finite Automaton

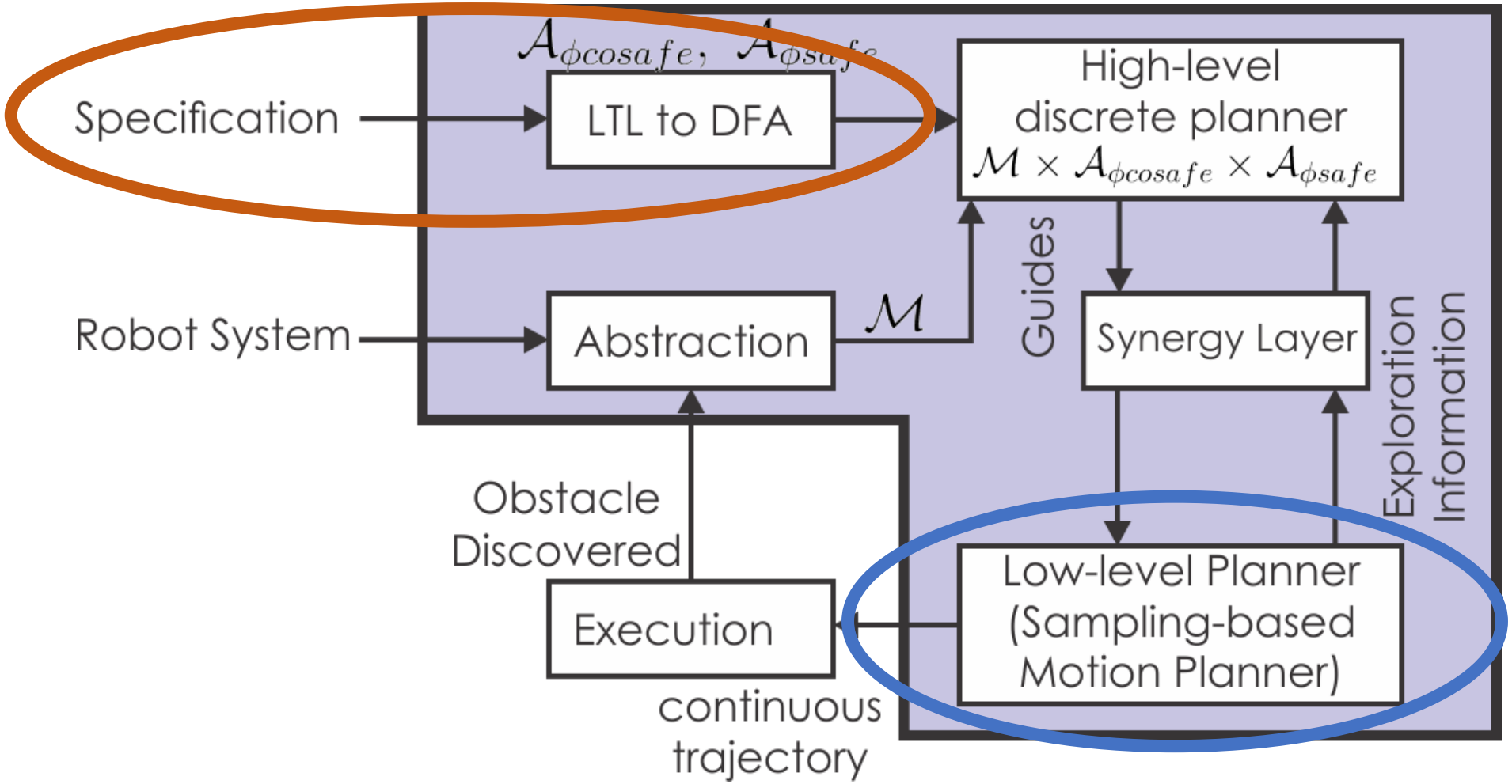
Solution employing Linear Temporal Logic



Solution employing Linear Temporal Logic



Solution employing Linear Temporal Logic



Linear Temporal Logic (LTL)

- Linear Temporal Logic (LTL) or linear-time temporal logic is a modal temporal logic with modalities referring to time.
  - **In LTL, one can encode formulae about the future of paths, e.g., a condition will eventually be true, a condition will be true until another fact becomes true, etc.**
- LTL was first proposed for the formal verification of computer programs by Amir Pnueli in 1977.
- **LTL is Built from:**
  - A set S of proposition variables: $S = \{p_0, p_1, \dots, p_N\}$
  - Boolean connectivities: And (&), Or (|), Not (¬)
  - Temporal connectivities: Next (X), Eventually (F), Always (G), Until (U)
- Traditionally used to successfully check properties of hardware and software.

Syntactically Co-Safe LTL

- **Co-safe LTL**

until

$$\phi := \pi \mid \neg\pi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \mathcal{X}\ \phi \mid \phi\ \mathcal{U}\ \phi \mid \mathcal{F}\ \phi$$

next

eventually

- Examples:

$$\phi = \mathcal{F}\ pickup$$
$$\phi = \mathcal{F}\ p_1 \wedge \mathcal{F}\ p_2 F\ p_3 \wedge \mathcal{F}\ p_4$$
$$\phi = \mathcal{F}\ (pickup \wedge \mathcal{X}(\neg pickup\ \mathcal{U}\ (dropoff\ f_1 \vee\ dropoff\ f_2))$$

Co-safe LTL – reduced subset of full LTL

Syntactically Co-Safe LTL

- **Co-safe LTL**

$$\phi := \pi \mid \neg\pi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \mathcal{X}\ \phi \mid \phi\ \mathcal{U}\ \phi \mid \mathcal{F}\ \phi$$

until

next

eventually

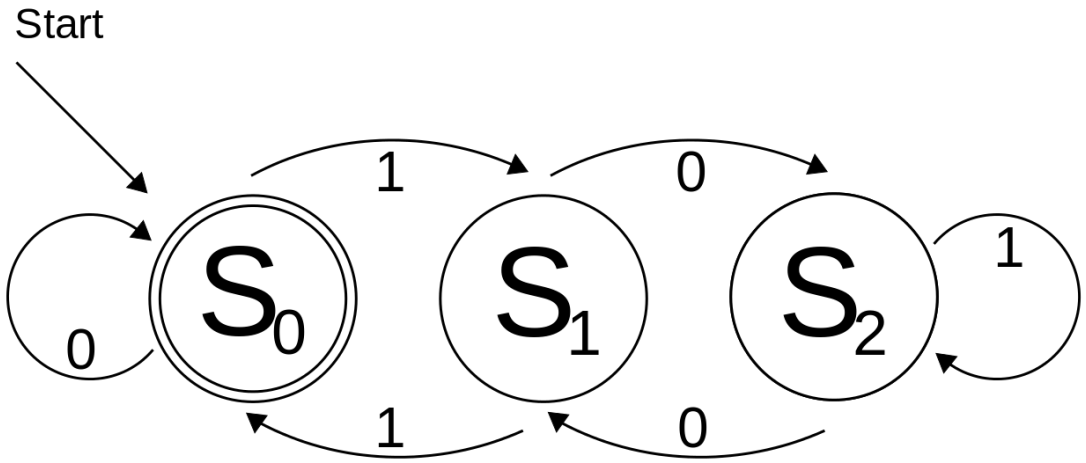
- Examples:

$$\phi = \mathcal{F}\ pickup$$
$$\phi = \mathcal{F}\ p_1 \wedge \mathcal{F}\ p_2 F\ p_3 \wedge \mathcal{F}\ p_4$$
$$\phi = \mathcal{F}\ (pickup \wedge \mathcal{X}(\neg pickup\ \mathcal{U}\ (dropoff\ f_1 \vee\ dropoff\ f_2))$$

- **DFA**: from a co-safe LTL φ, a Deterministic Finite Automaton A_φ can be constructed.

Deterministic Finite Automate (DFA)

- In the theory of computation, a Deterministic Finite Automaton (DFA)—also known as a deterministic finite accepter (DFA) and a deterministic finite state machine (DFSM)—is a finite-state machine that accepts and rejects strings of symbols and only produces a *unique computation* (or run) of the automaton for each input string.



- Upon reading a symbol, a DFA jumps deterministically from one state to another by following the transition arrow/transition prescribed.
- A DFA is defined as an abstract mathematical concept, but is often implemented in hardware and software for solving various specific problems. For example, a DFA can model software that decides whether or not online user input such as email addresses are valid.

Planning from LTL specifications

- Co-safe LTL

until

$$\phi := \pi \mid \neg\pi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \mathcal{X}\ \phi \mid \phi\ \mathcal{U}\ \phi \mid \mathcal{F}\ \phi$$

next

eventually

- Examples:

$$\phi = \mathcal{F}\ pickup$$
$$\phi = \mathcal{F}\ p_1 \wedge \mathcal{F}\ p_2 F\ p_3 \wedge \mathcal{F}\ p_4$$
$$\phi = \mathcal{F}\ (pickup \wedge \mathcal{X}(\neg pickup\ \mathcal{U}\ (dropoff\ f_1 \vee\ dropoff\ f_2))$$

- **DFA**: from a co-safe LTL φ, a Deterministic Finite Automaton A_φ can be constructed.

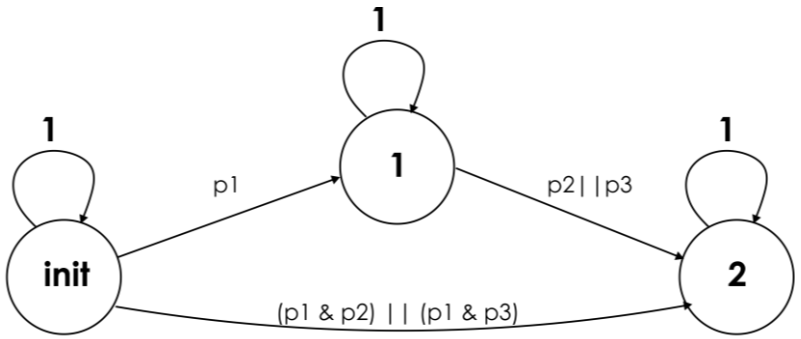
Planning from LTL specifications

- **In the future** visit p1, and then visit region p2 or p3.

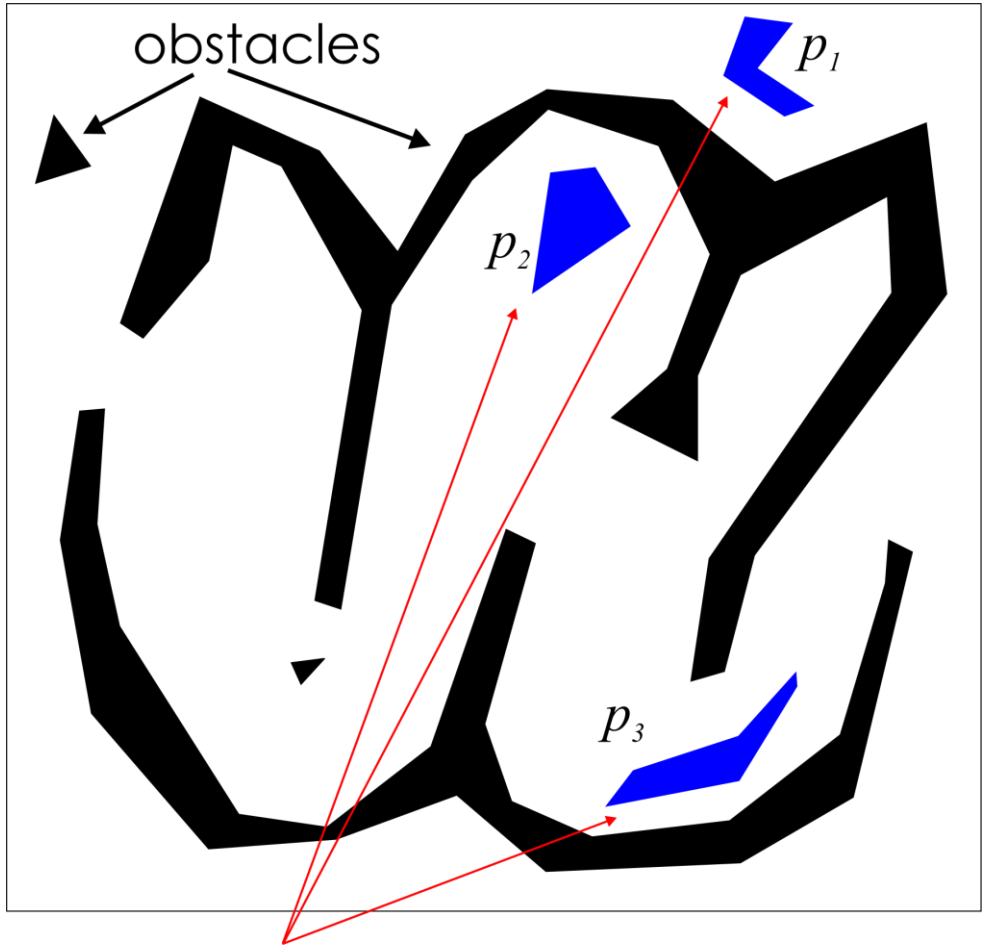
LTL

$$\varphi = F\ (p_1 \textbf{ AND } F\ (p_2 \textbf{ OR } p_3))$$

Automata Conversion

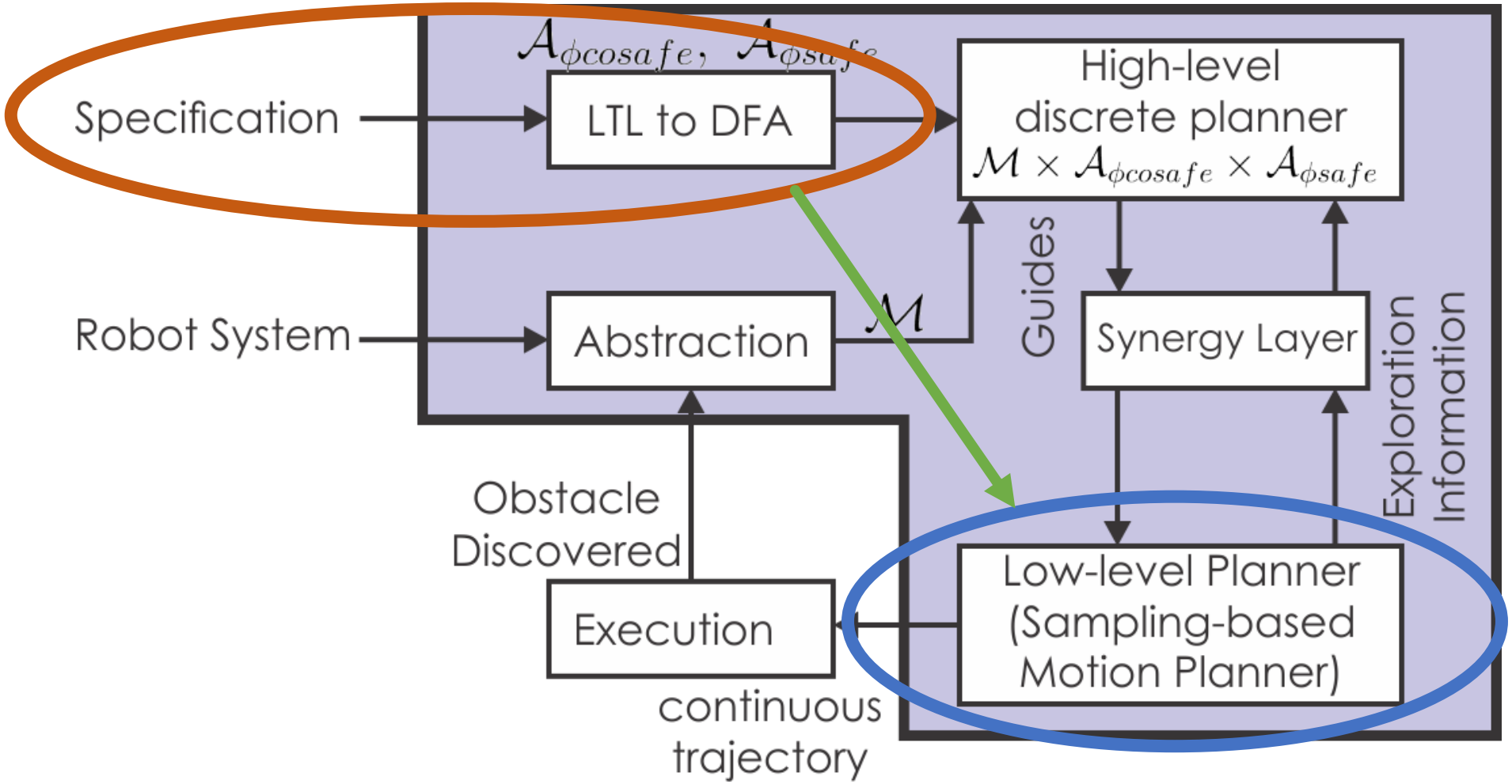


Solution: Plan, use Automation A_φ that encodes formula φ as a monitor

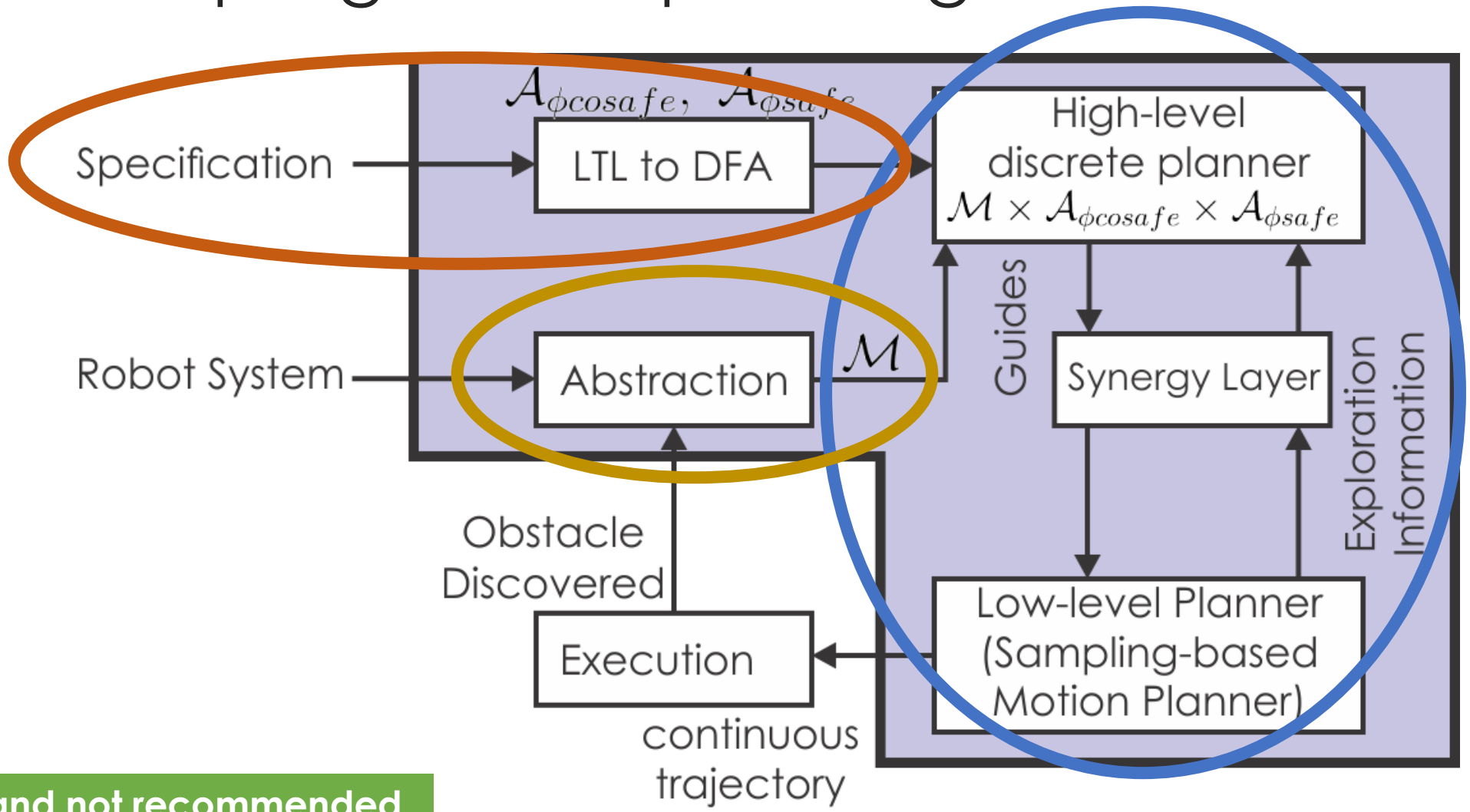


Regions of Interest (propositions)

Sampling-based planning alone is slow

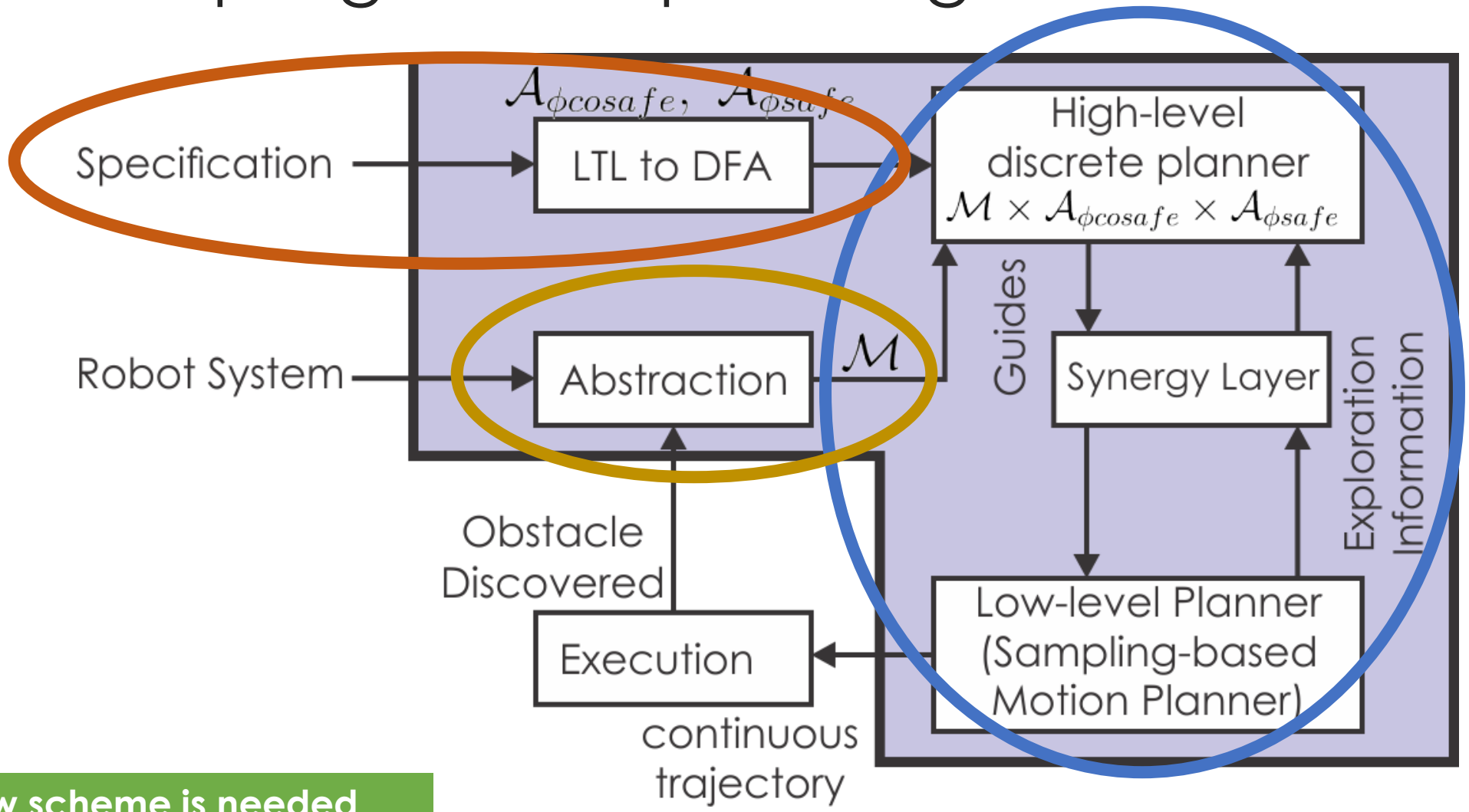


Sampling-based planning alone is slow



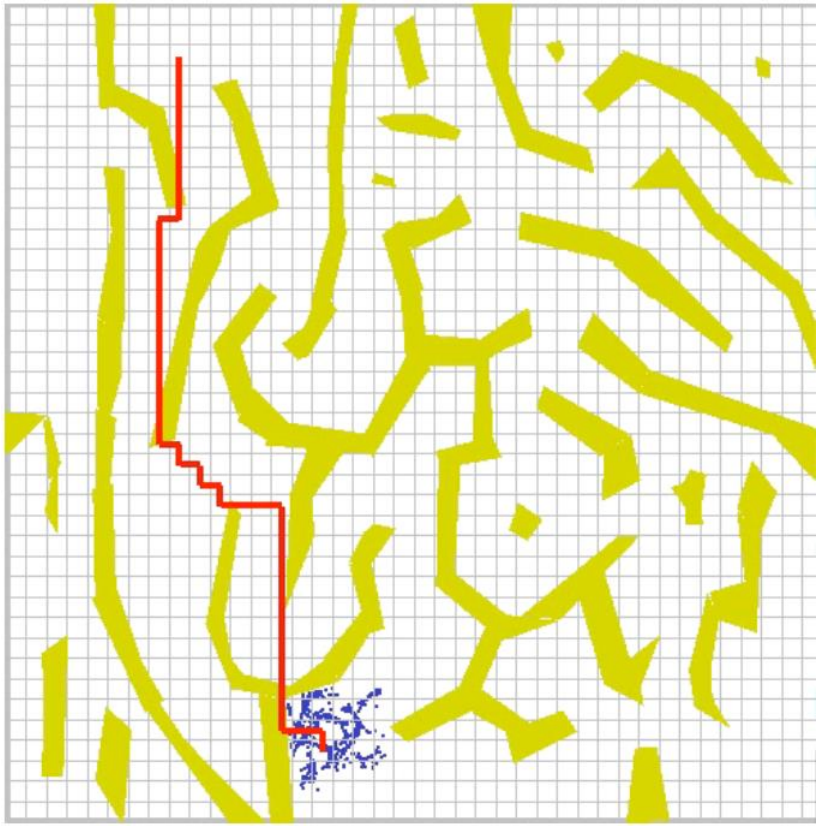
Slow and not recommended

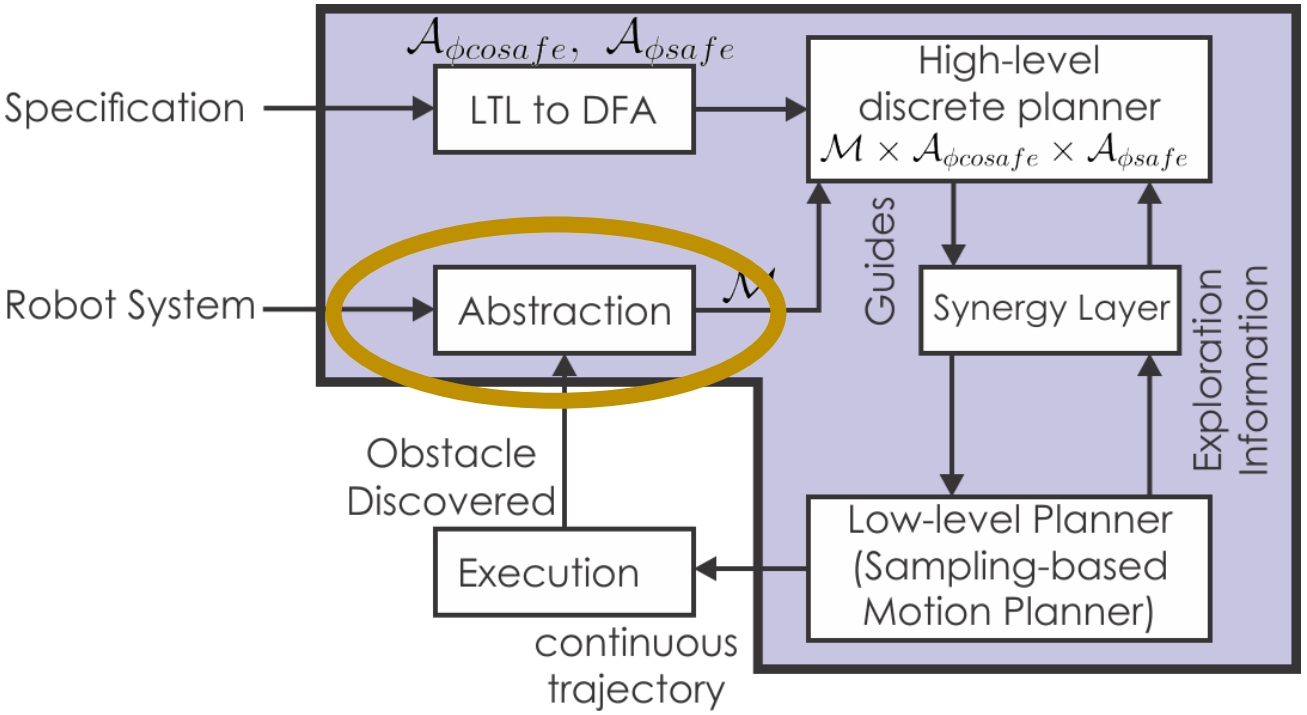
Sampling-based planning alone is slow



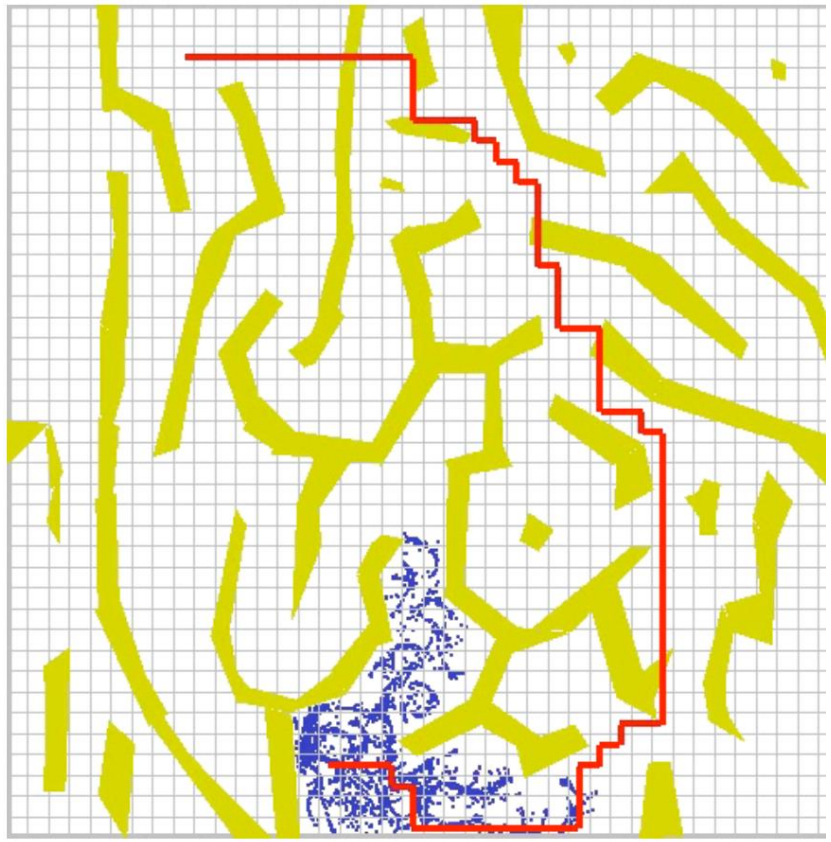
New scheme is needed

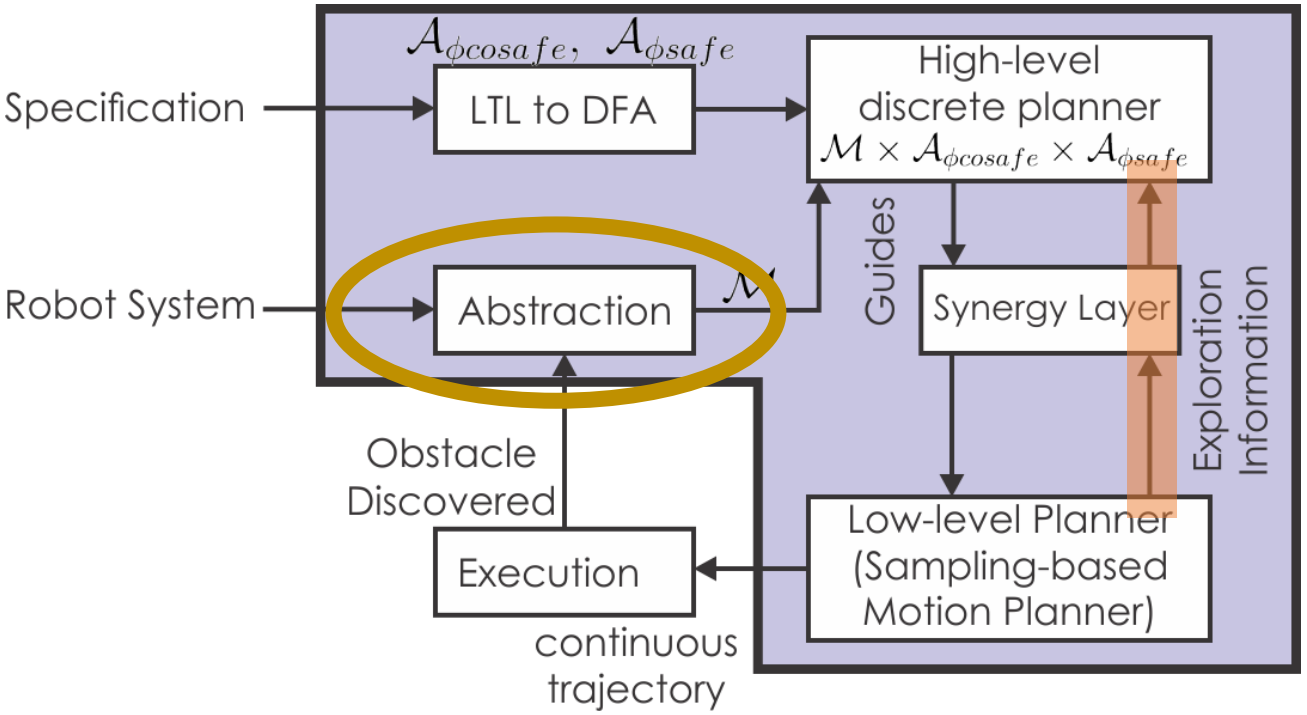
Sampling-based planning alone is slow





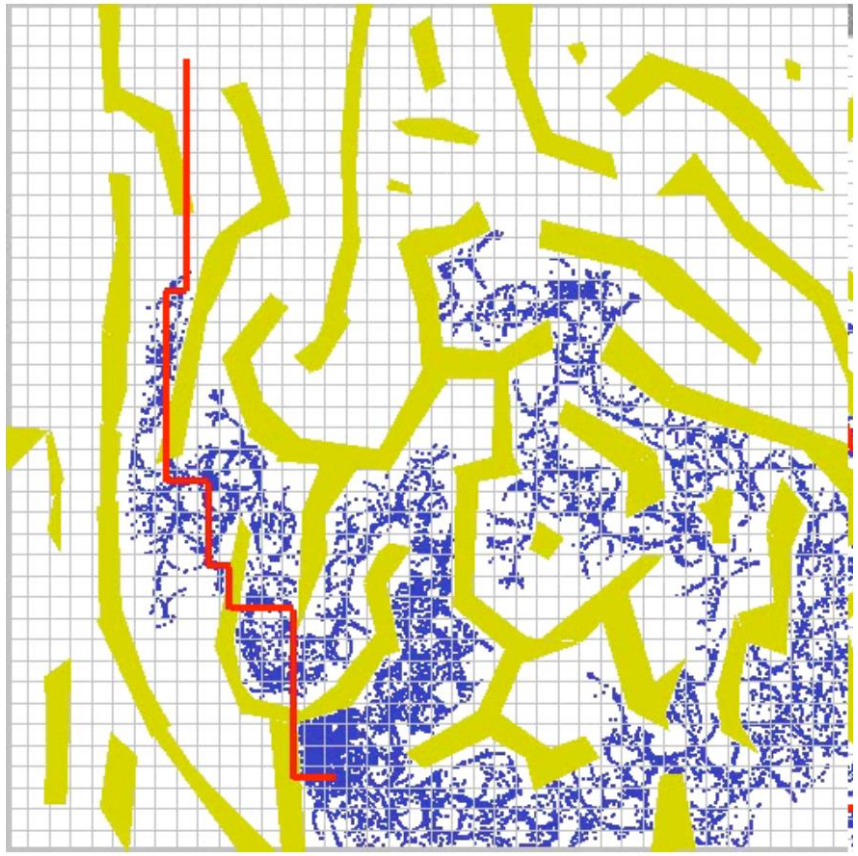
SyCLoP – A synergistic framework

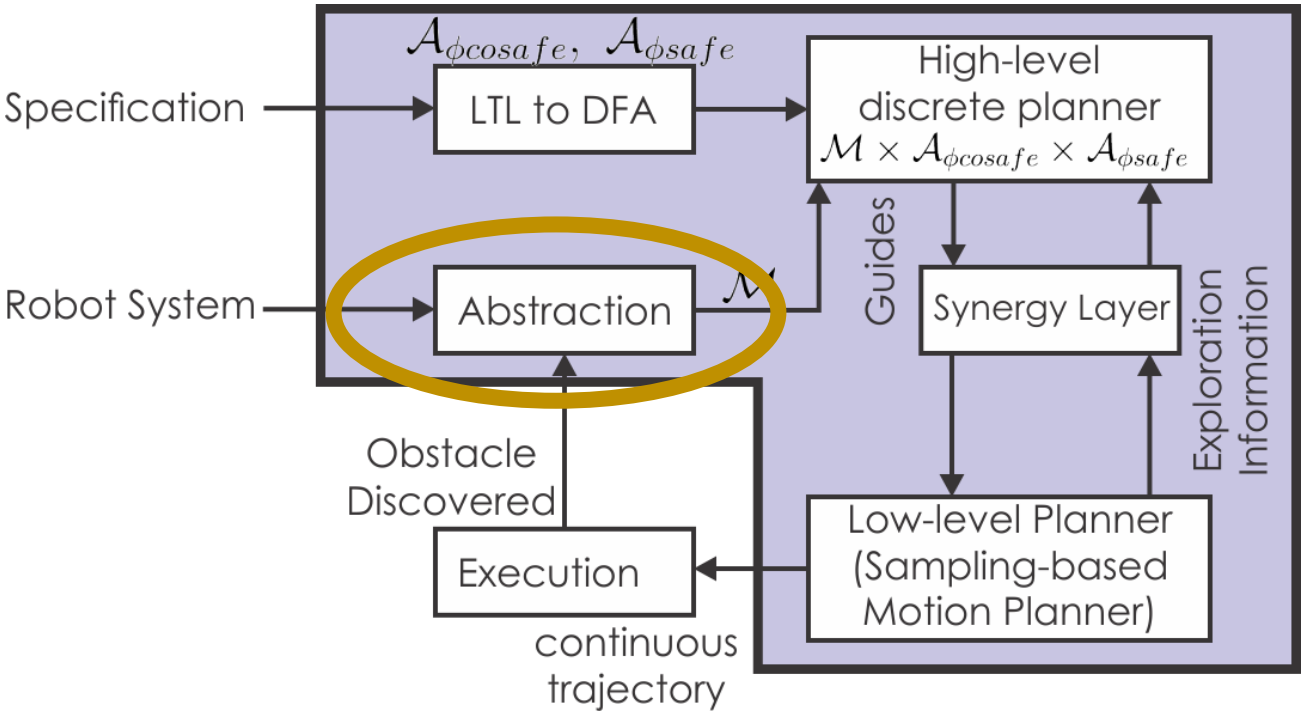




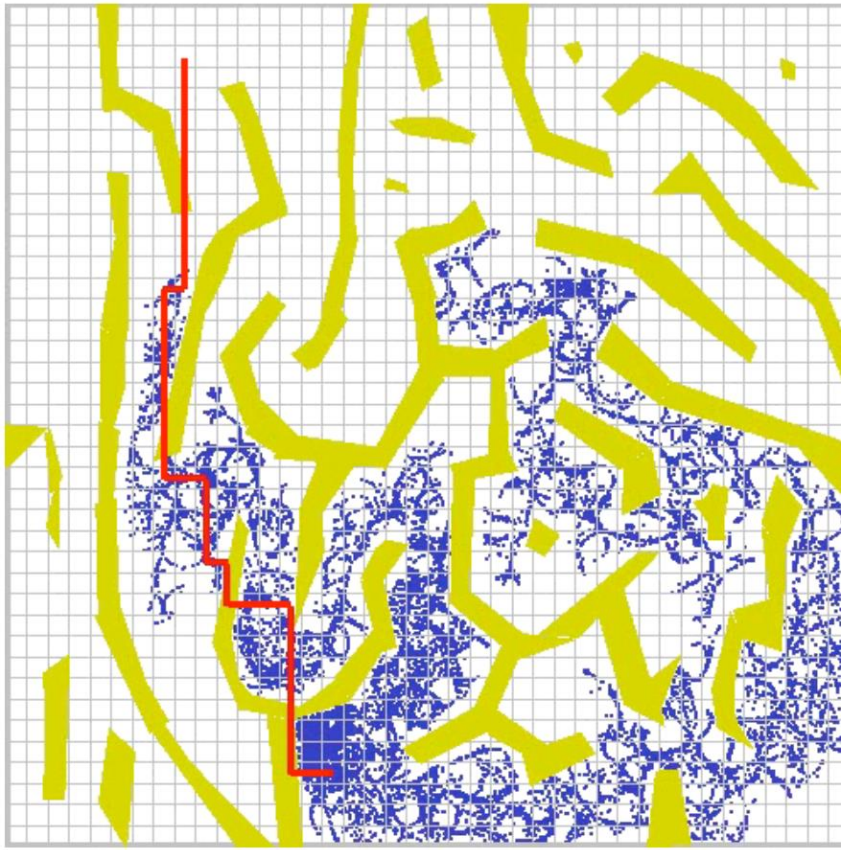
SyCLoP: Synergistic combination of layers of planning

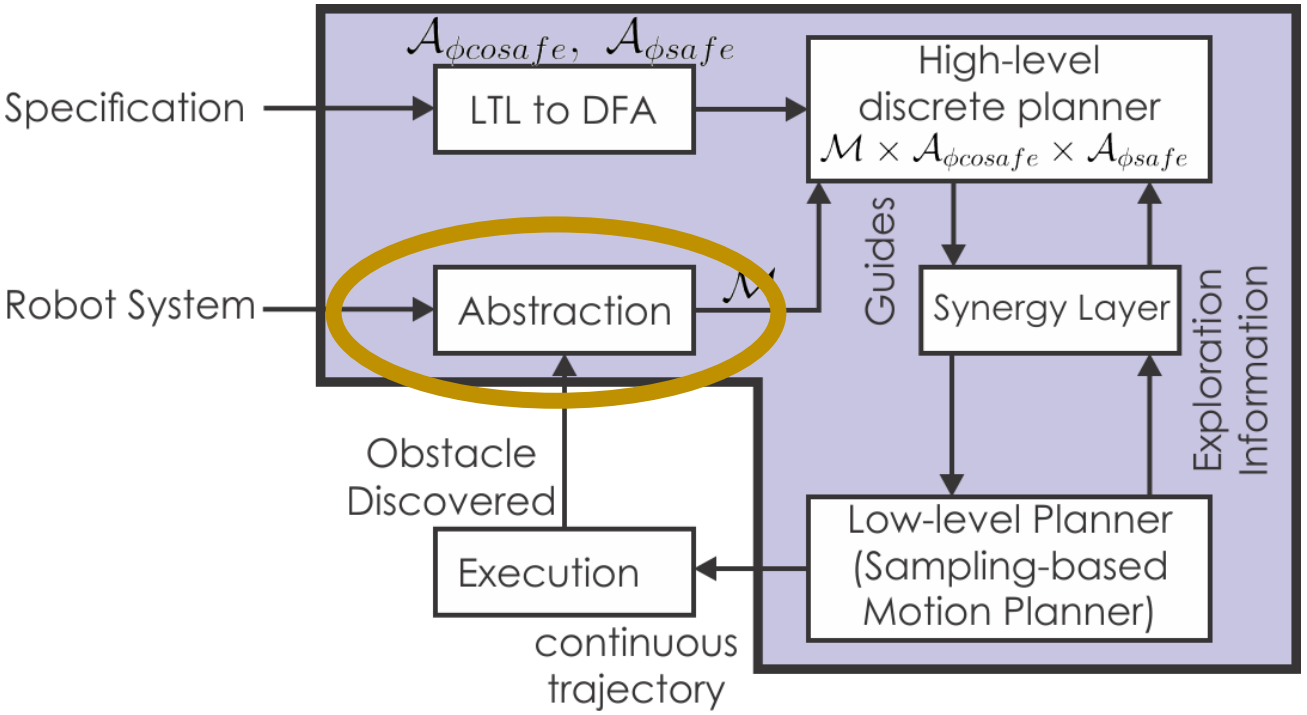
SyCLoP – A synergistic framework



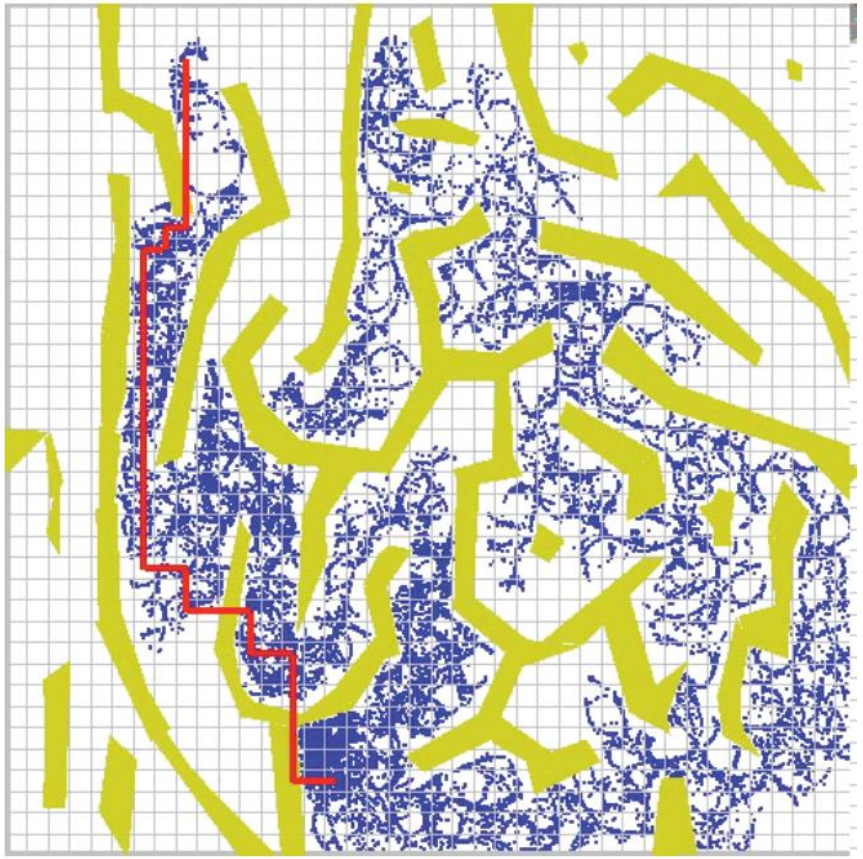


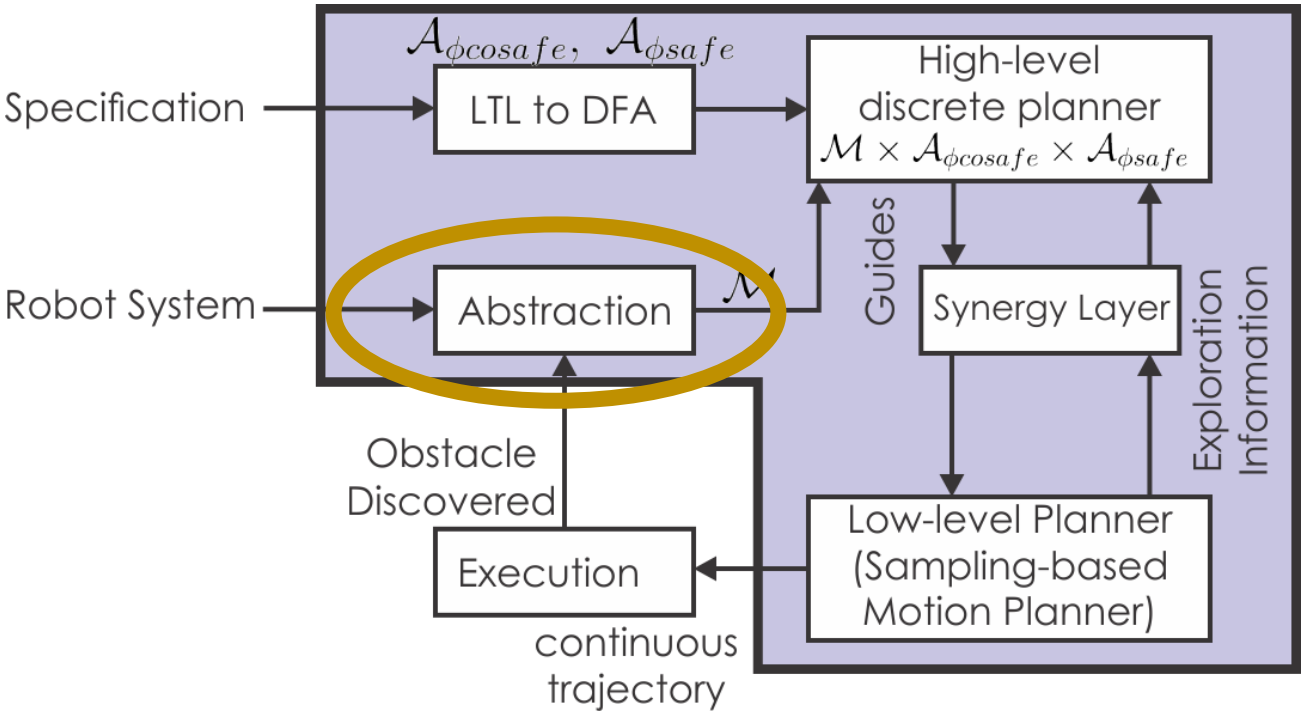
SyCLoP – A synergistic framework



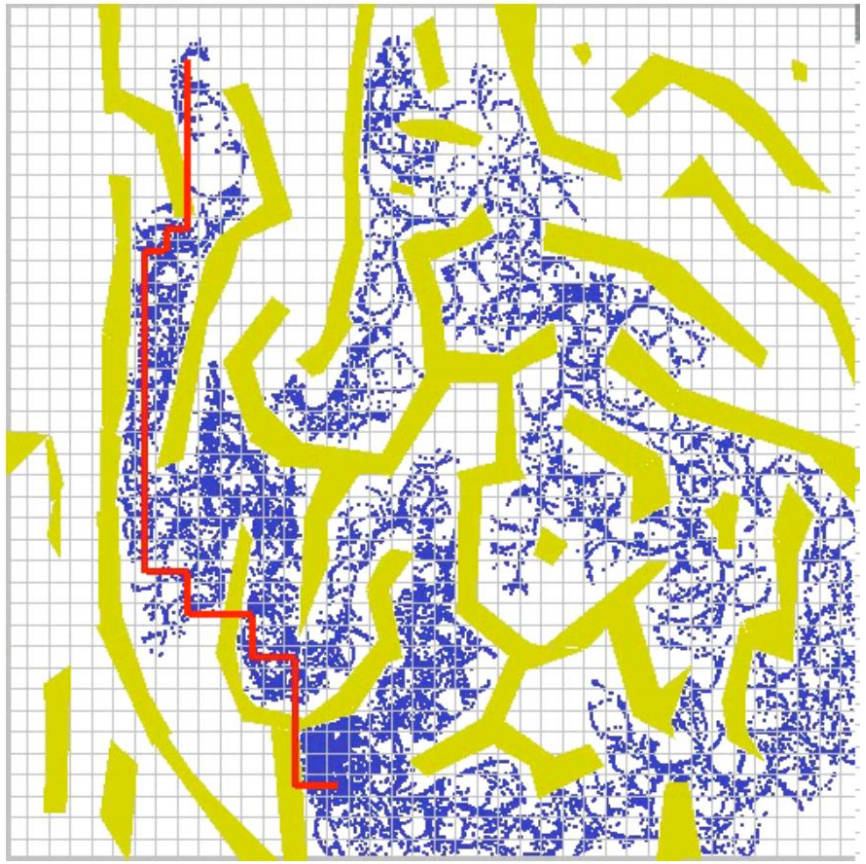


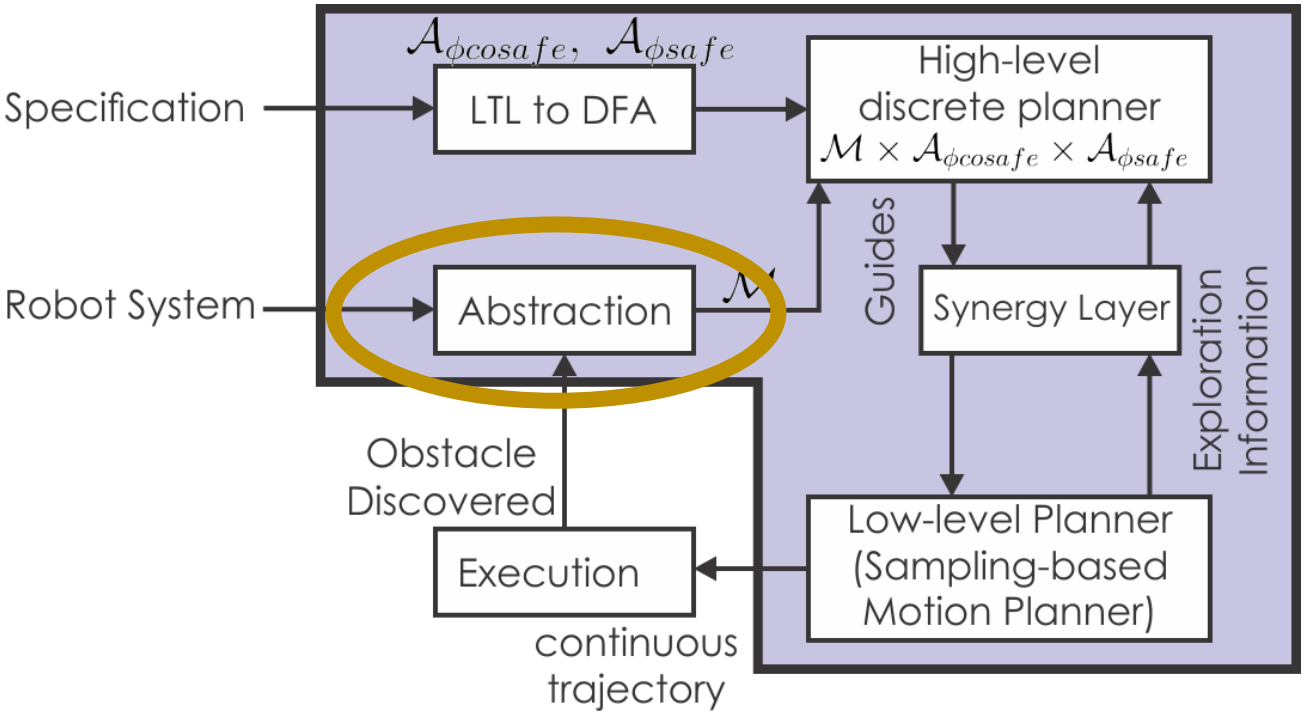
SyCLoP – A synergistic framework



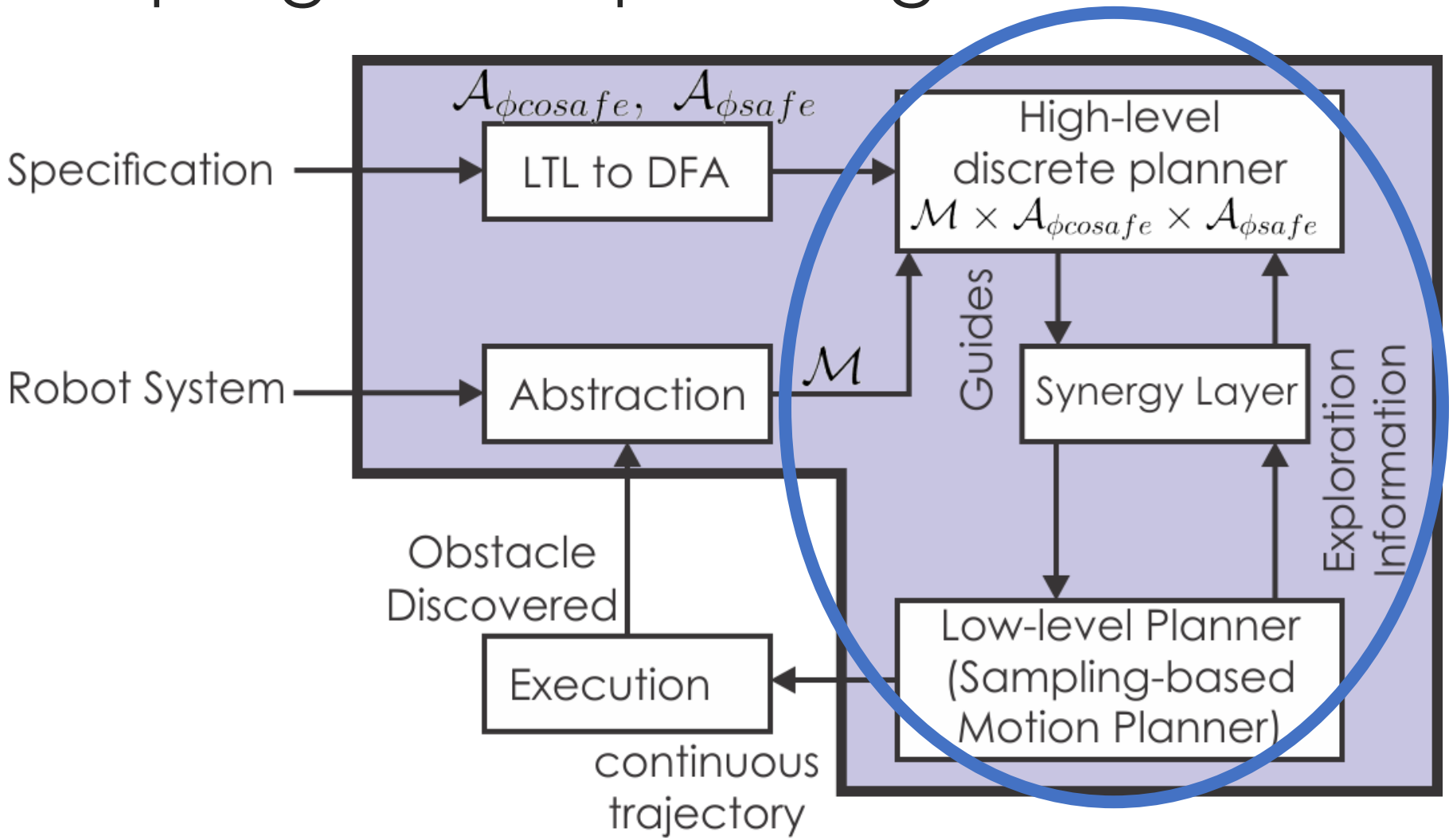


SyCLoP – A synergistic framework

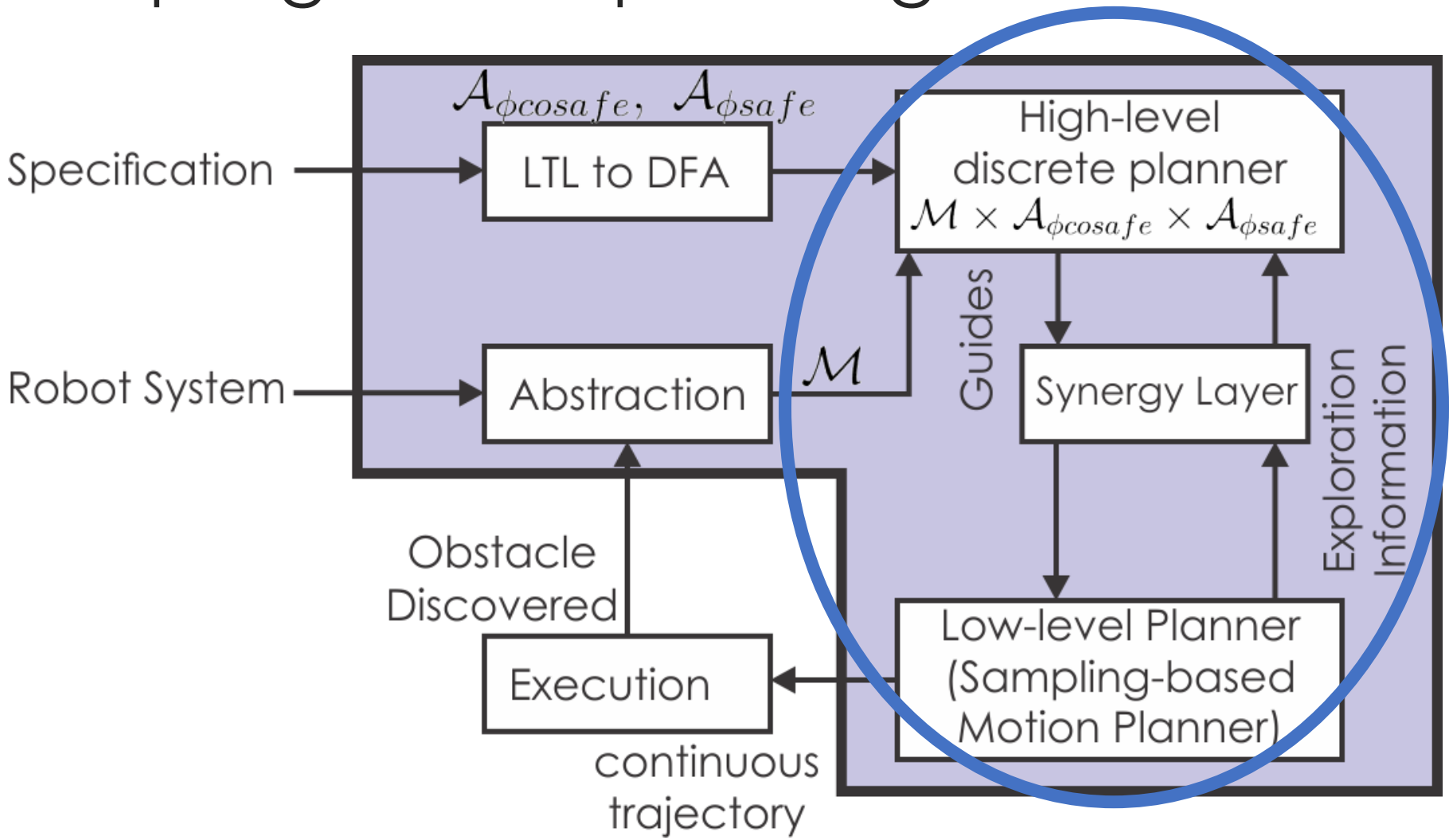




Sampling-based planning alone is slow

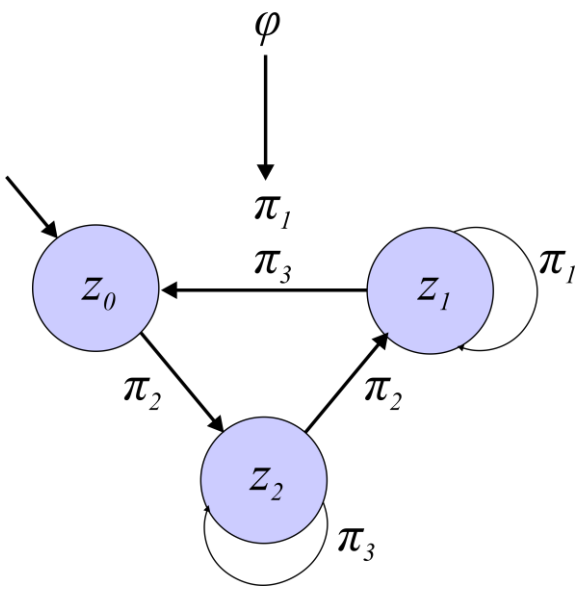


Sampling-based planning alone is slow

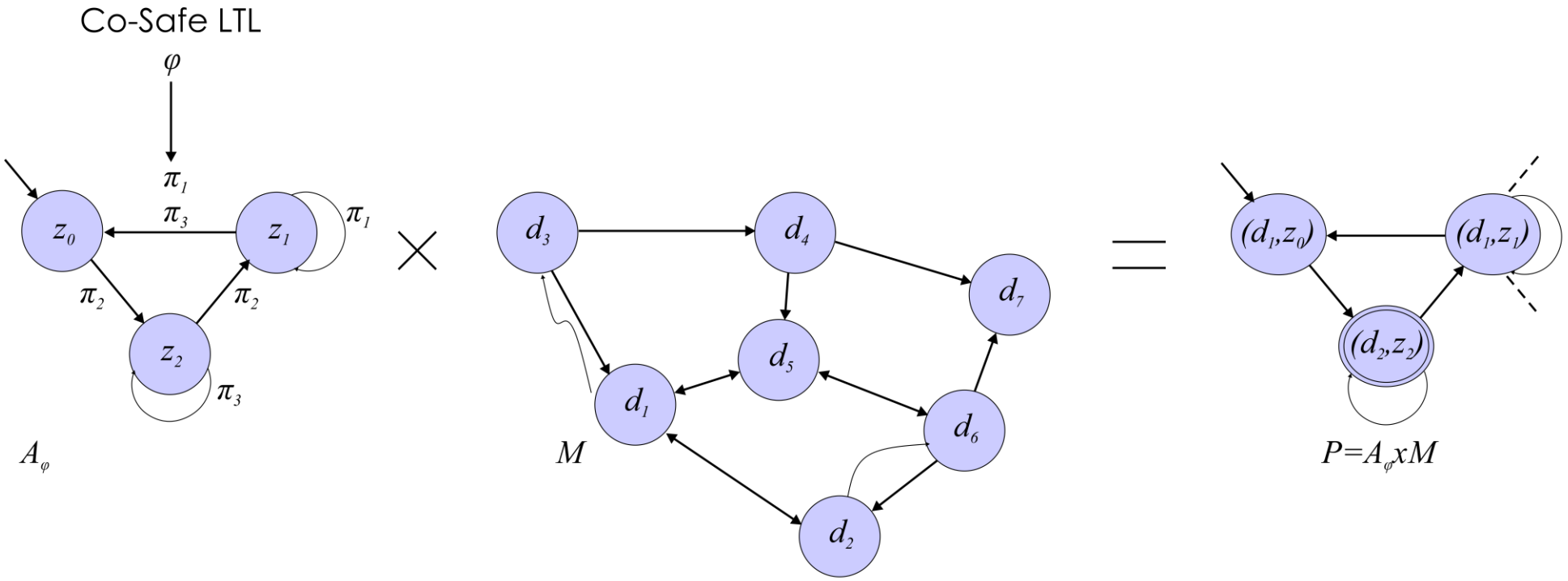


Synergistic framework

Co-Safe LTL



A guiding decomposition is needed

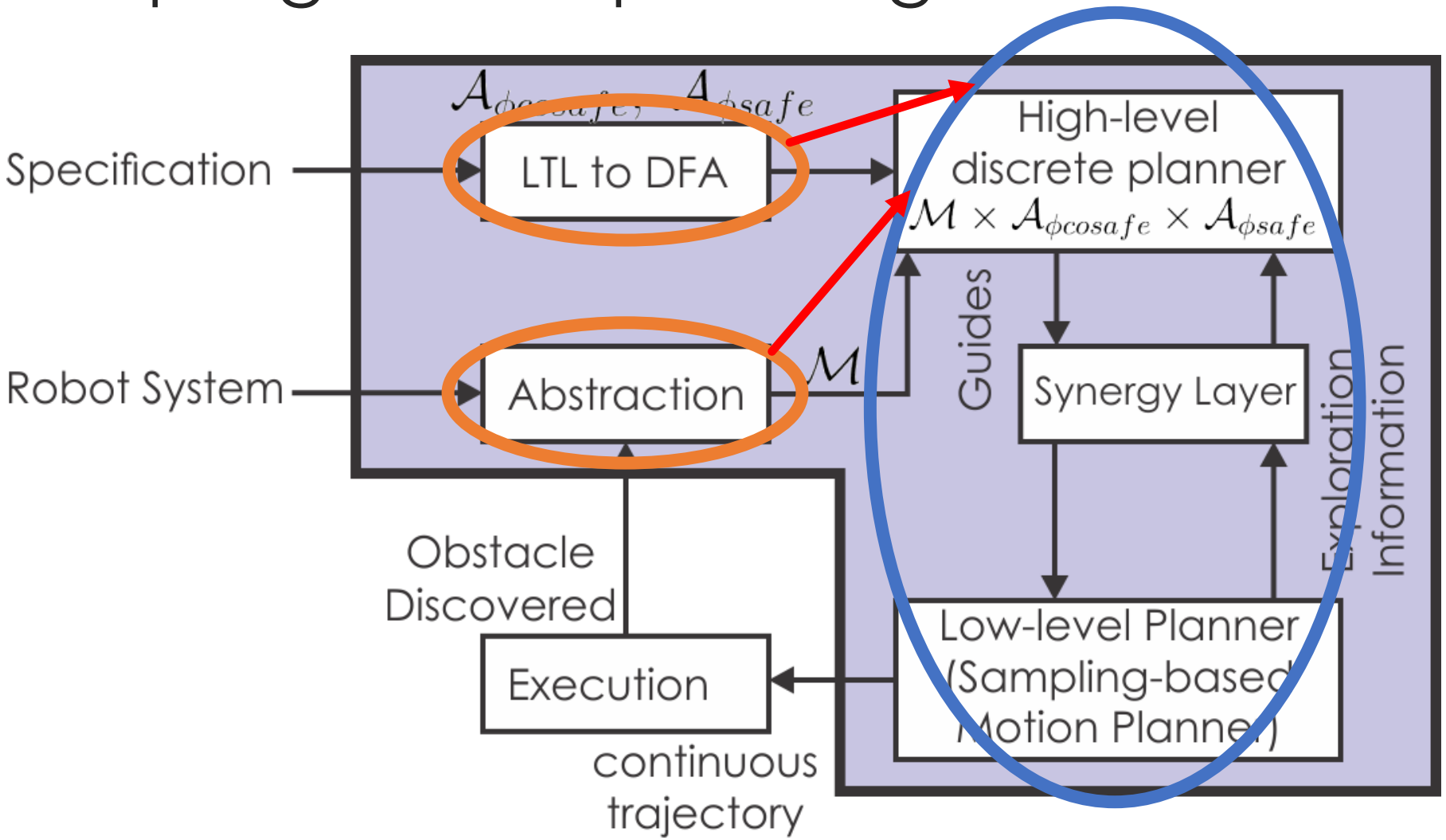


Automaton abstraction for co-safe LTL

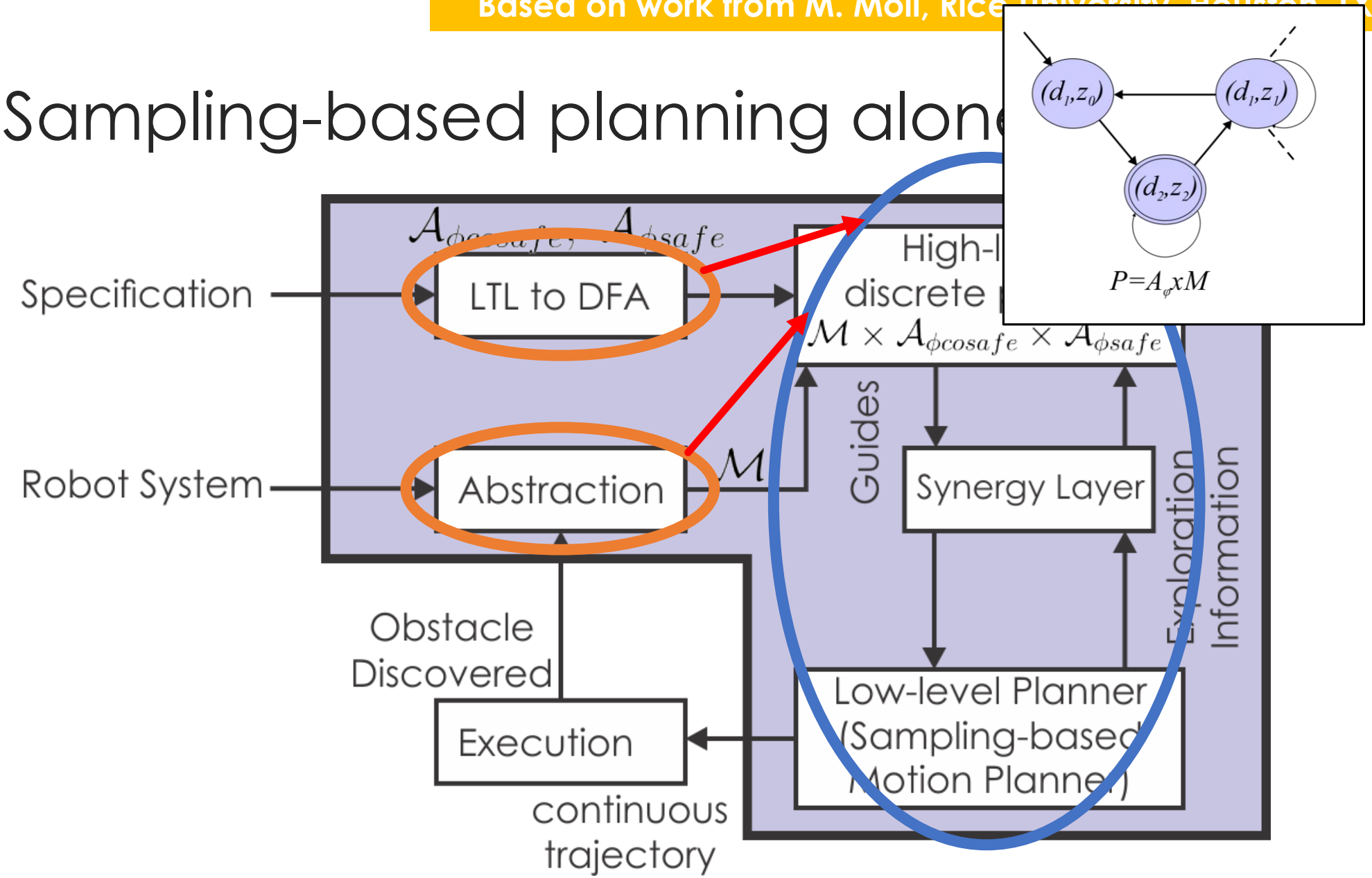
Abstraction for the workspace and robot

Overall discrete abstraction/ decomposition

Sampling-based planning alone is slow

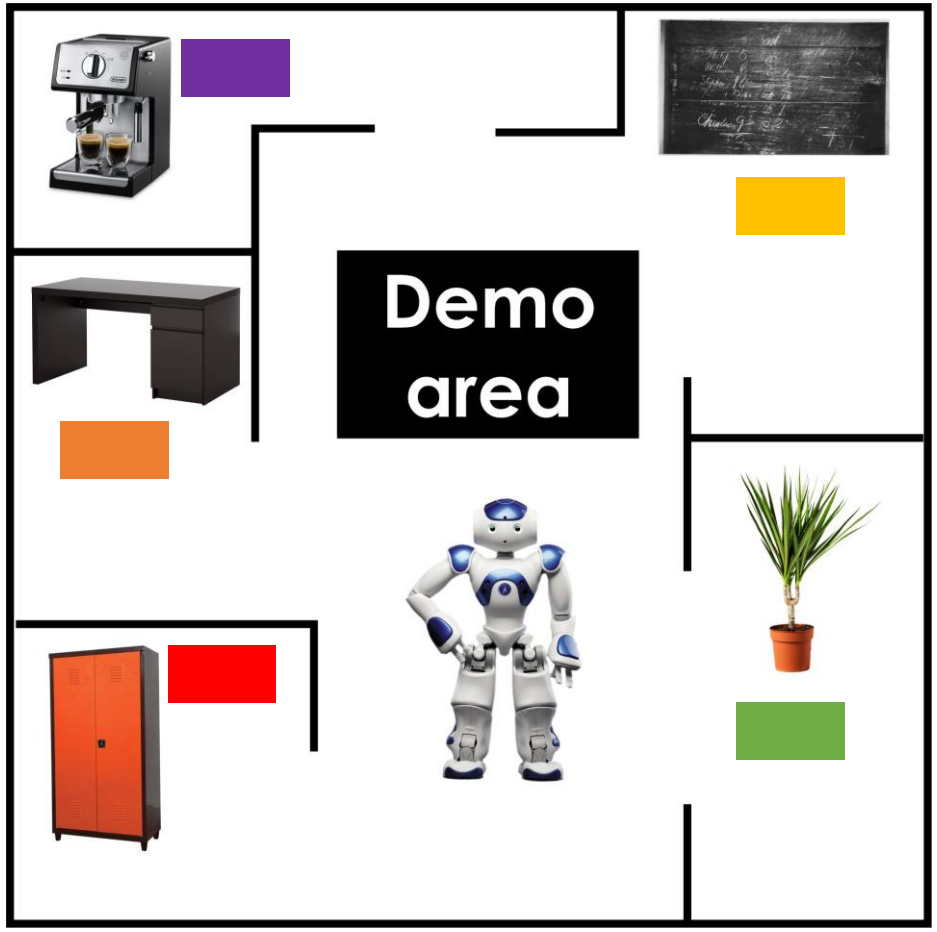


Sampling-based planning alone



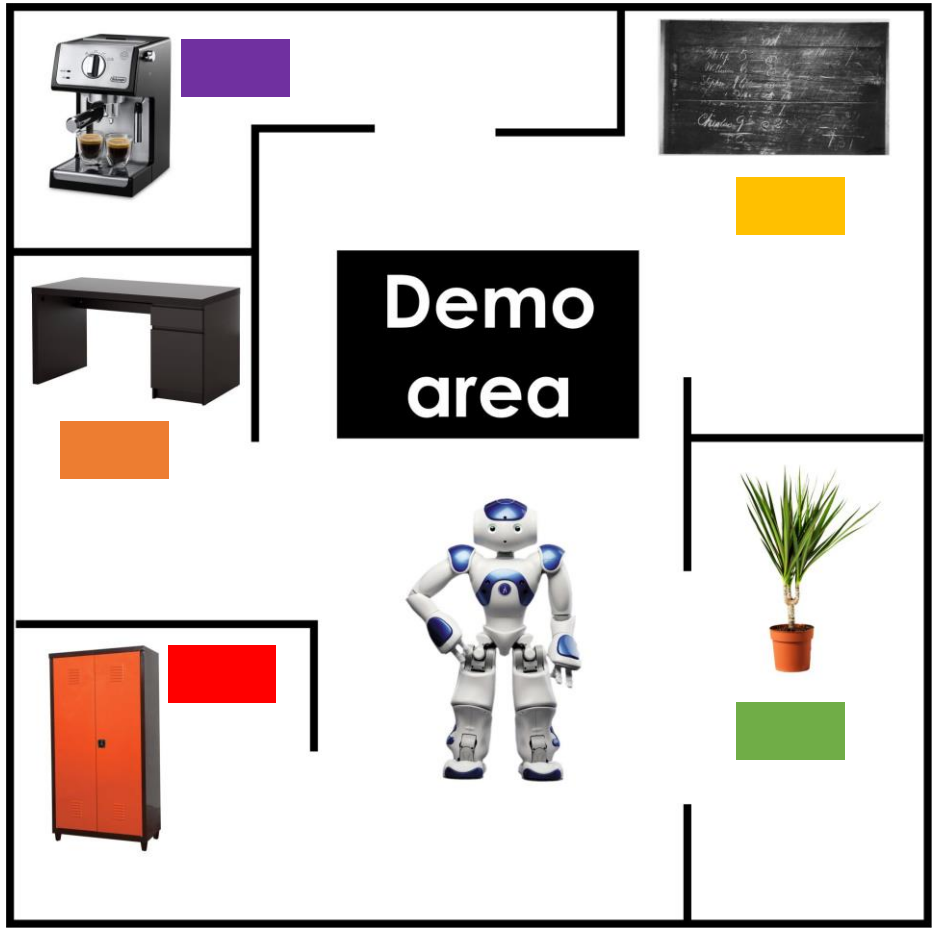
Moving beyond "reach the goal" paradigm

- **Mission:**
  - "Do the following in any order and always avoid the black demo area:
    - Water the plant
    - Clean the blackboard
    - Turn off the coffee maker
    - Pick up vacuum cleaner from the supply room, then vacuum the orange room and dust its table."



What we gain this way

- Safe LTL task planning
- Hybrid systems encode behaviors and planning tasks decomposition
- Changes in the environment are handled more easily
- Integration of more complex cases such as manipulation and planning under uncertainty.



Code Examples and Tasks

- https://github.com/unr-arl/autonomous_mobile_robot_design_course/tree/master/matlab/path-planning/rrt
- https://github.com/unr-arl/autonomous_mobile_robot_design_course/tree/master/ROS/path-planning/structural-inspection
- https://github.com/unr-arl/autonomous_mobile_robot_design_course/tree/master/ROS/path-planning/autonomous-exploration
- https://github.com/unr-arl/DubinsAirplane/tree/52ce13e4a6dea9005da702095e6b0acbb175e008
- https://github.com/unr-arl/autonomous_mobile_robot_design_course/tree/master/python/DubinsCar
- https://github.com/unr-arl/autonomous_mobile_robot_design_course/tree/master/python/HAV_BVS

Find out more

- **Further direct resources on our own work**
  - http://www.autonomousrobotslab.com/autonomous-navigation-and-exploration.html - our papers
  - http://www.autonomousrobotslab.com/holonomic-vehicle-bvs.html
  - http://www.autonomousrobotslab.com/dubins-airplane.html
  - http://www.autonomousrobotslab.com/collision-free-navigation.html
  - http://www.autonomousrobotslab.com/structural-inspection-path-planning.html
- **Good resources to Study**
  - http://biorobotics.ri.cmu.edu/book/
  - http://msl.cs.uiuc.edu/planning/
- **Resources for algorithms**
  - http://ompl.kavrakilab.org/
  - http://moveit.ros.org/
  - https://github.com/ros-planning/3d_navigation
  - http://planning.cs.uiuc.edu/
  - http://plannerarena.org/
  - https://spot.lrde.epita.fr/ and http://www.cs.rice.edu/~mmoll/icaps2016/ltl2a.h.html
  - https://github.com/ethz-asl
  - https://github.com/unr-arl

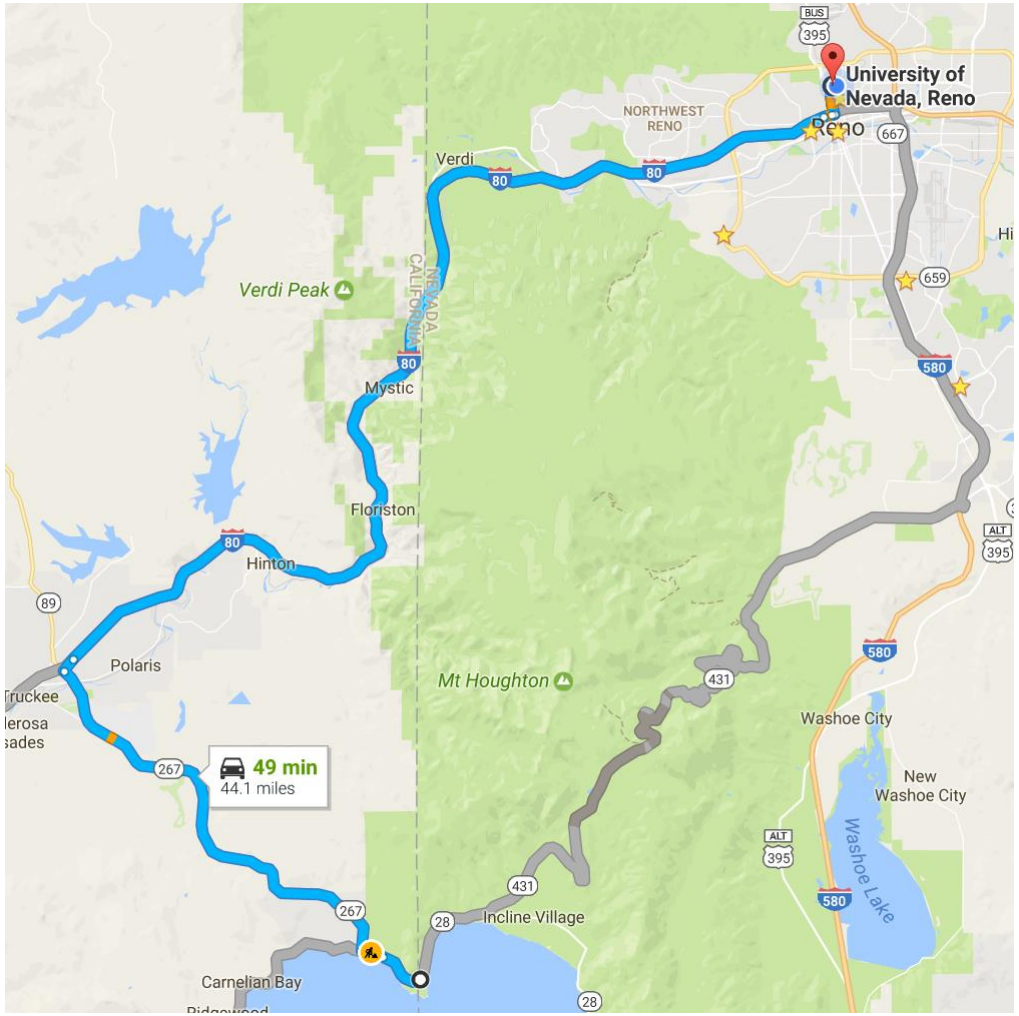
Papers

- Kavraki, Lydia E., et al. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces." *IEEE transactions on Robotics and Automation* 12.4 (1996): 566-580.
- LaValle, Steven M. "Rapidly-exploring random trees: A new tool for path planning." (1998).
- Nedunuri, Srinivas, et al. "SMT-based synthesis of integrated task and motion plans from plan outlines." *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014.
- Bry, Adam, and Nicholas Roy. "Rapidly-exploring random belief trees for motion planning under uncertainty." *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011.
- A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, R. Siegwart, "Receding Horizon "Next-Best-View" Planner for 3D Exploration", IEEE International Conference on Robotics and Automation 2016 (ICRA 2016), Stockholm, Sweden
- A. Bircher, K. Alexis, M. Burri, P. Oettershagen, S. Omari, T. Mantel, R. Siegwart, "Structural Inspection Path Planning via Iterative Viewpoint Resampling with Application to Aerial Robotics", IEEE International Conference on Robotics & Automation, May 26-30, 2015 (ICRA 2015), Seattle, Washington, USA .
- Christos Papachristos, Shehryar Khattak, Kostas Alexis, "Uncertainty-aware Receding Horizon Exploration and Mapping using Aerial Robots", IEEE International Conference on Robotics and Automation (ICRA), May 29-June 3, 2017, Singapore

Appendix

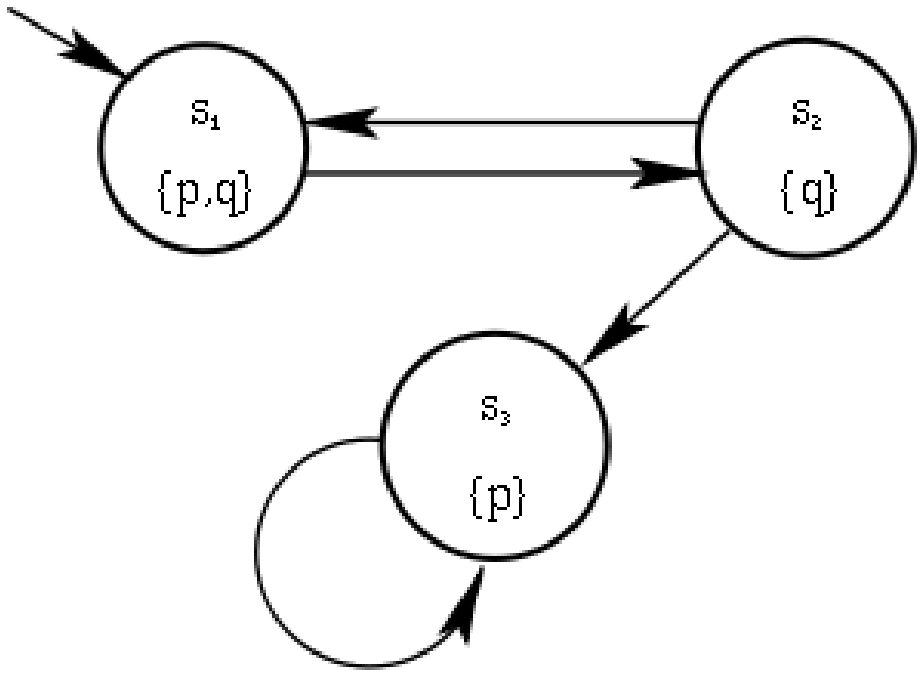
- **Dijkstra's Algorithm**
  - How do we go from UNR to Crystal Bay or vice versa?
  - Single-source shortest path problem
    - Weighted graph **G = [E,V]**
    - Find path from source vertex s to vertices v
    - Works with both directed and undirected graphs

```
dist[s] ← 0
forall v ∈ V-{s}
    do dist[v] ← inf
S ← empty
Q ← Ø
while Q ! ← Ø
    S ← S U u
    forall v ∈neighbors[u]
        do if dist[v] > dist[u] + w(u,v)
            then d[v] ← d[u] + w(u,v)
```



Appendix: Kripke Solution

- A Kripke structure is a variation of the transition system, originally proposed by Saul Kripke, used in model checking to represent the behavior of a system. It is basically a graph whose nodes represent the reachable states of the system and whose edges represent state transitions.



Appendix: Linear Temporal Logic

- **Idea:**
  - LTL as a temporal logic to express properties of paths in Kripke structures.
- **Syntax:**
  - LTL syntax contains rules for temporal operators.
- **Semantics:**
  - Interprets the meaning of LTL expressions.
  - Kripke structures are used to define semantics.

Appendix: Linear Temporal Logic

- Syntax of LTL (1/2):
  - Linear Temporal Logic reuses concepts from first order logic
    - Atomic propositions (labels of states in Kripke structures)
    - Boolean operators
    - Temporal operators
  - Evaluates ordering of states in computation (path)
  - **Temporal operators**
    - X (next)
      - Requires that a property holds in the next state of the path
    - F (eventually)
      - Is used to assert that a property will hold at some state in the path
    - G (always, globally)
      - Specifies that a property holds at every state on the path
    - U (a Until b)
      - There is a state on the path where b holds, and at every state before that, a holds

Appendix: Linear Temporal Logic

- Syntax of LTL (2/2):
  - The set of propositional logic formulae over a set of atomic propositions (AP) is defined by the following rules:
    - TRUE is a formula
    - Any atomic proposition is a formula
    - If f and g are formulae, then so are:
      - ¬f, f v g, and f ^ g
      - x f
      - F f
      - Gf , and
      - f U g

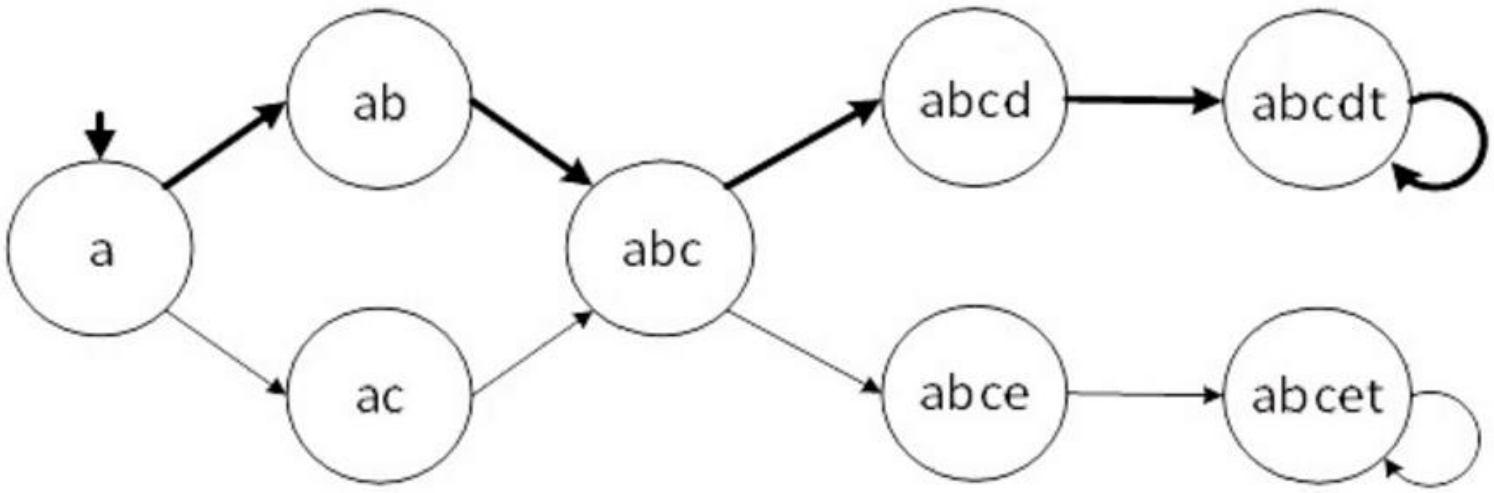
Appendix: Linear Temporal Logic

- Semantics of LTL: defined based on paths in Kripke structures.
- Kripke structure $M = (S, S_0, R, L)$ with set S of states, a transition relation R, and a labeling function L, assigning sets of atomic propositions to states.
- A path in M is an infinite sequence of states $\pi = s_0, s_1, \ldots$ s.t. $(s_i, s_{i+1}) \in R$ for all $i \geq 0$
- π^i is the suffix of π starting at s_i

Appendix: Linear Temporal Logic

- If f is a formula, M, s |= f means: in state s of the Kripke structure M holds formula f
- Each formula is a path formula. Path π satisfies f, IF its first state satisfies it.
- $M, \pi \models f \Leftrightarrow M, s_0 \models f$ where $\pi = s_0, s_1$

Appendix: Linear Temporal Logic



- Example:
  - π = {a}, {a,b}, {a,b,c}, {a,b,c,d}, {a,b,c,d,t}, {a,b,c,d,t,t}…

Thank you!

Please ask your question!