Motion Planning and Control

Adane Letta (PhD)

Dec 2024

Outline

- Advanced motion planning algorithms
- Optimal control and model predictive control
- Planning under uncertainty

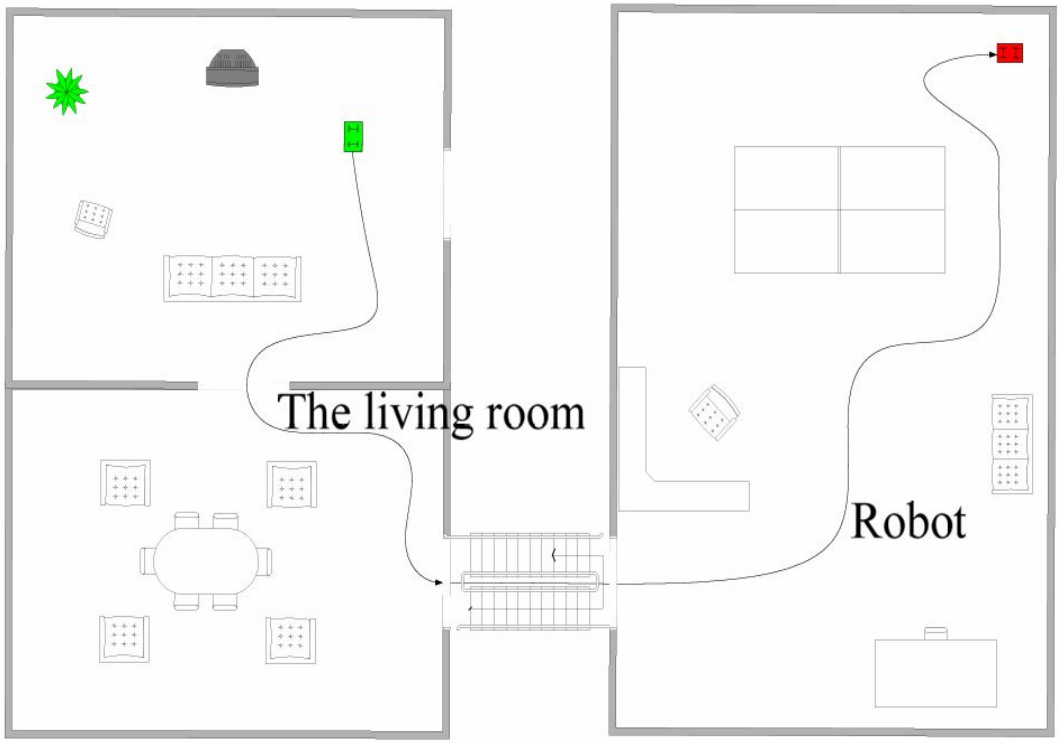
Introduction

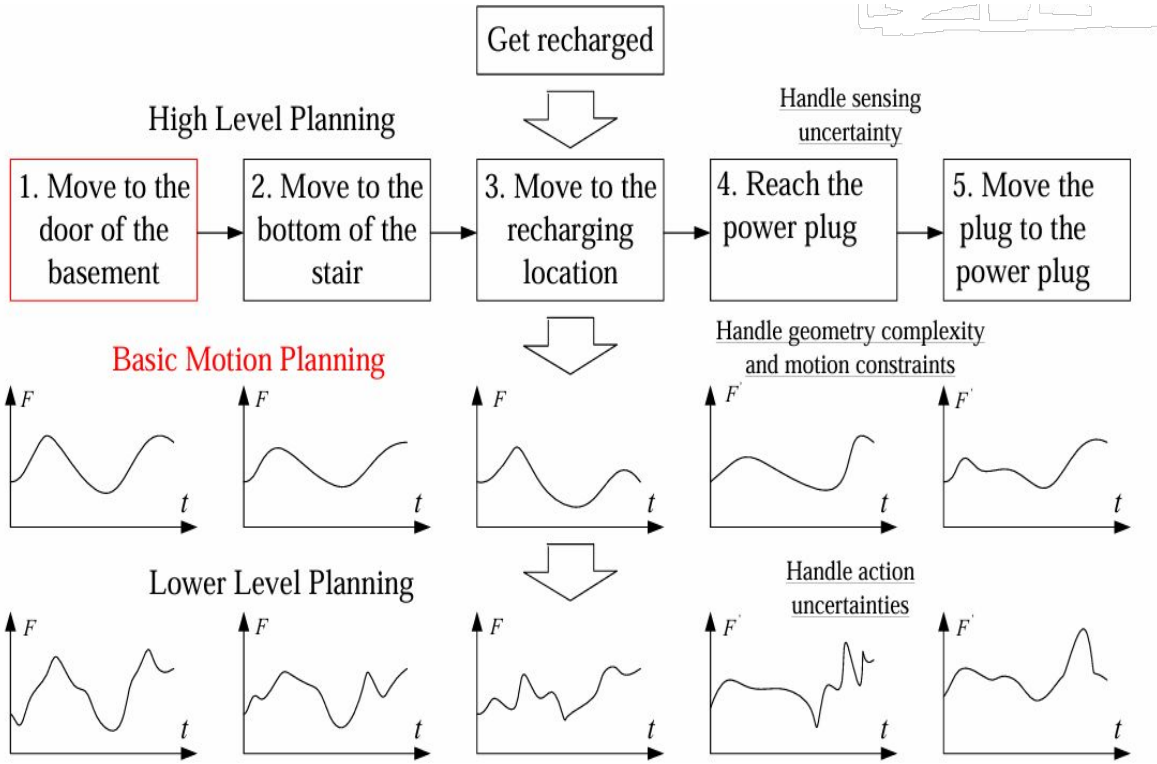
- The position of a robot is normally described by a number of variables: position and orientation of the robot
- A motion for a robot can, hence, be considered as a path in the configuration space.
- Such a path should remain in the subspace of configurations in which there is no collision between the robot and the obstacles, the so-called free space.
- The motion planning problem asks for determining such a path through the free space in an efficient way.

Positon planning: is the process of planning how a robot will move to achieve a specific goal

Goal: to determine a sequence of robot configurations, called trajectory, that can move the robot from its initial to its final position while avoiding collisions.

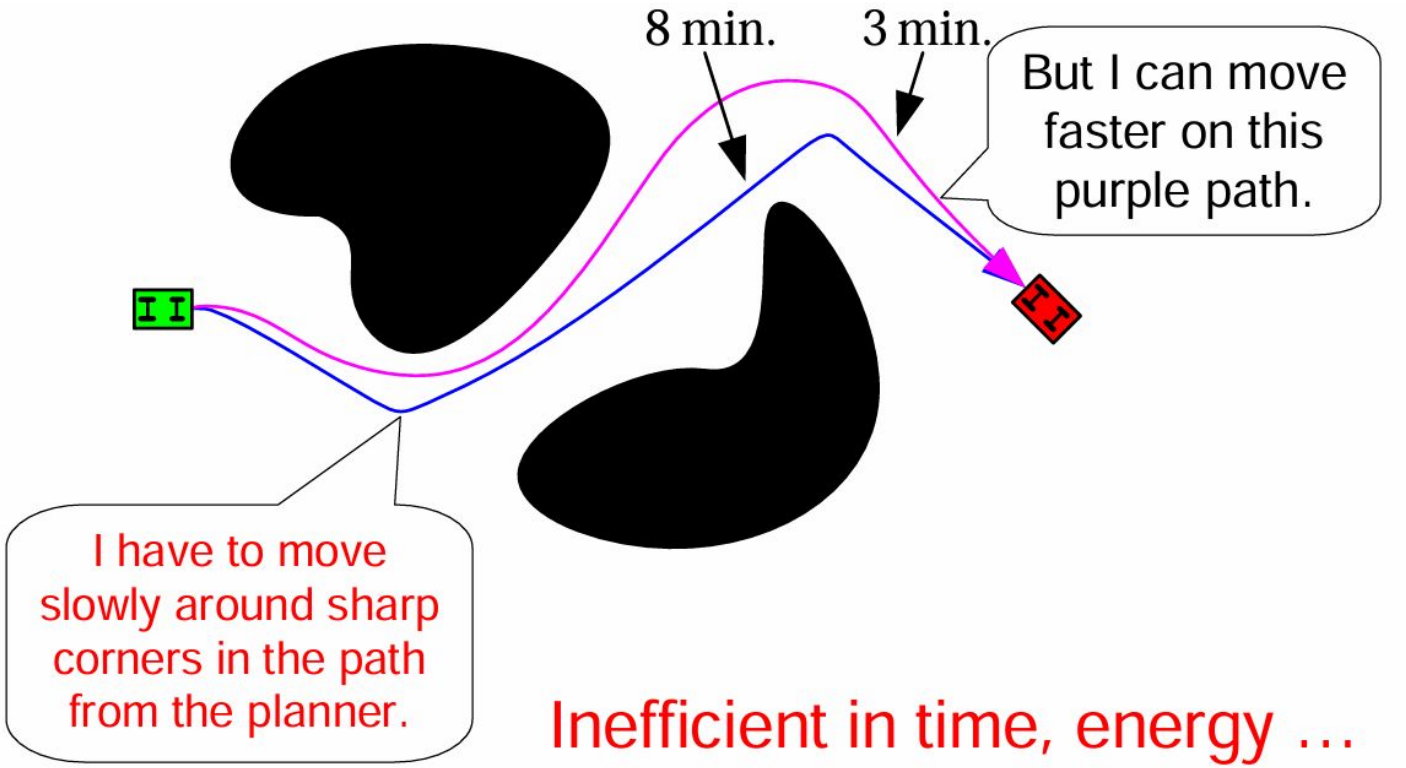
Introduction





Which motion model is appropriate?

Introduction



Introduction

- Motion planning can be split into two classes.
  - When all degrees of freedom can be changed independently-- holonomic motion planning.
    - The existence of a collision-free path is characterized by the existence of a connected component in the free configuration space.
    - Motion planning consists in building the free configuration space, and in finding a path in its connected components.
  - Planning motions in the presence of kinematic constraints (and always amidst obstacles)
    - When the degrees of freedom of a robot system are not independent we talk about nonholonomic motion planning.
    - Much more difficult than holonomic motion planning.
    - A fundamental issue for most types of mobile robots.

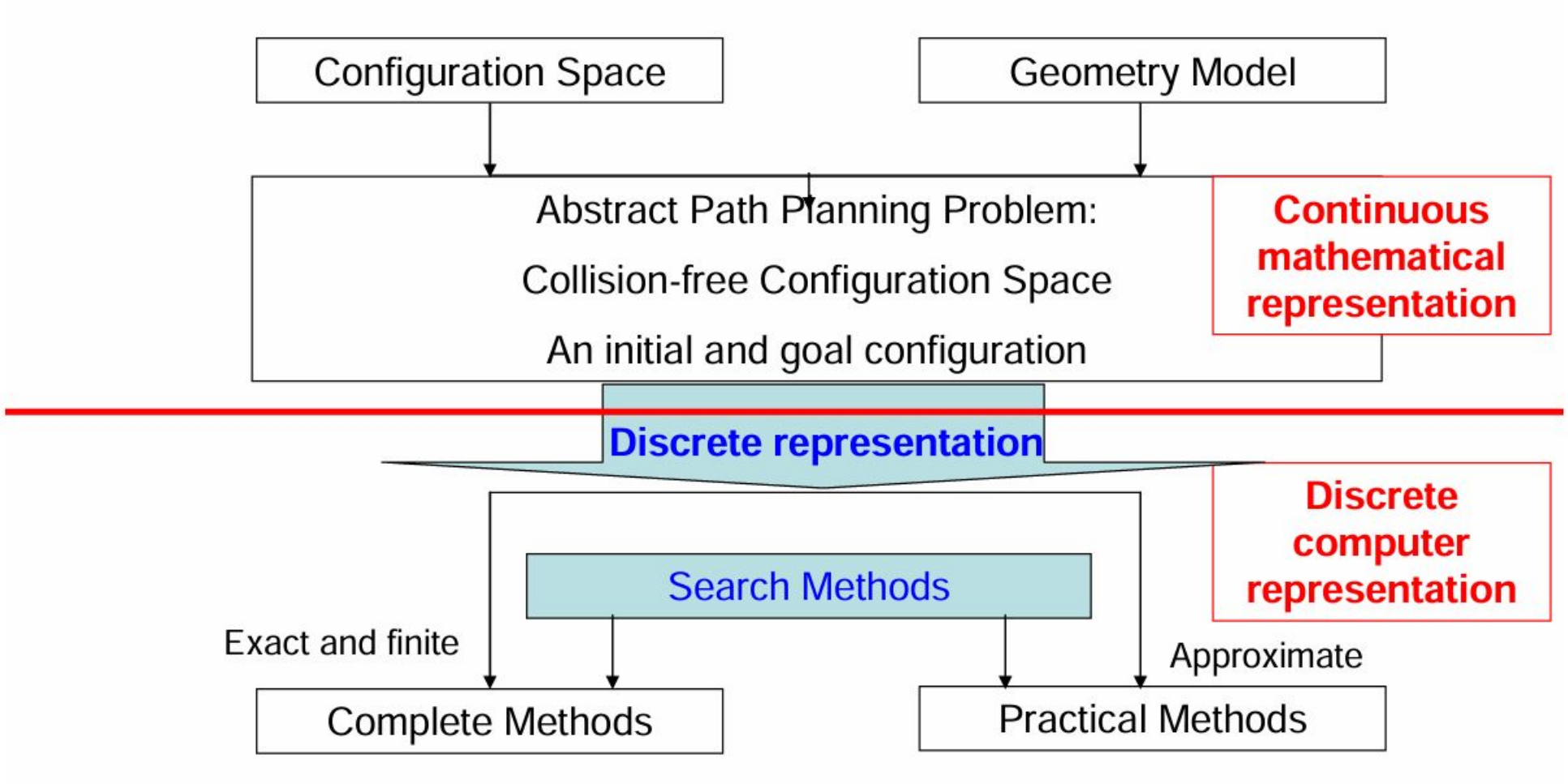
Introduction

- Problem
  - Given start state S, goal state G
  - Asked for: a sequence of control inputs that leads from start to goal

- Why tricky?
  - Need to avoid obstacles
  - For systems with underactuated dynamics: can't simply move along any coordinate at will
    - E.g., car, helicopter, airplane, but also robot manipulator hitting joint limits

- Motion planning is typical solved using algorithms that search the robot's configuration space (C-Space), which is the set of all possible positions and orientation that the robot can reach.
  - Configuration space is high-dimensional – search for a feasible trajectory a challenging problem

Introduction: Algorithms



Configuration Space

- A topological manifold
  - A topological manifold is a fundamental concept in topology and geometry. It is a space that, on a local scale, resembles Euclidean space, but on a global scale, it can have a more complex structure.

- A metric space
  - A metric space is a fundamental concept in mathematics that provides a framework to define and measure the "distance" between elements of a set.

- A differentiable manifold
  - A differentiable manifold is a type of topological manifold that allows for calculus (differentiation) to be performed on it. It extends the concept of curves and surfaces to higher dimensions and more abstract spaces while maintaining smoothness.

Configuration Representation

- Continuous mathematical representation
  - Mathematical formulation of the problem
  - Topology of the configuration space
  - Parameterization of configuration space
  - Computation of collision-free configuration space
  - Abstract path planning problems

- Discrete computational representations
  - Required for computer algorithms
  - Exact and finite representations of the set of collision free configurations
  - Approximate representations

Search Methods

- Discrete search methods
  - Search with finite states and actions
- Deterministic complete search
  - Best-first, A*, Dijstra, …
- Sampling-based algorithms
  - PRM(Probabilistic Road Map)-based methods
  - RRT (Rapidly Exploring Random Tree)-based methods
  - Heuristic design

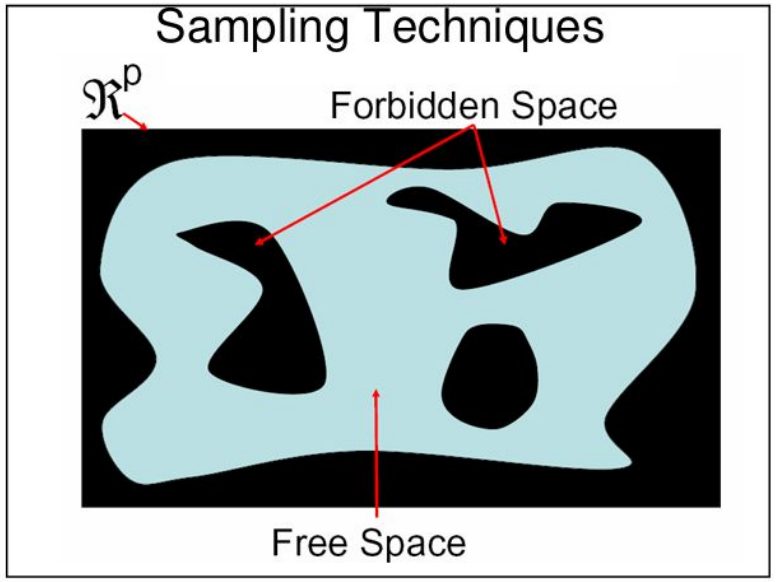
Common approach:

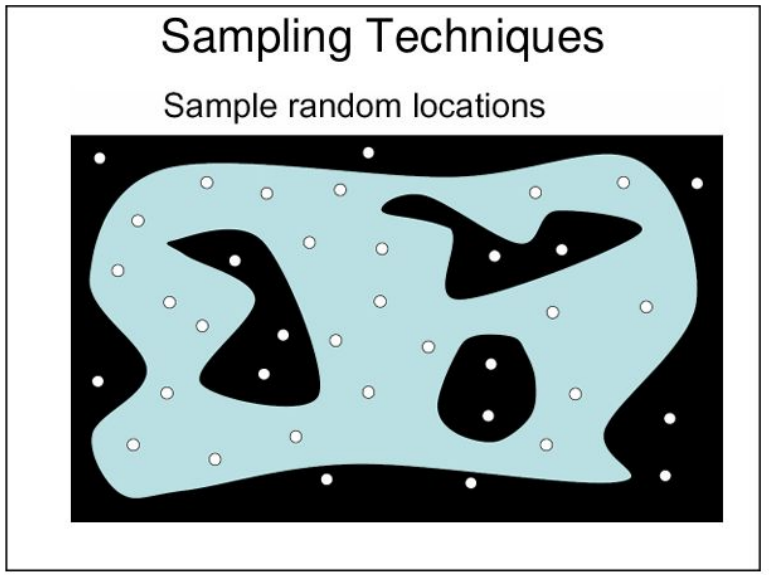
- Sample based algorithms: randomly sample the configuration
- Optimization based algorithms: motion planning problem is an optimization problem

Advanced motion planning algorithms: Sampling-based algorithms-PRM

Probabilistic Roadmap Method (PRM)

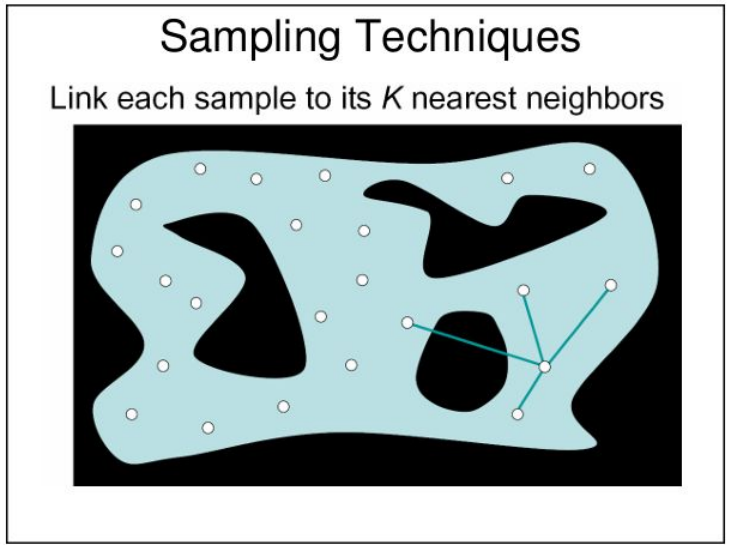
- Limited to finding a “good” sampling of the C-space

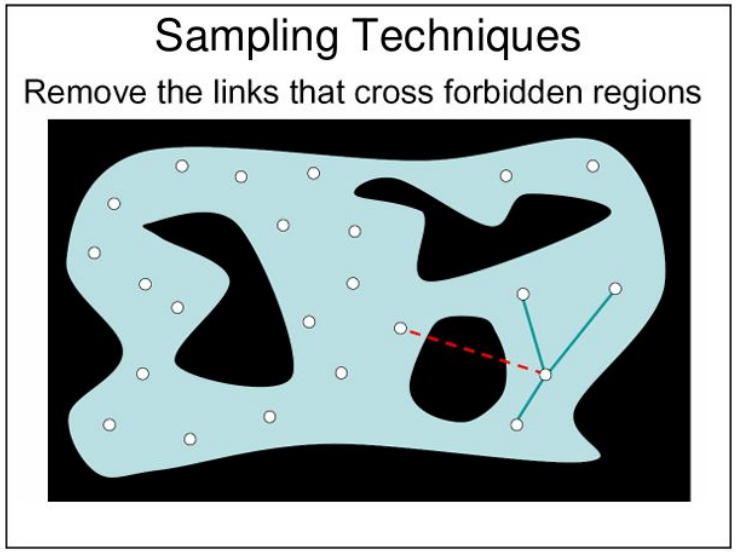


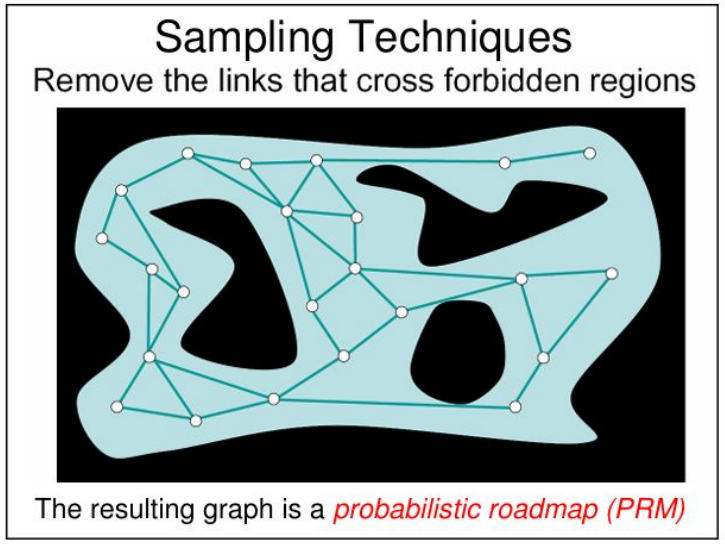


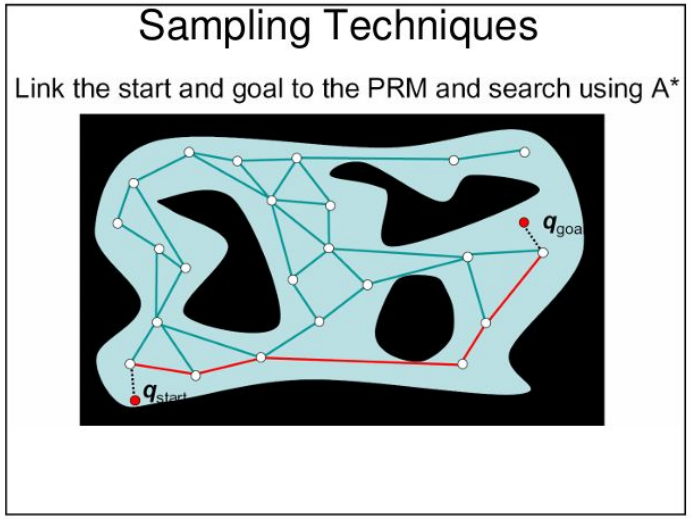
- Sampling based planners can often create plans in high-dimensional spaces efficiently
- Sacrifice completeness to gain simplicity and efficiency

Advanced motion planning algorithms: Sampling-based algorithms-PRM



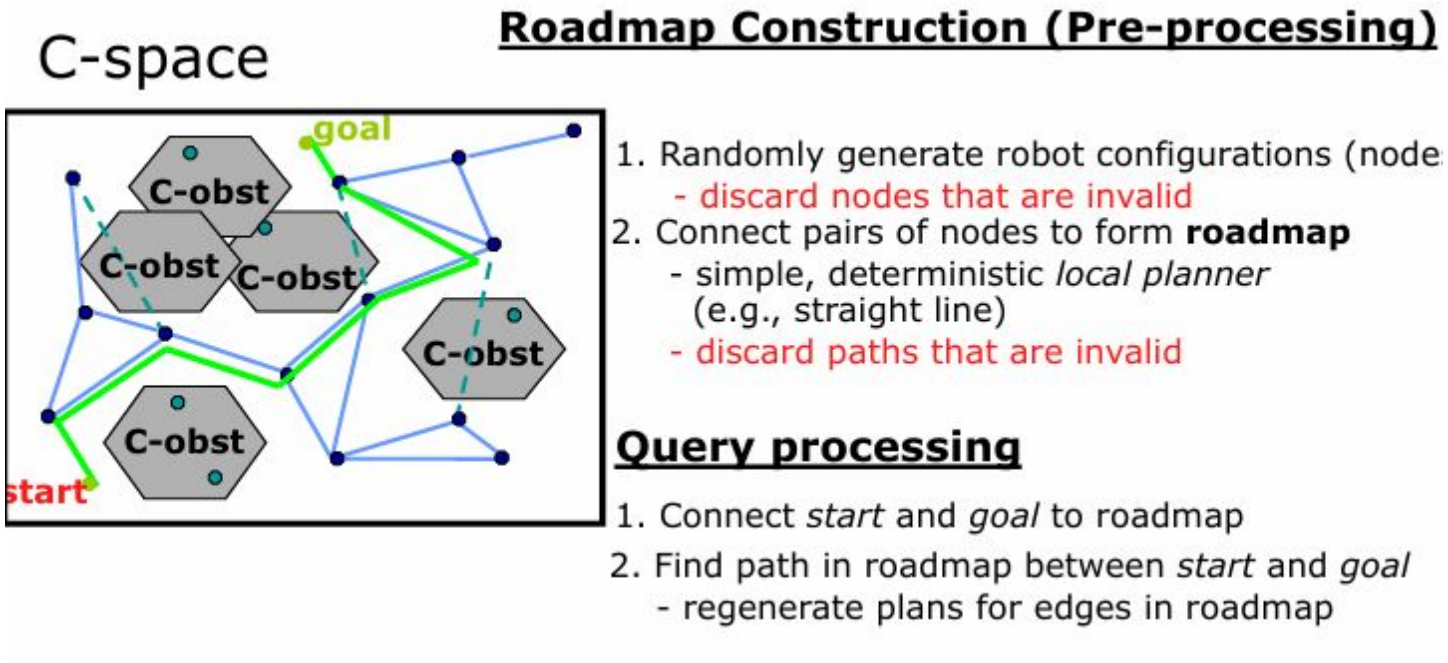






Advanced motion planning algorithms: Sampling-based algorithms-PRM

- Probabilistic Roadmap Method (PRM)



- **Important sub-routines**
  - Generate random configurations
  - Local planners
  - Distance metrics
  - Selecting k-nearest neighbors (becoming dominant in high dimensional space)
  - Collision detection (>80% computation)

Advanced motion planning algorithms: Sampling-based algorithms-PRM

- Initially: empty Graph G
- A configuration q is randomly chosen
- If q→Q_free then added to G (collision detection needed here)
- Repeat until N vertices chosen
- For each q, select k closest neighbors
- Local planner Δ connects q to neighbor q'
- If connect successful (i.e. collision free local path), add edge (q, q')

Algorithm 6 Roadmap Construction Algorithm

Input:

n : number of nodes to put in the roadmap

k : number of closest neighbors to examine for each configuration

Output:

A roadmap $G = (V, E)$

```
1:  V ← ∅
2:  E ← ∅
3:  while |V| < n do
4:     repeat
5:        q ← a random configuration in Q
6:     until q is collision-free
7:     V ← V ∪ {q}
8:  end while
9:  for all q ∈ V do
10:    N_q ← the k closest neighbors of q chosen from V according to dist
11:    for all q' ∈ N_q do
12:       if (q, q') ∉ E and Δ(q, q') ≠ NIL then
13:          E ← E ∪ {(q, q')}
14:       end if
15:    end for
16: end for
```

Advanced motion planning algorithms: Sampling-based algorithms-PRM

Finding a Path

- Given q_init and q_goal, need to connect each to the roadmap
- Find k nearest neigbors of q_init and q_goal in roadmap, plan local path Δ
- Problem: Roadmap Graph may have disconnected components…
- Need to find connections from q_init, q_goal to same component
- Once on roadmap, use different algorithms

Algorithm 7 Solve Query Algorithm

Input:

q_{init}: the initial configuration
q_{goal}: the goal configuration
k: the number of closest neighbors to examine for each configuration
$G = (V, E)$: the roadmap computed by algorithm 6

Output:

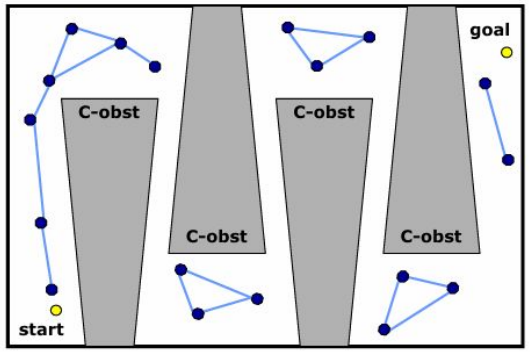
A path from q_{init} to q_{goal} or failure

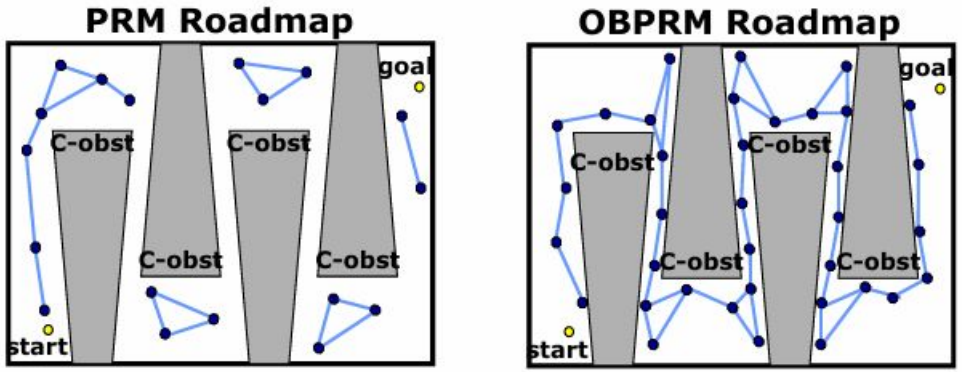
1: $N_{q_{\text{init}}} \leftarrow$ the k closest neighbors of q_{init} from V according to *dist*
2: $N_{q_{\text{goal}}} \leftarrow$ the k closest neighbors of q_{goal} from V according to *dist*
3: $V \leftarrow \{q_{\text{init}}\} \cup \{q_{\text{goal}}\} \cup V$
4: set q' to be the closest neighbor of q_{init} in $N_{q_{\text{init}}}$
5: **repeat**
6: **if** $\Delta(q_{\text{init}}, q') \neq \texttt{NIL}$ **then**
7: $E \leftarrow (q_{\text{init}}, q') \cup E$
8: **else**
9: set q' to be the next closest neighbor of q_{init} in $N_{q_{\text{init}}}$
10: **end if**
11: **until** a connection was succesful or the set $N_{q_{\text{init}}}$ is empty
12: set q' to be the closest neighbor of q_{goal} in $N_{q_{\text{goal}}}$
13: **repeat**
14: **if** $\Delta(q_{\text{goal}}, q') \neq \texttt{NIL}$ **then**
15: $E \leftarrow (q_{\text{goal}}, q') \cup E$
16: **else**
17: set q' to be the next closest neighbor of q_{goal} in $N_{q_{\text{goal}}}$
18: **end if**
19: **until** a connection was succesful or the set $N_{q_{\text{goal}}}$ is empty
20: $P \leftarrow$ shortest path$(q_{\text{init}}, q_{\text{goal}}, G)$
21: **if** P is not empty **then**
22: **return** P
23: **else**
24: **return** `failure`
25: **end if**

Advanced motion planning algorithms: Sampling-based algorithms-PRM

- Pros
  - PRMs are probabilistically complete
  - PRMs apply easily to high-dimensional C-space
  - PRMs support fast queries w/ enough preprocessing
  - Many success stories where PRMs solve previously unsolved problems

- Cons
  - PRMs don't work as well for some problems:
    - Unlikely to sample nodes in narrow passages
    - Hard to sample/connect nodes on constraint surfaces





Idea: Can we sample nodes near C-obstacle surfaces?

- we cannot explicitly construct the C-obstacles...
- we do have models of the (workspace) obstacles...

- Gaussian Sampling PRM

Advanced motion planning algorithms: Sampling-based algorithms-PRM

- Unanswered Questions
  - How are random configurations chosen?
  - How are closest neighbors found?
  - How do we choose distance function?
  - How are local path’s generated?

Advanced motion planning algorithms: Sampling-based algorithms-PRM

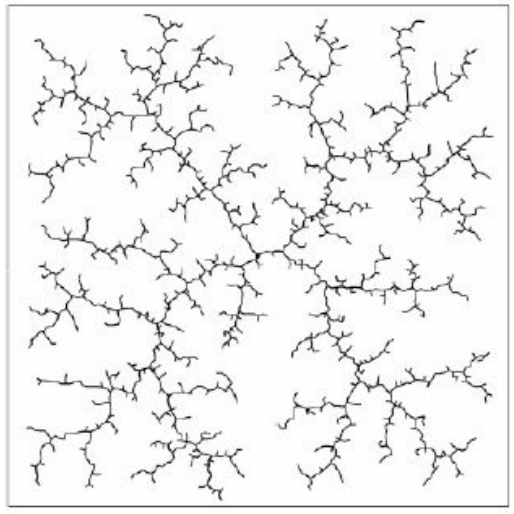
- Tree Based single shot planners – compute the respresentation of Cfree for single start and goal
- Extends to more advanced planning techniques
  - Integrates the control inputs to ensure that the kinodynamic constraints are satisfied

Advanced motion planning algorithms: Sampling-based algorithms

- Planning in high dimensional spaces
  - Single query planning $q_{initial}$ and q_{goal} are given once – no pre-computation (greedy search technique can take a long time)
  - Multiple query planning – spreads out uniformly, requires lot of samples to cover the space
  - Next incremental sampling and search methods that yields good performance without parameters tunning. Idea gradually construct search tree, such that it densely covers the space

Advanced motion planning algorithms: Sampling-based algorithms: Rapidly Exploring Random Tree (RRT)

- Single query model – given start and goal q find a path
- Analogy with the discrete search algorithms
- Samples are states, edges are paths connected them
- Graphs are undirected (Tree) Ingredients
- Idea: Incrementaly construct the search tree, that improves with resolution
  Previous incremental search methods could spend long time exploring nodes inside unimportant cavities



RRT algorithm possesses both search and random sampling properties, and thus has more potential to generate high-quality paths that can balance the global optimum and local optimum (Tong, 2024).

Take the initial position as the root node and then add leaf nodes by random sampling.

Advanced motion planning algorithms: Sampling-based algorithms: Rapidly Exploring Random Tree (RRT)

- Rapidly-Exploring Random Tree (RRT)
  - Tree Based single shot planners – compute the representation of C-free for single start and goal
  - Incrementally builds the roadmap tree
  - Extends to more advanced planning techniques
    - Integrates the control inputs to ensure that the kinodynamic constraints are satisfied

Advanced motion planning algorithms: Sampling-based algorithms: Rapidly Exploring Random Tree (RRT)

- How it works
  1. Initialize the graph
  2. Select vertex for expansion
  3. Generate set of new vertices
  4. For some new vertices run a local planner and check whether its collision free
  5. If yes insert an edge to the graph
  6. Keep on going until termination condition is satisfied

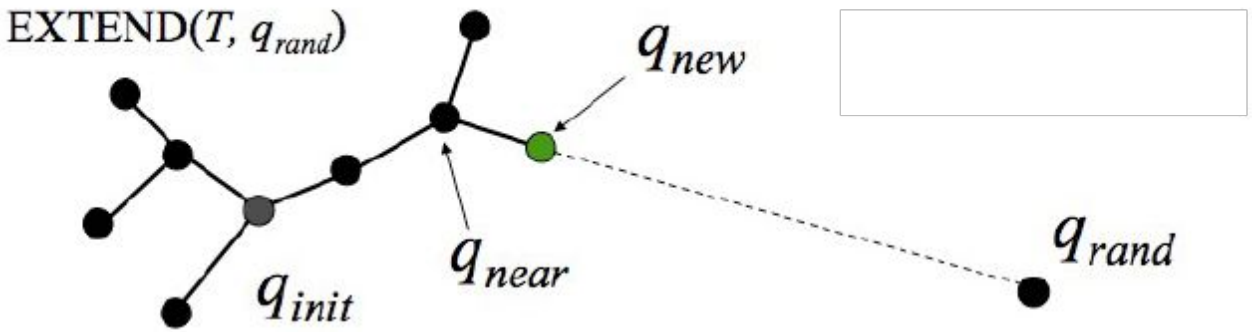
Start with the initial configuration q-initial build the graph (actually, tree) until the goal configuration q-goal is part of it.

Advanced motion planning algorithms: Sampling-based algorithms: Rapidly Exploring Random Tree (RRT)

- Path Planning with RRT Algorithm
  - 2 trees, T_init, T_goal, rooted at q_init, q_goal
  - Each tree is expanded by:
    - q_rand is generated from uniform dist.
    - q_near is found, nearest tree node to q_rand
    - move step-size along line (q_near, q_rand) to q_new. If no collision, add q_new to tree
  - If trees merge, path is found

Advanced motion planning algorithms: Sampling-based algorithms: Rapidly Exploring Random Tree (RRT)

```
BUILD_RRT (q_init) {
  T.init(q_init);
  for k = 1 to K do
    q_rand = RANDOM_CONFIG();
    EXTEND(T, q_rand)
}
```



- Random sample connects to the nearest node so far
- If the nearest point lies on an edge, the edge is split in two.

- Details:
  - Step length: how far to sample
  - Sample just at the end point
  - Sample all along, small steps
  - Extend returns the new edge

Advanced motion planning algorithms: Sampling-based algorithms: Rapidly Exploring Random Tree (RRT)

- RRT pseudo-code details
  - Add start node to the tree
  - Repeat n times
    - Generate random configuration x
    - If x is in free space using CollisionCheck

find y, the closes node in the tree to the configuration x
if dist(x,y) > delta – check if x is too far away from y
find a configuration z that is along the path from x to y such that dist(z, y) <= delta
x = z;
if (LocalPlanner(x,y) – check if you can get from x to y
if yes add x to the graph

Algorithm 10 Build RRT Algorithm

Input:

q_0: the configuration where the tree is rooted

n : the number of attempts to expand the tree

Output:

A tree $T = (V, E)$ that is rooted at q_0 and has $\leq n$ configurations

1: $V \leftarrow \{q_0\}$
2: $E \leftarrow \emptyset$
3: **for** $i = 1$ to n **do**
4: $q_{\text{rand}} \leftarrow$ a randomly chosen free configuration
5: extend RRT (T, q_{rand})
6: **end for**
7: **return** T

Algorithm 11 Extend RRT Algorithm

Input:

$T = (V, E)$: an RRT

q: a configuration toward which the tree T is grown

Output:

A new configuration q_{new} toward q, or NIL in case of failure

1: $q_{\text{near}} \leftarrow$ closest neighbor of q in T
2: $q_{\text{new}} \leftarrow$ progress q_{near} by step_size along the straight line in $\mathcal{Q}$ between q_{near} and q_{rand}
3: **if** q_{new} is collision-free **then**
4: $V \leftarrow V \cup \{q_{\text{new}}\}$
5: $E \leftarrow E \cup \{(q_{\text{near}}, q_{\text{new}})\}$
6: **return** q_{new}
7: **end if**
8: **return** NIL

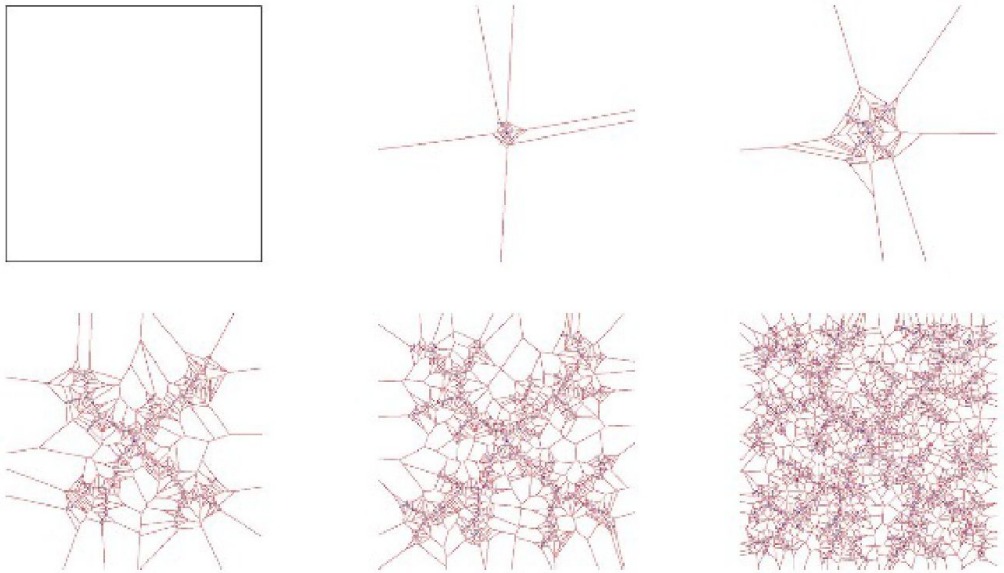
Advanced motion planning algorithms: Sampling-based algorithms: Rapidly Exploring Random Tree (RRT)

- Random sample: Naïve Random Tree
  - Start with middle
  - Sample near this node
  - Then pick a node at random in tree
  - Sample near it
  - End up staying in middle

Advanced motion planning algorithms: Sampling-based algorithms: Rapidly Exploring Random Tree (RRT)

- RRT's are biased towards large Voronoi cells

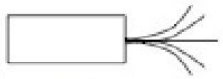
The nodes most likely to be closest to a randomly chosen point in state space are those with the largest Voronoi regions. The largest Voronoi regions belong to nodes along the frontier of the tree, so these frontier nodes are automatically favored when choosing which node to expand.



Advanced motion planning algorithms: Sampling-based algorithms: Rapidly Exploring Random Tree (RRT)

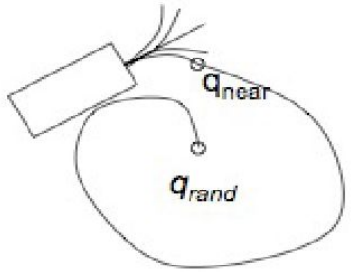
- Taking actions into account
  - Instead of moving in a straight line for some distance, take into account kinematic constraints

q_{near}

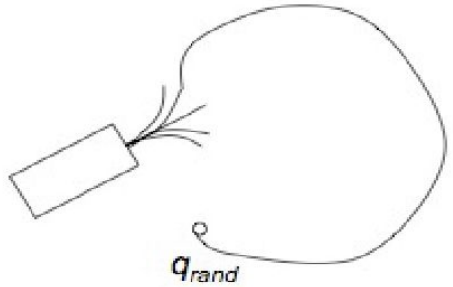


$q' = f(q,u)$ - - - use action u from q to arrive at q'

chose $u_* = \arg\min(d(q_{rand}, q'))$

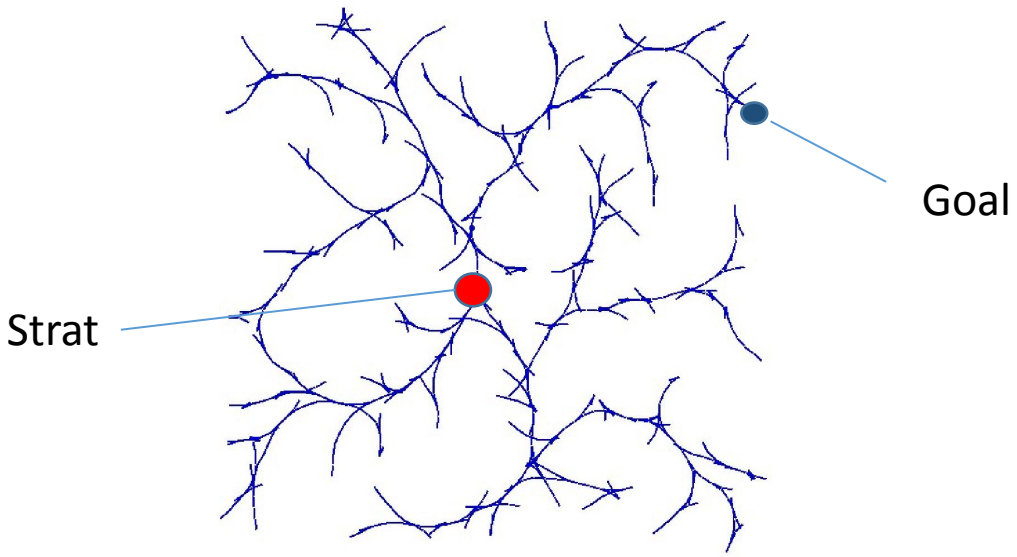


Is this the best?



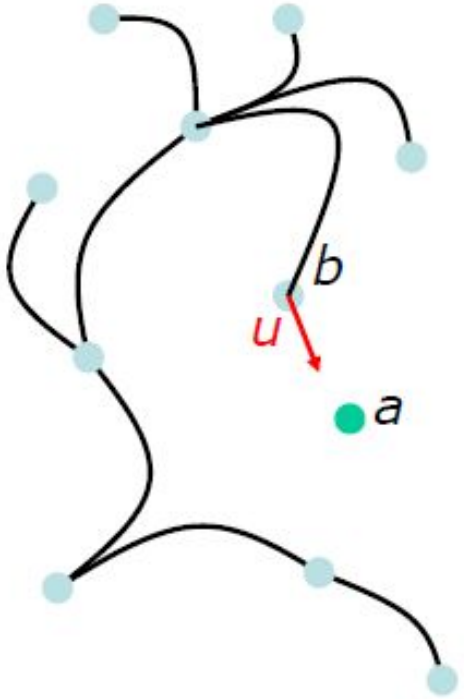
Advanced motion planning algorithms: Sampling-based algorithms: Rapidly Exploring Random Tree (RRT)

- How it Works
  - Build a rapidly-exploring random tree in state space (X), starting at s_{start}
  - Stop when tree gets sufficiently close to s_{goal}



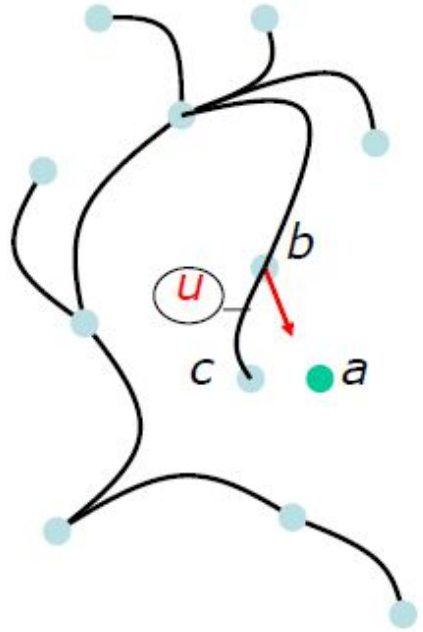
Advanced motion planning algorithms: Sampling-based algorithms: Rapidly Exploring Random Tree (RRT)

- Building an RRT
  - To extend an RRT:
  - Pick a random point *a* in *X*
  - Find *b*, the node of the tree closest to *a*
  - Find control inputs *u* to steer the robot from *b* to *a*



To extend an RRT (cont.)

- Apply control inputs u for time δ, so robot reaches c
- If no collisions occur in getting from a to c, add c to RRT and record u with new edge



Advanced motion planning algorithms: Sampling-based algorithms: Rapidly Exploring Random Tree (RRT)

- Executing the Path
  - Once the RRT reaches s_{goal}
    - Backtrack along tree to identify edges that lead from s_{start} to s_{goal}
    - Drive robot using control inputs stored along edges in the tree

Advanced motion planning algorithms: Sampling-based algorithms: Rapidly Exploring Random Tree (RRT)

Algorithm 12 Connect RRT Algorithm

Input:

$T = (V, E)$: an RRT

q: a configuration toward which the tree T is grown

Output:

`connected` if q is connected to T; `failure` otherwise

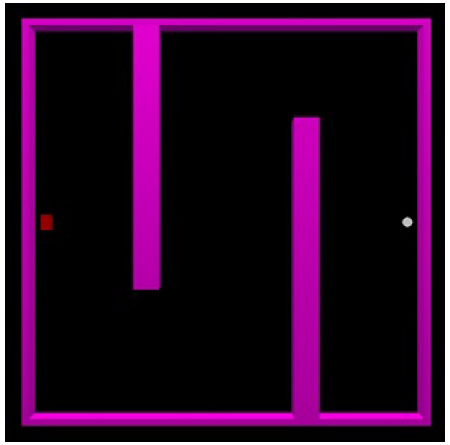
```
1: repeat
2:    q_new ← extend RRT (T, q)
3: until (q_new = q or q_new = NIL)
4: if q_new = q then
5:    return connected
6: else
7:    return failure
8: end if
```

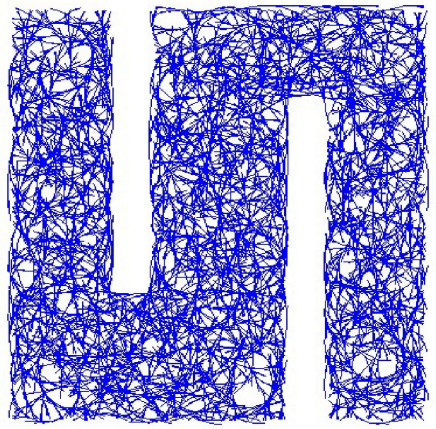
Advanced motion planning algorithms: Sampling-based algorithms: Rapidly Exploring Random Tree (RRT)

- RRT Tradeoffs
  - If step-size is small, many nodes generated, close together
  - As number of nodes increases, nearest-neighbor computation slows down
  - Suffer from the problems of difficult to design the hyper-parameters and weak generalization

Advanced motion planning algorithms: Sampling-based algorithms: Rapidly Exploring Random Tree (RRT)

- Problem of Simple RRT Planner

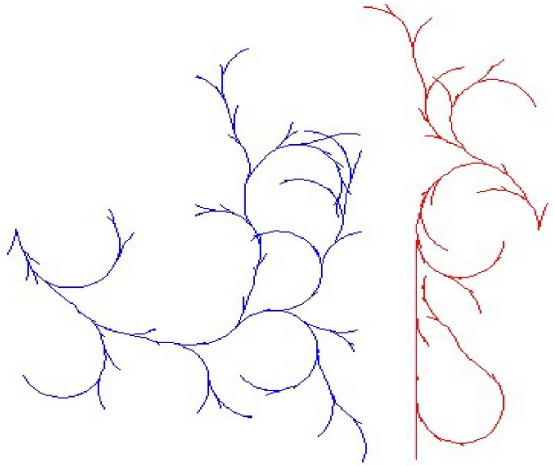




- Ordinary RRT explores *X* uniformly $\longrightarrow$ slow convergence
  - Solution: bias distribution towards the goal – once in a while choose goal as new random configuration
- If goal is chosen 100% time then it is randomized potential planner

Advanced motion planning algorithms: Sampling-based algorithms: Rapidly Exploring Random Tree (RRT)

- Bidirectional Planners
  - Build two RRTs, from start and goal state



- Complication: need to connect two RRTs
  - local planner will not work (dynamic constraints)
  - **bias** the distribution, so that the trees meet

Advanced motion planning algorithms: Sampling-based algorithms: Rapidly Exploring Random Tree (RRT)

- Applications
  - Welding robots
  - Assembly robots
  - Search and rescue robots
  - Surgical robots
  - Free-floating space robots, and inspection robots
- Improvements
  - Branching strategy improvement,
  - Sampling strategy improvement,
  - Post-processing improvement,
  - Model-driven RRT

Advanced motion planning algorithms: Sampling-based algorithms: Rapidly Exploring Random Tree (RRT)

- PRMs vs. RRTs
  - PRMs construct a roadmap and then searches it for a solution whenever q-initial, q-goal are given
    - Well-suited for repeated planning in between different pairs of q-initial, q-goal(multiple queries)
  - RRTs construct a tree for a given q-initial, q-goal until the tree has a solution
    - Well-suited for single-shot planning in between a single pair of q-initial, q-goal (single query)
    - There exist extensions of RRTs that try to reuse a previously constructed tree when re-planning in response to map updates

Advanced motion planning algorithms

Performance comparison of classical algorithms (Jong, 2024).

Algorithms	Principle	Advantages	Disadvantages
Dijkstra	Expand outward, layer by layer, centered on the starting point, until it reaches the target point.	Optimal paths can be found	Too many nodes traversed
A*	The cost function is designed based on the Dijkstra algorithm	Improved search efficiency compared to the Dijkstra algorithm	computationally expensive
D*	An improved version of A*	For dynamic path planning problems, it reduces the repeated computation of the same data by the A* algorithm and is an improved version of the A* algorithm.	Not applicable to changes occurring on the shortest path over long distances

Advanced motion planning algorithms

Visibility graph	Connect the obstacle to the target point with a line segment and remove the connecting line through the obstacle; the remaining connecting line is the desired visual graph.	The shortest path can be solved and path generation is also efficient	Algorithm complexity is limited by the shape and number of obstacles, and the representation of environmental information is too idealized
APF	Take the target point as the gravitational force and the obstacle as the repulsive force, and plan the obstacle avoidance path through the resultant force	Path generation is efficient	Prone to local minima and unable to find paths between neighboring obstacles
GA	Mimicking the process of natural selection and reproduction	Strong global search capability for suboptimal solutions	weak local search capability, very long time to get the optimal solution

Advanced motion planning algorithms

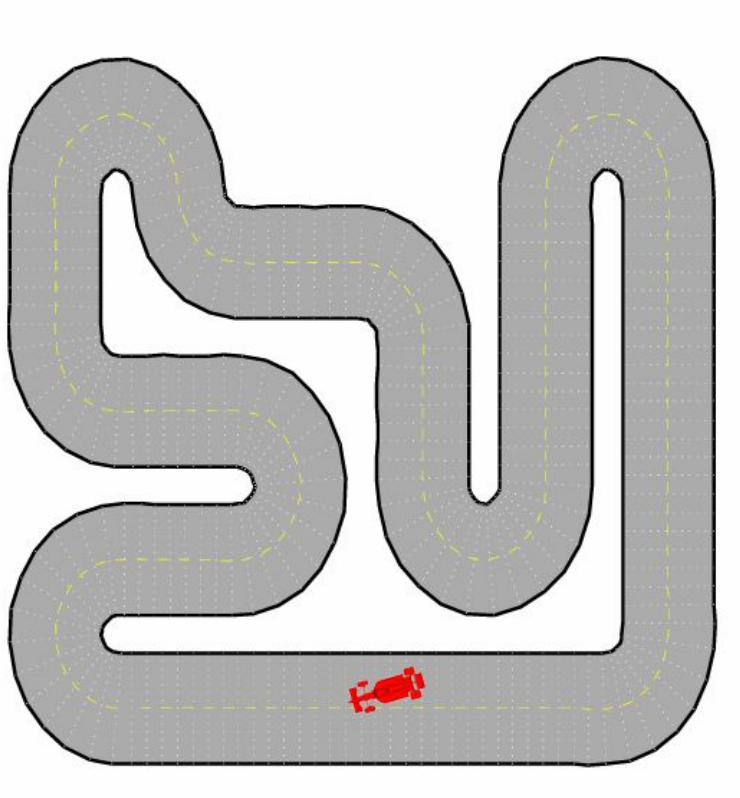
PSO	Mimicking a flock of birds to share information	Fast convergence	premature convergence, easy to fall into local optimal solutions
ACO	Mimicking an ant colony to share information	Fast convergence	Complex parameter settings leading to deviation from high quality solutions
PRM	Random sampling in obstacle maps	Plan paths in higher dimensional spaces; Paths can be conform to the constraints of the robot's kinematic model	Inefficient search in complex environments; Random sampling leads to redundancy of most nodes and increases computational cost
RRT	Build a tree that grows and spreads in all directions with the starting point as the root node, and nodes are also obtained by random sampling in the	In contrast to the PRM algorithm, the random sampling process can be oriented by target bias probability	Inefficient search in complex environments;

Model Predictive Control

- Model predictive control (MPC) is a widely used optimal control method for robot path planning and obstacle avoidance.
- MPCs are widely used in robotics due to their ability to handle constraints and run in real-time
- MPC requires a system model to optimize control over a finite time horizon and possible trajectories.
- Certain types of robots, such as soft robots, continuum robots, and transforming robots, can be challenging to model, especially in unstructured or unknown environments

Model Predictive Control

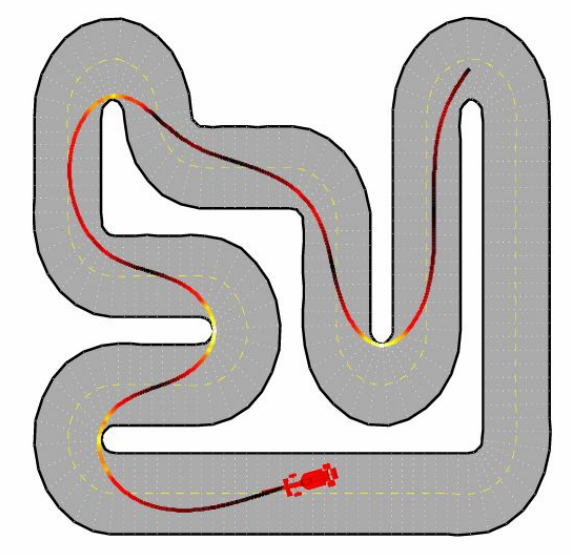
- Main Idea
  - Objective: Minimize lap time
  - Constraints:
    - Avoid other cars
    - Stay on road
    - Don’t skid
    - Limited acceleration
  - Intuitive approach:
    - Look forward and plan path based on
      - Road conditions
      - Upcoming corners
      - Abilities of car
      - etc...



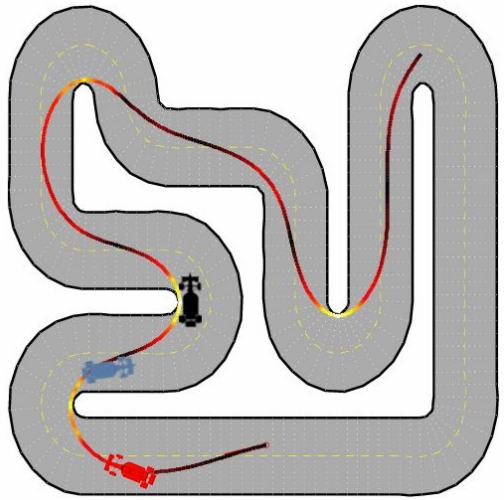
Ack: Manfred Morari, Institut f¨ur Automatik, ETH Z¨urich

Model Predictive Control

- Optimization-Based Control
  - Solve optimization problem to compute minimum-time path

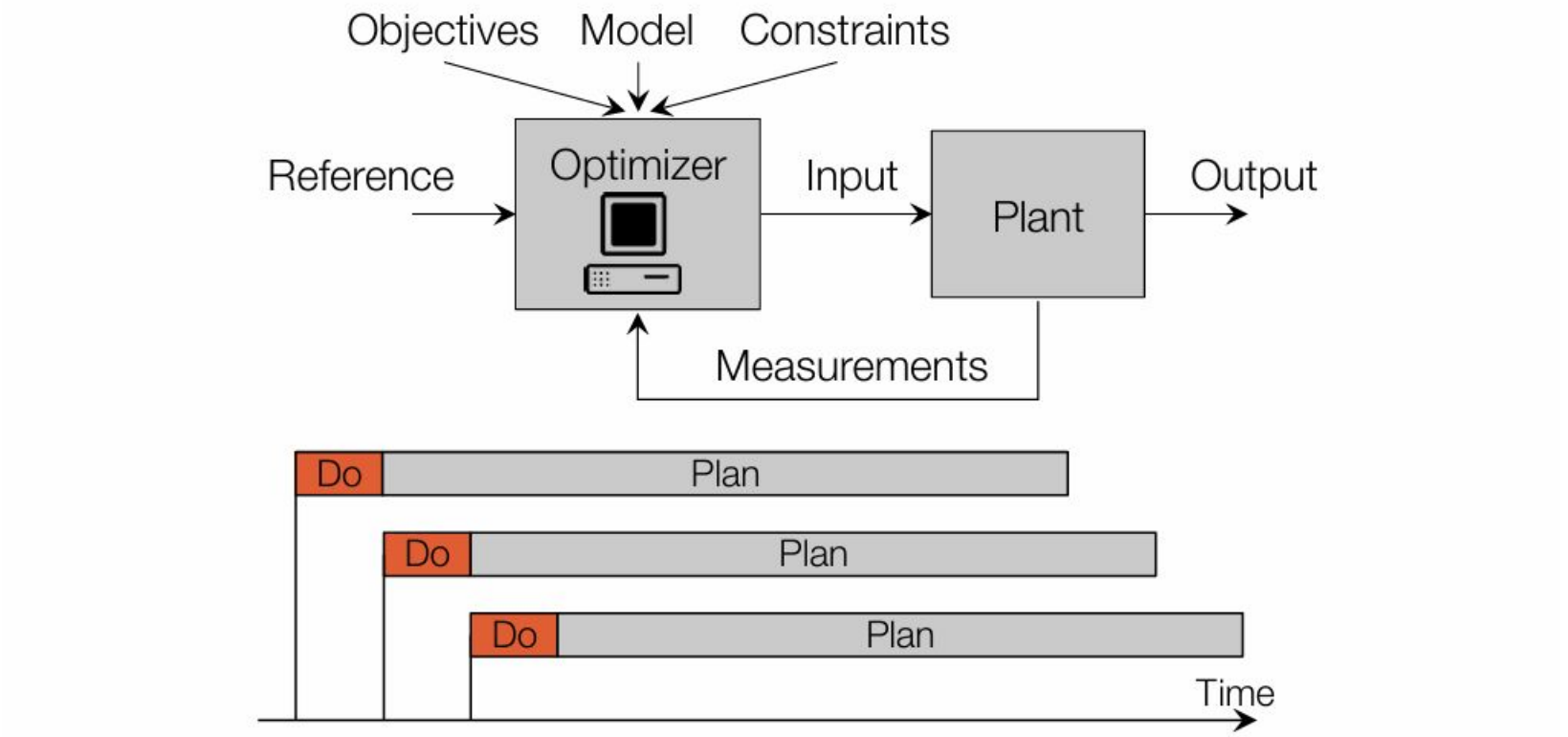


- What to do if something unexpected happens?
  - We didn't see a car around the corner!
  - Must introduce feedback



Obtain series of planned control actions
Apply first control action
Repeat the planning procedure

Model Predictive Control

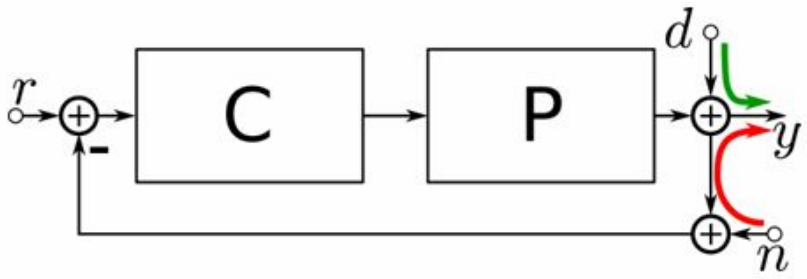


Receding horizon strategy introduces **feedback**.

Model Predictive Control

- Perspectives

Classical design: design C

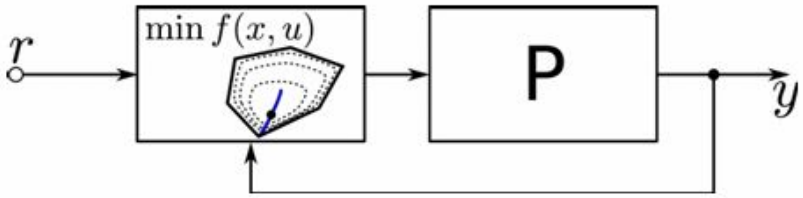


Dominant issues addressed

- Disturbance rejection $(d \rightarrow y)$
- Noise insensitivity $(n \rightarrow y)$
- Model uncertainty

(usually in *frequency domain*)

MPC: real-time, repeated optimiza tion to choose u(t)



Dominant issues addressed

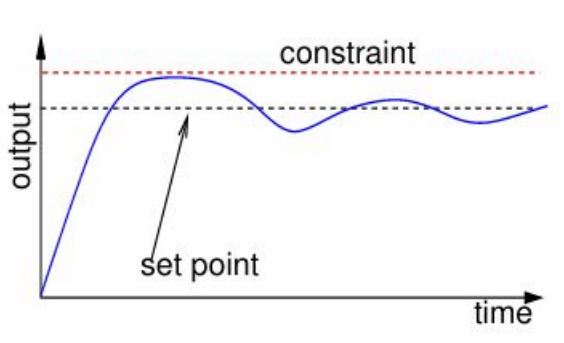
- Control constraints (limits)
- Process constraints (safety)

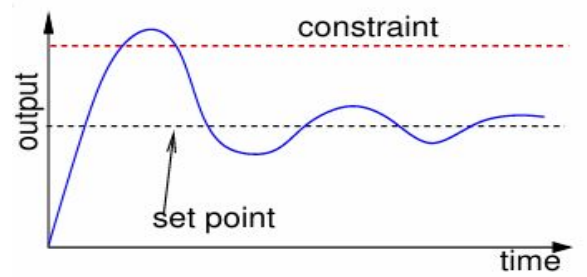
(usually in *time domain*)

Model Predictive Control

- Constraints in Control
  - All physical systems have constraints:
    - Physical constraints, e.g. actuator limits
    - Performance constraints, e.g. overshoot
    - Safety constraints, e.g. temperature/pressure limit
  - Optimal operating points are often near constraints.
  - Classical control methods:
    - Adhoc constraint management
    - Set points sufficiently far from constraints
    - Sub optimal plant operation

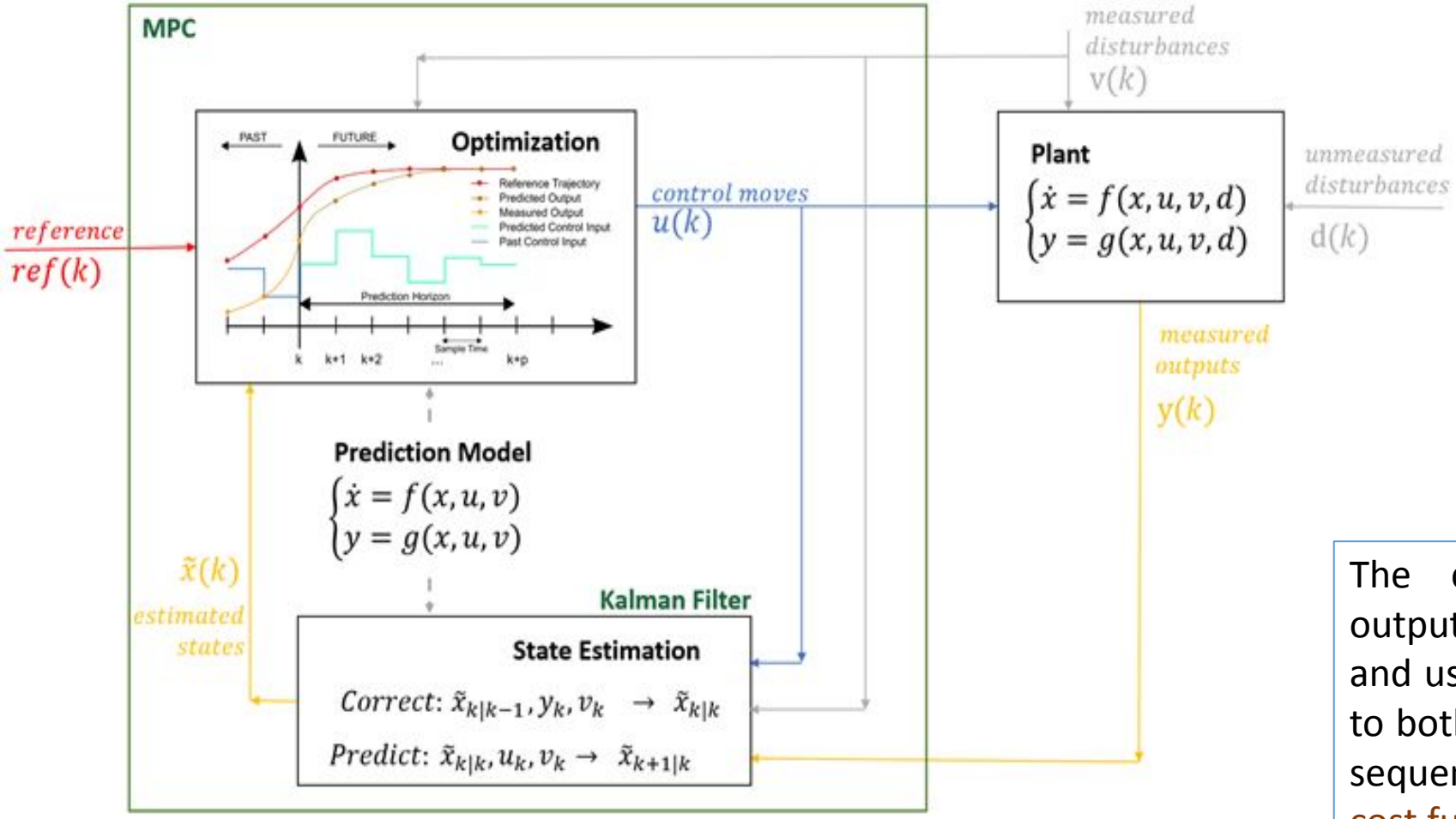
- Predictive control:
  - Constraints included in the design
  - Set point optimal
  - Optimal plant operation





Model Predictive Control

- MPC basic control loop



The controller receiving the measured outputs and disturbances from the plant, and using an internal prediction plant model to both estimate the state and to calculate a sequence of control moves that minimize a cost function over a given horizon.

Source: https://www.mathworks.com/help/mpc/gs/what-is-mpc.html

Model Predictive Control

- Mathematical Formulation

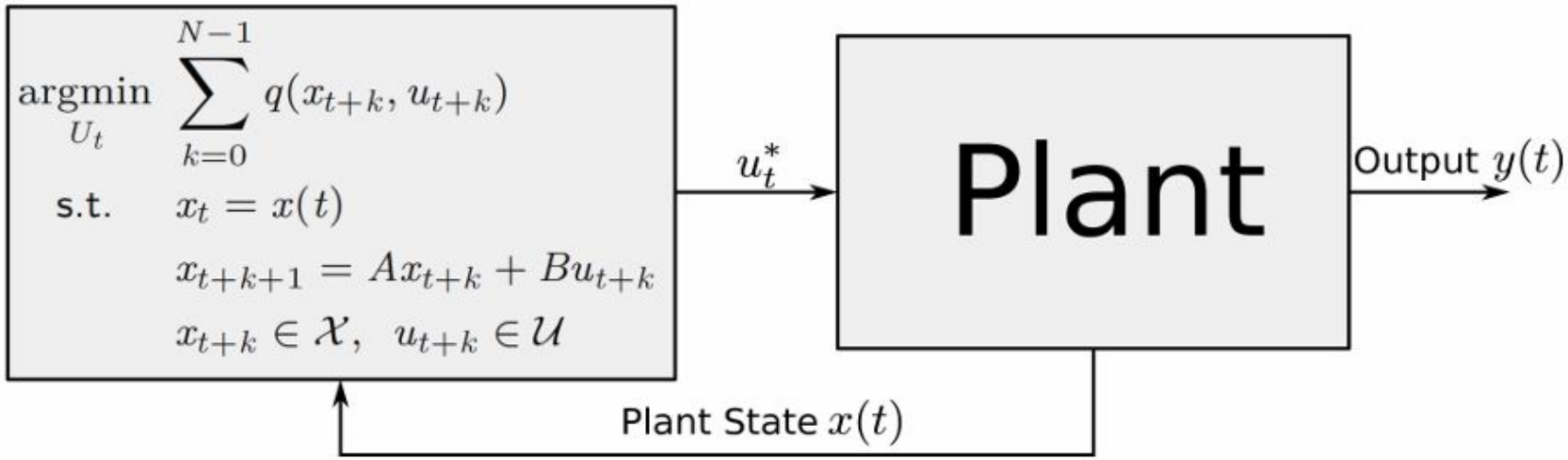
$$
\begin{aligned}
U_t^*(x(t)) := \underset{U_t}{\operatorname{argmin}} \; & \sum_{k=0}^{N-1} q(x_{t+k}, u_{t+k}) \\
\text{subj. to} \quad & x_t = x(t) && \text{measurement} \\
& x_{t+k+1} = Ax_{t+k} + Bu_{t+k} && \text{system model} \\
& x_{t+k} \in \mathcal{X} && \text{state constraints} \\
& u_{t+k} \in \mathcal{U} && \text{input constraints} \\
& U_t = \{u_0, u_1, \ldots, u_{N-1}\} && \text{optimization variables}
\end{aligned}
$$

Problem is defined by

- **Objective** that is minimized,
  e.g., distance from origin, sum of squared/absolute errors, economic,...
- Internal **system model** to predict system behavior
  e.g., linear, nonlinear, single-/multi-variable, ...
- **Constraints** that have to be satisfied
  e.g., on inputs, outputs, states, linear, quadratic,...

Model Predictive Control

Mathematical Formulation

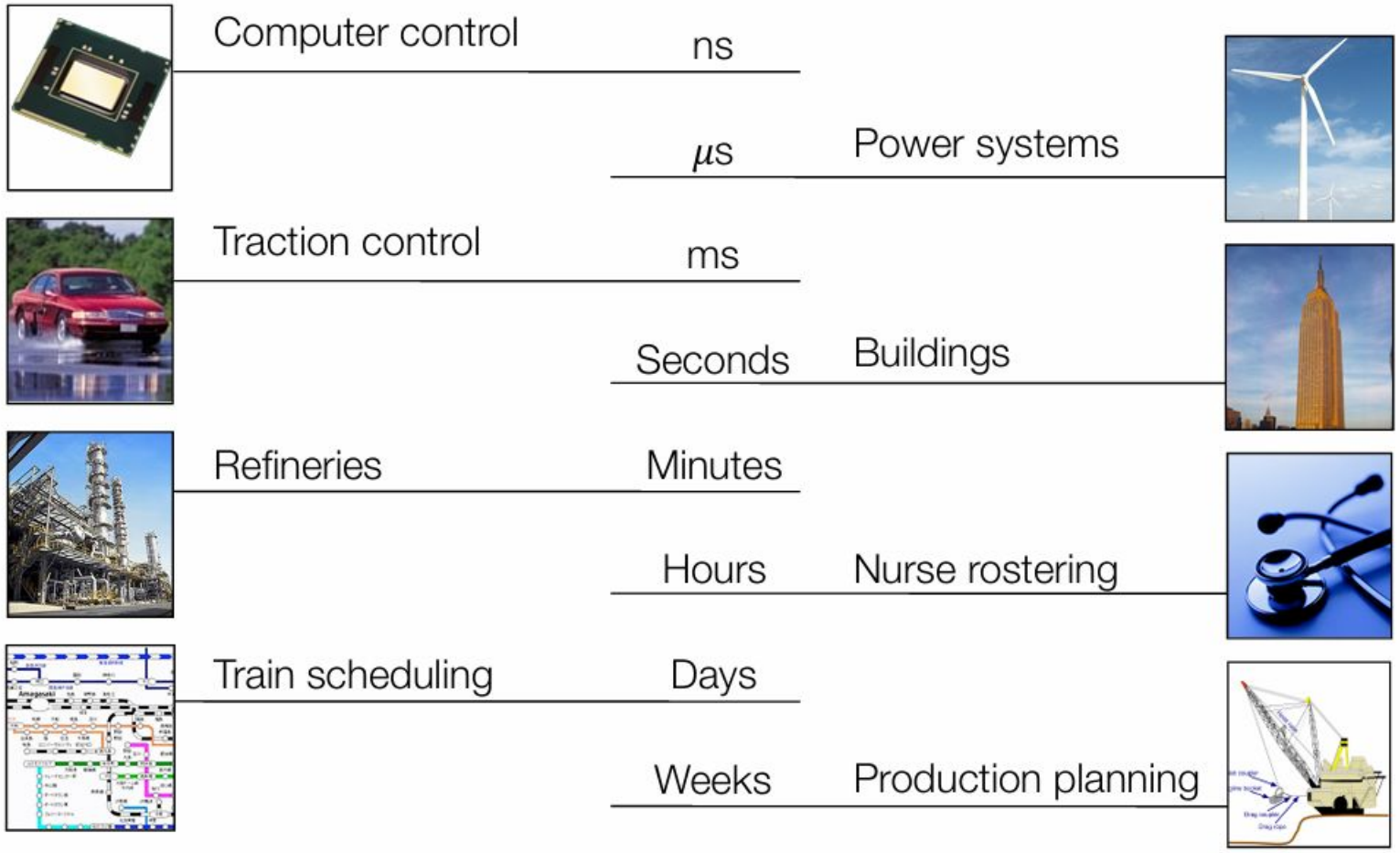


At each sample time:

- Measure / estimate current state $x(t)$
- Find the optimal input sequence for the entire planning window N: $U_t^* = \{u_t^*, u_{t+1}^*, \ldots, u_{t+N-1}^*\}$
- Implement only the *first* control action u_t^*

Model Predictive Control

- MPC: Applications



Model Predictive Control

- Advantages:
  - Systematic approach for handling constraints
  - High performance controller
- Challenges:

Implementation

- MPC problem has to be solved in real-time, i.e. within the sampling interval of the system, and with available hardware (storage, processor,...).

Stability

- Closed-loop stability, i.e. convergence, is not automatically guaranteed

Robustness

- The closed-loop system is not necessarily robust against uncertainties or disturbances

Feasibility

- Optimization problem may become infeasible at some future time step, i.e. there may not exist a plan satisfying all constraints