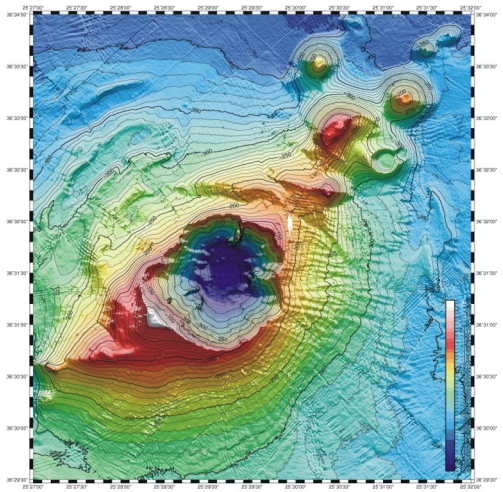
Coordinated Task and Motion Planning in the Real World



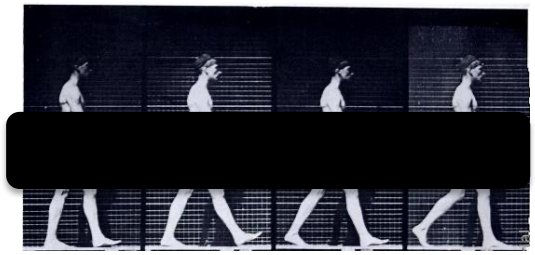
Prof. Brian Williams MIT CSAIL with help from

M. Reeves, V. Parimi, E. Fernandez, L. Blackmore, H. Ono, A. Jasour & J Chen,

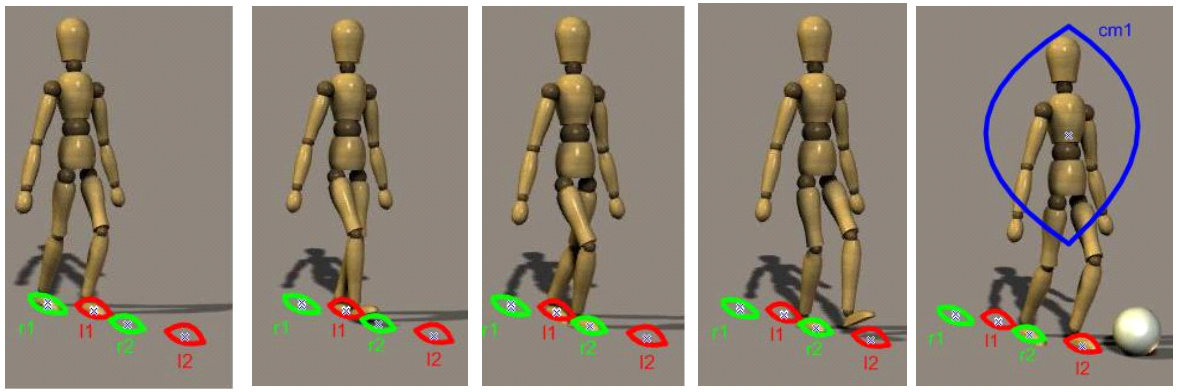
AAAI Bridge on Learning for Task and Motion Planning

Tuesday May 30th, 2025

Agents Walk According to a High-level Plan



Muybridge

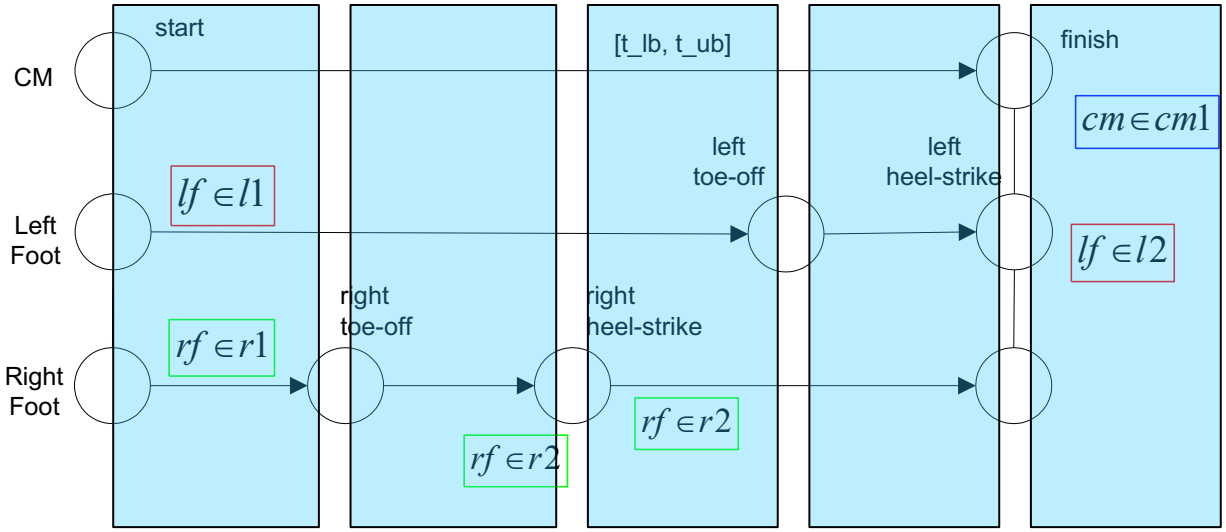


Input:

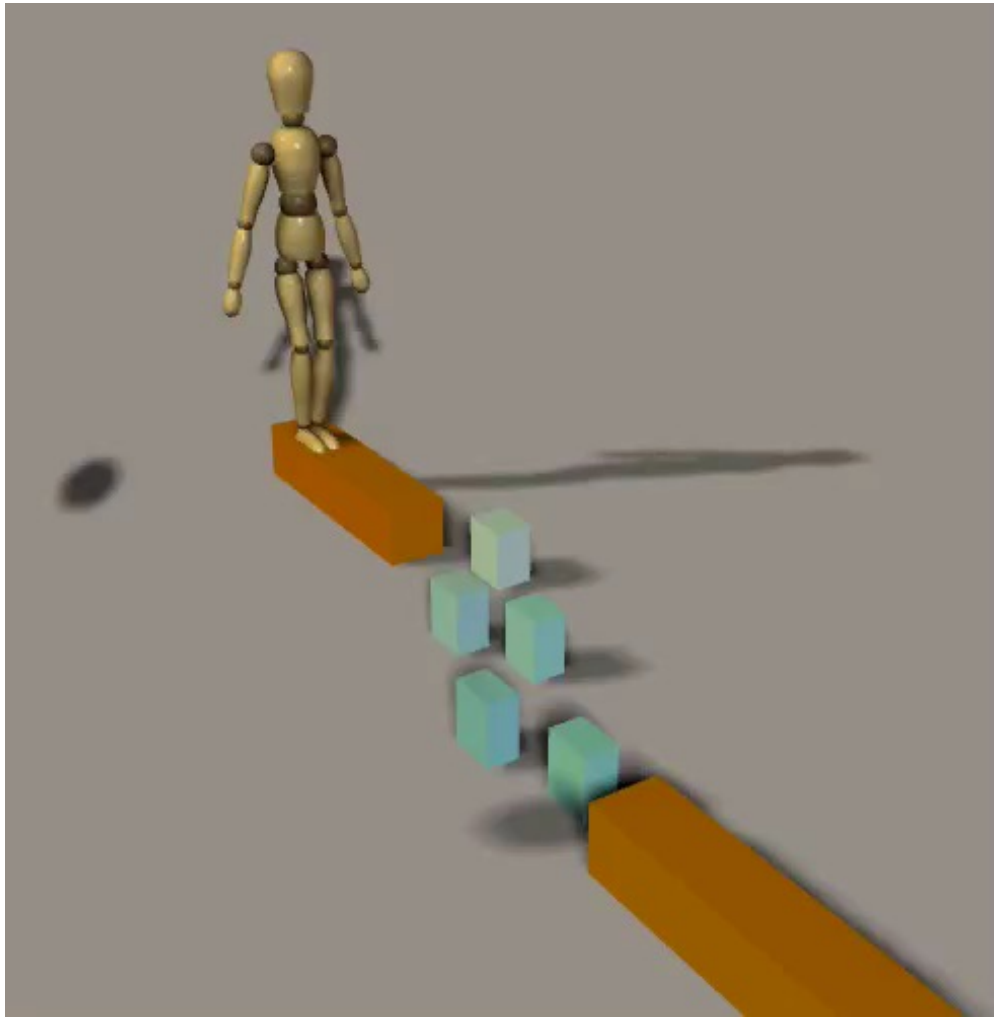
State Plan

Output:

Policy for generating motions



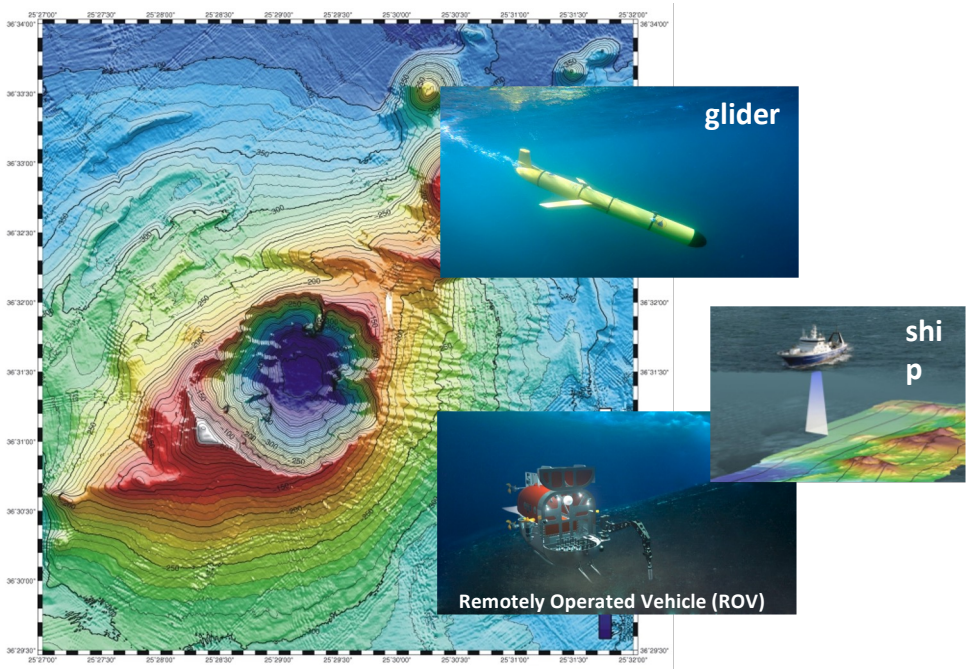
Coupling between Task and Motion is Needed for Efficiency and Robustness



Chekov Executive

Task and Motion Planning Coordinates Multiple Agents Across Complex Tasks

Emmett Planner



Scotty Planner and Magellan Executive

Key Takeaways

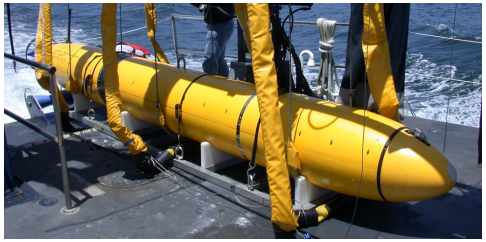
- For under-actuated robots, activities and motions couple through state and obstacles, timing and robot assignment.
- Planners should generate tasks and motions together, in light of higher-level goals.
- State Plans mediate between task and motion planning, by extracting essential constraints on state and time.
- State plans can be executed for multiple, underactuated vehicles over long horizons using convex optimization + model-predictive control.
- Enormous state spaces can be managed by combining convex optimization and combinatorial search (Appendix).
- Safety is improved with planners that maximize expected utility under bounded risk, using risk allocation.
- Safe planning with learned dynamics is enabled through risk-assessment of non-linear, non-Gaussian behaviors.

Outline

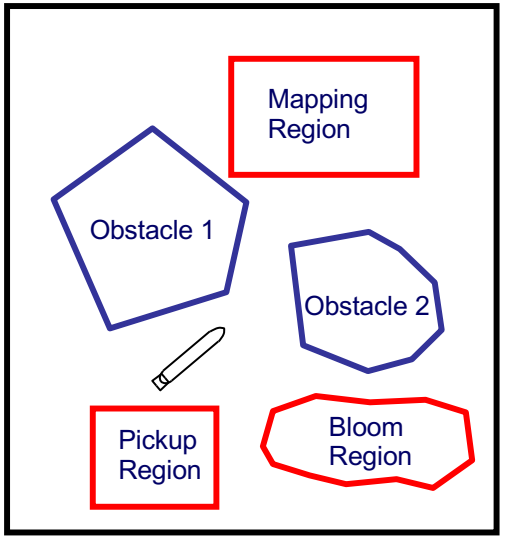
- **State Programs and State Plans**
- Model Predictive Control
- State Plan Motion Planning
- Risk-aware State Plan Motion Planning
- Risk-aware Planning with Learned Behaviors (LaPLASS)
- Avoiding Obstacles(Appendix)

Idea: Express Goals as Programs on State

Scientist: *"Explore bloom region for between 50 and 70 minutes. Afterwards, explore mapping region for between 40m and 50m. End in the pickup region. Avoid obstacles at all times. Complete the mission within 5hr."*



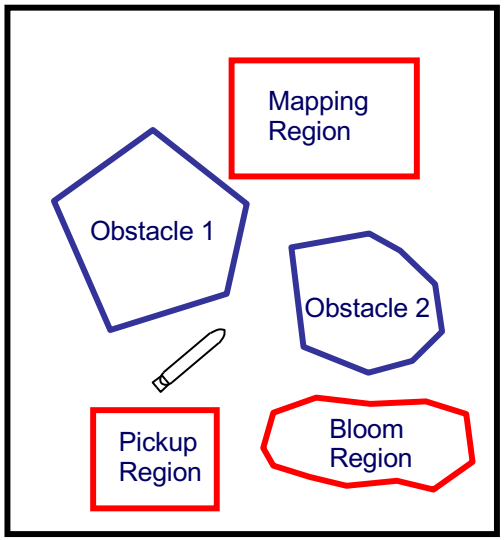
```
mapMonterreyBay(MapAUV)
[0, 5h]
{Parallel
  {Sequence
    [50m, 70m] MapUAV. remain-in(BloomRegion);
    [40m, 50m] MapUAV. remain-in(MapRegion);
    [0, infinity] MapUAV.end-in(PickUpRegion);
  },
  {Sequence
    [5m,10m] MapUAV.remain-in(SafeRegion);
  }
}
```



Leaute & Williams, AAAI 05

State Programs

Idea: State programs replace actions with constraints on state.




```
mapMonterreyBay(MapAUV)
[0, 5h]
{Parallel
   {Sequence
    [50m, 70m] MapUAV.remain-in(BloomRegion);
    [40m, 50m] MapUAV.remain-in(MapRegion);
    [0, infinity] MapUAV.end-in(PickUpRegion);
   },
   {Sequence
     [5m,10m] MapUAV.remain-in(SafeRegion);
   }
}
```

Programmer describes goals as **continuous regions** to visit **over time**.

Executive generates **paths** that **achieve goals** at their **proper time**.

Executing State Programs through Coupled Task and Motion Planning

State Program with Discrete States and Locations

```
class Main{
  UAV uav1;
  Lake lake1;
  Lake lake2;
  Fire fire1;
  Fire fire2;
 // constructor
  Main (){
      uav1 = new UAV();
      uav1.location= base_1_location;
      uav1.flying = no;
      uav1.loaded = no;

      lake1 = new Lake();
      lake1.location = lake_1_location;

      lake2 = new Lake();
      lake2.location = lake_2_location;

      fire1 = new Fire();
      fire1.location = fire_1_location;
      fire1 = high;

      fire2 = new Fire();
      fire2.location = fire_2_location;
      fire2 = high;
  }

 // "main" method
 method run() {
    sequence{
       (fire1 == out);
       (fire2 == out);
       (uav1.flying == no &&
       uav1.location == base_1_location);
    }
  }
}
```

Attributes specify state space

A program specifies the desired states

These are how UAV actions behave

```
class UAV {
  Roadmap location;
  Boolean flying;
  Boolean loaded;

  primitive method takeoff()
    flying == no => flying == yes;

  primitive method land()
    flying == yes => flying == no;

  primitive method load_water(Lake lakespot)
    ((flying == yes) && (loaded == no)
      && (lakespot.location == location)) => loaded == yes;

  primitive method drop_water_high_altitude(Fire firespot)
    ((flying == yes) && (loaded == yes)
     && (firespot.location == location) && (firespot == high))
     => ((loaded == no) && (firespot == medium));

  primitive method drop_water_low_altitude(Fire firespot)
    ((flying == yes) && (loaded == yes)
     && (firespot.location == location) && (firespot == medium))
     => ((loaded == no) && (firespot == out));

  #MOTION_PRIMITIVES(location, fly, flying==yes)
}
```

State Program with Discrete States and Locations

```
class Main{
  UAV uav1;
  Lake lake1;
  Lake lake2;
  Fire fire1;
  Fire fire2;
 // constructor
  Main (){
      uav1 = new UAV();
      uav1.location= base_1_location;
      uav1.flying = no;
      uav1.loaded = no;

      lake1 = new Lake();
      lake1.location = lake_1_location;

      lake2 = new Lake();
      lake2.location = lake_2_location;

      fire1 = new Fire();
      fire1.location = fire_1_location;
      fire1 = high;

      fire2 = new Fire();
      fire2.location = fire_2_location;
      fire2 = high;
  }

 // "main" method
 method run() {
    sequence{
       (fire1 == out);
       (fire2 == out);
       (uav1.flying == no &&
       uav1.location == base_1_location);
    }
  }
}
```

Attributes specify state space

A program specifies the desired states.

These are how UAV actions behave.

```
class UAV {
  Roadmap location;
  Boolean flying;
  Boolean loaded;

  primitive method takeoff()
    flying == no => flying == yes;

  primitive method land()
    flying == yes => flying == no;

  primitive method load_water(Lake lakespot)
    ((flying == yes) && (loaded == no)
      && (lakespot.location == location)) => loaded == yes;

  primitive method drop_water_high_altitude(Fire firespot)
    ((flying == yes) && (loaded == yes)
     && (firespot.location == location) && (firespot == high))
     => ((loaded == no) && (firespot == medium));

  primitive method drop_water_low_altitude(Fire firespot)
    ((flying == yes) && (loaded == yes)
     && (firespot.location == location) && (firespot == medium))
     => ((loaded == no) && (firespot == out));

  #MOTION_PRIMITIVES(location, fly, flying==yes)
}
```

State Program with Discrete States and Locations

```
class Main{
  UAV uav1;
  Lake lake1;
  Lake lake2;
  Fire fire1;
  Fire fire2;
 // constructor
  Main (){
      uav1 = new UAV();
      uav1.location= base_1_location;
      uav1.flying = no;
      uav1.loaded = no;

      lake1 = new Lake();
      lake1.location = lake_1_location;

      lake2 = new Lake();
      lake2.location = lake_2_location;

      fire1 = new Fire();
      fire1.location = fire_1_location;
      fire1 = high;

      fire2 = new Fire();
      fire2.location = fire_2_location;
      fire2 = high;
  }

 // "main" method
 method run() {
    sequence{
       (fire1 == out);
       (fire2 == out);
       (uav1.flying == no &&
       uav1.location == base_1_location);
    }
  }
}
```

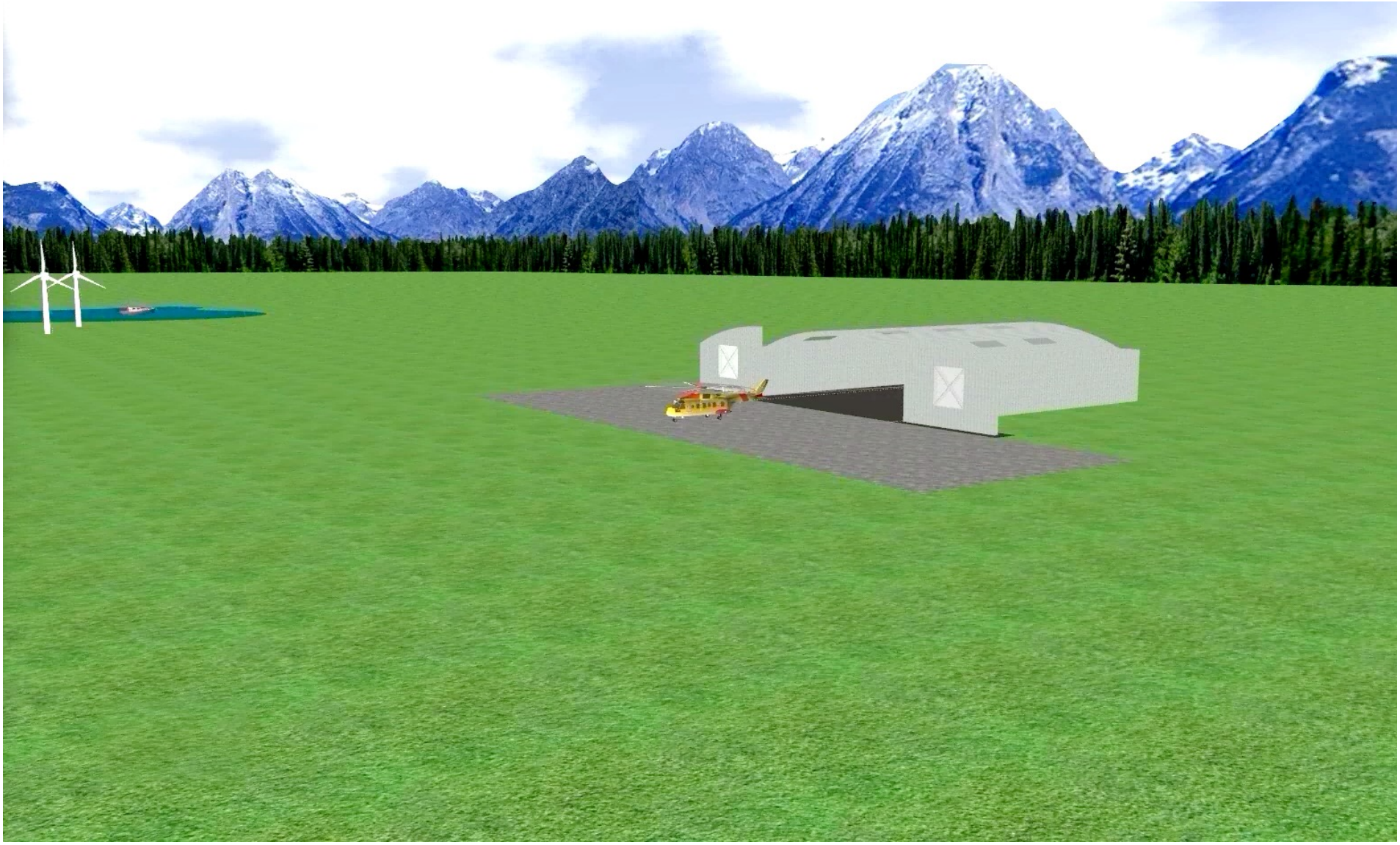
Attributes specify state space

A program specifies the desired states.

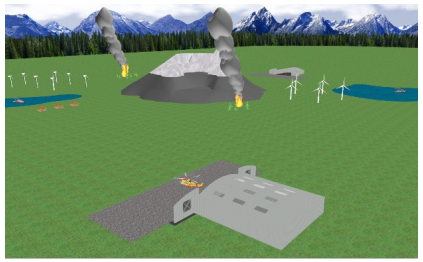
These are how UAV actions behave.

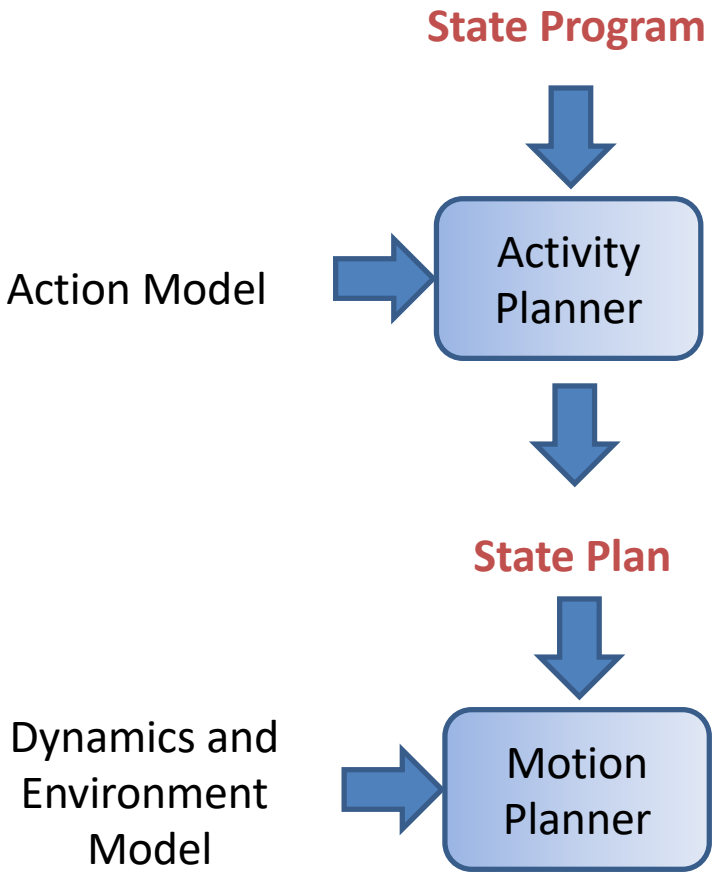
```
class UAV {
  Roadmap location;
  Boolean flying;
  Boolean loaded;

  primitive method takeoff()
    flying == no => flying == yes;

  primitive method land()
    flying == yes => flying == no;

  primitive method load_water(Lake lakespot)
    ((flying == yes) && (loaded == no)
      && (lakespot.location == location)) => loaded == yes;

  primitive method drop_water_high_altitude(Fire firespot)
    ((flying == yes) && (loaded == yes)
     && (firespot.location == location) && (firespot == high))
     => ((loaded == no) && (firespot == medium));

  primitive method drop_water_low_altitude(Fire firespot)
     ((flying == yes) && (loaded == yes)
      && (firespot.location == location) && (firespot == medium))
      => ((loaded == no) && (firespot == out));

  #MOTION_PRIMITIVES(location, fly, flying==yes)
}
```

Execution: “Put out fires and return to base”

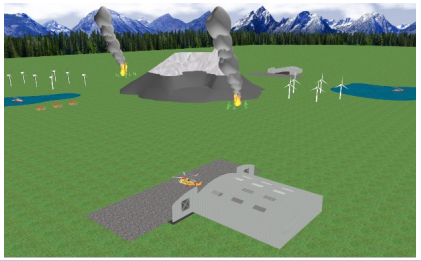


State Program Executives coordinate Activity and Motion Planning through State Plans





State Program Frames Activity Planning Problem



```
class Main{
  UAV uav1;
  Lake lake1;
  Lake lake2;
  Fire fire1;
  Fire fire2;
 // constructor
  Main (){
      uav1 = new UAV();
      uav1.location= base_1_location;
      uav1.flying = no;
      uav1.loaded = no;

      lake1 = new Lake();
      lake1.location = lake_1_location;

      lake2 = new Lake();
      lake2.location = lake_2_location;

      fire1 = new Fire();
      fire1.location = fire_1_location;
      fire1 = high;

      fire2 = new Fire();
      fire2.location = fire_2_location;
      fire2 = high;
  }

 // "main" method
 method run() {
    sequence{
       (fire1 == out);
       (fire2 == out);
       (uav1.flying == no &&
       uav1.location == base_1_location);
    }
  }
}
```

Initial State

Goal[s]

```
class UAV {
  Roadmap location;
  Boolean flying;
  Boolean loaded;

  primitive method takeoff()
    flying == no => flying == yes;

  primitive method land()
    flying == yes => flying == no;

  primitive method load_water(Lake lakespot)
    ((flying == yes) && (loaded == no)
      && (lakespot.location == location)) => loaded == yes;

  primitive method drop_water_high_altiture(Fire firespot)
    ((flying == yes) && (loaded == yes)
     && (firespot.location == location) && (firespot == high))
     => ((loaded == no) && (firespot == medium));

  primitive method drop_water_low_altiture(Fire firespot)
    ((flying == yes) && (loaded == yes)
     && (firespot.location == location) && (firespot == medium))
     => ((loaded == no) && (firespot == out));

  #MOTION_PRIMITIVES(location, fly, flying==yes)
}
```

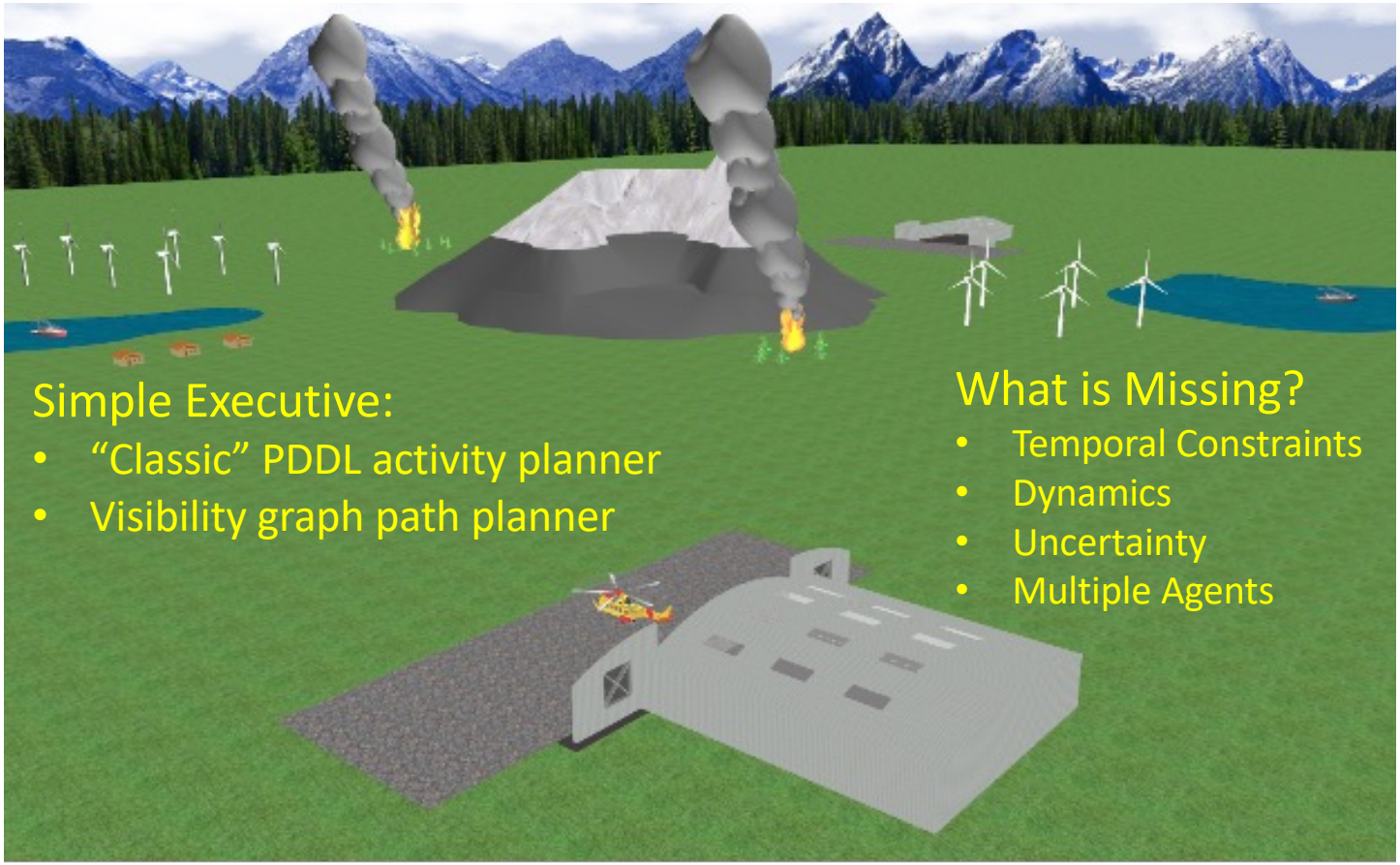
Actions (aka Operators)

Activity Planner

```
// "main" method
method run() {
   sequence{
     uav1.takeoff();
     uav1.fly(base_1_location,lake_2_location);
     uav1.load_water(lake2);
     uav1.fly(lake_2_location,fire_2_location);
     uav1.drop_water_high_altitude(fire2);
     … <13 additional activities> …
     uav1.land();
   }
  }
}
```

Plan

Execution: “Put out fires and return to base”



Today's Focus

- We assume an off-the-shelf temporal PDDL planner (LPG, Crikey, TFD, tBurton, Kirk, . . .)

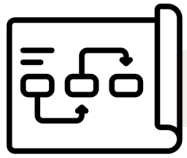
```
(:durative-action take-photoA
:duration (and (>= ?duration 15) (<= ?duration 15))
:condition (and (over all (mission-ongoing))
                (over all (uav-flying))
                (over all (inside (regionA (xb) (yb))))
                (at end (inside (regionA (xb) (yb))))
                (at start (uav-available)))
:effect (and (at start (not (uav-available)))
             (at end (uav-available))
             (at end (photo-takenA))))
```

➢ We focus on motion planning in support of activity planning.

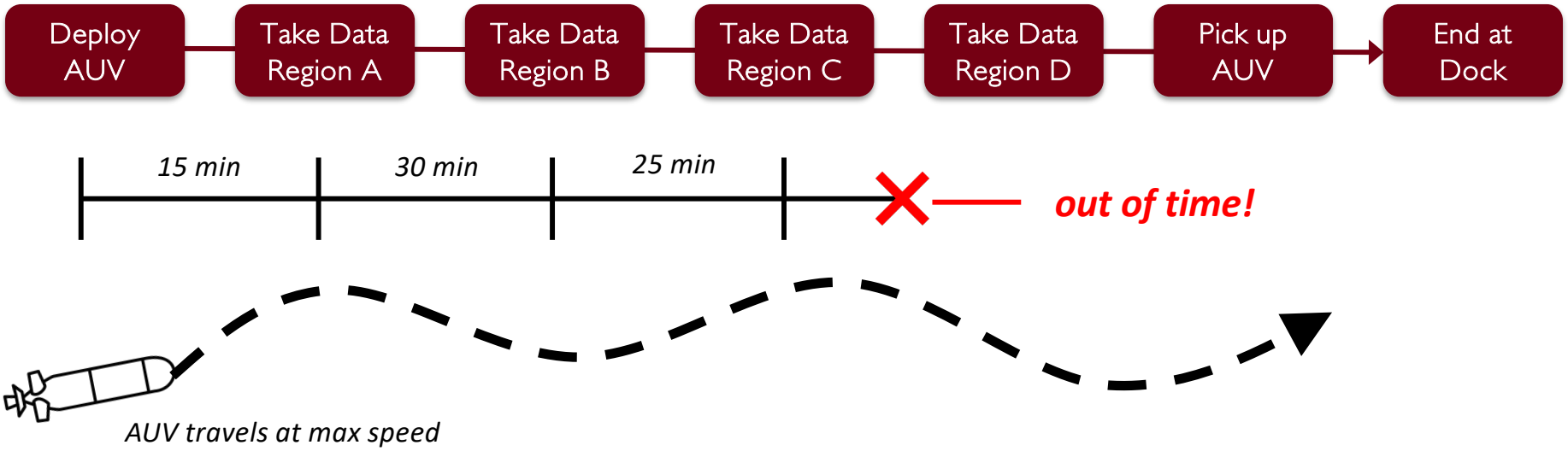
Next: How do tasks and motions couple?

When Planning Activities

Overview compliments of Marlyse Reeves
see Magellan [Reeves PhD 24]

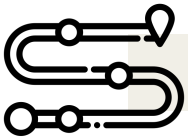


Suppose activity planner finds plan and schedule to collect data in 1 hr, and dispatches:

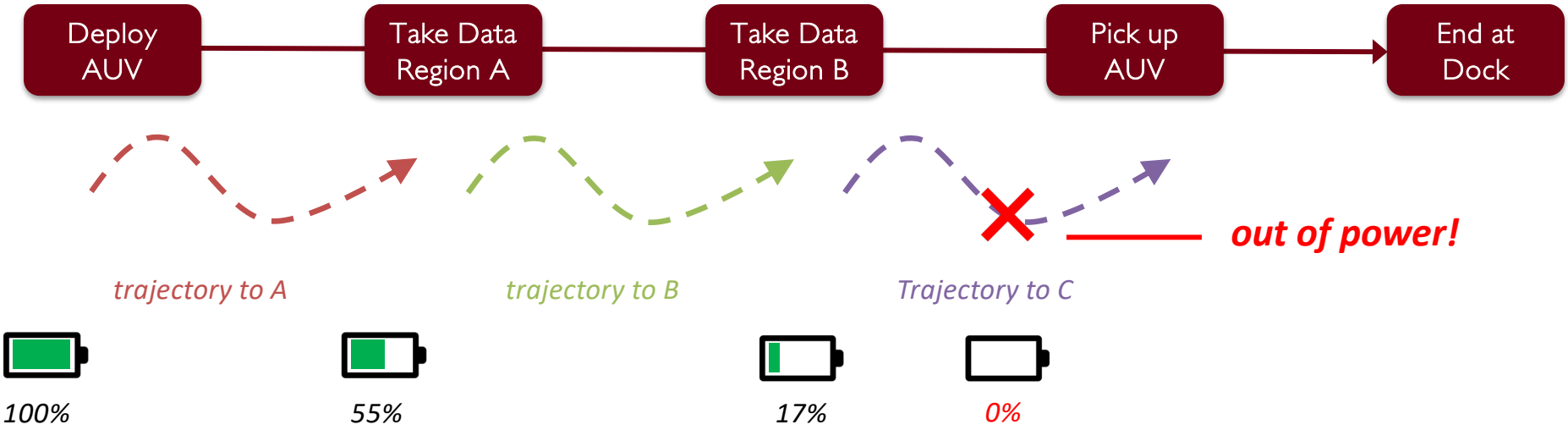


activity plan must consider dynamics & the environment

When Planning Motions



Suppose motion planner computes trajectories one activity at a time:

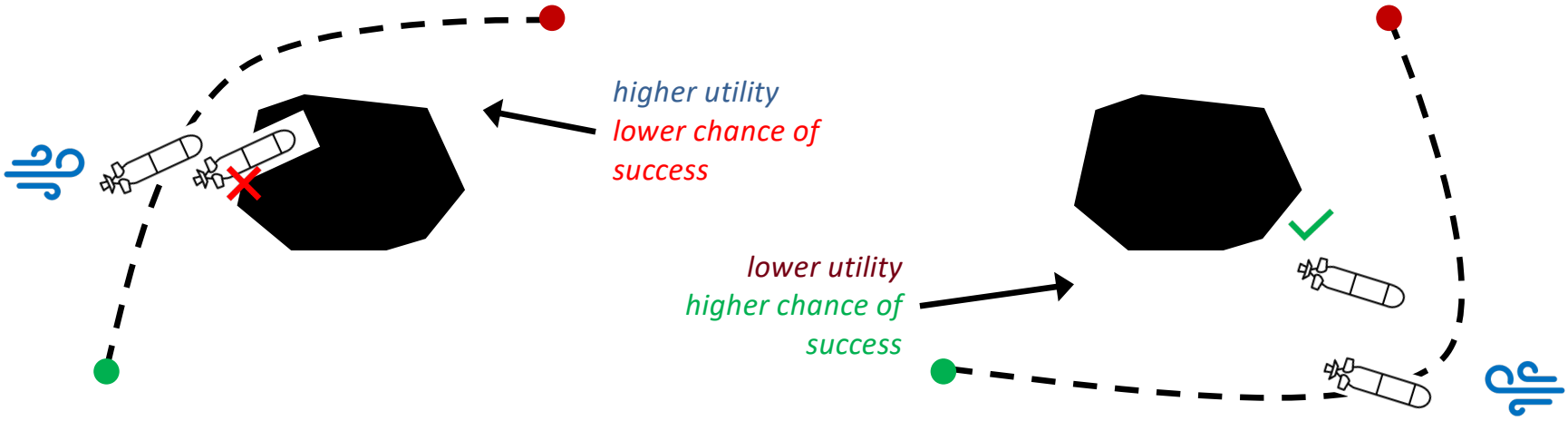


motion planner must consider global view of entire plan

21

To Stay Safe

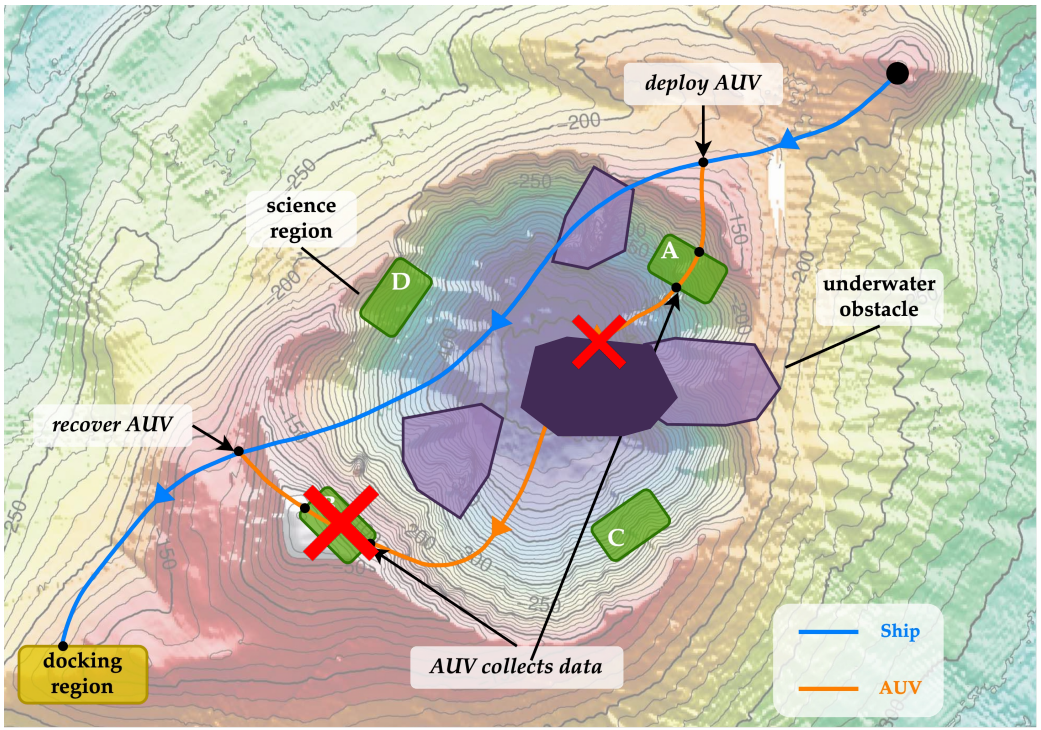
Operators care about mission success in the face of uncertainty…



plan must bound risk of failure for agents with non-linear or learned dynamics

Adapting on the Fly

Suppose we construct feasible activity and motion plans and execute …



Oh no! A drifting obstacle!

Ack! A broken sensor!

need a new plan!

system must estimate & adapt to disturbances on the fly

we refer to a system that plans and manages the execution of that plan an ***executive***

23

How Do We Communicate between Task and Motion Planning?

What the **Task Planner** Says:

Fill up the tank at the gas station;
Pickup water at the lake;
Drop water on the fire;

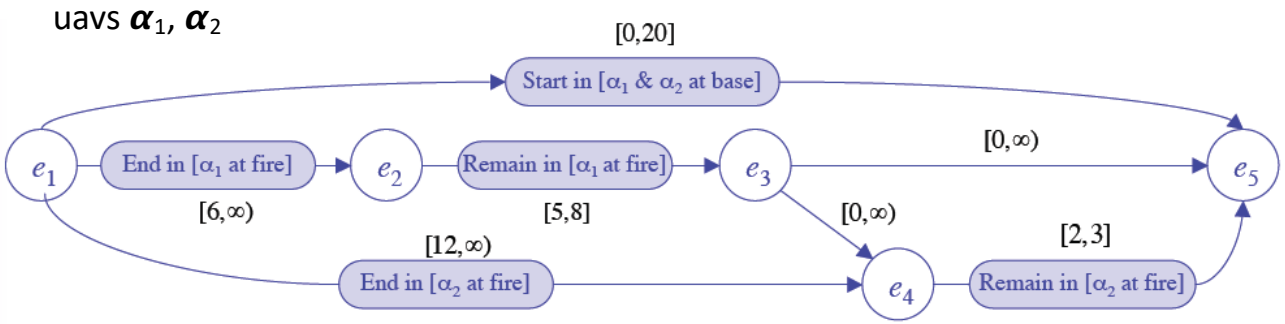
What the **Motion Planner** Should Hear:

blah blah blah at the **gas station**;
blah blah blah at the **lake**;
blah blah blah the **fire**;

State Plans specify what Motion Planners need to know about Task Plans

Temporal Plan

(generated by PDDP Planner like LPG, Crikey, TFD, Kirk, tBurton)

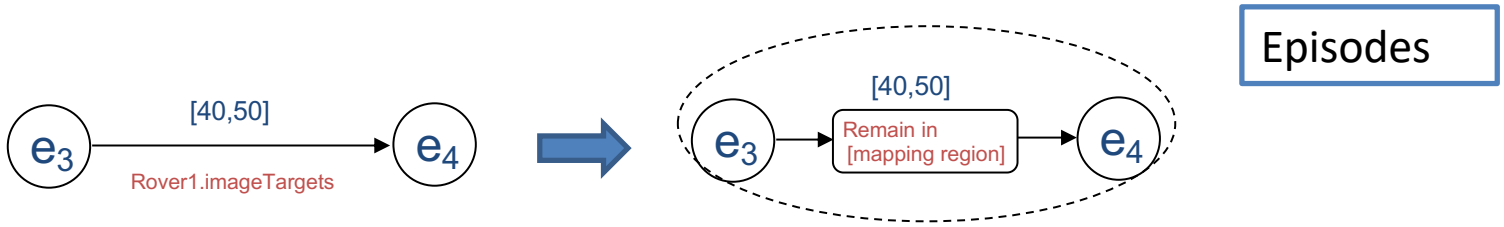


State Plan:

- episodes
- temporal constraints
- state constraints

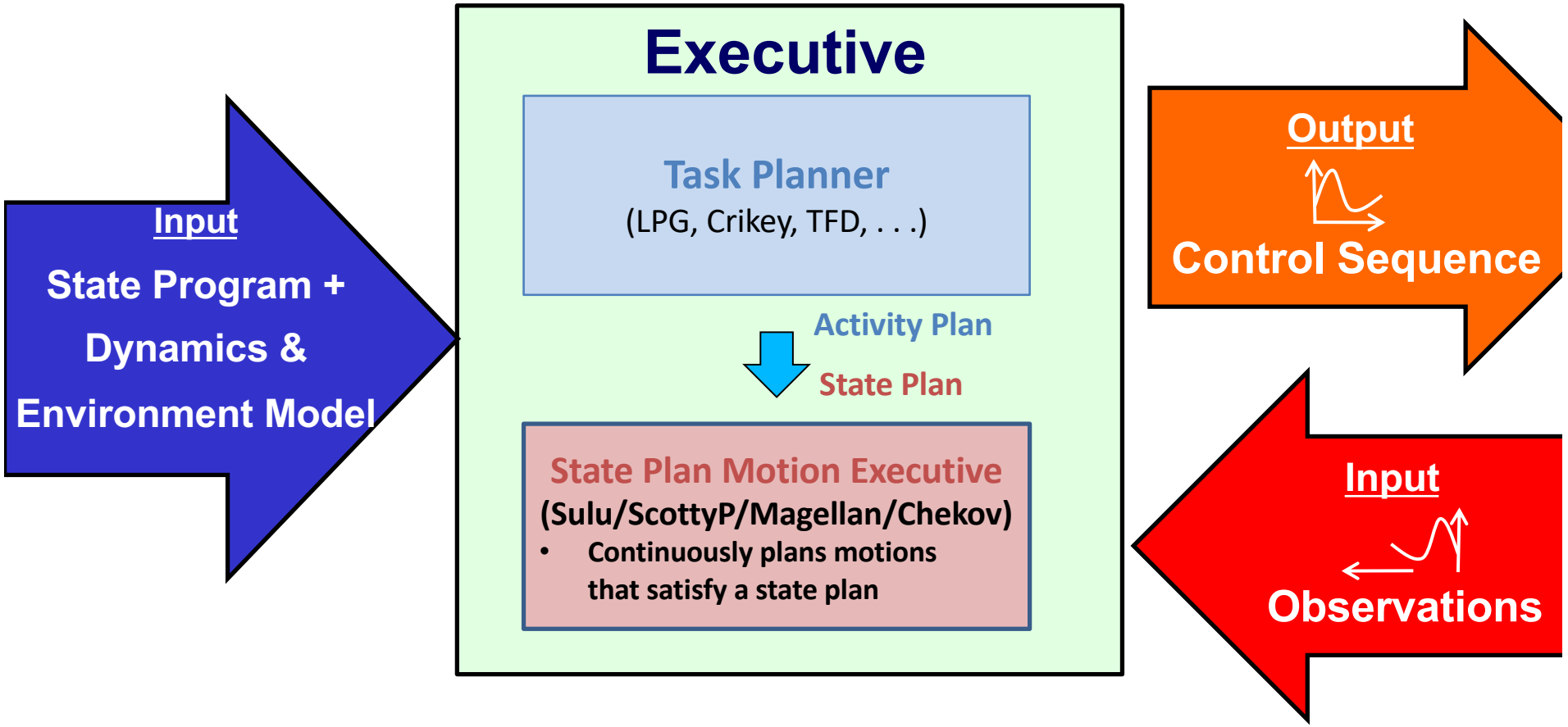
State Plan

- Episodes specify state constraints in place of actions.



- Each episode denotes possible state trajectories over a time interval that satisfy its spatial and temporal constraints.

- A state plan offers spatial and temporal flexibility, which a motion executive uses to handle disturbances.



1. Task Planner maps state program to activity plan.
2. Motion Executive abstracts activity plan to state plan.
3. Motion Executive generate motions that satisfy the state plan.

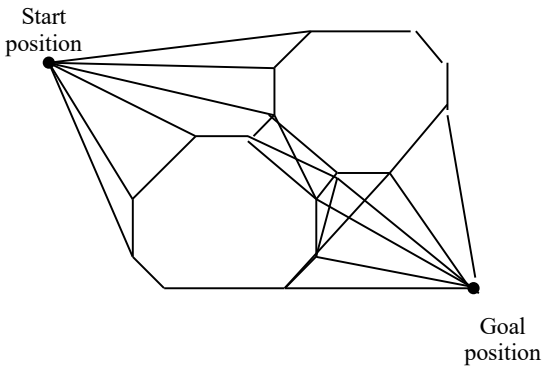
See Appendix for monolithic "hybrid" planners (Kongming, cKongming)

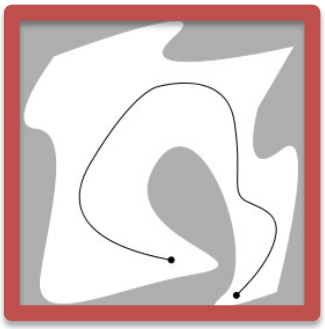
Generating Motions that Satisfy State Plan

Graph Search

Optimization

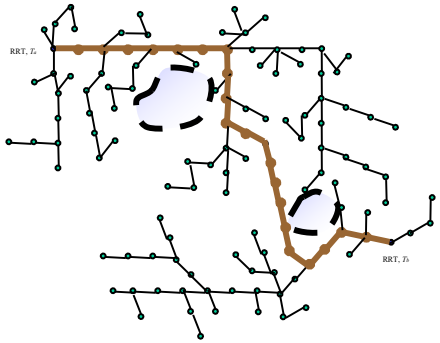
Exact

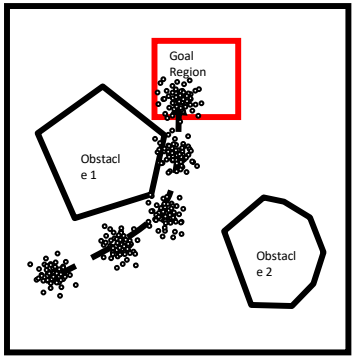




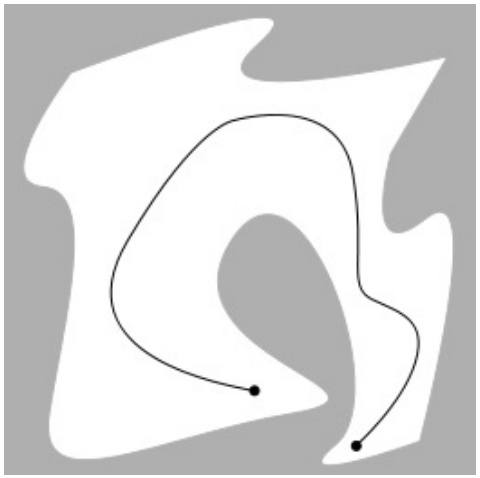
Probabilistic
Risk-bounded

Sample-based





Generating Continuous Motions that Satisfy a State Plan



Optimization

Approach 1: Solve online using mathematical programming.

[Schouwenaars, Moor, Feron & How, ECC 01]
[Leaute & Williams, AAAI 05]
[Fernandez, Karpas & Williams IJCAI 15]
[Reeves SM 20]

Approach 2: Precompute policy by combining dynamic scheduling and reach-set analysis.

[Hofmann & Williams, AAAI 06] see Appendix

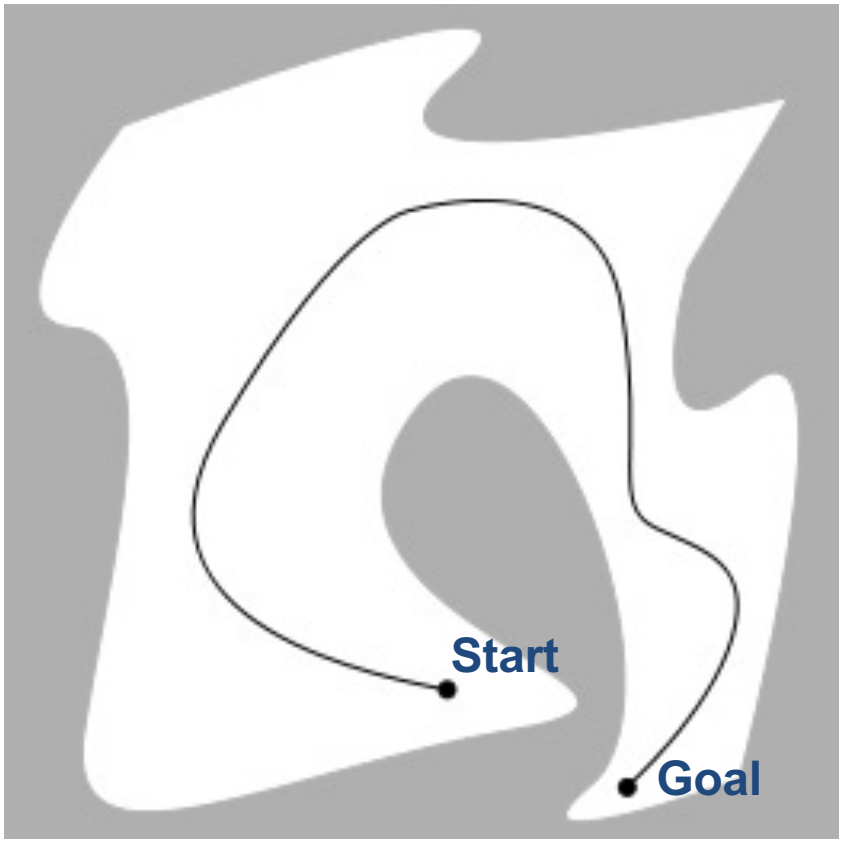
Approach 3: Factor multi-agent planning (MAPF) using prioritized search.

[Zhang, Chen, Li, Williams & Koenig, AAMAS 22]

Outline

- State Programs and State Plans
- **Model Predictive Control**
  - **Optimization**
  - **Online Adaptation**
- State Plan Motion Planning
- Risk-aware State Plan Motion Planning
- Risk-aware Planning with Learned Behaviors (LaPLASS)
- Avoiding Obstacles(Appendix)

Framing Path Planning as Trajectory Optimization



$$\min_{p} J(p)$$

$$s.t.$$

$$p \in P$$

p: path
P: Set of feasible paths
J: path cost

- Formulate as Linear (LP), Non-linear (NP), Convex (CP), Mixed Integer (MILP) or Mixed-Logic (MLLP) Program.

Finite Horizon Trajectory Optimization

- Example: Formulation as a Linear Program.

$$\min_{\mathbf{x}_{1:N},\mathbf{u}_{1:N}} J(\mathbf{x}_1 \cdots \mathbf{x}_N, \mathbf{u}_1 \cdots \mathbf{u}_N)$$

Cost function

$$s.t.$$

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \quad (k = 0,1,\cdots N-1)$$

Dynamics

$$\mathbf{H}\mathbf{x}_\mathbf{k} \le \mathbf{g} \quad (k = 0,1,\cdots N)$$

Spatial constraints

$$\mathbf{x}_0 = \mathbf{x}_{\text{start}}$$

Initial position and velocity

$$\mathbf{x}_N = \mathbf{x}_{\text{goal}}$$

Goal position and velocity

$$-\mathbf{u}_{\max} \le \mathbf{u}_k \le \mathbf{u}_{\max} \quad (k = 0,1,\cdots N-1)$$

Actuation limits

$$\mathbf{x}_k \equiv \begin{pmatrix} x_k & y_k & \dot{x}_k & \dot{y}_k \end{pmatrix}^T, \quad \mathbf{u}_k \equiv \begin{pmatrix} F_{x,k} & F_{y,k} \end{pmatrix}^T$$

Example Formulation

What are the . . .

- Control and State Variables?
- Costs?
- Constraints?

Example Formulation

Model accurate dynamics over a finite planning horizon.

$$\boldsymbol{s_i} = \begin{bmatrix} \boldsymbol{x_i} \\ \boldsymbol{u_i} \end{bmatrix} \qquad \boldsymbol{x_i} = \begin{bmatrix} \boldsymbol{x_i} \\ \boldsymbol{y_i} \\ \boldsymbol{v_{x,i}} \\ \boldsymbol{v_{y,i}} \\ \boldsymbol{b_i} \end{bmatrix} \qquad \boldsymbol{u_i} = \begin{bmatrix} \boldsymbol{a_{x,i}} \\ \boldsymbol{a_{y,i}} \end{bmatrix}$$

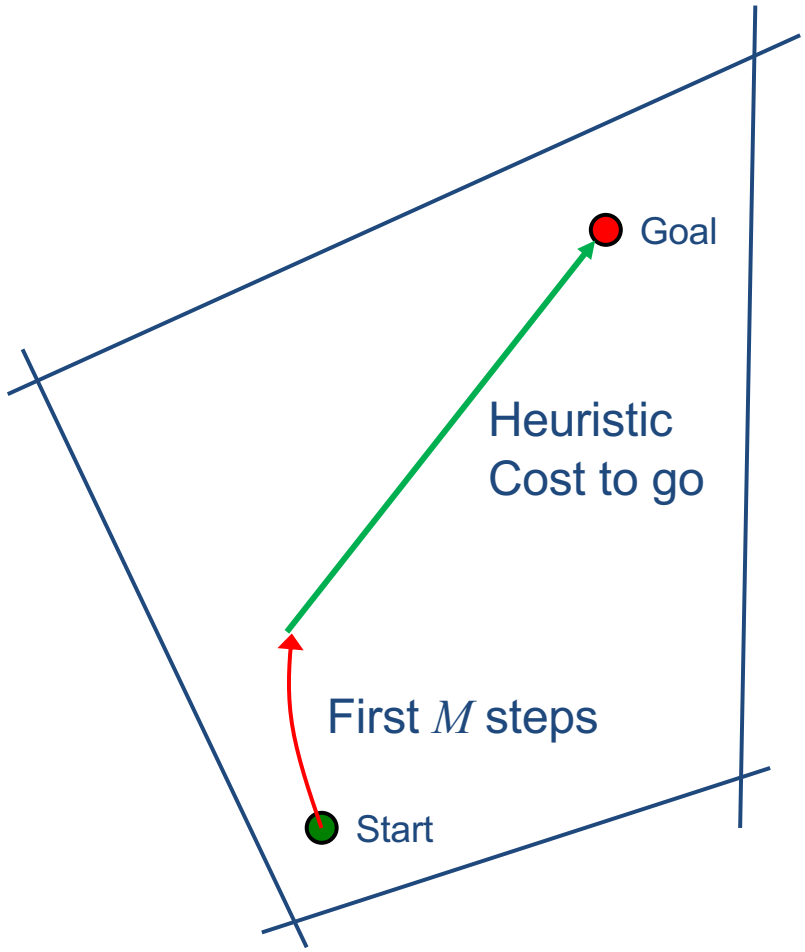
Using a fixed time step

$$T_P = n \cdot \Delta t \qquad for\ i = 0, \dots, n$$

	General Form	Examples	
quadratic objective	$\frac{1}{2}\boldsymbol{s}^T Q\, \boldsymbol{s} + \boldsymbol{p}^T \boldsymbol{s}$	$\frac{1}{2}\sum_{i=1}^{n} \boldsymbol{u}_i^2$	***minimize control effort*** *for the agent*
Linear equality constraints	$\boldsymbol{Fs} = \boldsymbol{g}$	$x_{i+1} = x_i + v_{i,x} \cdot \Delta t + \frac{\Delta t^2}{2} \cdot a_{i,x}$ $b_{i+1} = b_i - c \cdot v_{i,x}$ $\dots$ $x_n = x_{goal}$	***2nd order dynamics*** *& goal*
inequality constraints	$\boldsymbol{Fs} \leq \boldsymbol{g}$	$b_i \geq 0$	***battery*** *constraint*

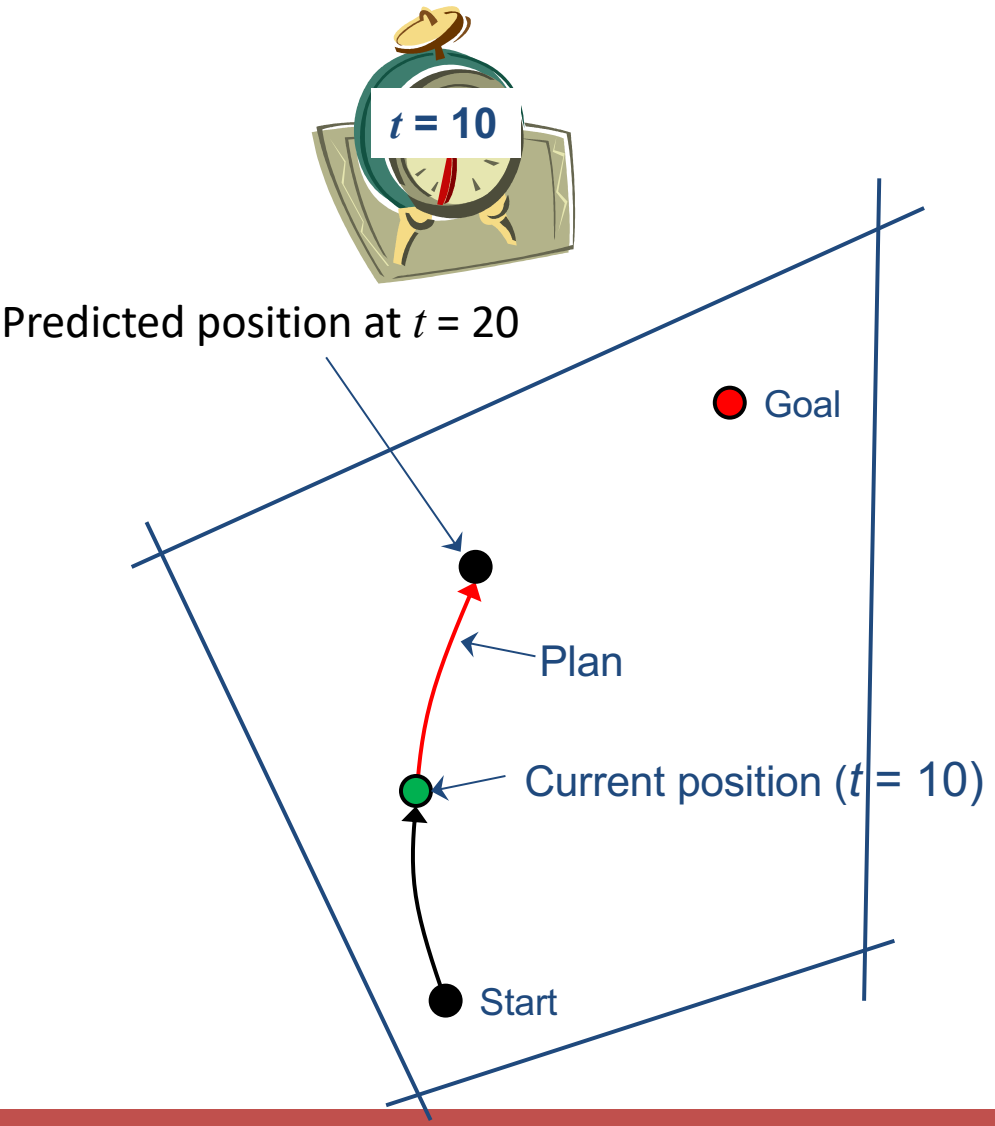
How Do We Execute Over Longer Horizons?

- In parts.



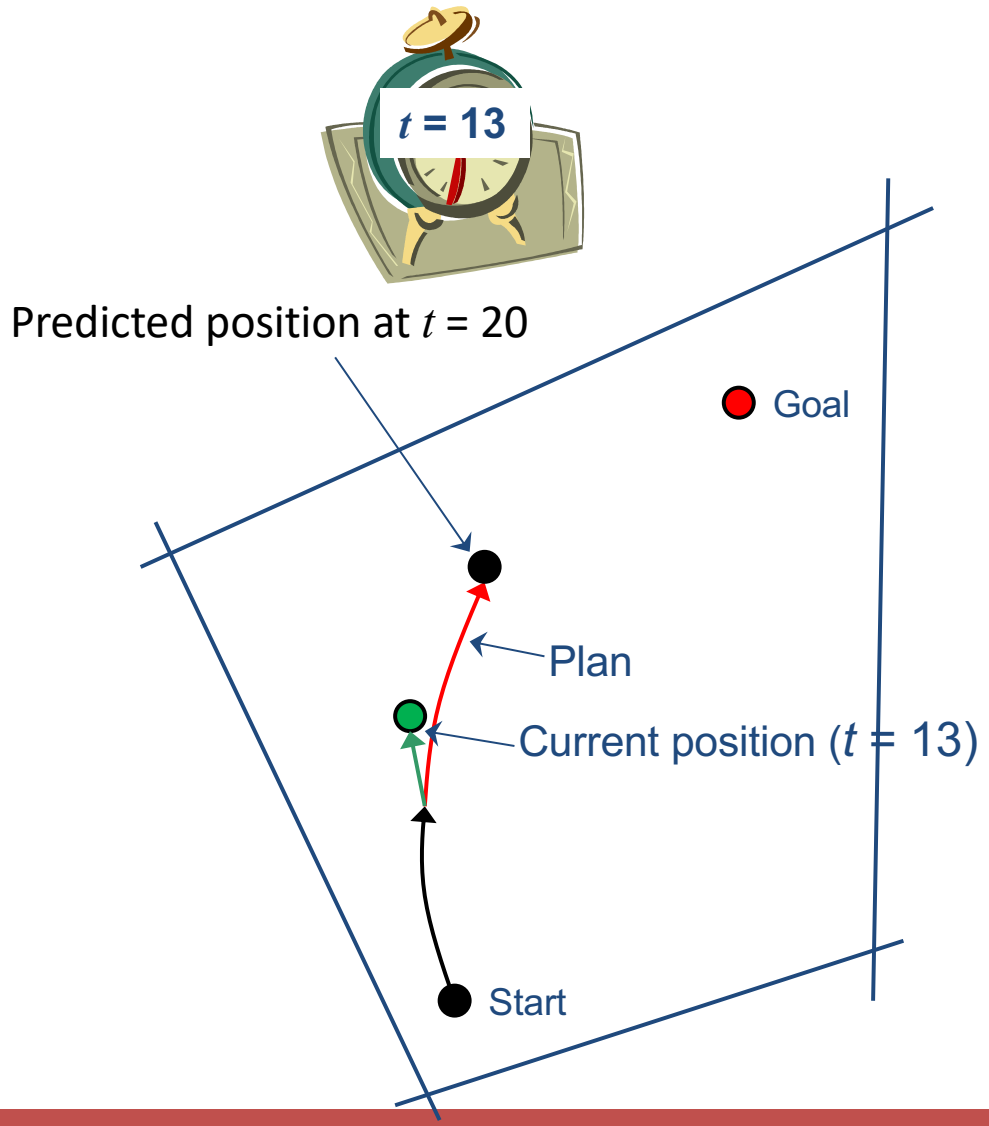
How Do We Execute Over Longer Horizons?

- Start executing plan for current position (t=10)



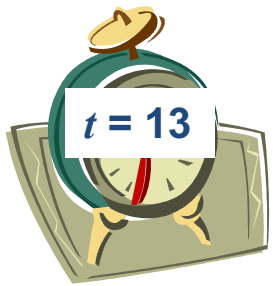
How Do We Execute Over Longer Horizons?

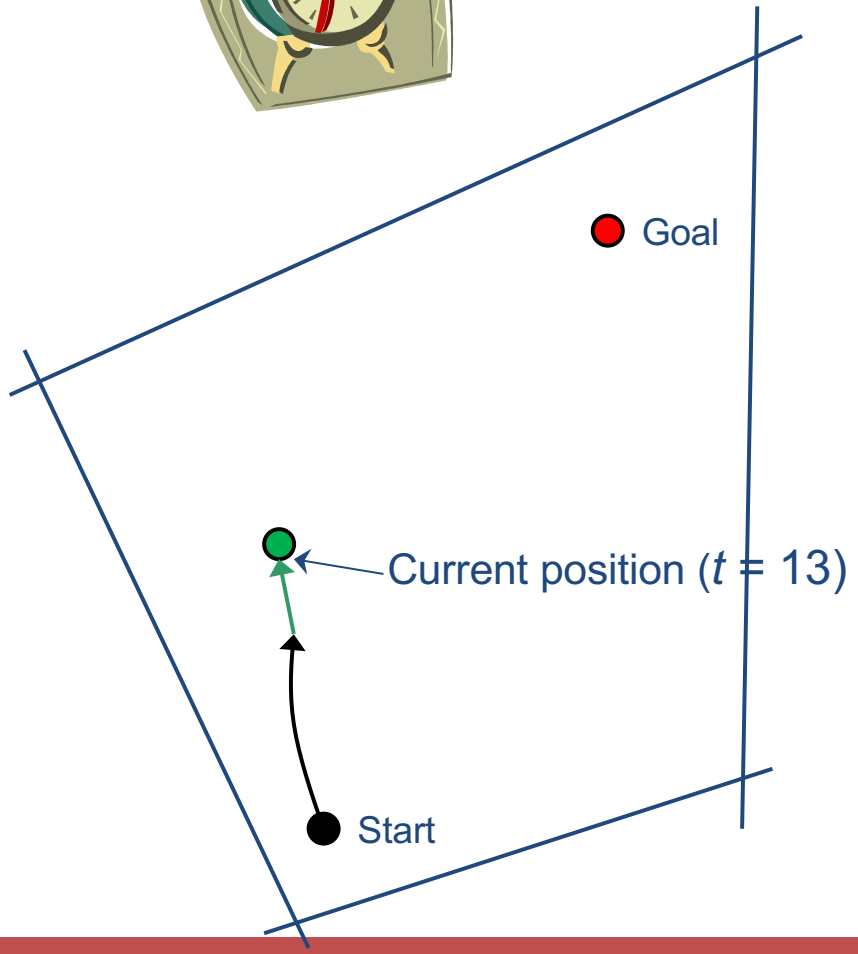
- After 3 seconds (t=13) only off a little



How Do We Execute Over Longer Horizons?

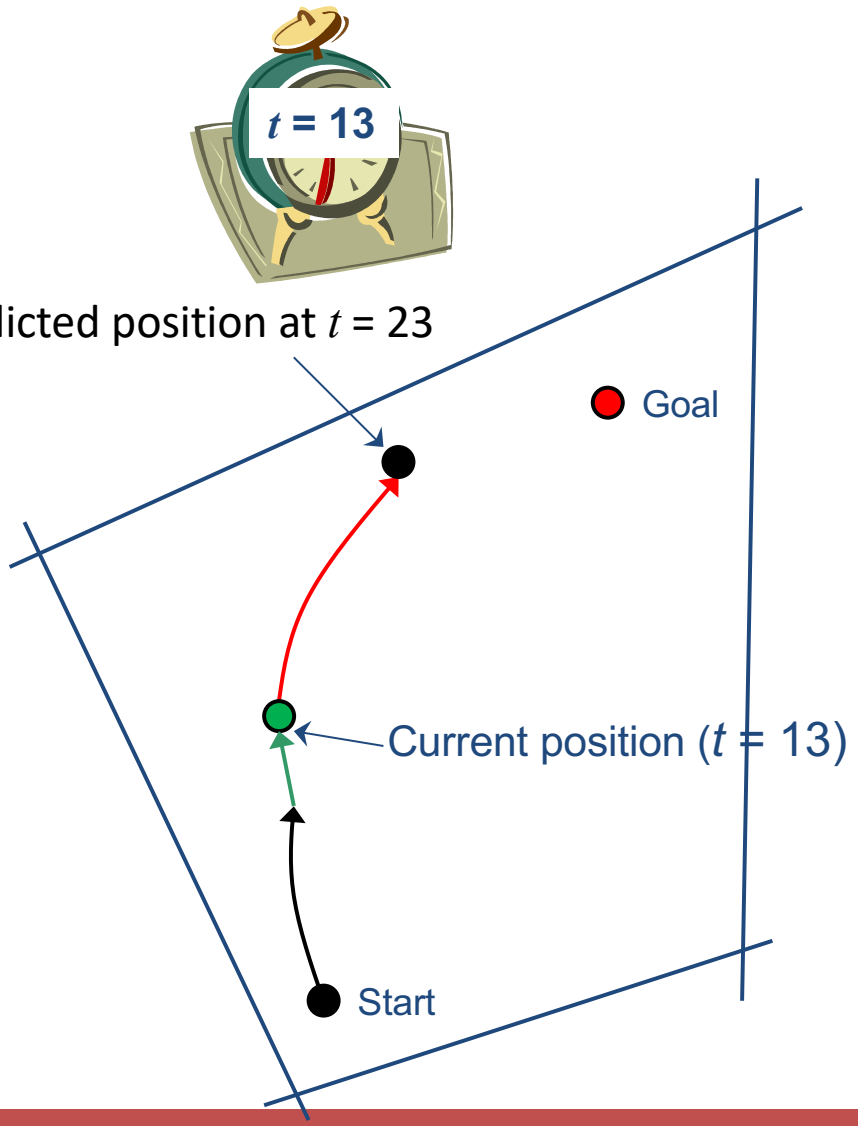
- Abandon plan at t=13





also called Model-Predictive Control

- Abandon plan at t=13
- Replan for new horizon and replace.
- Repeat.



Planner Formulation for Model-Predictive Control

$$\min_{\mathbf{x}_{1:N},\mathbf{u}_{1:N}} J(\mathbf{x}_1 \cdots \mathbf{x}_N, \mathbf{u}_1 \cdots \mathbf{u}_N) + f(\mathbf{x}_N)$$

Cost function

Cost-to-go function

$$s.t.$$

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \quad (k = 0,1,\cdots N-1)$$

Dynamics

$$\mathbf{H}\mathbf{x}_k \leq \mathbf{g} \quad (k = 0,1,\cdots N)$$

Spatial constraints

$$\mathbf{x}_0 = \mathbf{x}_{\text{start}}$$

Initial position and velocity

$$\mathbf{x}_N = \mathbf{x}_{\text{goal}}$$

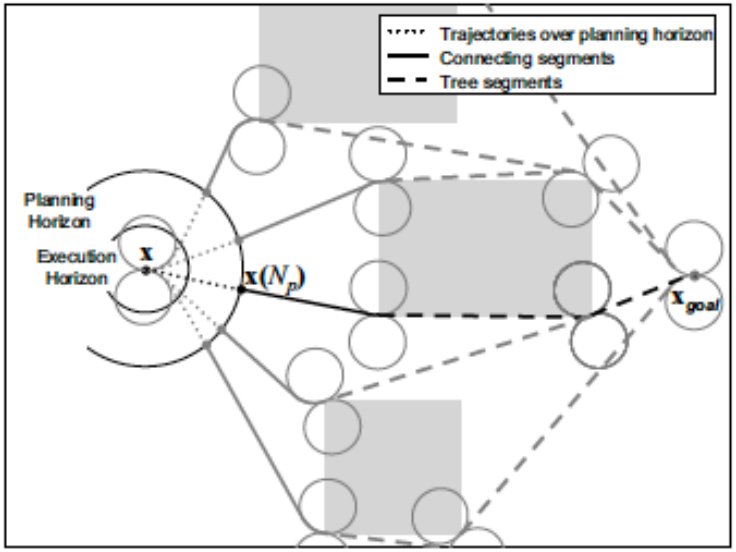
Goal position and velocity

$$-\mathbf{u}_{\max} \leq \mathbf{u}_k \leq \mathbf{u}_{\max} \quad (k = 0,1,\cdots N-1)$$

Thrust limits

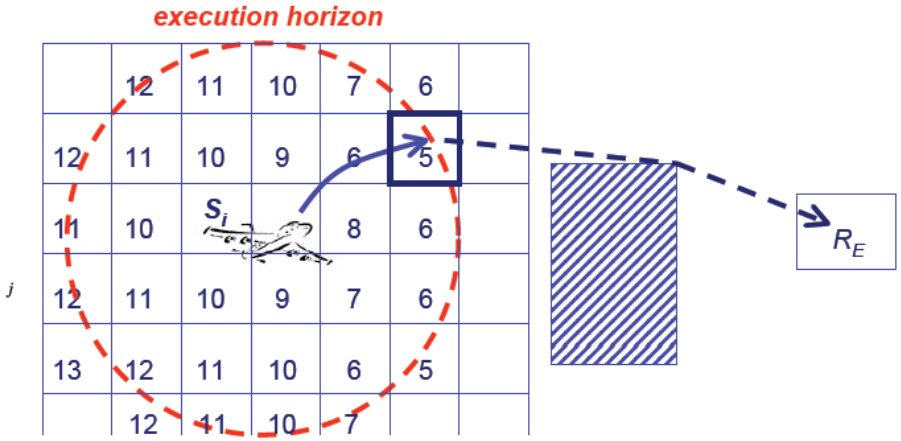
$$\mathbf{x}_k \equiv \begin{pmatrix} x_k & y_k & \dot{x}_k & \dot{y}_k \end{pmatrix}^T, \quad \mathbf{u}_k \equiv \begin{pmatrix} F_{x,k} & F_{y,k} \end{pmatrix}^T$$

Example Heuristic Function



1. Construct visibility graph based on obstacles and goal region.

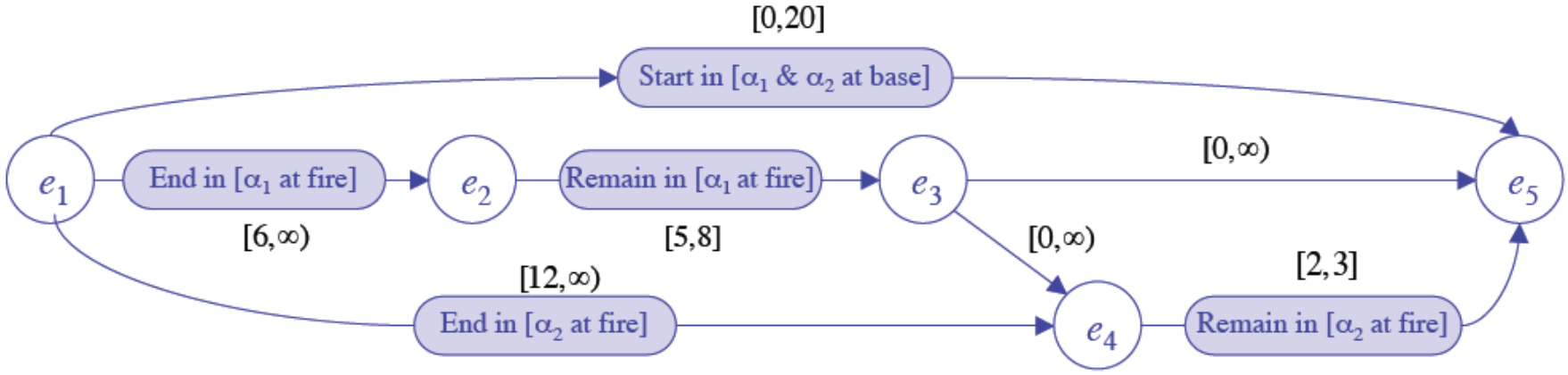
2. Guide trajectory toward the point ”closest” to the goal region.

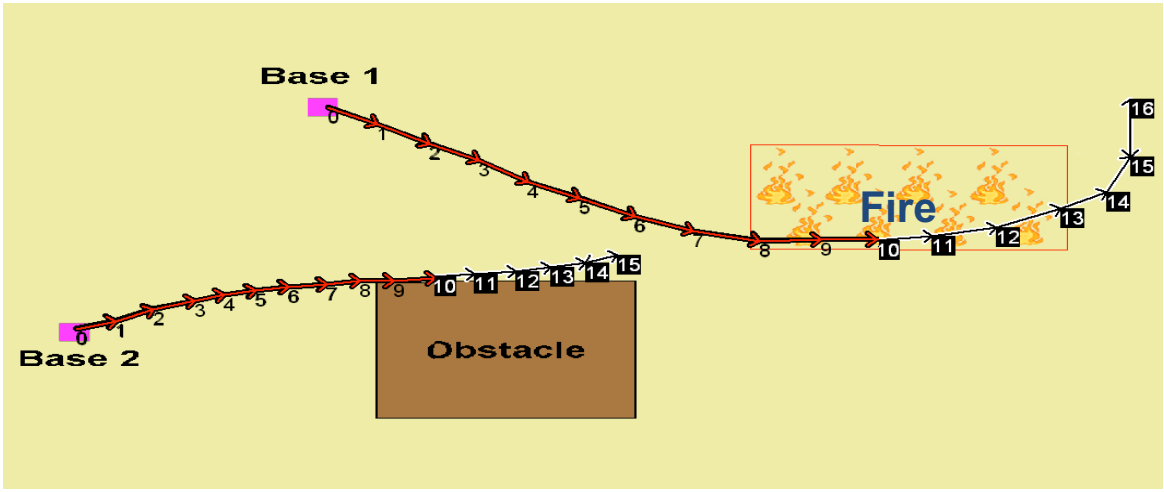


Outline

- State Programs and State Plans
- Model Predictive Control
- State Plan Motion Planning
  - **Sulu: A Discrete Time Formulation (convex)**
  - Scotty: A Continuous Time Formulation (convex)
  - Magellan: dynamic execution over long time horizons
- Risk-aware State Plan Motion Planning
- Risk-aware Planning with Learned Behaviors (LaPLASS)
- Avoiding Obstacles(Appendix)

Idea: Encode State Plan as Constraints in a Trajectory Optimization





•Thomas Léauté, Brian Williams, "Coordinating Agile Systems Through the Model-based Execution of Temporal Plans," *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05),* Pittsburgh, PA, July 2005, pp. 114-120.

Start with Discrete-Time Trajectory Optimization

$$\min_{\mathbf{x}_{1:N},\mathbf{u}_{1:N}} J(\mathbf{x}_1 \cdots \mathbf{x}_N, \mathbf{u}_1 \cdots \mathbf{u}_N)$$

Cost function

$$s.t.$$

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \quad (k = 0,1,\cdots N-1)$$

Dynamics

$$\mathbf{H}\mathbf{x}_\mathbf{k} \le \mathbf{g} \quad (k = 0,1,\cdots N)$$

Spatial constraints

$$\mathbf{x}_0 = \mathbf{x}_{\text{start}}$$

Initial position and velocity

$$\mathbf{x}_N = \mathbf{x}_{\text{goal}}$$

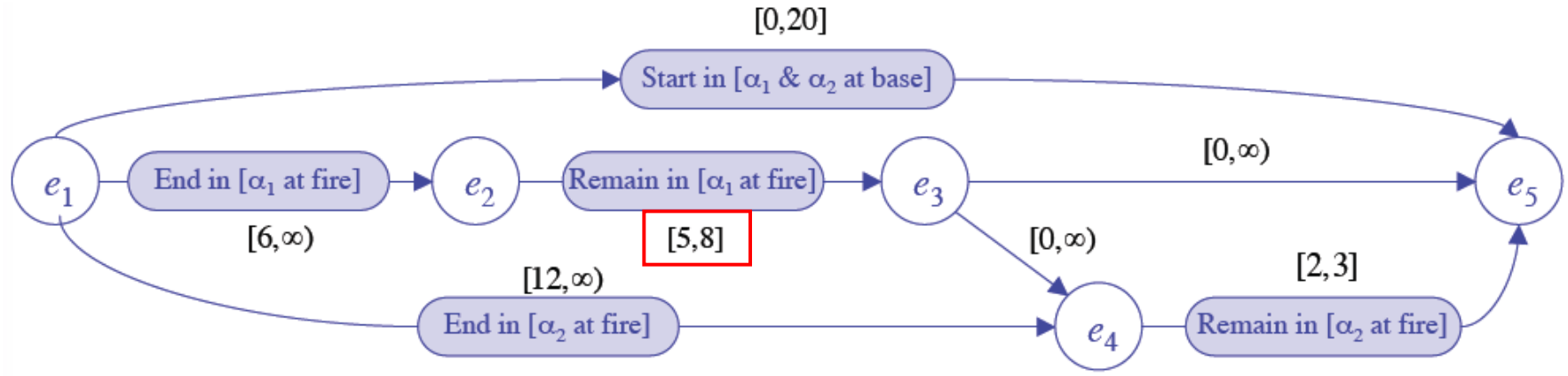
Goal position and velocity

$$-\mathbf{u}_{\max} \le \mathbf{u}_k \le \mathbf{u}_{\max} \quad (k = 0,1,\cdots N-1)$$

Actuation limits

$$\mathbf{x}_k \equiv \begin{pmatrix} x_k & y_k & \dot{x}_k & \dot{y}_k \end{pmatrix}^T, \ \mathbf{u}_k \equiv \begin{pmatrix} F_{x,k} & F_{y,k} \end{pmatrix}^T$$

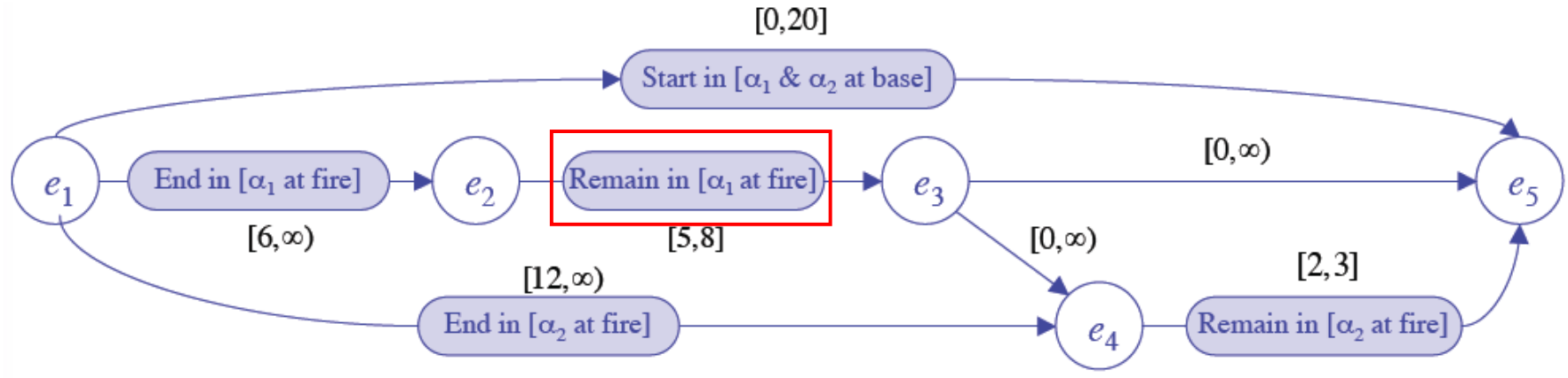
Encoding Temporal Constraints

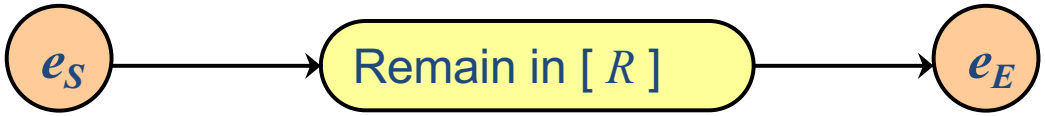


$T(e_E), T(e_S)$: decision variables

$$LB \leq T(e_E) - T(e_S) \leq UB$$

Encoding "Remain In" Constraints





$$\bigwedge_{k=0}^{k=N} \left\{ T(e_S) \le t_k \le T(e_E) \Rightarrow \mathbf{x}_k \in R \right\}$$

Use standard mapping of" implies"
to clauses or binary integer constraints (DLP or BIP).

Disjunctive Linear Program

$$Minimize\ f(x)$$

$$Subject\ to \bigwedge_{i=1,\ldots,n} \Big(\bigvee_{j=1,\ldots,m_i} C_{ij}(x) \leq 0 \Big)$$

clause

Binary Integer Linear Program

- Use **binary integer variable b** to decide which **constraint to turn off:**

  C1: $3x + 2y \leq 18 + b\,M$

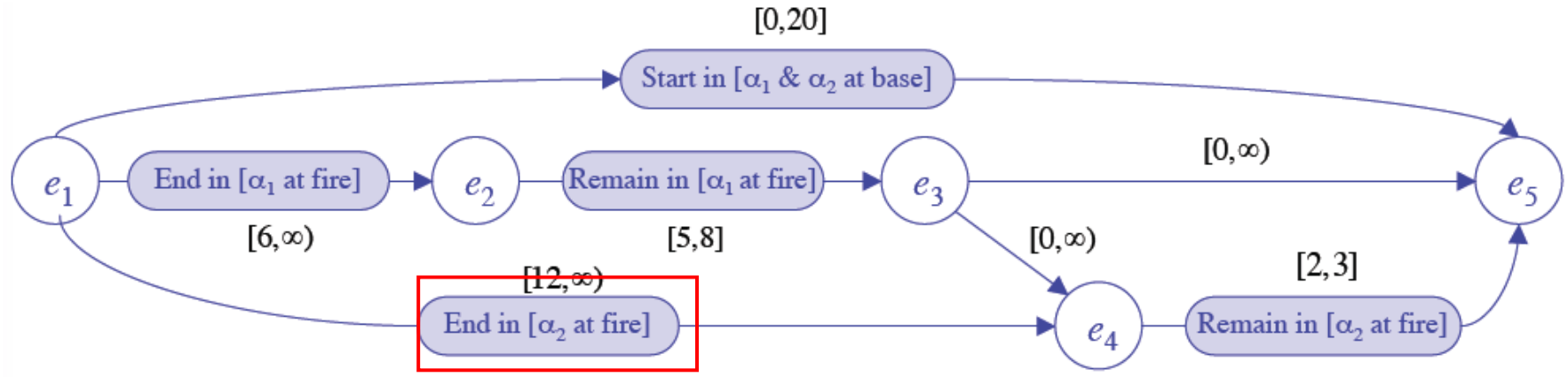
  C2: $x + 2y \leq 16 + (1-b)M$

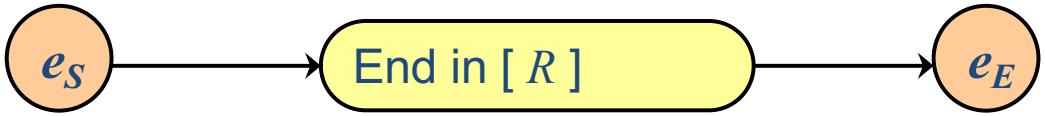
  $k \in \{0,1\}$, M is VERY large

b = 0 **implies** C1 holds

b = 1 **implies** C2 holds

Encoding "End In" Constraints





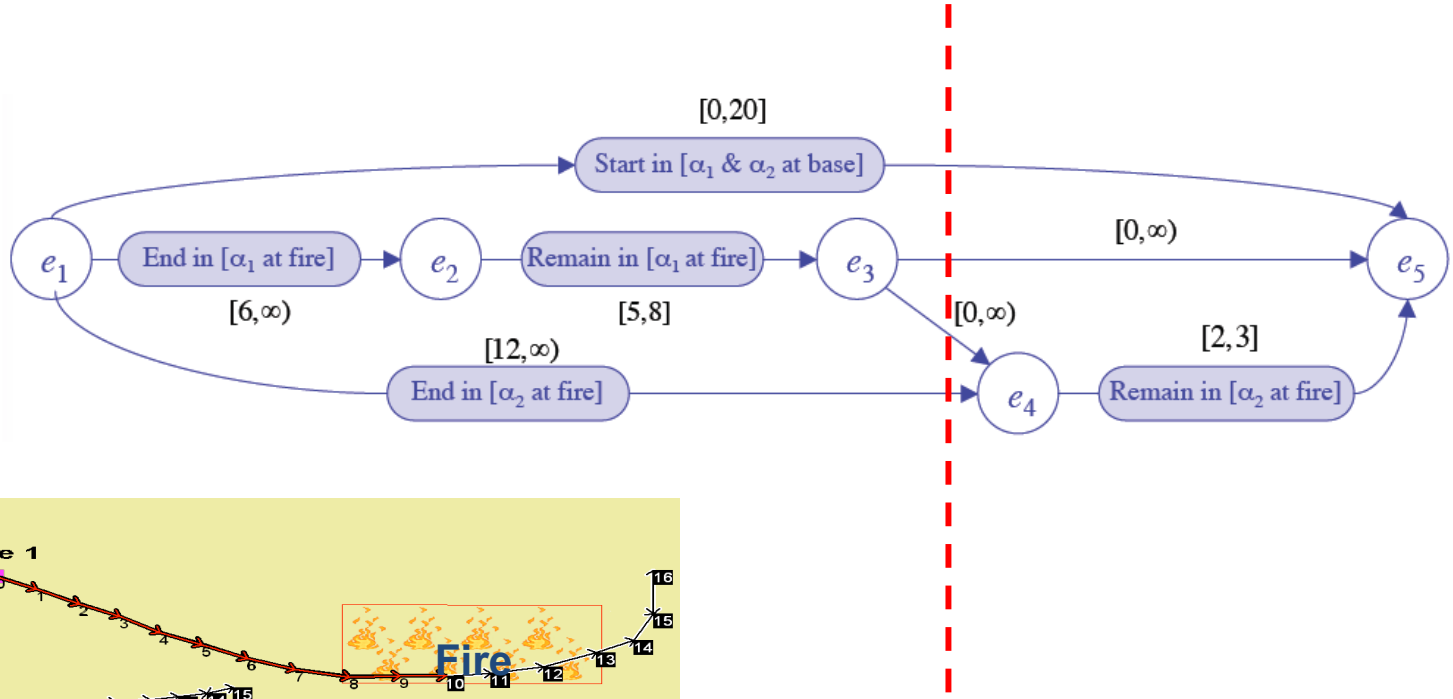
$$\bigvee_{k=0}^{k=N} \left\{ t_k - \frac{\Delta t}{2} \le T(e_E) \le t_k + \frac{\Delta t}{2} \wedge \mathbf{x}_k \in R \right\}$$

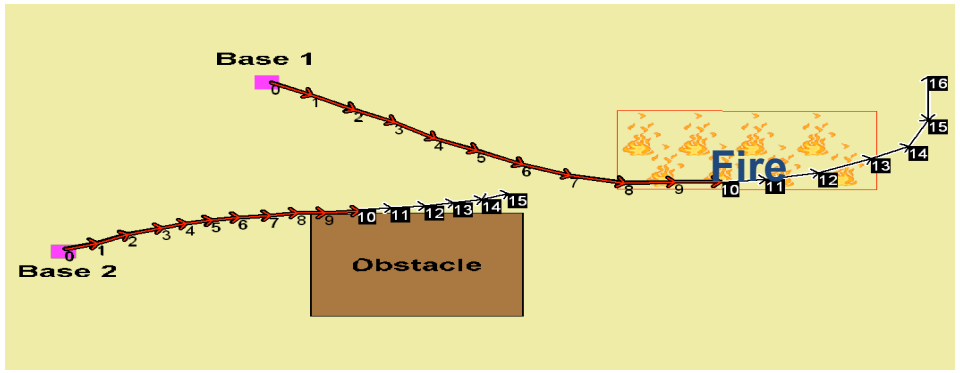
Variable

Constant

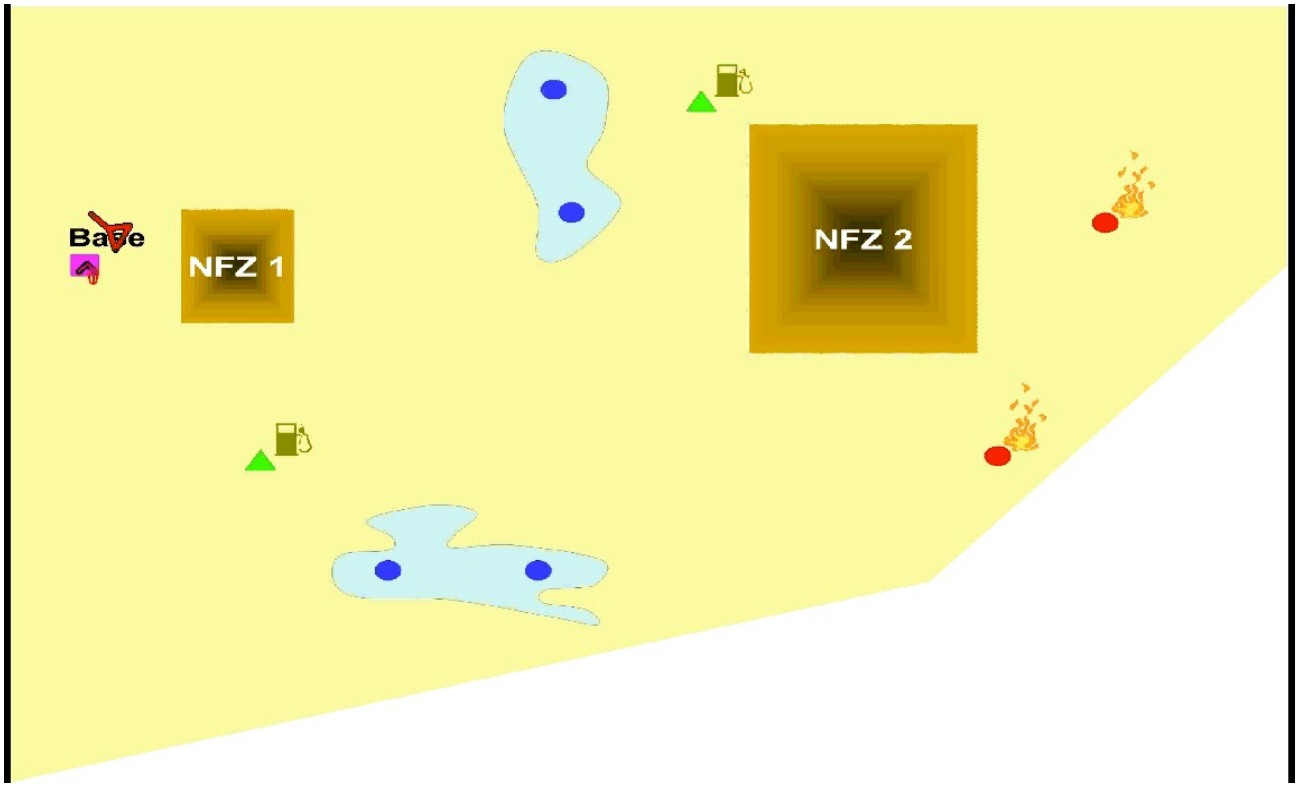
Receding Horizon

Abstract temporal and state constraints beyond current horizon.





Sulu in Action



Limits?

- Long Horizons
- Rich constraints
  (tethered agents)

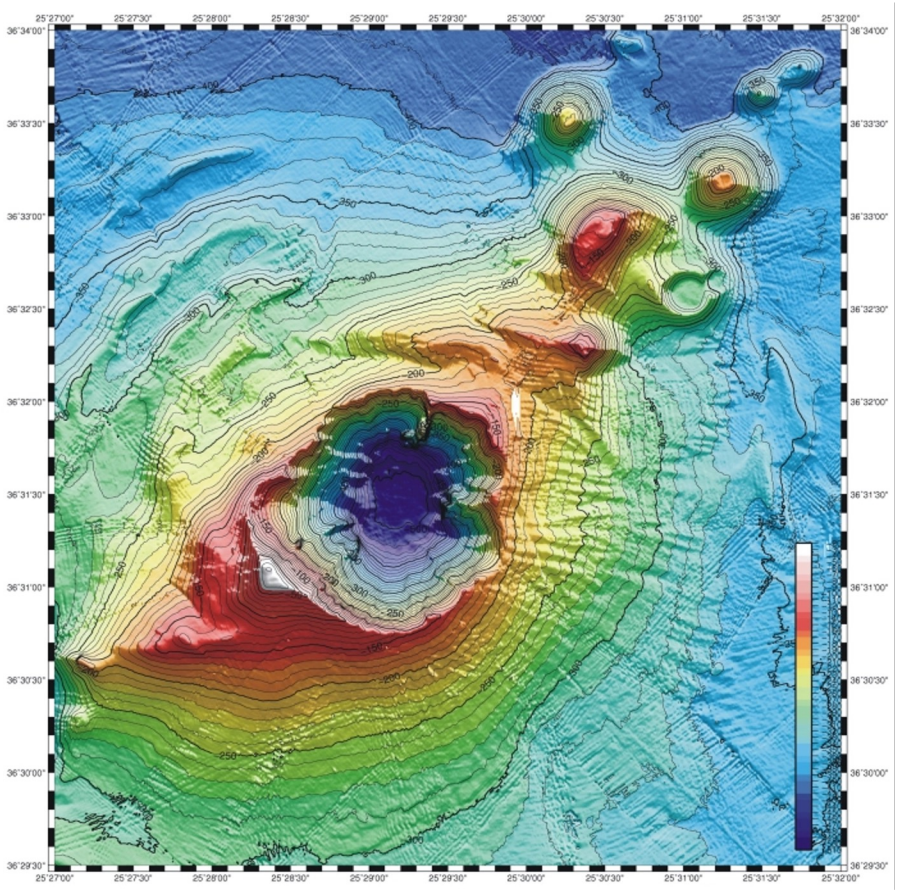
Solutions?

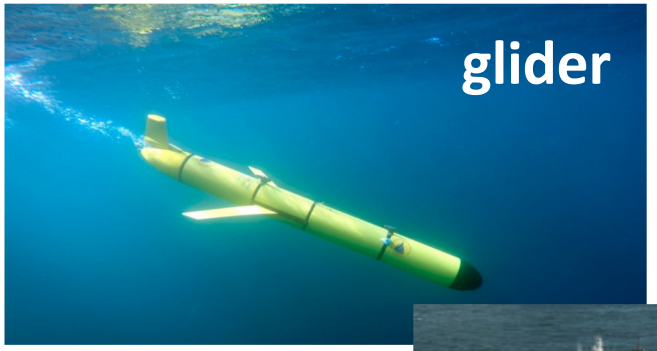
- Variable time steps
- Convex, non-linear constraints

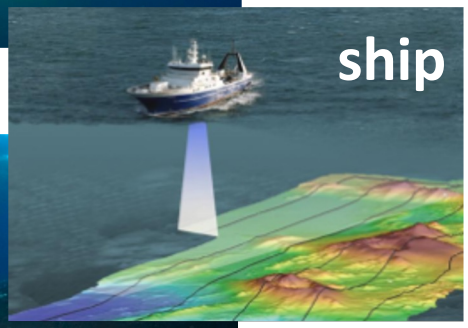
Outline

- State Programs and State Plans
- Model Predictive Control
- State Plan Motion Planning
  - Sulu: A Discrete Time Formulation (convex)
  - **Scotty: A Continuous Time Formulation (convex)**
  - Magellan: dynamic execution over long time horizons
- Risk-aware State Plan Motion Planning
- Risk-aware Planning with Learned Behaviors (LaPLASS)
- Avoiding Obstacles(Appendix)

Multi-Vehicle Campaigns

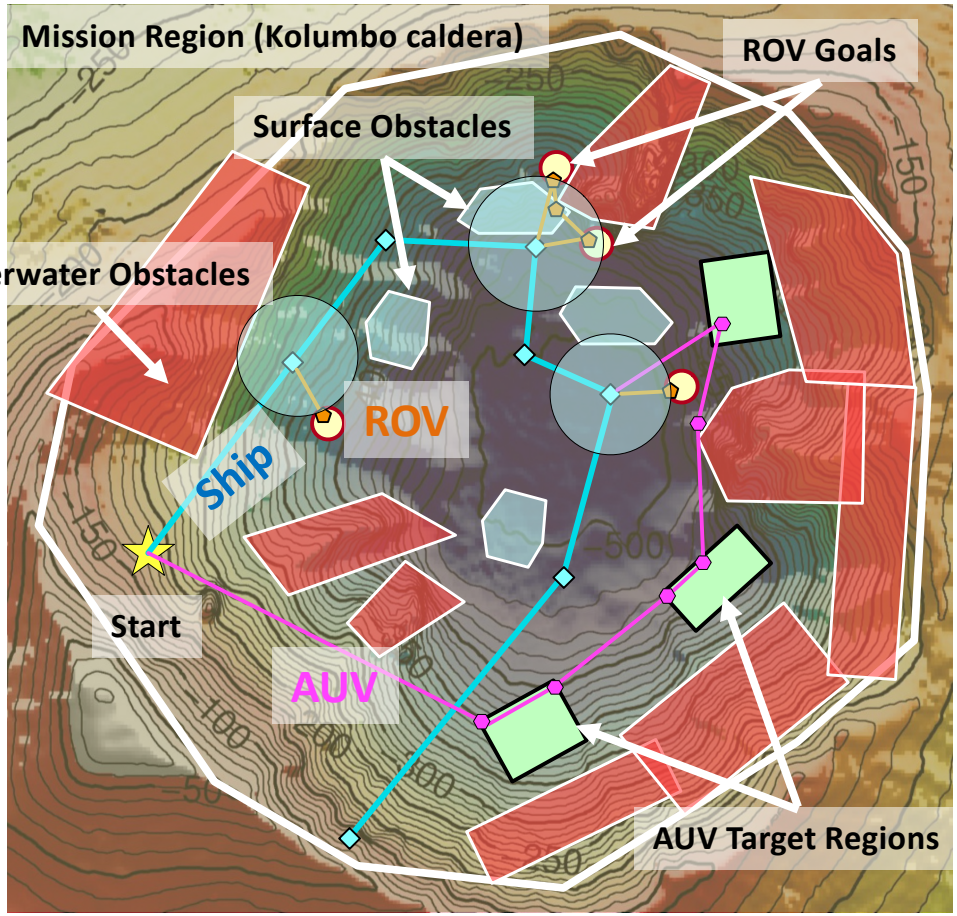






Kolumbo Deep-Sea Volcano near Santorini, Greece Team: WHOI, MIT, AFRC, U. Michigan 2019

Multi-Vehicle Campaigns



Characteristics of the problem:

Long horizons

- mission spans hours or days
- long transit times

Activities coupled via state constraints

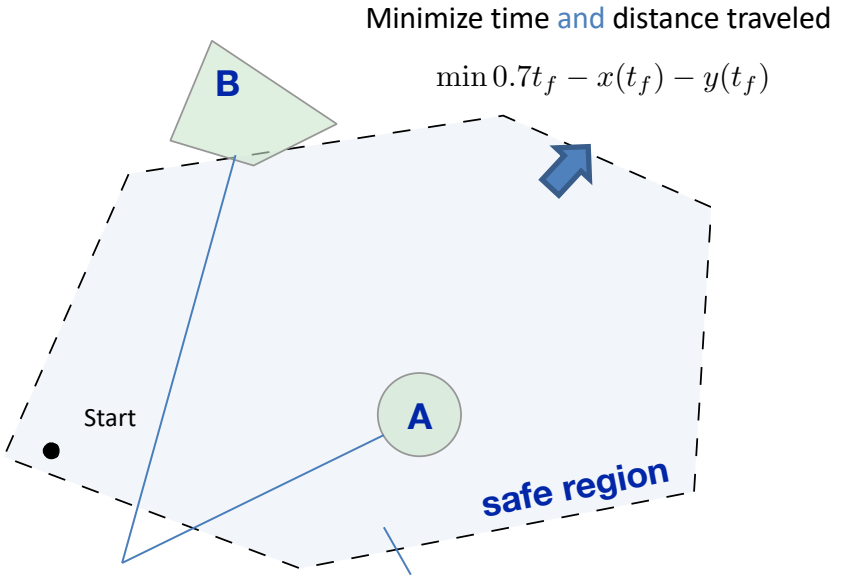
- ship-ROV tether range
- ship-AUV communication range
- AUV battery life

Use "continuous time" approach
(aka variable time steps)

Solution: Scotty Task & Motion Planner
[Fernandez, Karpas, Williams IJCAI 15]

- Today: ScottyPath

Input: Environment + State Plan + Linearized Dynamics + Cost

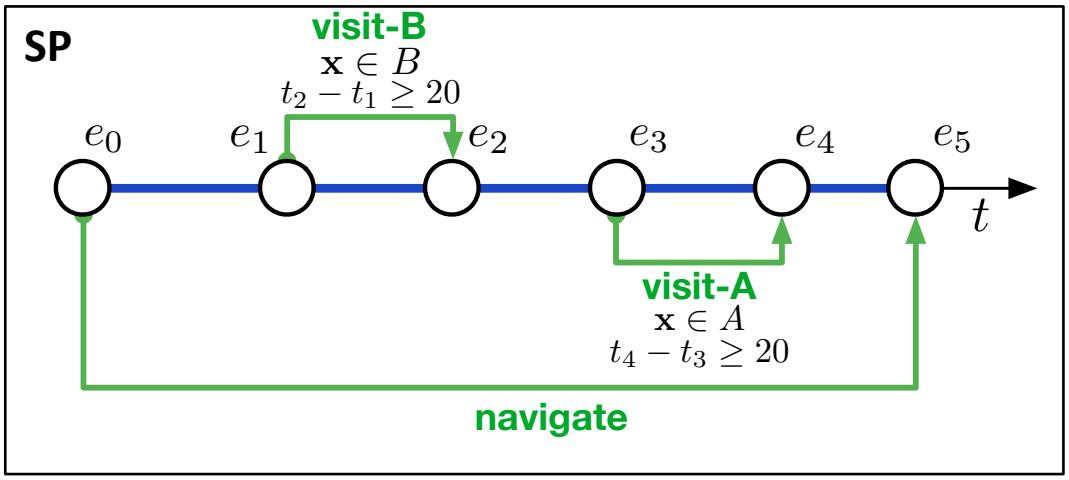


visit-A duration = 20 sec
over all: x ∈ A

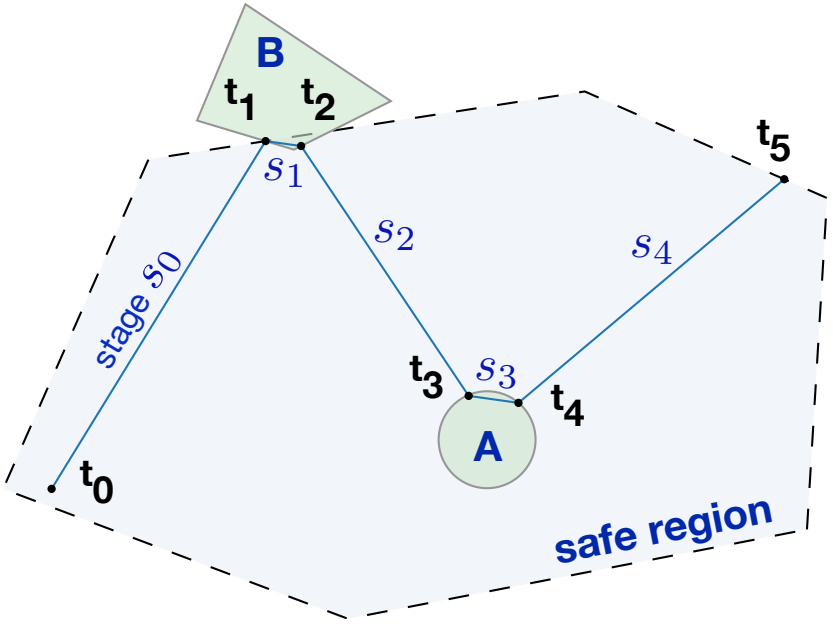
visit-B duration = 20 sec
over all: x ∈ B

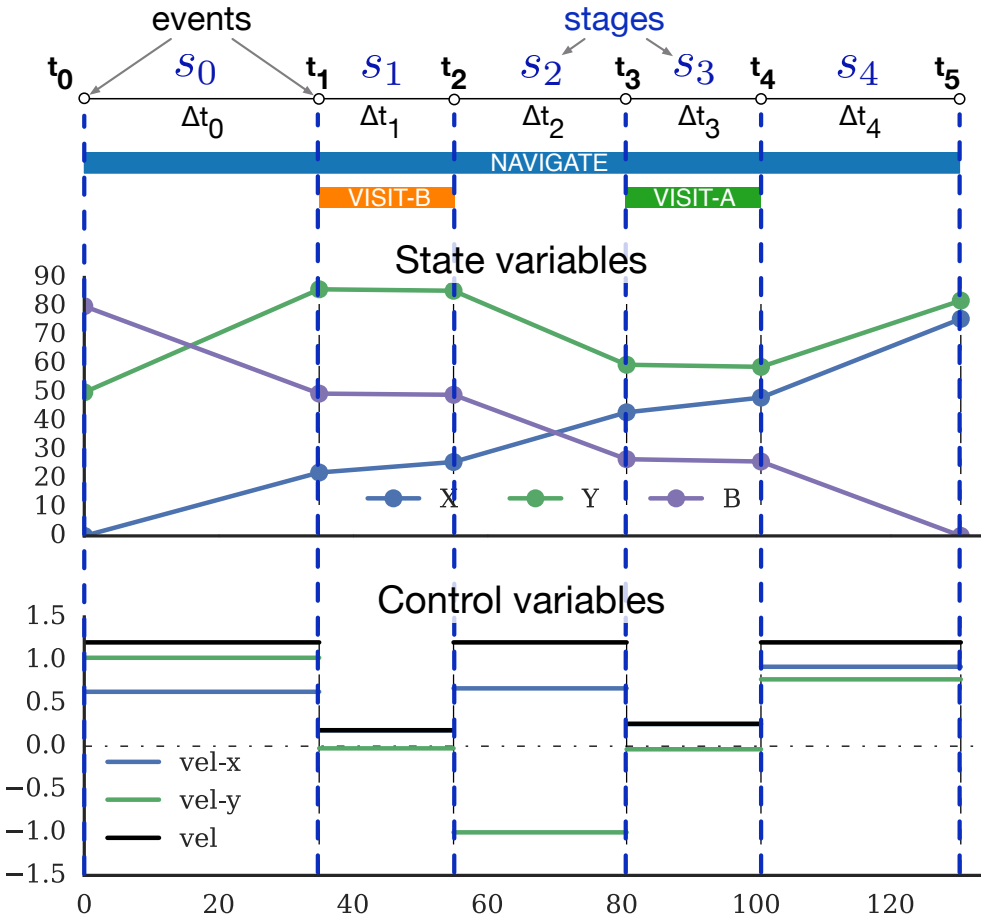
Navigate:

$$x(t) = x(t_0) + v_x \cdot (t - t_0)$$
$$y(t) = y(t_0) + v_y \cdot (t - t_0)$$



Output: Control Trajectory + State Trajectory + Schedule



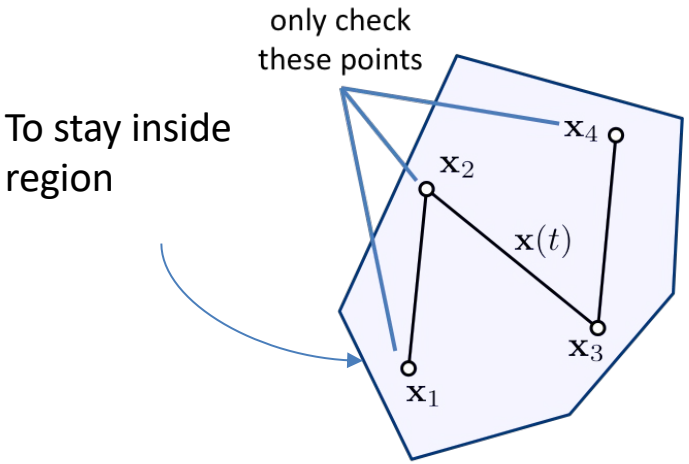


Solve by Encoding:

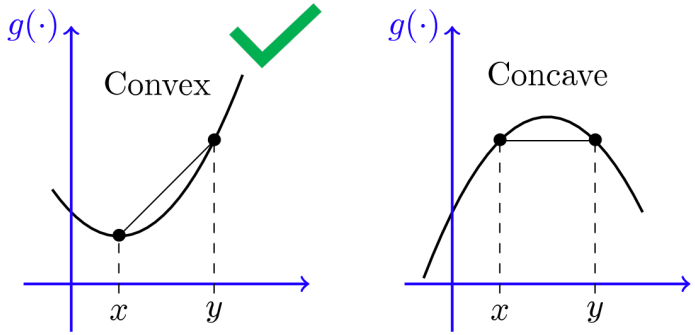
- **Convex program using continuous time**
- **Can be solved quickly and optimally over long horizons!**

Why does convexity matter?

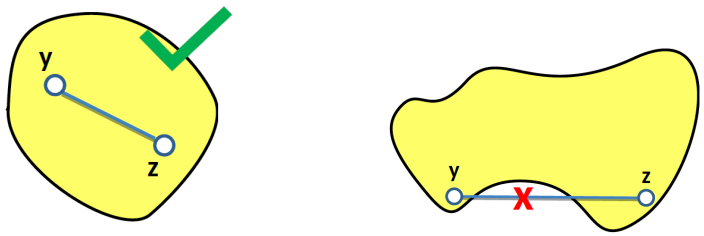
Scotty uses piecewise convex trajectories and convex constraints and costs



Convex Function:



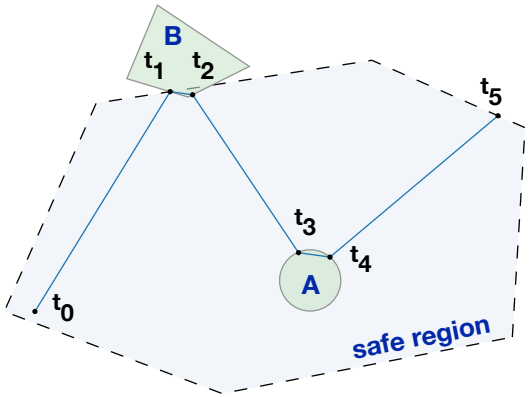
Convex Set:

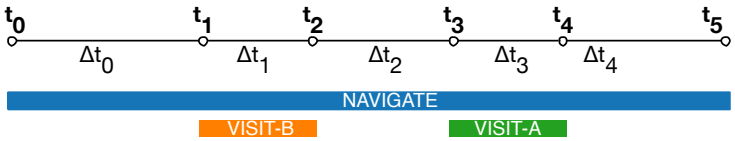


Properties of Convex Solvers:

- **Complete:** If no solution found, there is no solution.
- **Optimal:** Local minima are global minima → any solution returned is guaranteed optimal.
- **Efficient:** Methods are orders of magnitude faster than non-linear solvers.

What Challenge is introduced by Continuous Time?



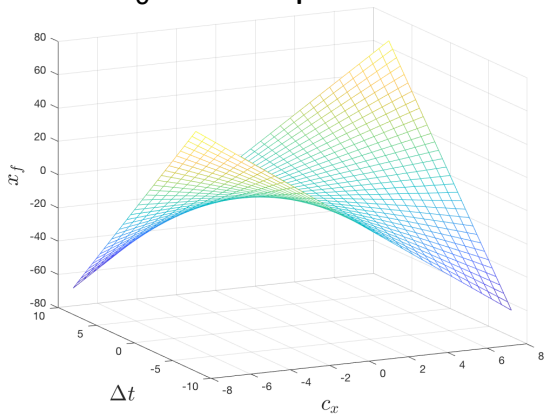


Dynamics constraint for a stage:

$$x_f = x_0 + c_x \cdot (t_f - t_0)$$

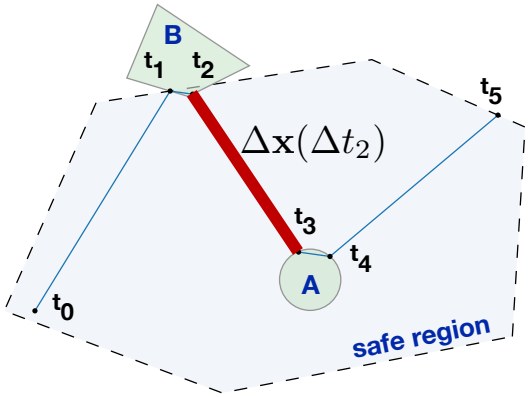
non-convex, non-linear constraint

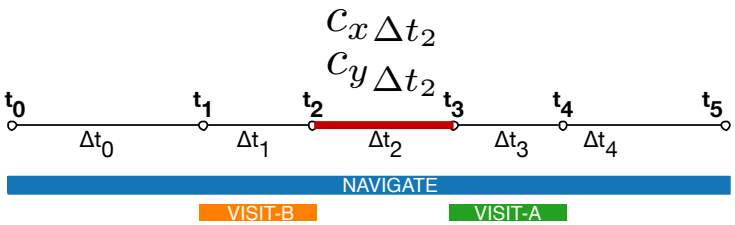
- Assume $x_0 = 0$ and plot this constraint



➢ Problem: A straightforward encoding of dynamics is **nonlinear** and **nonconvex**.

Scotty uses a Linear Reformulation "Trick"





Continuous effect

$$x_f = x_0 + c_x \cdot (t_f - t_0)$$

non-convex, non-linear constraint

Instead

proxy decision variable -
replaces non-linear term and C_x

$$c_{x\Delta t} = c_x \cdot (t_f - t_0) \qquad c_{x_{min}} \cdot (t_f - t_0) \leq c_{x\Delta t} \leq c_{x_{max}} \cdot (t_f - t_0)$$

State evolution is now linear $x_f = x_0 + c_{x\Delta t}$

Values of control variables can be retrieved, once optimization is solved.

Convex Optimization Supports Norm Constraints

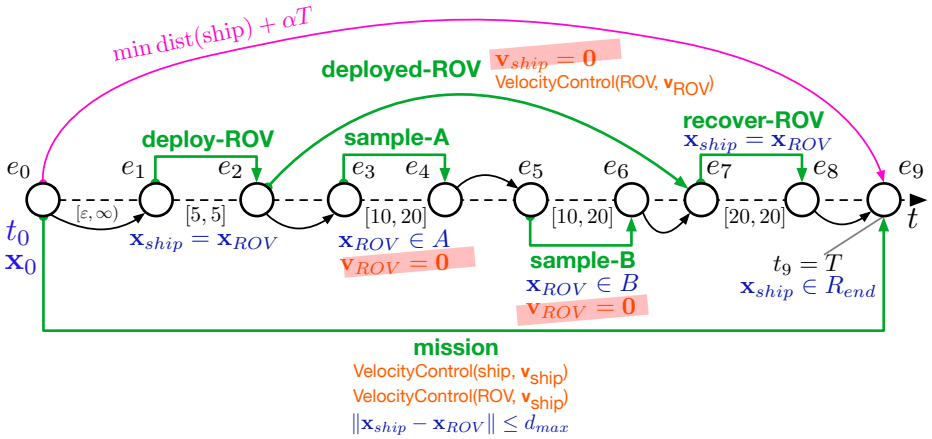
- Max distance between objects (e.g. ship-ROV)

$$\|\mathbf{x}_{ROV} - \mathbf{x}_{Ship}\|^2 \leq R^2_{tether}$$

- Battery decrease is proportional to magnitude of vehicle speed

$$\dot{b}(t) \leq -k \parallel \boldsymbol{c}_{\boldsymbol{\Delta t}} \parallel_2$$

Example Ship-ROV

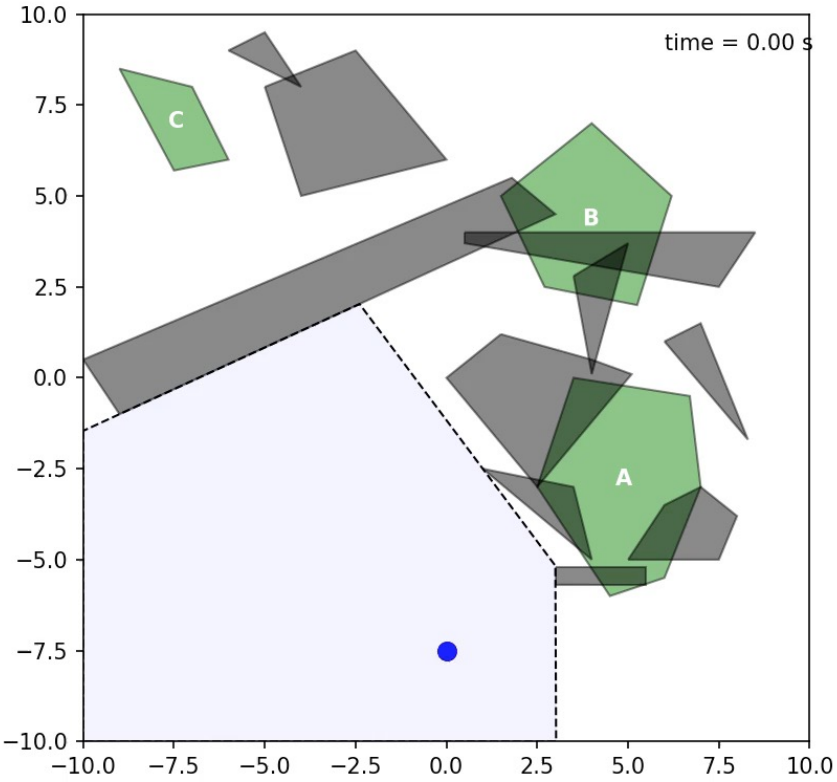


Minimize:

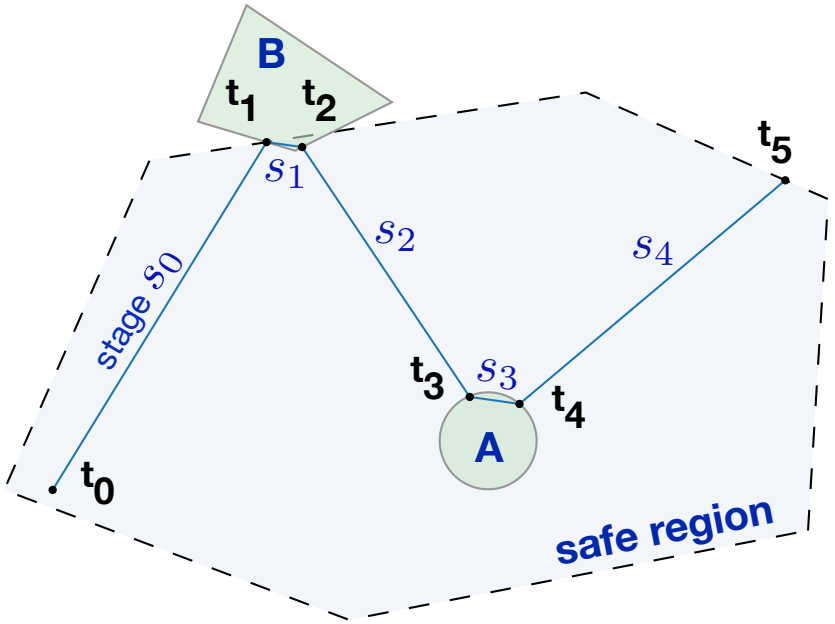
- Time to completion and
- Distance traveled by the ship

Ship and ROV can **both move** at **all times!**

Obstacles addressed in Appendix



Limitations of ScottyPath



Limitations?

- Piecewise constant control + first-order dynamics for long durations is too simplistic for real-time agents!

Outline

- State Programs and State Plans
- Model Predictive Control
- State Plan Motion Planning
  - Sulu: A Discrete Time Formulation (convex)
  - Scotty: A Continuous Time Formulation (convex)
  - **Magellan: dynamic execution over long time horizons**
- Risk-aware State Plan Motion Planning
- Risk-aware Planning with Learned Behaviors (LaPLASS)
- Avoiding Obstacles(Appendix)

How do we get the best of both worlds?

Sulu (Léauté, 2005)
a model-based executive for temporal plans

- ☑ **accurate model** over **short horizon** due to **small steps**
- ☑ uses **receding horizon** to **adapt** and gain **tractability**
- ☒ **heuristic** is **crude**; ⇒ myopic to finite horizon

ScottyPath (Fernàndez-Gonzàlez et. al, 2018)
hybrid activity + motion planner for multi-agent coordination

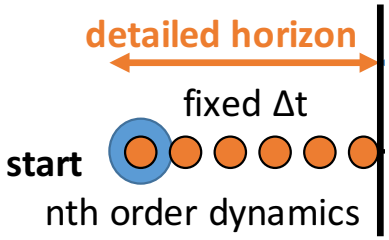
- ☑ reasons efficiently over arbitrarily **long horizons** while respecting **state plan**
- ☒ **simple dynamics** leads to less **precise trajectories** that can be **unsafe** if executed
- ☑ but long horizon SP trajectories offer **informative** (and admissible) **heuristics**

How do we execute continuous state plans?

use accurate dynamics for what is executed
shape over full mission horizon

Magellan:

- **Planner** formulation mixes **discrete-time** **and** **continuous-time**

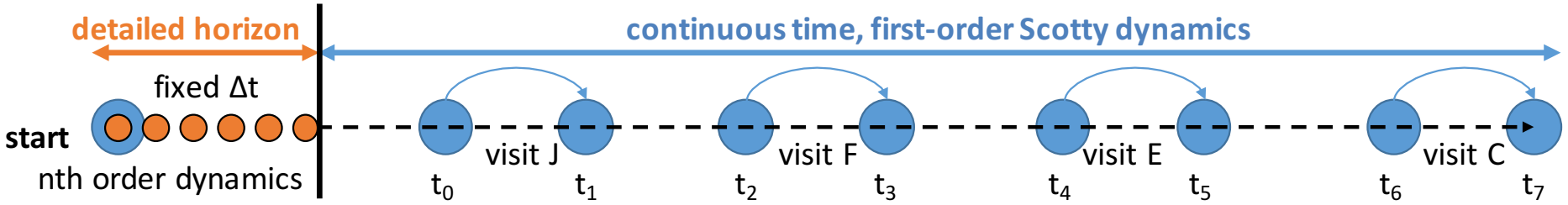


How do we execute continuous state plans?

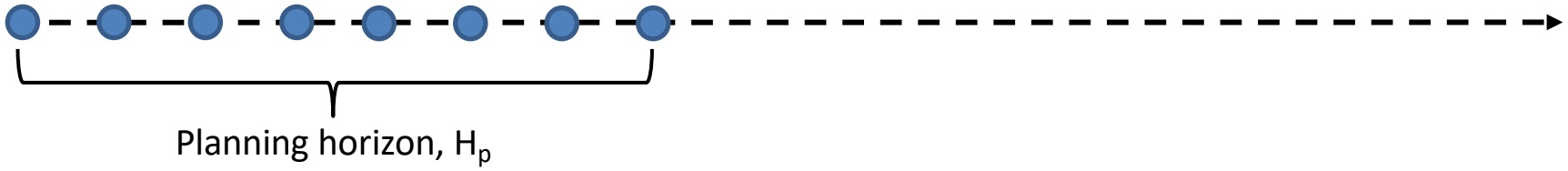
use accurate dynamics for what is executed
shape over full mission horizon

Magellan:

- **Planner** formulation mixes **discrete-time** **and** **continuous-time**



- Incorporated within **model predictive control**

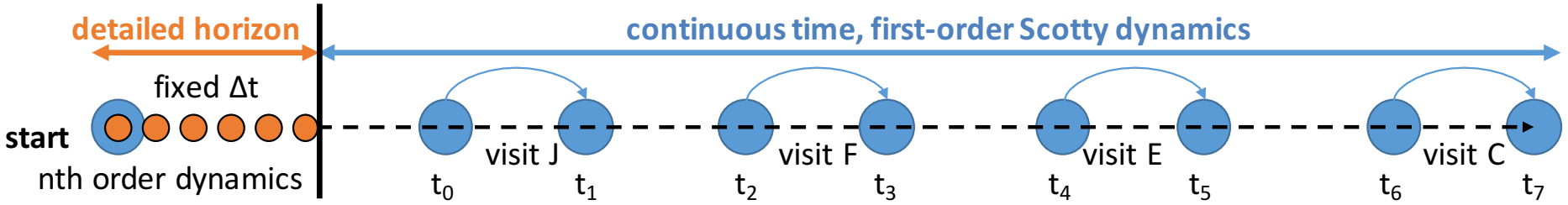


How do we execute continuous state plans?

use accurate dynamics for what is executed
shape over full mission horizon

Magellan:

- **Planner** formulation mixes **discrete-time** **and** **continuous-time**



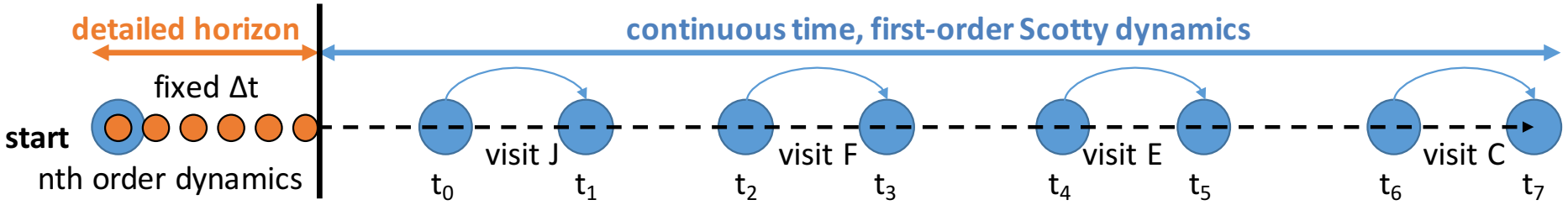
- Incorporated within **model predictive control**

How do we execute continuous state plans?

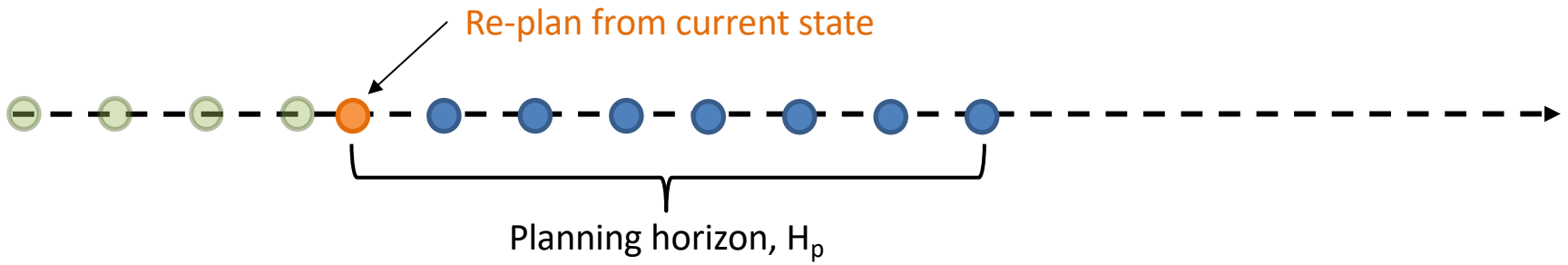
use accurate dynamics for what is executed
shape over full mission horizon

Magellan:

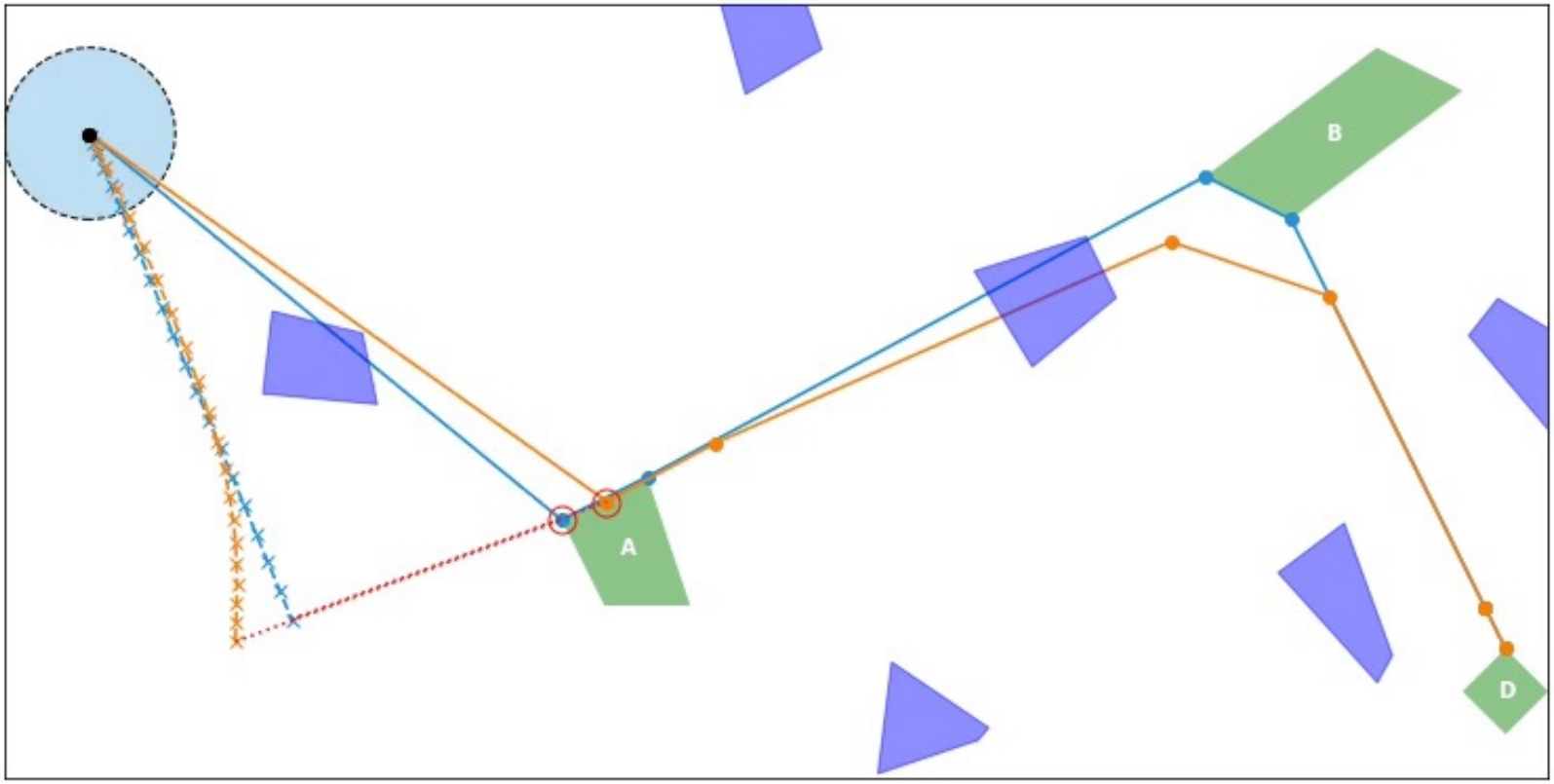
- **Planner** formulation mixes **discrete-time** **and** **continuous-time**



- Incorporated within **model predictive control**



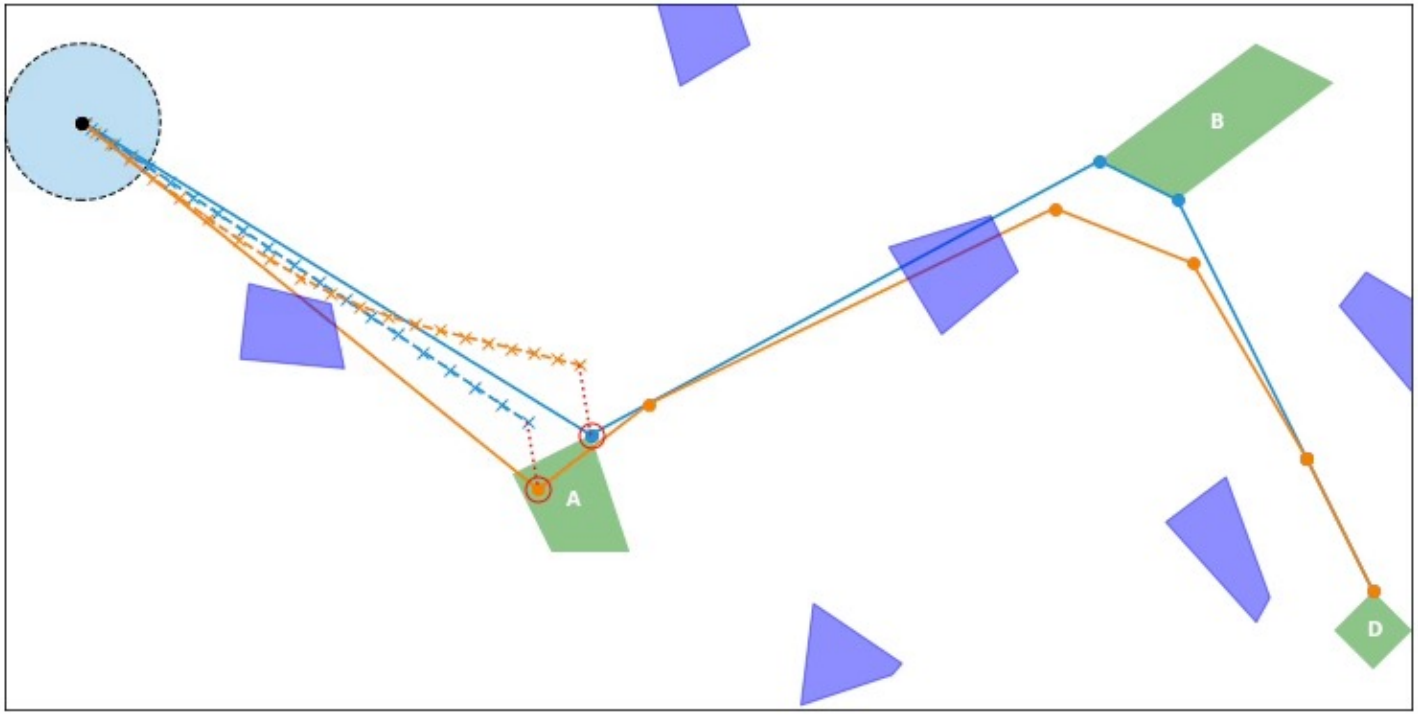
An Ocean Exploration Campaign Revisited



An Ocean Exploration Campaign Revisited

Suppose the AUV battery capacity becomes reduced,

Magellan has Ship rendezvous earlier to pick up the AUV



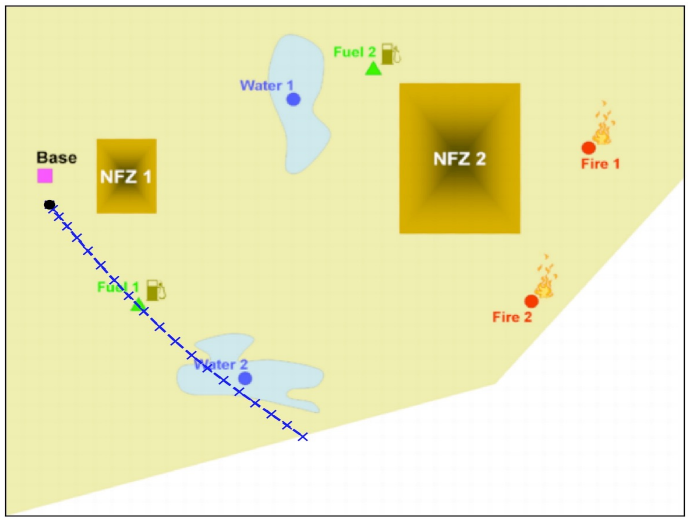
Comparison to Sulu

Planner	OBJ	t_o	t_f
Sulu	150.7	0.053	0.072
Magellan	57.2	0.044	0.036

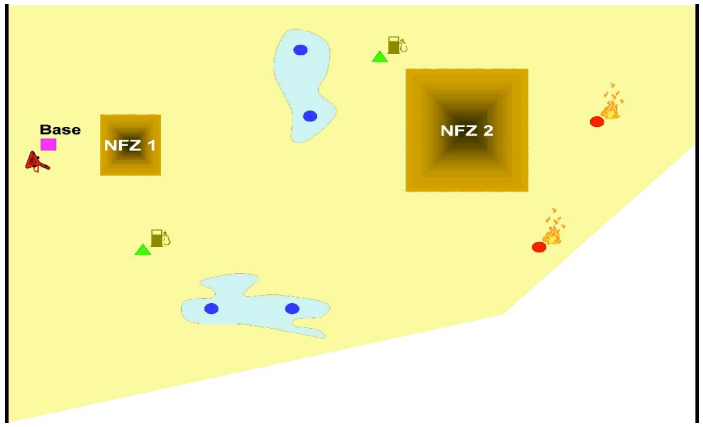
Magellan achieves a **2x improvement** in solution **quality** and planning **time** compared to Sulu (Léauté, 2005)

OBJ – objective value
t_o – mean solve time for single planning horizon
t_f – mean formulation time for single planning horizon

Magellan



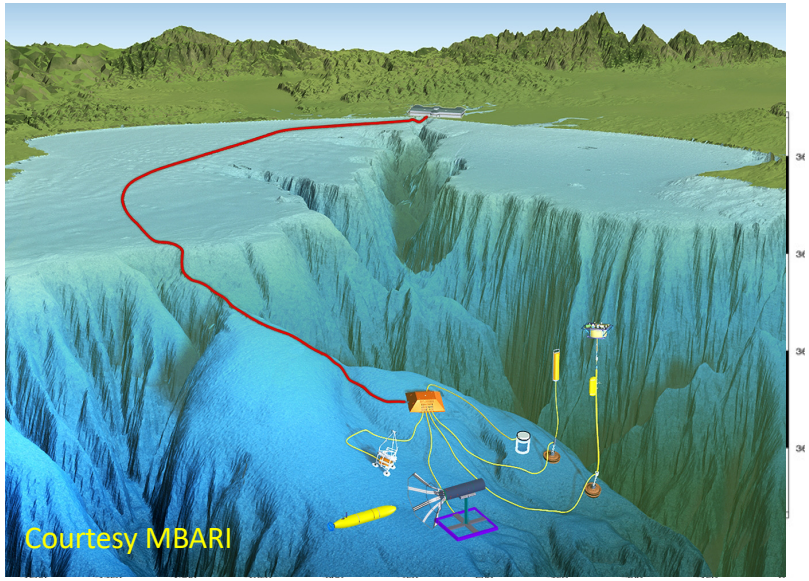
Sulu

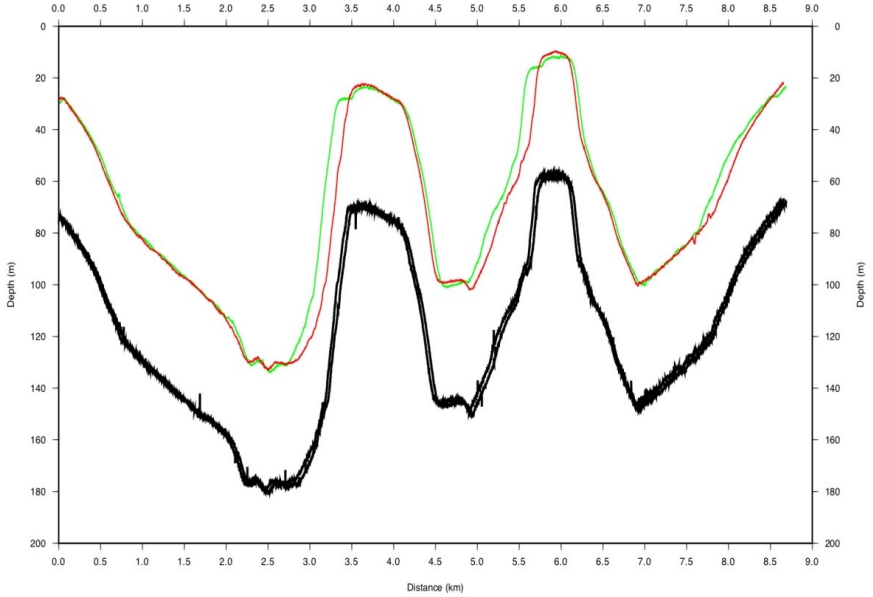


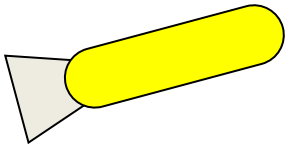
Outline

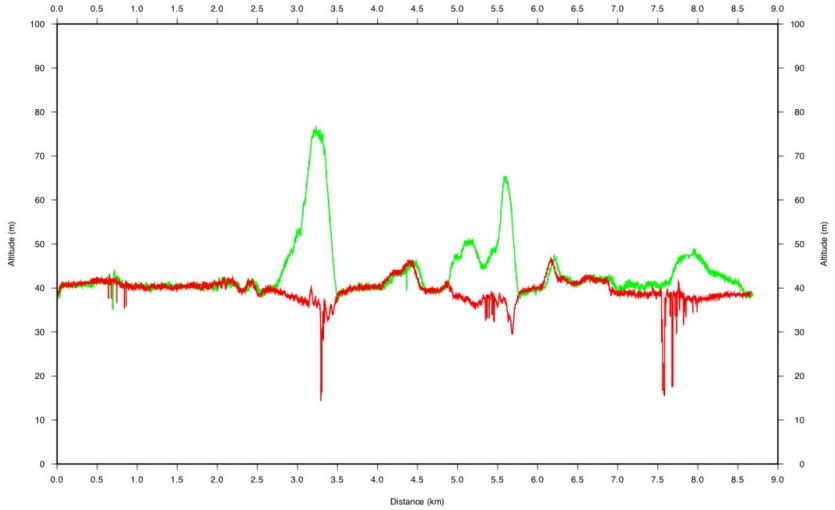
- State Programs and State Plans
- Model Predictive Control
- State Plan Motion Planning
- **Risk-aware State Plan Motion Planning**
- Risk-aware Planning with Learned behaviors (LaPLASS)
- Avoiding Obstacles (Appendix)

Depth Navigation for Bathymetric Mapping – Jan. 23rd, 2008





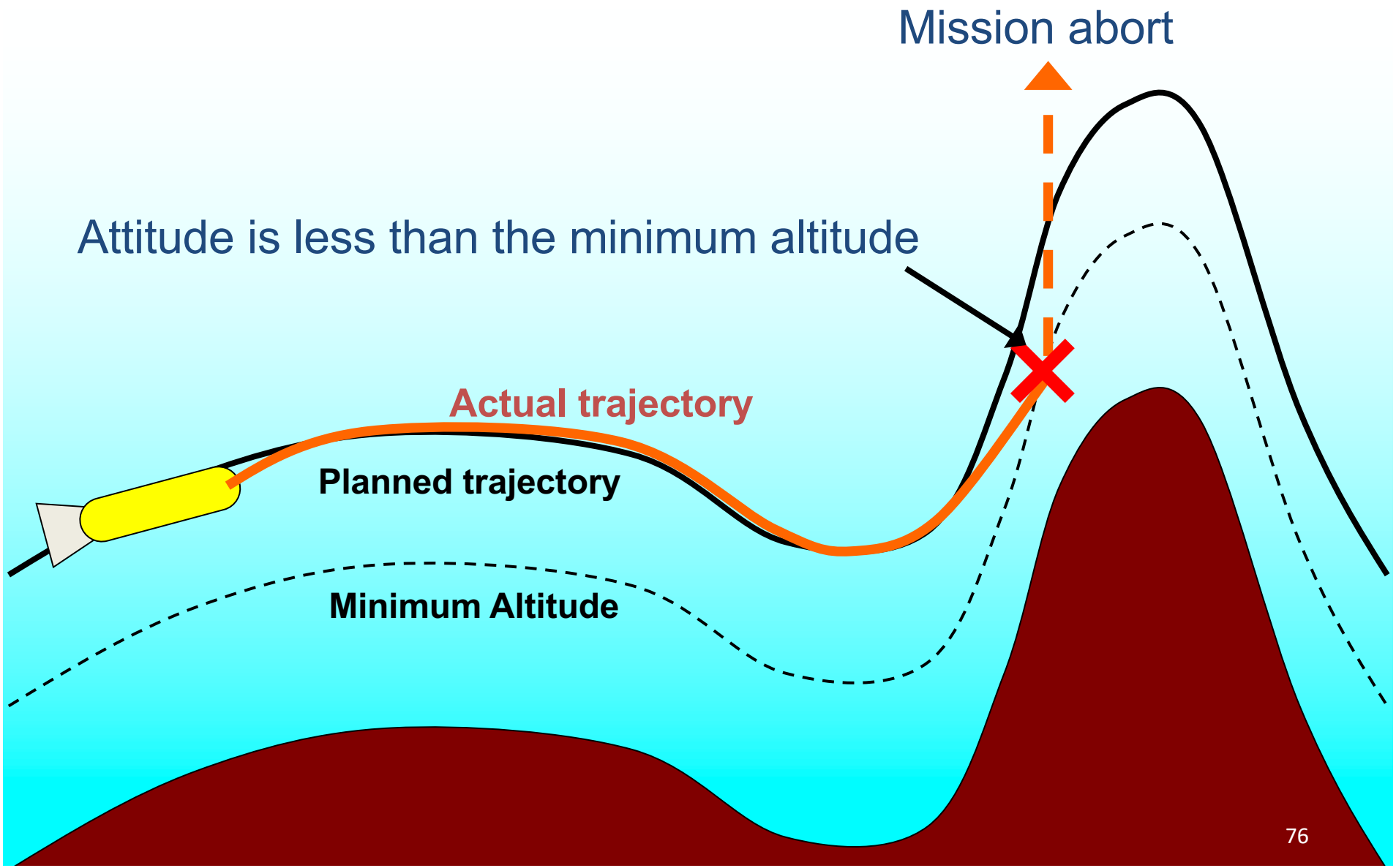




Black – sea floor
Green – adhoc planner
Red – linear program planner


>$2M

Issue: Frequent Mission Aborts



How Do We Improve Safety?

- Utilitarian

$$\min_{\mathbf{U}} J(\overline{\mathbf{X}}, \mathbf{U}) + pf(\mathbf{U})$$

Penalty (constant)

Probability of failure

- **Chance constrained**

$$\min_{\mathbf{U}} J(\overline{\mathbf{X}}, \mathbf{U})$$

$$s.t. \quad f(\mathbf{U}) \leq \Delta$$

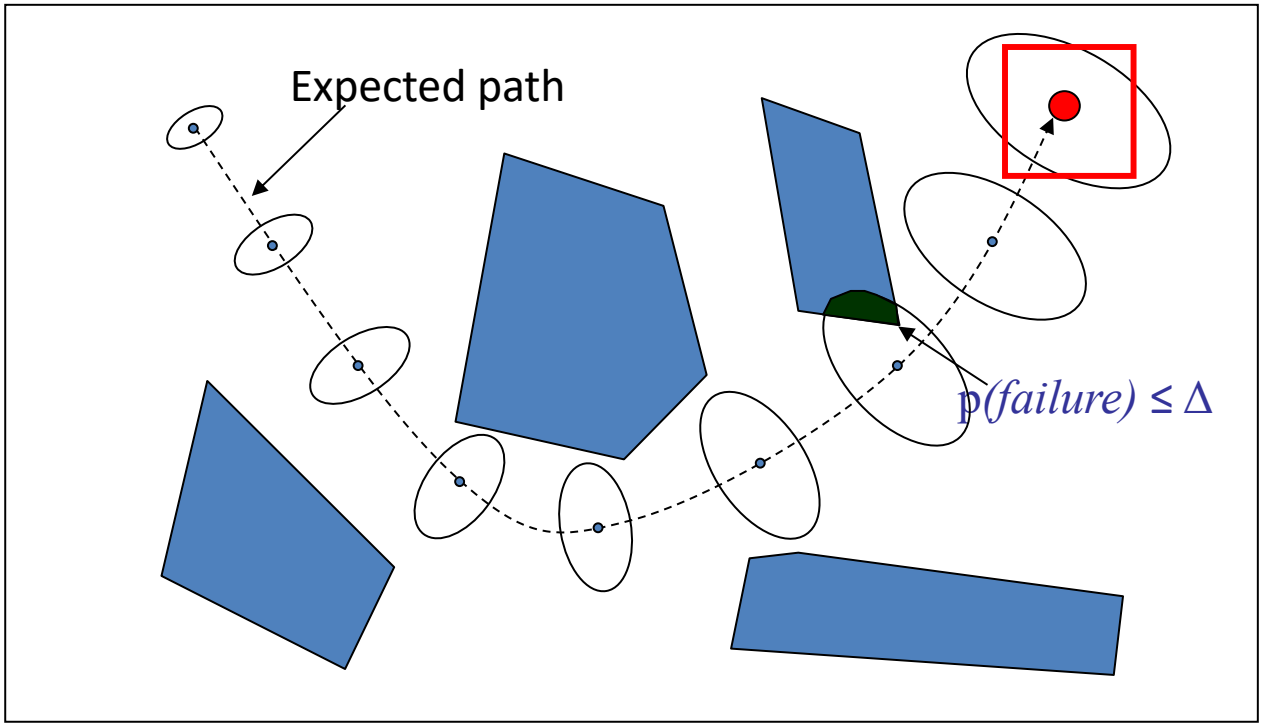
Probability of failure

Risk bound

“Risk-bounded” Path Planning

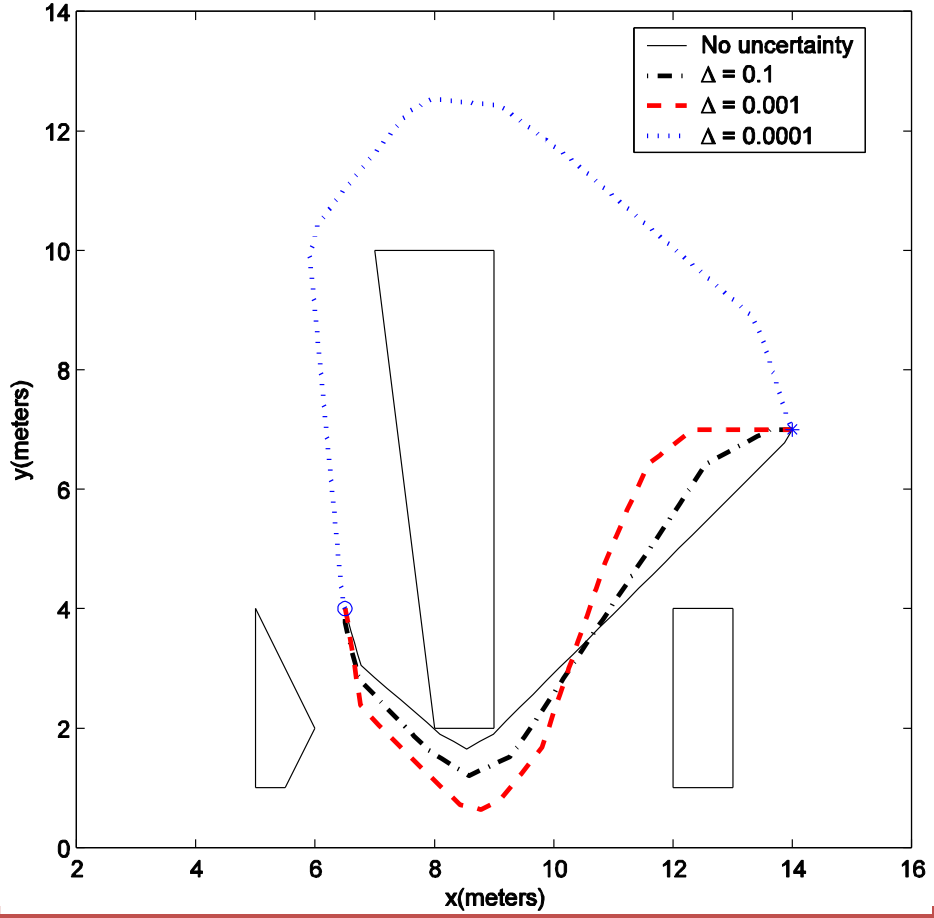
– “Plan optimal path to goal such that p(failure) ≤ Δ.”

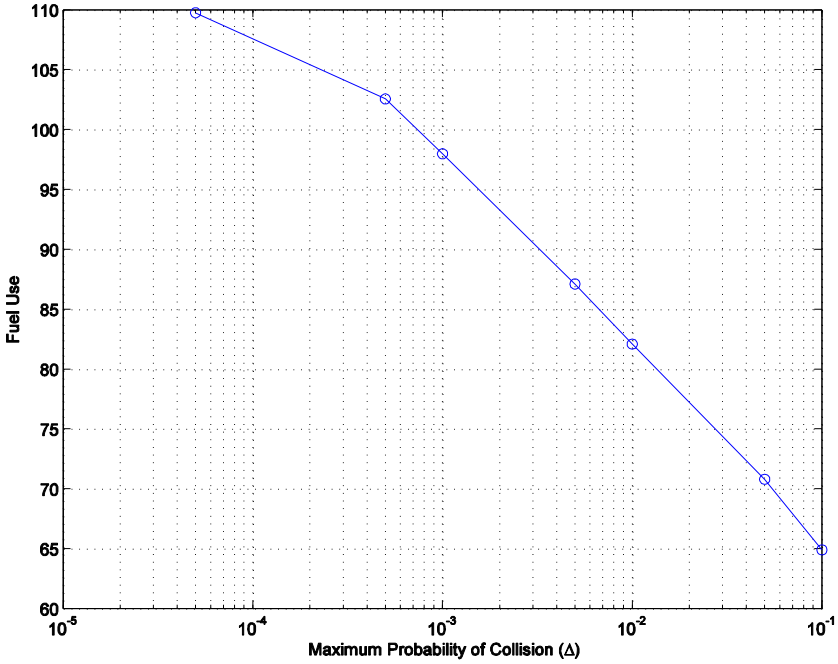
Chance Constraint(CC)



Risk – Performance Tradeoff

- Operator decides maximum probability of failure, thus trading performance against risk-aversion.

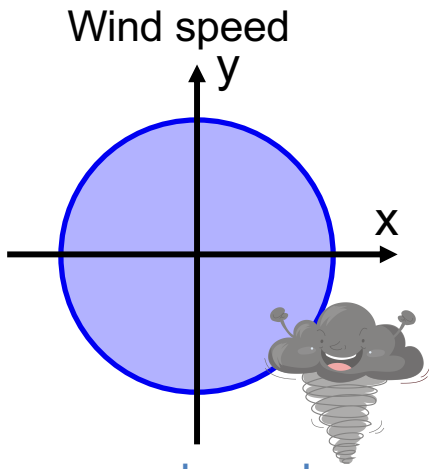




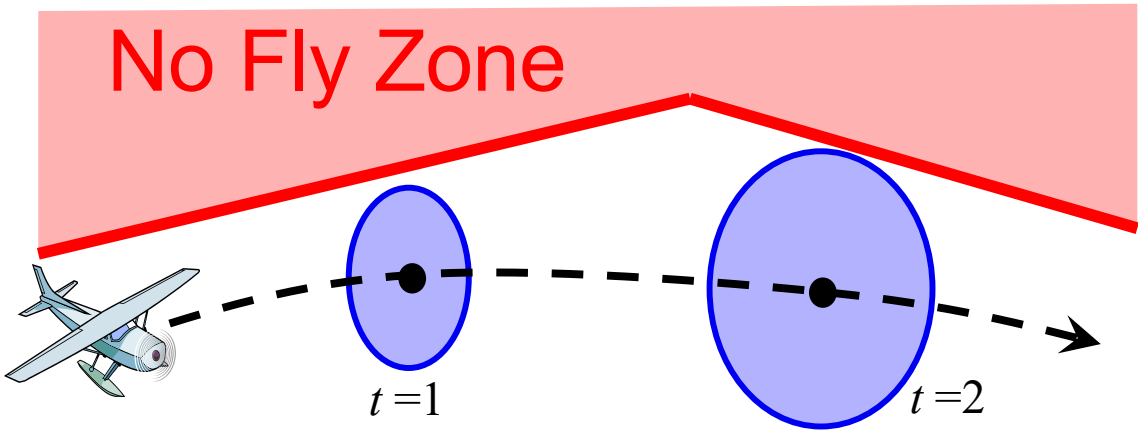
Method: Uniform Risk Allocation
[Blackmore PhD, 07]

Safety: Robust versus Chance Constrained

Traditional Robust Predictive Control

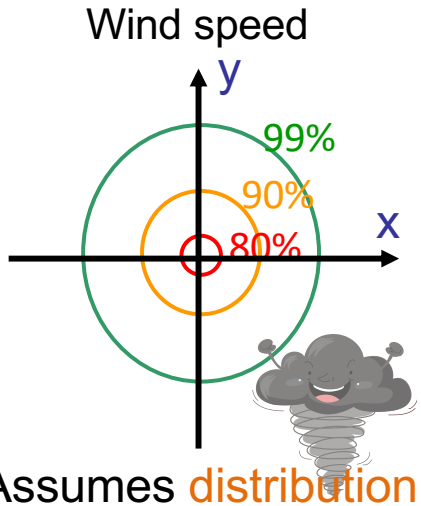


Assumes bounds on uncertainty.

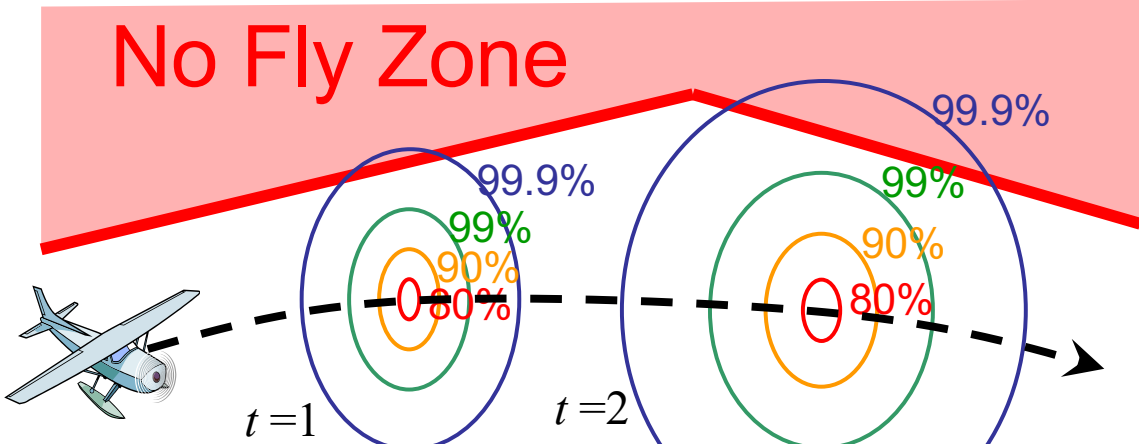


- Predict position with bounded uncertainty.
- Task: Find control sequence that satisfies constraints for ALL realizations of uncertainty.

Chance-constrained Predictive Control



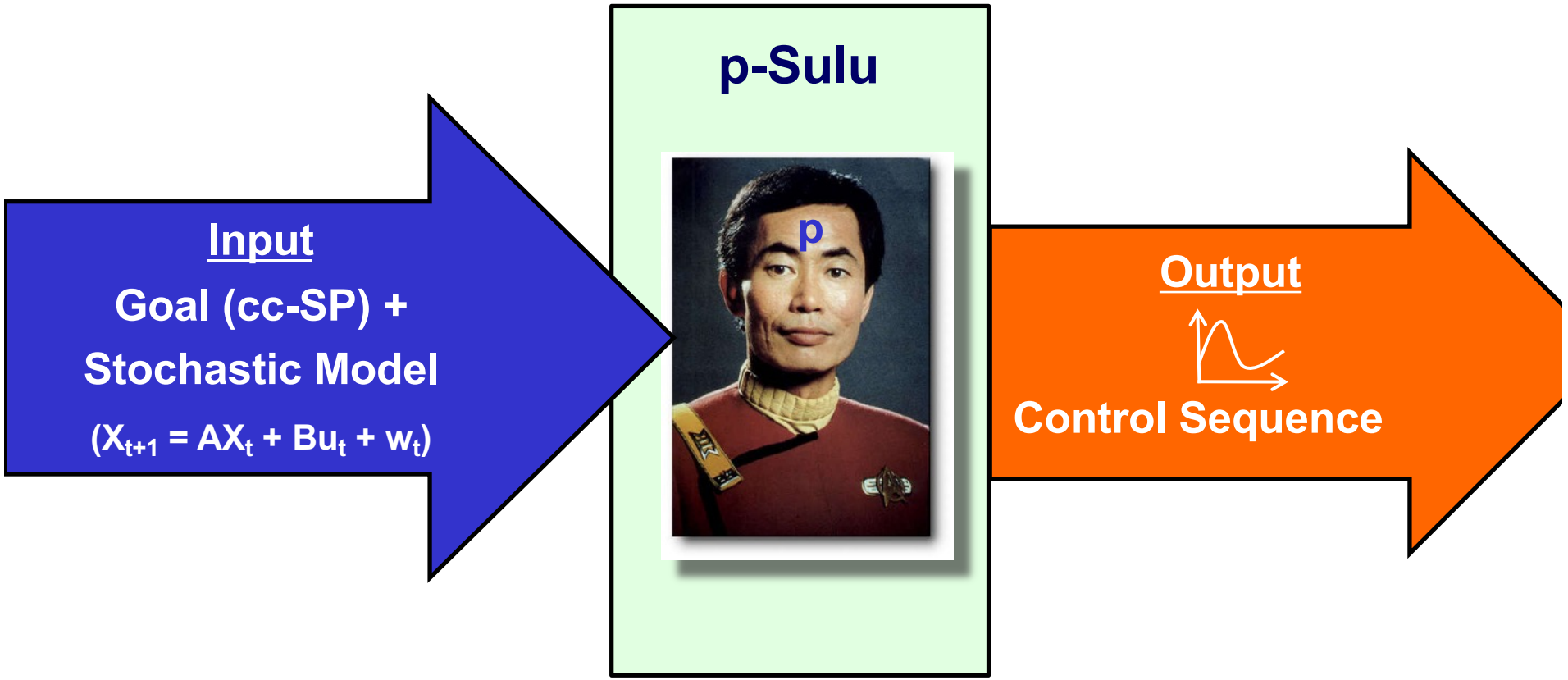
Assumes distribution that characterizes uncertainty.



- Predict position with probabilistic uncertainty.
- Task: Find control sequence that satisfies constraints within probability bound (*chance constraint*)

Probabilistic Sulu (p-Sulu)

- cc State Plan Executive

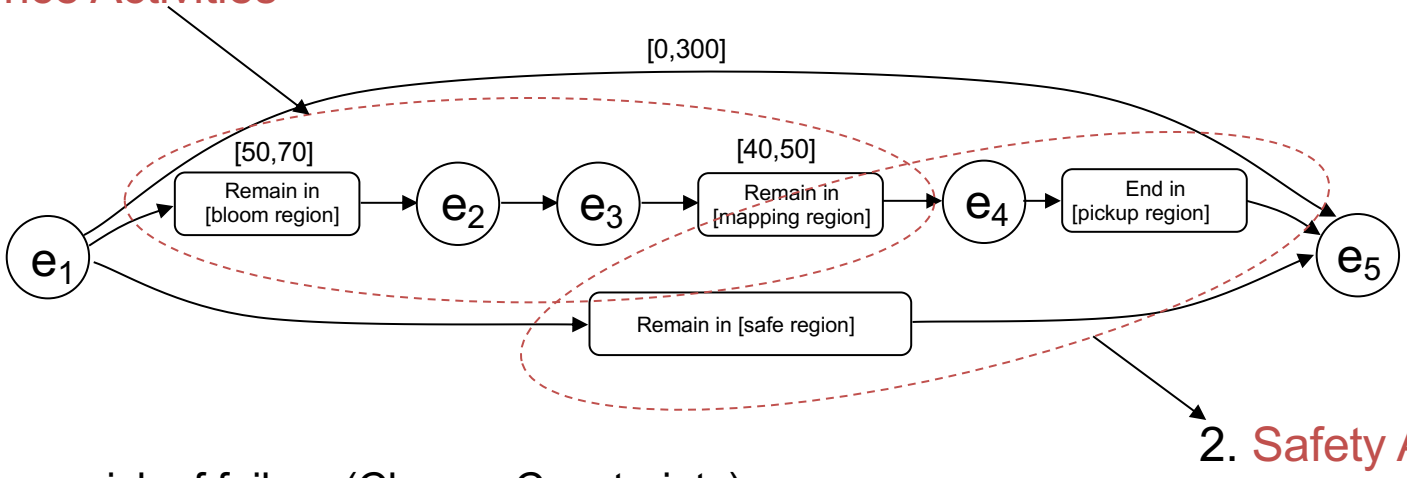


Approach: Frame as an instance of **Stochastic Programming**

Chance-Constrained State Plans (cc-SP)

"Stay over science region with 95% success.
Avoid collision and achieve pickup with 99.9% success."

1. Science Activities



2. Safety Activities

Constraints on risk of failure (Chance Constraints):

1. *p(* Remain in [bloom region] *fails* OR Remain in [mapping region] *fails*) < 5%.
2. *p(* End in [pickup region] *fails* OR Remain in [safe region] *fails*) < .1%.

Deterministic Finite-Horizon Optimal Planning

$$\min_{u_{1:T} \in \mathbf{U}^T} J(u_{1:T})$$

Convex function

Cost function (e.g. fuel consumption)

$$s.t.$$

Discrete-time linear dynamics

$$\bigwedge_{t=0}^{T-1} x_{t+1} = Ax_t + Bu_t$$

State constraints (*Convex*)

$$\bigwedge_{t=1}^{T} \bigwedge_{i=1}^{N} h_t^{iT} x_t \leq g_t^i$$

Chance-Constrained FH Optimal Planning

$$\min_{u_{1:T} \in \mathbf{U}^T} J(u_{1:T})$$

$$s.t.$$

Stochastic dynamics

$$\bigwedge_{t=0}^{T-1} x_{t+1} = Ax_t + Bu_t + w_t$$

Gaussian distribution

$$w_t \overset{iid}{\sim} N(0, \Sigma_w)$$ Exogenous disturbance

State constraints

$$\bigwedge_{t=1}^{T} \bigwedge_{i=1}^{N_t} h_t^i x_t \leq g_t^i$$

$$\bar{x}_0 \sim N(\bar{x}_0, \Sigma_{x,0})$$ State estimation error

Chance-Constrained FH Optimal Planning

$$\min_{u_{1:T} \in \mathbf{U}^T} J(u_{1:T})$$

$$s.t.$$

Stochastic dynamics

$$\bigwedge_{t=0}^{T-1} x_{t+1} = Ax_t + Bu_t + w_t$$

$$w_t \sim N(0, \Sigma_t)$$

$$x_0 \sim N(\bar{x}_0, \Sigma_{x,0})$$

State constraints

$$\bigwedge_{t=1}^{T} \bigwedge_{i=1}^{N} h_t^{iT} x_t \leq g_t^i$$

Chance-Constrained FH Optimal Planning

$$\min_{u_{1:T} \in \mathbf{U}^T} J(u_{1:T})$$

$$s.t.$$

Stochastic dynamics

$$\bigwedge_{t=0}^{T-1} x_{t+1} = Ax_t + Bu_t + w_t$$

$$w_t \sim N(0, \Sigma_t)$$

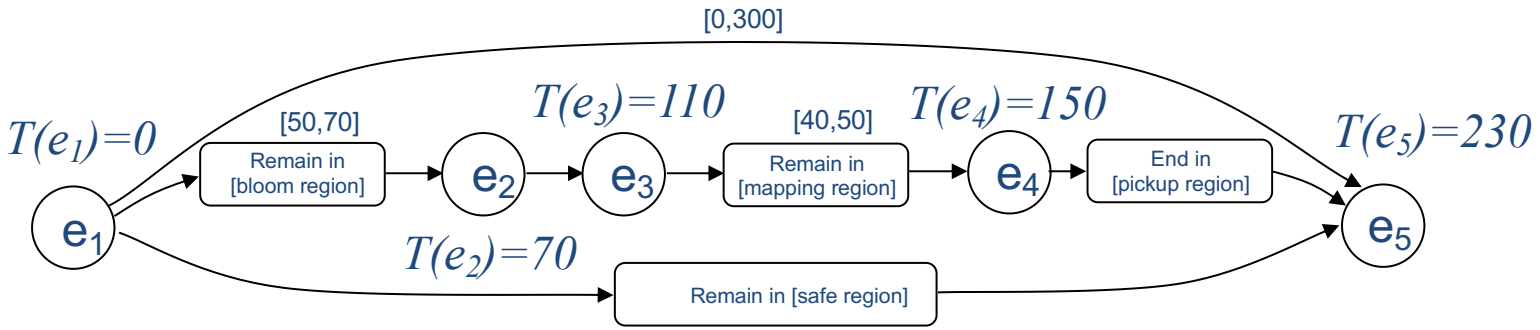
$$x_0 \sim N(\bar{x}_0, \Sigma_{x,0})$$

Chance constraint

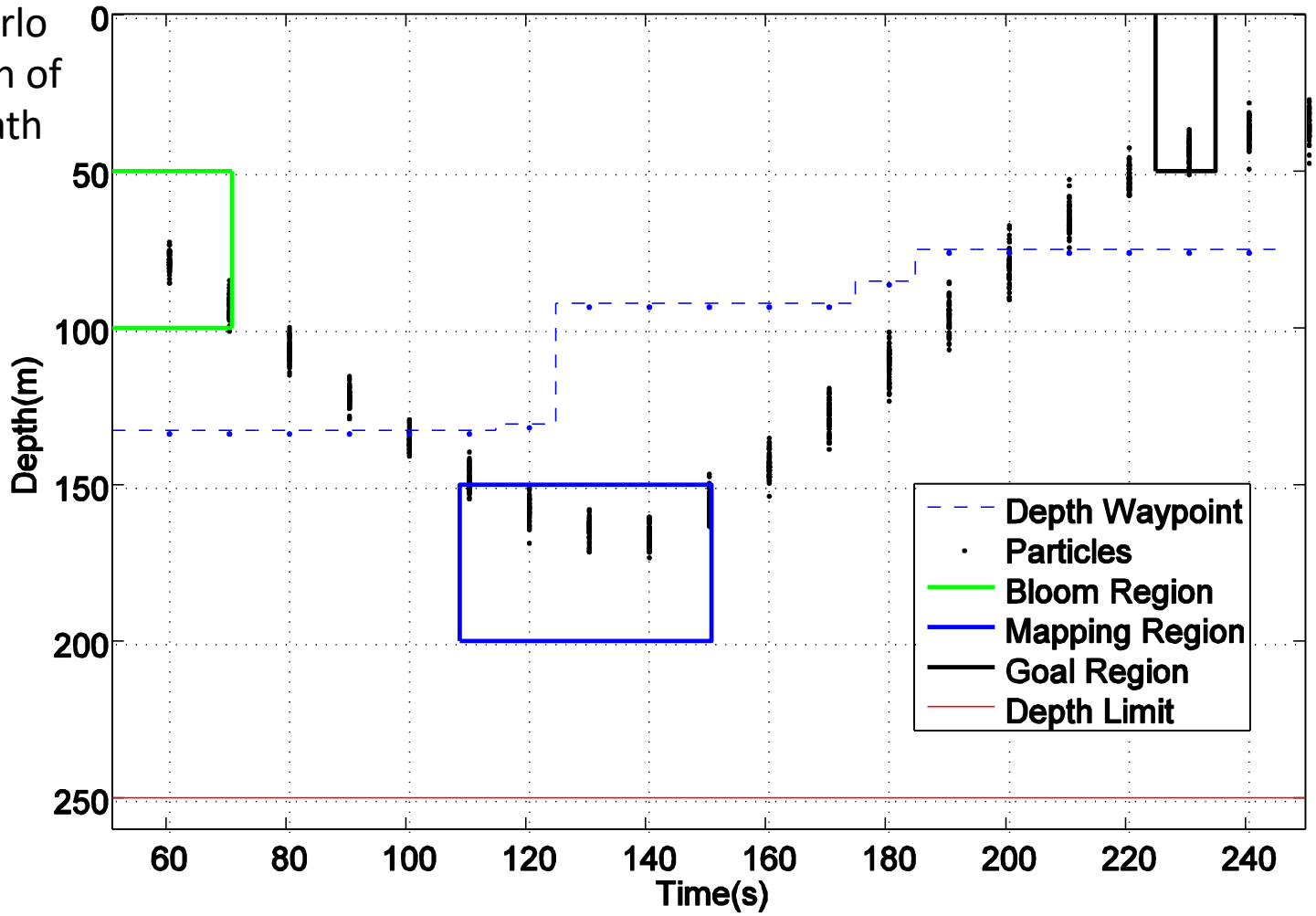
$$\Pr\left[\bigwedge_{t=1}^{T}\bigwedge_{i=1}^{N} h_t^{iT} x_t \le g_t^i\right] \ge 1 - \Delta$$

Risk bound
(Upper bound on the probability of failure)

Assume: $\Delta < 0.5$



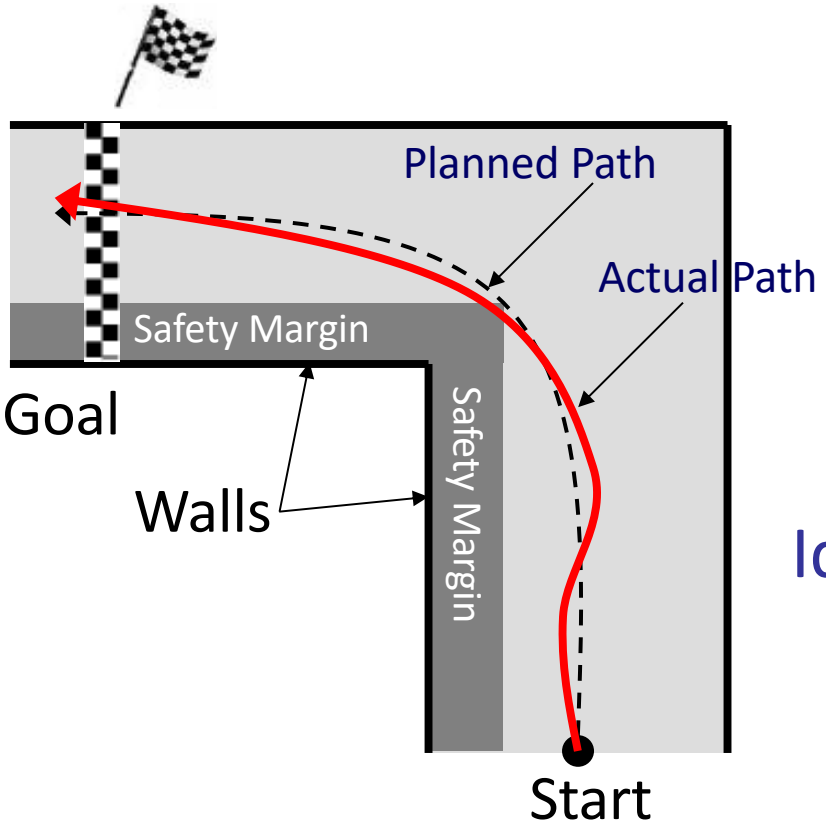
Monte Carlo Simulation of Output Path



Outline

- State Programs and State Plans
- Model Predictive Control
- State Plan Motion Planning
- Risk-aware State Plan Motion Planning
  - **Risk allocation**
  - Iterative Risk Allocation
  - Convex Risk Allocation
- Risk-aware Planning with Learned behaviors (LaPLASS)
- Avoiding Obstacles (Appendix)

Example: Race Car Path Planning



Problem

Find the fastest path to the goal, while limiting the probability of crash throughout the race to 0.1%

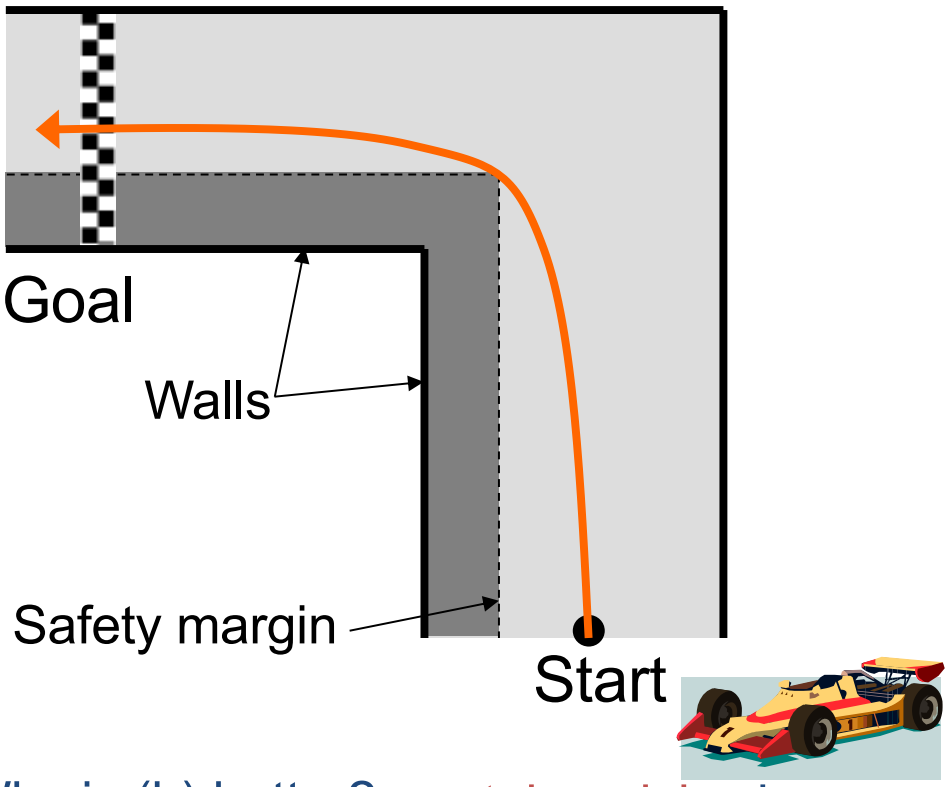
Risk bound

Idea:

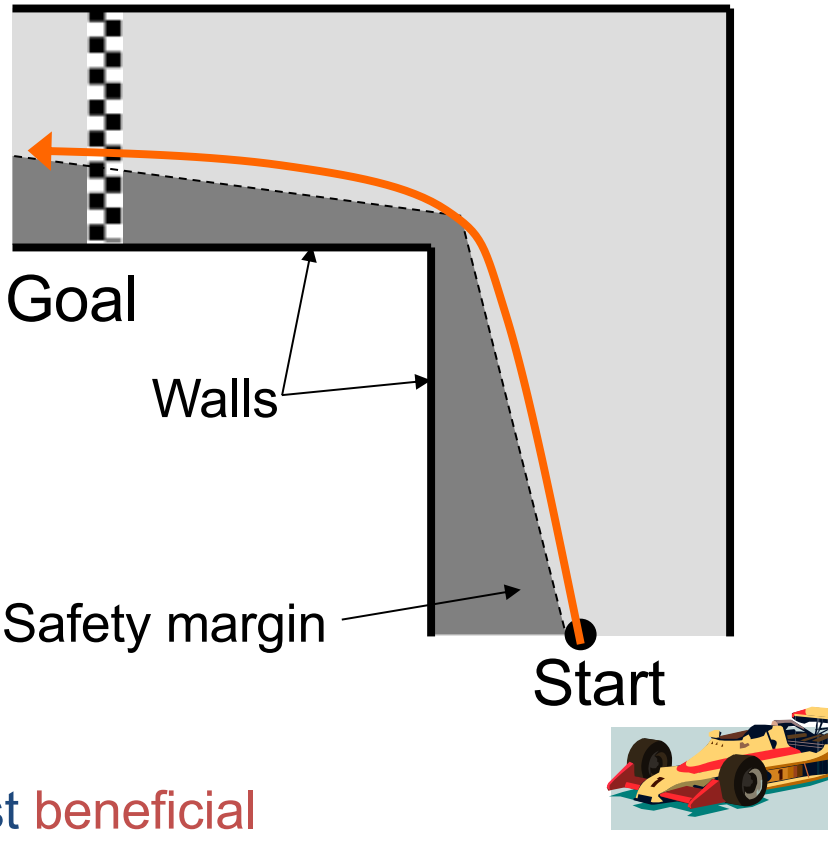
1. Create safety margin that satisfies risk bound from start to goal.
2. Reduce to simpler, deterministic optimization problem.
3. Build safety margin by spending risk at every time step.

A good safety margin
satisfies risk bound and maximizes expected utility

(a) Uniform width safety margin



(b) Uneven width safety margin



Why is (b) better? → takes risk where most beneficial

[Ono & Williams, AAAI 08]

Approach: Algorithmic Risk Allocation

- Allocates risk across time and constraints

Outline

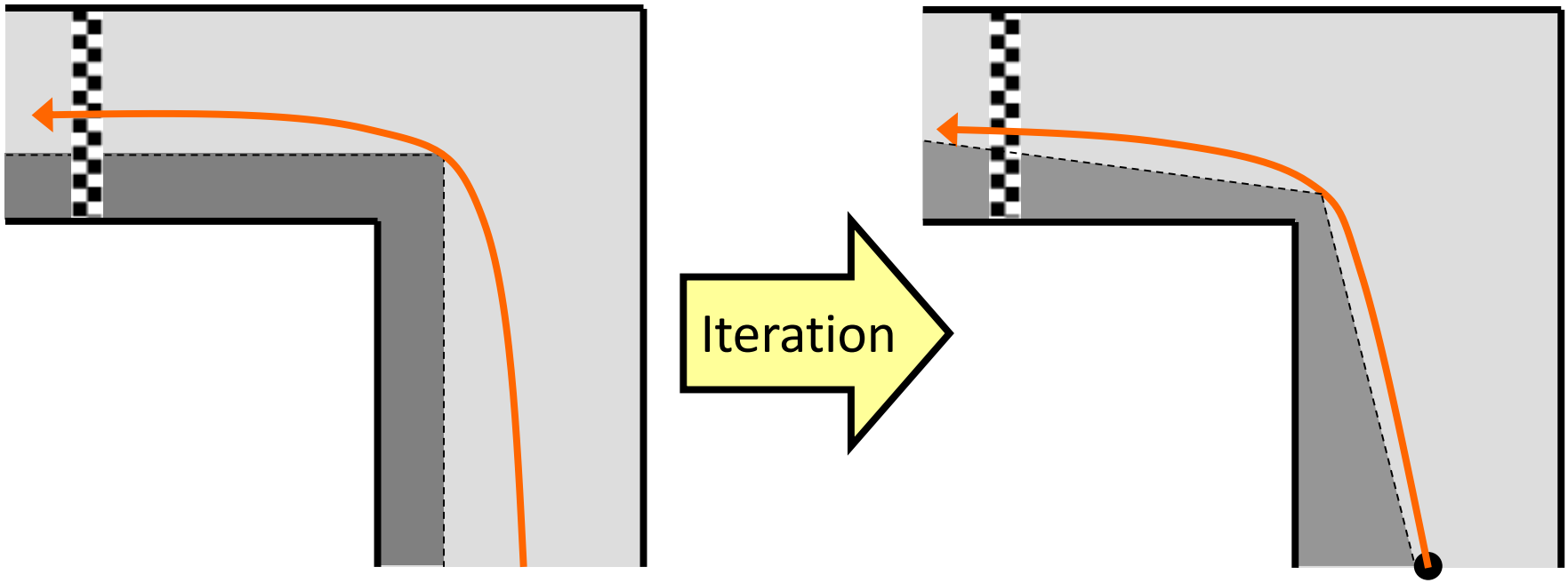
- State Programs and State Plans
- Model Predictive Control
- State Plan Motion Planning
- Risk-aware State Plan Motion Planning
  - Risk Allocation
  - **Iterative Risk Allocation**
  - Convex Risk Allocation
- Risk-aware Planning with Learned behaviors (LaPLASS)
- Avoiding Obstacles (Appendix)

Iterative Risk Allocation (IRA) Algorithm

- Descent algorithm

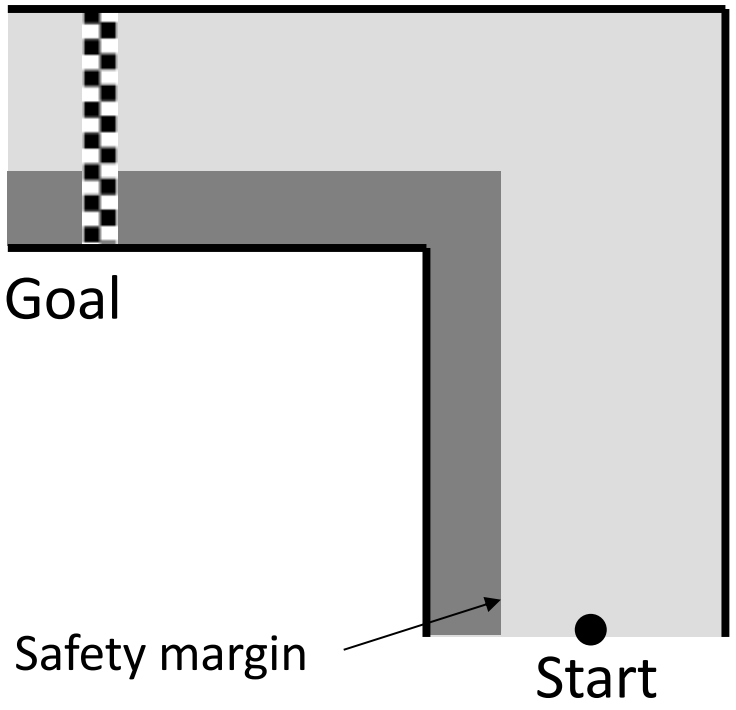
$$\bar{J}^*(\boldsymbol{\delta}_0) \geq \bar{J}^*(\boldsymbol{\delta}_1) \geq \bar{J}^*(\boldsymbol{\delta}_2) \cdots$$

(Refer to papers for proof)



- Starts with a suboptimal risk allocation.
- Improves allocation through iteration.

Iterative Risk Allocation Algorithm



Algorithm IRA

1 Initialize with arbitrary risk allocation

2 Loop

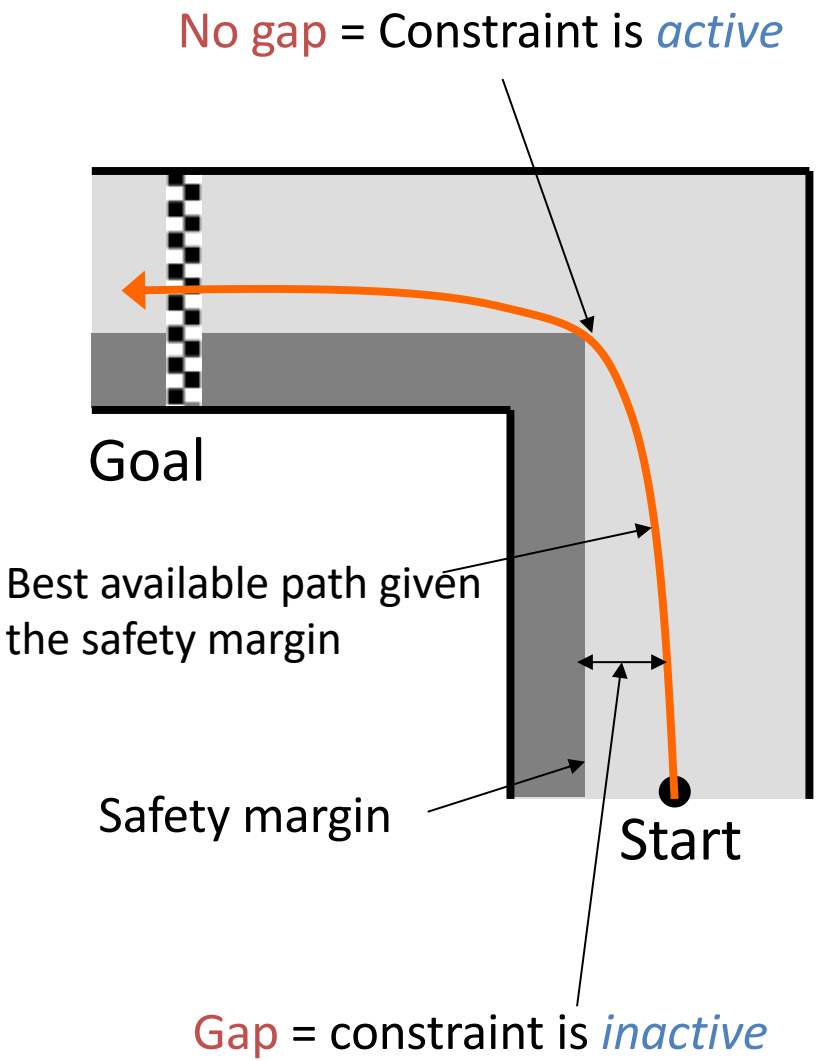
3 Compute the best available path given the current risk allocation

4 Decrease the risk where the constraint is inactive

5 Increase the risk where the constraint is active

6 End loop

Iterative Risk Allocation Algorithm



Algorithm IRA

1 Initialize with arbitrary risk allocation

2 Loop

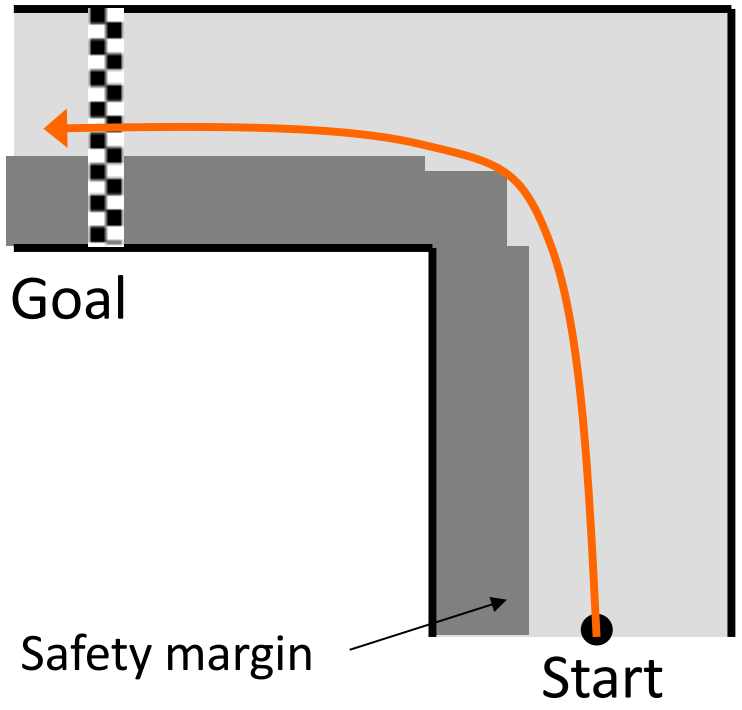
3 Compute the best available path given the current risk allocation

4 Decrease the risk where the constraint is inactive

5 Increase the risk where the constraint is active

6 End loop

Iterative Risk Allocation Algorithm



Algorithm IRA

1 Initialize with arbitrary risk allocation

2 Loop

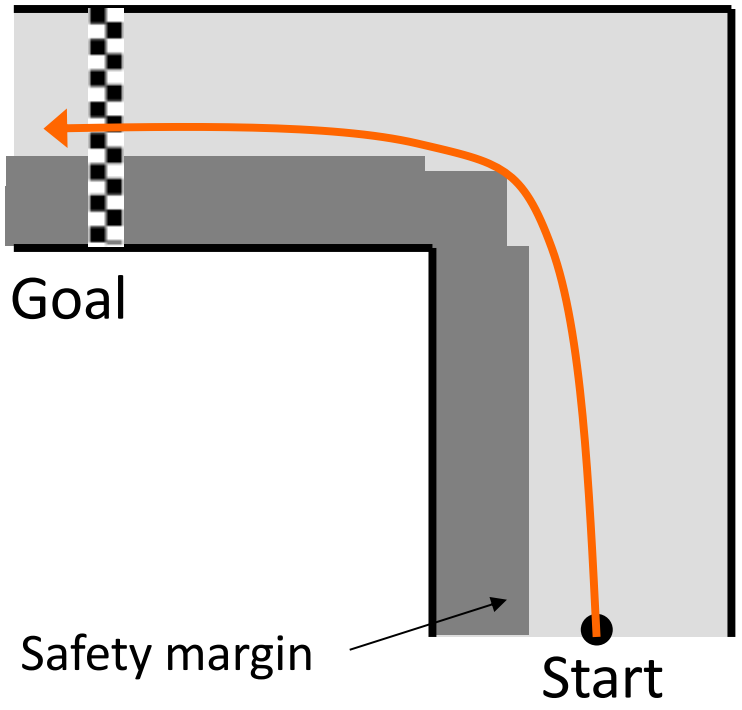
3 Compute the best available path given the current risk allocation

4 Decrease the risk where the constraint is **inactive**

5 Increase the risk where the constraint is active

6 End loop

Iterative Risk Allocation Algorithm



Algorithm IRA

1 Initialize with arbitrary risk allocation

2 Loop

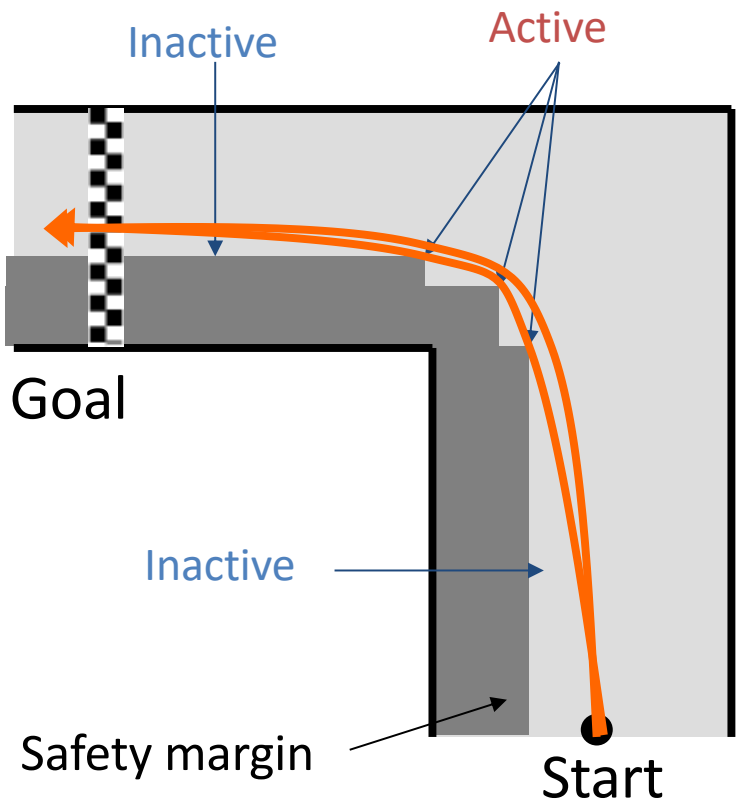
3 Compute the best available path given the current risk allocation

4 Decrease the risk where the constraint is inactive

5 Increase the risk where the constraint is **active**

6 End loop

Iterative Risk Allocation Algorithm



Algorithm IRA

1 Initialize with arbitrary risk allocation

2 Loop

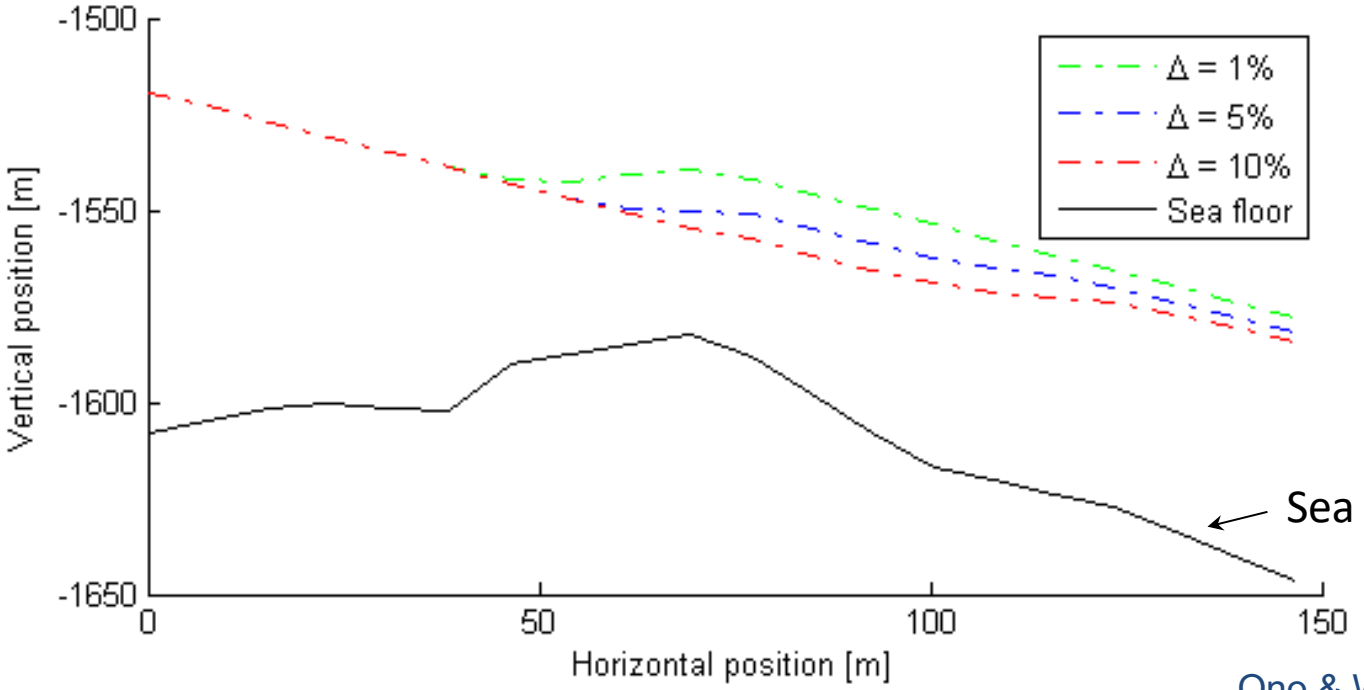
3 Compute the best available path given the current risk allocation

4 Decrease the risk where the constraint is inactive

5 Increase the risk where the constraint is active

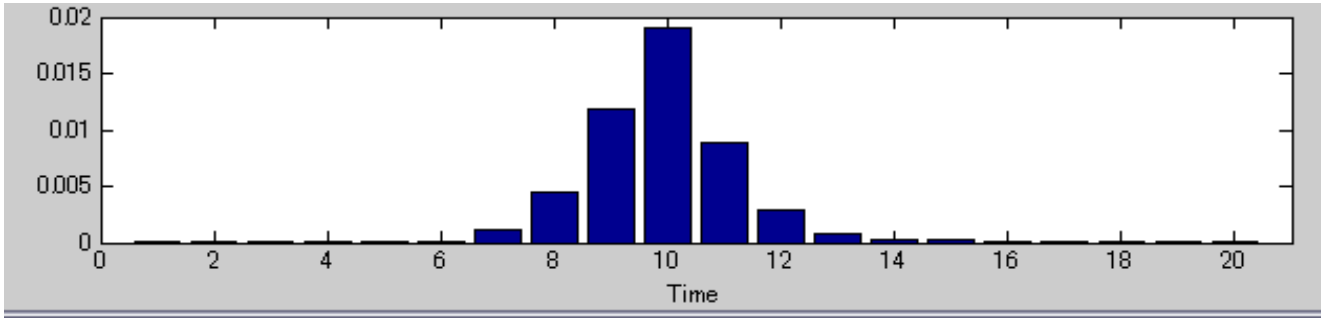
6 End loop

Monterey Bay Mapping Example



Ono & Williams, AAAI 08

Risk allocation ($\Delta = 5\%$)



Outline

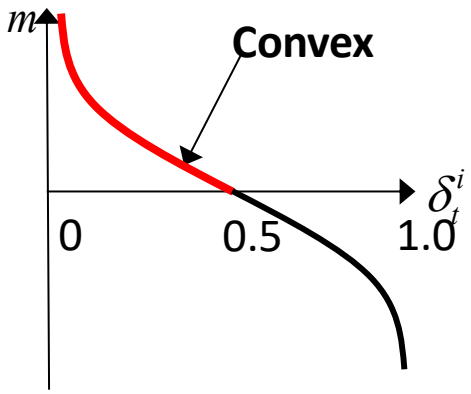
- State Programs and State Plans
- Model Predictive Control
- State Plan Motion Planning
- Risk-aware State Plan Motion Planning
  - Iterative Risk Allocation
  - **Convex Risk Allocation**
- Risk-aware Planning with Learned behaviors (LaPLASS)
- Avoiding Obstacles (Appendix)

Solve Optimally using a Convex Program over Risk Allocation and Decision Variables

$$\min_{\boldsymbol{\delta}} \min_{u_{1:T} \in \mathbf{U}^T} J(u_{1:T})$$

$$s.t.$$

$$\bigwedge_{t=0}^{T-1} x_{t+1} = A\bar{x}_t + Bu_t$$



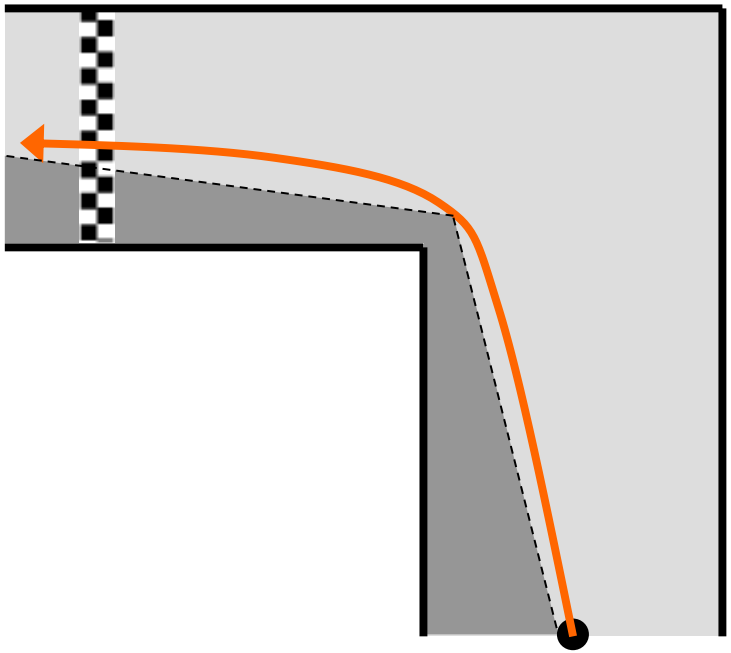
$$\bigwedge_{t=1}^{T} \bigwedge_{i=1}^{I} h_t^{iT} \bar{x}_t \le g_t^i - m_t^i\left(\delta_t^i\right)$$

Convex if $\delta < 0.5$

$$\sum_{t,i} \delta_t^i \le \Delta$$

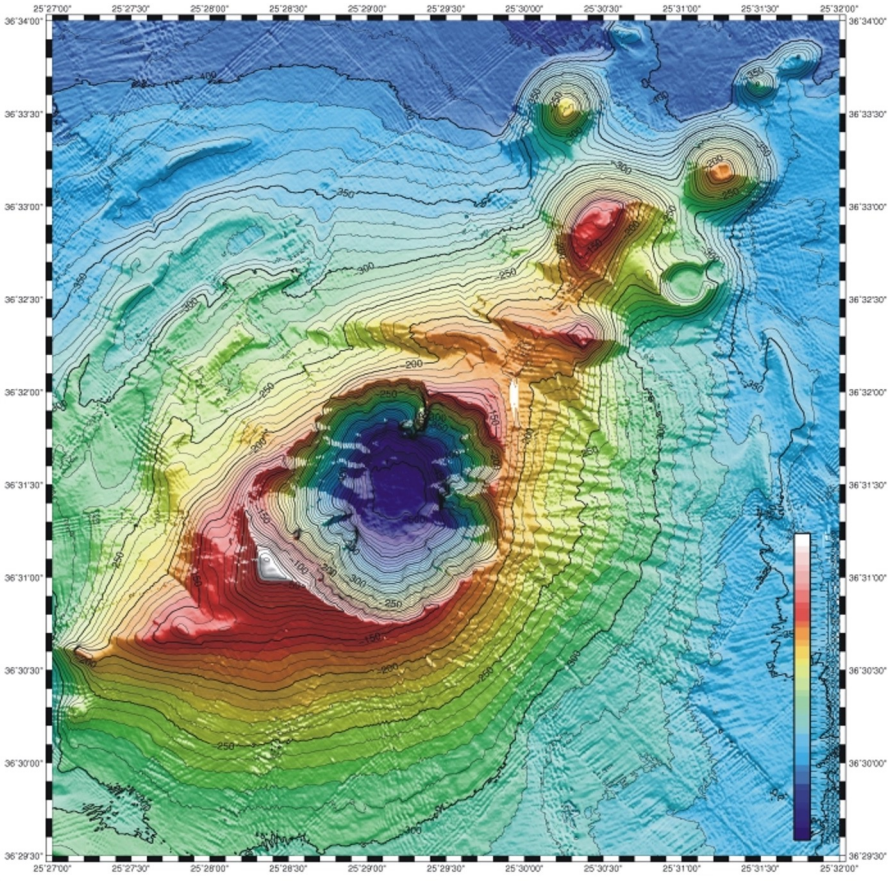
Summary:
Determinizing through Risk Allocation

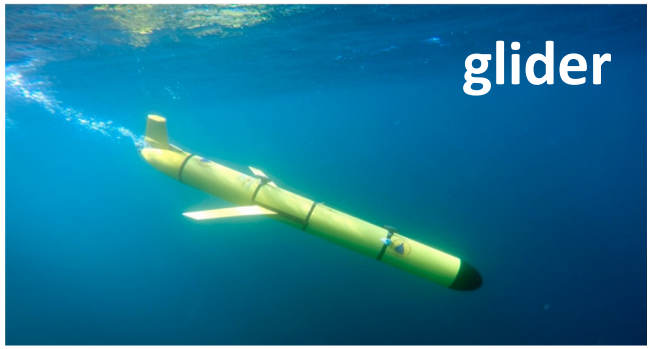
1. IRA: reallocate risk in outer loop.
2. CRA: use closed form solver to reallocate risk.



Outline

- State Programs and State Plans
- Model Predictive Control
- State Plan Motion Planning
- Risk-aware State Plan Motion Planning
- **Risk-aware Planning with Learned behaviors (LaPLASS)**
- Avoiding Obstacles (Appendix)





How do we reason over non-linear dynamics?

Jasour et al

How do we reason over non-Gaussian distributions?

How do we plan with **learned** dynamics?

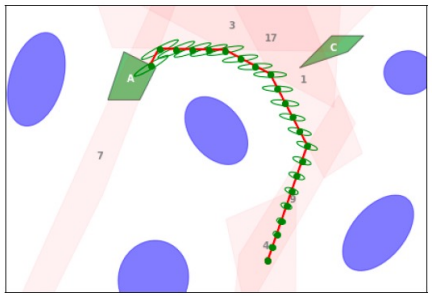
Reeves PhD

Limitations of Existing Work

Robust Control

Uniformly buffer hazards or trajectory using worst-case uncertainty estimate

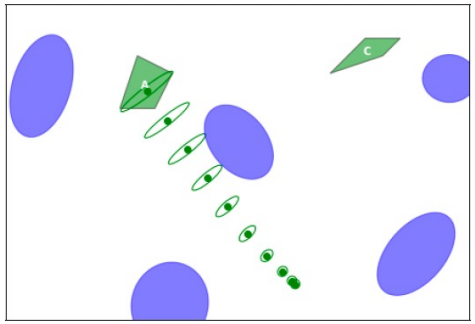
- ☑ computationally **efficient**
- ☒ overly **conservative**



Risk Allocation

(Ono, Blackmore et al, 2008)
Reformulate into deterministic problem using safety margins

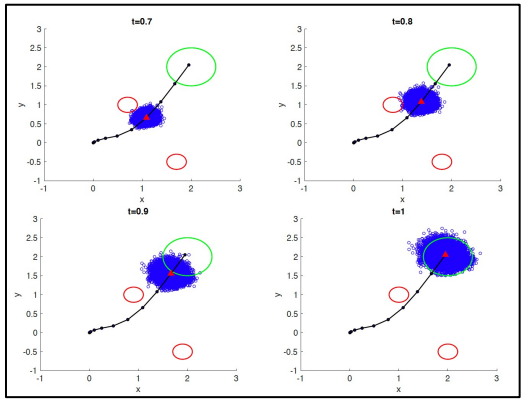
- ☑ computationally **efficient**
- ☒ Restricted to **linear** dynamics



Sum of Moments (Han, Jasour et al, 2022)

Nonlinear optimization by converting chance constraints into deterministic constraints on moments

- ☒ **not** computationally efficient
- ☑ uses **nonlinear** dynamics



All assume known model

"Generate & Validate" Approach

Reeves PhD 24

linear dynamics is inaccurate

Competing needs

planning over nonlinear dynamics is expensive

Insight: Recent **risk assessment methods** **compute risk** of an uncertain trajectory **quickly** *(Wang et al. 2020)*

Our Approach

Fast Candidate Generator
(deterministic convex trajectory optimization)

candidate trajectory plan →

← *safety constraint to ↓ risk of failure*

Accurate Safety Validator
(non-linear, non-Gaussian risk assessment & safety constraint extraction)

☑ efficiently computes trajectories using linear dynamics

☑ uses nonlinear dynamics to verify safety of candidate trajectory

"Generate & Validate" Approach

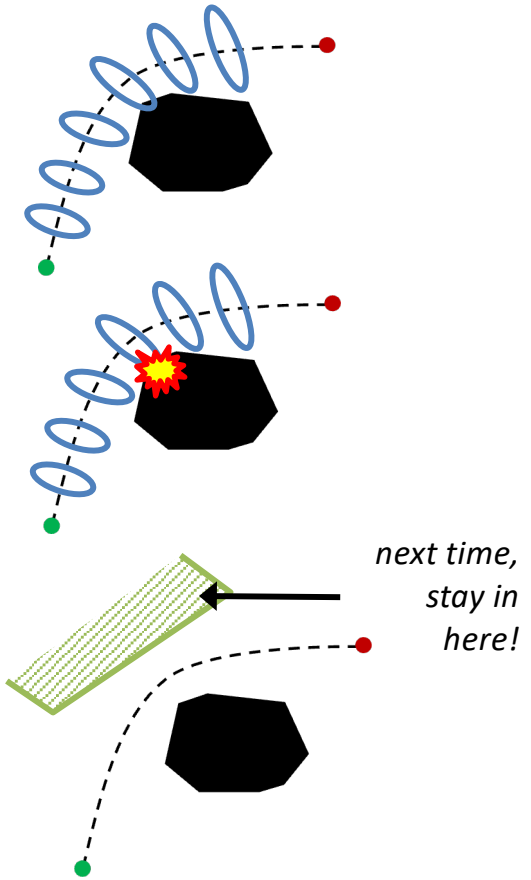
Candidate Generator
(deterministic convex trajectory optimization)

safety constraint to ↓ risk of failure

candidate trajectory plan

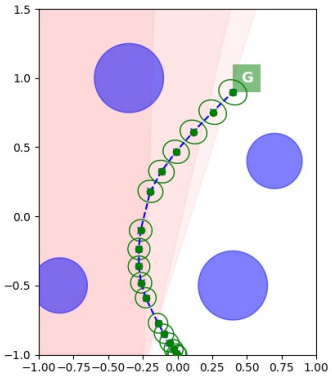
Safety Validator
(risk assessment & safety constraint etraction)

1. Quantifies uncertainty around candidate trajectory
2. Computes risk of constraint violation
3. If too risky, generates convex safety constraint to decrease risk.



Experimental Results

For nonlinear risk-bounded planning with closed-form dynamics.
"Generate & Validate" is >10x more efficient than Sum of Moments (Han et al.)
without significant loss of safety or optimality.



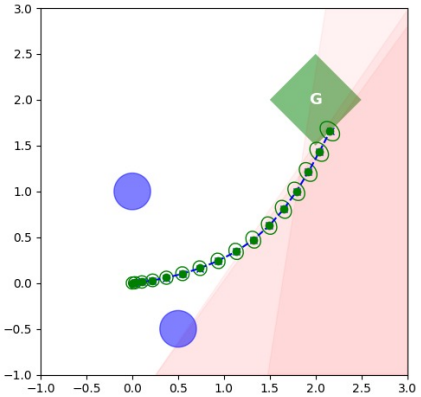
Planar AUV

$$x_{t+1} = x_t + \Delta T(v_t + \omega_{vt})\cos(\theta_t + \omega_{\theta t})$$
$$y_{t+1} = y_t + \Delta T(v_t + \omega_{vt})\sin(\theta_t + \omega_{\theta t})$$

Ground Vehicle
(w/ dynamic obstacles)

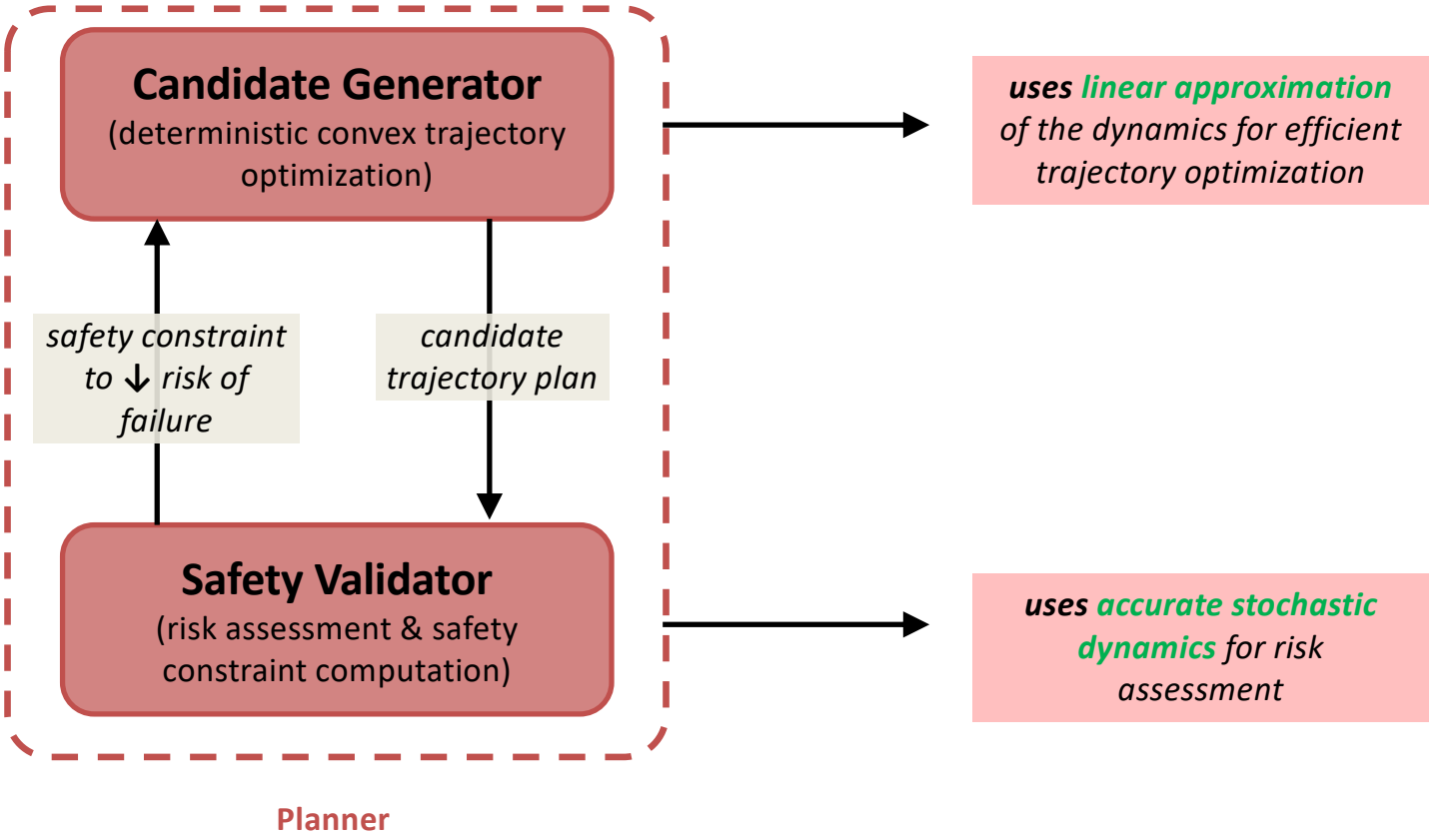
$$x_{t+1} = x_t + \Delta T v_t \cos(\theta_t)$$
$$y_{t+1} = y_t + \Delta T v_t \sin(\theta_t)$$
$$v_{t+1} = v_t + \Delta T(a_t + \omega_{v_t})$$
$$\theta_{t+1} = \theta_t + \Delta T(u_t + \omega_{\theta_t})$$

Reach goal while minimizing control effort and satisfying a risk bound of **Δ** = 0.1

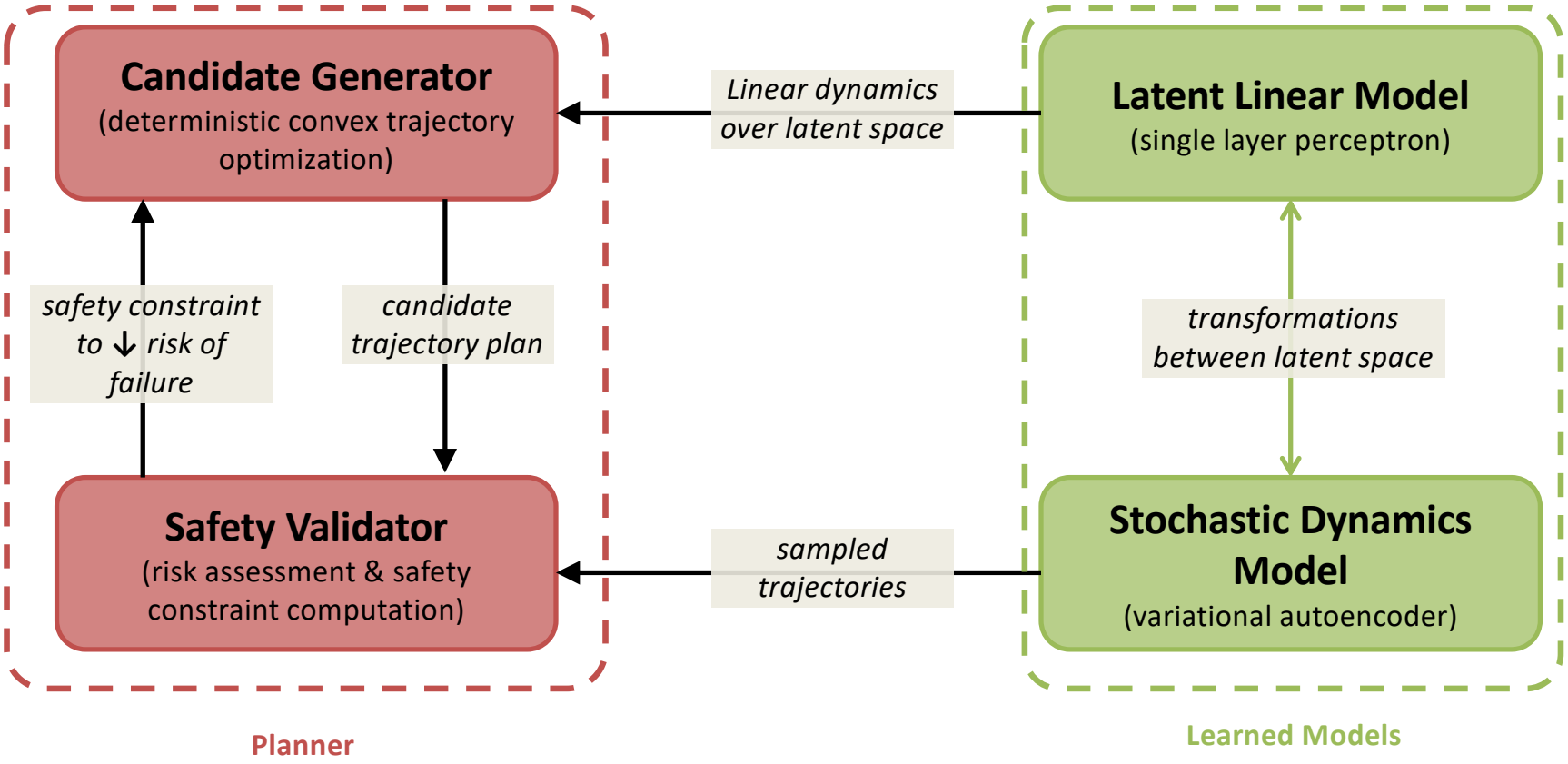


Metrics	Scenario: Planar AUV		Scenario: Ground Vehicle	
	(Han, Jasour et al, 2022)	**G & V**	(Han, Jasour et al, 2022)	**G & V**
Solve Time	96.86 s	**10.45 s**	300 s	**5.43 s**
Trajectory Cost	60.35	80.52	463.55	397.49
Risk Incurred	0.00782	0.0326	0.0017	<0.001

Models Needed for Risk-aware Planning

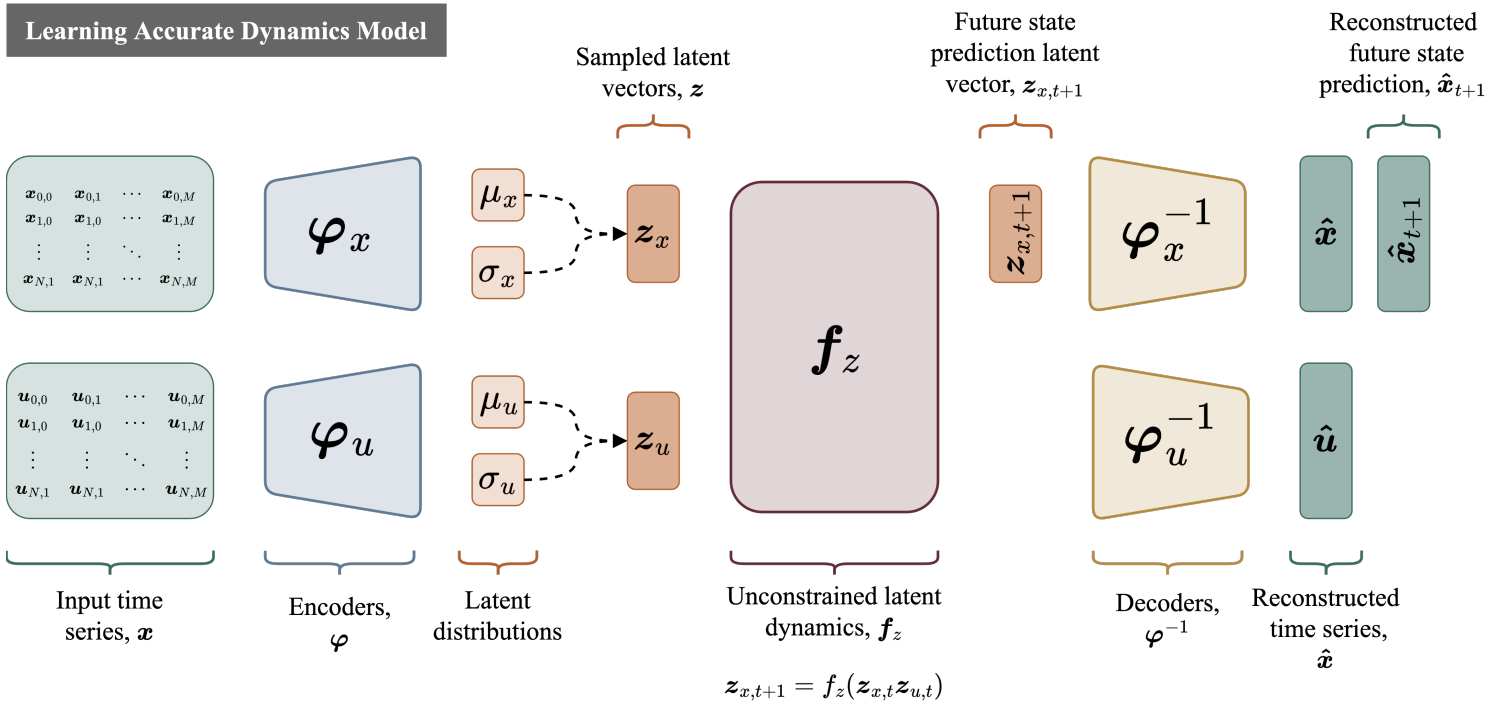


Risk-aware Planning with Learned Behaviors



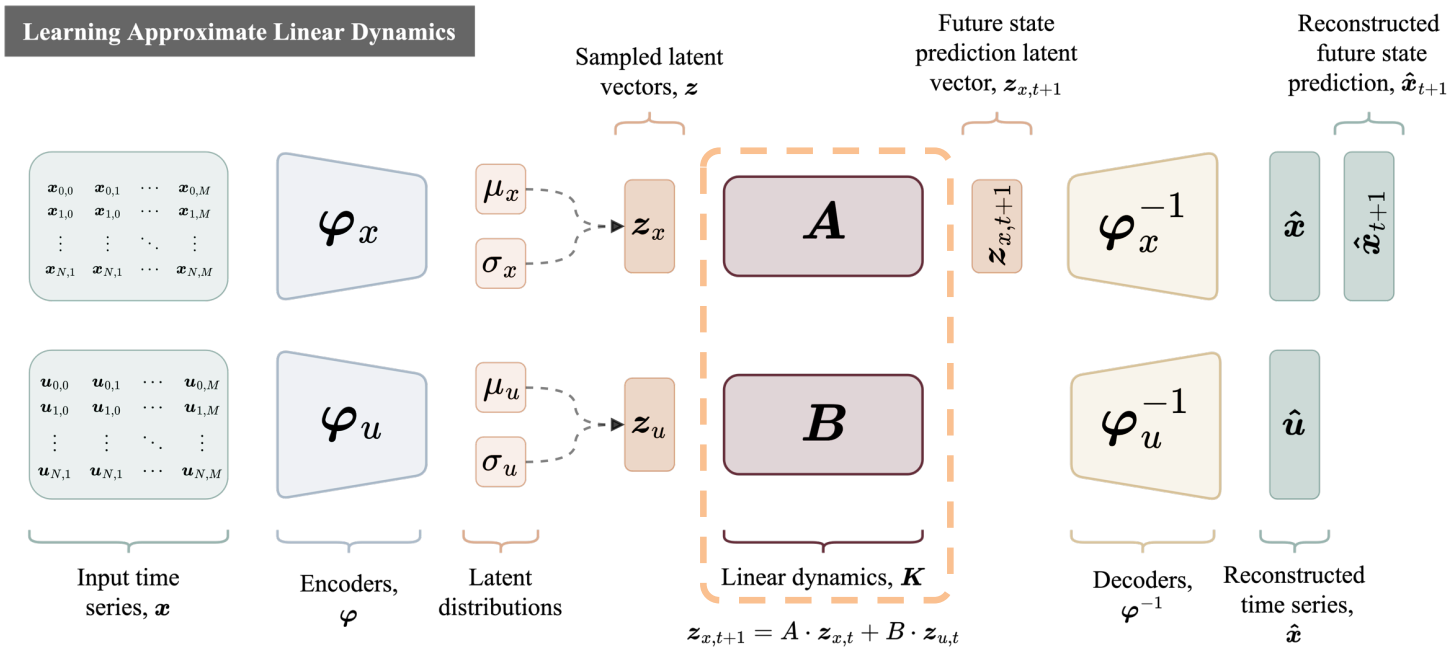
Variational Autoencoder (VAE) for Stochastic Dynamics

$$L_{VAE} = \overbrace{\| \boldsymbol{x} - \hat{\boldsymbol{x}} \|}^{\text{reconstruction loss}} + \overbrace{\| \boldsymbol{x}_{t+m} - \hat{\boldsymbol{x}}_{t+m} \|}^{\text{prediction loss}} + \underbrace{D_{KL}(\mathcal{N}_x)}_{\text{KL divergence}}$$



VAE for Latent Linear Dynamics

$$L_K = \overbrace{\| \boldsymbol{z}_{t+m} - K^m \boldsymbol{z}_t \|}^{\text{linearity loss}} + \underbrace{\| \boldsymbol{x}_{t+m} - \hat{\boldsymbol{x}}_{t+m} \|}_{\text{prediction loss}}$$



fix VAE latent space → only train **single layer linear dynamics model**

Outline

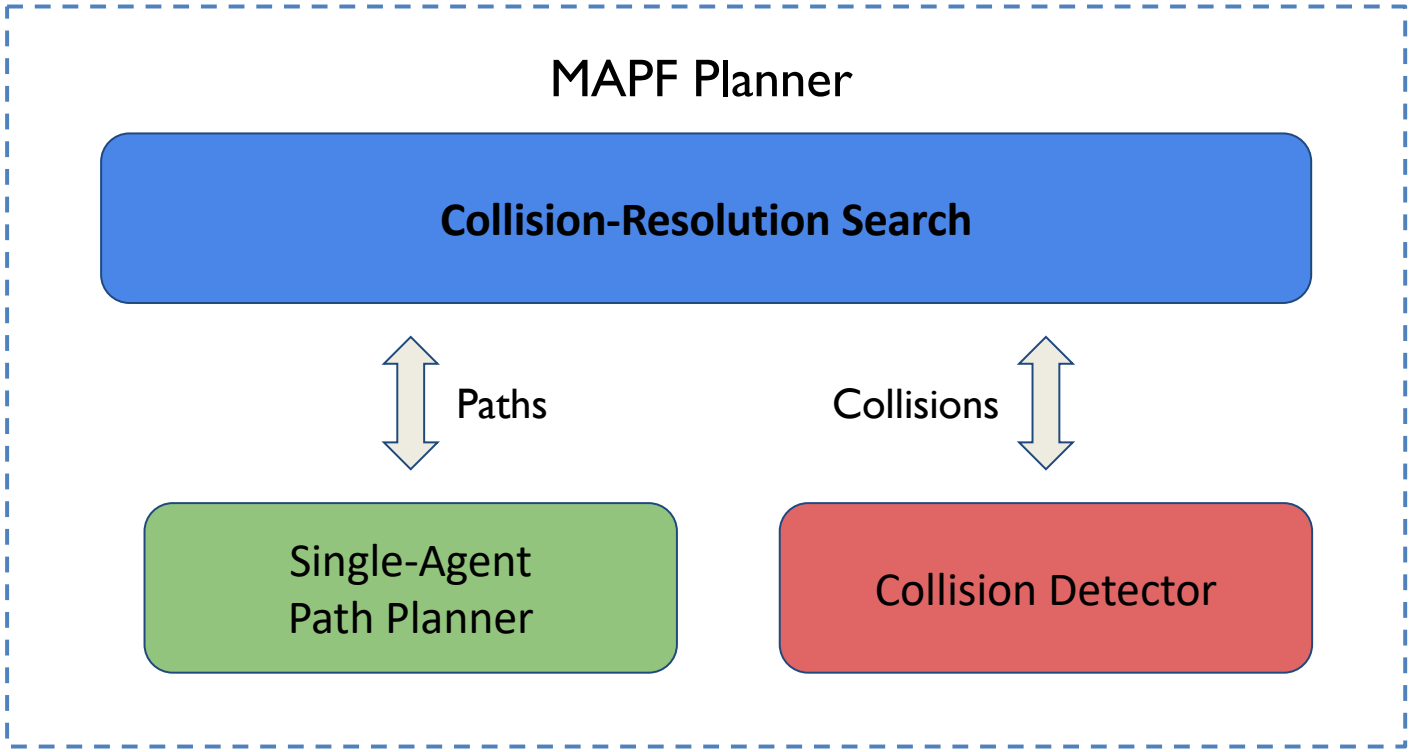
- State Programs and State Plans
- Model Predictive Control
- State Plan Motion Planning
- Risk-aware State Plan Motion Planning
- Risk-aware Planning with Learned Behaviors (LaPLASS)
- Avoiding Obstacles (Appendix)
  - **Large numbers of agents**
  - Large numbers of obstacles

Multi-Arm Assembly

Chen PhD 22, Parimi et al. in progress

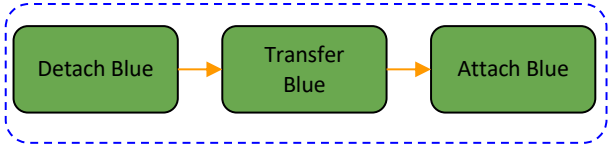
Multi Agent Pathfinding (MAPF)

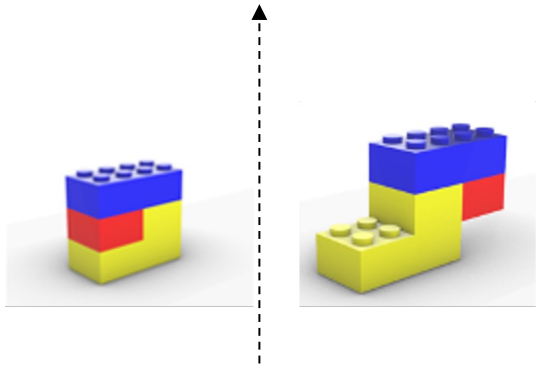
Key Insight - Factoring



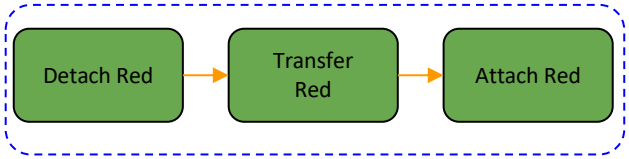
Lego Assembly: Sub-Tasks

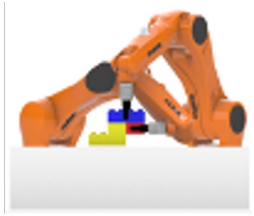
Assemble Blue



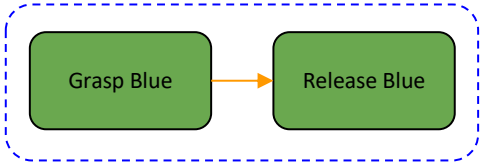


Assemble Red



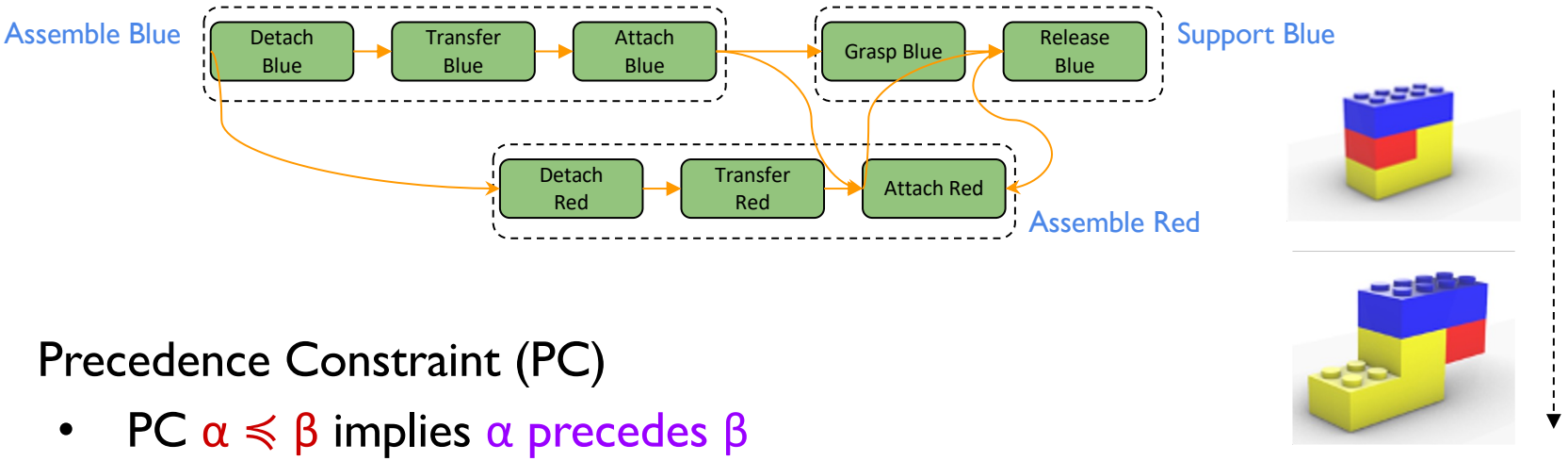


Support Blue



- Sub-Tasks
  - sequences of motion primitives (poses) to be achieved by a suitable robot
  - ➢ state plans

- Task Plan is a state plan of sub-tasks with precedence constraints

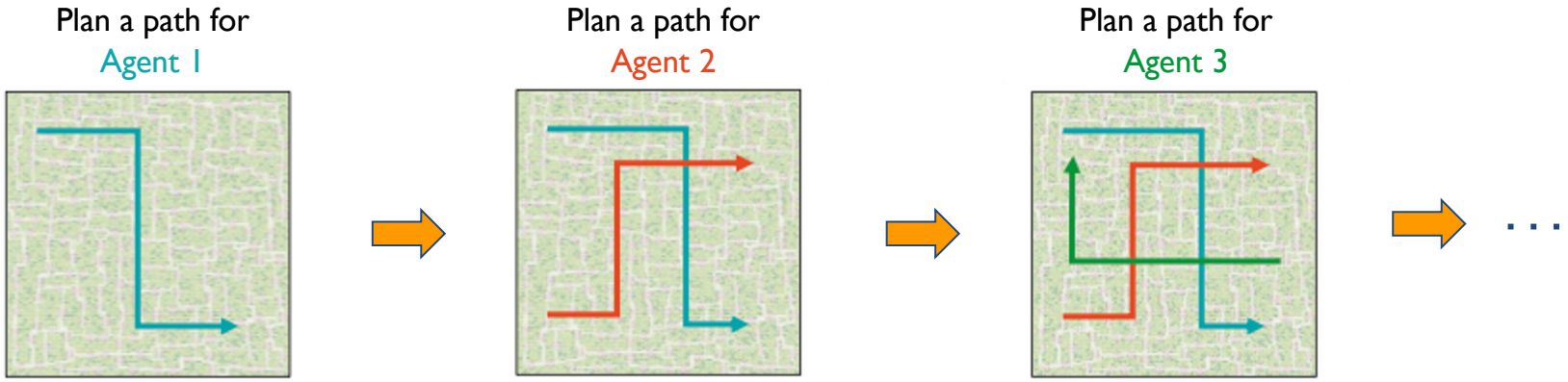


- Precedence Constraint (PC)
  - PC $\alpha \preccurlyeq \beta$ implies α precedes β

State Plan Priority-Based Search (PBS)

- PBS: Given agent priority order,

  agents with lower priority must avoid collision with higher priority agents



- For State Plans,
  - a course-grain, agent priority order leads to infeasibility

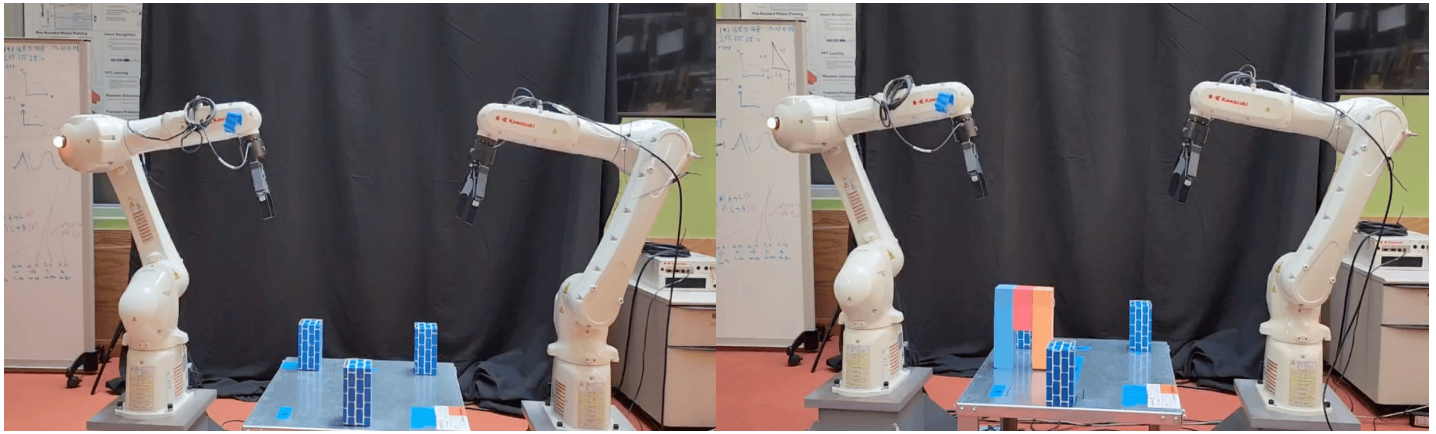
    achieve state-plan for Agent 1, state-plan for Agent 2 . . .
  - A fine-grain, priority order of individual episodes (goals) leads to success!

    achieve Episode 1 for Agent 1, achieve Episode 3 for Agent 2 . . .

MAPF-PC *[Zhang, Chen, Li, Williams & Koenig, AAMAS 22]*

State Plan Motion Planning Using MAPF-PC

Cooperative assembly for different
object orientations and ***cluttered spaces***

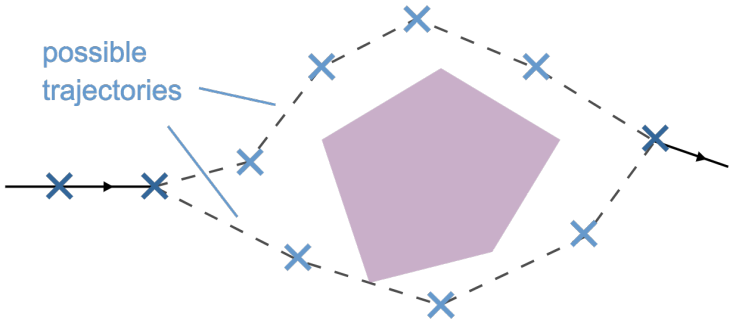


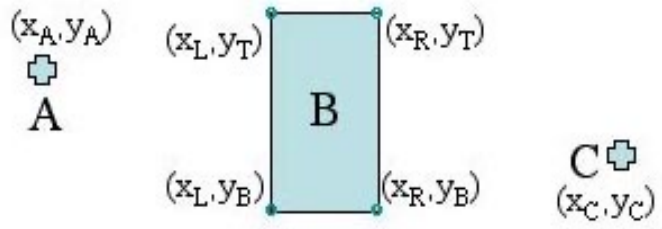
MAPF-PC *[Zhang, Chen, Li, Williams & Koenig, AAMAS 22]*

Outline

- State Programs and State Plans
- Model Predictive Control
- State Plan Motion Planning
- Risk-aware State Plan Motion Planning
- Risk-aware Planning with Learned Behaviors (LaPLASS)
- Avoiding Obstacles (Appendix)
  - Large numbers of agents
  - **Large numbers of obstacles**

Encoding Obstacle Avoidance Using Choice





$$\textit{Minimize } f(x)$$

$$\textit{Subject to } \; g(x) \leq 0$$

$$x_t \leq x_L \vee x_t \geq x_R \vee y_t \leq y_B \vee y_t \geq y_T, \; \forall t = 1, \ldots, n$$

Non-convex (combinatorial)!

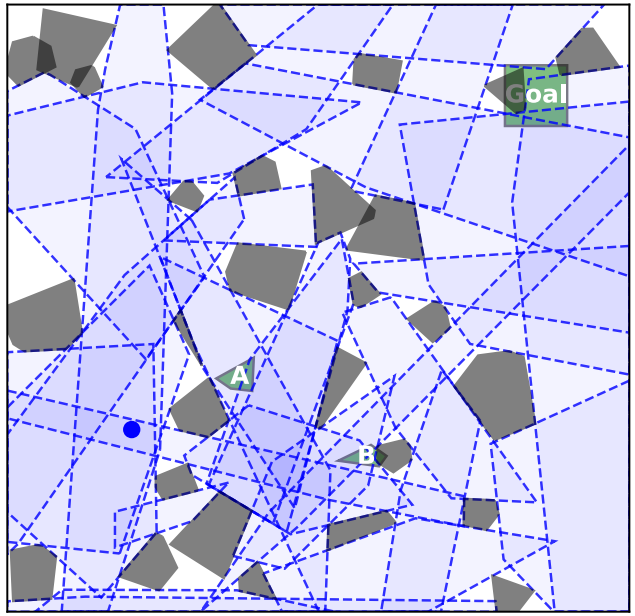
Solving Problems with Choice is Hard!

Idea:

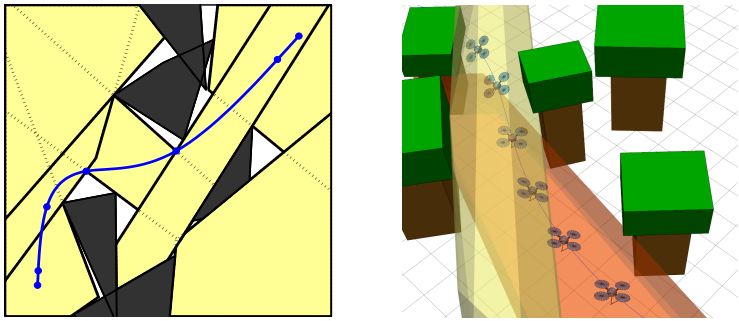
- **Don't** use obstacle constraints (no binary variables or disjuncts)
- **Do** force robots to remain in sequence of convex, obstacle-free regions

➢ Apply **convex path planner** once given sequence of "safe" regions

➢ How do we find the best sequence?



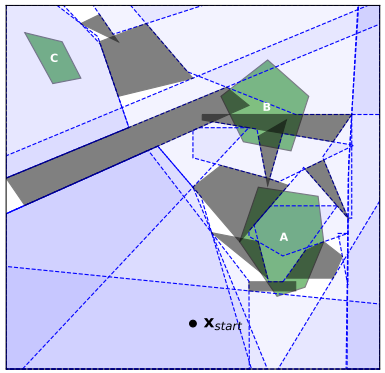
Compute safe regions using IRIS (Deits & Tedrake 2014)



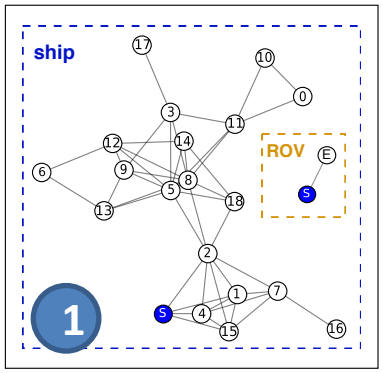
Deits, Tedrake (2015)

ScottyPath

safe regions

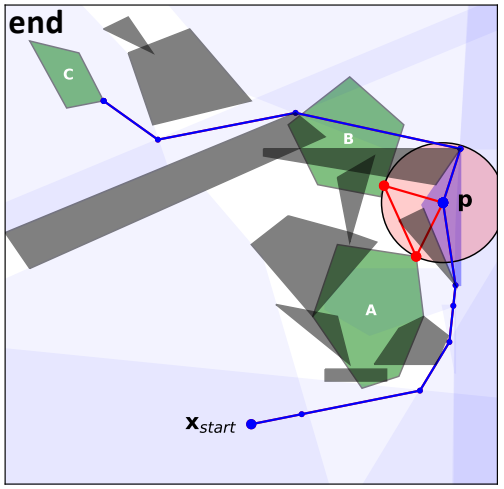


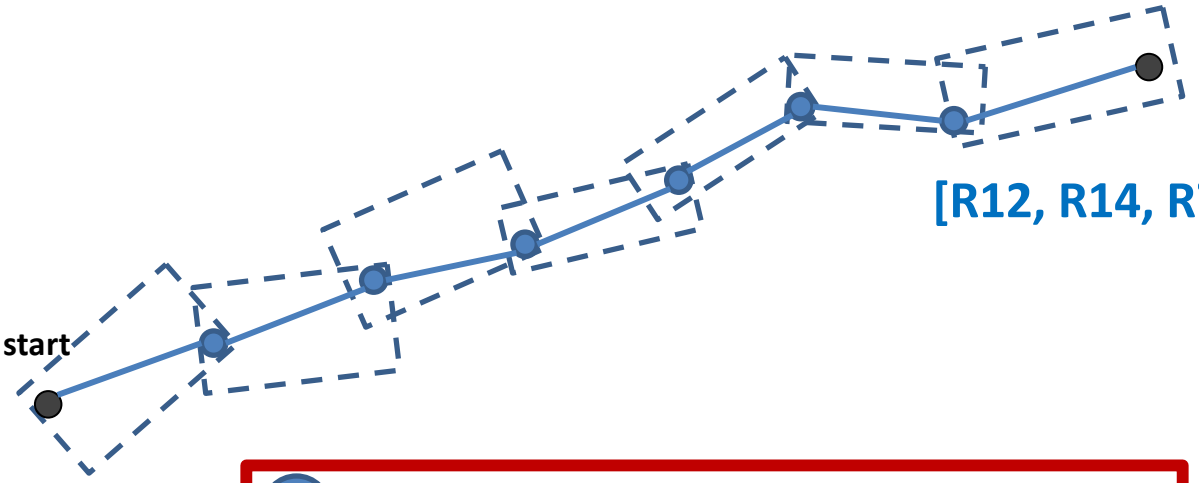
connectivity graph



precomputes using IRIS algorithm

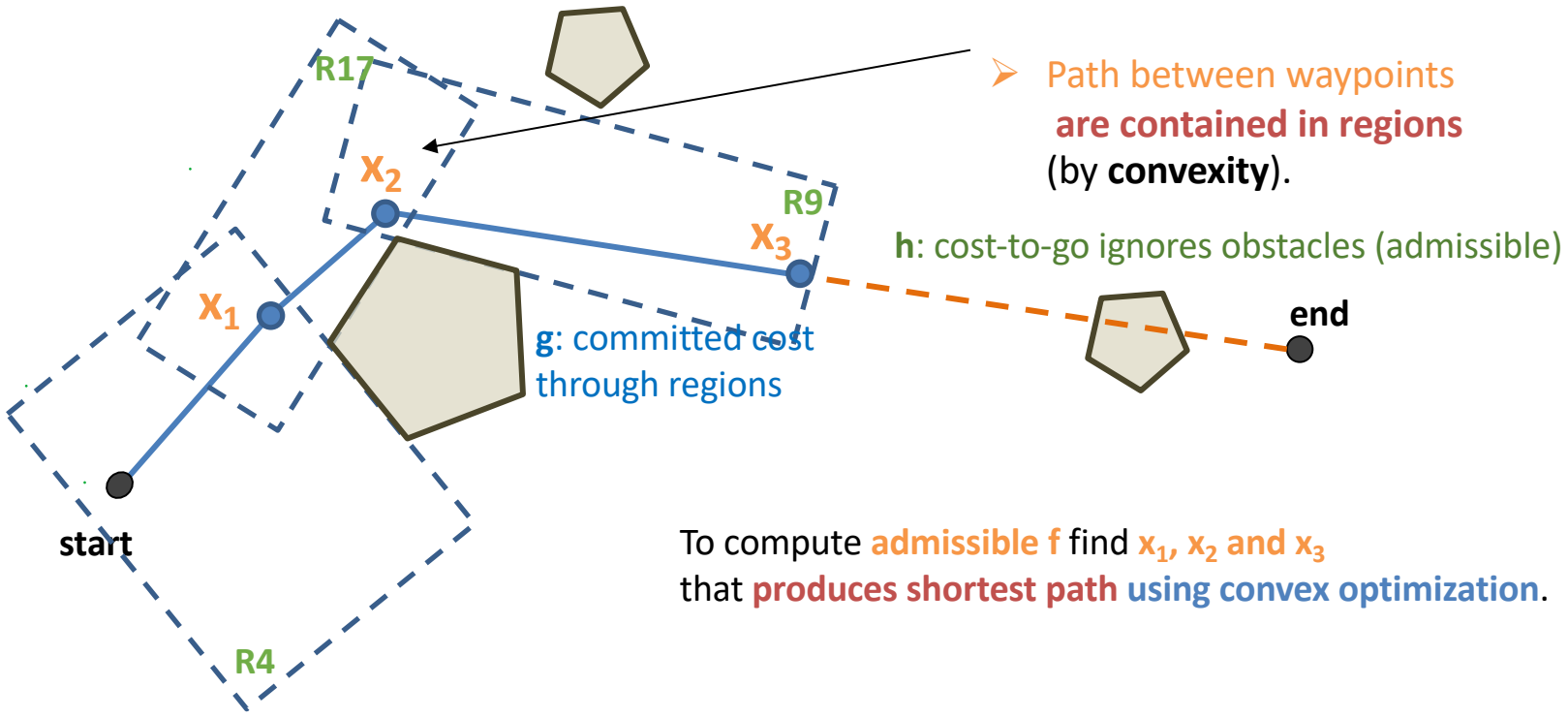
3 Finds **optimal trajectory** through region sequence **using ScottyConvexPath**





2 Extracts optimal **Region Sequence** from graph **using A***

A* evaluates f by finding shortest path through candidate sequence, e.g., [R4, R17, R9]



A* queues [R4,R17,R9] with priority **f=g + h**

Finding region sequence in ScottyPath

start

↓

Region 12

↓

Region 14

↓

Region 7

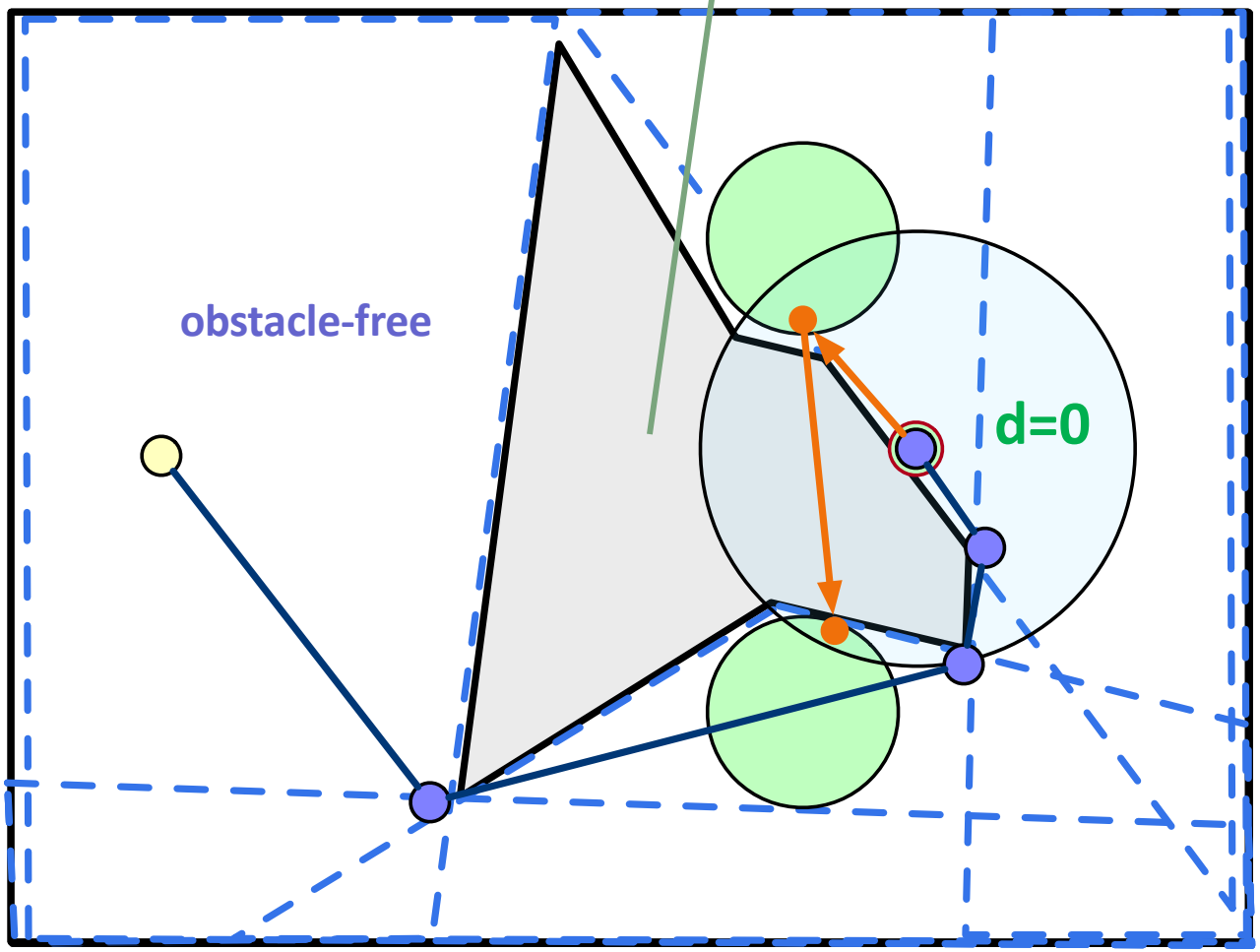
↓

…

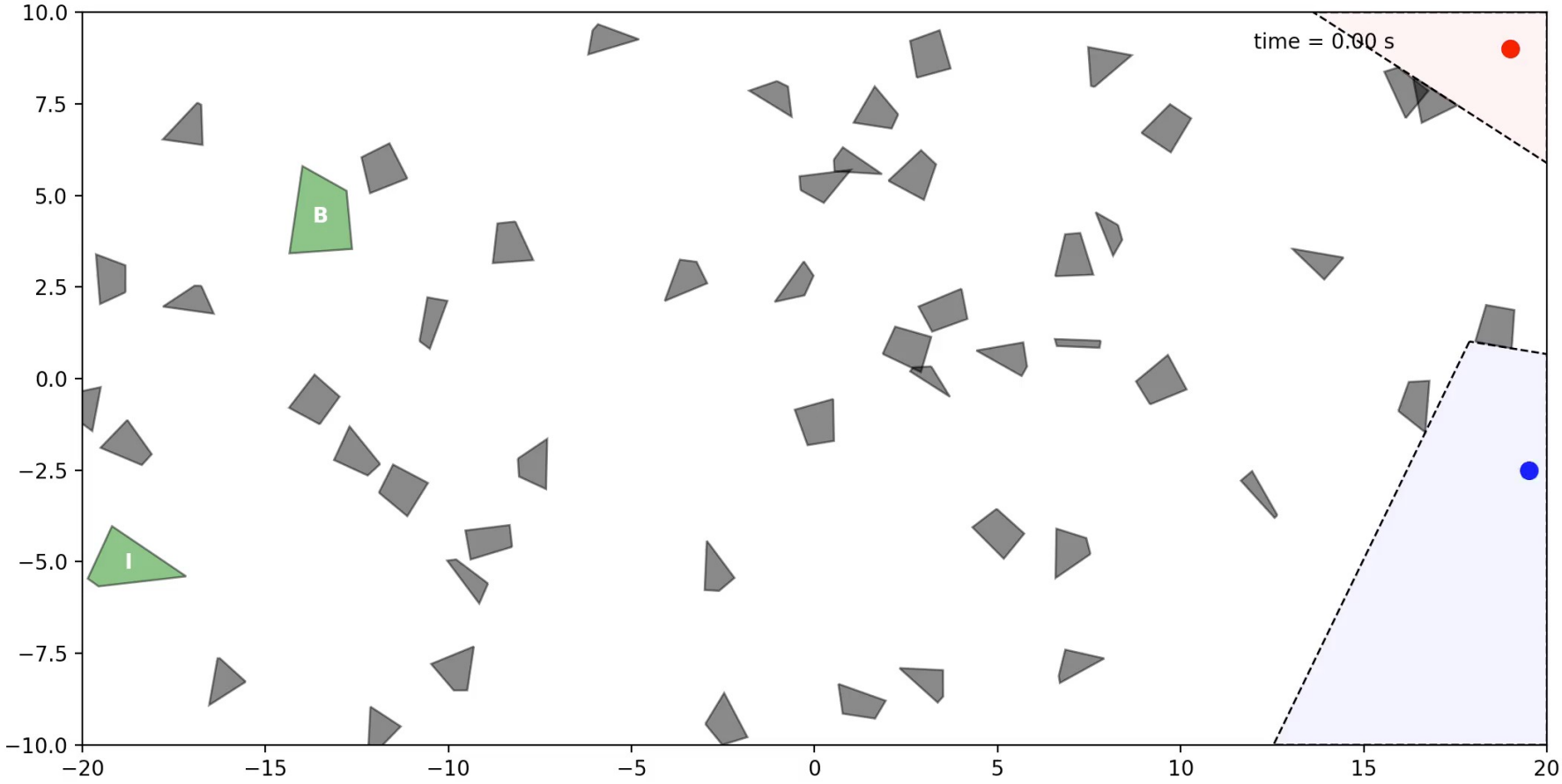
↓

Region 2

connection!



Example: Concurrent Sensing with Complimentary Sensors



Move together for 10 seconds and then visit distinct regions.

Key Takeaways

- For under-actuated robots, activities and motions couple through state and obstacles, timing and robot assignment.
- Planners should generate tasks and motions together, in light of higher-level goals.
- State Plans mediate between task and motion planning, by extracting essential constraints on state and time.
- State plans can be executed for multiple, underactuated vehicles over long horizons using convex optimization + model-predictive control.
- Enormous state spaces can be managed by combining convex optimization and combinatorial search.
- Safety is improved with planners that maximize expected utility under bounded risk, using risk allocation.
- Safe planning with learned dynamics is enabled through risk-assessment of non-linear, non-Gaussian behaviors.

QUESTIONS?