Task and Motion Planning (TAMP)

Caelan Reed Garrett
6.881 - Intelligent Robot Manipulation
11/12/2019

manipulation.csail.mit.edu/
github.com/caelan/pddlstream
(Probable) Roadmap

1. Background
   1. Task Planning
   2. Motion Planning

2. Hybrid Planning
   1. Prediscretized & Numeric Planning
   2. Multi-Modal Motion Planning
   3. Integrated TAMP

3. STRIPStream
   4. Temporal TAMP
   5. TAMP under Uncertainty

[Fig from Erion Plaku]
Robot must select both **high-level** actions & **low-level** controls

**Application areas**: semi-structured and human environments

- Household
- Food service
- Construction
- Warehouse fulfilment

Planning for Autonomous Robots
Task and Motion Planning (TAMP)

- Plan in a **factored, hybrid** space
  - **Discrete** and **continuous** variables & actions

- **Variables**
  - **Continuous**: robot configuration, object poses, door joint positions,
  - **Discrete**: is-on, is-in-hand, is-holding-water, is-cooked, ...

- **Actions**: move, pick, place, push, pull, pour, detect, cook, ...
Cooking and Stacking
Preparing Coffee
Automated Fabrication

- Plan sequence of 306 3D printing extrusions (actions)
- Collision, kinematic, **stability** and **stiffness** constraints

[Huang, Garrett, & Mueller 2018]
Problem Class

- **Discrete-time**
  - Plans are finite sequences of controls
- **Deterministic** (for now)
  - Actions always produce the intended effect
  - Solutions are plans (instead of policies)
- **Observable** (for now)
  - Access to the full world state
- **Hybrid**
  - States & controls composed of mixed discrete-continuous variables
Review: Task Planning (10/31/19)

- Plan in a large **discrete** space with **many variables**
- **Planning languages:** STRIPS/PDDL  
  - Fikes 1971
  - Aeronautiques 1998
- **Facts:** boolean state variables
- **Parameterized** actions
- **Preconditions** test validity
- **Effects** change the state
- **Heuristic search algorithms**  
  [Hoffman 2001][Helmert 2006]

```
(:action stack  
 :parameters (?b1 ?b2)  
 :precondition (and  
   (Holding ?b1)  
   (Clear ?b2)))  
 :effect (and  
   (HandEmpty)  
   (On ?b1 ?b2)  
   (not (Holding ?b1))  
   (not (Clear ?b2))))
```
Plan a **path** for a robot from an initial configuration to a goal configuration that **avoids obstacles**

- Sequence of **continuous** configurations
- Configurations often are **high-dimensional**
  - Example: 7 DOFs

**High-level approaches:**
- Geometric decomposition
- **Sampling-based**
- Optimization-based
Sampling-Based Motion Planning

- **Discretize** configuration space by **sampling**
  - Sampling be deterministic or **random**
- **Implicitly** represent the collision-free configuration space using an blackbox **collision checker**
- **Abstracts away** complex robot geometry

- **Algorithms**
  - **Probabilistic Roadmap (PRM)**
  - Rapidly-Exploring Random Tree (RRT)
  - **Bidirectional RRT (BiRRT)**

[Kavraki 1994][Kuffner 2000][LaValle 2006]
Find a path from init to goal that avoids the obstacles.
Sample a set of configurations
Probabilistic Roadmap (3/7)

Remove configurations that collide with the obstacles
Connect nearby configurations
Prune connections that collide with the obstacles
The resulting structure is a finite roadmap (graph)
Probabilistic Roadmap (7/7)

Search for the shortest-path on the roadmap

[Fig from Erion Plaku]
Collision Checking is Expensive

- Collision checking **dominates** runtime
- **Complex geometries & fine resolutions** (for safety)
- Many edges clearly do not lie on a low-cost path
- **Optimistically plan without collisions**
- Check collisions **lazily** only by only evaluating candidate plans
Lazy PRM (1/10)

Construct a PRM ignoring collisions

[Fig from Erion Plaku]
Search for the **shortest-path** on the roadmap

[Fig from Erion Plaku]
Remove plan edges that collide with obstacles
Lazy PRM (4/10)

Search for the new **shortest-path** on the roadmap
Lazy PRM (5/10)

Check the edges on the plan for collisions

[Fig from Erion Plaku]
Lazy PRM (6/10)

Check the edges on the plan for collisions (with increased resolution)

[Fig from Erion Plaku]
Lazy PRM (7/10)

Remove plan edges that collide with obstacles

[Fig from Erion Plaku]
Search for the new **shortest-path** on the roadmap
Lazy PRM (9/10)

Check the edges on the plan for collisions

[Fig from Erion Plaku]
Return the **current path** as a solution
Lazy Motion Planning

- **Defer** collision checking until a path is found
- **Remove** colliding edges path from the roadmap
- **Repeat** this process with a new path
- **Terminate** when a collision-free path is found

77 checks  23 checks

**Eager** (during search)  **Lazy**

[Bohlin 2000][Dellin 2016]
Theoretical Properties

- Sampling-based algorithms cannot prove infeasibility nor even solve every feasible problem.
- Robustly feasible: a problem that admits a solution for which all local perturbations are also solutions.
- Probabilistic complete: an algorithm that solves any robustly feasible problem with probability 1.

[Fig from Jenny Barry]
Frame motion planning as a non-convex constrained optimization problem & solve for local minima

minimize $f(x)$
subject to
$$g_i(x) \leq 0, \quad i = 1, 2, \ldots, n_{ineq}$$
$$h_i(x) = 0, \quad i = 1, 2, \ldots, n_{eq}$$

Collision constraints enforced via signed distance (sd)

$sd > 0$  $sd < 0$

[Ratliff 2009][Schulman 2013]
Task and Motion Planning (TAMP)
Shakey the Robot (1969)

- **First autonomous mobile manipulator** (via pushing)
- Visibility graph, A* search, and STRIPS!
- **Decoupled** task and motion planning
- Task planning **then** motion planning

```
type(robot robot) type(object object)
name(robot shakey) name(object box)
at(robot 4.1 7.2) at(object 3.1 5.2)
theta(robot 90 1) inroom(object r1)
shape(object wedge) radius(object 3.1)
```

```
GOTHRU(d,r1,r2)
Precondition INROOM(ROBOT,r1) & CONNECTS(d,r1,r2)
Delete List INROOM(ROBOT,$)
Add List INROOM(ROBOT,r2)
```

[Fikes 1971]
[Nilsson 1984]
Obstacle Blocks Shakey’s Path

- What if a movable block *prevented* Shakey from safely moving into the adjacent room?
- Shakey could **push** it out of the way or **go around** it
- What’s more efficient? How to push it? ...
Decoupled vs Integrated TAMP

- **Decoupled**: discrete (task) planning then continuous (motion) planning
- Requires a strong **downward refinement** assumption
- **Every** correct discrete plan can be **refined** into a correct continuous plan (from hierarchal planning)
- **Integrated**: **simultaneous** discrete & continuous planning
Geometric Constraints Affect Plan

- Inherits challenges of both motion & classical planning
  - High-dimensional, continuous state-spaces
  - State-space exponential in number of variables
  - Long horizons

- Continuous constraints limit high-level strategies
  - Kinematics, reachability, joint limits, collisions, grasp, visibility, stability, stiffness, torque limits, ...
Pouring Among Obstacles
Preparing a Meal for Two
Breaking Down “Preparing a Meal”

- Clean 3 blue cups and clean/cook 2 green cabbages
- 64 continuous and 10 discrete variables

1. High-dimensional
2. Long horizon
3. Discrete state
4. Geometric constraints
Block in Left Cabinet & Doors Closed

- Robot forced to **regrasp** the object
- Change from a **top** grasp to a **side** grasp

- **Non-monotonic** problem
- Plan must **undo goals** to solve
- **Open** then **close** the cabinet door

- Physical constraints can be subtle!
Hybrid Planning Spectrum

Purely Discrete
- Task Planning
  - Prediscretized Planning

Hybrid
- Numeric Planning
- Multi-Modal Motion Planning
  - Integrated Task and Motion Planning

Purely Continuous
- Motion Planning
Prediscretized & Numeric Planning
Prediscretized Planning

- Assumes that a **finite set** of object placements, object grasps, and (sometimes) robot configurations are **given**
- Can **directly** perform discrete task planning
- Still need to evaluate **reachability**
  - Eagerly in **batch** [Lozano-Pérez 2014][Garrett 2017][Ferrer-Mestres 2017]
  - Eagerly during **search** [Dornhege 2009]
  - **Lazily** [Erdem 2011][Dantam 2018][Lo 2018]
Discrete-Control Numeric Planning

- Classical planning with real-valued variables and durative actions
  - **Examples:** time and energy
  - Most planners only support linear/polynomial dynamics
  - **Non-linear** dynamics addressed by discretizing time
  - **Example:** battery domain

\[
\frac{d\delta}{dt} = \frac{i(t)}{c} - k'\delta
\]

\[
\frac{d\gamma}{dt} = -i(t)
\]

\[
\delta(t) = \frac{I}{c} \cdot \frac{1 - e^{-k't}}{k'}
\]

\[
\gamma(t) = C - It
\]

[Fox 2003][Hoffmann 2003][Eyerich 2009]
Continuous-Control Numeric Planning

- **Continuous control** parameters
- Tackle **convex dynamics** using cone programming
- Non-convexity handled by **partitioning** the state-space

- **In contrast**, TAMP is often:
  - High-dimensional
  - Non-convex
  - 3D collision constraints
  - Less sophisticated dynamically

[Deits 2015][Shoukry 2016]
[Fernandez-Gonzalez 2018]
Multi-Modal Motion Planning
Multi-Modal Motion Planning

- Collision-free configuration space changes when objects are manipulated
- Use a sequence of motion planning problems each defined by a mode
- Mode: a set of motion constraints
  - Gripper is empty
  - Relative object pose remains constant

Low-dimensional Intersections

- Need samples that **connect** adjacent modes
- Intersection of two modes is often **low-dimensional**
  - **Special-purpose** samplers are needed
- **Example**: transition from gripper **empty** to **holding**
- Configurations at the **intersection** obtained using **inverse kinematics (IK)**

[Hauser 2011]
Sampling-Based Multi-Modal Planning

1. Sample from the set of modes
2. Sample at the low-dimensional intersection of adjacent modes
3. Sample a roadmap within each mode
4. Discrete search on the multi-modal roadmap

[Hauser 2011]
Mixed Integer Programming (MIP)

- Continuous and integer variables
- Convex constraints and costs
- **Branch-and-bound**
  - Split on integer variables
- **Integrality relaxation**
  - Lower bound on cost
  - Loose when *logical* operations

- Planning limitation
- # of variables may be *exponential* in problem size
Optimization-Based Multi-Modal Motion Planning

- Discrete search over sequences of mode switches
- Sequences have varying length
- Each sequence induces a non-convex constrained optimization problem
- Sequences can be pruned using lower bounds obtained by relaxing some constraints

\[
\min_{x, a_1:K, s_1:K} \int_0^T f_{\text{path}}(\ddot{x}(t)) \, dt + f_{\text{goal}}(x(T)) \\
\text{s.t.} \quad x(0) = x_0, \quad h_{\text{goal}}(x(T)) = 0, \quad g_{\text{goal}}(x(T)) \leq 0, \\
\forall t \in [0, T]: \quad h_{\text{path}}(\dot{x}(t), s_k(t)) = 0, \\
\quad g_{\text{path}}(\ddot{x}(t), s_k(t)) \leq 0 \\
\forall k \in \{1, \ldots, K\}: \quad h_{\text{switch}}(\dot{x}(t_k), a_k) = 0, \\
\quad g_{\text{switch}}(\ddot{x}(t_k), a_k) \leq 0, \\
\quad s_k \in \text{succ}(s_{k-1}, a_k) .
\]

[Toussaint 2015] [Toussaint 2018] [Lagrieffoul 2014]
Integrated TAMP

- Geometric search **guided** by classical planning
  - Both heuristic and sampling guidance [Gravot 2005][Plaku 2010]
- Task and motion planning **interface**
  - Maintain **separate** discrete and continuous descriptions
- Custom interface to communicate between the two
- How are failures **diagnosed**?
  - [Erdem 2011][De Silva 2013]
  - [Srivastava 2014][Dantam 2018]
- Direct search in **combined** state-space
  - [Kaelbling 2011] [Garrett 2018a] [Garrett 2018b]
Hybrid Planning Spectrum Revisited

Discrete  Hybrid  Continuous

Task Planning  Numeric Planning  Motion Planning

Prediscretized Planning  Multi-Modal Motion Planning

Integrated Task and Motion Planning
Our Approach: STRIPStream

- **No general-purpose, flexible framework** for planning in a variety of TAMP domains
- Extends **PDDL** to incorporate **sampling procedures**
  - Can model domains with **infinitely-many** actions

- Develop **domain-independent** algorithms that treat the samplers as **blackbox inputs**
- Algorithms solve a **sequence of finite** PDDL problems
  - Leverage existing **classical planners** as subroutines
- Algorithms are particularly **fast when downward refinement holds** while remaining complete
STRIPStream Language
Benefits of Extending PDDL

- **Standardized** action description language
- Emphasis on describing and solving problems in a **domain-independent** way

- Large wealth of efficient, **existing algorithms** that exploit **factored** state & action structure

- Encodes the **difference** between two states using preconditions & effects
  - Most variables are **unchanged**
  - Actions can be described using **few parameters**
STRIPStream + Drake
Solved Using the Same Algorithm

Framework not specific to a single robot or robotics at all!

Goal: Believe Spam in Closed Bottom Drawer x2
Motivating Pick & Place Example

- Single object **prevents** a goal object from being reachable
- Focus on a compact 2D version
- Formulation almost the same for 3D
- Algorithms agnostic to number of DOFs
2D Pick-and-Place Example

- **Goal**: block A within the red region
- Robot and block poses are continuous (x, y) pairs
- Block B obstructs the placement of A
2D Pick-and-Place Solution

- One (of infinitely many) possible solutions
  - move, pick B, move, place B,
  - move, pick A, move, place A
Some constants are **numpy arrays**

**Static initial facts** - value is **constant** over time

- (Block, A), (Block, B), (Region, red), (Region, grey), (Conf, [-7.5 5.]), (Pose, A, [0. 0.]), (Pose, B, [7.5 0.]), (Grasp, A, [0. -2.5]), (Grasp, B, [0. -2.5])

**Fluent initial facts** - value **changes** over time

- (AtConf, [-7.5 5.]), (HandEmpty), (AtPose, A, [0. 0.]), (AtPose, B, [7.5 0.])

**Goal formula:** \((\exists p) (\text{and} (\text{Contained A } p \text{ red}) (\text{AtPose A } p)))\)
2D Pick-and-Place Actions

- Typical PDDL action description except that arguments are high-dimensional & continuous!
- To use the actions, must **prove** the following **static facts**:

  \[(\text{Motion } ?q_1 \ t \ ?q_2), (\text{Kin } ?b \ ?p \ ?g \ ?q)\]


define action move
parameters (?q1 ?t ?q2)
precondition (and \(\text{(Motion } ?q_1 \ t \ ?q_2) \ (\text{AtConf } ?q_1)\))
effect (and \(\text{(AtConf } ?q_2) \ (\text{not} (\text{AtConf } ?q_1)))\))

define action pick
parameters (?b ?p ?g ?q)
precondition (and \(\text{(Kin } ?b \ ?p \ ?g \ ?q) \ (\text{AtConf } ?q) \ (\text{AtPose } ?b \ ?p) \ (\text{HandEmpty}))\))
effect (and \(\text{(AtGrasp } ?b \ ?g) \ (\text{not} (\text{AtPose } ?b \ ?p)) \ (\text{not} (\text{HandEmpty})))\))
BFS in Discretized State-Space

Suppose we were given the following additional static facts:

- \((\text{Motion}, [-7.5, 5.], \tau_1, [0.2.5])\)
- \((\text{Motion}, [-7.5, 5.], \tau_2, [-5.5])\)
- \((\text{Motion}, [-5.5, 5.], \tau_3, [0.2.5])\)
- \((\text{Kin}, A, [0.0], [0.-2.5], [0.2.5])\)

\[
\text{(move, [-7.5, 5.], } \tau_1, [0.2.5])
\]
\[(\text{AtConf, [0.2.5]})
\quad (\text{AtPose, A, [0.0]})
\quad (\text{AtPose, B, [7.5, 0.0]})
\quad (\text{HandEmpty})\]

\[
\text{(move, [-7.5, 5.], } \tau_2, [-5.5])
\]
\[(\text{AtConf, [-5.5]})
\quad (\text{AtPose, A, [0.0]})
\quad (\text{AtPose, B, [7.5, 0.0]})
\quad (\text{HandEmpty})\]

\[
\text{(move, [-5.5, 5.], } \tau_3, [0.2.5])
\]
\[(\text{AtConf, [0.2.5]})
\quad (\text{AtGrasp, A, [0.-2.5]})
\quad (\text{AtPose, B, [7.5, 0.0]})\]
No a Priori Discretization

- **Values given at start:**
  1. initial configuration: (Conf, [-7.5 5.])
  2. initial poses: (Pose, A, [0. 0.]), (Pose, B, [7.5 0.])
  2. grasps: (Grasp, A, [0. -2.5]), (Grasp, B, [0. -2.5])

- **Planner needs to find:**
  1. pose within a region: (Contain A ?p red)
  4. grasping configurations: (Kin ?b ?p ?g ?q)
  4. robot trajectories: (Motion ?q1 ?t ?q2)
What Samplers Do We Need?

- **Low-dimensional** placement stability constraint (*Contain*)
  - i.e. 1D manifold embedded in 2D pose space
  - Directly sample values that satisfy the constraint
  - May need **arbitrarily many** samples
  - Gradually enumerate an **infinite sequence**
Intersection of Constraints

- **Kinematic constraint** ($\text{Kin}$) involves poses, grasps, and configurations
- **Conditional samplers** - samplers with inputs

Inverse Kinematics

Pose $p$

Grasp $g$

Config $q_1$, $q_2$, ...
Composing Conditional Samplers

- Outputs of one conditional sampler are the inputs to another
- Directed acyclic graph (DAG) of conditional samplers

Diagram:
- Placement Sampler
  - Pose
  - Grasp g
- Inverse Kinematics
  - Config
  - Motion Planner
  - Trajectory \( \tau_1, \tau_2, \ldots \)
Stream: a function to a generator

- **Advantages**
  - Programmatic implementation
  - Compositional
  - Supports infinite sequences

- **Stream** - function from an **input object tuple** \((x_1, x_2, x_3)\) to a (potentially infinite) sequence of **output object tuples** \([ (y_1, y_2), (y'_1, y'_2), \ldots ]\)

```python
def stream(x1, x2, x3):
    i = 0
    while True:
        y1 = i*(x1 + x2)
        y2 = i*(x2 + x3)
        yield (y1, y2)
        i += 1
```

[Kaelbling 2011][Srivastava 2014][Garrett 2018a][Garrett 2018b]
Stream Certified Facts

- Objects alone aren’t helpful: **what do they represent?**
  - Communicate semantics using **predicates**!

- Augment stream specification with:
  - **Domain facts** - static facts declaring legal **inputs**
    - e.g. only configurations can be motion inputs
  - **Certified facts** - static facts that all **outputs** satisfy with their corresponding **inputs**
    - e.g. poses sampled from a region are within it
Sampling Contained Poses

(:stream sample-region
 :inputs (?b ?r)
 :domain (and (Block ?b) (Region ?r))
 :outputs (?p)
 :certified (and (Pose ?b ?p) (Contain ?b ?p ?r)))

```python
def sample_region(b, r):
    x_min, x_max = REGIONS[r]
    w = BLOCKS[b].width
    while True:
        x = random.uniform(x_min + w/2, x_max - w/2)
        p = np.array([x, 0.])
        yield (p,)
```

Block b
Region r

sample-region

Pose [(p), (p'), (p''), ...]
Sampling IK Solutions

- **Inverse kinematics (IK)** to produce robot grasping configuration
- Trivial in 2D, non-trivial in general (e.g. 7 DOF arm)

```
(:stream sample-ik
 :inputs (?b ?p ?g)
 :domain (and (Pose ?b ?p) (Grasp ?b ?g))
 :outputs (?q)
 :certified (and (Conf ?q) (Kin ?b ?p ?g ?q)))
```
Calling a Motion Planner

- “Sample” (e.g. via a PRM) **multi-waypoint trajectories**
- Include **joint limits & fixed obstacle collisions**, but not movable object collisions

```
(stream sample-motion
  :inputs (?q1 ?q2)
  :domain (and (Conf ?q1) (Conf ?q2))
  :outputs (?t)
  :certified (and (Traj ?t) (Motion ?q1 ?t ?q2)))
```
2D Place Collisions

- Add parameters for the pose of each block - **bad**!
- Use a **derived predicate** for whether currently **unsafe**
  - Predicate defined by **logical formula**
  - Enables lightweight **logical inference**
  - Decomposes collision checking into a logical **AND**

```
(:action place
 :parameters (?b ?p ?g ?q)
 :precondition (and ...
               (not (UnsafePose ?b ?p)))
 :effect (and ...)

(:derived (UnsafePose ?b1 ?p1)
 (exists (?b2 ?p2) (and (Pose ?b1 ?p1) (Pose ?b2 ?p2)
                        (not (= ?b1 ?b2)) (AtPose ?b2 ?p2)
                        (not (CFree ?b1 ?p1 ?b2 ?p2)))))
```
Check Block Collisions

- **Test stream**: stream without output objects
- Return True if **collision-free** placement (e.g. via querying a collision checker)

```
(:stream test-cfree
 :inputs (?b1 ?p1 ?b2 ?p2)
 :domain (and (Pose ?b1 ?p1) (Pose ?b2 ?p2))
 :outputs ()
 :certified (CFree ?b1 ?p1 ?b2 ?p2))
```
STRIPStream = STRIPS + Streams

- **Domain dynamics** (*domain.pddl*): declares actions
- **Stream properties** (*stream.pddl*)
  - Declares stream inputs, outputs, and certified facts
- **Problem and stream implementation** (*problem.py*)
  - Initial state, *Python constants*, & goal formula
- **Stream implementation using Python generators**

User provides:
- Domain
- Streams
- Init & Goal

STRIPStream Planner

Plan
Supporting Facts

[Garrett 2018b]
Two STRIPStream Algorithms

- STRIPStream planners decide which streams to use
- Algorithms alternate between searching & sampling:
  1. **Search** a finite PDDL problem for plan
  2. **Modify** the PDDL problem (depending on the plan)
- Search implemented using **off-the-shelf algorithms**
  - **Off-the-shelf AI planner** - FastDownward
  - Exploits factoring in its search heuristics (e.g. \( h_{\text{FF}} \))
- **Probabilistically complete** given sufficient samplers

[Garrett 2018a]
[Garrett 2018b]
Incremental Algorithm

- Incrementally construct all possible initial facts
- Periodically check if a solution exists
- Repeat:
  1. Compose and evaluate a finite number of streams to unveil more facts in the initial state
  2. Search the current PDDL problem for plan
  3. Terminate when a plan is found

[Garrett 2018a]
[Garrett 2018b]
Incremental: Sampling Iteration 1

**Iteration 1 - 14 stream evaluations**

- **Sampled:**
  - 2 new robot configurations:
  - 4 new block poses:
  - 2 new trajectories:
Incremental: Search Iteration 1

- Pass current discretization to FastDownward
- If **infeasible**, the current set of samples is insufficient
Incremental: Sampling Iteration 2

Iteration 2 - 54 stream evaluations

- **Sampled:**
  - 4 new robot configurations:
  - 4 new block poses:
  - 10 new trajectories:
Incremental: Search Iteration 2

- Pass current discretization to FastDownward
- If **infeasible**, the current set of samples is insufficient
Incremental Example: Iterations 3-4

**Iteration 3** - 118 stream evaluations

**Iteration 4** - 182 stream evaluations

**Solution:**

1) move \([-7.5\ 5.\] [[-7.5 5.], [-7.5 5.], [7.5 5.], [7.5 2.5]] [7.5 2.5]
2) pick B [7.5 0. ] [0. -2.5] [7.5 2.5]
3) move [7.5 2.5] [[7.5 2.5], [7.5 5.], [10.97 5.], [10.97 2.5 ]] [10.97 2.5 ]
4) place B [10.97 0. ] [0. -2.5] [10.97 2.5 ]
5) move [10.97 2.5 ] [[10.97 2.5 ], [10.97 5.], [0. 5.], [0. 2.5]] [0. 2.5]
6) pick A [0. 0.] [0. -2.5] [0. 2.5]
7) move [0. 2.5] [[0. 2.5], [0. 5.], [7.65 5.], [7.65 2.5 ]] [7.65 2.5 ]
8) place A [7.65 0. ] [0. -2.5] [7.65 2.5 ]

- **Drawback** - many unnecessary samples produced
- **Computationally expensive** to generate
- **Induces large discrete-planning problems**
Optimistic Stream Outputs

- Many TAMP streams are exceptionally expensive
- Inverse kinematics, motion planning, collision checking
- Only query streams that are identified as useful
- Plan with optimistic hypothetical outputs [Srivastava 2014]
- Inductively create unique placeholder output objects for each stream instance (has # as its prefix)

Optimistic evaluations:
1. s-region:(b0, red)->(#p0)
2. s-ik:(b0, [0. 0.], [0. -2.5])->(#q0),
3. s-ik:(b0, #p0, [0. -2.5]) ->(#q2) [Garrett 2018a] [Garrett 2018b]
Focused Algorithm

- **Lazily** plan using optimistic outputs *before* real outputs
- **Recover** set of streams used by the optimistic plan
- **Repeat:**
  1. Construct active **optimistic** objects
  2. **Search** with **real & optimistic** objects
  3. If only **real objects** used, **return plan**
  4. **Sample** used streams
  5. **Disable** used streams

[Garrett 2018a][Garrett 2018b]
Focused Example 1

Optimistic Plan:
\[
\text{move}([-5. \ 5.], \#t0, \#q0), \ 	ext{pick}(A, [0. \ 0.], [-0. \ -2.5], \#q0), \\
\text{move}(\#q0, \#t2, \#q1), \ 	ext{place}(A, \#p0, [-0. \ -2.5], \#q1)
\]

Constraints:
\[
(\text{kin, } A, \#q0, \#p0, [-0. \ -2.5]), \\
(\text{kin, } A, \#q1, [0. \ 0.], [-0. \ -2.5]), \\
(\text{motion, } [-5. \ 5.], \#t1, \#q1), \\
(\text{motion, } \#q1, \#t2, \#q0), \\
(\text{contain, } A, \#p0, \text{red}),
\]

\[
\text{s-region}(A, \text{red}) \rightarrow (\#p0) \\
\text{s-ik}(A, \#p0, [-0. \ -2.5]) \rightarrow (\#q0) \\
\text{s-ik}(A, [0. \ 0.], [-0. \ -2.5]) \rightarrow (\#q1) \\
\text{s-motion}(\#q1, \#q0) \rightarrow (\#t2) \\
\text{s-motion}([-5. \ 5.], \#q1) \rightarrow (\#t1)
\]
Focused Example 2: Iteration 1

Optimistic Plan:
\[
\text{move}([-5. 5.], \#t0, \#q0), \text{pick}(A, [0. 0.], [-0. -2.5], \#q0), \\
\text{move}(\#q0, \#t2, \#q1), \text{place}(A, \#p0, [-0. -2.5], \#q1)
\]

Constraints:
- (\text{cfree, A, } \#p0, B, [7.5 0. ]), (\text{contain, A, } \#p0, \text{red}),
- (\text{kin, A, } \#q0, [0. 0.], [-0. -2.5]), (\text{kin, A, } \#q1, \#p0, [-0. -2.5]),
- (\text{motion, } \#q0, \#t2, \#q1), (\text{motion, } [-5. 5.], \#t0, \#q0)

Stream evaluations:
1. **s-region**:(A, red)\rightarrow[[[8.21 0. ]]]
2. **t-cfree**:(A, [8.21 0. ], B, [7.5 0. ])=\textbf{False}

These stream instances are \textbf{removed} from subsequent searches.
Focused Example: Iteration 2

Optimistic Plan:

move([-5. 5.], #t4, #q2), pick(B, [7.5 0.], [-0. -2.5], #q2),
move(#q2, #t9, #q3), place(B, #p1, [-0. -2.5], #q3),
move(#q3, #t6, #q0), pick(A, [0. 0.], [-0. -2.5], #q0),
move(#q0, #t8, #q4), place(A, [8.21 0.], [-0. -2.5], #q4)

\textbf{t-cfree:}(A, [8.21 0.], B, [7.5 0.]) \textbf{previously failed}
\textbf{t-cfree:}(A, [8.21 0.], B, #p1) \textbf{might succeed}

\textbf{s-region:}(B, grey)\textbf{->(#p1)}

\textbf{t-cfree:}(B, #p1, A, [0. 0.])\textbf{->()}

\textbf{t-cfree:}(A, [8.21 0.], B, #p1)\textbf{->()}

\textbf{s-ik:}(B, [7.5 0.], [-0. -2.5])\textbf{->(#q3)}

\textbf{s-motion:}([-5. 5.], #q3)\textbf{->(#t4)}
Focused Outperforms Incremental

Incremental ~20 s
Focused ~10 s

Incremental N/A
Focused ~25 s

Incremental N/A
Focused ~20 s

[Garrett 2018a]
Cost-Minimizing Planning

- Actions costs specified as nonnegative functions

(action move
 :parameters (?q1 ?t ?q2)
 :precondition (and (Motion ?q1 ?t ?q2)
 (AtConf ?q1))
 :effect (and (AtConf ?q2)
 (not (AtConf ?q1)))
 (increase (total-cost) (Length ?t)))

- Function specification similar to derived predicates

  def Length(t):
  return sum(np.linalg.norm(q2 - q1)
  for q1, q2 in zip(t[:-1], t[1:]))

- Asymptotically optimal algorithms
Goal: Hold Any Green Block

- Lower bounds on costs improve focused performance
Multi-Robot TAMP
Centralized Scheduling of Robots

- PDDL rovers domain with **visibility** and **reachability**
- How to plan for **simultaneous** execution?
- Use a **temporal planner** as search subroutine (e.g. Temporal FastDownward)

[Eyerich 2009]
Swap Initial Configurations
Temporal Task & Motion Planning

- Temporally annotated preconditions and effects
  - “at start”, “over all”, and “at end” (PDDL2.1) [Fox 2003]

`:durative-action move
 :parameters (?r ?q1 ?t ?q2)
 :duration (= ?duration (/ (Distance ?t) (Speed ?r)))

:condition (and
  (at start (Robot ?r))
  (at start (Motion ?q1 ?t ?q2))
  (at start (AtConf ?r ?q1))
  (over all (not (UnsafeTraj ?r ?t))))

:effect (and
  (at start (not (AtConf ?r ?q1)))
  (at start (OnTraj ?r ?t))
  (at end (not (OnTraj ?r ?t)))
  (at end (AtConf ?r ?q2))))
Enforcing Collision Constraints

- Robots might collide during the execution of their trajectories
- Planner doesn’t know exact position along trajectory
- Conservatively, check all configuration pairs per segment (over all (not (UnsafeTraj ?r ?t)))

- Derived predicate evaluated at each time event (:derived (UnsafeTraj ?r1 ?t1) (exists (?r2 ?t2) (and (Robot ?r1) (Robot ?r2) (not (= ?r1 ?r2)) (TrajTrajCollision ?t1 ?t2) (OnTraj ?r2 ?t2))))
Swap with Rechargeable Battery
Robot movement **depletes battery charge** proportional to distance traversed

**Infinitely-many** possible move action instances

```prolog
(:durative-action move 
 :parameters (?r ?q1 ?t ?q2) 
 :duration (= ?duration (/ (Distance ?t) (Speed ?r))))

:condition (and ... 
 (at start (<= (* (ConsumeRate ?r) ?duration) 
           (Energy ?r))))

:effect (and ... 
 (at start (decrease (Energy ?r) 
                    (* (ConsumeRate ?r) ?duration))))
```
Robots can recharge battery via solar power

Can perform 3D robotic planning while benefiting from state-of-the art numeric heuristics

(:durative-action recharge
 :parameters (?r ?q)
 :duration (= ?duration (/ (- (Capacity ?r) (Energy ?r)) (RechargeRate ?r)))
 :condition (...)
 :effect (and ...
   (at end (increase (Energy ?r)
    (* (RechargeRate ?r) ?duration)))))
TAMP Under Uncertainty
MDP: Stochastic Action Effects

- Approximate as cost-sensitive **deterministic** problem
- **Policy** computed online via replanning

**Goal:** Holding Cooked Spam x4
POMDP: Partially-Observable State

- Update a belief (probability distribution) over states
- Plan in the space of beliefs (belief space planning)
- Intentionally take observation actions
Geometric & Probabilistic Constraints

Goal: Believe Spam in Closed Bottom Drawer
x4
Belief-Space TAMP System

- Convolutional Neural Network (CNN) Object Detector
- Point cloud **plane estimation** to identify surfaces
- Point cloud **pose estimation** for objects
- Occupancy grid for non-manipulable
- Plan, execute, & observe in **real time**
Takeaways

- **Task and Motion Planning (TAMP):** Hybrid planning where continuous constraints affect discrete decisions.

- **Sampling** is powerful for exploring continuous spaces.

- **STRIPStream**: Planning language that supports sampling procedures as blackbox streams.

- Domain-independent algorithms.

- Lazy/optimistic planning intelligently queries only a small number of samplers (focused algorithm).

- Ongoing work involving cost-sensitive, multi-agent, probabilistic & partially observable TAMP.
Questions? (and Outtakes!)
References
Task Planning


Motion Planning

Prediscretized Planning


Numeric Planning


Multi-Modal Motion Planning


Task and Motion Planning


Probabilistic & Partially-Observable


