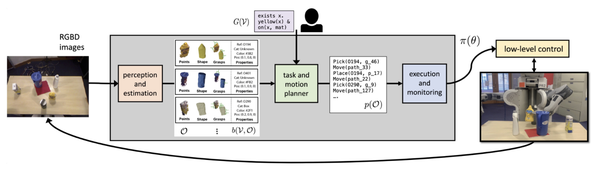
Flexible manipulation without prior shape models

Principal Investigators: Tomás Lozano-Pérez and Leslie Pack Kaelbling, MIT CSAIL

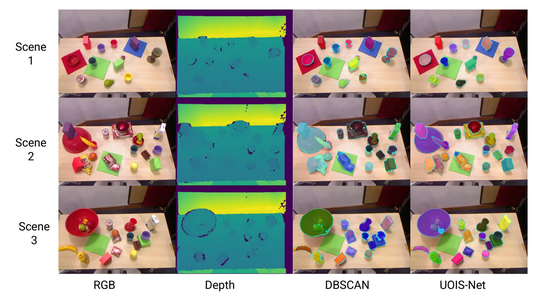
M0M: Manipulation with zero models

M0M: Manipulation with 0 Models

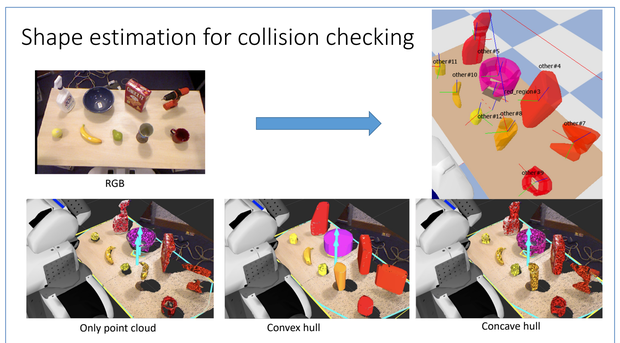


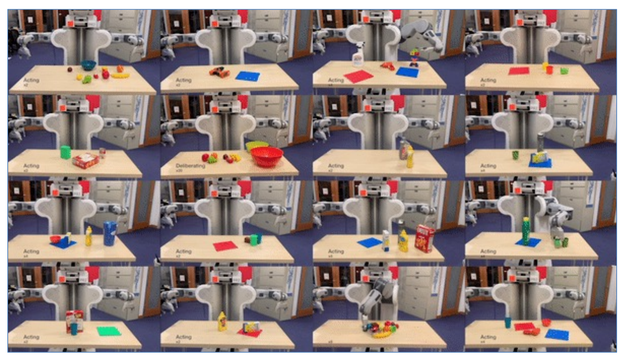
Curtis, Fang, Kaelbling, Lozano-Perez, Garrett; ICRA22

Segmentation



Shape estimation for collision checking





PDSketch: Integrated learning and planning

PDSketch: Integrated Planning Domain Programming and Learning

```
def press_button(b: Button):
  for m in objects:
    if belongs_to(b, m) and is_painter(m):
      for x: Any in objects:
        if in(x, m):
          x.color = mix(x.color, m.color)
    if ...
```

```
def press_button(b: Button):
  for m in objects:
    if belongs_to(b, m) and ???(m):
      for x: Any in objects:
        if ???(x, m):
          x.color = ???(x.color, m.color)
    if ...
```

Intuition:

Engineers decide how much amount of knowledge they want to program in.

??? indicates functions that they don't know how to write/don't want to write manually.

But programming the overall "structures" is usually easy.

Mao, Lozano-Perez, Tenenbaum, Kaelbling; NeurIPS22

Existing Frameworks

Domain Programming

```
def facing(agent, object): ...

def move_forward(s):
  if not any(
    facing(s.agent, x) and
    is_obstacle(x)
    for x in s.objects
  ):
    if s.agent.facing == 0:
      s.agent.x -= 1
    elif s.agent.facing == 1:
      s.agent.y += 1
    elif ...
```

A lot of prior knowledge.
No/Minimal training data.
Fast planning.

PDSketch (Our Work)

```
def move_forward(s):
  if not any(
    ??(s.agent, x)
    for x in s.objects
  ):
    s.agent = ??(s.agent)
```

Small amount of prior knowledge.
Small amount of training data.
Fast planning.

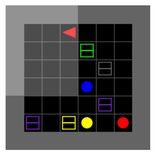
Neural Network Learning

```
def move_forward(s):
  s.agent = ??(s)
  for i in range(n):
    s.objects[i] = ??(s)
```

??: Trainable Neural Networks.

Minimal prior knowledge.
A lot of training data.
Slow planning.

MiniGrid Example



State Space:
`s.agent = (x, y, facing)`
`s.objects[i] = (x, y, image)`

Predicates
`next_to(agent, object)`
`is_yellow(object)`
`is_box(object)`
`.....`

Transition Model
`def move_forward(s): ...`
`def rturn(s): ...`
`def toggle(s): ...`
`.....`

Domain Model

State → Domain-Independent Planner → Action (RTurn)

Goal: "Go to a red object" ∃x. is_yellow(x) & next_to(agent, x)

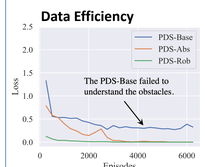
Learning and Planning Efficiency

PDS-Rob
Full robot movement models.
Need to learn object classifiers.

PDS-Abs
Abstract robot models.
(Sparse and local structures)

PDS-Base
Graph neural network.
(Weakest prior)

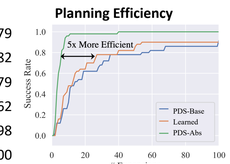
Data Efficiency



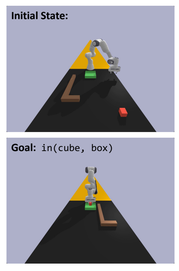
Success Rate

Behavior Cloning	0.79
Decision Xformer	0.82
DreamerV2	0.79
PDS-Base	0.62
PDS-Abs	0.98
PDS-Rob	1.00

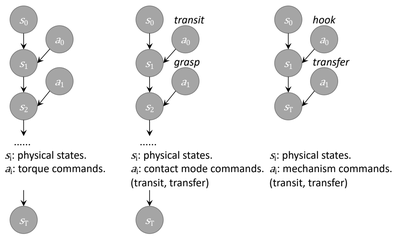
Planning Efficiency



Planning For Complex Manipulation



Simplified Example On a Table: Three Strategies

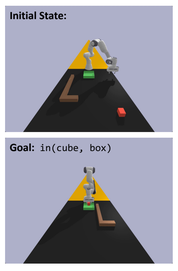


s_t: physical states.
a_t: torque commands.
Torque Level

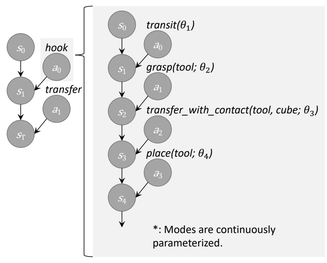
s_t: physical states.
a_t: contact mode commands. (transit, transfer)
Contact Mode Level

s_t: physical states.
a_t: mechanism commands. (transit, transfer)
"Operator" Level

Mechanisms: Representation



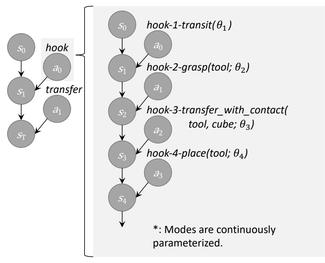
A mechanism is a "macro over contact modes".



*: Modes are continuously parameterized.

Mechanisms: Representation

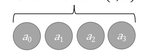
A mechanism is a "macro over contact modes".



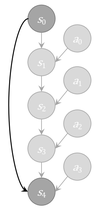
*: Modes are continuously parameterized.

(Joint) Sampler for Action Parameters

$\text{PowerfulNeuralNet}(\cdot\,;\theta)$



Transition Model

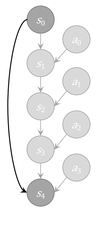


Mechanisms: Learning

(Joint) Sampler for Action Parameters

$\text{PowerfulNeuralNet}(\cdot\,;\theta)$

Transition Model



If I have "a lot of" data, then I can just train my favorite neural network.

What if I only have very few demos?

In practice, we don't want to have thousands of videos to teach the robot just one "trick."

- The intuition is to *exploit the contact mode structure of mechanisms to explore*.

Mechanisms: Representation in PDSketch

Recall we have three layers of transition model:

- Simulation (PyBullet), Contact Primitives (Transit, Transfer, etc.), Mechanisms.

?robot —Fixed→ ?tool —Fixed*→ ?target ←Slide— ?support

```
(:action indirect-push
    :parameters (?robot - robot ?target - item ?tool - item ?support - item
                 ?param - indirect-push-param ?qt - qpos-trajectory)
    :precondition (and
        (robot-holding-item ?robot ?tool)
        (support ?target ?support)
        (valid-indirect-push-param  ...  ?param)
        (valid-indirect-push-trajectory ... ?param ?qt)
    )
    :effect (and
        (robot-qpos::assign[simulation=true] ?robot (??f1 ?qt))
        (item-pose::assign [simulation=true] ?target (??f3 ?qt))
        (item-pose::assign [simulation=true] ?tool (??f2 ?qt))
    )
    :controller (run-indirect-push ... ?param ?qt)
)
```

Specified in Python: use a simulator to compute the effect (a.k.a. no amortized model)

An underlying "controller" for this mode.

Mechanisms: Representation in PDSketch

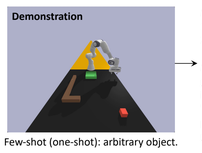
Mechanisms are macros over contact-level primitives.

```
(:macro
    (hook-and-grasp ?robot - robot ?tool - item ?target - item ?support - item ...)
    (then
        (grasp ?robot ?tool ?support ...)
        (indirect-push ?robot ?target ?tool ?support ...)
        (place ?robot ?tool ?support ...)
        (grasp ?robot ?target ?support ...)
    )
)
```

- During search time, this macro will be treated as a new primitive action.
- It will be expanded into four primitives during search.

Mechanisms: Learning Pipeline

Demonstration

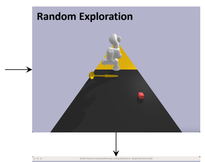


Few-shot (one-shot): arbitrary object.

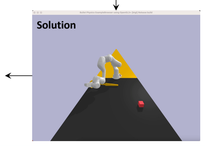
Extract from demonstration:
1. Contact mode skeleton.
2. Goal expression (by comparing before/after state).

In this case:
grasp('robot', 'tool1', 'table', '??', '??')
indirect_push('robot', 'box1', 'tool1', 'table', '??', '??')
place('robot', 'tool1', 'table', '??', '??')
grasp('robot', 'box1', 'table', '??', '??')

Random Exploration



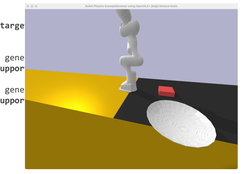
Solution



Planning ← $\text{PowerfulNeuralNet}(\cdot\,;\theta)$ ← **Training**

Advantage: Handling "Quasi-Dynamic" Motion

```
(:action place
    :parameters (?robot - robot ?target - item ?support - item ?targe
    :precondition (and
        (robot-holding-item ?robot ?target)
        (valid-placement-param      [generator_placeholder=true, gene
identifier ?robot) (item-identifier ?target) (item-identifier ?suppor
pose ?support) ?target-pose)
        (valid-placement-trajectory [generator_placeholder=true, gene
identifier ?robot) (item-identifier ?target) (item-identifier ?suppor
pose ?target) (item-pose ?support) ?target-pose ?qt)
    )
    :effect (and
        (robot-hands-free ?robot)
        (not (robot-holding-item ?robot ?target))
        (forall (?s - item)
            (support::assign ?target ?s (??f2 ?qt))
        )
        (robot-qpos::assign ?robot (??f1 ?qt))
        (item-pose::assign ?target ?target-pose)
        (item-pose::assign ?support (??f2 ?qt))
    )
    :controller (run-place (robot-identifier ?robot) (item-identifier ?target) (item-
identifier ?support) ?target-pose ?qt)
)
```



Task: Supported(P, B) and not Supported(P, Table)

RDDLStream: TAMP under uncertainty

RDDLStream Outcome Uncertainty Example

Initial Belief State:
(at a a_pose), (at c c_pose), (at c c_pose)
(shape a sq), (shape b sq), (shape c nsq)

Samplers:
force(?o) –> [?v]
placement(?o) –> [?p]
grasp(?o) –> [?g]

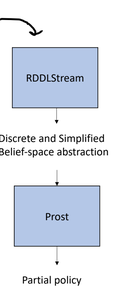
Actions:
pick(?o, ?p, ?g) –> []
– pre: [(not (holding ?o)), (at ?o ?p)]
– veff: [(holding ?o)]
- eff: [(not (at ?o ?p))]
place(?o, ?p, ?g) –>[]
– pre: [(not (holding ?o)), (at ?o ?p)]
– eff: [(holding ?o), (not (at ?o ?p))]
push(?o, ?v) –> [?p]
– eff: [at ?o ?p]

Goal: Create a stack three blocks

RDDLStream Compilation

Pick (a, a_pose) –> {(holding a): 1.0}
Pick (b, b_pose) –> {(holding b): 1.0}
Pick (c, c_pose) –> {(holding c): 0.2}

Push (a, v_01) –> {(at p01): 0.5, (at p02): 0.5}
...



RDDLStream Information Gathering

Initial Belief State:
(at a a_pose)
(at b b_pose)

Samplers:
None

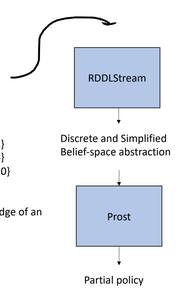
Actions:
pick(?o, ?p) –> []
– pre: [(not (holding ?o)), (at ?o ?p)]
– eff: [(not (at ?o ?p)), (holding ?o)]
place(?o, ?p) –>[]
– pre: [(not (holding ?o)), (at ?o ?p)]
– eff: [(holding ?o), (not (at ?o ?p))]
observe(?o, ?o*, ?p*) –> [?p]
– pre: [(handempty), (at ?o*, ?p*)]
– veff: [(at ?o ?p)]

Goal: Hold C

RDDLStream Compilation

Pick (a, a_pose) –> {(holding a): 1.0}
Place (a, a_pose_2) –> {(at a a_pose_2): 1.0}
Pick (b, b_pose) –> {(holding b): 1.0}
Place (b, b_pose_2) –> {(at b b_pose_2): 1.0}
Observe (c, [a,b], [ap, bp])) –> {(at c cpose): 0}
Observe (c, [a,b], [ap, bp2])) –> {(at c cpose): 0.2}
Observe (c, [a,b], [ap2, bp])) –> {(at c cpose): 0.4}
Observe (c, [a,b], [ap2, bp2])) –> {(at c cpose): 1.0}

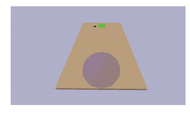
How is this better than an "offline" POMCP?
1.) Pick cannot be executed without full knowledge of an object's pose (property of belief)
2.) Observations only either generate full pose information or not
3.) Outcome sampling is guided by optimisitic observations

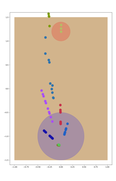


RDDLStream

Simple Outcome Uncertainty

- Puck starts in a fixed location
- The agent can push the puck with a force in R^2 if it is within a kinematically feasible region
- There is uncertainty in the mass, which results in outcome uncertainty with the pushing action
- The goal is to land the puck in the green region





RDDLStream

Outcome Uncertainty with IG

- Actions can gather information that change the belief state
- Information gathering actions must stay within the reachable configuration regions so they can be pushed again
- Works reliably with
  - Depth=2
  - Outcome samples = 10
  - Force vector samples = 10