Using Classical Planners for Tasks with Continuous Actions in Robotics

Stuart Russell

Joint work with Siddharth Srivastava, Lorenzo Riano, Pieter Abbeel

Using Classical Planners for Tasks with Continuous Actions in Robotics

Stuart Russell

Real work done by
~~Joint work with~~ Siddharth Srivastava, Lorenzo Riano, Pieter Abbeel

Outline

- Can we apply classical planners to robotics problems?
  - Challenges: continuous action arguments, geometric reasoning
- Main ideas:
  - Symbolic references to continuous values
  - Optimistic model with symbolic corrections from low-level geometric motion planner, followed by replanning
- Why does this idea work? Can it be generalized?
  - Roughly analogous to theorem-proving with quantifier elimination
  - Current algorithm complete under strong assumptions
- Will it work for real-world problems?
  - Results on PR2 simulator, PR2

Combining Task and Motion Planners

- Discrete/classical planners:
  + Effective algorithms for combinatorial discrete spaces (e.g., automated heuristic generation)
  – Not directly applicable to continuous spaces
- Continuous/motion planners:
  + Effective algorithms for high-dimensional continuous space (e.g., PRM, RRT)
  – Not directly applicable to discrete spaces induced by contact changes (e.g., pickup/putdown)

Combining Task and Motion Planners

- Discrete/classical planners:
  - + Effective algorithms for combinatorial discrete spaces (e.g., automated heuristic generation)
  - – Not directly applicable to continuous spaces
- Continuous/motion planners:
  - + Effective algorithms for high-dimensional continuous space (e.g., PRM, RRT)
  - – Not directly applicable to discrete spaces induced by contact changes (e.g., pickup/putdown)
- Obvious solution:
  - Use task planner for discrete actions
  - Implement those actions using continuous planner

Discrete blocks-world PickUp

PickUp(block1):

precondition $OnTable(block1) \wedge Empty(gripper)$

effect $Holding(block1) \wedge$
$\neg\ OnTable(block1) \wedge$
$\neg\ Empty(gripper)$

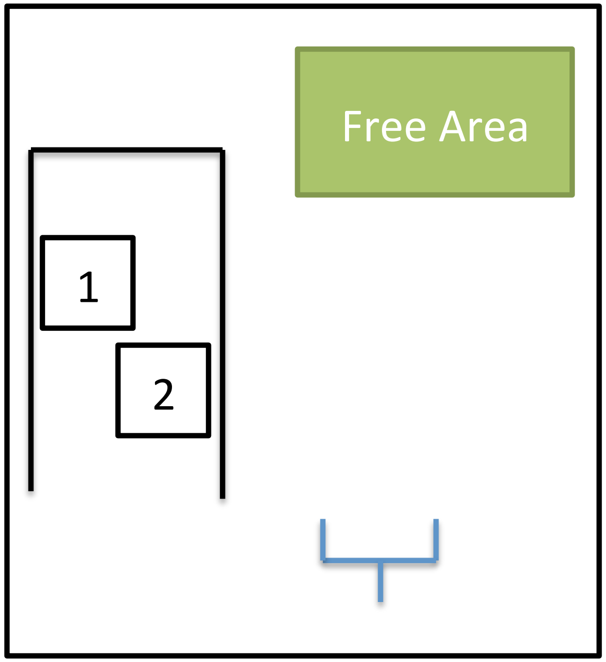
Geometric locations of robot, hand, or object
not considered

A Continuous Version of Blocks World

PickUp(b1, l1, l2, l3, p):

precondition GripperAt(l1) ∧
Empty(gripper) ∧
IsGraspingPose(l2, b1) ∧
At(b1, l3) ∧
∀b2 ¬ Obstructs(b2, p, l1, l2)

effect Holding(b1) ∧
¬ At(b1, l3) ∧
¬ Empty(gripper) ∧
GripperAt(l2)

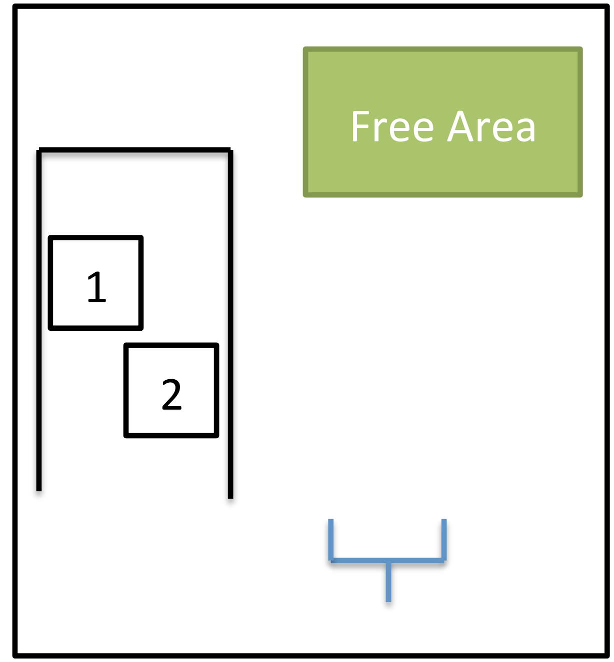


A Continuous Version of Blocks World

PickUp(b1, l1, l2, l3, p):

precondition GripperAt(l1) ∧
Empty(gripper) ∧
IsGraspingPose(l2, b1) ∧
At(b1, l3) ∧
∀b2 ¬ Obstructs(b2, p, l1, l2)

effect Holding(b1) ∧
¬ At(b1, l3) ∧
¬ Empty(gripper) ∧
GripperAt(l2)



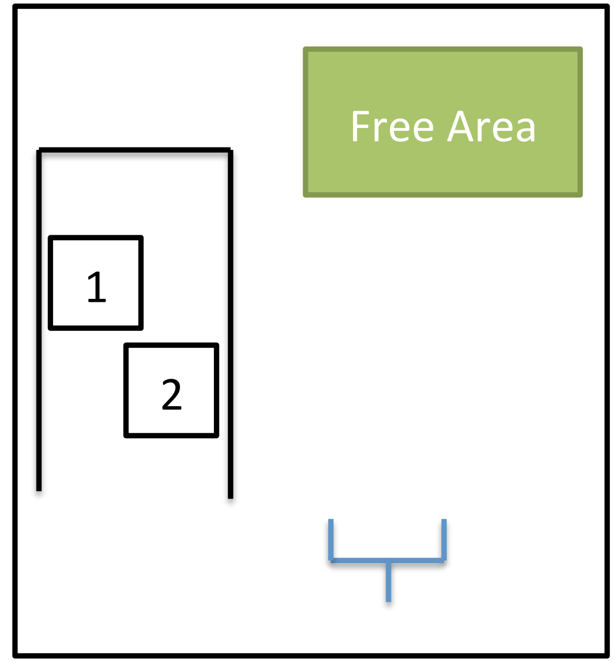
Oops: infinitely many facts, infinite branching factor

A Continuous Version of Blocks World

PickUp(b1, l1, l2, l3, p):

precondition GripperAt(l1) ∧
Empty(gripper) ∧
IsGraspingPose(l2, b1) ∧
At(b1, l3) ∧
∀b2 ¬ Obstructs(b2, p, l1, l2)

effect Holding(b1) ∧
¬ At(b1, l3) ∧
¬ Empty(gripper) ∧
GripperAt(l2)

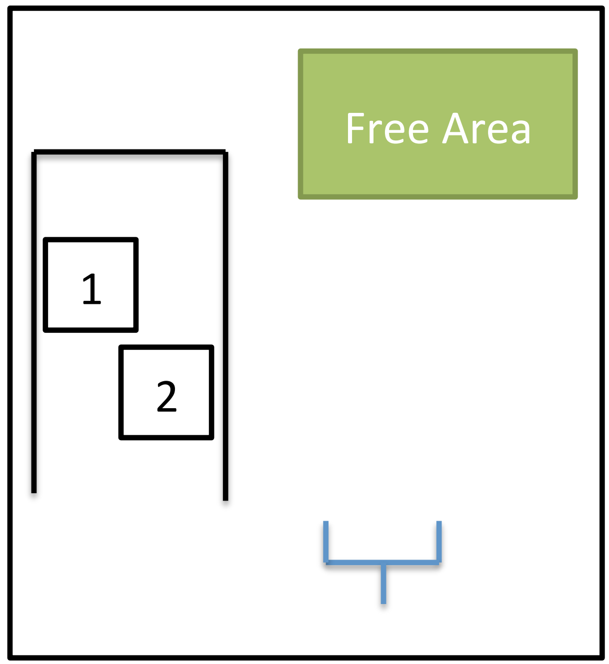


Oops: infinitely many facts, infinite branching factor

Solution: discretization

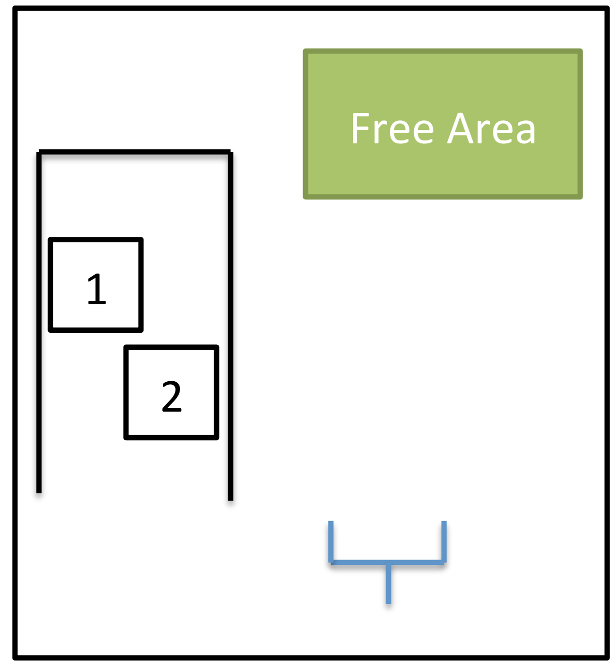
Discretization

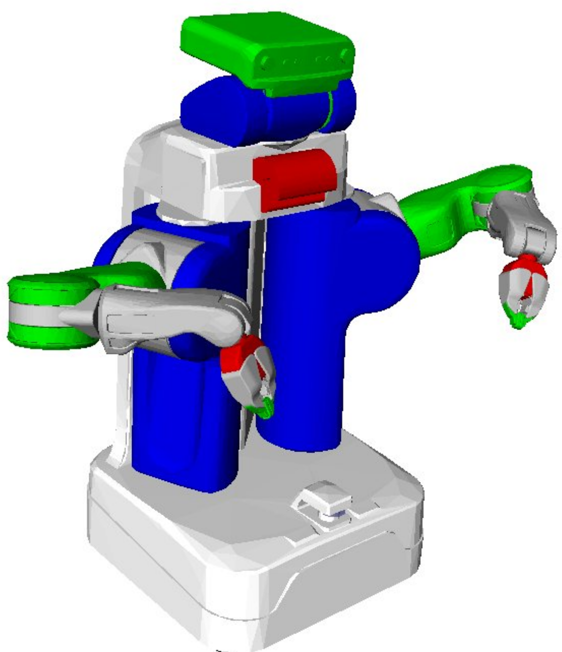
- 10 points each in x, y
- Precompute
  - IsGraspingPose(l, b)
  - Obstructs(b, p, l1, l2)
- 5 objects = 50,000 facts



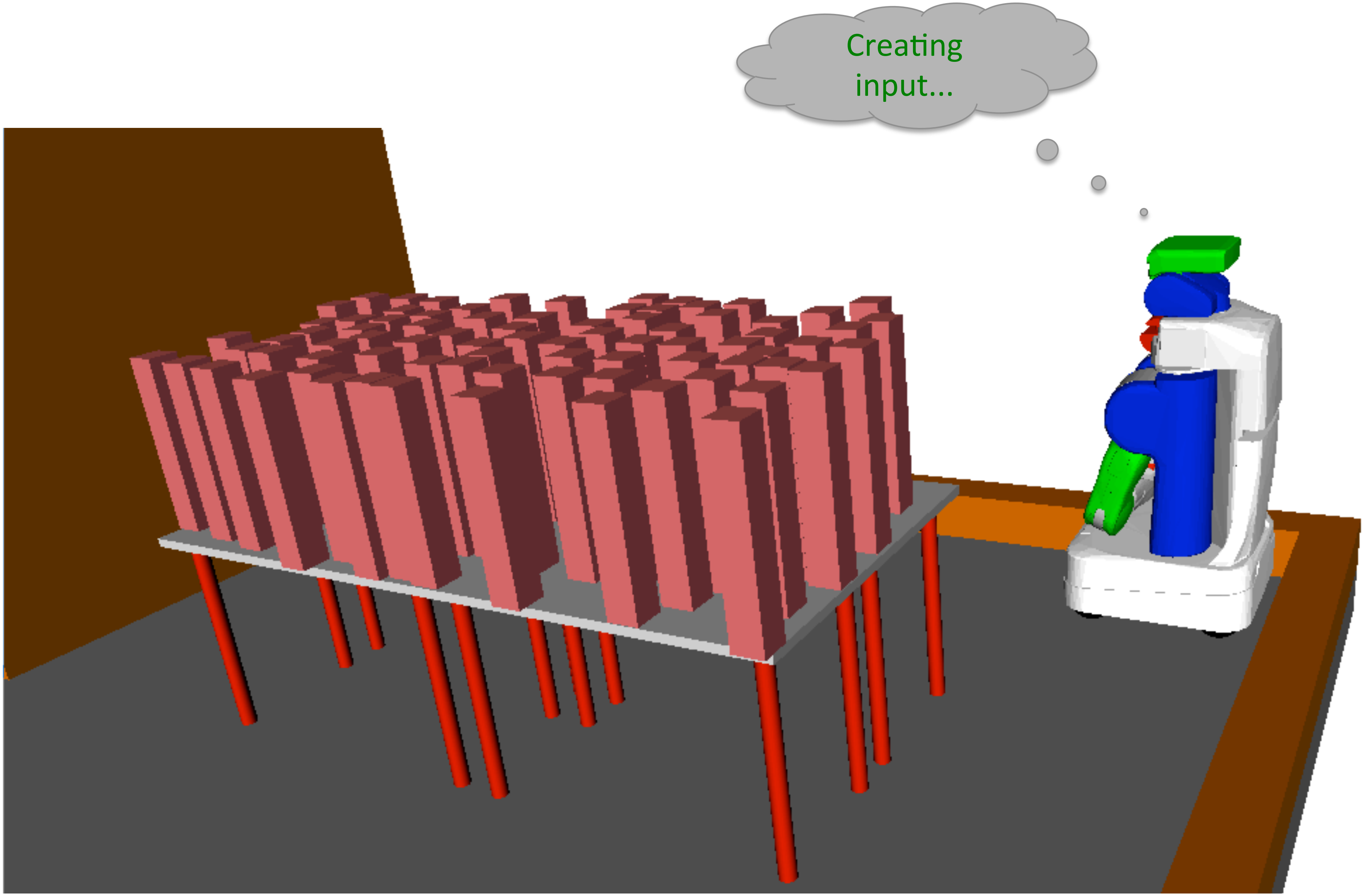
Discretization

- 10 points each in x, y
- Precompute
  - IsGraspingPose(l, b)
  - Obstructs(b, p, l1, l2)
- 5 objects = 50,000 facts





7DOF arm + 4DOF base/torso + 80 objects =~ 10^{14} facts


Creating
input...

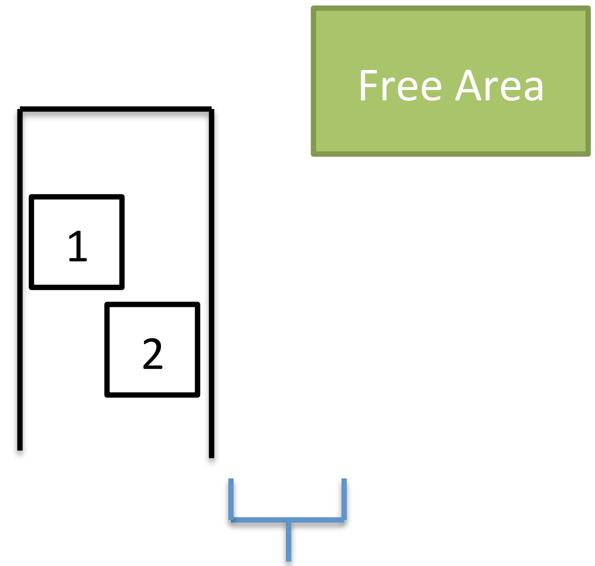
Our approach

- PDDL planner uses "location references"
  - Number of references depends on number of objects and on discrete plan size – no discretization
  - Low-level motion planner interprets these references
- Low-level infeasibility is re-expressed as new PDDL facts about obstructions
  - Expressed using location references
- PDDL planner replans with new information

A SIMPLE EXAMPLE

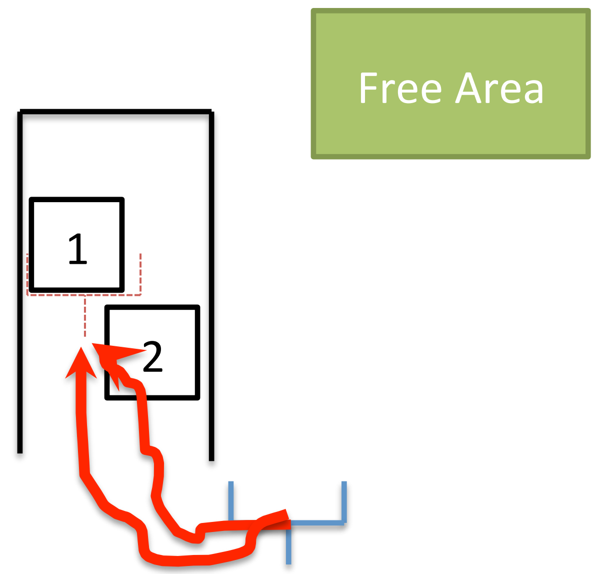
Discrete state: GripperAt(initLoc), At(block1, block1_loc), At(block2, block2_loc)

- High level intuitive plan:
  - pick block1 after going to its grasping pose



Discrete state: GripperAt(initLoc), At(block1, block1_loc), At(block2, block2_loc)

- High level intuitive plan:
  - pick block1 after going to its grasping pose

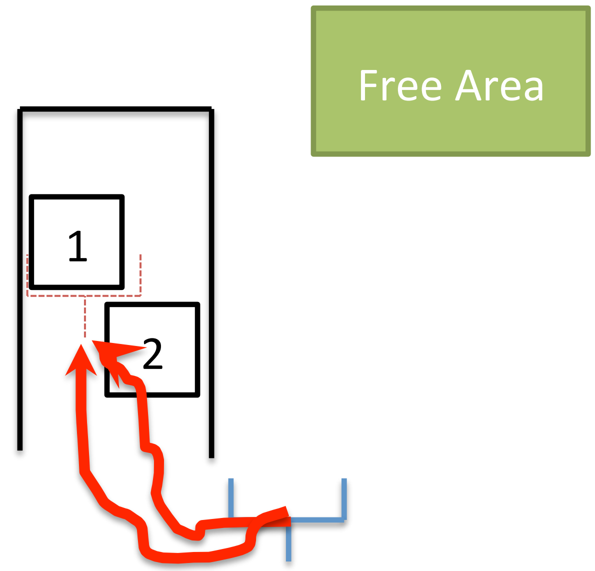


1. Low level instantiates a grasping pose for block 1 independent of other block
2. Low level searches for a motion plan to reach grasping pose; finds no collision-free solution

Discrete state += “block2 obstructs grasping pose for block1 in path from initial location”

- High level intuitive plan:
  - ~~pick block1 after going to its grasping pose~~

Failed



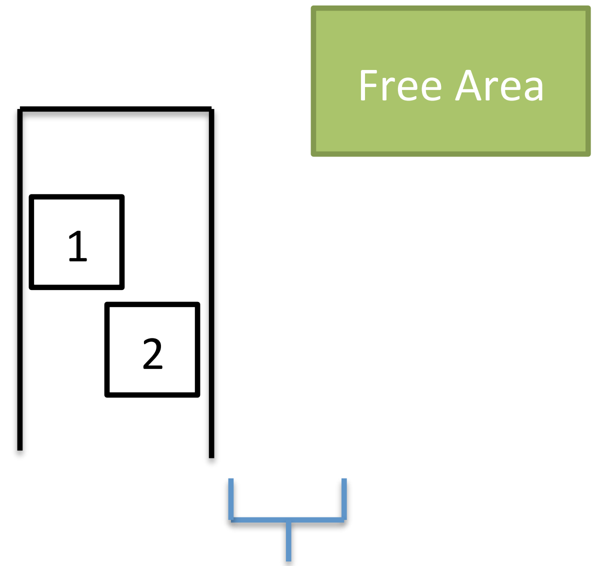
1. Low level instantiates a grasping pose for block 1 independent of other block
2. Low level searchers for a motion plan to reach grasping pose; finds no collision-free solution
3. Reports obstruction to high level

“block2 obstructs grasping pose for block1 from initial location”

Discrete state += “block2 obstructs grasping pose for block1 in path from initial location”

- High level intuitive plan:
  - ~~pick block1 after going to its grasping pose~~

  REPLAN
  - pick block2 after going to its grasping pose
  - release block2 in after going to release pose for free area
  - pick block1 after going to its grasping pose



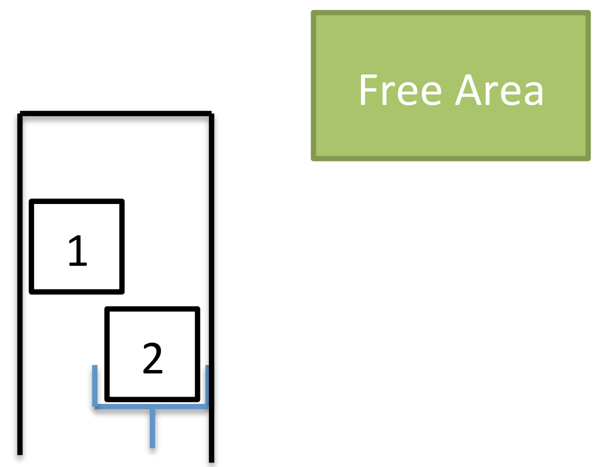
1. Low level instantiates a grasping pose for block 1 independent of other block
2. Low level searchers for a motion plan to reach grasping pose; finds no collision-free solution
3. Reports obstruction to high level
4. High level updates state, replans

Discrete state diff: GripperAt “grasping pose for block2”, Holding(block2)

- High level intuitive plan:
  - ~~pick block1 after going to its grasping pose~~

  REPLAN

  - pick block2 after going to its grasping pose
  - release block2 in after going to release pose for free area
  - pick block1 after going to its grasping pose

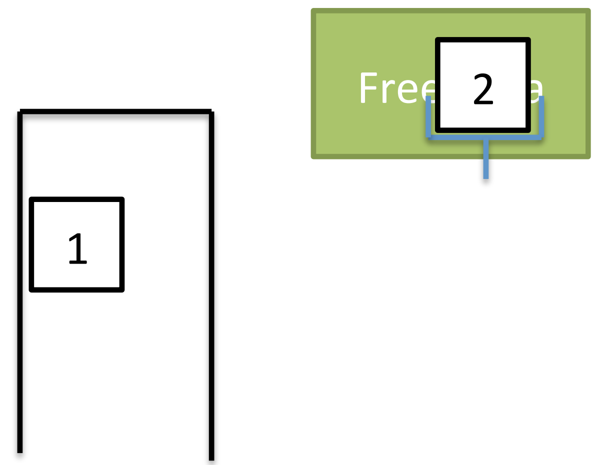


Discrete state diff: At(block2, FreeArea), Empty(gripper)

- High level intuitive plan:
  - ~~pick block1 after going to its grasping pose~~

  REPLAN

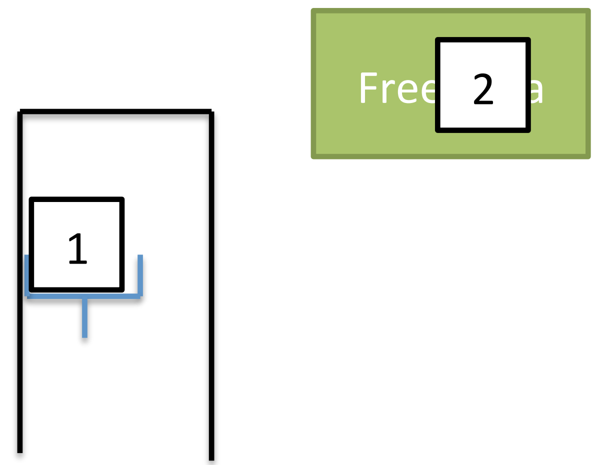
  - pick block2 after going to its grasping pose
  - release block2 in after going to release pose for free area
  - pick block1 after going to its grasping pose



Discrete state diff: GripperAt “grasping pose for 1”, Holding(block1)

- High level intuitive plan:
  - ~~pick block1 after going to its grasping pose~~

  REPLAN
  - pick block2 after going to its grasping pose
  - release block2 in after going to release pose for free area
  - pick block1 after going to its grasping pose



Goal Reached!

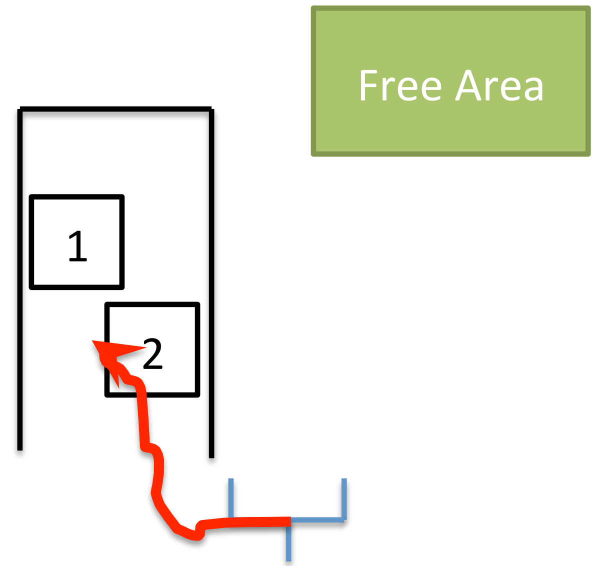
SAME EXAMPLE IN FORMAL SYNTAX

Discrete state += Obstructs(block2, initLoc, gp(block1), path(initLoc, gp(block1)))

- High level intuitive plan:
  - ~~PickUp(block1, initLoc, gp(block1), loc(block1),path(initLoc,gp(block1)))~~ Failed

  REPLAN

  - PickUp(block2, initLoc, gp(block2), loc(block2),path(initLoc,gp(block2)))
  - PutDown(gp(block2), free_area, rp(free_area),path(gp(block2), rp(free_area)))
  - PickUp(block1, rp(free_area), gp(block1), loc(block1), path(rp(free_area), gp(block1)))

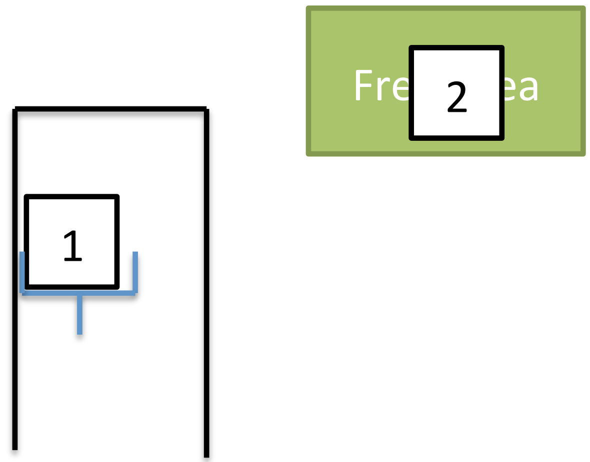


Discrete state diffs: GripperAt(gp(block1)), Empty(gripper), Holding(block1)

- High level intuitive plan:
  - ~~PickUp(block1, initLoc, gp(block1), loc(block1),path(initLoc,gp(block1)))~~

  REPLAN

  - PickUp(block2, initLoc, gp(block2), loc(block2),path(initLoc,gp(block2)))
  - PutDown(gp(block2), free_area, rp(free_area),path(gp(block2), rp(free_area)))
  - PickUp(block1, rp(free_area), gp(block1), loc(block1), path(rp(free_area), gp(block1)))



Goal Reached!

WHY DOES IT WORK??

Actions with Continuous Arguments

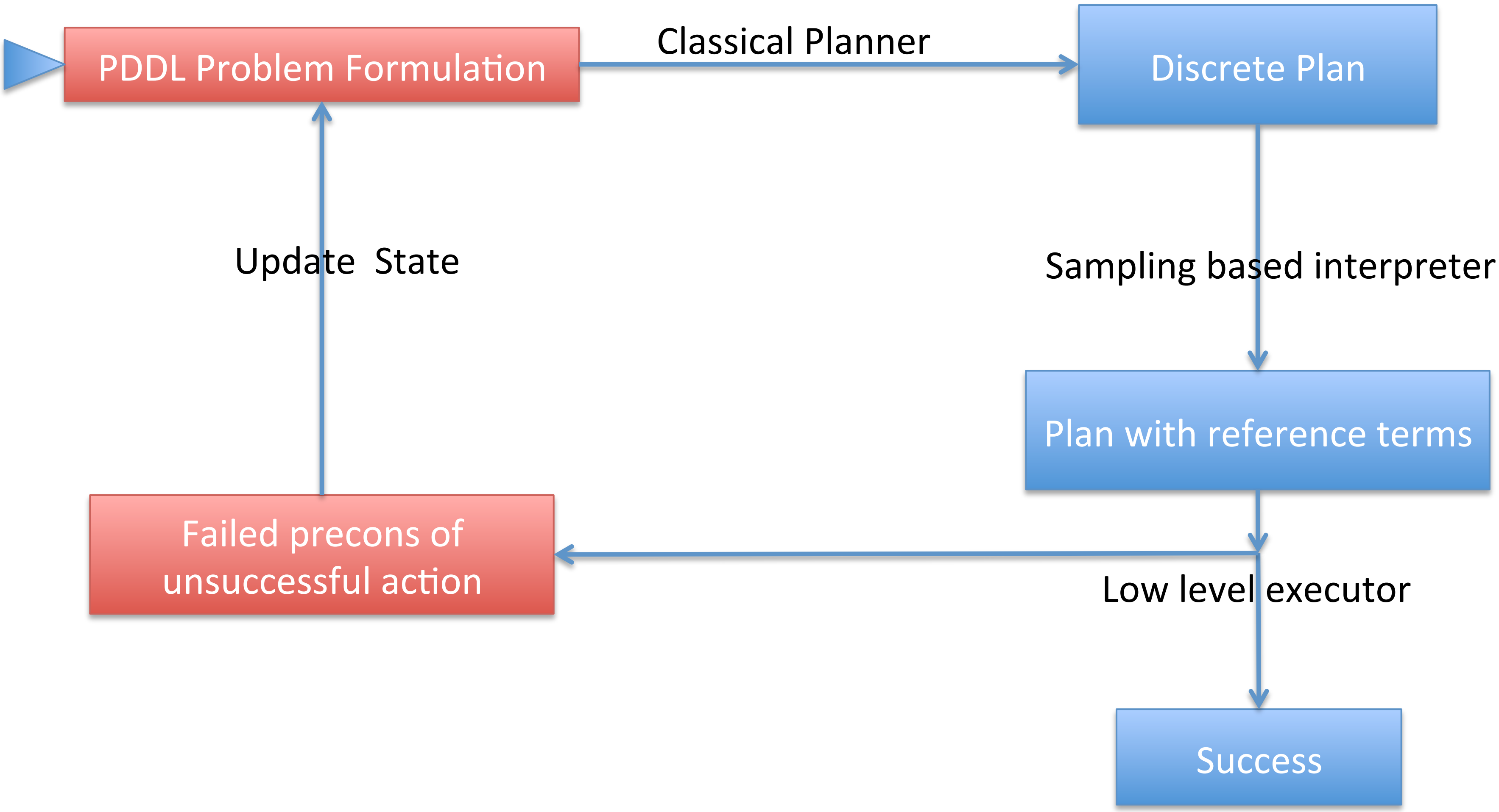
- Effect axioms for actions like "grasp" have the form

  $\forall x \forall y\ (p(x,y) \Rightarrow q(x) \wedge r(x,g(y)))$

  where p is the precondition, q is the post-condition

  x: object, y: continuous arguments
- In order to apply the action to achieve $q(x)$, need to find *some* y (from infinitely many) satisfying $p(x,y)$
- Treat low-level motion planner as an unknown function $f()$ s.t. $p(x, f(x))$ holds
- Planner can assume facts: $p(x, f(x))$ for each x
  - Treat "$f(x)$" like any other object in the world

Overall Approach



Sufficient Conditions for Guaranteed Solutions

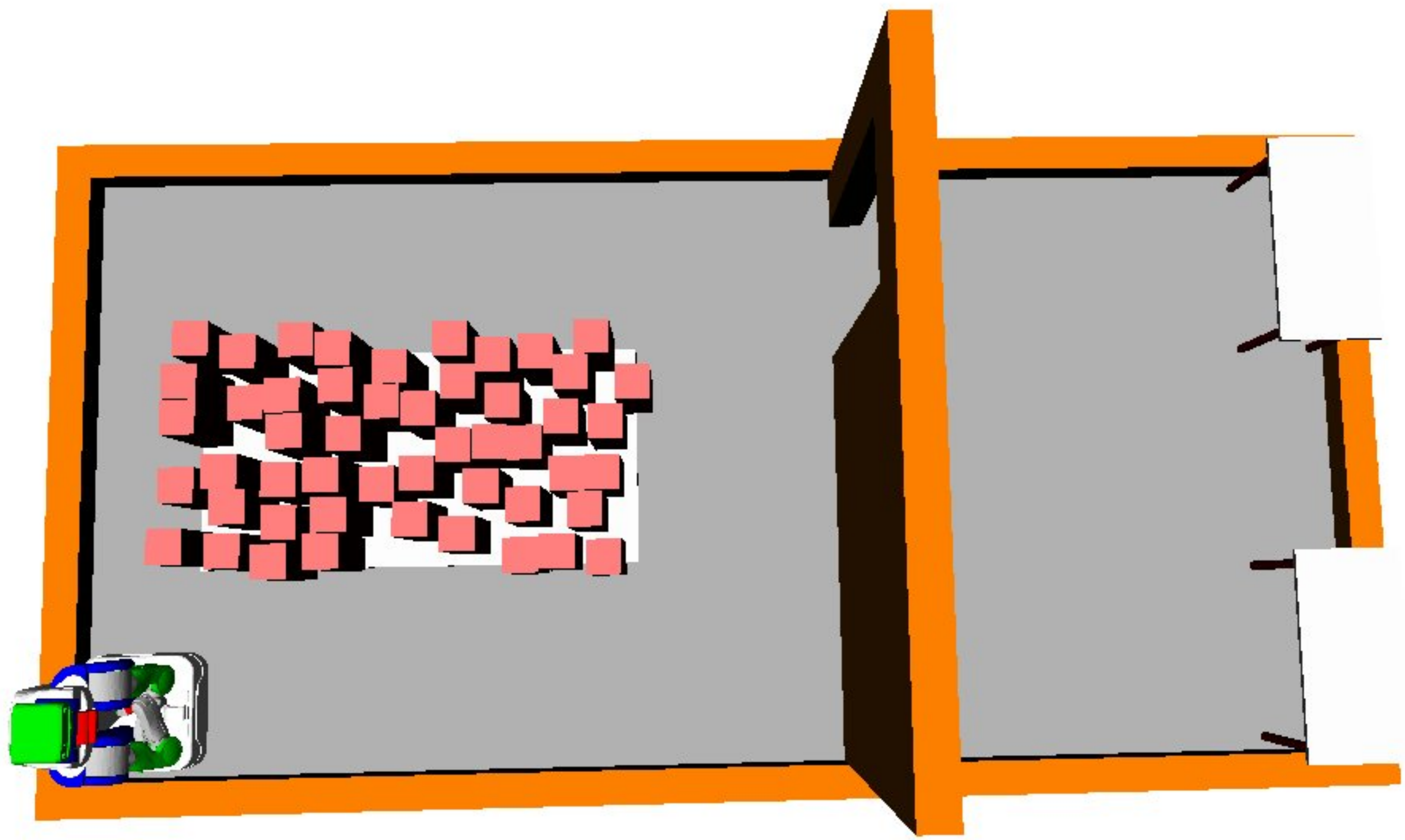
- Standard limitations of replanning:
  - Initial PDDL model is incorrect, but algorithm may act anyway
  - Can fail with dead ends and infinite loops
- BUT the model does improve with every non-executable action
- Theorem: Algorithm is sound and complete provided:
  - Low level sampling terminates, succeeds when possible
  - Problem has no dead ends
  - Negative geometric preconditions can be deleted but not added
  - Positive geometric preconditions can be added but not deleted
- For details, see paper or ask Siddharth

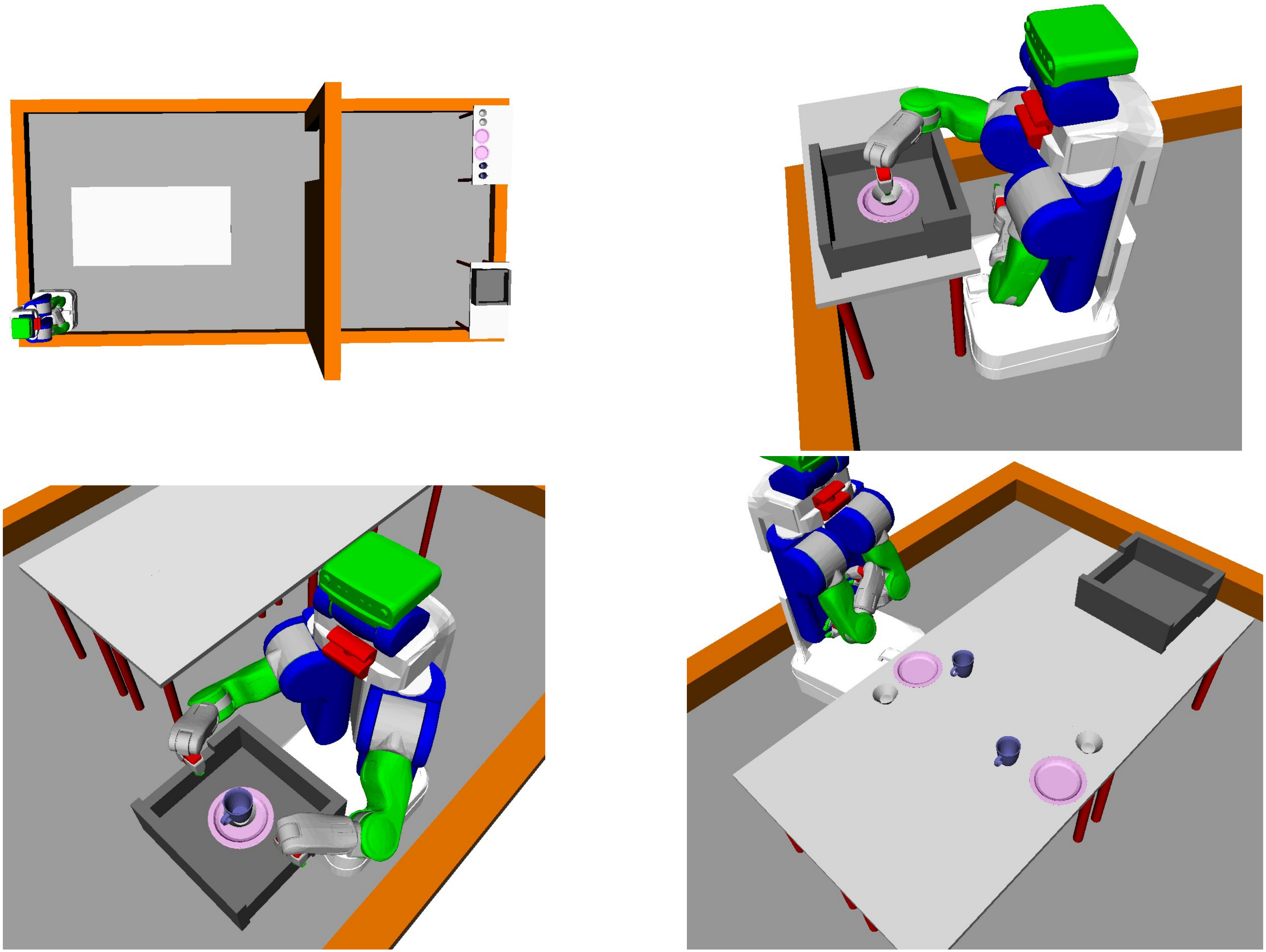
RESULTS ON A PR2 SIMULATOR

Experiments

- Used OpenRave for simulation, IK and grasp computation
- Scenario 1: pick and place with obstructions
  - Many (50, 65, 80) randomly placed objects
  - 3 tests (50, 65, 80 objects), 10 runs each
  - Used FF planner (optimality not a concern)
- Scenario 2: setting a dinner table
  - 2 cups, 2 mugs, 2 plates to be placed at predefined locations
  - Tray available to carry multiple objects
  - Stability constraints for item stacking not known a priori
  - Used FD anytime planner with timeout

Cluttered Table, 50 Objects





Results

- Cluttered table, averages over 10 runs:

#Objects	Time(s)	#Replan	# Obstrns
50	139	2.1	1.8
65	228	2.6	2.0
80	602	2.3	2.6

  - Most of the time spent in low level planning*
- Dinner table: planning + execution time ~230s
  - Most of the time was spent in high level planning

Simulations

Non-simulations

Conclusions

- A method for using classical planners with motion planners in a modular fashion
  - Avoiding exponential discretization complexity
  - Solution based on naming just the discrete-plan-relevant locations with uninterpreted functions
  - Execution errors must be observable and expressible as new PDDL facts

- Still works with no internal low-level model
- Alternative algorithmic approaches could yield stronger guarantees given a low-level simulator