MoveIt! Task Constructor
A framework for planning task sequences

Robert Haschke$^1$, Michael Görner$^2$

$^1$Center of Excellence Cognitive Interaction Technology (CITEC), Bielefeld University, Germany
$^2$TAMS Group, Hamburg University, Germany
Motivation

MTC build instructions: https://github.com/rhaschke/lecture/wiki/MoveIt-Task-Constructor
Objectives

• Definition + Planning of non-trivial manipulation sequences
  • Modular
  • Customizable
• Multiple arms/hands (cf. Felix‘ talk)
• Cost-ranking of alternative solutions
• Understandable failure cases (cf. Felix‘ UX remarks)
• Combine various planners (cf. Pilz‘ talk)

• Replace MoveIt‘s manipulation pipeline
  • Limited to single-arm pick-and-place
  • No introspection

• No Symbolic Task Planning
  • Assuming task structure is known
  • Planning on level of alternative solution paths
Overview

- **Pipeline** composed from **Stages**
- Each stage connects a *start* to an *end InterfaceState* via 1...n **SubSolutions**
• **Pipeline** composed from **Stages**
• Each stage connects a *start* to an *end InterfaceState* via 1...n **SubSolutions**
• **Pipeline** composed from **Stages**
• Each stage connects a *start* to an *end* **InterfaceState** via 1...n **SubSolutions**

• Stages interface each other via *list* of **InterfaceStates**
• Solution = fully-connected path through pipeline
Overview

- **Pipeline** composed from **Stages**
- Each stage connects a *start* to an *end** **InterfaceState** via 1...n **SubSolutions**

- Stages interface each other via *list* of InterfaceStates
- Solution = fully-connected path through pipeline

- **InterfaceState**
  - MoveIt’s *PlanningScene*
  - Properties, e.g.
    - grasp type
    - end effector to use for grasping
Hierarchical Structuring

- SerialContainer
  - *Sequential* chaining of sub tasks
- ParallelContainer
  - Alternatives
    - Consider all solutions of children
  - Fallback
    - Consider children one by one
  - Merger
    - Combine solutions of children for parallel execution
    - Example: arm approaching + hand opening
    - Requires extra feasibility check!
- Wrapper
  - Filter / duplicate / modify solutions
InterfaceStateLists: Implementation Details

- Each stage has its own starts/ends interface if *reading from* there
- Not instantiating the interface, indicates that the stage is *not* reading from that direction
InterfaceStateLists: Implementation Details

- Each stage has its own starts/ends interface if reading from there
- Not instantiating the interface, indicates that the stage is not reading from that direction
- The pointers `prev_ends_` and `next_starts_` reference to the ends / starts interface of the previous / next stage. They indicate whether the stage is `writing` in that direction
Stage Types: SerialContainer

- serially chain several stages
- a solution is any path connecting any start to any end state
- container interface
  - starts_ / ends_: incoming from prev / next sibling stage
    forwarded to first / last child
Stage Types: SerialContainer

- serially chain several stages
- a solution is any path connecting any start to any end state
- container interface
  - starts_ / ends_: incoming from prev / next sibling stage 
    forwarded to first / last child
  - onNewSolution: lift full solution(s) to external InterfaceList
Semantic Stage Types

• Planning proceeds non-linearly:
  • generators: seed for planning
  • propagation: advance partial solutions
  • connectors: connect partial solutions

• Example: Pick-n-Place with Handover

↑ current state
∞ connect
↑ pick with right hand
↓ move to handover pose
∞ connect
↑ pick with left hand
↓ move to place
Semantic Stage Types

• Planning proceeds non-linearly:
  • generators: seed for planning
  • propagation: advance partial solutions
  • connectors: connect partial solutions

• Example: Pick-n-Place with Handover

↑ current state
∞ connect
↑ pick with right hand
↓ move to handover pose
∞ connect
↑ pick with left hand
↓ move to place
Semantic Stage Types

• Planning proceeds non-linearly:
  • generators: seed for planning
  • propagation: advance partial solutions
  • connectors: connect partial solutions

• Example: Pick-n-Place with Handover

 shorthand:

 current state
 connect
 pick with right hand
 move to handover pose
 connect
 pick with left hand
 move to place
Semantic Stage Types

- Planning proceeds non-linearly:
  - generators: seed for planning
  - propagation: advance partial solutions
  - connectors: connect partial solutions

- Example: Pick-n-Place with Handover

↑ current state
∞ connect
↑ pick with right hand
↓ move to handover pose
∞ connect
↑ pick with left hand
↓ move to place
Stage Types by Interface

- Type determined by what is read from / written to interfaces

- Generator
  - No reading, Write to both interfaces
  - Examples: CurrentState, FixedState, GraspGenerator

- Propagator
  - Read from one, write to opposite interface
  - Examples: Approach, Lift

- Connector
  - Read both interfaces
  - Combinatorial explosion
  - Check compatibility of states
Available Primitive Stages

- Generators
  - Fetch current Planning Scene from move_group
  - Cartesian pose generator / sampler
  - ComputeIK
  - Simple grasp generator

- Propagators
  - MoveTo: plan towards absolute goal
  - MoveRelative: plan relative motion
  - Manipulate Planning Scene
    - Attach / Detach objects
    - Modify ACM

- Connect
Connect

• Connect 2 InterfaceStates via planning
• Might involve multiple planning groups
  – Arm(s)
  – Hand(s)

• Approach:
  – List all groups with corresponding planners
  – Plan for groups in given sequence
  – Try to merge trajectories for parallel execution
Planners

- Individual stages can employ different planners
- MoveIt's PipelinePlanner
- OMPL
- STOMP
- CHOMP
- ...
- Straight-line Cartesian path
- Straight-line Joint-space path
```cpp
Task task;
task.add(std::make_unique<stages::CurrentState>());

auto cartesian = std::make_shared<solvers::CartesianPath>();
// Cartesian motion along a vector in world
auto move = std::make_unique<stages::MoveRelative>("x", cartesian);
move->setGroup(„panda_arm“);
geometry_msgs::Vector3Stamped direction;
direction.header.frame_id = "world";
direction.vector.x = 0.2;
move->setDirection(direction);
task.add(std::move(move));
...
```

```
$ roslaunch moveit_task_constructor_demo demo.launch &
$ rosrun moveit_task_constructor_demo cartesian
```
Basic Example: C++

```cpp
...
// create an arbitrary twist motion relative to current pose
move = std::make_unique<stages::MoveRelative>("z", cartesian);
move->setGroup("panda_arm");
geometry_msgs::TwistStamped twist;
direction.header.frame_id = "world";
twist.twist.angular.z = M_PI / 4.;
move->setDirection(twist);
task.add(std::move(move));
...
```

$ roslaunch moveit_task_constructor_demo demo.launch &
$ rosrun moveit_task_constructor_demo cartesian
Basic Example: C++

```cpp
// move from reached state back to the original state
Connect::GroupPlannerVector planners = {{{"panda_arm", cartesian}}};
auto connect = std::make_unique<Connect>("connect", planners);
task.add(std::move(connect));

// final state is original state again
task.add(std::make_unique<CurrentState>()) ;
```

$ roslaunch moveit_task_constructor_demo demo.launch &
$ rosrun moveit_task_constructor_demo demo cartesian
Basic Example: C++

```cpp
// move from reached state back to the original state
auto ji = std::make_shared<solvers::JointInterpolationPlanner>();
Connect::GroupPlannerVector planners = {{"panda_arm", ji}};
auto connect = std::make_unique<Connect>("connect", planners);
task.add(std::move(connect));

// final state is original state again
task.add(std::make_unique<CurrentState>();
```

$ roslaunch moveit_task_constructor_demo demo.launch &
$ rosrun moveit_task_constructor_demo cartesian
Basic Example: Python

```python
task = core.Task()

# start from current robot state
task.add(stages.CurrentState("current state"))

# Cartesian motion along x
move = stages.MoveRelative("x +0.2", core.CartesianPath())
move.group = group

dir = Vector3Stamped(header=Header(frame_id = "world"),
                     vector=Vector3(0.2,0,0))

move.setDirection(dir)
task.add(move)
```

Basic Example: Python

```bash
$ roslaunch moveit_task_constructor_demo demo.launch &
$ rosrun moveit_task_constructor_demo cartesian.py
```
Basic Example: Python

task = core.Task()

# start from current robot state
task.add(stages.CurrentState("current state"))

# Cartesian motion along x
move = stages.MoveRelative("x +0.2", core.CartesianPath())
move.group = group

dir = Vector3Stamped(
    header=Header(frame_id = "world"),
    vector=Vector3(0.2,0,0))

move.setDirection(dir)
task.add(move)
...

$ roslaunch moveit_task_constructor_demo demo.launch &
$ rosrunc moveit_task_constructor_demo cartesian.py
Basic Example: Python

...

```python
# moveTo named posture
move = stages.MoveTo("moveTo ready", cartesian)
move.group = group
move.setGoal("ready")
task.add(move)

if task.plan():
    task.publish(task.solutions[0])
```

$ roslaunch moveit_task_constructor_demo demo.launch 
$ rosrun moveit_task_constructor_demo cartesian.py
Containers as Wrappers for reusable sub tasks

- Combine stages into reusable sub tasks
- Examples: Pick / Place or Grasp / Release

<table>
<thead>
<tr>
<th>Pick</th>
<th>Place</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Approach</td>
<td>- Place</td>
</tr>
<tr>
<td>- Grasp</td>
<td>- UnGrasp</td>
</tr>
<tr>
<td>- Lift</td>
<td>- Retract</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Grasp</th>
<th>UnGrasp</th>
</tr>
</thead>
<tbody>
<tr>
<td>- ComputeIK</td>
<td>- ComputeIK</td>
</tr>
<tr>
<td>- GraspProvider</td>
<td>- PlaceProvider</td>
</tr>
<tr>
<td>- Allow Object Collision</td>
<td>- Detach Object</td>
</tr>
<tr>
<td>- Close Gripper</td>
<td>- Open Gripper</td>
</tr>
<tr>
<td>- Attach Object</td>
<td>- Forbid Object Collision</td>
</tr>
</tbody>
</table>
Property Inheritance

- Need a method to derive stage properties
  - from parent
  - from passed-in solution
- Explicit property handling
  - declared with name and type
  - explicit inheritance or forwarding
- `Property::configureInitFrom(source, const InitializerFunction& f);`
- `Property::configureInitFrom(source, other_name);`
- `PropertyMap::configureInitFrom(source, names);`
- `source = PARENT | INTERFACE`

```bash
$ roslaunch moveit_task_constructor_demo demo.launch &
$ rosrun moveit_task_constructor_demo_demo modular
```
MonitoringGenerator

- Generator might need input from a remote stage
- Grasp/Place an object at the current position
- MonitoringGenerators hook into solutions of another stage

↑ current state
→ connect
↑ pick
→ connect
↓ place
Providing Custom Stages

class MyStage : public PropagatingForward {
public:
    MyStage(string name);

    void computeForward(const InterfaceState& from) override {
        ...
        SubTrajectory solution(trajectory, cost, comment);
        solution.markers().push_back(marker);
        sendForward(from, move(end_scene), move(solution));
    }
};
Outlook: Envisioned Features

- Drop-In replacement for MoveIt‘s Pick+Place capability
- Interactive GUI
  - Configure + validate task pipeline in rviz
    - Save / load YAML
    - C++ / python code generation
- Execution Handling
  - Premature execution of planned sub tasks
  - Choose controllers for sub tasks (force control, servoing)
- Failure handling
  - Replan from current situation
  - Revert to previous stage
Scheduling

- Find „good“ solutions fast!

- Priority queues @ different levels
  - InterfaceState: remember best solution only
  - InterfaceStateList: sort by length and accumulated cost of partial solution
  - Stage scheduling (TODO)
    - Interface type
    - success rate
    - estimated computation time

- Compute stages in parallel threads
Cost Functions

• Currently costs explicitly computed in stages
• Future: Provide set of cost functions to choose from
  – accumulated amount of joint-space / Cartesian motion
  – distance from preferred pose
  – clearance to obstacles
  – ...
• Generic mechanism to set cost functions per stage
• Plugins?
• What are stage-specific useful defaults?
More Advanced Examples

- **Pick + Place**
  ```
  $ roslaunch moveit_task_constructor_demo demo.launch &
  $ roslaunch moveit_task_constructor_demo pickplace.launch
  ```

- **Bimodal Pick + Place**
  - Choose left or right arm based on costs

- **Long-Distance Pick-and-Place with Handovers**
  [GitHub](https://github.com/ubi-agni/mtc_demos)

- **Pouring**
  [GitHub](https://github.com/TAMS-Group/mtc_pour)