Design & Implementation of Cognition-enabled Robot Agents

Module 7: Planning and Execution

Lecture 1: Action Models for Mobile Manipulation

Institute for Artificial Intelligence
Universität Bremen

Winter Term 2020/21

Learning Goals

- Analyze the challenges of mobile manipulation in robotics
- Explain causes of execution failures and contrast different failure handling approaches
- Formulate reactive robot plans

Lecture 1
Action Models for Mobile Manipulation

Mobile Manipulation Actions

Challenges Tackled by the Plan Executive

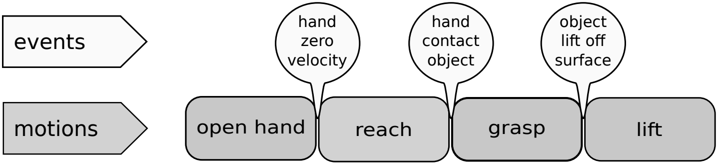
1. Define which actions to execute to achieve the goal.
2. Infer which parameters to use for each action.
3. Monitor task execution and react to failures.

Motion Model

Segmentation Based on Force-Contact Events

The part of the plan that executes an atomic trajectory is a **motion**.

We use the Flanagan model[1] for segmentation + a zero velocity event:



Motion model of the picking-up action

[1] J Randall Flanagan, Miles C Bowman, Roland S Johansson, "Control strategies in object manipulation tasks", in *Current Opinion in Neurobiology*, Volume 16, Issue 6, 2006, Pages 650-659, ISSN 0959-4388, https://doi.org/10.1016/j.conb.2006.10.005

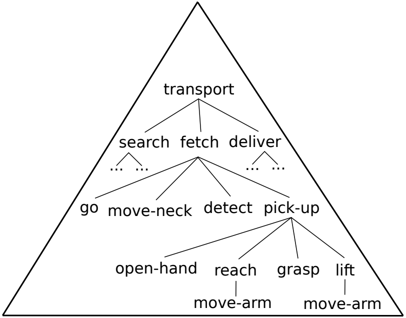
Primitives: Motions and Percepts

Primitives of Mobile Pick and Place for PR2-like Robots

Primitive	Description
going	drive or walk or fly to the goal pose
moving-torso	move torso to the goal joint position
moving-neck	move the neck to direct the gaze
moving-arm	execute a trajectory in Cartesian or joint space
grasping/releasing	move the fingers to grasp or release an object
opening-hand/cl.	move the fingers to open or close the hand
monitoring-joints	monitor the positions of robot body parts in space
detecting	perceive the described object in the environment
moving-eye	move the eye in the socket to direct the gaze
…	

Action Model

Model of Mobile Pick & Place and a Simple Plan Written in CPL




```
def_plan reach (goal ...)
  perform (a primitive
             (type moving-arm)
             (cart-goal ?goal)
             ...)

def_plan pick_up (...)
  perform (a primitive
             (type opening-hand))
  perform (an action
              (type reaching) ...)
  perform (a primitive
             (type grasping) ...)
  perform (an action
              (type lifting) ...))
```

Summary

- This lecture:
  - motion model
  - primitives
  - sequence of primitives
  - actions
  - action hierarchy
- Next lecture:
  - parameters of primitives and actions
- For hands-on experience in programming cognition-enabled robots, check out this zero-prerequisite tutorial: `http://cram-system.org/tutorials/demo/fetch_and_place`

Design & Implementation of Cognition-enabled Robot Agents

Module 7: Planning and Execution

Lecture 2: Motion Parameters and Execution Failures

Institute for Artificial Intelligence
Universität Bremen

Winter Term 2020/21

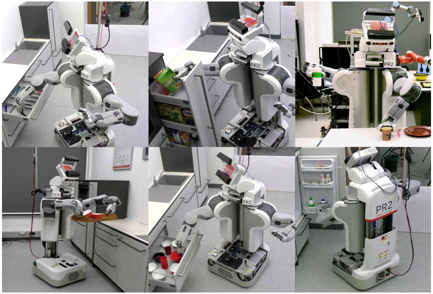
Lecture 2
Motion Parameters and Execution Failures

Parameters of Motion and Perception Primitives

Primitive	Parameters
going	goal_pose, ..., speed, ...
moving-torso	goal_position, ...
moving-neck	goal_positions, goal_coordinate_to_look_at, ...
moving-arm	goal_pose_for_hand, goal_positions, collisions, ...
grasping/releasing	hand, grasping_force, object_properties, ...
opening-hand/cl.	hand, ...
monitoring-joints	joint_name, joint_value, monitoring_function, ...
detecting	object_description, ...

Calculating parameter values that maximize success probability: heuristics, learning from experience, imitation learning, ask a human, ...

Choice of Parameter Values is Crucial For Success



- Often very many possible values to choose from

  Example: from which side and with which hand to grasp?

- Effects can be:
  - immediate
  - short-term
  - long-term

Perceiving Goal States and Detecting Failures

Ensuring that the goal was achieved can be done through:

- extrinsic perception of the scene
  - after placing the object, perceive the scene to ensure that it is actually there
- intrinsic perception of the robot's body part positions with respect to each other
  - ensure that the arm / base / neck reached the goal
  - ensure that the gripper did not close completely if an object was expected to be grasped
- other kinds of perception
  - estimate the weight of the object in the hand
  - use tactile perception
  - react to sounds, smells, etc.

Failure Types

- Low-level (primitive) vs high-level (action) failures
  - low-level failures are thrown if a primitive was not successful, e.g., *going_low_level_failure*, *arm_low_level_failure*, *hand_low_level_failure*, *neck_low_level_failure*, *torso_low_level_failure*, *perception_low_level_failure*, ...
  - high-level failures are thrown if an action did not succeed, e.g., *picking_up_failure*, *searching_failure*
- Planning time vs execution time vs post-execution failures
  - planning time failures are thrown if the robot anticipates that the action will fail **before** executing it, e.g., by using simulation
  - execution time failures are signaled if a deviation from the intended course of action is detecting **during** action execution
  - post-execution failures are those that are detected **after** action execution has finished

Summary

- This lecture:
  - parameters of primitives and actions,
  - failures,
  - failure taxonomy.
- Next lecture:
  - failure handling.

Design & Implementation of Cognition-enabled Robot Agents

Module 7: Planning and Execution

Lecture 3: Failure Handling

Institute for Artificial Intelligence
Universität Bremen

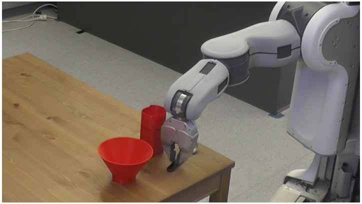
Winter Term 2020/21

Lecture 3
Failure Handling

Strategy 1: Retrying Without Change

The sequence of actions that aims to negate the unwanted effects of the failure, together with the new sequence of actions that leads to success, is the **failure handling strategy**.

- The real world is non-deterministic:
  executing the same action the same way can have different effects.
- Simplest strategy: simply retry executing the action the same way.
- Example: if grasping failed, simply try to perceive and grasp again.



Retrying Without Change Strategy in CPL

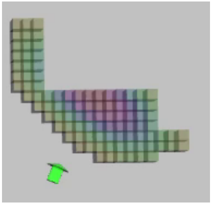
```
with_retry_counters grasp_counter = 3
  with_failure_handling

      perform (an action
                  (type detecting)
                  (object (an object
                              (type spoon)))
                  ...)
      perform (an action
                  (type picking-up)
                  (object (the object
                              (type spoon))))
                  (hand right-hand)
                  (grasp top-grasp)
                  ...)

    catch grasping_failure
      do_retry grasp_counter
        retry
```

Strategy 2: Retrying with Changing the Parameters

- Strategy: pick another parameter value and retry
- Parameter values can be represented using a probability distribution
- Choosing the next parameter: random sampling, gradient descent, A*, any other type of search
- If the action has multiple parameters, do a search over all the parameters, the search tree grows exponentially



Retrying with Changing the Parameters in CPL

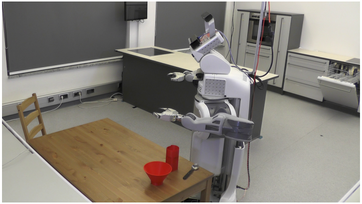
```
robot_base_location = (a location
                        (to open)
                        (object (an object
                                     (type refrigerator)))))
with_retry_counters grasp_counter = 3
  with_failure_handling

      perform (an action
                   (type going)
                   (target ?robot_base_location))))
      perform (an action
                   (type opening)
                   (object (an object
                                (type refrigerator)))
                   (hand right-hand)
                   ...))

    catch grasping_failure
      if exists next(robot_base_location)
        robot_base_location = next(robot_base_location)
        do_retry grasp_counter
          retry
```

Strategy 3: Retrying with Changing the Actions

- Sometimes retrying the action is not sufficient: one needs additional actions
  - If stuck, wiggle yourself to get unstuck.
  - If object cannot be found, move the torso to a different configuration.
  - If everything fails, ask a human for help.
- One can put these if-else cases explicitly as a part of the plan and not the failure handling.
  - For efficiency, skip actions, where the goal has already been achieved.



Strategy 3 in CPL

```
with_retry_counters perception_counter = 3
  with_failure_handling

      perform (an action
                 (type detecting)
                 (object (an object
                             (type spoon)))
                 ...))

    catch perception_failure
      perform (a primitive
                (type moving-torso)
                (joint-position highest)))
      do_retry perception_counter
        retry
```

Summary

- This lecture:
  - failure handling strategies for planning time and post execution time failures.
- For more information on alternative approaches watch this lecture: https://www.youtube.com/watch?v=5R-xL9YmdR0
- Next lecture:
  - failure handling strategies for execution time failures.

Design & Implementation of Cognition-enabled Robot Agents

Module 7: Planning and Execution

Lecture 4: Reactive Planning

Institute for Artificial Intelligence
Universität Bremen

Winter Term 2020/21

Lecture 4
Reactive Planning

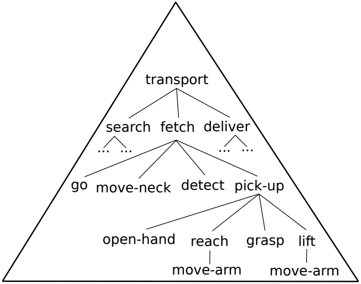
Reactive Planning

- When performing actions, it is sometimes necessary to react to events immediately and to constantly monitor action execution
- Examples:
  - object is slipping → monitor gripper opening angle
  - spilling occurs during pouring → monitor the state of the poured fluid
  - another actor approaches during cutting → monitor the cutting area
- Monitoring strategies detect failures at execution time
- **Reactive plans** are those that describe concurrent-reactive behavior, i.e. behavior where multiple things happen concurrently and the robot reacts to events immediately
- To implement monitoring, we need multithreading
- Multithreading and synchronization code is difficult to write without a proper background

  → need a convenient library for writing concurrent-reactive behavior

CPL Operators for Concurrent-Reactive Behaviors

in_parallel_do: Perform the Child Expressions in Parallel Threads




```
def_plan pick_up (...)
  perform (a primitive
             (type opening-hand)))
  perform (an action
             (type reaching) ...))
  perform (a primitive
             (type grasping) ...))
  perform (an action
             (type lifting) ...)))
```

↓

```
def_plan pick_up (...)
  in_parallel_do
    perform (a primitive (type opening-hand))
    perform (an action (type reaching) ...)
  perform (a primitive (type grasping) ...)
  perform (an action (type lifting) ...)
```

CPL Operators for Concurrent-Reactive Behaviors

try_all, in_sequence_do, try_in_order

- **try_all** performs the children concurrently, like **in_parallel_do** but only fails if all children fail, and succeeds otherwise.

```
try_all
  perform (a primitive
              (type detecting) (object ...)
              (algorithm color-segmentation)))
  perform (a primitive
              (type detecting) (object ...)
              (algorithm depth-segmentation))))
```

- **in_sequence_do** simply performs the children in a sequence.
- **try_in_order** performs in a sequence but only fails if all fail.

```
try_in_order
  perform (an action
               (type picking-up) (object ...) (hand left-hand))
  perform (an action
               (type picking-up) (object ...) (hand right-hand))
```

CPL Operators for Concurrent-Reactive Behaviors

Fluents

- A **fluent** is a data structure for storing values that change over time.
- Fluents are perfect for representing continuous streams of sensor data or any other robot state data. For example:

```
current_pose_fluent = make_fluent()
```

- To keep it updated, one sets the value in the state callback function

```
defun localization_callback (pose_msg)
  current_pose_fluent.fl_value() = get_pose(pose_msg)
```

- Fluents **pulse** when their value changes.
- A **fluent network** is a combination of multiple fluents – **fl_funcall**. It updates its value whenever one of the constituent fluents pulses.

```
fl_funcall <
           fl_funcall euclidean_distance
                      current_pose_fluent.fl_value()
                      goal_location
           distance_threshold
```

CPL Operators for Concurrent-Reactive Behaviors

whenever, wait_for

To react to the changes in the fluent value, one can use the **whenever** and **wait_for** operators of the CPL language.
They block the thread until the fluent pulses.

- **whenever** runs on an infinite loop and executes its body whenever the fluent pulses.

```
whenever current_pose_fluent
  print "Robot just changed its location."
```

- **wait_for** blocks until the fluent pulses once, then continues execution.

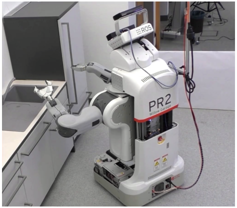
```
wait_for
  fl_funcall <
               fl_funcall euclidean_distance
                          current_pose_fluent.fl_value()
                          goal_location
               distance_threshold
print "Robot reached the goal location."
```

CPL Operators for Concurrent-Reactive Behaviors

pursue: Perform Children in Parallel Until One Finishes

- **pursue** terminates all its child threads when one of the children finishes, i.e. the child that finishes first kills everyone else.
- **pursue** is perfect for implementing monitoring strategies.

```
pursue
  perform (an action (type opening) ...)
  in_sequence_do
    wait_for gripper_closed_completely_fluent
    fail gripper_closed_completely_failure
```



Summary

- This lecture:
  - reacting to execution time failures and events,
  - reactive planning,
  - fluents.
- This module:
  - motion segmentation and action hierarchies,
  - choosing the correct parameters for the actions,
  - taxonomy of failures and failure handling strategies,
  - reactive planning.