Lecture 18: Planning and plan execution, applications

- Planning, plan execution, replanning
- Planner as a part of an autonomous robot: robot architectures
- The subsumption architecture
- The 3-tier architecture: deliberator, sequencer, skills
Interleaved planning and execution: motivation

- Conditional plans may be too big to compute completely.
- If the formalization of the world is not quite correct, there will be a mismatch between the world and the predictions, and planning only once, before execution, might not work.

⇒ plan one action, execute, plan one action, execute, ...
Interleaved planning and execution: general approach

Let \( \Pi = \langle P, I, O, G, B \rangle \) be a problem instance in conditional planning with partial observability.

Let \( I \) be a set of observationally indistinguishable states.

Problem: compute \( o \in O \) so that there exists a conditional plan for \( \Pi \) that starts with \( o \).
Interleaved P & E: general approach

Let $B$ the set of initial states.

1. $B := B \cap C$, i.e. intersect $B$ with the observational class $C$.
2. If $B \subseteq G$, stop.
3. Choose an operator $o \in O$ and execute it, reaching $img_o(B)$.
4. $B := img_o(B)$.
5. Goto 1.
Interleaved P & E: general approach, a problem

What guarantees that consecutive actions make sense?

Goal: go and by a bottle of milk.

action 1: Walk north toward Aldi
action 2: Walk south toward Edeka
action 3: Walk north toward Aldi
action 4: Walk south toward Edeka
action 5: Walk north toward Aldi
...

(Actions are not optimal under some optimality measure.)
Interleaved P & E vs. conditional planning

Interleaved planning and execution does not make planning easier, it just avoids the generation of a possibly huge plan.

THEOREM Testing whether there is an operator $o$ having the following property is 2-EXP-hard.

If there is a conditional plan that reaches $G$ from state $B$, then there is a conditional plan that reaches $G$ from $img_o(B)$.

This is a very weak property that the chosen actions/operators must satisfy.
Interleaved P & E vs. conditional planning

PROOF: We reduce any plan existence problem for conditional planning to the problem.

Let $\Pi = \langle P, I, O, G, B \rangle$ be a problem instance.

Then $\langle a, \neg a \land \neg b \rangle$ preserves the reachability of the goals for

$$\Pi' = \langle P \cup \{a, b\}, I \land a \land \neg b, \{\langle c \land \neg a, e \rangle | \langle c, e \rangle \in O \} \cup \{\langle a, \neg a \land \neg b \rangle, \langle a, b \rangle\}, G \lor b, B \rangle$$

if and only if there is a plan for $\Pi$.  

Jussi Rintanen
Interleaved P & E: practical approaches

People often use deterministic planning with autonomous robots.

- Closed World Assumption (CWA): Certain facts are known true, everything else is assumed false $\Rightarrow$ one initial state.

- Actions are considered deterministic: failures are ignored.

- Uncertainty about the behavior of the environment is ignored.

All nondeterminism and uncertainty is handled by the plan execution mechanism through replanning.
Interleaved P & E: practical approaches

Use of deterministic planning is OK when

1. One possible state / effect much more likely than others.
   ⇒ Usually the planner produces a reasonable action.

2. Actions are reversible.
   ⇒ Wrong actions do not do harm.

3. Efficiency is not important.
   ⇒ Spending time taking wrong actions does not do harm.
Architectures for autonomous robots

1. Sense, Plan, Act
2. the Subsumption architecture of Brooks
3. the 3-tier architecture of many people
Sense, Plan, Act

1. Take input from sensors (camera, sonar, laser-finder, microphone, ...).
2. Build a representation of the current world state from the sensor inputs.
3. Run a planner for deciding which action to take.
4. Take that action.
5. Continue from 1.

Do this e.g. 3 times or 10 times a second.
Problems with Sense, Plan, Act

- Building an accurate World Model is extremely difficult.
  - No robot/program can recognize a wide range of objects under all circumstances.
  - Geometric properties of the world...
  - Even in simple cases this may take a lot of time.

- Planning may take a lot of time.
  - Fast only with very few (< 50?) state variables and full observability.
  - With partial observability some dozens of states!!!
Problems with Sense, Plan, Act: consequences

- Environment changes fast: what sensor interpretation & planning is done may not correspond to the current state of the world.

  ⇒ Actions may be false because of the delay.
  ⇒ Reacting very fast is not possible.

  *Traditionally mobile robot projects have delayed handling moving objects in the environment beyond the scientific life of the project.* (Brooks, 1985)
Robotics without knowledge representation

Rodney Brooks says in 1985, 1986 that

1. Interpreting sensor input and representing knowledge in the robot is not necessary for many kinds of applications.
2. Many animals do not do it, and still do very well in the world.
3. This will lead to better robots.

(Brooks also started work on truly intelligent robots with the idea “intelligence without representation”, but so far nothing interesting has come out of that.)
Figure 1-2. Animals seem to use incomplete models for many activities. Baby seagulls respond just as well to the mockup on the right as they do to their own parent (left). The critical features are that the object must be pointed and must have a red spot.
Figure 1-3. The coastal snail may be controlled by a fixed hierarchy of behaviors. The combined effects of these behaviors enables the snail to navigate to its feeding area.
Brooks’ subsumption architecture

- There are several paths from sensors to effectors.
- Each path has a different task, can work in different speed and independently of the others.
- Same effectors may be controlled by several paths: paths with a higher priority may override those with lower priority.
- Little reliance on explicit representation of knowledge.
Figure 3. Control is layered with higher level layers subsuming the roles of lower level layers when they wish to take control. The system can be partitioned at any level, and the layers below form a complete operational control system.
Sensors → reason about behavior of objects → plan changes to the world → identify objects → monitor changes → build maps → explore → wander → avoid objects → Actuators
Brooks’ subsumption architecture

- Multiple goals: Fulfill the task, Avoid collisions, ...

- Multiple sensors: Sensors may be used independently of each other.

- Robustness: Behaviors are independent of each other.

- Additivity: New behavior can be added with minimally modified the robot.
Subsumption architecture: the Herbert robot

One of the first robots Brooks et al. built was called Herbert.

- Move around, visit rooms
- Look for coke cans, grab them, return them to “home”
- No centralized control for everything
- Very simple software
Figure 1-4. In our colony architecture there are a large number of independent control units which all operate in parallel. Each of these agents is responsible for some small part of the robot's task and compete to control the robot's actuators.
Figure 5 - Herbert's primary goal is to collect empty soda cans. It locates them with a laser light-stripper, grabs them with a long arm, then uses proximity sensors and a compass to bring them back.
Herbert: Behaviors of highest layers, navigation and movement

- Wander around, following walls.
- In junctions record the path taken.
- When holding a coke can, return home the same path.
Herbert: Behaviors of lowest layers

- Go forward following a wall; stop when wall ends.
- Stop if about to collide with wall etc.
- Go away if some object is about to collide.
Problems with the Subsumption architecture

• Too little reliance on memory / internal state.

• No centralized control: the robots do not work very predictably because different layers do not coordinate their activities sufficiently.

• Many applications would benefit from stricter coordination.

⇒ Borrow the best ideas, and put centralized control back.
Lessons from the Subsumption architecture

- Fast reactions to some sensor stimuli.
- Different layers with different competences.
- Brooks’ general idea of no centralized control whatsoever is bad!! Some centralized control is good; too much may be bad.
The 3 tier (3T) architecture

1. planning (no restriction on time consumption)

2. sequencing of tasks (coordination of skills for reaching goals)

3. skills (sensors, effectors, immediate reaction to sense data)

Immediate reactions in the skill level independent of internal state / memory.
3T: task / sequencing layer

- There is fixed set of high-level tasks, or tasks are produced by a planner one by one.

- Non-atomic tasks are reduced to simpler ones by methods

- Atomic tasks are achieved by skills.
3T: skills, direct access to sensors and effectors

Examples of skills:

- get pictures from camera
- measure distance to obstacles with sonars, IR sensors, laser
- move hand/manipulator
- immediate reaction to obstacles too close: stop or move robot
- tight cooperation of sensors and effectors: grab an object
3T: example

1. Planner: “enter room A”, “look for mail to deliver”, “pick up mail”, “go to corridor”, “walk N along the corridor”, “enter room B”, ...

2. sequencing: managing a map, moving the robot between map locations, managing objects, high-level commands to the manipulator, ...

3. sensors, effectors
Implementing the sequencer

- There are several programming languages for implementing the sequencer layer: RAP, PRS, REX/GAPPS, ESL, ...

- These manage and execute high-level tasks generated by the planner.

- Tasks are reduced to simpler ones; reduction context-dependent

- Concurrent execution of tasks
Reactive Action Packages (Firby 1989)

- Working memory (≈ observable state variables)
- Task agenda
- Used in many robotics projects in Europe and in the US.
- Implemented in Common Lisp
RAP: task indices (names)

(INDEX (handle-low-fuel))
(INDEX (arm-pickup ?arm ?thing))
(INDEX (find-object ?object => ?place))
RAP: methods

(METHOD
  (CONTEXT formula)
  (TASK-NET tasknetworkdescription))

(METHOD
  (CONTEXT formula)
  (PRIMITIVE skillrequest))
RAP: a task net

(TASK-NET
  (step1-tag priority step-specification annotation
    ...
    (step2-tag priority step-specification annotation
      ...
      )))

Annotations defined ordering constraints (formula FOR tag)

(TASK-NET
  (t1 (arm-grasp arml ?thing)
    ((arm-holding arml ?thing) FOR t2))
  (t2 (arm-throw arml ?thing)))
RAP: a RAP definition

(DEFINE-RAP
  (INDEX (move-to ?thing ?place))
  (SUCCEED (location ?thing ?place))
  (METHOD
    (CONTEXT (and (location ?thing ?loc)
      (not (= ?loc unknown)))))
  (TASK-NET
    (t0 (goto ?loc)
      ((truck-location ?loc) FOR t1)))
  (METHOD
(CONTEXT (and (location ?thing ?loc)
    (not (= ?loc unknown))))

(TASK-NET
  (t0 (goto ?loc)
    ((truck-location ?loc) FOR t1))
  (t1 (pickup ?thing)
    ((truck-holding ?thing) FOR t2)
    ((truck-holding ?thing) FOR t2))
  (t2 (goto ?place)
    ((truck-location ?place) FOR t3))
  (t3 (putdown ?thing))))
RAP: a RAP definition, cont’d

(METHOD
  (CONTEXT (and (location ?thing unknown)
                 (not (truck-location warehouse)))))

(TASK-NET
  (t0 (goto warehouse)
       ((truck-location warehouse) FOR t1))
  (t1 (pickup ?thing)
       ((truck-holding ?thing) FOR t2)
       ((truck-holding ?thing) FOR t3))
  (t2 (goto ?place)
       ((truck-location ?place) FOR t3))
  (t3 (putdown ?thing))))
RAP: instantiated task net

t0: (goto lot-45)
t1: (pickup box-1)
t2: (goto factory-5)
t3: (putdown box-1)
RAP: the RAP interpreter

For a given task in the agenda:

1. If the succeed clause is satisfied, finish.

2. Choose a method.

3. Execute the method.

4. Go to 1.

Different methods are tried until the succeed clause is satisfied.