COSC 4P78

Control Architectures — Hierarchical/Deliberative

Week 5

(Murphy and Matarić)

Brock University

Control Architectures

Recall from last week, when we discussed Control Loops.

- Pretty cool stuff, right?
- We can create a mapping from a sensor to an actuator, simulating input if necessary, and achieve some reasonably interesting behaviours

But we never really solved the problem of how to combine multiple — potentially mutually exclusive — behaviours. How can we resolve conflicting commands?

- More basically, we never even really addressed how to express more complicated problems.

Control Architectures

What's that?

As we discussed last time, a *Control Architecture* is a set of guidelines, principles, or mappings used to coordinate the behaviours of a robot.

Among other things, it's responsible for:

- Decision making
- Planning
- Responses to stimuli

Basically, the 'brain' of the robot; it's how you model a behavioural solution.

Robot Primitives

SPA

Before we get any further, we should first briefly discuss the three basic robot primitives:

- Sense
- Plan
- Act

From a high-level perspective, pretty much everything a robot does will fall under one of these categories, and our approach will be connected to how we tie them in together.

Robot Primitives

Sense

Huge surprise: *Sense* describes how the robot senses its environment.

However, it can easily be a bit more complicated than that.

- If your robot is maintaining a *world model*, then sensing will be used to update that totality of its environment
- This could include things like incorporating conclusions and knowledge inferred from the sensors as being part of the inputs
- This can also include adding *state*, or other virtual information into the world model

Robot Primitives

Plan

For many tasks, a robot will need to step through some pretty complicated plans, at several layers of detail, and with several different levels of granularity (both in detail, and in time frames).

Of course, planning for some task also often includes *searching* within a knowledge base or other virtual representation

- Basically, some robots could require quite a bit of thinky power

Robot Primitives

Act

Obviously, in order for a robot to be able to accomplish anything, it will need to eventually include actuation.

- This could mean direct motor control
- This could mean higher-level instructions
- This could even include commands to update internal state information

Before We Continue

One last preliminary point

Because we'll be dealing with both sensing and planning, we're really going to need to start addressing representation better. We'll talk more about maps and such when we discuss localization and mapping later, but it's still worth considering:

- How can we represent a set of rooms, connected by hallways?
- How do we represent our own state? (e.g. proprioception, movement, position)
- Are obstacles separate entities, or oddly-shaped walls?
- Should the robot's current (or past) actions influence its world-view?
- How can we actually represent a task/goal?
- How can we represent the actions available to us?

Before We Continue

Additional considerations

- For several of these, greater precision (or finer granularity) will afford more complex behaviours, but may also trigger a combinatorial explosion
- Conversely, simplification may be limiting, but also avoid intractable problems

Oftentimes, we might end up having to address both the fine-grained and the coarse, but end up doing so on different levels or at different stages/times.

Before We Continue

On representation and sensing

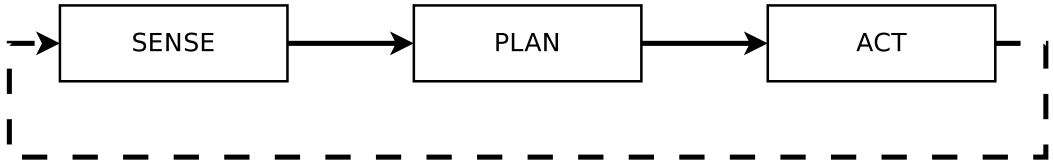
It's also important to note that we may need to start incorporating an ongoing *world model*

- That model may be entirely defined in advance (*a priori*)
- Our whole world may be whatever we can currently sense
- We could maintain a cognitive model, that uses both sensors and the rules we know about the environment to infer the world's current state at a given time

Of course, this will require some careful planning, since we'll at least need a reasonably easy way to make queries, and might also need to be able to make updates.

The Deliberative Model

So, how do we incorporate Sense, Plan, and Act into the Deliberative model?



- Yup. That's it
- Seems a bit underwhelming at first, don't it?

Sense-Plan-Act

(SPA)

It actually makes sense:

- Before we waste time on calculations or actions, we first ensure that we have up-to-date information about our environment; updating as necessary
- We then work out exactly how we want to solve the problem, based on our current situation
- After acting for some length of time, you return to the beginning, to get a new update on the world

And clearly that means it's the best approach, and we're going to be learning alternatives later just to show why people *didn't* abandon this paradigm a couple decades ago...

Strips

An oldie, but a goodie

Let's start by taking a glance at a practically ancient system: Strips

- Stanford Research Institute Problem Solver
- A high-level planner that described a problem in terms of an initial state, goal state(s), and available actions
  - Actions are comprised of:
    - ★ Actual steps/actions to perform
    - ★ Preconditions that must be satisfied in order to trigger the action
    - ★ Postconditions that update the state of the environment
- Basically, you start at the initial state, work out the difference relative to the goal state, and then (depending on the available actions, and their pre- and postconditions) step through the actions trying to minimize that difference and arrive at the goal

Strips

A very simplified example

Suppose you want to get to Stanford from Tampa:

```
Initial State: Tampa, Florida (0,0)
Goal State: Stanford, California (1000, 2828)
---
Difference: 3000
```

- What we need here is a little *Means-ends analysis*
- We'll need a listing of available operators for different distances

Strips

Difference tables

Suppose we were to try using the following *distance table*

difference	operator
$d \geq 200$	fly
$100 < d < 200$	ride_train
$d \leq 100$	drive
$d < 1$	walk

Would this be effective?

Strips

Let's refine it a bit

difference	operator	pre-conditions	add-list	delete-list
$d \geq 200$	fly		at Y at airport	at X
$100 < d < 200$	ride_train		at Y at station	at X
$d \leq 100$	drive_rental	at_airport		
	drive_personal	at_home		
$d < 1$	walk			

Note that, if we wanted to, we could apply this basic approach to just about any pathfinding application.

- (However, also note that, at this level of discourse, we haven't even addressed how we'd do something like "walk")

Strips

Let's try something a bit more applicable...

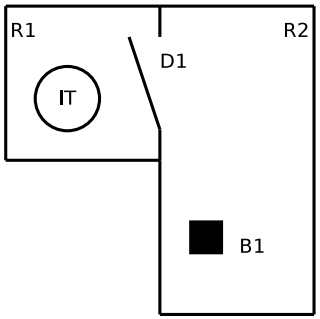
Suppose a robot's knowledge could be described according to the following predicates:

```
INROOM(x, r)   where x is an object of type movable_object,
               r is type room
NEXTTO(x, t)   where x is movable_object,
               t is type door or movable_object
STATUS(d,s)    where d is type door,
               s is enumerated type: OPEN or CLOSED
CONNECTS(d, rx, ry)
               where d is type door,
               rx, ry are the room
```

Strips

Initial State

Then:



yields:

```
initial state:
INROOM(IT, R1)
INROOM(B1, R2)
CONNECTS(D1, R1, R2)
CONNECTS(D1, R2, R1)
STATUS(D1, OPEN)
```

Strips

Goal State

Then, our goal state could be:

```
goal state:
INROOM(IT, R2)
INROOM(B1, R2)
CONNECTS(D1, R1, R2)
CONNECTS(D1, R2, R1)
STATUS(D1, OPEN)
```

Of course, this would require a much simpler distance evaluator than our previous example. Instead of Euclidean distance, we're just using predicate logic. If we're not in the desired room, then ¬`INROOM(IT,R2)`.

Strips

Operators

So, what operations could we have?

operator	preconditions	add-list	delete-list
OP1: GOTODOOR(IT,dx)	INROOM(IT,rk) CONNECT(dx,rk,rm)	NEXTTO(IT,dx)	
OP2: GOTHRUDOOR(IT,dx)	CONNECT(dx,rk,rm) NEXTTO(IT,dx) STATUS(dx,OPEN) INROOM(it,RX)	INROOM(IT,rm)	INROOM(IT,rk)

Strips

What's the problem?

That actually seemed like a pretty good way to describe the problem, goal state, *and* actions!
So, what's the problem?

Strips

Well...

- It's actually difficult to design
  - World model representation
  - Difference table and operators, preconditions, and add/delete lists
  - Difference evaluator
  - That might have been great for describing how to get from Tampa to Stanford, but how would you use it to explain how to get from where I currently am to the door?
- Closed World Assumption
  - Surprises are bad, m'kay?
  - The robot must be able to assume that it knows *everything* in the world, and that it's entirely static outside of what the robot can directly know
- Frame Problem
  - Okay then, I want my robot to get from here to J327; either via an indoor route, an outdoor route, or a combination thereof
  - Is that very much information to know?

What do we actually want from an architecture?

We'll at least want to evaluate an architecture according to:

- Modularity
  - Good software engineering principles that ensure flexibility and interoperability
- Niche targetability
  - How well does it work for the intended application? Is this a square peg/round hole problem?
- Portability to other domains
  - Is this planner limited to only the testing environment used for its design? For that matter, could it be applied to a different robot entirely?
    - ★ Logically speaking, should the high-level planning of a wheeled robot be portable to a legged robot?
- Robustness
  - Where is the system vulnerable? How does it try to reduce that vulnerability?

Okay. That's it. I have to ask...

Deliberative is an easy term to understand (deliberates before acting), but why is it also called *Hierarchical*?

We've mostly only been discussing the planning aspect; not how to actually integrate it into a real robot.

- There are typically multiple levels of abstraction and encapsulation; not just the separation of sense/plan/act:
  - Sensing actually requires a world model, possibly a knowledge base, and sensors to update them both
  - Planning is typically a layered system of *mission planner*, *navigator*, and *pilot* (if not more as well)
  - Actuation requires a low-level controller, driving, and steering (at the minimum)

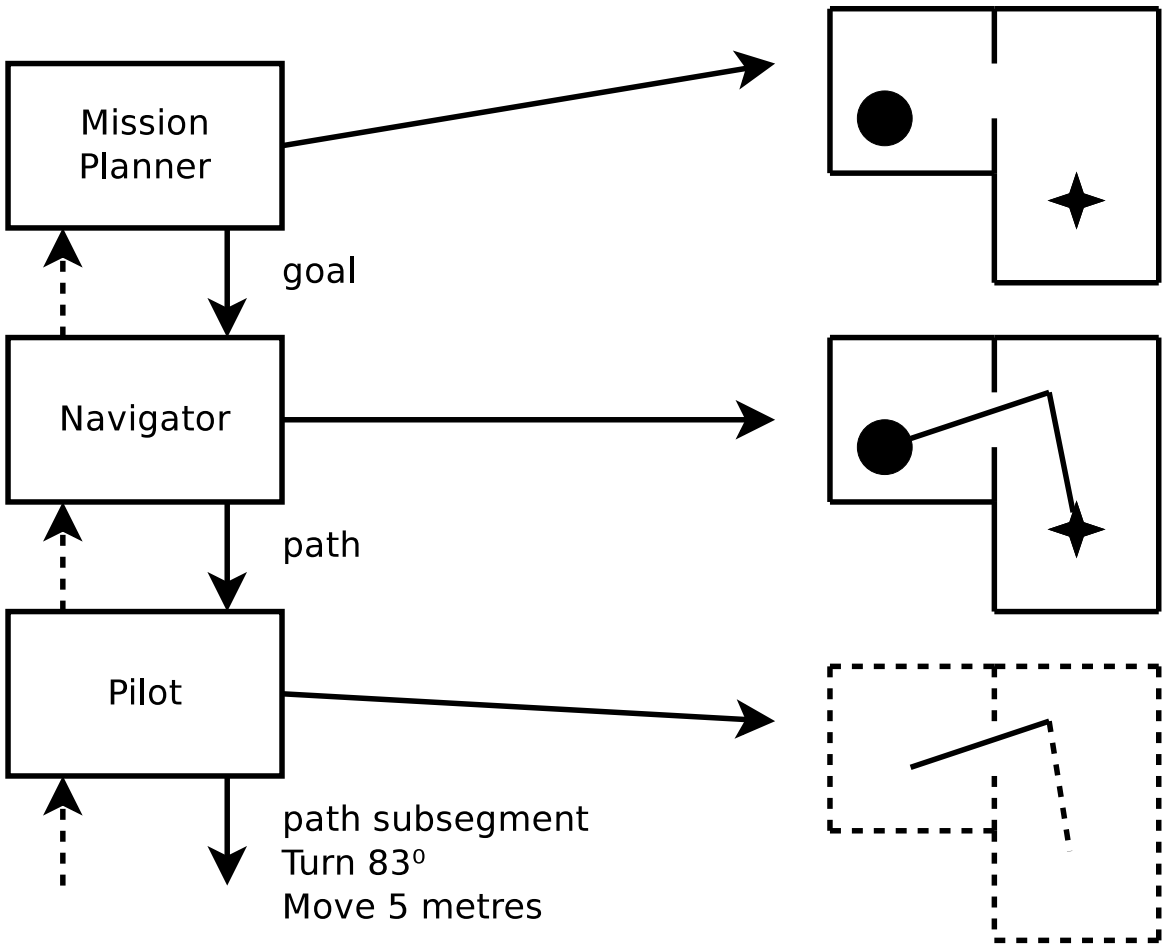
And...?

And... that's the point

- The planning module draws on the consolidated world model provided by *Sense*
- The planning module then issues instructions to the *Act* component
- The planning module itself is arranged into a hierarchical fashion, wherein the:
  - Mission Planner can work out high-level pathfinding like the next target location
  - The navigator can work out plotting the next trajectory
  - The pilot can ensure that the robot is actually following the desired lines
- It's also worth noting that for the different tasks/goals, there are typically also different time frames
  - For example, the mission planner might only stop the robot to update and re-plan every few minutes (or only when an obstacle/surprise is encountered)
  - The navigator could be invoked once the next waypoint is reached (e.g. seconds apart)
  - The pilot is probably running continuously (or darn close)

One Possible Planner

This one's from NHC, but we don't want to get too bogged down on specific planners (RCA is another one, if you're feeling bored and want some 'light' reading).



So then, is Deliberative good?

Well...

It was fantastic in its time, but it still had lots of issues:

- Time-Scale
  - Before taking any action, we need to re-evaluate the entire problem and precompute everything
  - This is ludicrous for a dynamic environment
  - Even for a *mostly* static one, it's still pretty inefficient
- Space
  - The frame problem we mentioned earlier is a very real issue
  - If we're relying on (near) total knowledge, then we're storing a lot of information
  - Keep in mind that we don't even necessarily need to *use* all of that information
- Information
  - Speaking of information, we need to collect *a lot* of it!
  - Between the collection of it, and all of the extra processing over it (including a possible combinatorial explosion from the search space), it's just a big hassle all around

Final thoughts on Deliberative
(cont)

- Use of plans
  - Plans are awesome. They're the whole point in using a planner
  - But the actions become too highly coupled with the generated plans
  - If there's even the smallest change in the environment, or the most minimal of surprises, then you need to recompute the entire plan
    - ★ Even if you've devised a way to separate and reuse some of that work, it's still a lot
    - ★ Depending on the design, even things like minor effector inaccuracies could throw off a plan
  - In other words, what works for a simulated, or highly-constrained environment, might not actually be very practical

Okay...

So then, what's an alternative?

- Is there something that can just immediately response to stimulus?
- Can we get a robot to act, without first needing to plot out its entire future?

Sure. All we need to do is get it to operate on *instinct*

- (uh. sort of)

Interjection!

How do animals ever 'get anything done'?

- This actually takes us back to where we started, when we first began learning about AI
- What is the actual phenomenon we wish to represent?
- How can we express that behaviour via a connection between input and output?
- How can we implement that model to actually realize the behaviour?

How do we see what we see?

Suppose we look at a square.

- How do I know I'm looking at a square?
- (We're not even going to get into the magic that is a chair, or a table)

Let's just briefly discuss what's going on with our own vision for a bit (we'll get into actual vision much later on).

What is a *Behaviour*?

How can we define one?

Let's formally define a behaviour as a mapping from sensory input to actuator command/output.

- This could be on a high level, or a low level

Actually...

Let's stop there. We're encroaching on next week's topic

- In the meantime, questions?