COSC 4P78

Controllers and Control Architectures

Week 4

(Murphy and Matarić))

Brock University

Control Loops

A sample problem

Suppose a room is 62°, and you want it to be 70°.

- We'll assume you have a digitally-controllable source of heat
- For this situation, what is the obvious action to take?
  - And then... ?

Control Loops

Most likely, at some point in time, we'll probably want to take another reading, and then make another decision.

- Maybe the room has heated up enough?
- Maybe it still has a ways to go?
- Maybe we've overshot it?

Clearly, we'll need to introduce some notion of *repetition*, so we know what that means: a loop.

Control Loops

Other comparable problems

Before we continue, this is probably obvious, but this is the type of problem that will keep popping up, over and over, and not just for things like temperature. For example:

- Maintaining distance from a wall
- Balancing a platform
- Closing the gap between a robot and its destination

Basically, we're looking at any situation where we can sense our current state, know our preferred state, and make a decision with the goal of getting us closer to that preferred state.

Control Loops

Open or closed?

There are two basic classes of control loops:

- **Open Loop**
  - Also known as *ballistic*, or *feedforward*
  - "Set it, and forget it!"
  - Based on input, model of the environment, or other controller decisions, instructions are given to an actuator
    - ★ There's no *direct* correction mechanism if the actual result doesn't match the expected result or output
  - Can we think of some examples?
- **Closed Loop**
  - Also known as *feedback control*
  - Reevaluates sensor readings or updated state information to provide an amended actuator instruction
    - ★ This one tends to be more interesting, so let's discuss it some more

Control Loops

(Interjection)

It's worth noting that either approach is typically used on a very low-level. i.e. we'll eventually be looking at more complicated control architectures that can take advantage of these controllers.

States

At any given time, the robot, system, or some combination thereof may be described in terms of some *state*.

- e.g. current position, current distance from the wall, current temperature, etc.

Besides the current state, we also care about the *desired* or *goal* state.

- So, there's how things currently are, and how we eventually want them to be

States

Types of Goal States

There are different ways of describing goal states. Two common ones are:

- *Achievement Goals* — a final state to reach
  - e.g. a destination, the end of a maze, getting a ball into a goal
  - Once you achieve that state, you're done
- *Maintenance Goals* — a continuous requirement, that must be maintained even after being achieved
  - e.g. keeping a robot the same distance from a wall, keeping the temperature comfortable, keeping a segway balanced

Irrespective of which type of goal state we're using, we're still likely to rely on the concept of *error*.

I AM ERROR

Note that we don't mean *error* in the "you made a mistake" sense, but rather in the more traditional AI sense — the difference between the current state and the desired state.

- If a feedback control loop can steadily reduce the error of a system, then that means it can eventually achieve (or maintain) the desired goal

Error

How do we express error?

There are different ways to qualify or quantify error:

- We *can* use zero/non-zero error
  - i.e. simply saying, *right* or *wrong*
  - Outside of some forms of reinforcement learning, or unless used in conjunction with additional controllers, not typically terribly useful
- Error magnitude
  - "How wrong is the current reading?" or, "how far do we have left to go?"
  - Not usually hugely helpful, unless it includes the next point:
- Error direction
  - e.g. *too hot!*, *to the left*, etc.
  - Particularly powerful when combined with error magnitude

We'll also care about how frequently we can receive these error values.

Error

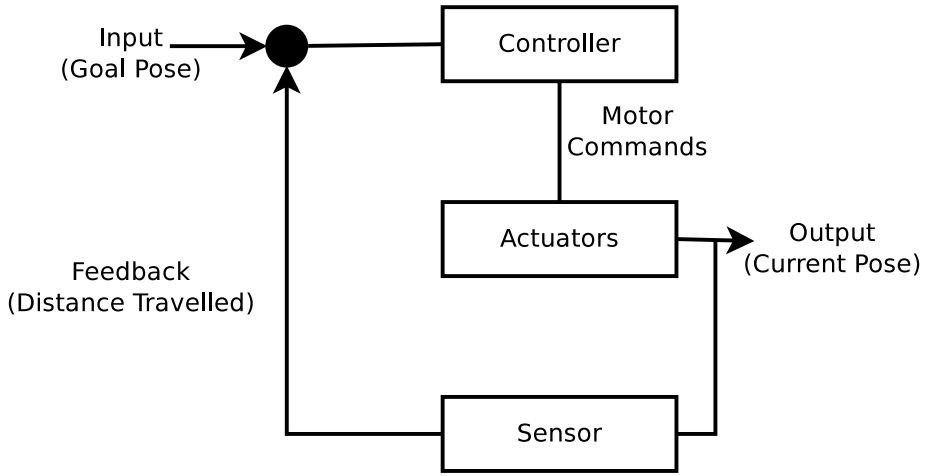
One last comment

Just to tie things together before proceeding...

- If the magnitude of the error is zero, what does that likely tell us?
- Goal states, and thus also error, can be considered either external to the robot, or internal to it
  - Silly example: a robot that sprays coolant onto another device that might overheat
  - Internal example: a robot that realizes its own battery strength, and can control its own behaviour or make adjustments like shifting solar panels towards the sun

Feedback (closed-loop) Control

Consider the following:



Sample Problem

A wall-following robot

Let's consider a wall-following robot.

- What sensors would we need? What information would they give the controller?
  - A bump sensor wouldn't give very much, right? Direction, yes, but magnitude? Nope
  - Infrared? Laser? Ultrasonic? Stereoscopic machine vision?
  - Whatever sensors we pick would also dictate the extent of our *feedback*

What sort of behaviour could we describe to achieve this task?

Sample Problem

An informal behaviour

```
If distance-to-wall is in the right range,
    then keep going.
If distance-to-wall is larger than desired,
    then turn toward the wall,
    else turn away from the wall.
```

Would this work?

Sample Problem

An informal behaviour (cont)

Probably not. Maybe we can make it a little better:

```
If distance-to-wall is in the right range,
    then keep going.
If distance-to-wall is larger than desired,
    then turn by 45 degrees toward the wall,
    else turn by 45 degrees away from the wall.
```

Any better?

Sample Problem

What *is* the problem?

Still not much better.
What actual behaviour would we expect from such a robot? Can we draw it on the board?

Oscillation

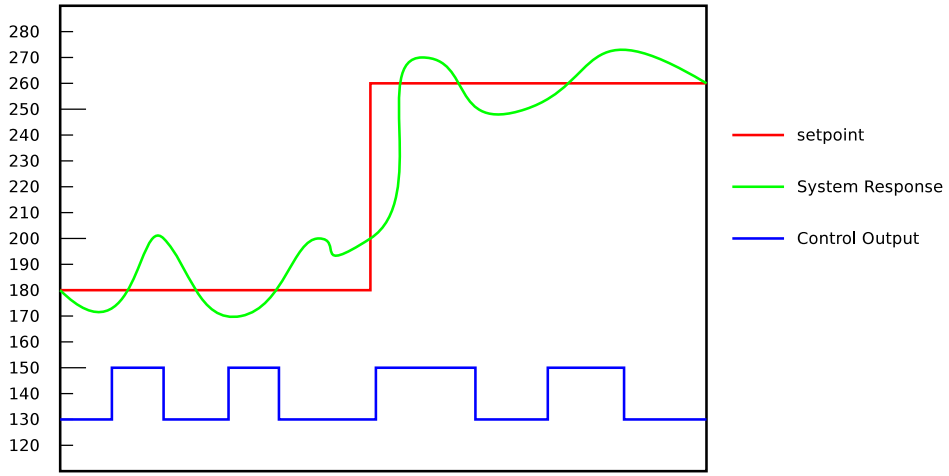
For most problems, if we only consider, "oops. This is wrong. Better fix it!", without giving any additional thought to things like magnitude, momentum, delays or inaccuracies from sampling frequency, etc., then we're pretty much guaranteed to always stumble back and forth. Consider something simple, like controlling an oven (say, for annealing, or soldering surface-mount components):

Let's look at a totally real example of legitimate non-made-up data.

Oscillation

Turning a heating element on and off

Consider the following:



How closely does the actual temperature match the desired setpoints?

Tweaking control parameters, or increasing the *sampling rate* can only go so far...

What we'd very much like to do is to introduce some notion of *damping*.

Feedback Control

So, we want to make use of our sensors, to actively update the instructions sent to our actuator. But, how do we map that sensory input to a new output?

- Chances are, we aren't going to find a simple, single solution that will address all of our needs
- We're likely to have to come up with a multifaceted approach
  - Our approach for now will be to try to devise a mathematical mapping, that incorporates as much information as we can squeeze out of both our sensors and our knowledge of the problem

It's worth noting that, even though everything we've looked at thus far has been *stateless*, and mostly still should be, we may end up cheating just a little bit.

Control Theory

How big is the error?

Suppose you see that a large picture frame is crooked on the wall. How would you respond to each of the following cases?

1. The frame is only off by a smidgen
2. The frame is tilted so far, it almost looks like a diamond

Would you correct both cases to the same extent?

Control Theory

Proportional Control

Proportional Control allows a control system to respond with a magnitude that is (surprise surprise) proportional to the magnitude of the error.

- Tiny error? Tiny adjustment!
- Huge error? Huge adjustment!

If the output is o, and the input is i, then the we could say:

$$o = K_p i$$

where Kp is some proportionality constant.

Control Theory

Proportional Control (cont)

What we're looking for are appropriate *gains*.

- Basically, the magnitude of the response
- A correctly-scaled proportional control system will give good proportional gains
- Our selection of K_p will have a strong impact here

Recall the oscillations we saw a little while ago. Those are still a concern.

Control Theory

So, what's the problem?

Actually, there are lots of (potential) problems.

- Maybe the sensor isn't perfect?
- Maybe the actuator doesn't always behave predictably (*actuator uncertainty*)

Heck, what could we easily see happening, even in our frame example?

- "Well, this is off by a very large margin, so I'm going to hit it as hard as I can. Should work. probably."

The problem is *overshoot*.
The worst-case scenario would be that instead of *damping* the oscillations, or even just retaining them, we add so much to our gains that we end up with *divergent oscillations*.

Control Theory

(Before we continue...)

One quick addendum:

- Choosing the right parameter K_p is going to be tricky, and that'll be true of the other math we see next as well. There are basically two ways to figure these things out:
  - Perfectly model the entire environment, to mathematically derive the values you need
  - Empirical testing (trial and error)

Sometimes, you use a bit of both.

Control Theory

Rate of change in error

What's missing is some acknowledgement that we might be 'building up some steam'.

Consider the example of trying to move a hovercraft towards a target (or a body in space, or underwater, etc.).

- Just because you haven't reached your target yet, does that mean you should still be applying throttle forwards?

Maybe we need to acknowledge the rate at which the error is changing...

Control Theory

Derivative Control

Derivative control relies on the fact that you could already be making progress towards reducing the error, even without any additional output. (We know what momentum is, right? Yes? Good)

$$o = K_d \frac{di}{dt}$$

In other words, a derivative controller would make its decision based on how the error was already changing.

Obviously, this wouldn't be useful, by itself. But before we get to the conclusion, let's introduce one more concept...

Control Theory

Accumulated error

No matter how perfectly your proportional and derivative controllers could work things out in theory, there'd still be some remaining error.

- Pick a reason. Any reason. (Sensors, actuators, rounding errors, alignment of the stars, etc.)

What we need to do is to address the *accumulated error*.

Control Theory

Integral Control

Suppose you had a term as follows:

$$o = K_f \int i(t)dt$$

Basically, what if we could track the error that's always present?
In practice, we probably won't do this in the pure mathematical sense.
Instead, what's often done is to retain the error across multiple samples, sum them, and consider some small portion of them either continuously or after they start adding up to a significant extent.

- For something like wall-following, it might not be that interesting/useful
- For some task that relies on knowing where you are (or are going) based on tracking the changes in your angle, you might need to occasionally tip it back towards the correct configuration just a bit

Control Theory

PD and PID Control

So then, how does this help us?

Well, again, none of those components individually would create a complete (useful) controller, so we instead combine them:

$$o = K_p i + K_f \int i(t)dt + K_d \frac{di}{dt}$$

This is what we call a *PID Controller*.

- K_p, K_f, and K_d need to be very carefully chosen, to be both appropriate for the robot/environment, but also to synergize well with each other
- Omit the $K_f \int i(t)dt$ term for a *PD Controller*

Control Theory

(Final comments)

Besides everything else, it's worth noting that which components you'd actually use, and to what extent, would be *highly* problem-specific.

In particular, because the integral component introduces a concept of state, and because it will sometimes only affect the smallest differences between two asymptotes, it really isn't unheard of to simply rely on a simpler PD controller.

Also, related to the sampling rate is the possibility of holding patterns, according to state.

Control Architectures

So, we've designed a controller! Yay! Ready to build a robot now?

- I suppose that depends on what you want your robot to do
- Consider the wall-following robot. What *else* was it doing?
  - ... going forward?
    - ... forever?
    - ... towards wherever?
- So then, let's add another goal: go to the target painted on the floor, partway down the wall
  - No problem! Just make a controller that knows whether or not it's on that target!
  - And, as for the wall-following...
    - Oh. Crud.

Control Architectures

Basically, sooner or later, we're going to need to model some more complicated behaviours. Maybe *this* example didn't really need it, but we'll get there eventually.

- As previously stated, these control mechanisms are great for solving a low-level task
- Sooner or later, we'll need to address how to solve high-level tasks
  - Particularly when those high-level tasks rely on multiple low-level tasks

A *control architecture* is exactly what it sounds like: something that allows you to organize multiple control systems, into a single design (intelligent agent).

- These will allow us to design far more robust robots, and tackle far more complicated tasks
- Particularly when we need to address either conflicting goals/errors, or when we need to start dealing with *planning*

Food for Thought

Representation

Nothing to talk about yet. Just want us to start thinking about it.

Questions?

Comments?

- Does anyone actually bother specifically using jelly for PB&J sandwiches?