Designing for Failure

MIT 6.270
January 2012

Why design for failure?

- Murphy's law
- If something fails, it no longer matters how “unlikely” or “improbable” it was – the only thing that matters is whether or not your robot can recover
- With double-elimination final competition, a single failure is extremely costly

Better yet, design to **avoid** failure

- Before planning for failure, take steps to avoid it in the first place:
  - Keep code clean and organized – it's hard to spot bugs if you don't understand the code
  - Avoid “magic numbers” in code
  - Make code self-describing (no “foo()” functions or “x” variables) – writing comments is no excuse for ugly or unclear code
  - (See McConnell's Code Complete or Robert C Martin's Clean Code)
  - Look for structural weak points
  - Make sure all solder joints are sturdy
  - Keep things simple, both mechanically and in software

Figure out what fails - testing!

- After you've attempted to avoid failure, test **extensively** to find out what fails
- Make sure to try many cases:
  - “normal” case – standard configuration
  - Edge cases, for example:
    - Start robot nears walls, gearbox
    - Leave some balls on the field near your robot
- Test frequently during development – don't wait until robot is “finished” - cost to fix issues increases exponentially as time goes on
- Write down everything that fails and steps to reproduce
- Run regression tests – after fixing one error, make sure old errors aren't reintroduced

Example: why testing is important

- Features that were added because of failures during testing:
  - Double chain
  - Redundant IR LED Phototransistor pair
  - Lift switch
  - Wire/HappyBoard covers

Handling Failure Well

- We all fail sometimes – it's ok, but do something about it!
- Write code to check for and handle exceptional cases

Tip #1: Add timeouts

- Don't continue action forever if you aren't making progress
- Robots trying to drive through a wall for 2 minutes makes for a boring competition!
- Easy but effective timeout: If you tell robot to go somewhere, but it doesn't get there fast enough, back up and try again

Tip #2: Escalate Response

- "Insanity is doing the same thing over and over again and expecting different results"
- If you timeout/fail more than once, try something different
- Maybe even use randomness in response

Tip #3: Use extra sensors to check for failures

- Add switches to detect wall collisions
- Check motor current for stalls
- Modify servo to get position feedback

Tip #4: Use redundancy

- Make sure critical and error-prone parts are redundant
  - e.g. chains, certain sensors

Tip #5: Reorient after failures

- Collisions often occur because of inaccurate location info – should reorient before continuing
- Can use vision system to recalibrate position/heading
- Can also drive into a wall at full force if your robot has a flat front (watch out for balls though)

Tip #6: Test actuators and sensors on startup

- Write a test mode that will extend actuators and check for sensor inputs
- Run tests during setup period before each round

Tip #7: Use Checklists!

- Use a checklist to look over your robot for issues before every round
- Checklists work: pilots use checklists to avoid forgetting crucial steps during takeoffs and landings
- 2009 World Health Organization study: basic checklist for doctors and nurses reduced number of deaths from surgery by more than 40%