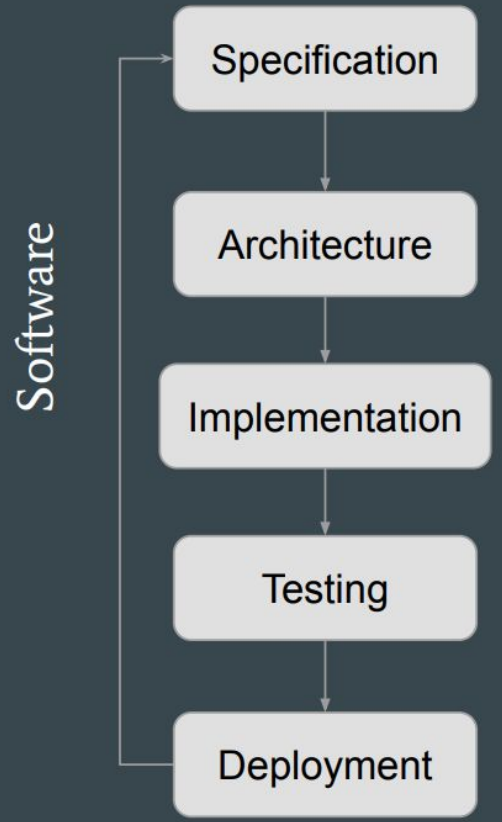
CS4501
Robotics for Soft Eng

•••

Specifications and Safety

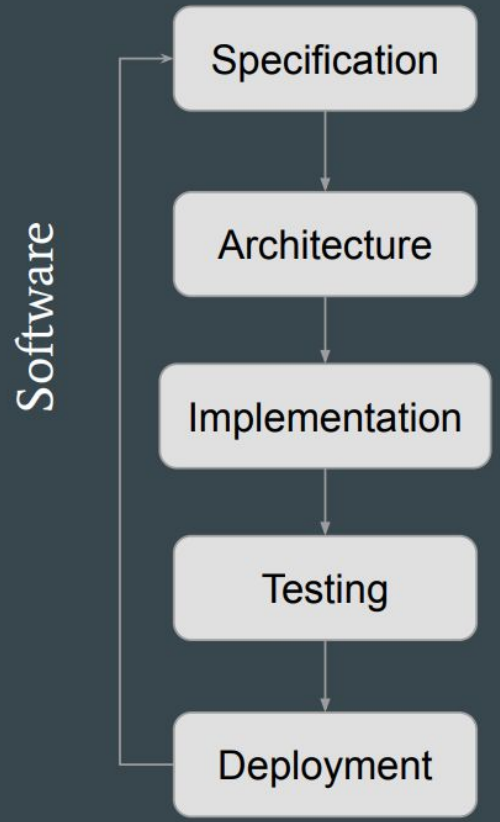
Development Lifecycle

- *What* do we specify?
- *How* do we know it is correct?

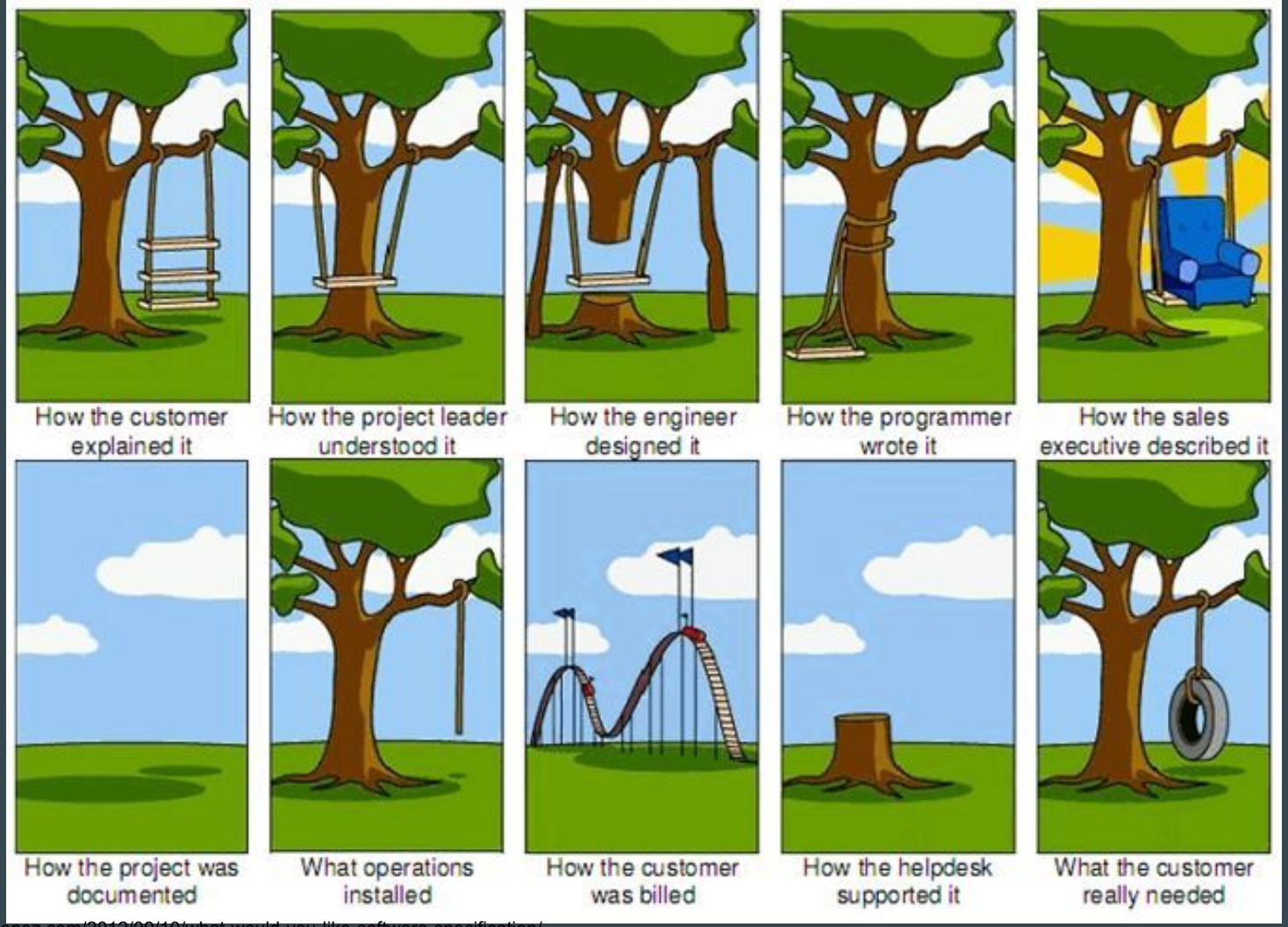


Development Lifecycle

- *What* do we specify?
- *How* do we know it is correct?
- *How* do we know that it is safe?
  - System Safety aims to achieve:
    - Acceptable risk from a systems perspective
    - Within constraints of time, cost, and system capability
    - Treating the system holistically, accounting for interactions among its constituent parts, *including parts outside the system*



https://sma.nasa.gov/sma-disciplines/system-safety



http://www.ricardoruizlopez.com/2012/09/10/what-would-you-like-software-specification/

Requirements vs Specifications

Requirement:

- Broad description of what needs to be accomplished - User Stories
- Not directly testable

Specification:

- Formal, detailed description of how it will be done -
- Testable, checkable

Requirements vs Specifications: Path-finding robot

Requirements vs Specifications: Path-finding robot

- System level requirements
  - It must be able to traverse an indoor environment
  - It must be able to travel between its starting position and a given goal position

- System level specifications
  - It must be able to drive at least 1 m/s on a floor of linoleum at a grade of at most 10 degrees
  - If a path exists such that the robot can safely* navigate to the goal, it must be able to find it

Specification Goals

- Explain *what* to do, but now *how* to do it

Overall specifications should be:

- Complete
- Consistent
- Precise
- Concise

Ernst, *Specifications*, 2005

Example Requirements

- What are the requirements for an automated robot vacuum cleaner?

Robot Vacuum Requirements

- Robot should be able to clean common indoor floor types
- Robot should perform its cleaning duties on a regular schedule

Robot Vacuum *Safety* Requirements

Robot Vacuum *Safety* Requirements

- Always return to charging station before running out of charge
- Never stall running the debris intake
- Can be disabled instantly from remote control

Example Specifications

- What are the specifications for an automated robot vacuum cleaner?

System vs Component Specifications

- Robot Vacuum System
  - Cleans a room of up to 100 sq ft. in less than 10 minutes

- Vacuum component
  - Maintain constant suction between 10-15 cubic feet per minute while vacuuming
- Sensing component
  - Give distance measurements to surrounding obstacles within +/- 2 cm at 60Hz
- Localization and planning subcomponent
  - When placed in an area, will generate a plan that will explore all reachable areas

System vs Component Specifications

- Robot Vacuum System
  - Cleans a room of up to 100 sq ft. in less than 10 minutes

- Vacuum component
  - Maintain constant suction between 10-15 cubic feet per minute while vacuuming
  - If there is a stall in the vacuum motor, it shuts off
- Sensing component
  - Give distance measurements to surrounding obstacles within +/- 2 cm at 60Hz
- Localization and planning subcomponent
  - When placed in an area, will generate a plan that will explore all reachable areas

System vs Component Specifications

- Robot Vacuum System
  - Cleans a room of up to 100 sq ft. in less than 10 minutes

- Vacuum component
  - Maintain constant suction between 10-15 cubic feet per minute while vacuuming
  - If there is a stall in the vacuum motor, it shuts off
- Sensing component
  - Give distance measurements to surrounding obstacles within +/- 2 cm at 60Hz
- Localization and planning subcomponent
  - When placed in an area, will generate a plan that will explore all reachable areas
  - Are there specifications about how Sensing and Localization interact?

Why do we need good specifications?

Mars Polar Lander (1999)

- $165 million robot
- Sent to study soil @ Martian south pole
- Crash landed after the software disengaged the engine too early

Artists Depiction, NASA/JPL

Why do we need good specifications?

Mars Polar Lander (1999)

- “A magnetic sensor is provided in each of the three landing legs to sense touchdown when the lander contacts the surface, initiating the shutdown of the descent engines”

- “The software—intended to ignore touchdown indications prior to the enabling of the touchdown sensing logic—was not properly implemented, and the spurious touchdown indication was retained. ”

Artists Depiction, NASA/JPL

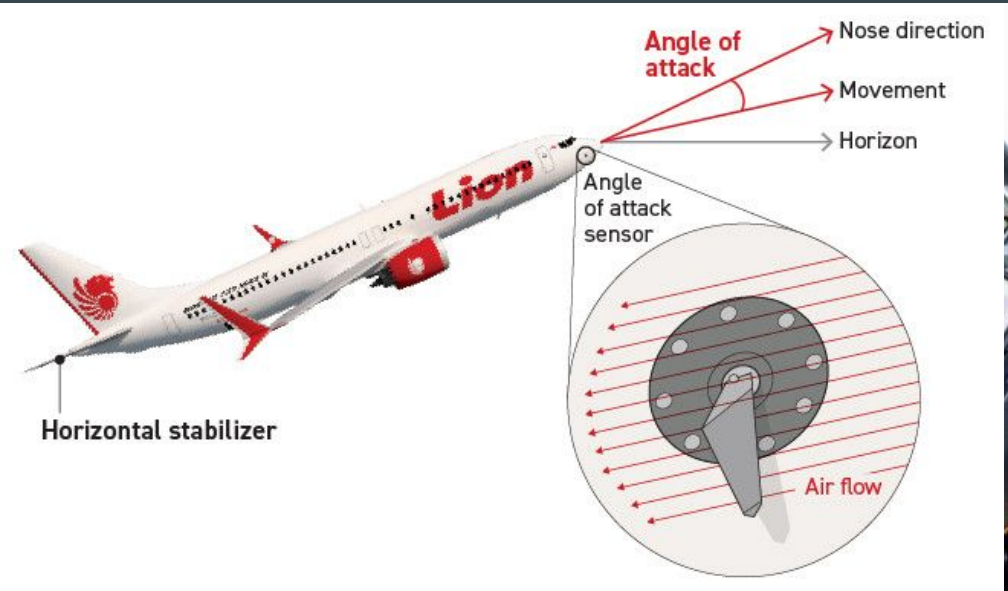
Paying attention to specs...

- Therac-25 - 1982-1987
  - Software race conditions caused massive overdoses in radiation. 3 injuries, 3 fatalities
- Space Shuttle Challenger - 1986
  - O-Rings known to fail at low temp, launched outside of range. Exploded 73s into flight, 7 fatalities
- Ariane 5 - 1996
  - Re-used software from Ariane 4, specs not updated, crashed
- Mars Climate Orbiter - 1999
  - One component used metric units, another used imperial units - led to bad values, crashed
- Boeing 737 MAX - 2018, 2019
  - MCAS system over-corrected the plane's pitch. Two crashes totalling 346 fatalities

737 MAX

MCAS Software to prevent stalling; faulty sensor caused nosedive

Required manual override by the pilots to disengage. What is the system here?



AOA Sensor Stab Trim Cutout

Specification Goals

- Explain *what* to do, but now *how* to do it

Overall specifications should be:

- Complete
- Consistent
- Precise
- Concise

How do we verify, validate, and enforce our specifications?

Ernst, *Specifications*, 2005

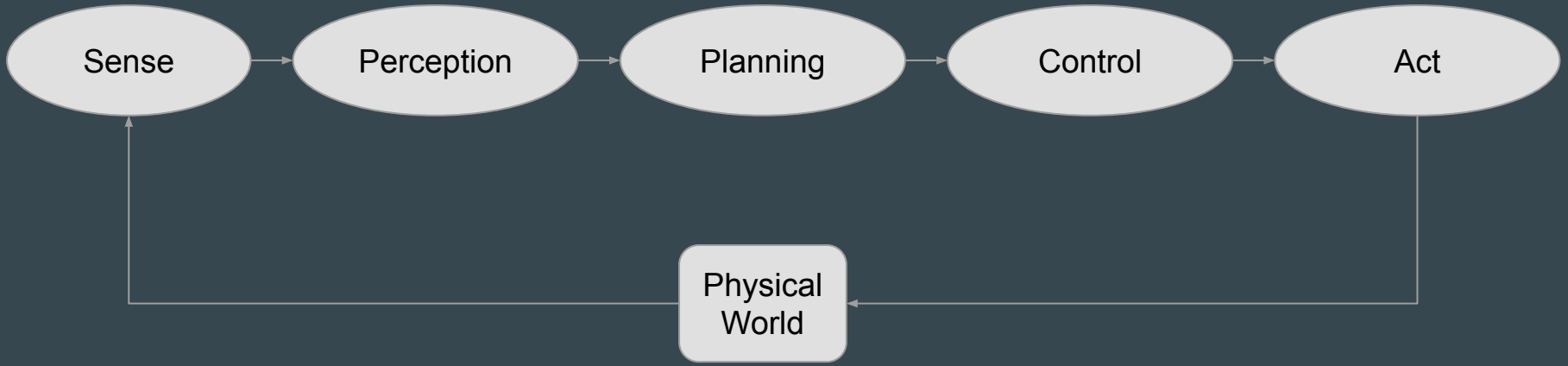
How can we design/check specifications?

If a path exists such that the robot can safely* navigate to the goal, it must be able to find it

*What we mean by “safely” can depend on the robot, its environment, etc. and must be rigorously specified
For example, recall in Lab 7 we added safe_distance as a tunable parameter
This altered the performance of the algorithm based on what threshold of safety was required

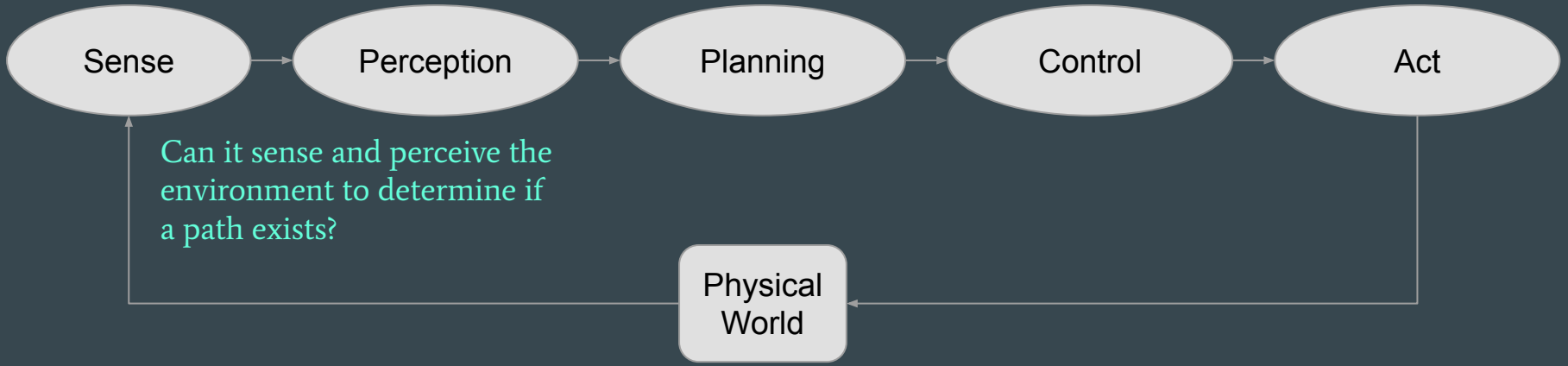
How can we design to specifications?

If a path exists such that the robot can safely* navigate to the goal, it must be able to find it



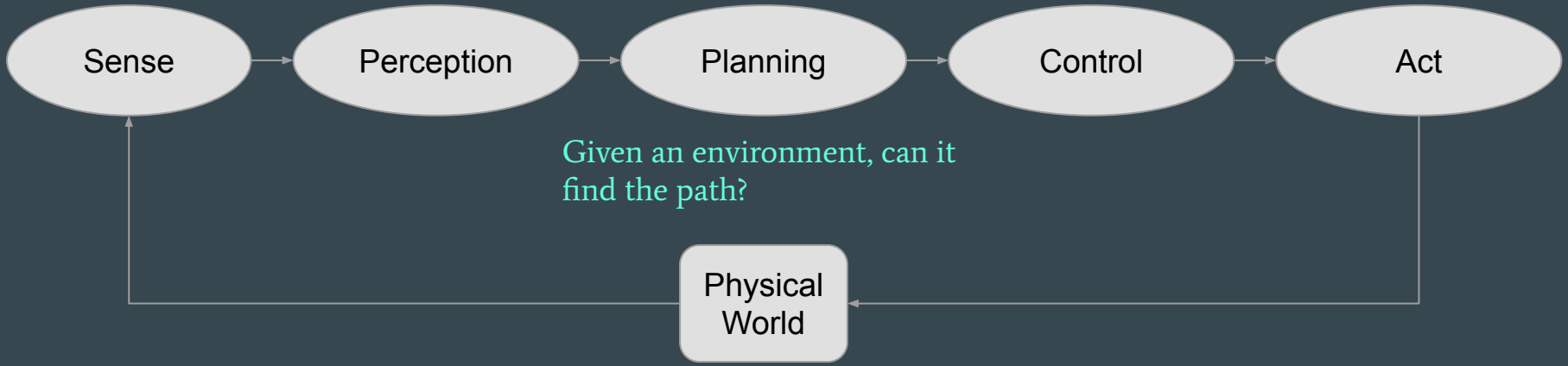
How can we design to specifications?

If a path exists such that the robot can safely* navigate to the goal, it must be able to find it



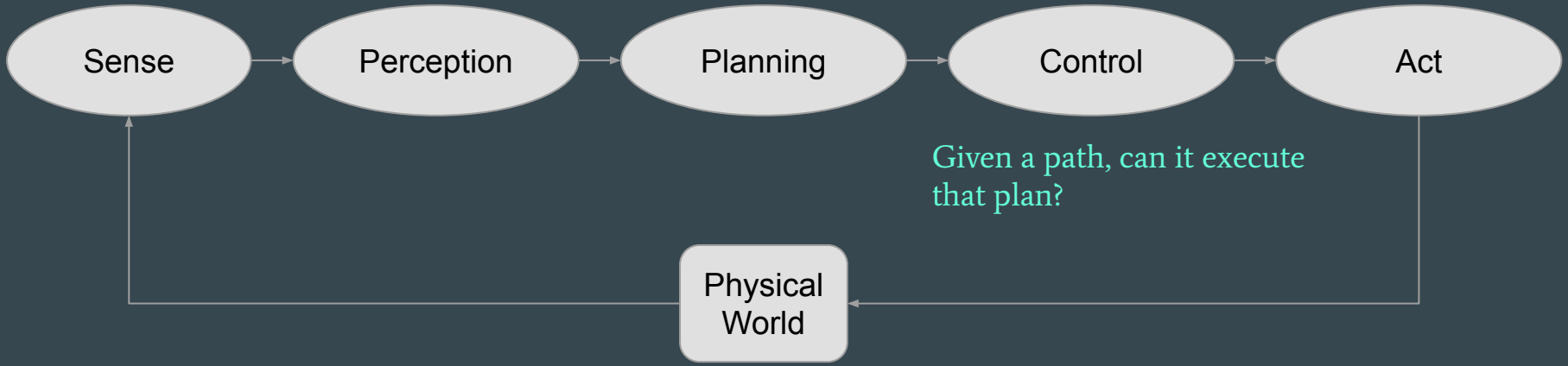
How can we design to specifications?

If a path exists such that the robot can safely* navigate to the goal, it must be able to find it



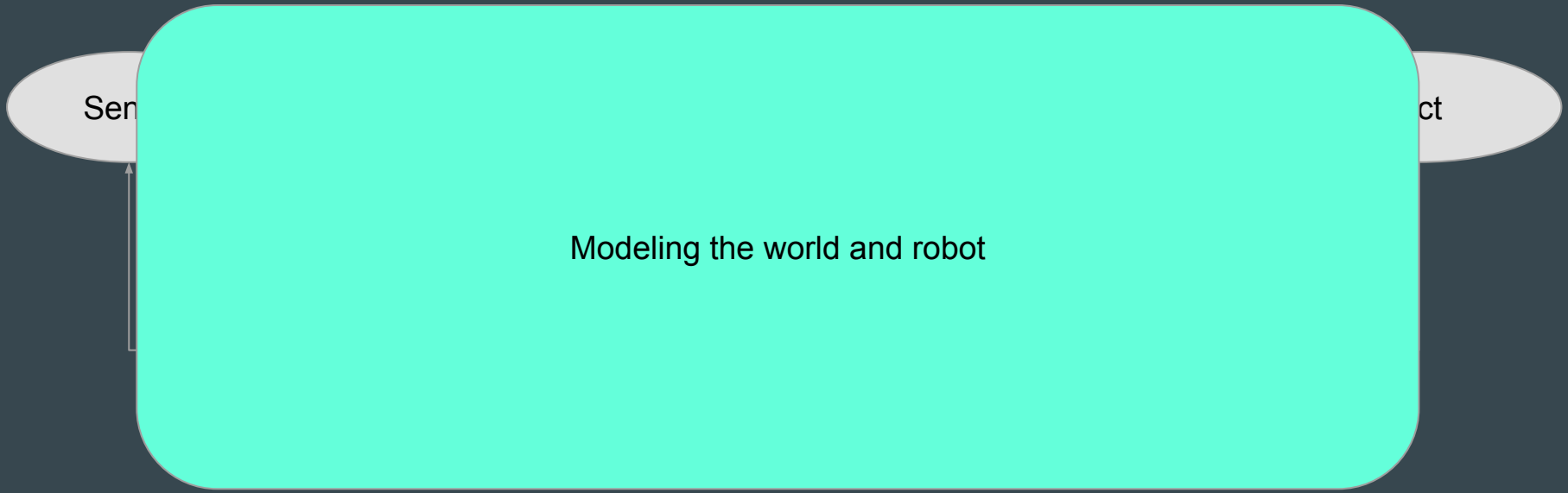
How can we design to specifications?

If a path exists such that the robot can safely* navigate to the goal, it must be able to find it



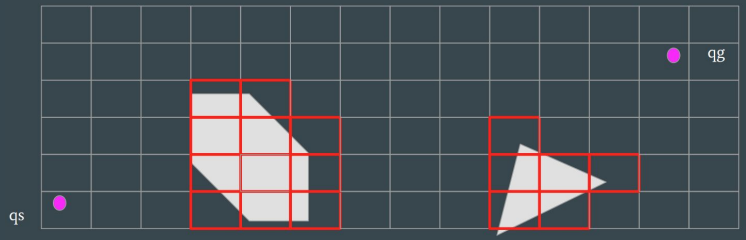
How can we design to specifications?

If a path exists such that the robot can safely* navigate to the goal, it must be able to find it



Modeling for Specifications

- Use abstractions!
- World:
  - Collection of objects in 2D space
    - Coarsely estimate objects as polygons
  - Occupancy Grid in 2D or 3D
    - All obstacles become grid cells
- Robot
  - A point in space that can move left, right, up, down
  - A point in space that can only move straight or turn



Modeling for Specifications

- Use abstractions!
- World:
  - Collection of objects in 2D space
    - Coarsely estimate objects as polygons
  - Occupancy Grid in 2D or 3D
    - All obstacles become grid cells
- Robot
  - A point in space that can move left, right, up, down
  - A point in space that can only move straight or turn

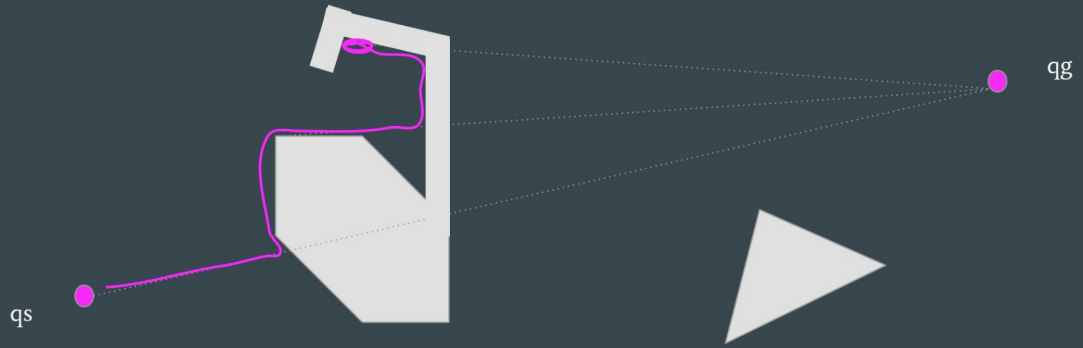
If a path exists such that the robot can safely* navigate to the goal, it must be able to find it

Given an 2D space and collection of obstacles, if a path is possible, the robot must be able to plan a path consisting of straight movement and turns

How can we design/check specifications?

Given an 2D space and collection of obstacles, if a path is possible, the robot must be able to plan a path consisting of straight movement and turns

Bug Algorithm 1



Repeat until Robot-pose = Goal
    Head towards goal
    If obstacle detected then
        Navigate next to wall to the left until heading towards goal is possible

Modeling for Specification

- Given a model we can:
  - Test/check the model for our specifications/safety properties
  - Evaluate how well our model captures the system

- What if the model is too different from the system?
  - False Positives: the model is found to be in violation of the specification, but the system does *not* violate the specification
  - False Negatives: the model is found to comply with the specification, but the system violates the specification

Modeling for Specification

- Given a model we can:
  - Evaluate how well our model captures the system
  - Test/check the model for our specifications/safety properties

- What if the model is too different from the system?
  - False Positives: the model is found to be in violation of the specification, but the system does *not* violate the specification
  - False Negatives: the model is found to comply with the specification, but the system violates the specification

How do we ensure safety?

We have specifications, so use them

Verification

- Analytically certify that the specification cannot be violated

Validation

- Gain empirical evidence that the specification is not violated

Monitoring

- While deployed, monitor that the specification is not violated

Verification

Can we be *prove* that a specification will hold in all cases?

Specification: thrust will always be positive

How do we build a model of this system?

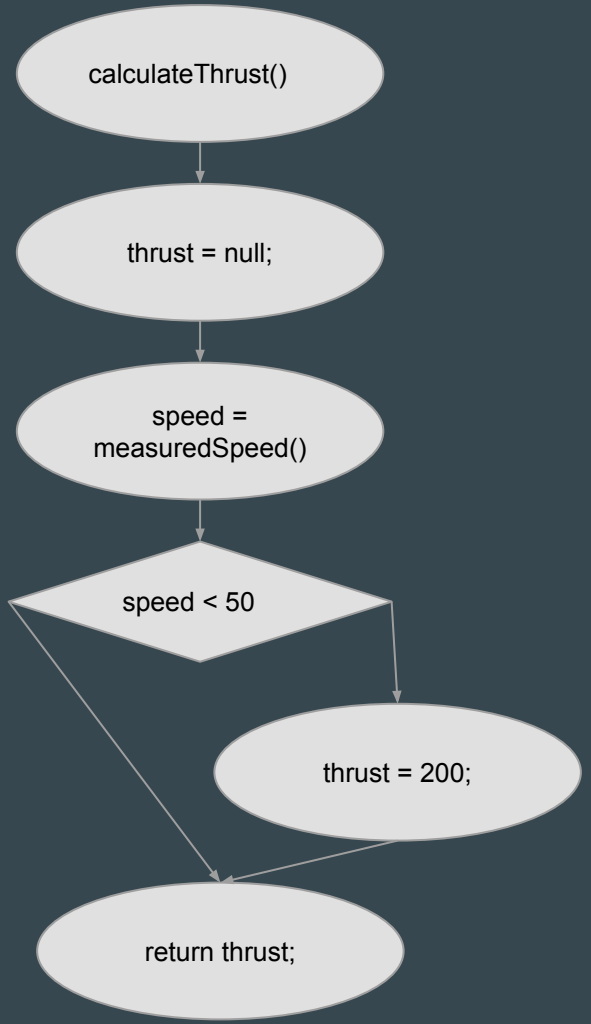
```
calculateThrust() {
    thrust = null;
    if (measureSpeed() < 50) {
      thrust = 200;
    }
   return thrust; // Will thrust always be positive?
}
```

Verification

Can we be *prove* that a specification will hold in all cases?

Specification: thrust will always be positive

```
calculateThrust() {
    thrust = null;
    if (measureSpeed() < 50) {
      thrust = 200;
    }
   return thrust;
}
```

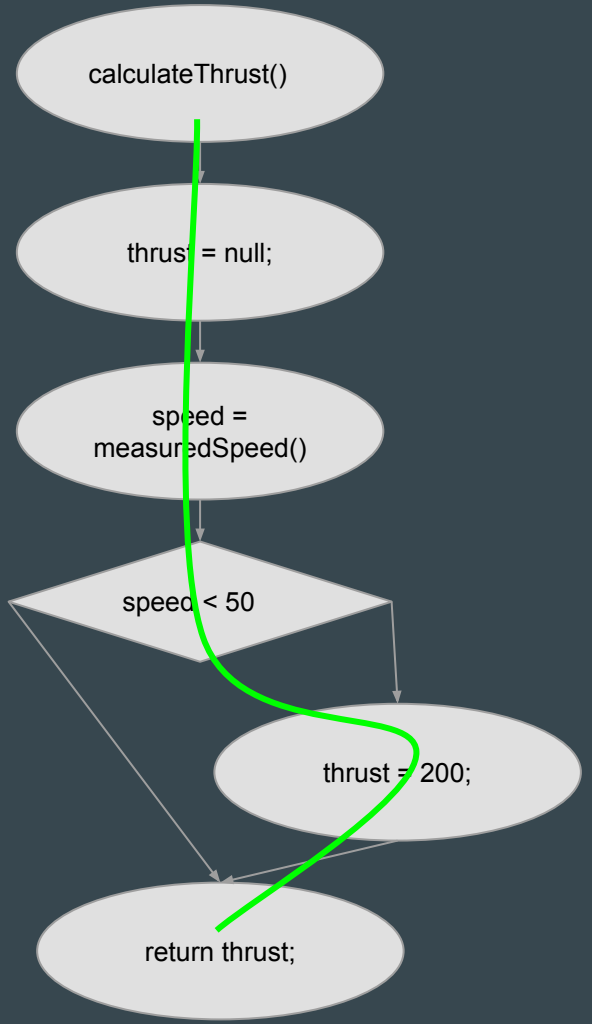


Verification

Can we be *prove* that a specification will hold in all cases?

Specification: thrust will always be positive

```
calculateThrust() {
    thrust = null;
    if (measureSpeed() < 50) {
      thrust = 200;
    }
   return thrust;
}
```

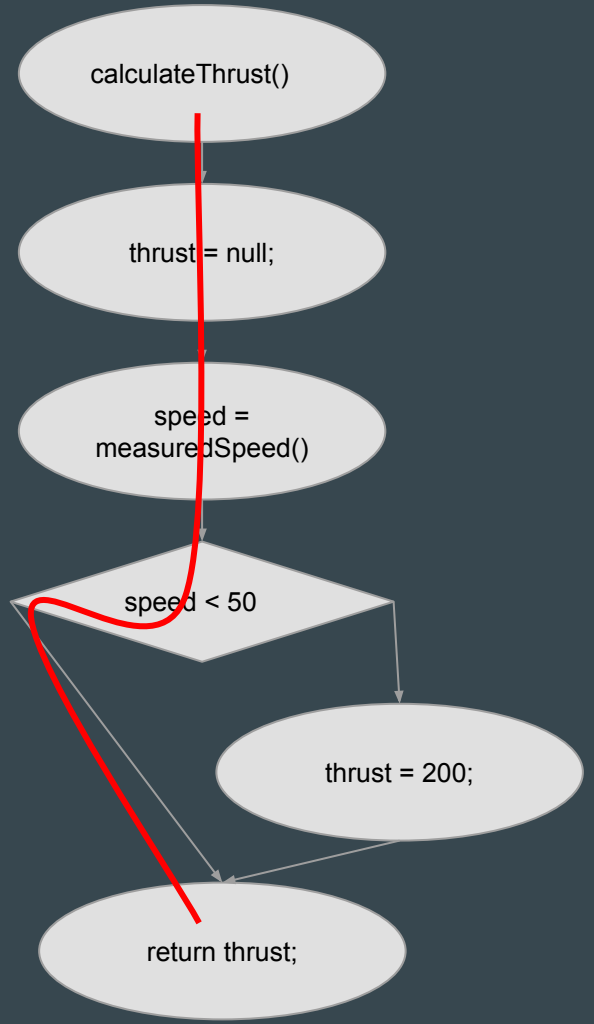


Verification

Can we be *prove* that a specification will hold in all cases?

Specification: thrust will always be positive

```
calculateThrust() {
    thrust = null;
    if (measureSpeed() < 50) {
      thrust = 200;
    }
   return thrust;
}
```



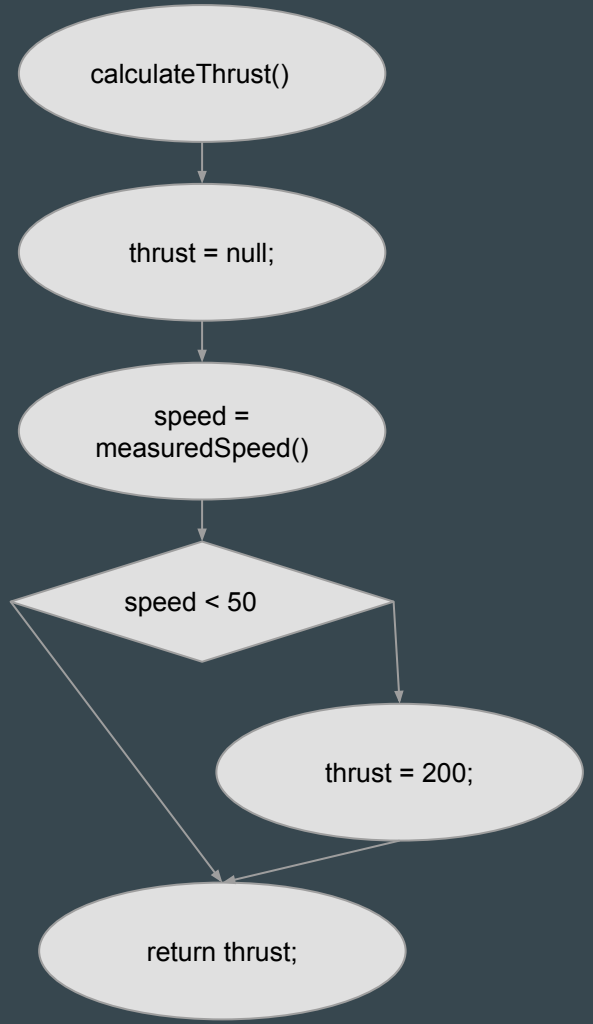
Verification

Can we be *prove* that a specification will hold in all cases?

Specification: thrust will always be positive

```
calculateThrust() {
    thrust = null;
    if (measureSpeed() < 50) {
      thrust = 200;
    }
   return thrust;
}
```

Any number - Spec Violated

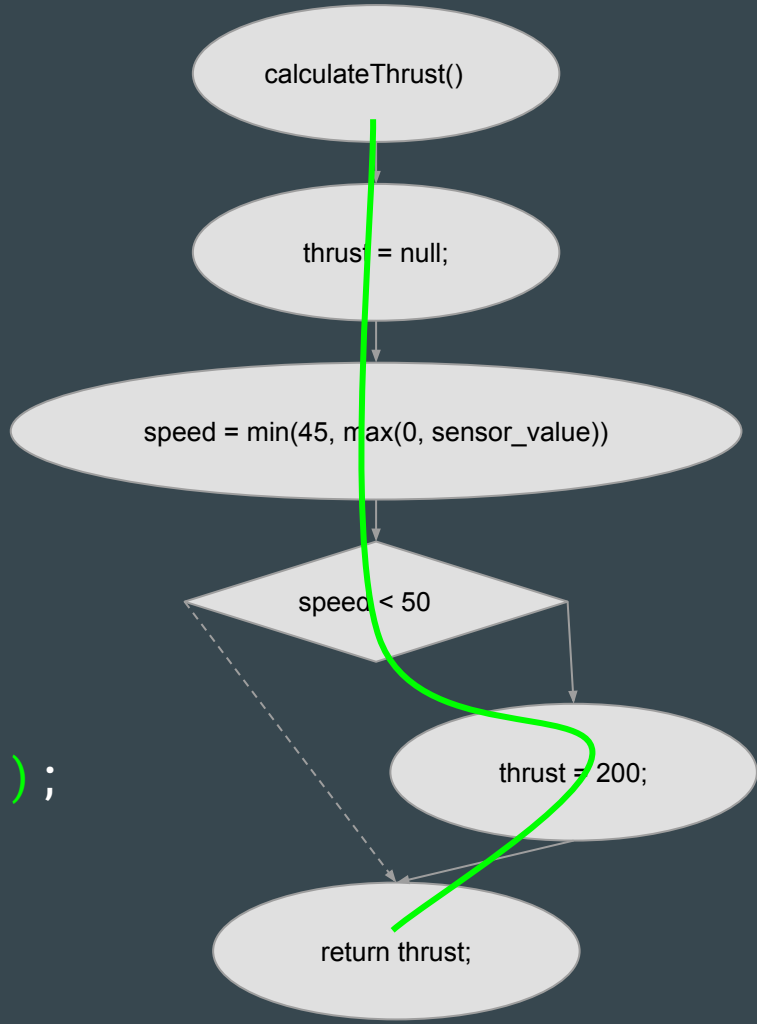


Verification

```
calculateThrust() {
    thrust = null;
    if (measureSpeed() < 50) {
      thrust = 200;
    }
   return thrust; // Will thrust always be positive?
}
measureSpeed() {
  return min(45, max(0, sensor_value));
}
```

Verification

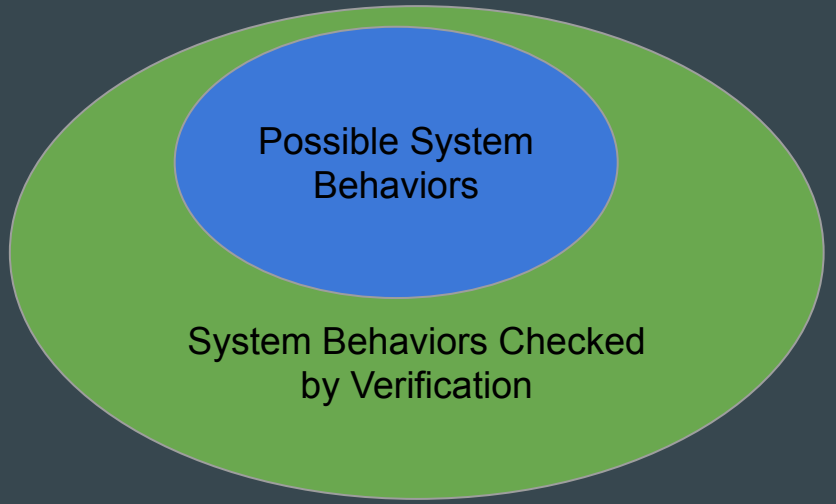
```
calculateThrust() {
    thrust = null;
    if (measureSpeed() < 50) {
      thrust = 200;
    }
   return thrust;
}
measureSpeed() {
  return min(45, max(0, sensor_value));
}
```



Verification and Over-approximation

- Verification says “for every system behavior, does the specification hold”
- For this to be useful, we must *over-approximate* the possible behaviors

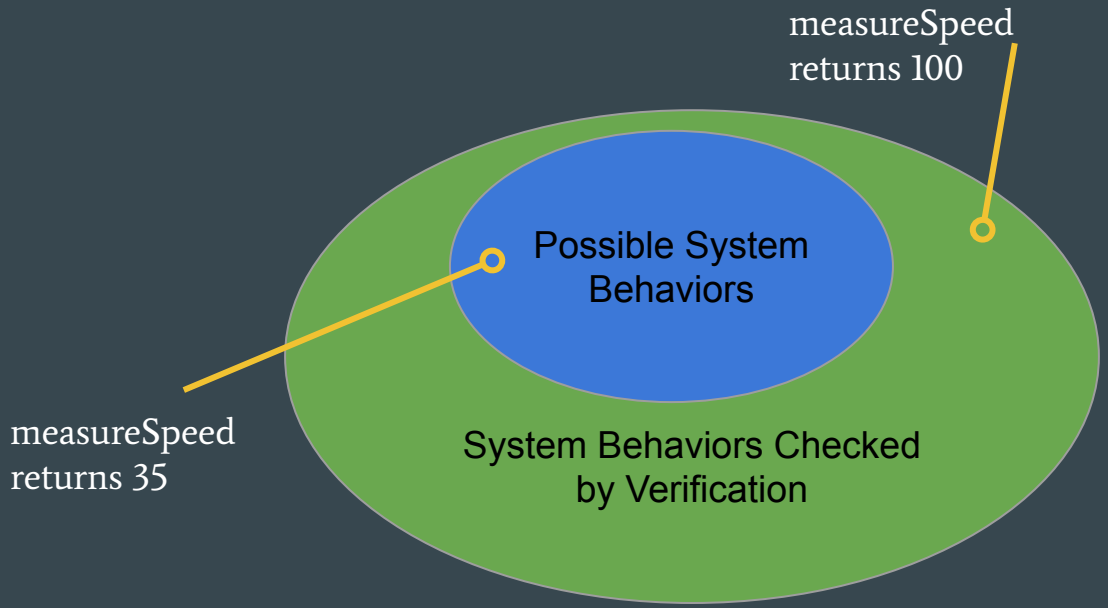
```
calculateThrust() {
    thrust = null;
    if (measureSpeed() < 50) {
      thrust = 200;
    }
   return thrust;
}
```



Verification and Over-approximation

- Verification says “for every system behavior, does the specification hold”
- For this to be useful, we must *over-approximate* the possible behaviors

```
calculateThrust() {
    thrust = null;
    if (measureSpeed() < 50) {
      thrust = 200;
    }
   return thrust;
}
```

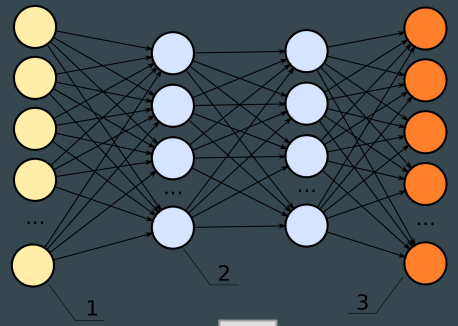


Verification of Complex Properties

- Machine Learning output
  - Given a range of inputs,
    verify there will be a specific range of outputs
- Control Plans
  - Temporal properties:
    verify that once the drone enters an area, it will never leave

Verification can be *expensive*:

- Computation time to check all behaviors
- Developing a model of the system
- False positives from over-approximation



Validation

Can we *test* that a specification is not violated?

Specification: thrust will always be positive

```
calculateThrust(measured_speed) {
    thrust = null;
    if (measured_speed < 50) {
      thrust = 200;
    }
   return thrust; // Will thrust always be positive?
}
```

Validation

Can we *test* that a specification is not violated?

Specification: thrust will always be positive

```
calculateThrust(measured_speed) {
    thrust = null;
    if (measured_speed < 50) {
      thrust = 200;
    }
   return thrust; // Will thrust always be positive?
}
```

Input (measured_speed)	Pass/Fail
0	Pass
-1	Pass
1	Pass
2	Pass

Validation

Can we *test* that a specification is not violated?

Specification: thrust will always be positive

```
calculateThrust(measured_speed) {
    thrust = null;
    if (measured_speed < 50) {
      thrust = 200;
    }
   return thrust; // Will thrust always be positive?
}
```

Input (measured_speed)	Pass/Fail
50	Fail
2147483647	Fail
Infinity	Fail
NaN	Fail

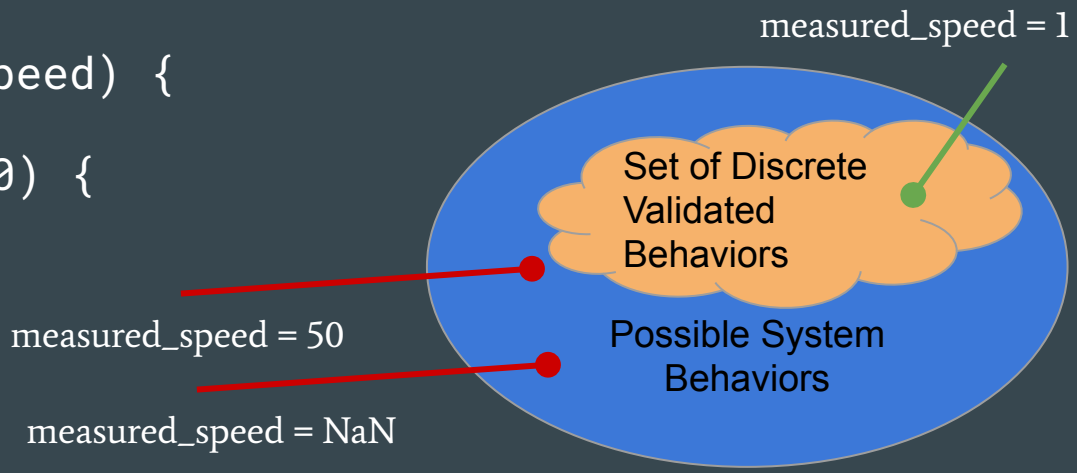
Validation and Under-approximation

Validation provides the most utility when it finds inputs that lead to violations

A lack of failures *does not* tell us no failures exist
Successful tests only provide *evidence* that a specification is not violated *for that input*

It is important to design a robust test plan to exercise your system

```
calculateThrust(measured_speed) {
    thrust = null;
    if (measured_speed < 50) {
      thrust = 200;
    }
   return thrust;
}
```



Runtime Monitoring

- While deployed, constantly check if a specification has been violated
- If a specification is violated (or about to be violated), intervene to take safe action

Watchdog timers:

- Constantly check if the system is “stuck” and intervene

Runtime Monitoring

- While deployed, constantly check if a specification has been violated
- If a specification is violated (or about to be violated), intervene to take safe action

```
thrust_monitor() {
  thrust = calculateThrust();
  if (monitor_violated(thrust)) { // Specification violated!
    turnOffEngine();
  }
}

monitor_violated(thrust) {
  return !(thrust > 0)
}
```

Runtime Monitoring

Therac-25: a radiation therapy system designed to deliver controlled amounts of targeted radiation to patients.

- Software bugs caused the system to emit lethal doses of radiation under certain conditions.

- How could a runtime monitor have prevented this?

Runtime Monitoring - Kill Switch

- A consistent way to incapacitate, immobilize, disarm, or otherwise disable a robot.
- Used in:
  - Industrial robotics
  - Field robotics
  - Nuclear reactors
  - Our robot vacuum example

“Lab” 9: In-class Exercise on Designing a Specification 5 points

- You will be given an initial system specification for a robot that needs to satisfy a go-to-goal mission in an indoor environment.
- You will use this initial specification to develop a refined specification *to handle all possible scenarios*
- **Before class on Wednesday** - submit on Collab answers to a series of questions about the initial specification (2.5 points)
- **During class on Wednesday** - we will divide into groups and have short discussions on prompts to refine your specifications (2.5 points)
- **By the end of Wednesday** - as a class we will have developed a cohesive specification

Scenario Description

Your robot serves the inside of Rice Hall, delivering Einstein Bros. throughout the building. The robot starts each mission at Einstein Bros. fully charged, is loaded with the customer's order, and is sent on a go-to-goal mission to one of the rooms in Rice. The robot has a map of the static environment (walls, doors, elevators, etc.) but does not know the locations of any dynamic obstacles (tables, chairs, people, etc.). By connecting wirelessly to the building's network, the robot can contact the elevator to wirelessly "push" any of the elevator's buttons. The robot must navigate through the building to the door outside of the customer, wait for the food to be delivered, then return to Einstein Bros. to charge or receive the next order.

Commands Sent to Robot

Robot has the following commands:

- Turn on
- Engage
- Disengage
- Emergency Stop
- Set Virtual Cage: takes in rectangular region in 2D space*
- Set exclusion zones: takes in set of polygons in 2D space* the robot is not allowed to enter
- Go to goal: takes in location in 2D space* *in the robot's current frame*
- Exclusion zone override: boolean whether or not to ignore the exclusion zone

*Each floor of the building can be a disjoint part of the 2D map

Robot capabilities

- LiDAR mounted on top of the robot. *(may assume infinite angular precision)*
- Bump sensor covering the entire front of the robot
- Motors attached to non-slip rubber tires can maintain speed of at least 5 m/s
- Robot can only rotate in place or move forward
- Battery capable of operation for at least 30 minutes
- Wireless capabilities to:
  - “Push” any button on the elevator. The elevator responds with its current state
  - Communicate back to the user/customer
- Top mounted stereo speakers
- Hatch that can open to release the order to the customer

Robot Requirements/Specifications

- The robot must always reach its goal, deliver the order, and return to base
- When the robot turns on, it is not engaged
- When e-stopped, the robot immediately ceases operation until it is powered off and on
- When disengaged, the robot safely stops operation
- The robot must never enter an exclusion zone or leave the virtual cage
- The robot must always preserve its ability to complete the mission
- The robot must never move while disengaged
- The robot must never run out of battery
- The robot must never collide with an obstacle at a speed of >2 m/s
- The robot must immediately stop contact if it collides with an obstacle
- If the goal cannot be achieved because of an exclusion zone, the robot must ask the user if it can ignore the exclusion zone to reach the goal

Initial Questions (2.5 points)

In order to satisfy the requirements:

1. What happens when a GOAL command is received after an ESTOP command?
2. What is the mission cruising speed of the robot?
3. How does the robot respond when the virtual cage command sends a new cage that does not contain the robot?
4. What happens if the robot approaches an obstacle on the way to its goal?

Other question:

Given these requirements, describe one scenario you are unsure about designing a specification for (e.g. design a question like the above)