



Preparing Software Engineers to Develop Robot Systems

Carl Hildebrandt, Meriel von Stein, Trey Woodlief, and Sebastian Elbaum

The University of Virginia

hildebrandt.carl@virginia.edu

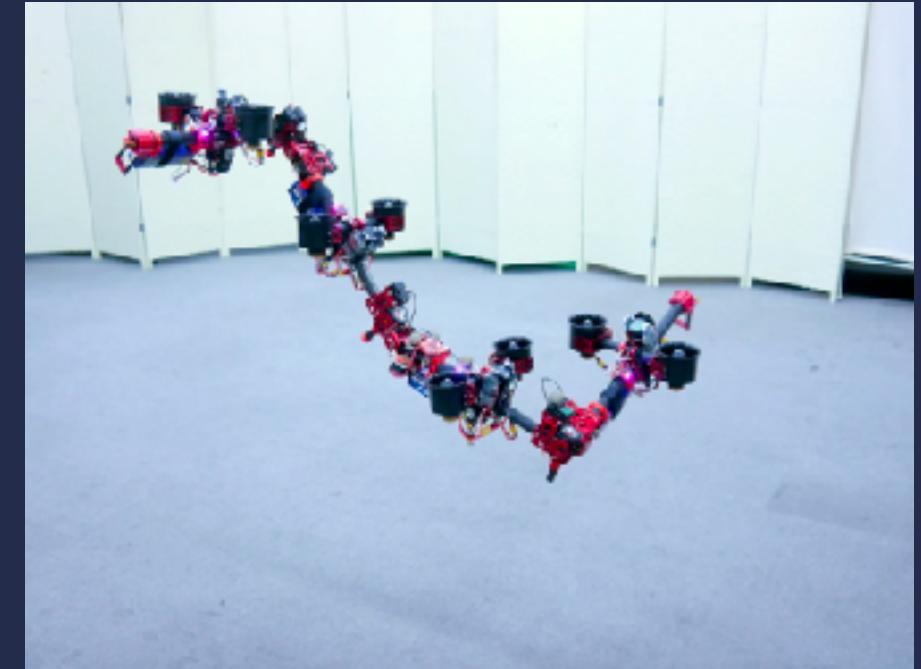


<https://less-lab-uva.github.io/CS4501-Website/>



Robotics

Robotics as a field has grown steadily for the last two decades



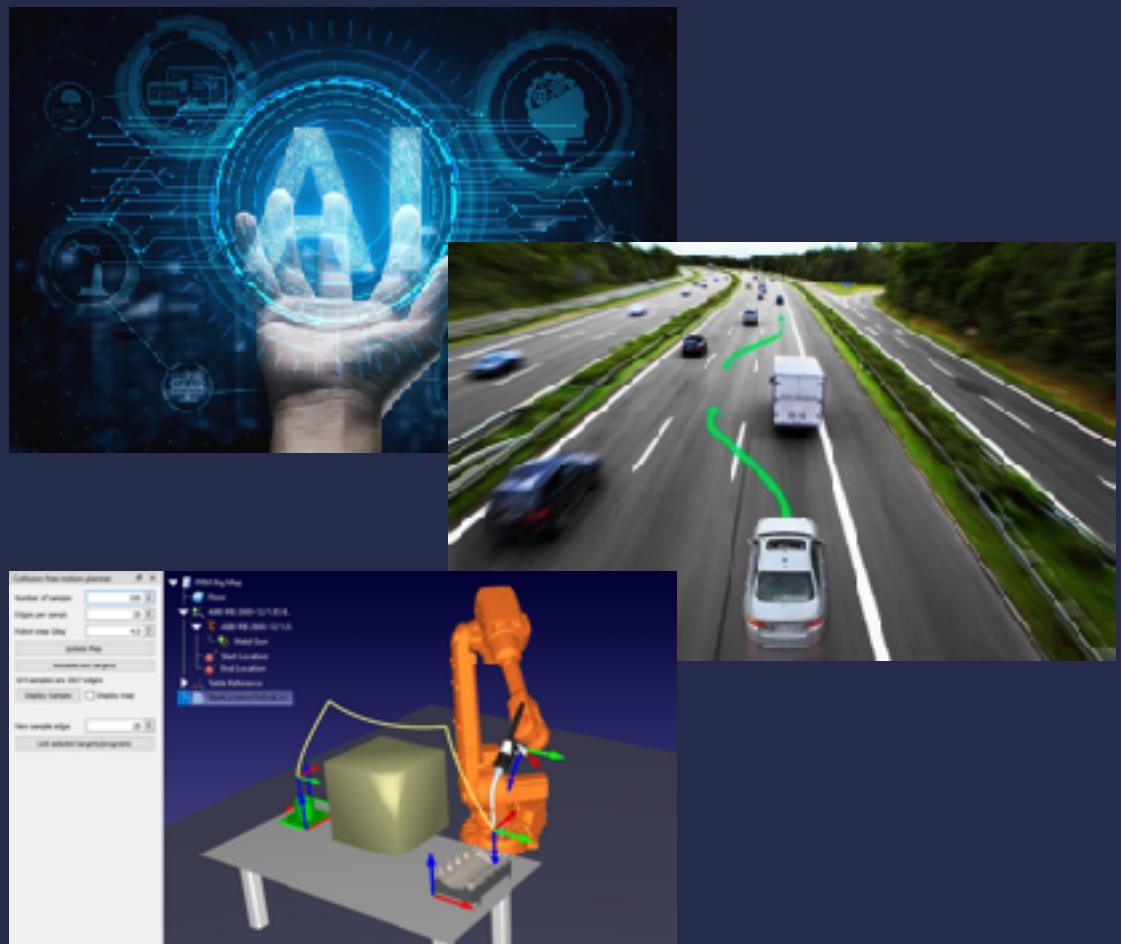
Robotics

The robotics industry is projected to grow by 25% between 2020-2025

Traditional CS Curricula

Preparing our future software engineers for this has been met primarily through:

Specialized Graduate-Level Courses



CPS/Embedded Systems Courses

CS 3630
Introduction to Robotics and Perception

Course I
SPEECH RECOGNITION
Cognitive Science
Machine Perception
Robotics
Logistics

Requirements
Prerequisites / Introductory
Carnegie Mellon University
Carnegie Mellon University
Robotics Institute
Contact: Barbara Webb (B. J. Webb) bwmw@cs.cmu.edu

Course I
SPEECH RECOGNITION
Cognitive Science
Machine Perception
Robotics
Logistics

Requirements
Prerequisites / Introductory
Carnegie Mellon University
Carnegie Mellon University
Robotics Institute
Contact: Barbara Webb (B. J. Webb) bwmw@cs.cmu.edu

Massive Open Online Courses

MITOPENCOURSEWARE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

FIND COURSES
For Educators
Give Now
About
Search

Introduction to Robotics

Course Home
COURSES
SEARCH
ABOUT
ASSIST
DRAW
PROJECT
DOWN
MAIL

ROBOT LOCALIZATION WITH PARTICLE FILTERS

Robot localization with particle filters

Robot localization with particle filters

Traditional SE Courses

Online Bachelor of Science in Software Engineering

With an online software engineering degree, you can pursue professional paths in application development, database and systems administration, software and web development, and more. The demand for software engineers is growing.

George Mason University - Mason

Software Engineering masters drives the digital space

Advances in software engineering are everywhere. Software powers the apps in our handheld devices, and the GPS in our cars.

It helps ride services, orders dinner, helps physicians diagnose disease, and protects us from cyber-attacks. With a master's degree in Software Engineering from George Mason University you will:

- Improve software -

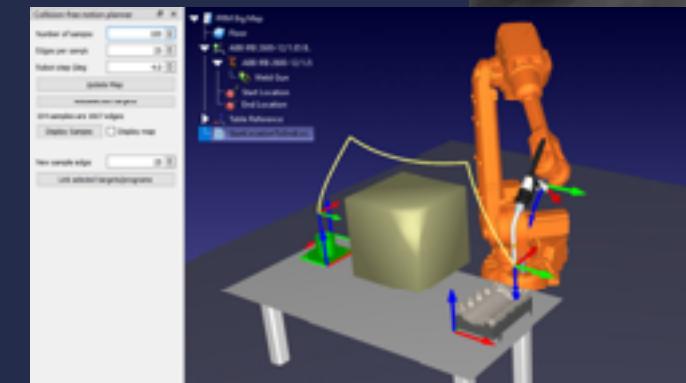
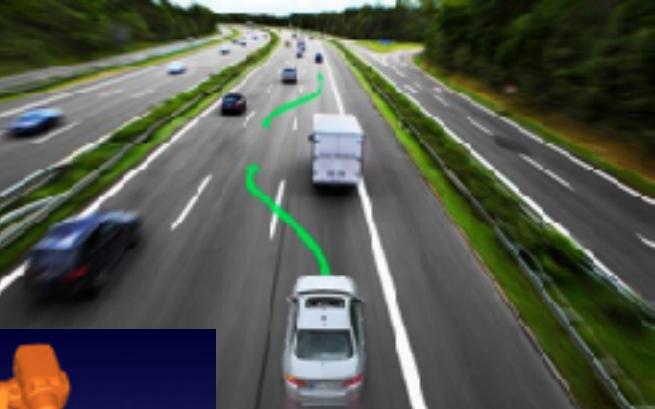
Create innovations for computer games, business applications, operating systems, network control systems, and middleware.

Traditional CS Curricula

Preparing our future software engineers for this has been met primarily through:

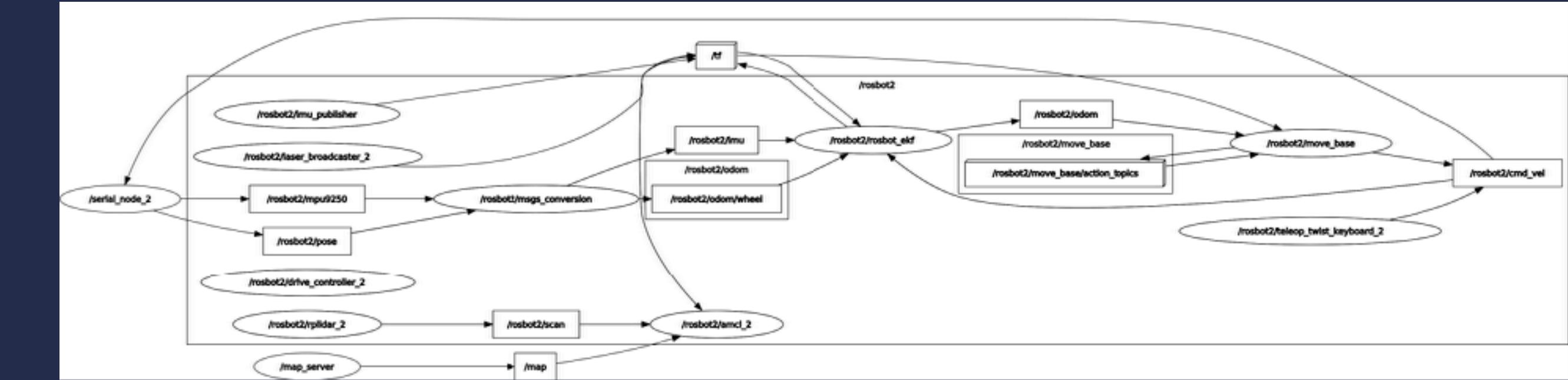
Specialized

Graduate-Level Courses



Overlooks that robotics heavily relies on software.

Assumes students are familiar with software engineering.



```
252
253
254
255
256
257
258
259
260
261
262
263
264
265 }
```

```
document.getElementById('photoDescription').innerHTML = descriptions[page * 5 + currentImageNumber() - 1];
document.getElementById('bigImageDesc').innerHTML = descriptions[page * 5 + currentImageNumber() - 1].description;
}

function updatePhotoDescription(){
if(descriptions.length > (page * 5) + (currentImageNumber() - 1)){
document.getElementById('bigImageDesc').innerHTML = descriptions[page * 5 + currentImageNumber() - 1].description;
}

}

function updateAllImages(){
var i = 1;
while(i < 10){
var elementId = 'foto' + i;
var elementIdBig = 'bigImage' + i;
}
```

Traditional CS Curricula

Preparing our future software engineers for this has been met primarily through:

Specialized Graduate-Level Courses



Overlooks that robotics heavily relies on software.

Assumes students are familiar with software engineering.

CPS/Embedded Systems Courses

CS 3630
Introduction to Robotics and Perception

Requirements

Courseware

Animation

Machine Perception

Cognition and Reasoning

Massive Open Online Courses

MITOPENCOURSEWARE

Introduction to Robotics

Robot Localization with Particle Filters

In this guided project, you will:

- Code a particle filter from scratch in Python and use it to solve the robot localization problem.

Traditional SE Courses

Online Bachelor of Science in Software Engineering

With an online software engineering degree, you can pursue professional paths in application development, database and systems administration, software and web development, and more. The demand for software engineers is growing rapidly, and there are many opportunities available.

Software Engineering masters drives the digital space

Advances in software engineering are everywhere. Software powers the apps in our handheld devices, and the GPS in our cars. It helps ride services, orders dinner, helps physicians diagnose disease, and protects us from cyber-attacks. With a master's degree in Software Engineering from George Mason University you will:

- Improve software -

Create innovations for computer games, business applications, operating systems, network control systems, and middleware.

Traditional CS Curricula

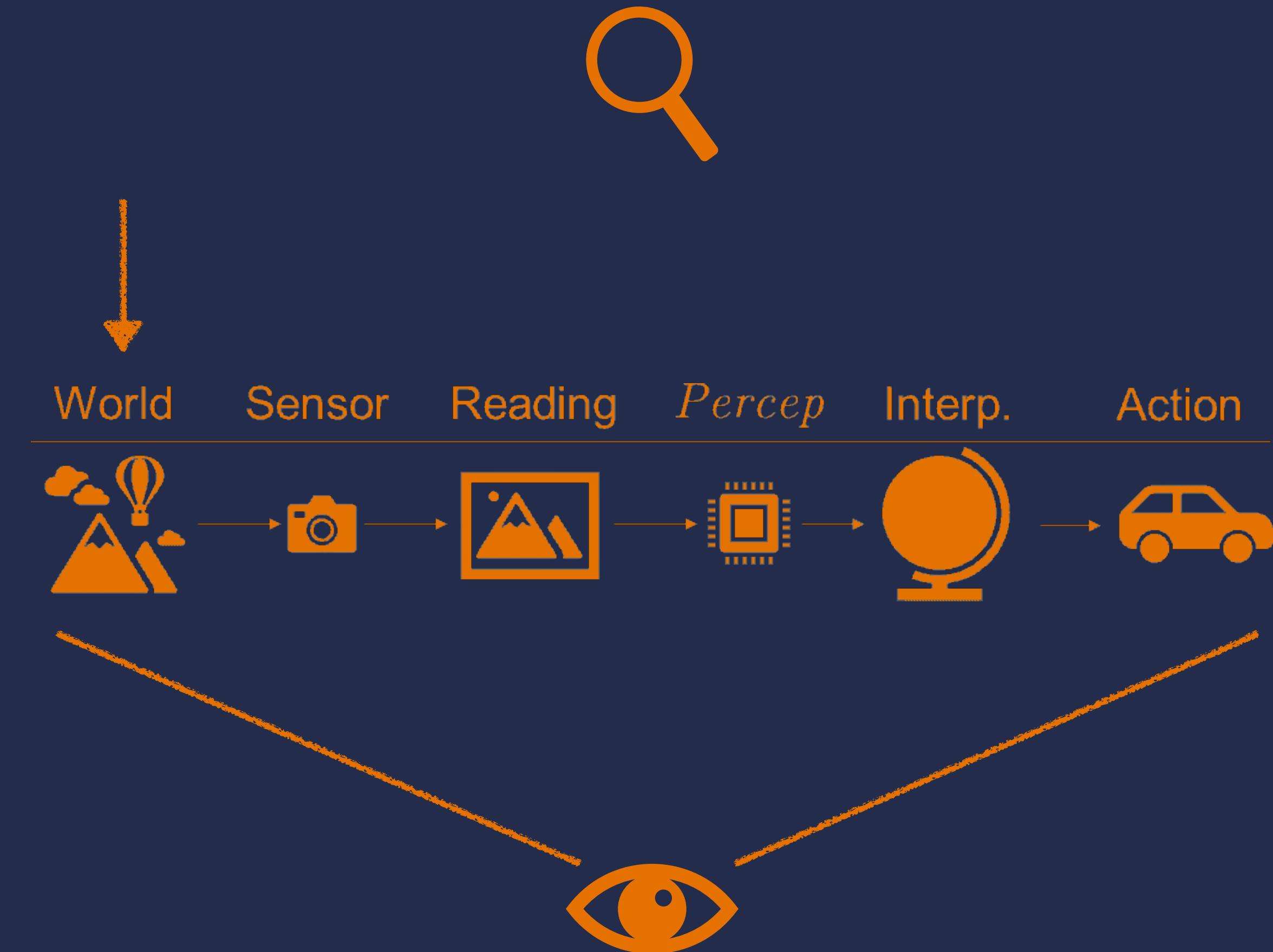
Preparing our future software engineers for this has been met primarily through:

CPS/Embedded
Systems Courses

The screenshot shows the course page for CS 3630: Introduction to Robotics and Perception at Carnegie Mellon University. The page includes a navigation bar with links like Home, News, About, People, Research, Education, News, Events, and NREC. A sidebar on the left provides course details such as 'Syllabus' and 'Logistics'. The main content area lists course requirements, including 'Requirements' (Control Theory), 'Course Information' (Robotics), 'Carnegie' (Kinematics), 'Machine Perception' (Computer Vision), and 'Cognition and Reasoning' (Artificial Intelligence). Each section lists specific courses and descriptions.

Tends to focus on **particular aspects** of robot pipeline.

Misses opportunities to discuss **broader crosscutting issues**.



Traditional CS Curricula

Preparing our future software engineers for this has been met primarily through:

Specialized Graduate-Level Courses



Overlooks that robotics heavily **relies on** software.

Assumes students are familiar with software engineering.

CPS/Embedded Systems Courses

CS 3630
Introduction to Robotics and Perception

Requirements

Courseware

Animation

Machine Perception

Cognition and Reasoning

Tends to focus on **particular** aspects of robot pipeline.

Misses opportunities to discuss **broader crosscutting** issues.

Massive Open Online Courses

MITOPENCOURSEWARE

Introduction to Robotics

Robot Localization with Particle Filters

Traditional SE Courses

Online Bachelor of Science in Software Engineering

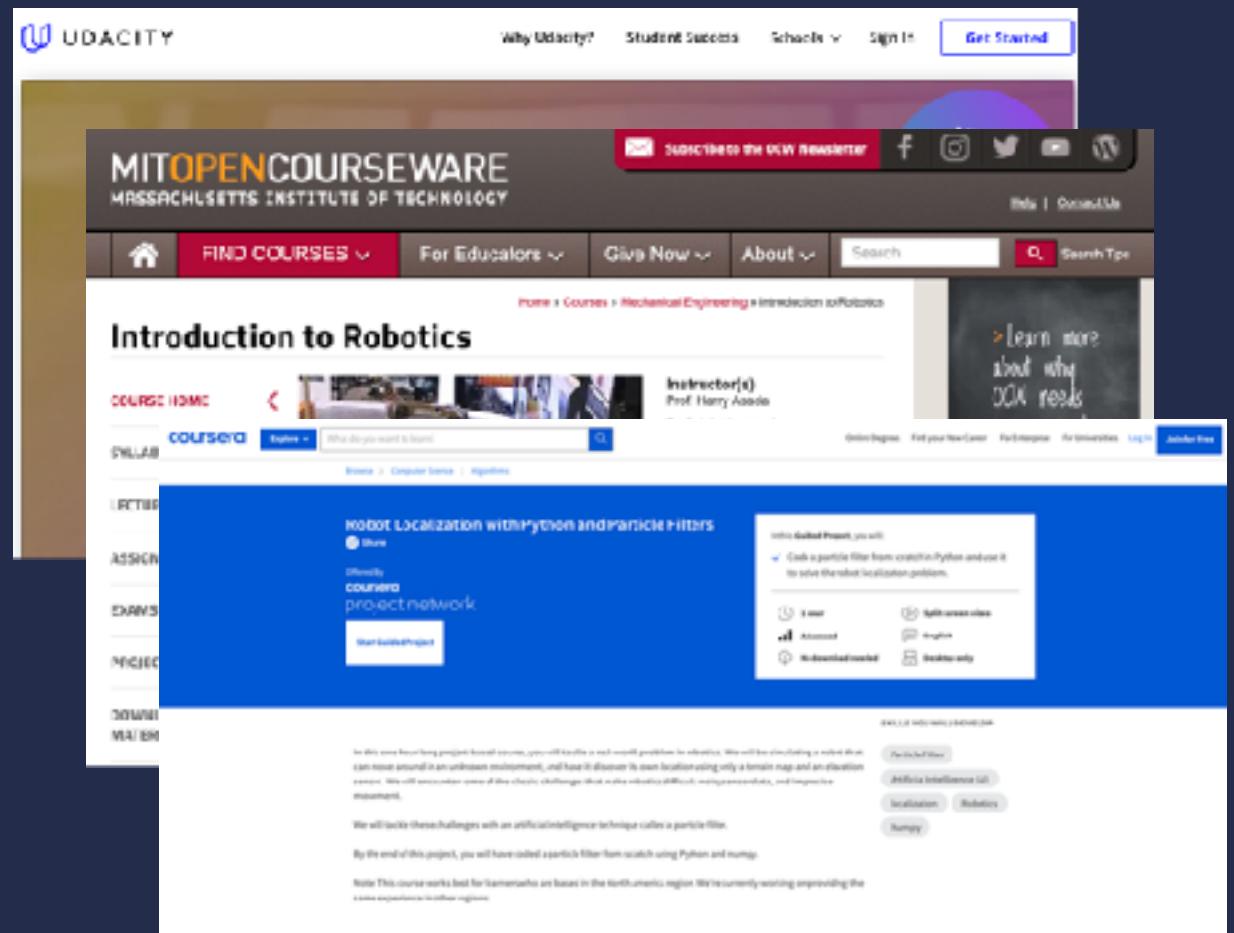
Software Engineering masters drives the digital space

Build solutions with code

Traditional CS Curricula

Preparing our future software engineers for this has been met primarily through:

Massive Open
Online Courses



Aims for a **breadth** of student applicants.

No prerequisites resulting in students that may **lack** **fundamental** software engineering **principles**.

Traditional CS Curricula

Preparing our future software engineers for this has been met primarily through:

Specialized Graduate-Level Courses



Overlooks that robotics heavily **relies on** software.

Assumes students are familiar with software engineering.

CPS/Embedded Systems Courses

CS 3630
Introduction to Robotics and Perception

Requirements

Course Details

Schedule

Topics

Robotics

Machine Perception

Cognition and Reasoning

Tends to focus on **particular** aspects of robot pipeline.

Misses opportunities to discuss **broader crosscutting** issues.

Massive Open Online Courses

MITOPENCOURSEWARE

Introduction to Robotics

Syllabus

Project Network

Aims for a **breadth** of student applicants.

No prerequisites resulting in students that may **lack** fundamental software engineering principles.

Traditional SE Courses

Online Bachelor of Science in Software Engineering

Software Engineering masters drives the digital space

Traditional CS Curricula

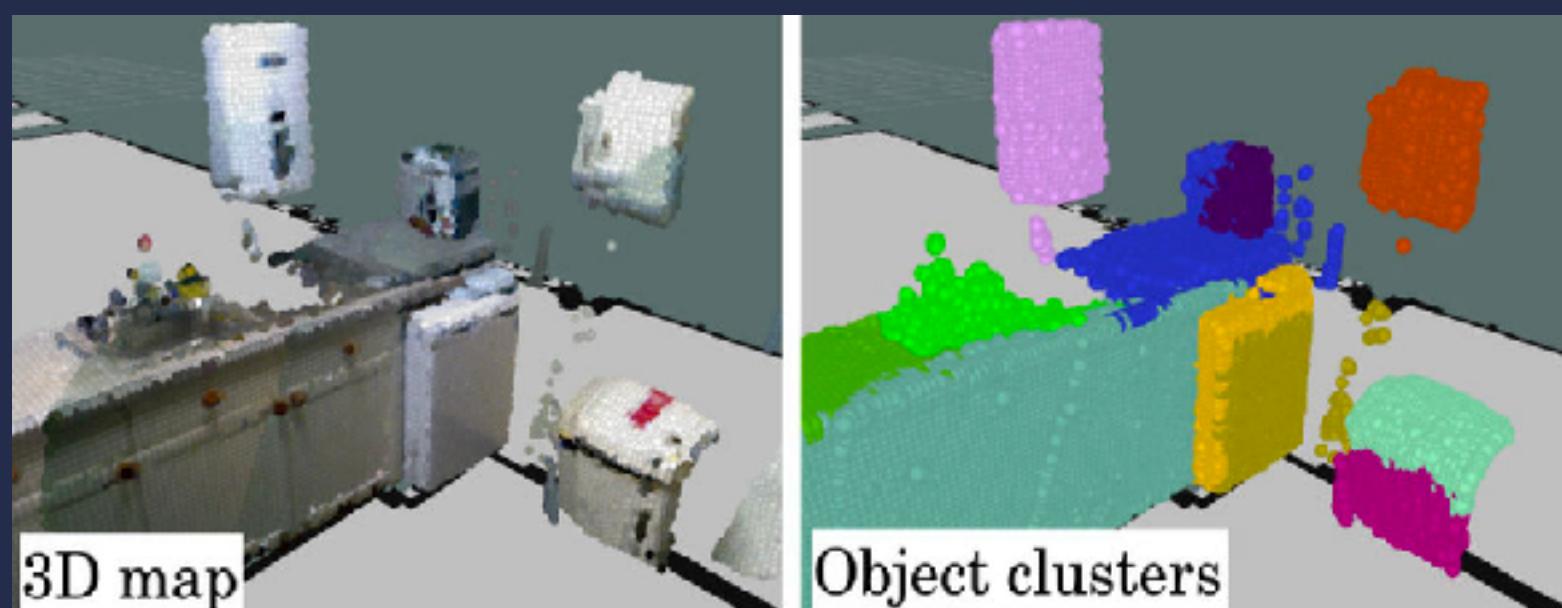
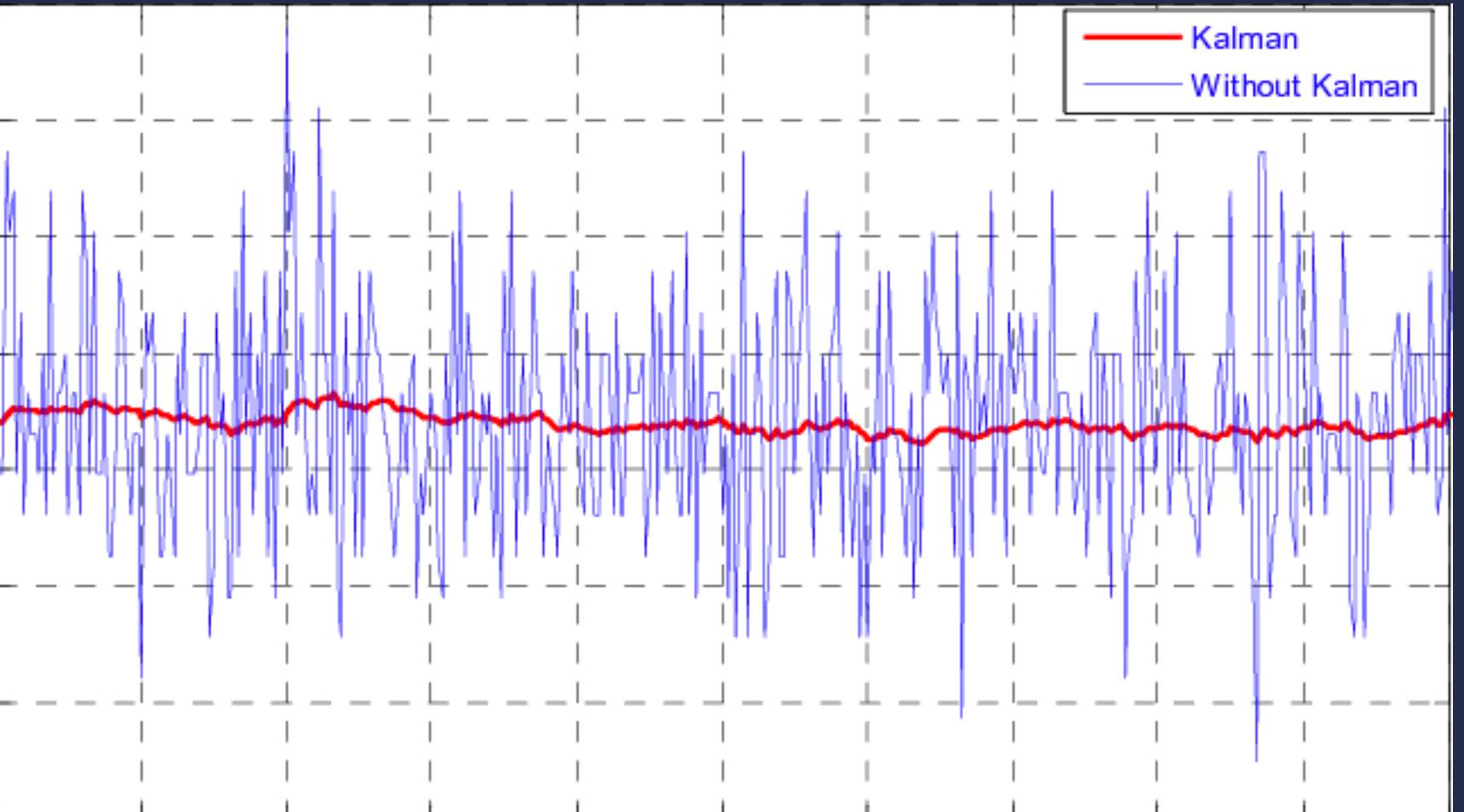
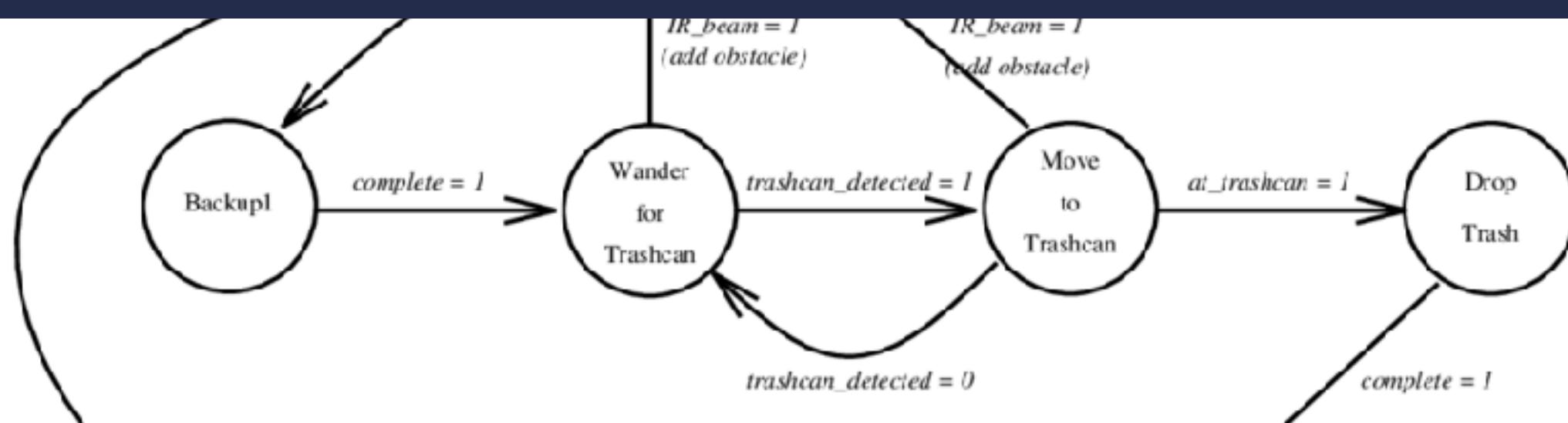
Preparing our future software engineers for this has been met primarily through:

Traditional SE Courses



Does not handle aspects specific to robotic systems.

For example, representation of environment, noise, complex definitions of state.



Traditional CS Curricula

Preparing our future software engineers for this has been met primarily through:

Specialized Graduate-Level Courses



Overlooks that robotics heavily **relies on** software.

Assumes students are familiar with **software engineering**.

CPS/Embedded Systems Courses

Course Requirements:

- Requirements: Choose to earn credit in each requirement (you can earn credit in all)
- Prerequisites: None
- Course Details:
 - Course Name: CS 3630
 - Course Description: Introduction to Robotics and Perception
 - Prerequisites: None
 - Offered: Fall 2023
 - Instructor: Prof. Harry Asada
 - Office Hours: TBA
 - Office Location: TBA
 - Office Phone: TBA
 - Office Email: TBA
 - Office Address: TBA
 - Office Hours: TBA
 - Office Location: TBA
 - Office Phone: TBA
 - Office Email: TBA
 - Office Address: TBA
- Syllabus: TBA

Tends to focus on **particular aspects** of robot pipeline.

Misses opportunities to discuss **broader crosscutting issues**.

Massive Open Online Courses

Course Details:

- Course Name: Introduction to Robotics
- Prerequisites: None
- Offered: Fall 2023
- Instructor: Prof. Harry Asada
- Office Hours: TBA
- Office Location: TBA
- Office Phone: TBA
- Office Email: TBA
- Office Address: TBA

Project Network:

- Robot Localization with Particle Filters
- Mobile Robots
- Project Network

Aims for a **breadth** of student applicants.

No prerequisites resulting in students that may **lack fundamental software engineering principles**.

Traditional SE Courses

Online Bachelor of Science in Software Engineering

With an online software engineering degree, you can pursue professional paths in application development, database and systems administration, software and web development, and more. The demand for software engineers is high, and the need for skilled professionals is growing.

Software Engineering masters drives the digital space

Advances in software engineering are everywhere. Software powers the apps in our handheld devices, and the GPS in our cars. It helps ride services, orders dinner, helps physicians diagnose disease, and protects us from cyber-attacks. With a master's degree in Software Engineering from George Mason University you will:

- Improve software -

Create innovations for computer games, business applications, operating systems, network control systems, and middleware.

Does not handle aspects specific to **robotic systems**.

For example, representation of **environment, noise, complex definitions of state**.

Goal

Developing a course that would **enable upper-level undergraduate students** in computational disciplines to gain expertise on foundational aspects of **software development for robotics**

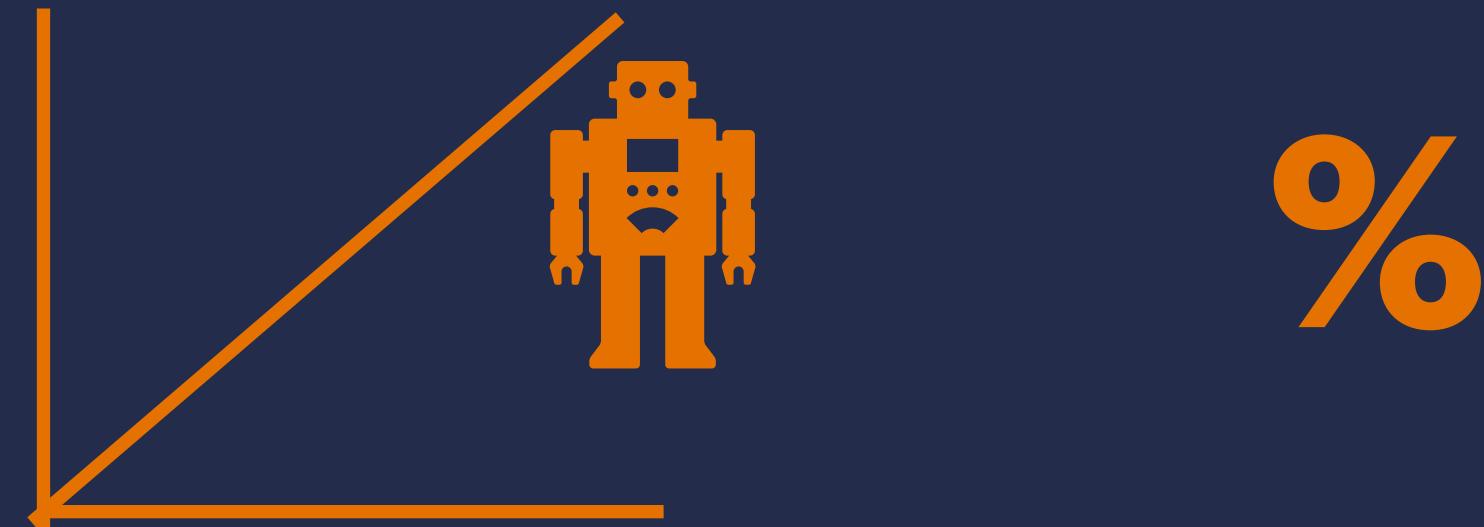
Whats New?

Link between Software Engineering and Robotics

Specifications



Testing



Uncertainty representation

%

Design patterns



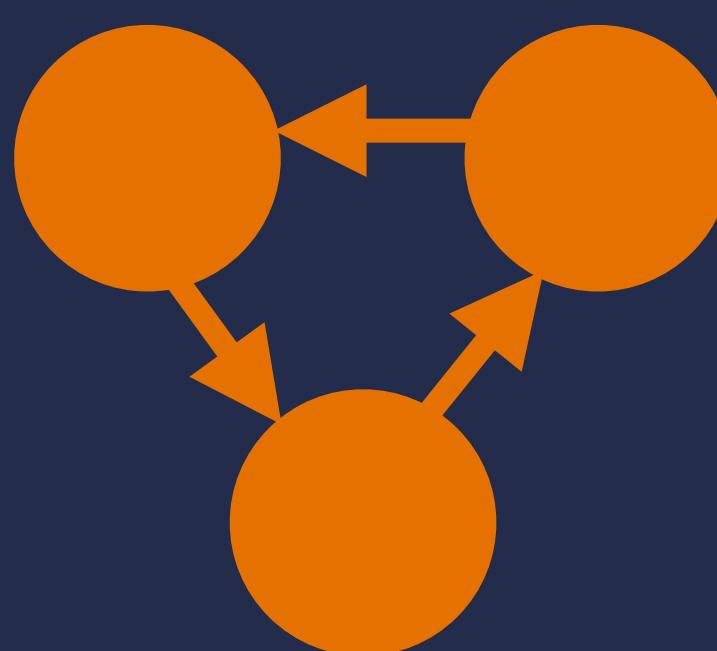
Reuse



Abstractions



States

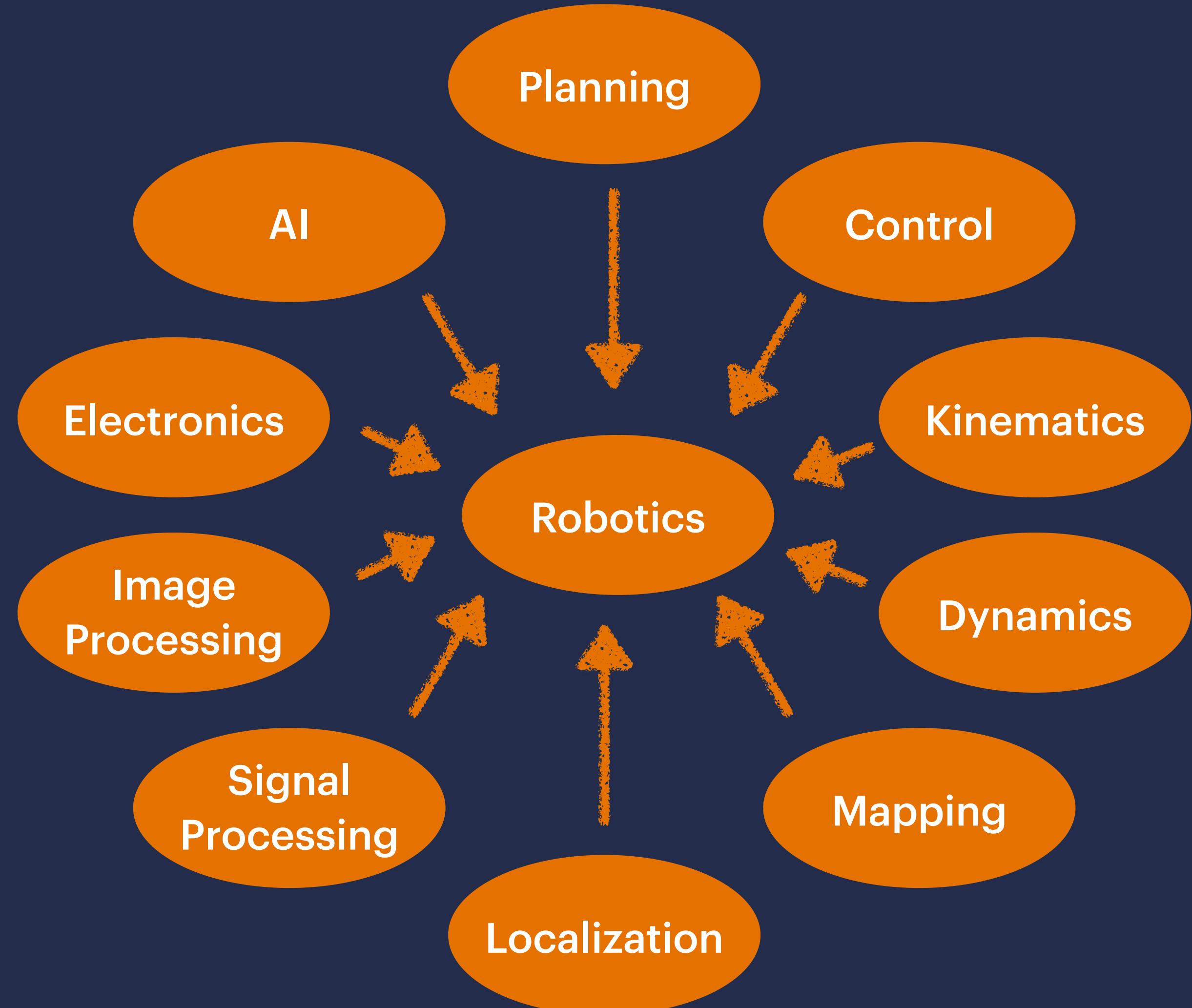


Challenges

1. Multidisciplinary and rapidly expanding field
2. How to distribute the emphasis between robotics and software engineering
3. No available integrated platform
4. Robotics courses can require significant upfront investment in equipment

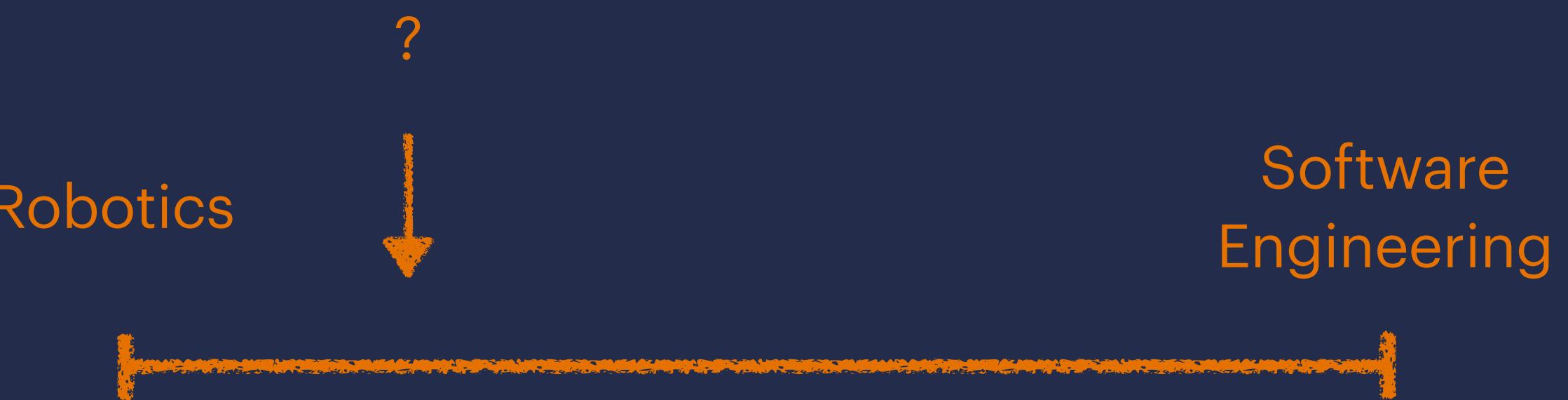
Challenges

1. Multidisciplinary and rapidly expanding field
2. How to distribute the emphasis between robotics and software engineering
3. No available integrated platform
4. Robotics courses can require significant upfront investment in equipment



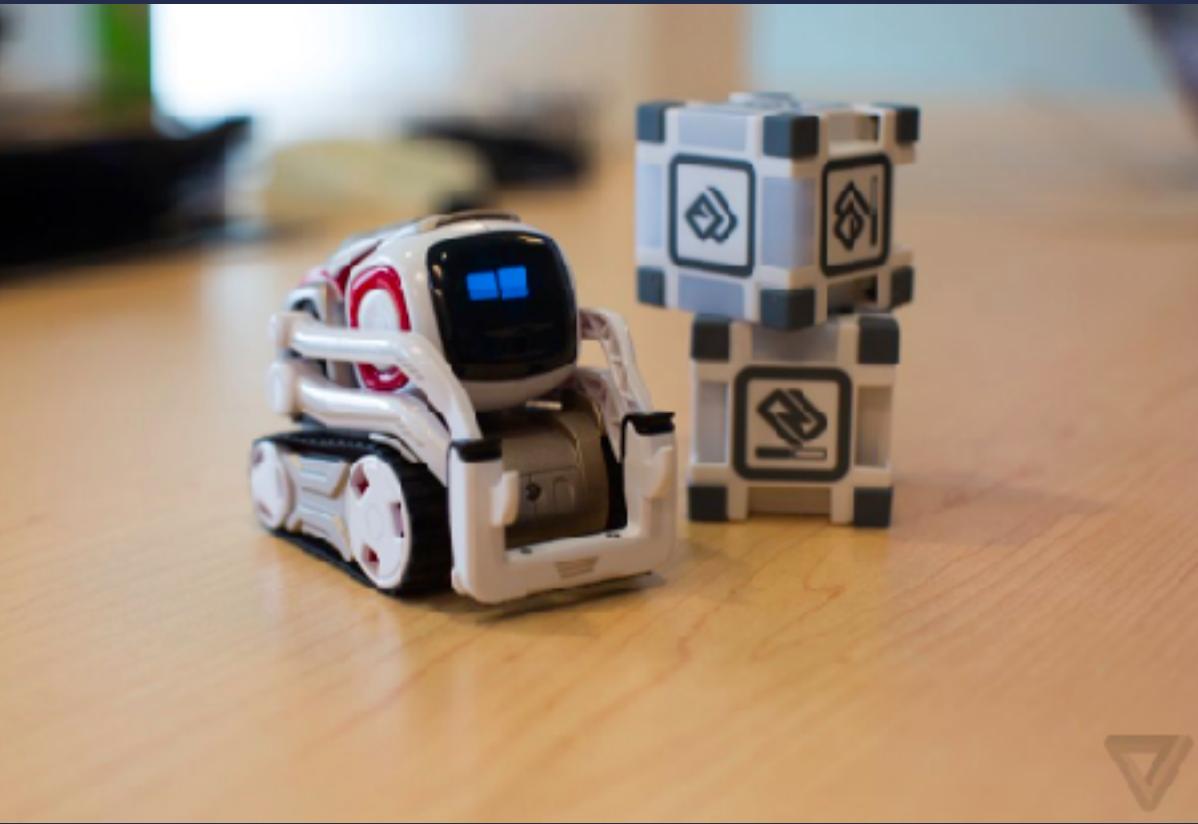
Challenges

1. Multidisciplinary and rapidly expanding field
2. How to distribute the emphasis between robotics and software engineering
3. No available integrated platform
4. Robotics courses can require significant upfront investment in equipment



Challenges

1. Multidisciplinary and rapidly expanding field
2. How to distribute the emphasis between robotics and software engineering
3. No available integrated platform
4. Robotics courses can require significant upfront investment in equipment



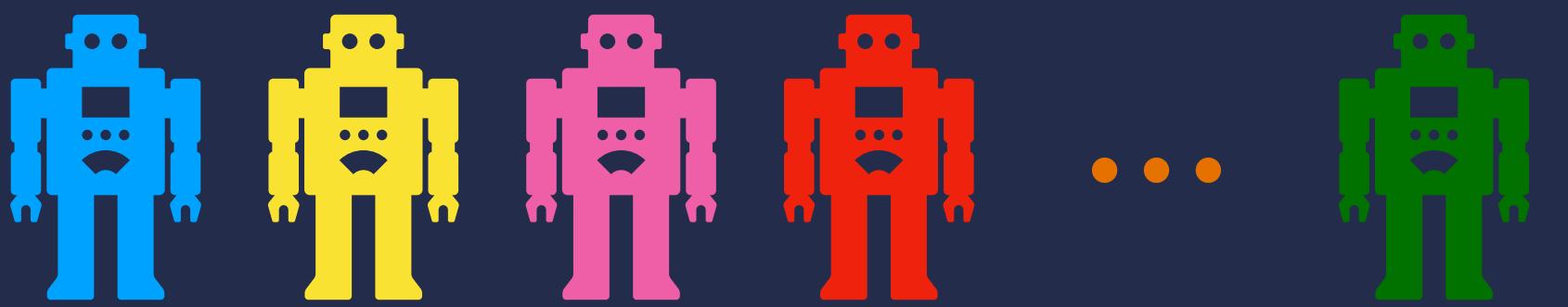
Anki Cosmo



Duckytown

Challenges

1. Multidisciplinary and rapidly expanding field
2. How to distribute the emphasis between robotics and software engineering
3. No available integrated platform
4. Robotics courses can require significant upfront investment in equipment



Principles

	Principle
P1	Prioritize the challenges of robotics that are unique from other CS systems
P2	Focus on the unique software engineering techniques and practices required by robot system development
P3	Provide opportunities for experiential learning to encourage students to practice and reflect on their experience
P4	Lower adoption barriers by making the material more accessible
P5	Reinforce foundational material across both SE and robotics

Principles

Multidisciplinary and rapidly expanding field

	Principle
P1	Prioritize the challenges of robotics that are unique from other CS systems
P2	Focus on the unique software engineering techniques and practices required by robot system development
P3	Provide opportunities for experiential learning to encourage students to practice and reflect on their experience
P4	Lower adoption barriers by making the material more accessible
P5	Reinforce foundational material across both SE and robotics

Principles

How to distribute the emphasis between robotics and software engineering

	Principle
P1	Prioritize the challenges of robotics that are unique from other CS systems
P2	Focus on the unique software engineering techniques and practices required by robot system development
P3	Provide opportunities for experiential learning to encourage students to practice and reflect on their experience
P4	Lower adoption barriers by making the material more accessible
P5	Reinforce foundational material across both SE and robotics

Principles

No available integrated platform

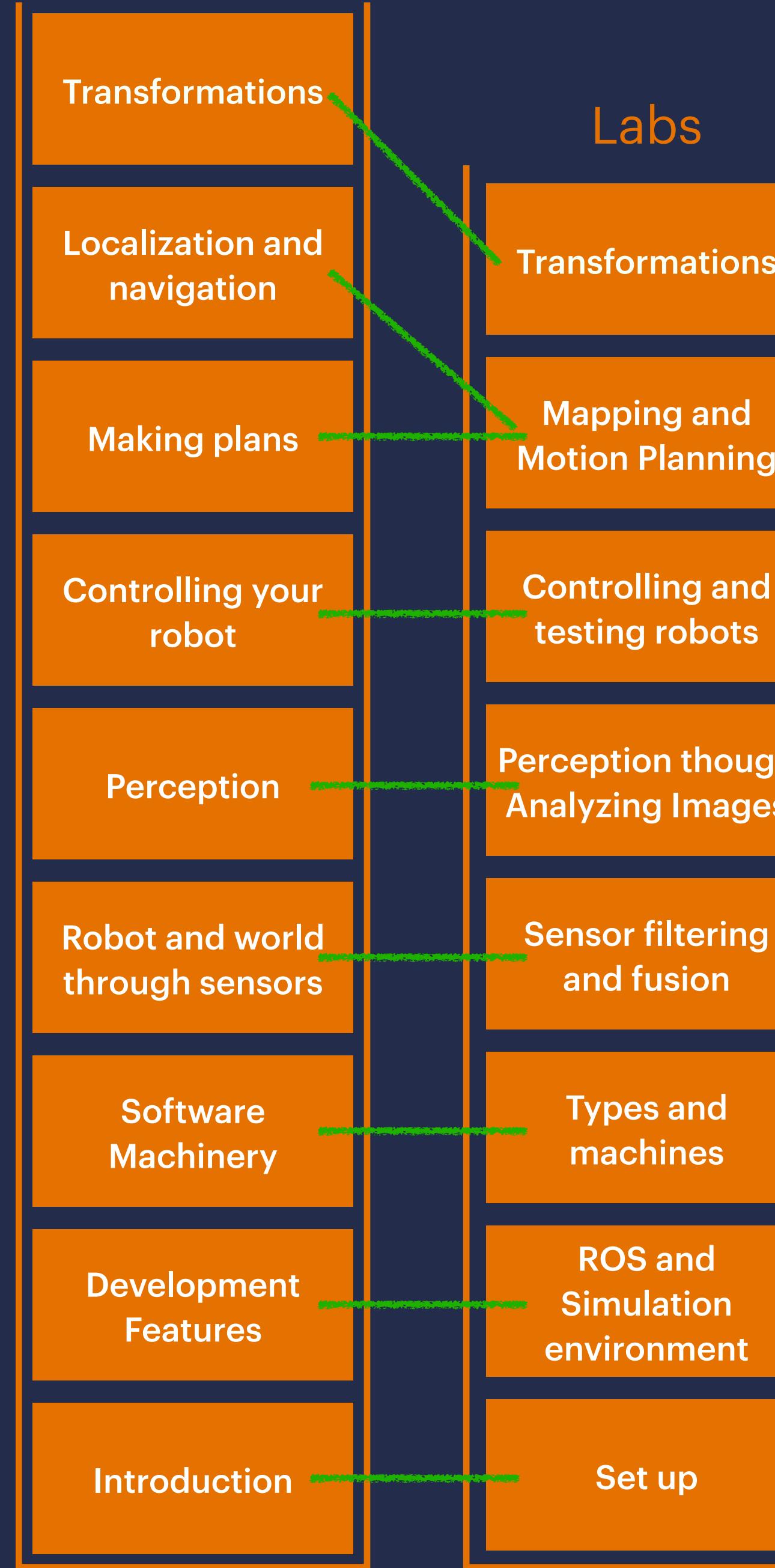
	Principle
P1	Prioritize the challenges of robotics that are unique from other CS systems
P2	Focus on the unique software engineering techniques and practices required by robot system development
P3	Provide opportunities for experiential learning to encourage students to practice and reflect on their experience
P4	Lower adoption barriers by making the material more accessible
P5	Reinforce foundational material across both SE and robotics

Principles

Robotics courses can require significant upfront investment in equipment

	Principle
P1	Prioritize the challenges of robotics that are unique from other CS systems
P2	Focus on the unique software engineering techniques and practices required by robot system development
P3	Provide opportunities for experiential learning to encourage students to practice and reflect on their experience
P4	Lower adoption barriers by making the material more accessible
P5	Reinforce foundational material across both SE and robotics

Lectures



Course Overview

We used these principles when designing the course

Principle

P1	Prioritize the challenges of robotics that are unique from other CS systems
P2	Focus on the unique software engineering techniques and practices required by robot system development
P3	Provide opportunities for experiential learning to encourage students to practice and reflect on their experience
P4	Lower adoption barriers by making the material more accessible
P5	Reinforce foundational material across both SE and robotics

Lectures

Lecture Lab Pairing

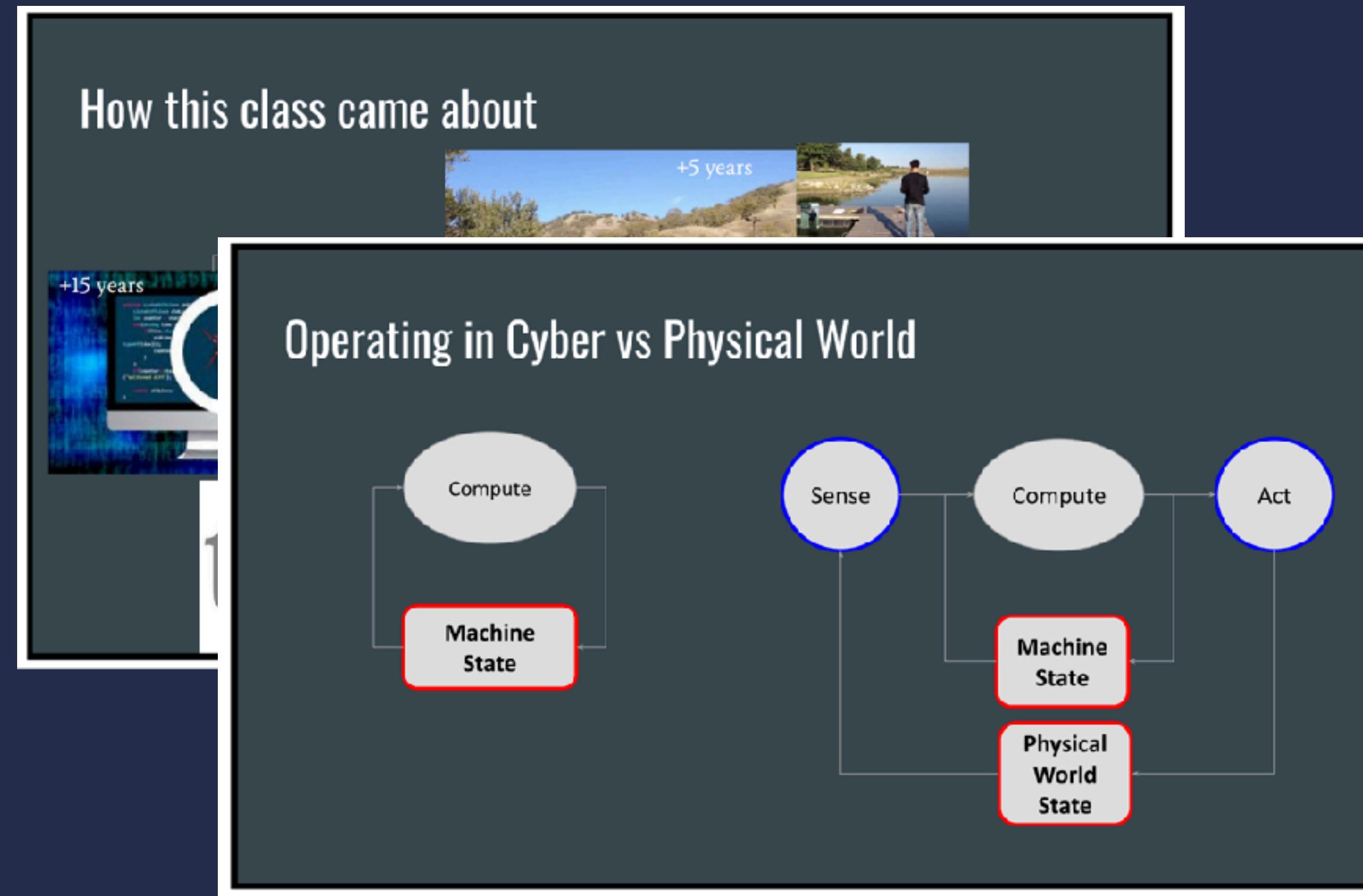
Labs

Lectures

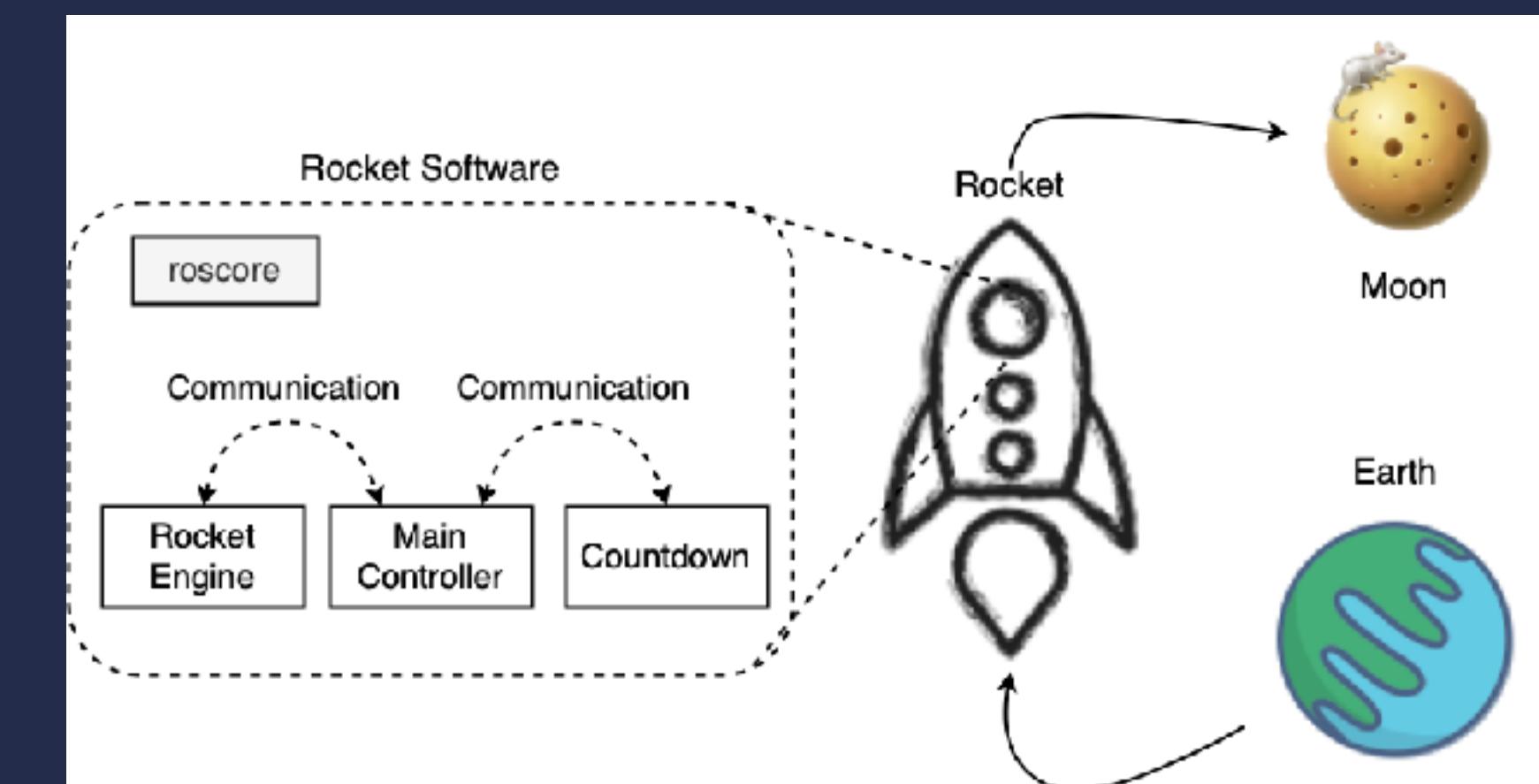
Lecture Lab Pairing

Introduction

Labs



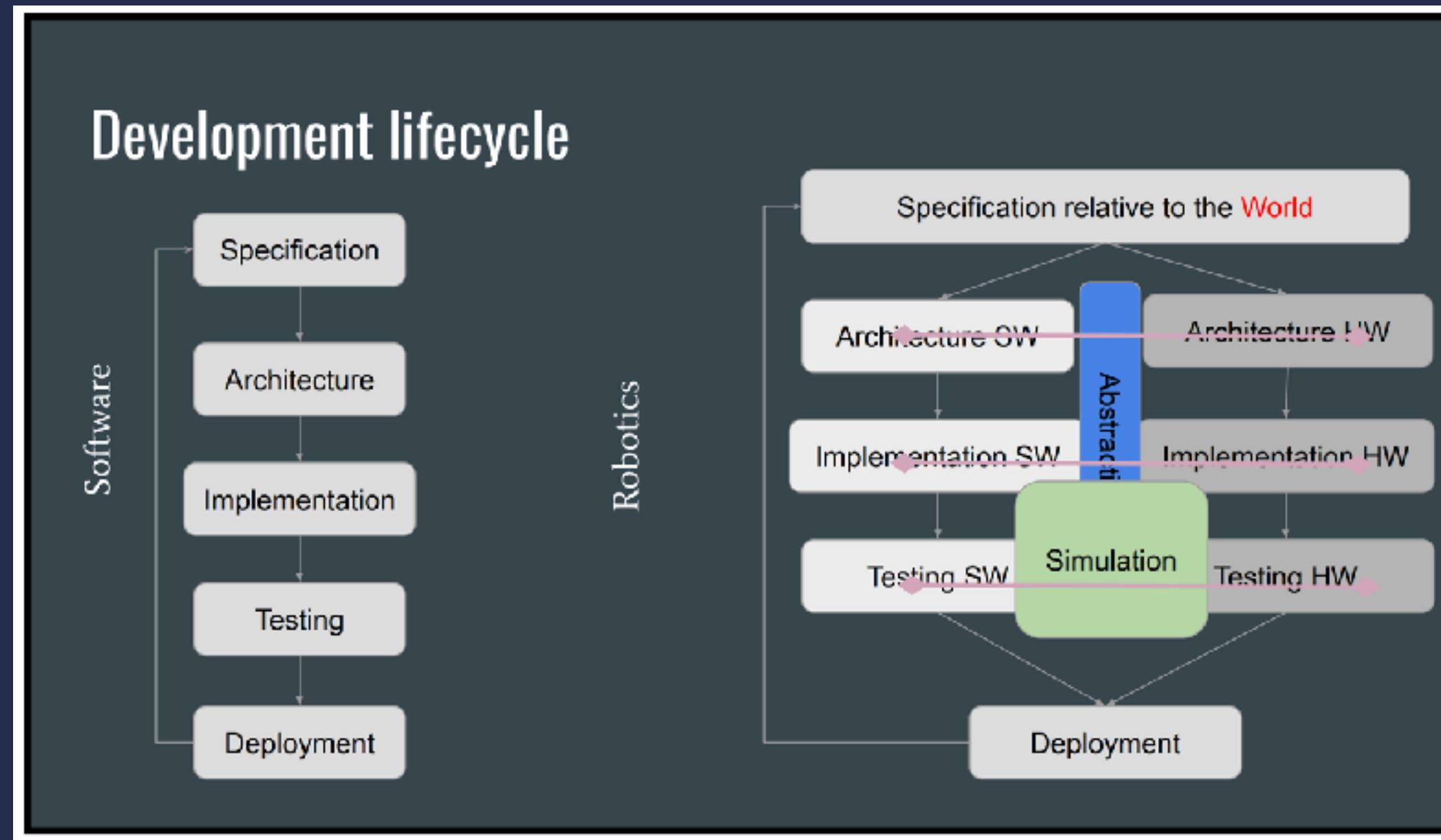
Introduction to ROS



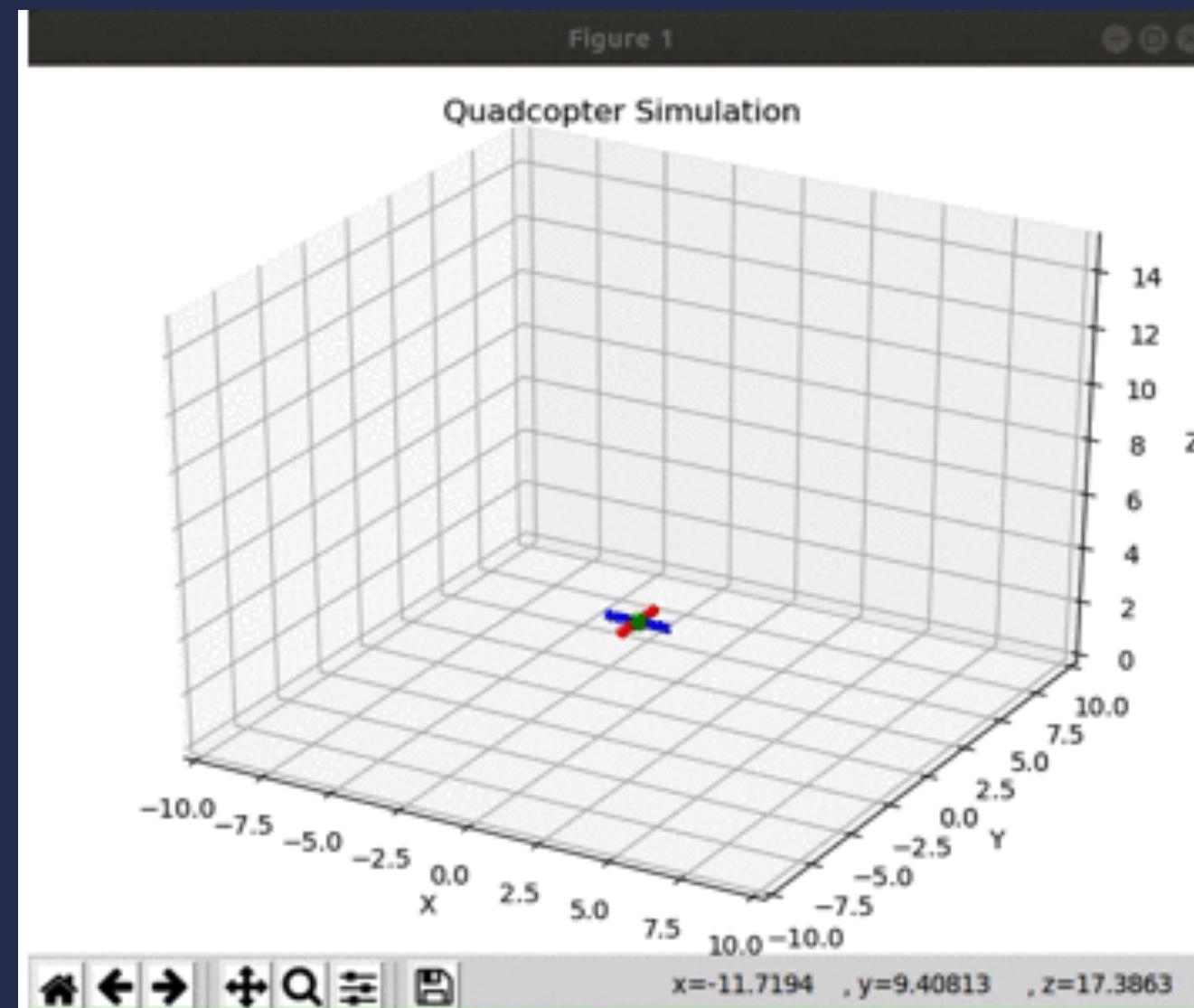
Introducing robotics

Set up

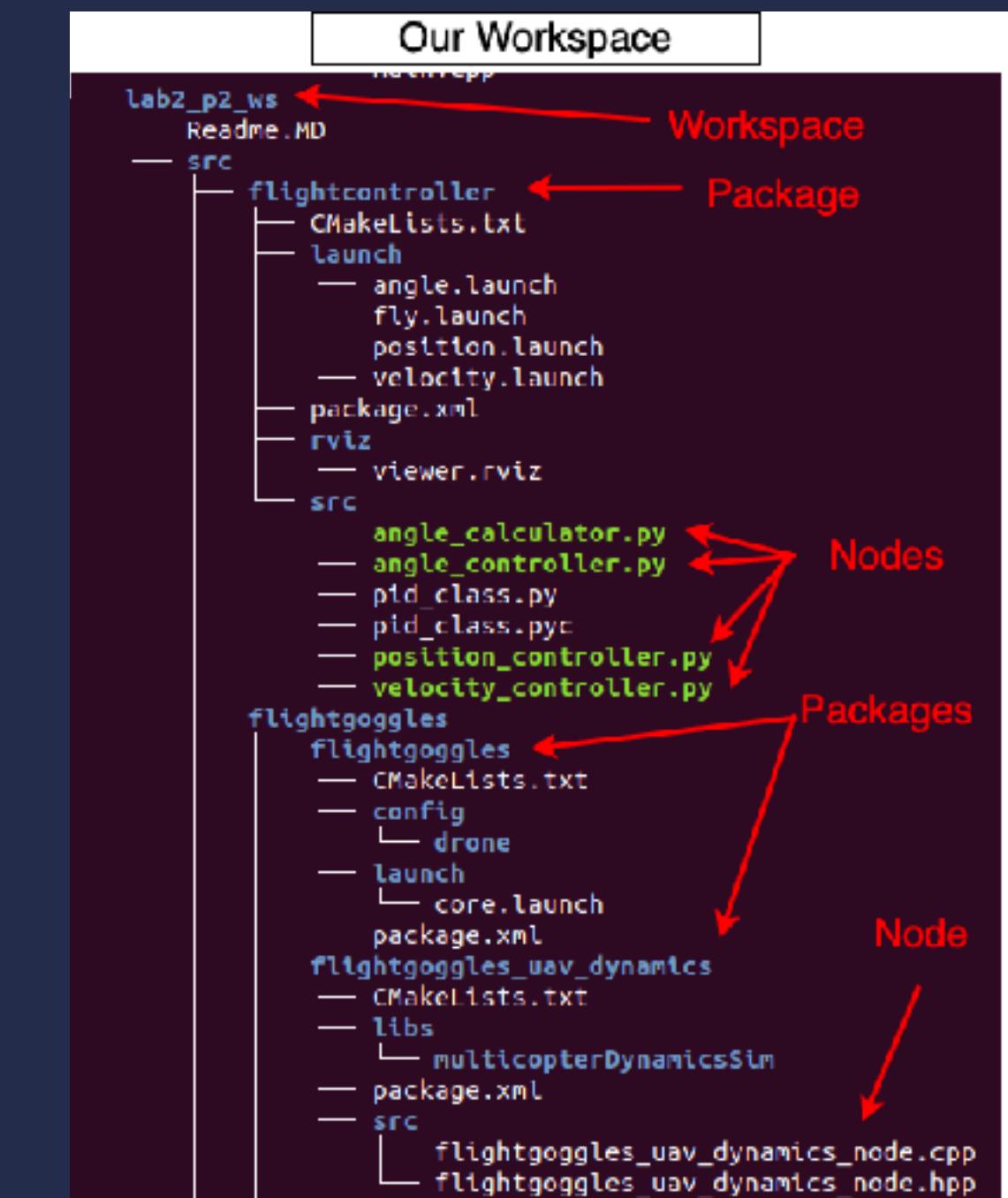
Lecture Lab Pairing



Introducing the simulation



Describing the development lifecycle



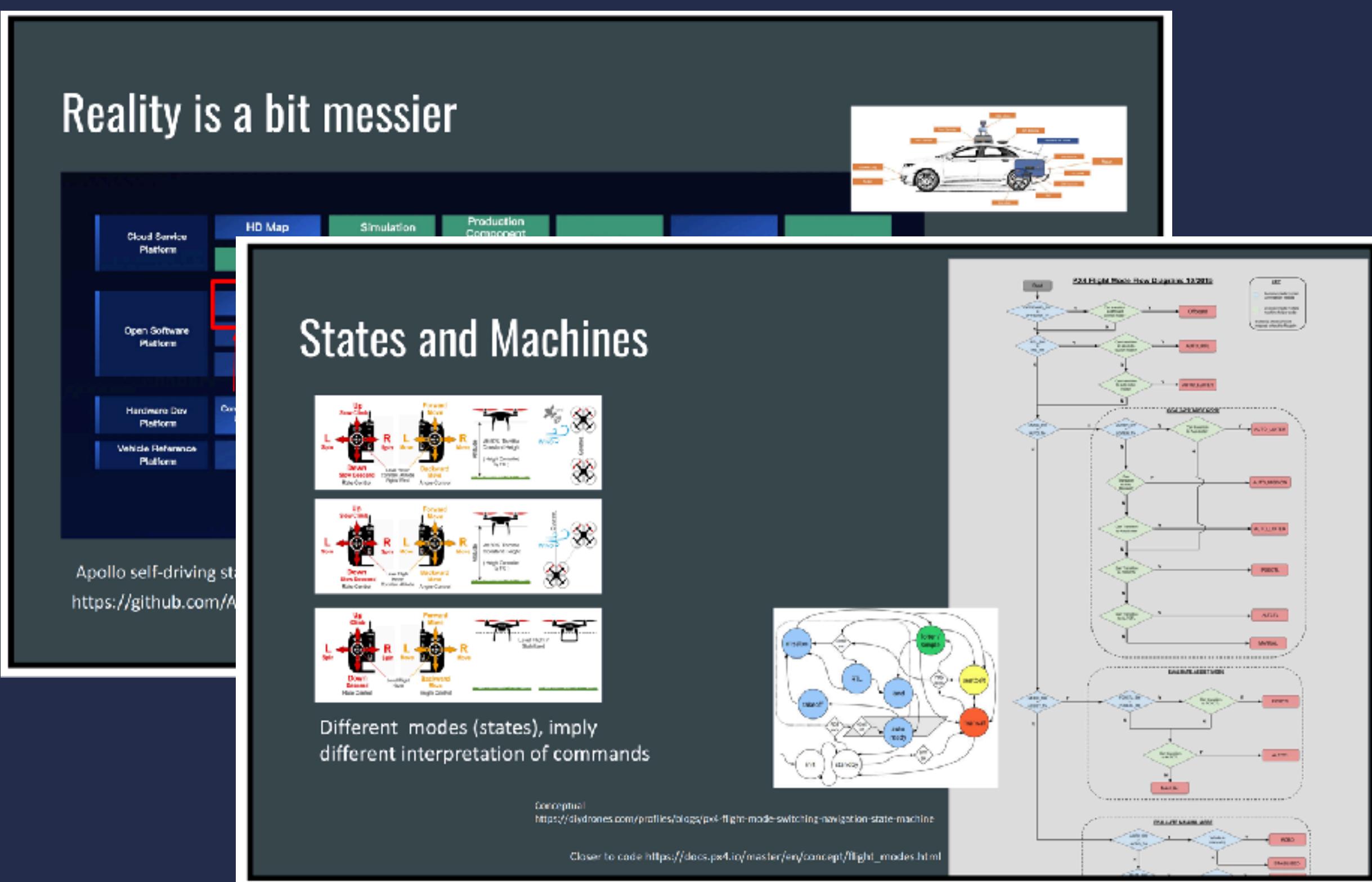
ROS and Simulation environment

Set up

Lectures

Lecture Lab Pairing

Reality is a bit messier



Describing how systems are implemented in reality

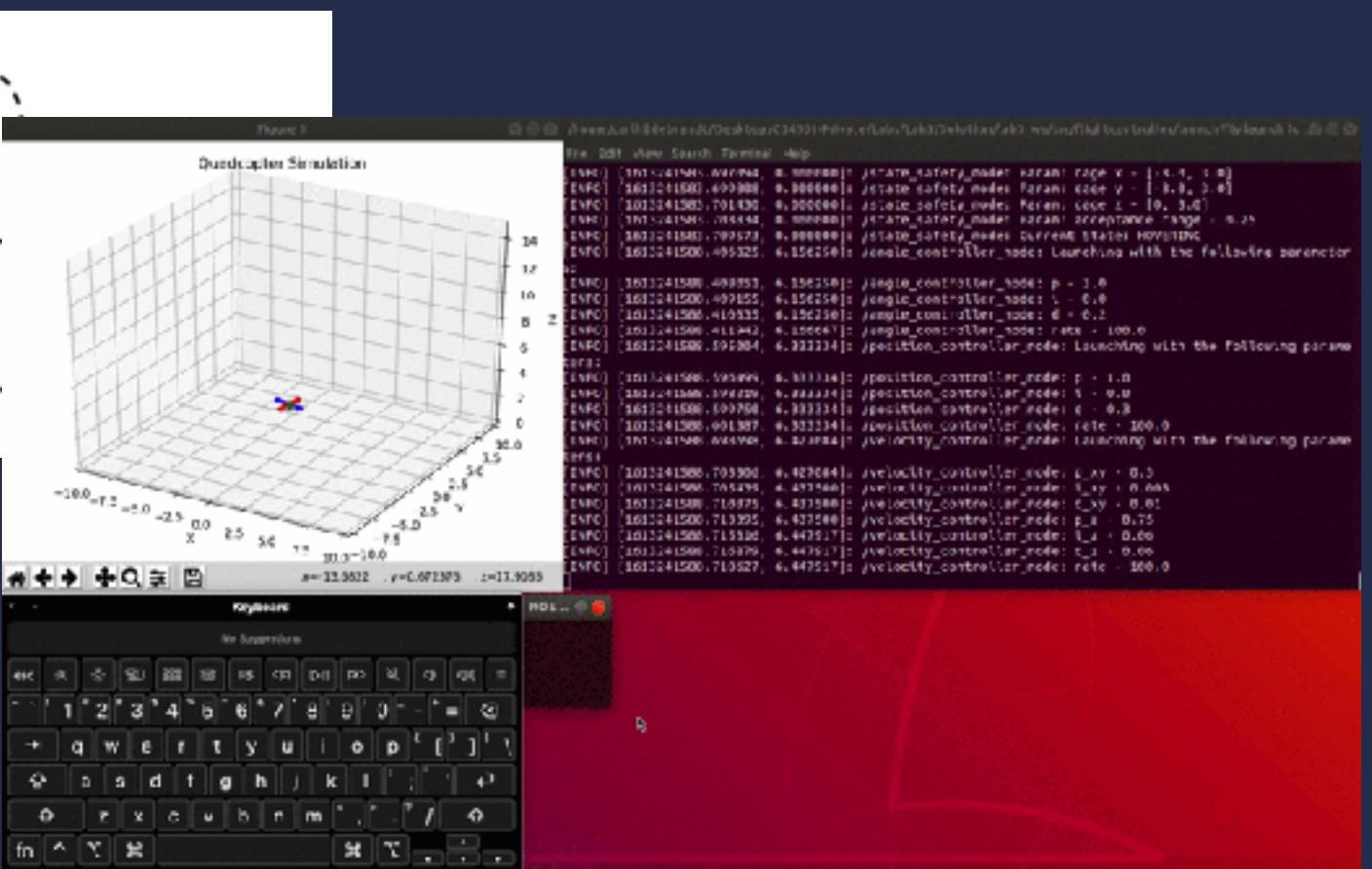
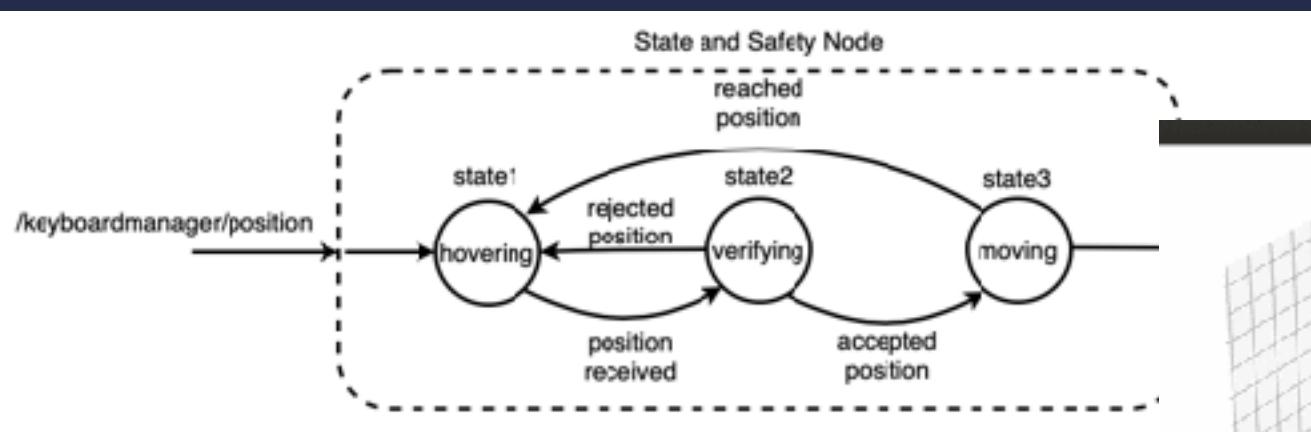
Introducing state machines

Software Machinery

Development Features

Introduction

Using state machines to create a keyboard controller



Labs

Types and machines

ROS and Simulation environment

Set up

Lecture Lab Pairing

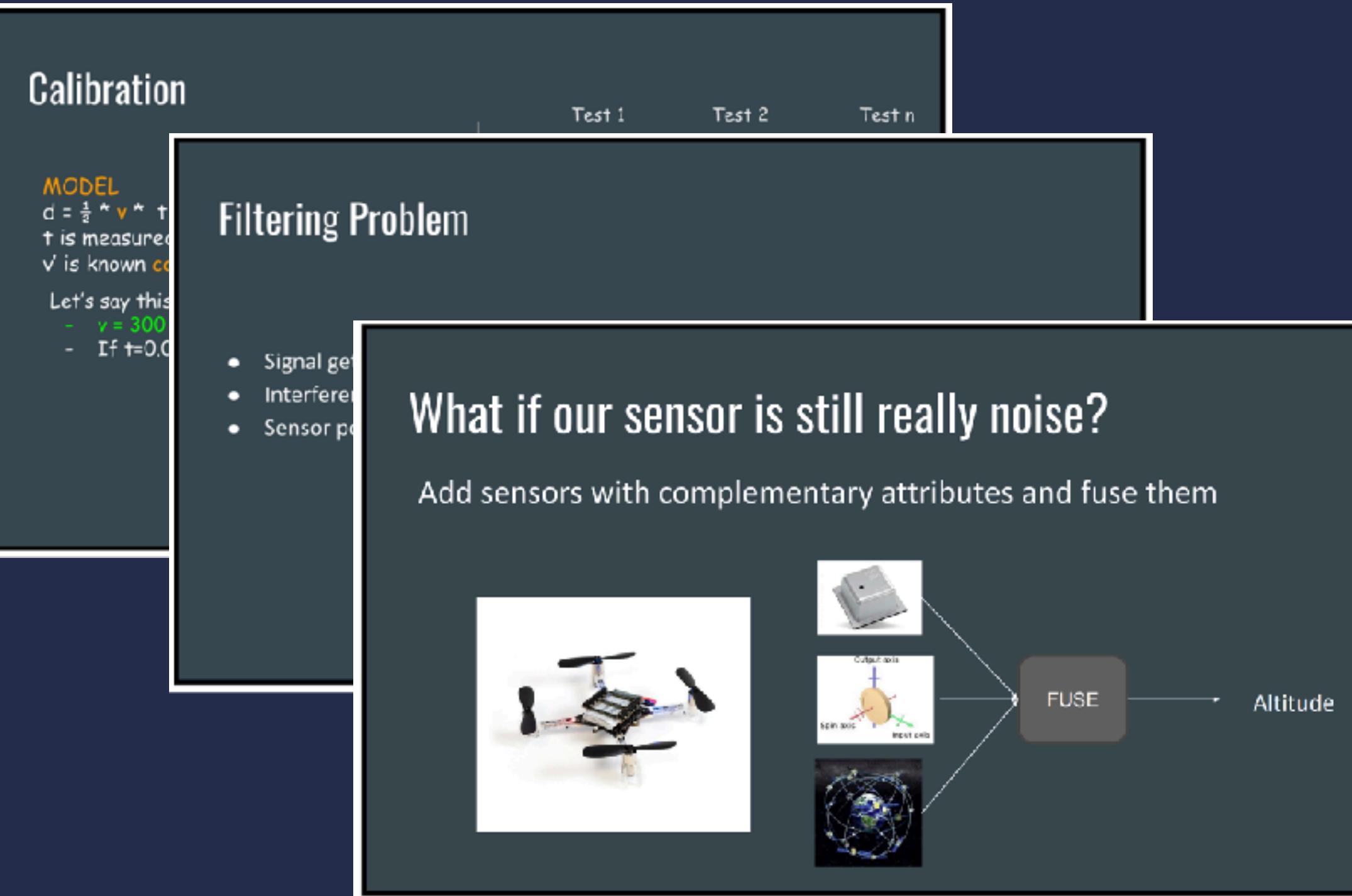
Robot and world through sensors

Software Machinery

Development Features

Introduction

Calibration



Showcasing how sensor noise affects readings and implementing filters to compensate for it

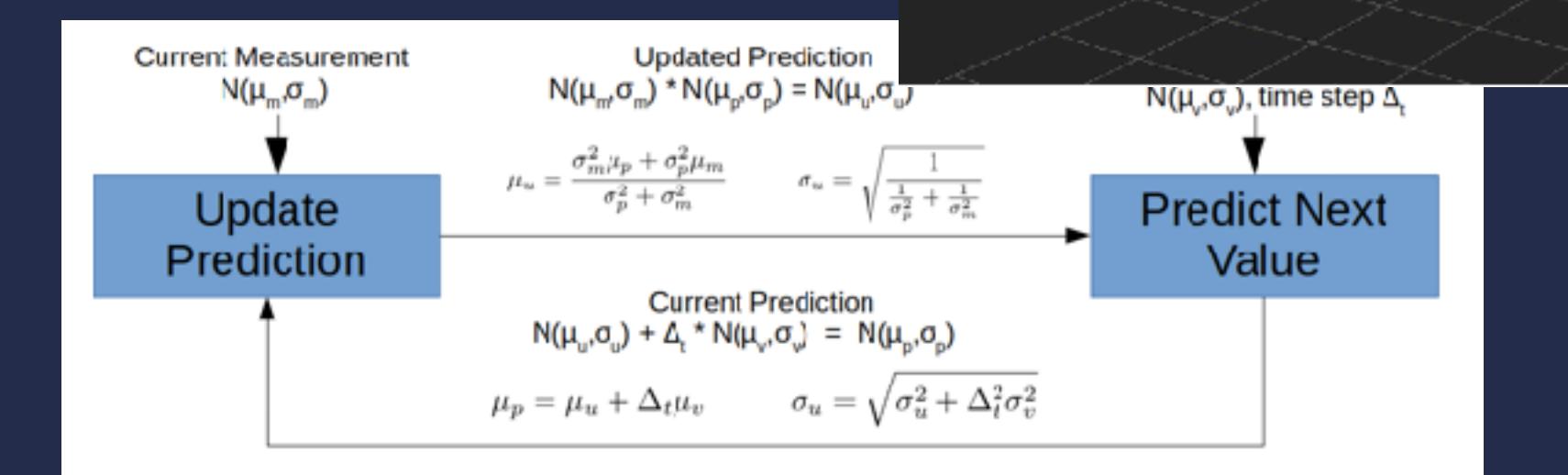
Introducing calibration, filtering, and fusion as ways to handle sensor noise

Sensor filtering and fusion

Types and machines

ROS and Simulation environment

Set up



Lecture Lab Pairing

Perception

Robot and world through sensors

Software Machinery

Development Features

Introduction

Perception Examples

How: Processing sensor data to create a higher-level abstraction of the data

Image Processing Techniques

- Traffic Light:
- Blur
- Smoothing
- Background subtraction
- Edge detection
- Color detection
- Feature extraction
- Handwritten digit recognition
- ...

Machine Learning

What happens if we have a much more complicated task?

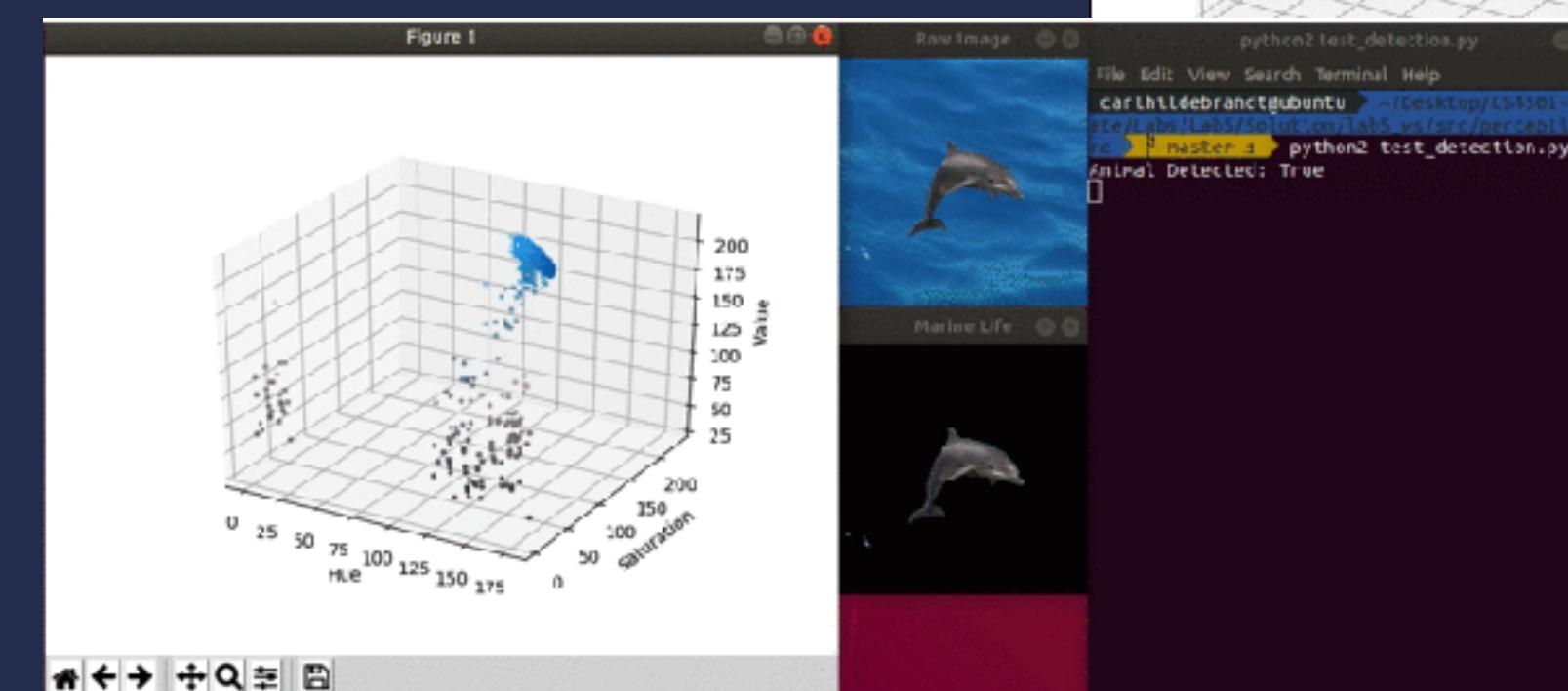
Machine learning learns this function given enough data

Input → $\text{output} = f[\text{input}]$ → Output

3779

Plane, Car, Bird, Cat

Using a down facing camera on the simulated drone to detect sea creatures in the images



Introducing different types of perception

Perception through Analyzing Images

Sensor filtering and fusion

Types and machines

ROS and Simulation environment

Set up

Lecture Lab Pairing

Controlling your robot

Perception

Robot and world through sensors

Software Machinery

Development Features

Introduction

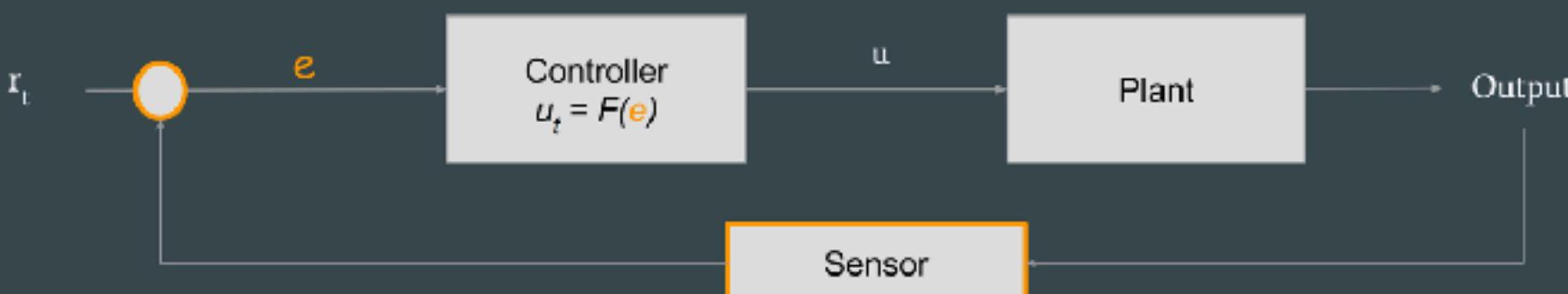
Open-loop controller

r_t

- Assumes we
- Computes u_t

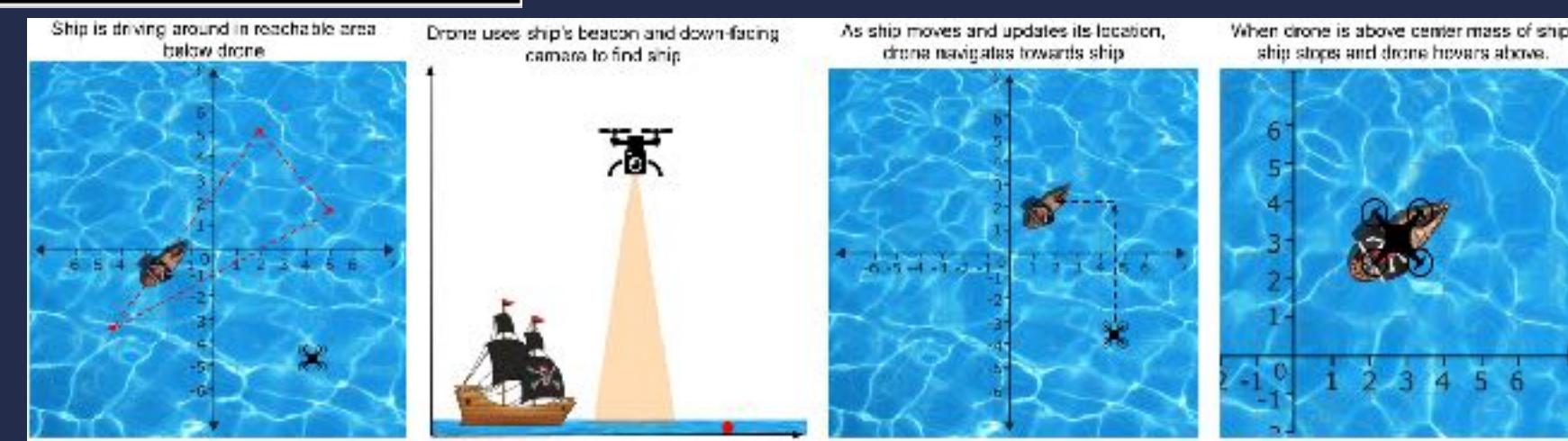
Close-Loop Controller

- Incorporates feedback to the Controller
 - Knows impacts of actions
 - Diffs setpoint and sensed output
 - Aims to make that difference zero



Using the control to implement a drone that follows a ship. Testing the that the behavior matches specifications.

Introducing different control schemes



```

class TestDroneBehavior(unittest.TestCase):

    def __init__(self, *args):
        super(TestDroneBehavior, self).__init__(*args)
        rospy.init_node("test_behavior", anonymous=True)
        # Publish the debug information
        self.debug_logger = rospy.Publisher('/test_debug', String, queue_size=1)
        # Get the test duration
        self.test_duration = rospy.get_param(rospy.get_name() + '/duration')
  
```

Controlling and testing robots

Perception through Analyzing Images

Sensor filtering and fusion

Types and machines

ROS and Simulation environment

Set up

Lectures

Localization and navigation

Making plans

Controlling your robot

Perception

Robot and world through sensors

Software Machinery

Development Features

Introduction

Lecture Lab Pairing

Motion Planning Problem

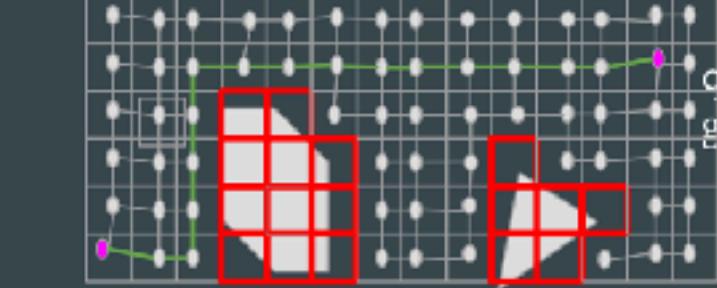
Key data structures in ROS for motion

Model-based Approaches Produced a Graph

Path Planning: Visibility Methods



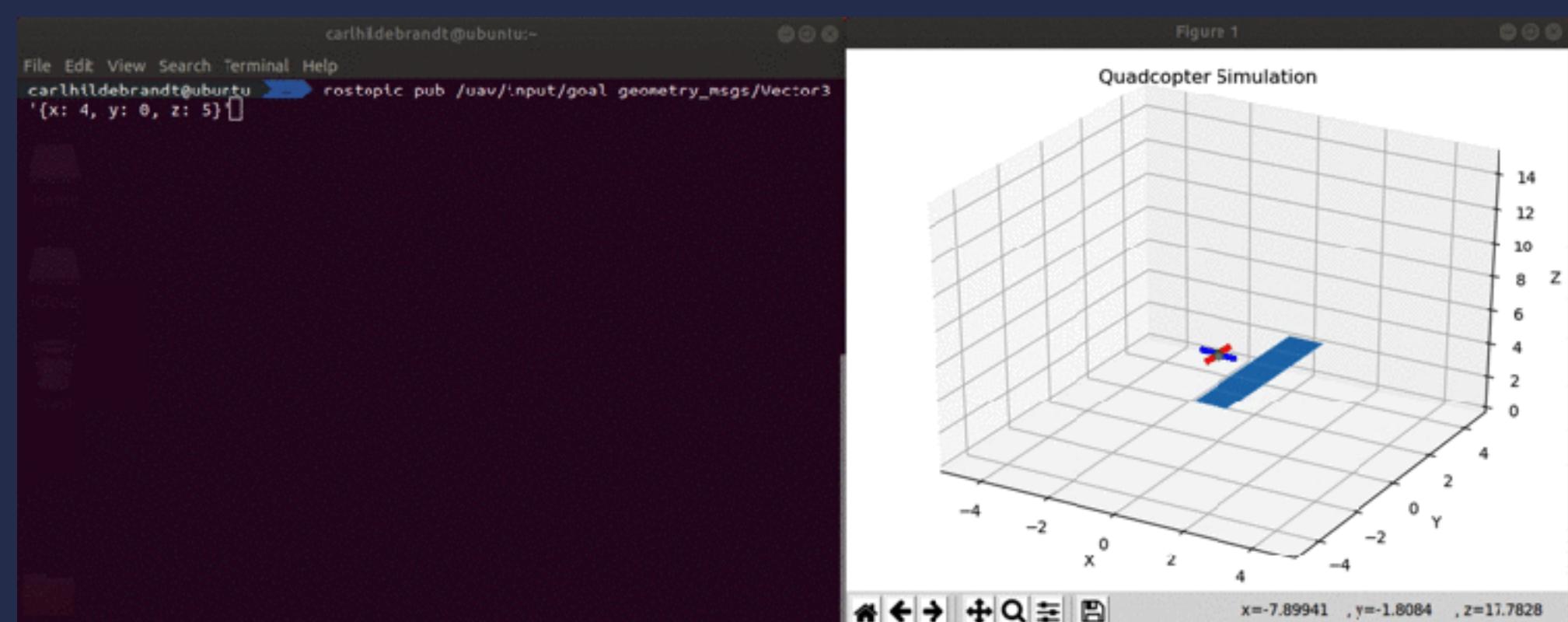
Path Planning: Grid Methods



Path Planning: Probabilistic Roadmap



Introducing motion planning and data structures used to by software engineers for creating plans



Implementing mapping and path planning in simulation

Labs

Mapping and Motion Planning

Controlling and testing robots

Perception though Analyzing Images

Sensor filtering and fusion

Types and machines

ROS and Simulation environment

Set up

Lectures

Transformations

Localization and navigation

Making plans

Controlling your robot

Perception

Robot and world through sensors

Software Machinery

Development Features

Introduction

Lecture Lab Pairing

2D Transform - rotation

Where is P in Q?

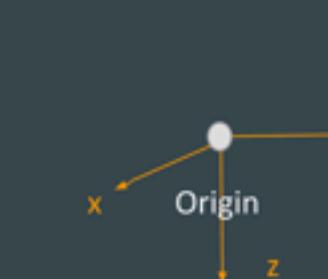
Multiple Coordinate Systems

- 3D World reference frames
- Multiple conventions

ENU - East, North, UP

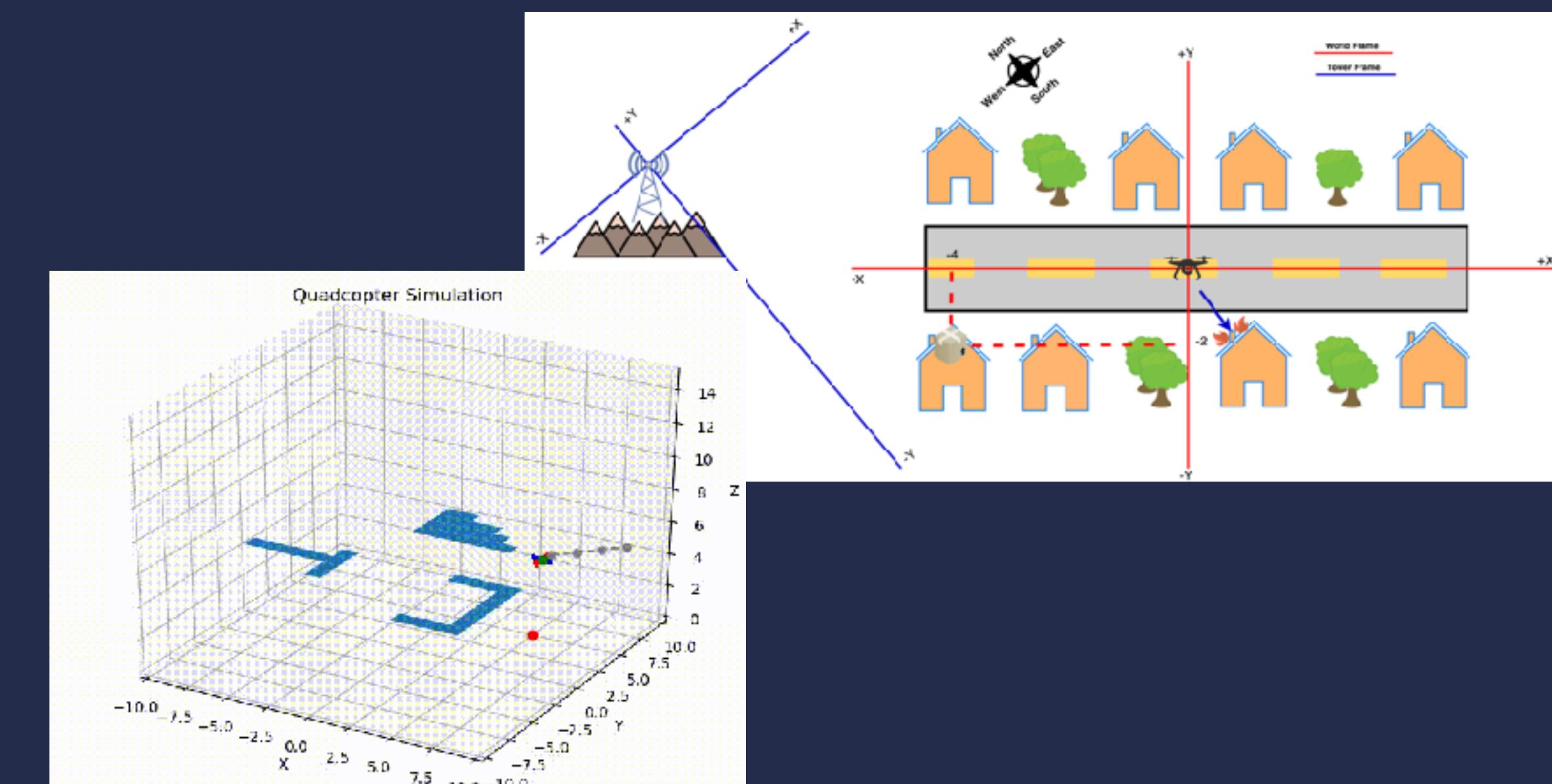


NED - North, East, Down



Introducing the coordinate systems and the math behind the transformations

Using transformations to track a ground robot transmitting in a different coordinate system



Labs

Transformations

Mapping and Motion Planning

Controlling and testing robots

Perception through Analyzing Images

Sensor filtering and fusion

Types and machines

ROS and Simulation environment

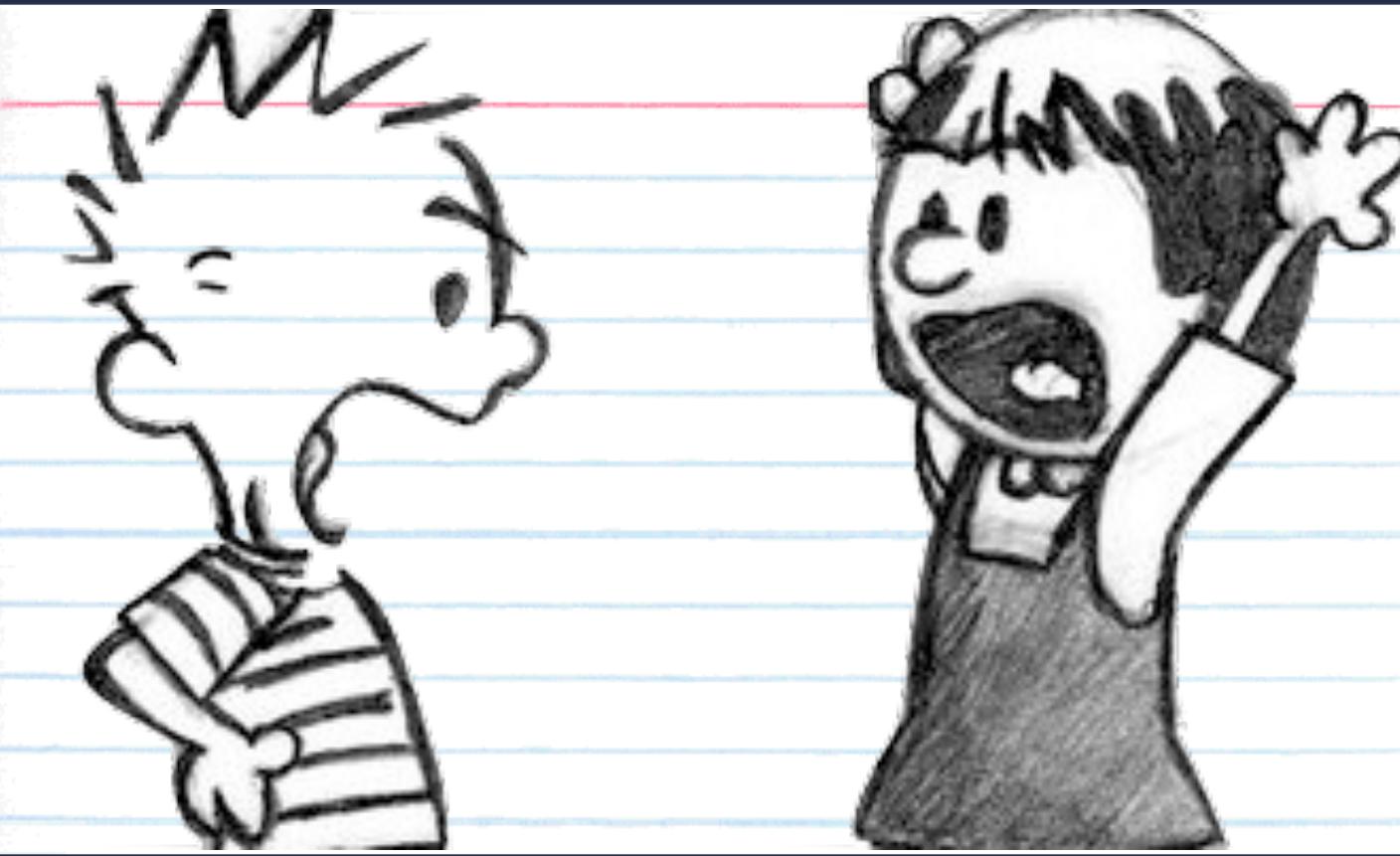
Set up

Crosscutting Issues

Industry perspective: Guest Speaker



Ethics Lab



Allowed students to interact and ask questions about what the issues are in industry, and how what they are learning will be applied in the real world

Allowed students to debate ethical issues that will arise as robotics becomes more apparent in all our lives. No right or wrong answers, we just wanted to allow students to start thinking about these issues.

Innovations

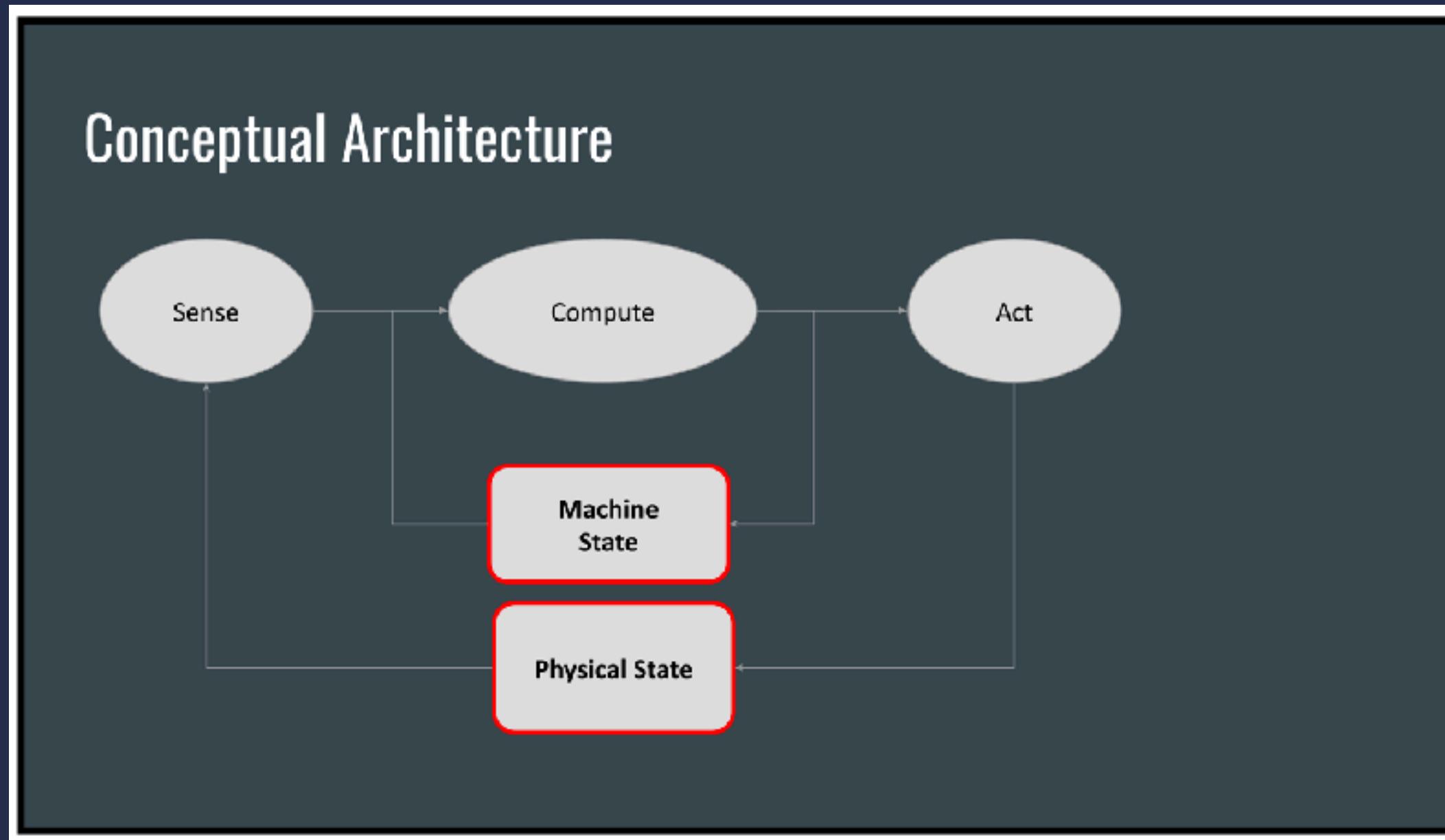
We apply these principles to bring **several innovations** to this course

Innovations

We apply these principles to bring several innovations to this course

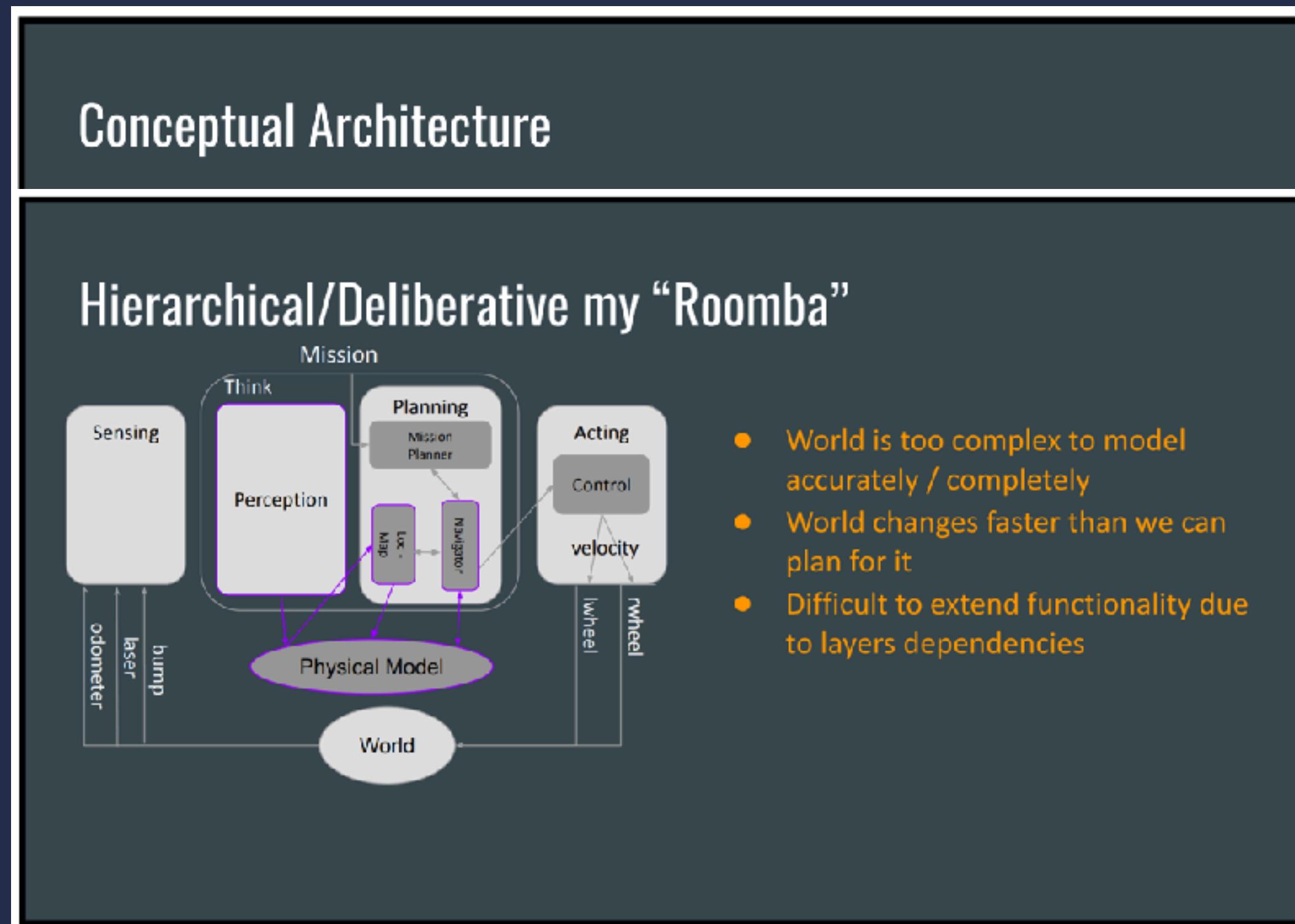
	Innovation	P1	P2	P3	P4	P5
I1	Cover robotics fundamentals	✓	✓			
I2	Offer different levels of abstraction	✓			✓	
I3	Pair SE and Robotics topics throughout the course	✓	✓			
I4	Enable students to make design and implementation decisions in labs and project			✓		✓
I5	Make use of demonstration, conversation, and checkpoints			✓		✓
I6	Use a drone simulator for hands-on experience			✓	✓	✓
I7	Incrementally build concepts to minimize required background knowledge in robotics			✓	✓	✓
I8	Incorporate flexibility into the course schedule and allow for self-paced labs			✓	✓	

Example Lecture



Aim: introduce the fundamental concepts related to robotics architecture and modeling machinery in robotics

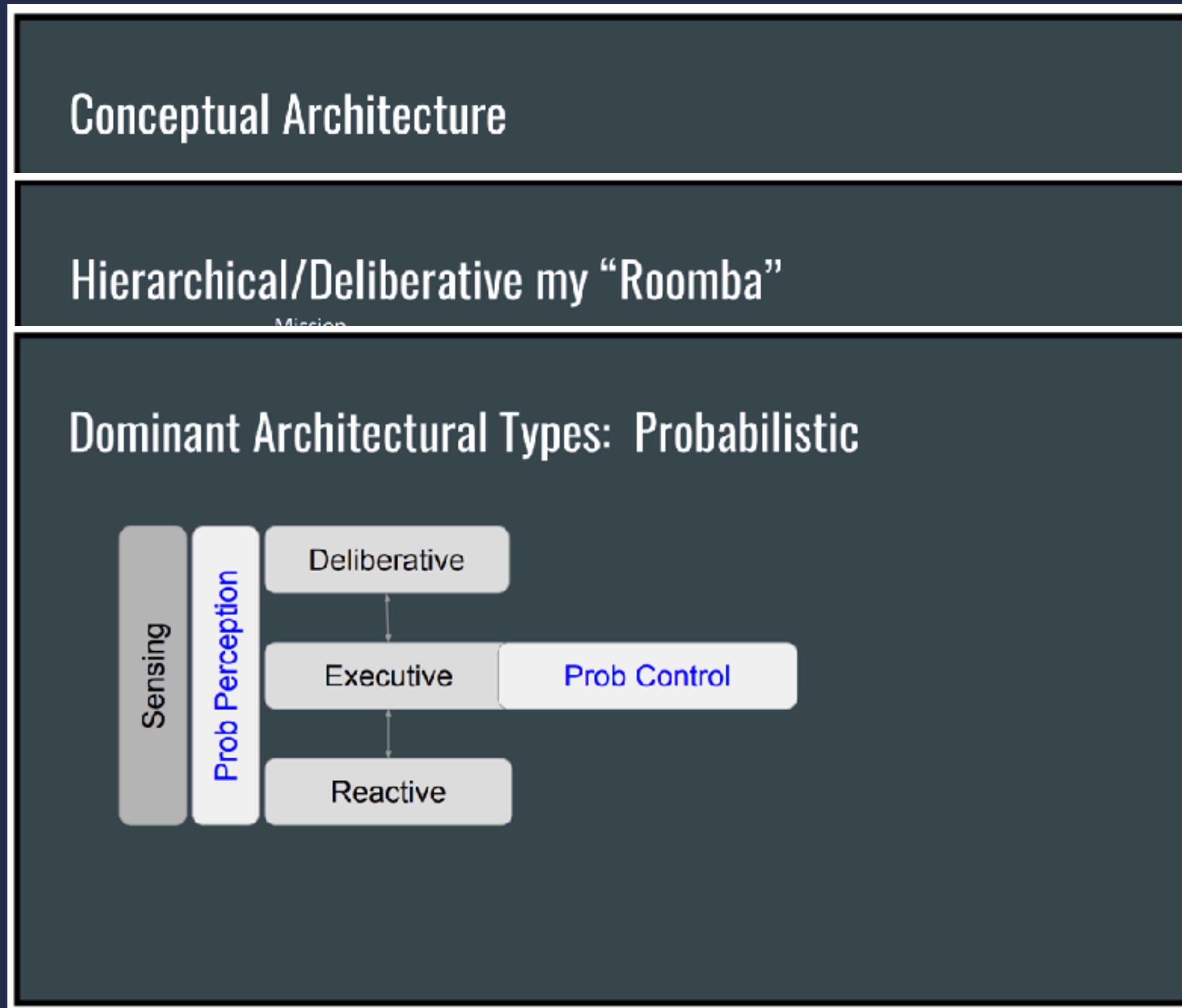
Example Lecture



Aim: introduce the fundamental concepts related to robotics architecture and modeling machinery in robotics

← Begins with the basic conceptual architecture of robotics
(I1 - Cover robotics fundamentals)

Example Lecture



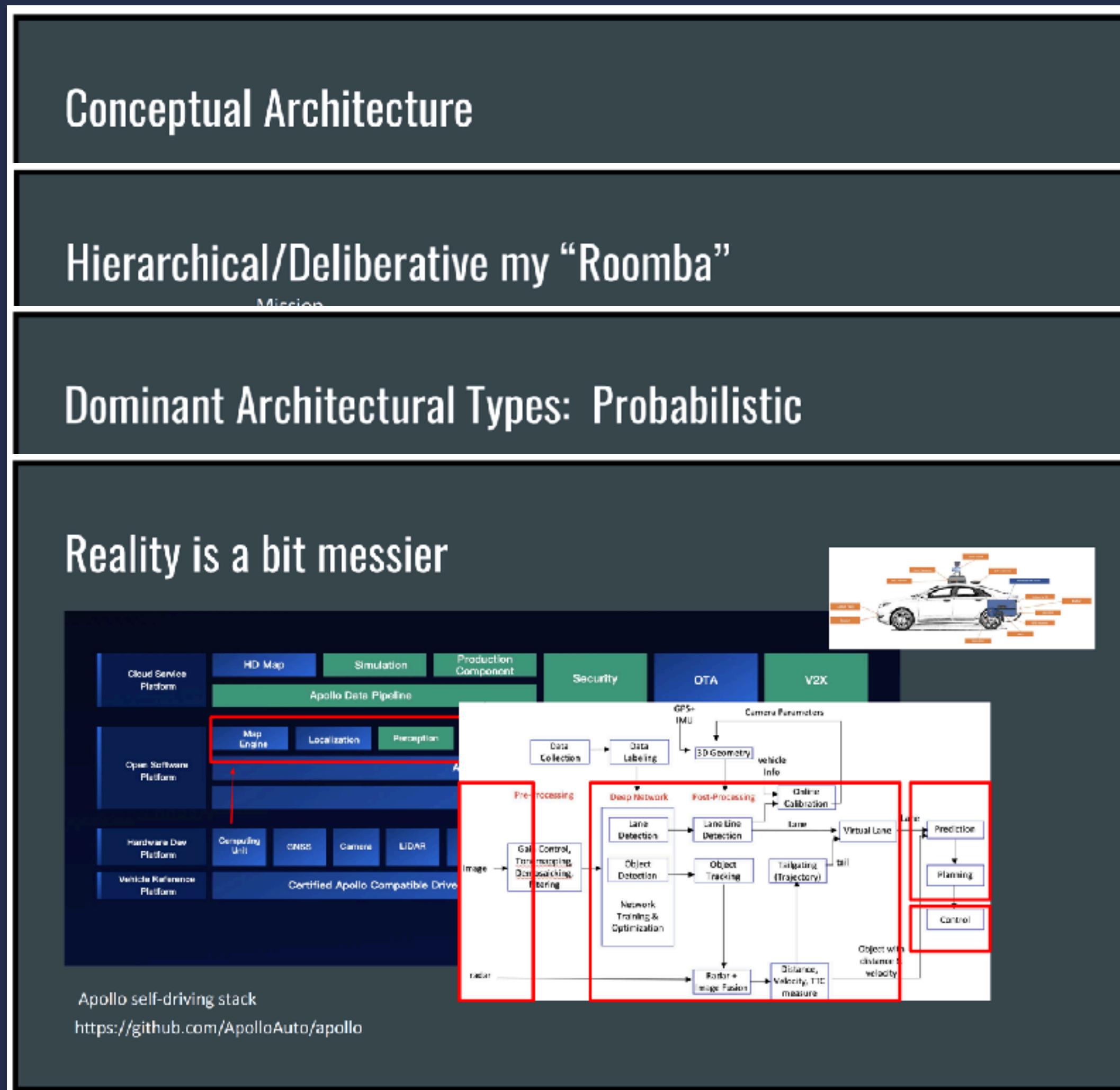
Aim: introduce the fundamental concepts related to robotics architecture and modeling machinery in robotics

Begins with the basic conceptual architecture of robotics
(I1 - Cover robotics fundamentals)



Covers critical domain-specific architectures
(I7 - Incrementally build concepts)

Example Lecture



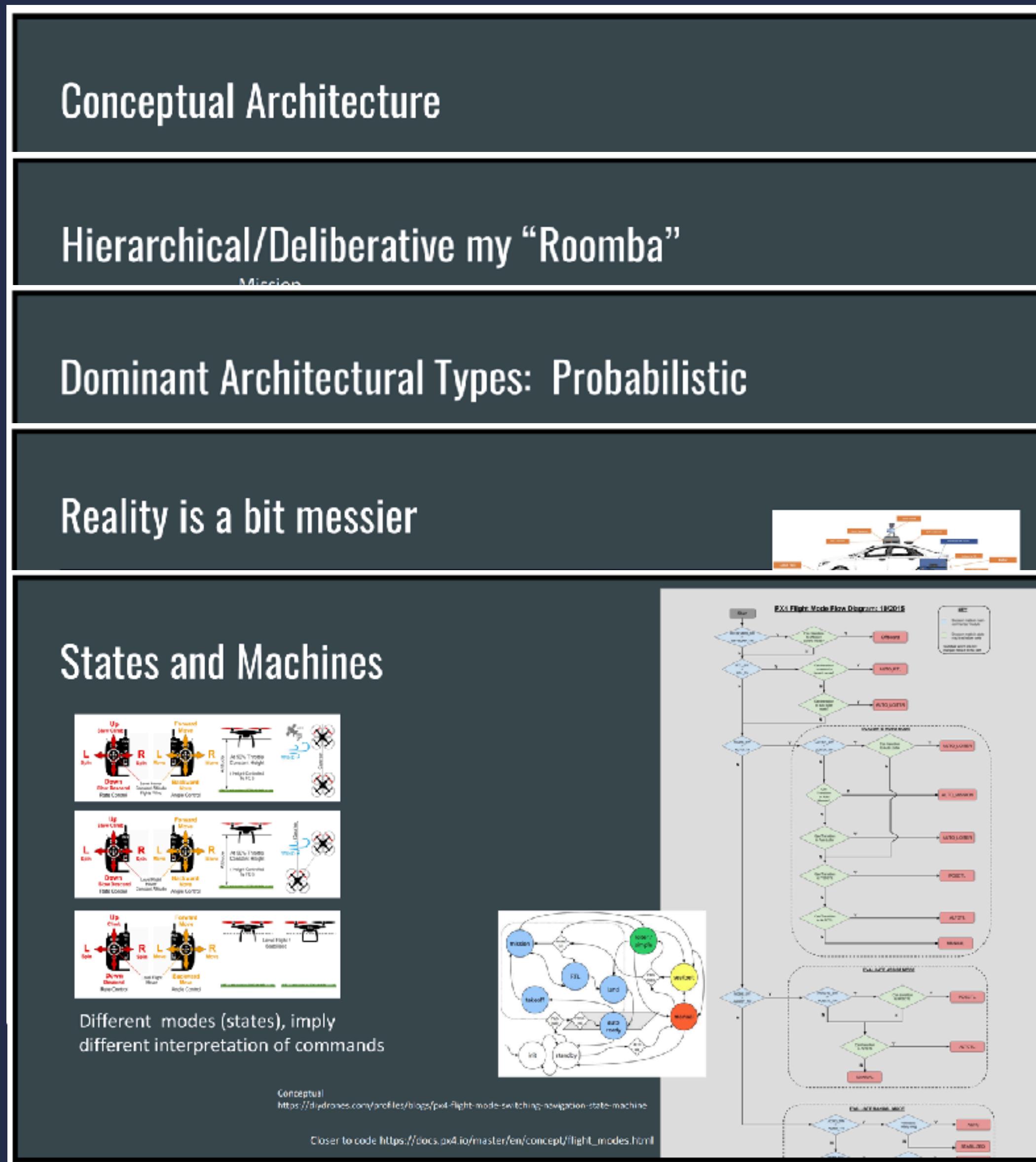
Aim: introduce the fundamental concepts related to robotics architecture and modeling machinery in robotics

Begins with the basic conceptual architecture of robotics
(I1 - Cover robotics fundamentals)

Covers critical domain-specific architectures
(I7 - Incrementally build concepts)

← Discuss design tradeoffs over different scenarios
(I5 - Demonstration, conversation, and checkpoints)

Example Lecture



Aim: introduce the fundamental concepts related to robotics architecture and modeling machinery in robotics

Begins with the basic conceptual architecture of robotics
(I1 - Cover robotics fundamentals)

Covers critical domain-specific architectures
(I7 - Incrementally build concepts)

Discuss design tradeoffs over different scenarios
(I5 - Demonstration, conversation, and checkpoints)

Introduce FSMs

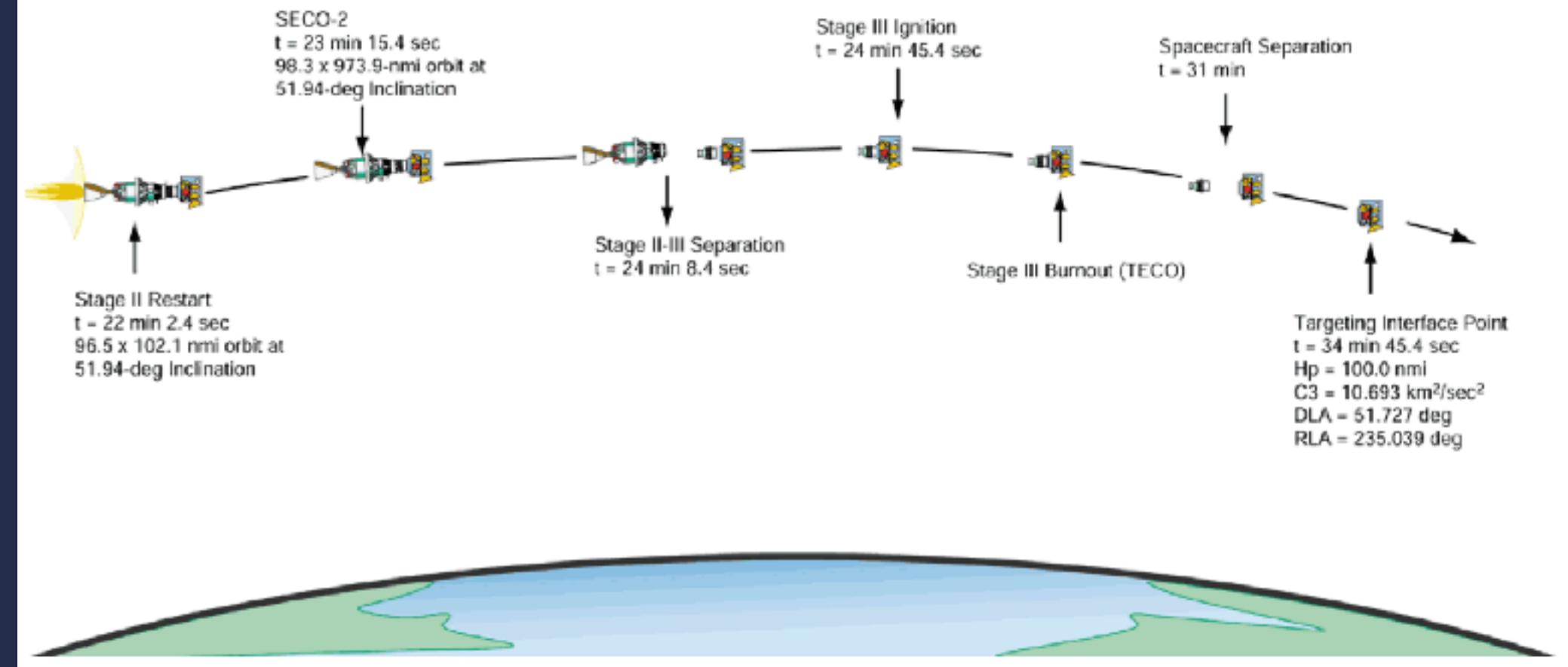
- what types of states they can encode
 - how they can assist in understanding the real world
 - how they are represented in code
 - how to scale them up to develop robotic systems
- (I3 - Pair SE and Robotics topic)

Example Lab

Abstractions to manage complexity

In this lab, we will work on three types of abstractions that we use in robotics to help us manage system complexity. First, we will keep working on separating functionality into ROS nodes. We will then further separate code within a node based on a system's natural discrete states. Such discrete states are commonly found in robotics and often managed through finite state machines. For instance, the [2001 Mars Odyssey](#) spacecraft, NASA's longest-lasting spacecraft at Mars, consists of multiple stages (states). At each of these states, the spaceship is performing unique functionality, and under certain events, it will transition from one state to the next.

Second, we will work on generalizing the applicability of robot systems by parameterizing their functionality. Abstracting parameters from the code and placing them in a more accessible place is common in software engineering. By making parameters configurable during deployment, the system functionality can be tailored without modifying the code. Last, we will work on abstracting functionality that must be provided synchronously by defining our own services.



Starts highlighting how what is learned is implemented in real systems.

Example Lab

Abstractions to manage complexity

In this lab, we will work on three types of abstractions that we use in robotics to help us manage system complexity. First, we will keep working on separating

Create the State and Safety Node

The first step in improving the drone's control software will be to create the node. We will start the implementation of this node by only considering the first objective:

- Track the mission state of the drone

To start pull the latest code inside your virtual machine.

```
# Change to lab directory  
$ cd ~/Desktop/CS4501-labs/  
# Clone the code  
$ git pull
```

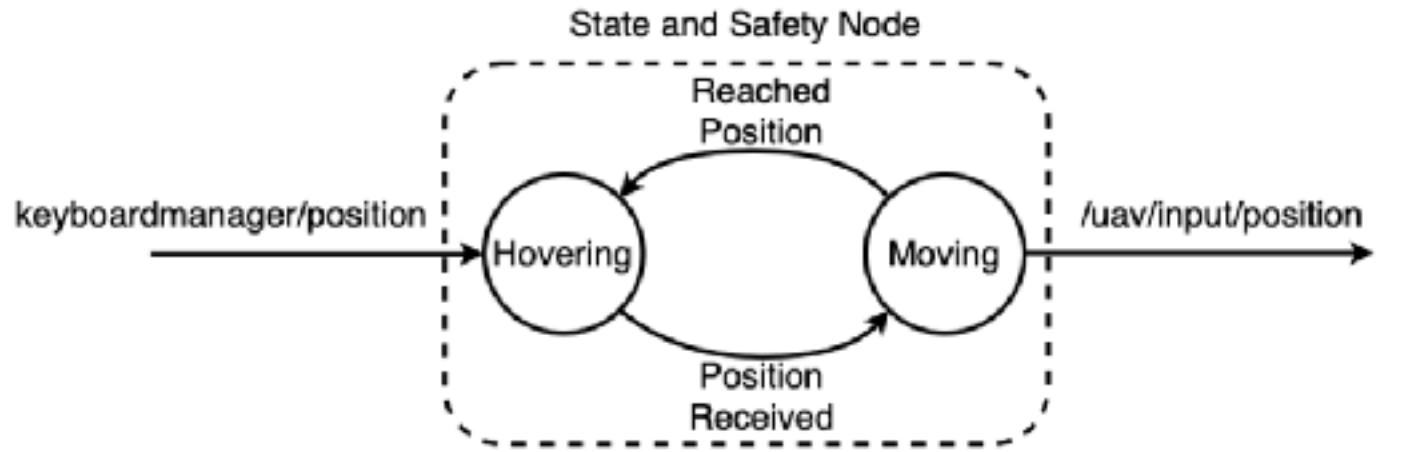
You should see a new workspace `lab3_ws`. This workspace will have the keyboard node and keyboard manager already implemented for you.

To track the drone's mission state, we are going to need to create a new node in the `simple_control` package. Create a new node in `simple_control` package in the `lab3_ws` workspace called `state_and_safety.py` using what you have learned from Lab 1 and Lab 2.

Note: remember to give the node execution permissions using the `chmod` command.

The software of a robot operation can be complex. One way to manage this complexity is to decouple the functionality based on the system discrete states and then organize the system as a Finite State Automaton (FSA). An FSA is a mathematical computation model that can be in exactly one of a finite number of states at any given time, and where the system can make well-defined transitions from one state to another in response to inputs or events. Using an FSA we will design the

`state_and_safety.py` node as follows



Starts highlighting how what is learned is implemented in real systems.

Starts with implementing a basic FSM
(I3 - Pair SE and Robotics topic)

Example Lab

Abstractions to manage complexity

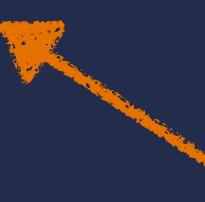
In this lab, we will work on three types of abstractions that we use in robotics to help us manage system complexity. First, we will keep working on separating

Create the State and Safety Node

The first step in improving the drone's control software will be to create the node. We will start the implementation of this node by only considering the first objective:

A useful resource to notice: ROS Logging

You will notice that this code no longer uses a standard print command. Each of the print commands has been replaced with a `rospy.loginfo(str(rospy.get_name()) + "...")` command. Using a log command in ROS is a good practice. A print command will print a message to the terminal, with no extra logging information. A `rospy.loginfo()` command will print a message to the terminal, as well as keep that message inside the ROS logs. That way, you can go back and review your robot's behavior at a later stage. We also added the following string to each log: `str(rospy.get_name())`. This is beneficial when there are more than two nodes printing messages to the terminal, as we will be able to differentiate messages from separate nodes more easily. ROS logging allows you to create levels of messages so that important messages and less important messages can be distinguished. The most important messages are called fatal messages. To publish a fatal message, you use the command `rospy.logfatal()`. The lowest level of a message is a debug message which can be logged using `rospy.Logdebug()`. More on ROS logging can be found on the [ROS Wiki](#).



Starts highlighting how what is learned is implemented in real systems.

Starts with implementing a basic FSM
(I3 - Pair SE and Robotics topic)

Highlight how ROS allows for looking information allowing for easy debugging
(I1 - Cover robotics fundamentals)

Example Lab

Abstractions to manage complexity

In this lab, we will work on three types of abstractions that we use in robotics to help us manage system complexity. First, we will keep working on separating

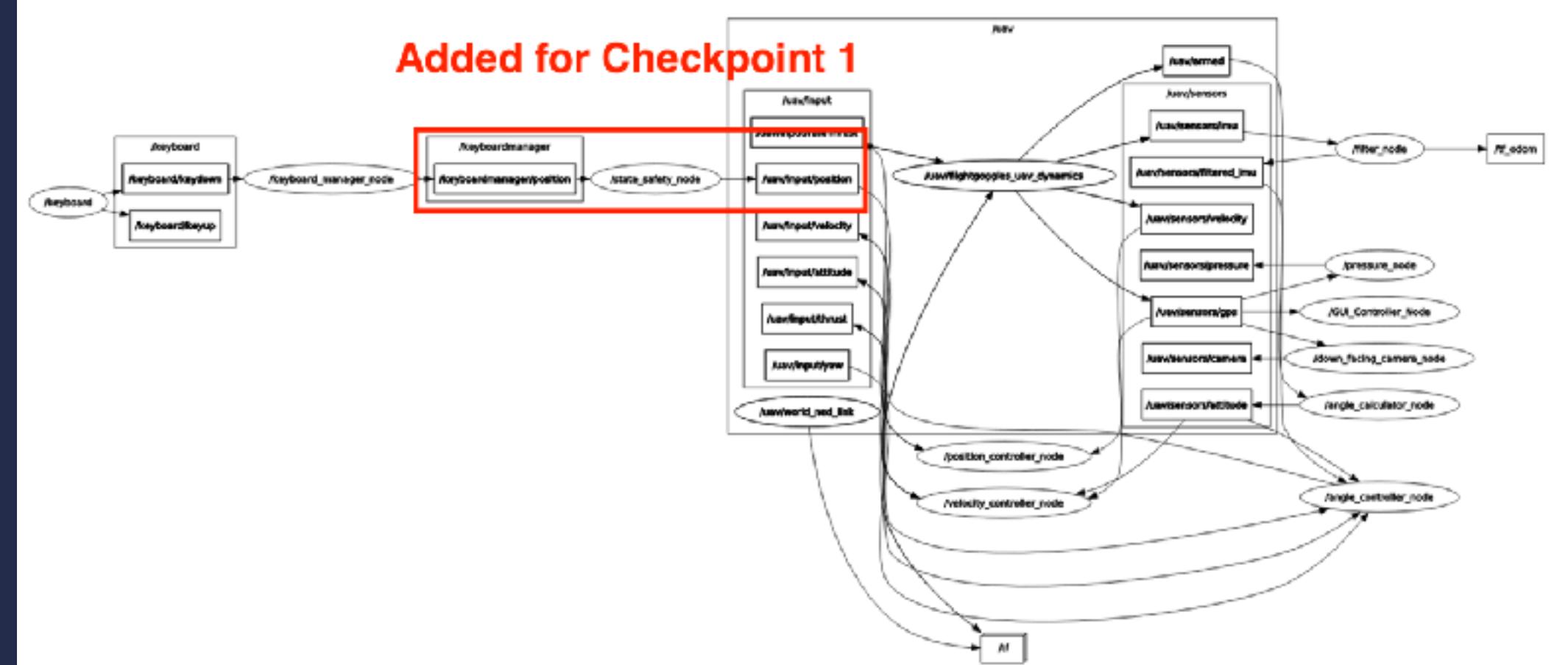
Create the State and Safety Node

The first step in improving the drone's control software will be to create the node. We will start the implementation of this node by only considering the first objective:

A useful resource to notice: ROS Logging

Checkpoint 1

Launch the simulator and check that you have correctly created the state and safety node as well as changed the keyboard manager to publish the correct data. Your ROS computation graph should look like the one below. Take a screenshot of the ROS computation graph:



Next, try to fly the quadcopter using the keyboard. Change the requested position using the keyboard keys. Once you have selected a position, hit the ENTER key to move the drone.

1. What happens when you hit the enter key? Answer this in terms of the FSA states we implemented

2. What happens when you request the drone to fly to a second position? Answer this in terms of the actual code used in `state_and_safety.py`.

Starts highlighting how what is learned is implemented in real systems.

Starts with implementing a basic FSM
(I3 - Pair SE and Robotics topic)

Highlight how ROS allows for looking information allowing for easy debugging
(I1 - Cover robotics fundamentals)

Checkpoint allowing reflection and discussion while allowing freedom to implement their own solution
(I4 - Students make design and implementation decisions)
(I5 - Demonstration, conversation, and checkpoints)

Example Lab

Abstractions to manage complexity

In this lab, we will work on three types of abstractions that we use in robotics to help us manage system complexity. First, we will keep working on separating

Create the State and Safety Node

The first step in improving the drone's control software will be to create the node. We will start the implementation of this node by only considering the first objective:

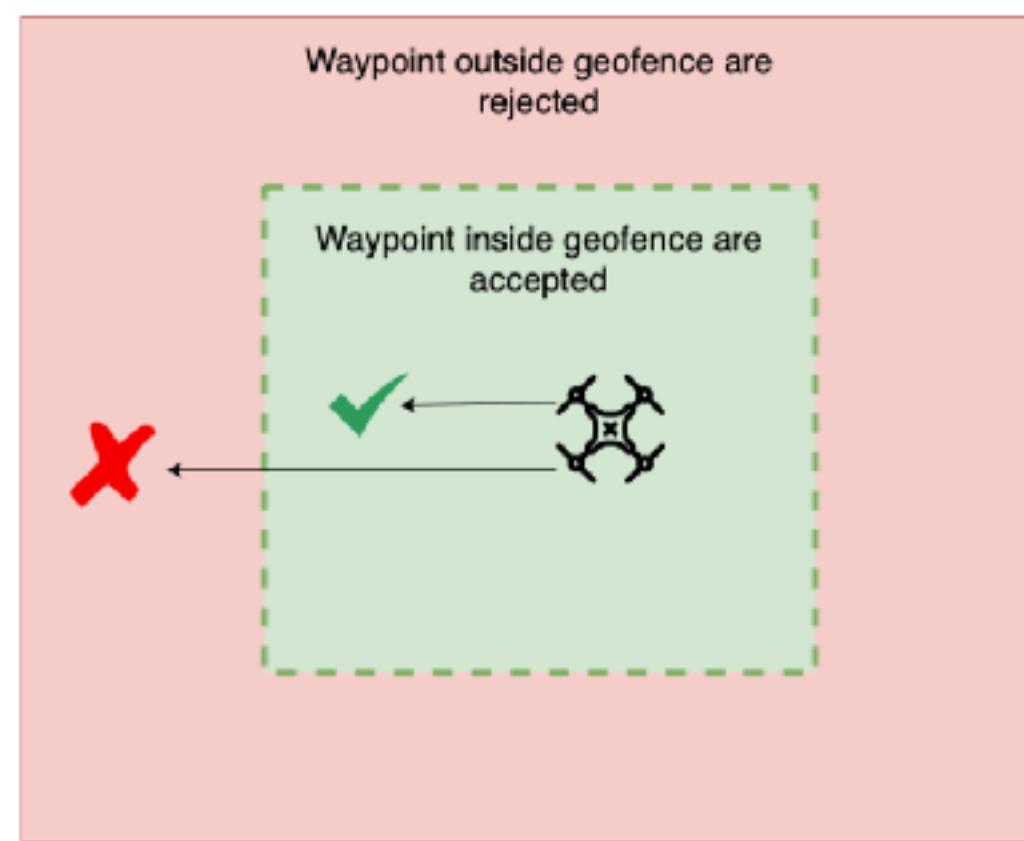
A useful resource to notice: ROS Logging

Checkpoint 1

Launch the simulator and check that you have correctly created the state and safety node as well as changed the keyboard manager to publish the correct data. Your ROS logs should now contain the following message: [INFO] [ros@ros:ros]\$ rosrun ros_tutorials state_and_safety_node

Adding a Verifying State

To learn and apply parameter servers, let's start by adding a verification state to forbid the quadrotor from flying outside of a virtual cage. Verifying that a waypoint is within a geofence (a virtual cage) is a good practice as it makes sure that you do not accidentally send a waypoint to the quadrotor that causes it to fly away or crash into a known obstacle. In general, most commands sent to a robot that is going to result in the robot performing some action in the real-world should be verified for the safety of both the people around it and itself.



Starts highlighting how what is learned is implemented in real systems.

Starts with implementing a basic FSM
(I3 - Pair SE and Robotics topic)

Highlight how ROS allows for looking information allowing for easy debugging
(I1 - Cover robotics fundamentals)

Checkpoint allowing reflection and discussion while allowing freedom to implement their own solution
(I4 - Students make design and implementation decisions)
(I5 - Demonstration, conversation, and checkpoints)

Add a verifying state that emphasizes developing code that is easily parameterizable allowing easy reuse
(I3 - Pair SE and Robotics topic)

Example Lab

Abstractions to manage complexity

In this lab, we will work on three types of abstractions that we use in robotics to help us manage system complexity. First, we will keep working on separating

Create the State and Safety Node

The first step in improving the drone's control software will be to create the node. We will start the implementation of this node by only considering the first objective:

A useful resource to notice: ROS Logging

Checkpoint 1

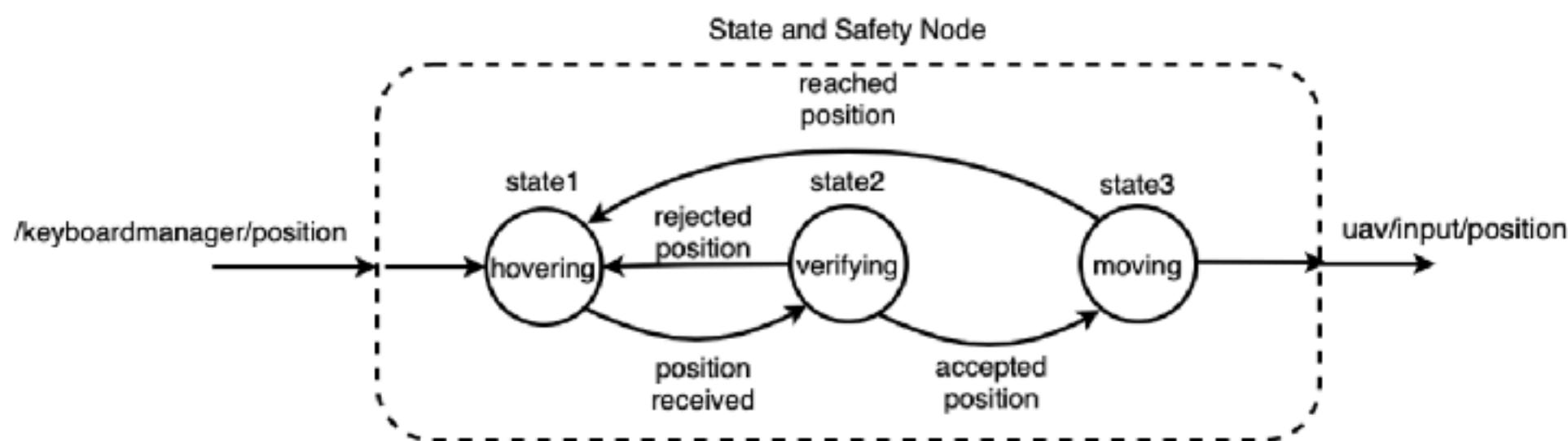
Launch the simulator and check that you have correctly created the state and safety node as well as changed the keyboard manager to publish the correct data. Your ROS workspace should look like this:

Adding a Verifying State

To learn and apply parameter servers, let's start by adding a verification state to forbid the quadrotor from flying outside of a virtual cage. Verifying that a waypoint is

Verifying that Waypoints are within Cage

Next, let's adapt our FSA to include a verifying state. This verification state will verify the command position and make sure it is inside the cage before transitioning to a moving state. The design for the final `state_and_safety` node will be as follows:



Starts highlighting how what is learned is implemented in real systems.

Starts with implementing a basic FSM
(I3 - Pair SE and Robotics topic)

Highlight how ROS allows for looking information allowing for easy debugging
(I1 - Cover robotics fundamentals)

Checkpoint allowing reflection and discussion while allowing freedom to implement their own solution
(I4 - Students make design and implementation decisions)
(I5 - Demonstration, conversation, and checkpoints)

Add a verifying state that emphasizes developing code that is easily parameterizable allowing easy reuse
(I3 - Pair SE and Robotics topic)

Make the FSM slightly more complicated, allowing for more complex behavior.
(I7 - Incrementally build concepts)

Example Lab

Abstractions to manage complexity

In this lab, we will work on three types of abstractions that we use in robotics to help us manage system complexity. First, we will keep working on separating

Create the State and Safety Node

The first step in improving the drone's control software will be to create the node. We will start the implementation of this node by only considering the first objective:

A useful resource to notice: ROS Logging

Checkpoint 1

Launch the simulator and check that you have correctly created the state and safety node as well as changed the keyboard manager to publish the correct data. Your ROS computation graph should look like, as shown below. Take a screenshot of the ROS computation graph:

Adding a Verifying State

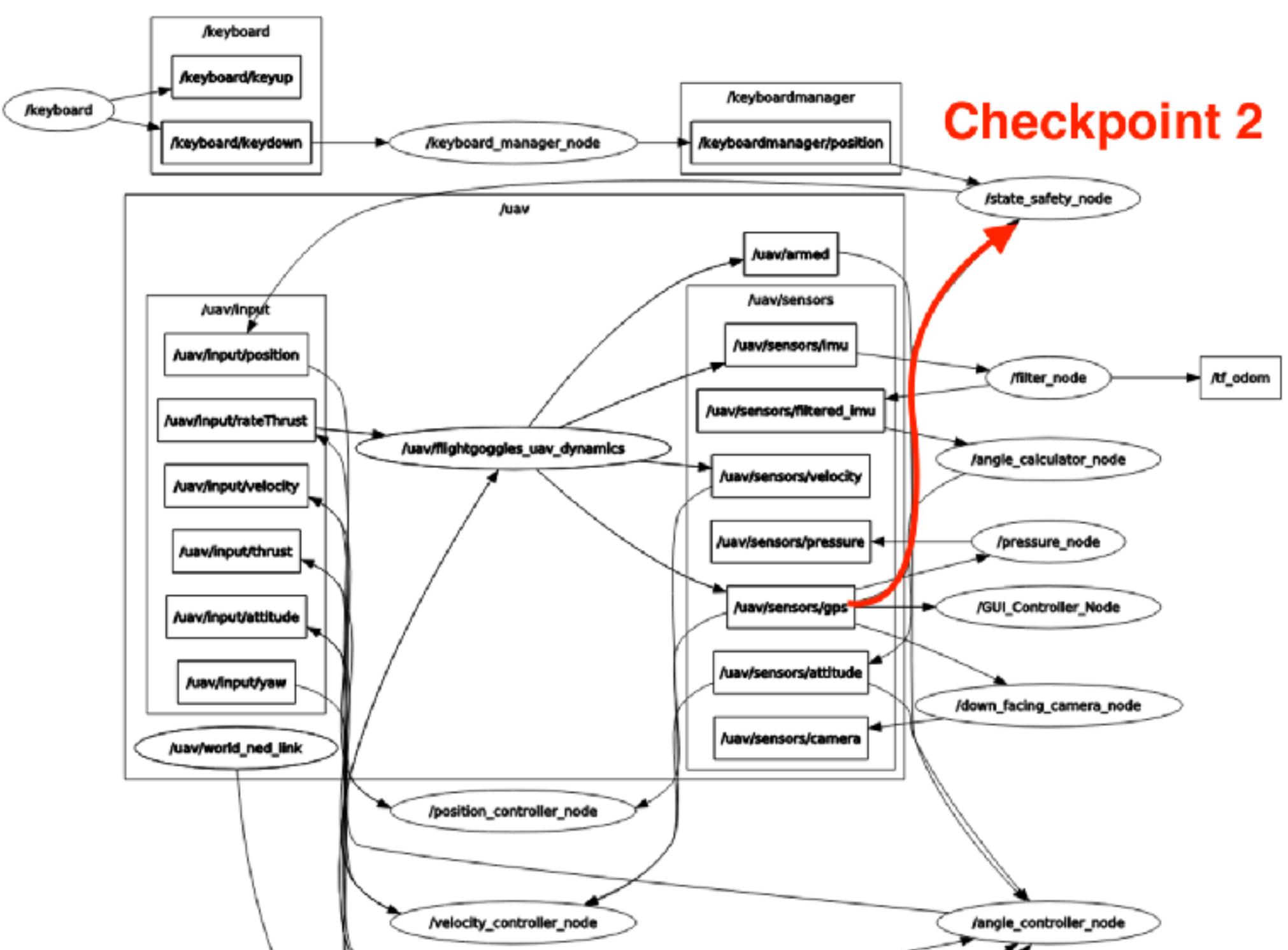
To learn and apply parameter servers, let's start by adding a verification state to forbid the quadrotor from flying outside of a virtual cage. Verifying that a waypoint is

Verifying that Waypoints are within Cage

Now let's adapt our FSA to include a verifying state. This verification state will verify the command position and make sure it is inside the cage before transitioning to a

Checkpoint 2

Launch the simulator and check that you have correctly created the state and safety node as well as changed the keyboard manager to publish the correct data. Your ROS computation graph should look like, as shown below. Take a screenshot of the ROS computation graph:



Starts highlighting how what is learned is implemented in real systems.

Starts with implementing a basic FSM
(I3 - Pair SE and Robotics topic)

Highlight how ROS allows for looking information allowing for easy debugging
(I1 - Cover robotics fundamentals)

Checkpoint allowing reflection and discussion while allowing freedom to implement their own solution
(I4 - Students make design and implementation decisions)
(I5 - Demonstration, conversation, and checkpoints)

Add a verifying state that emphasizes developing code that is easily parameterizable allowing easy reuse
(I3 - Pair SE and Robotics topic)

Make the FSM slightly more complicated, allowing for more complex behavior.
(I7 - Incrementally build concepts)

Example Lab

Abstractions to manage complexity

In this lab, we will work on three types of abstractions that we use in robotics to help us manage system complexity. First, we will keep working on separating

Create the State and Safety Node

The first step in improving the drone's control software will be to create the node. We will start the implementation of this node by only considering the first objective:

A useful resource to notice: ROS Logging

Checkpoint 1

Launch the simulator and check that you have correctly created the state and safety node as well as changed the keyboard manager to publish the correct data. Your ROS computation graph should look like, as shown below. Take a screenshot of the ROS computation graph:

Adding a Verifying State

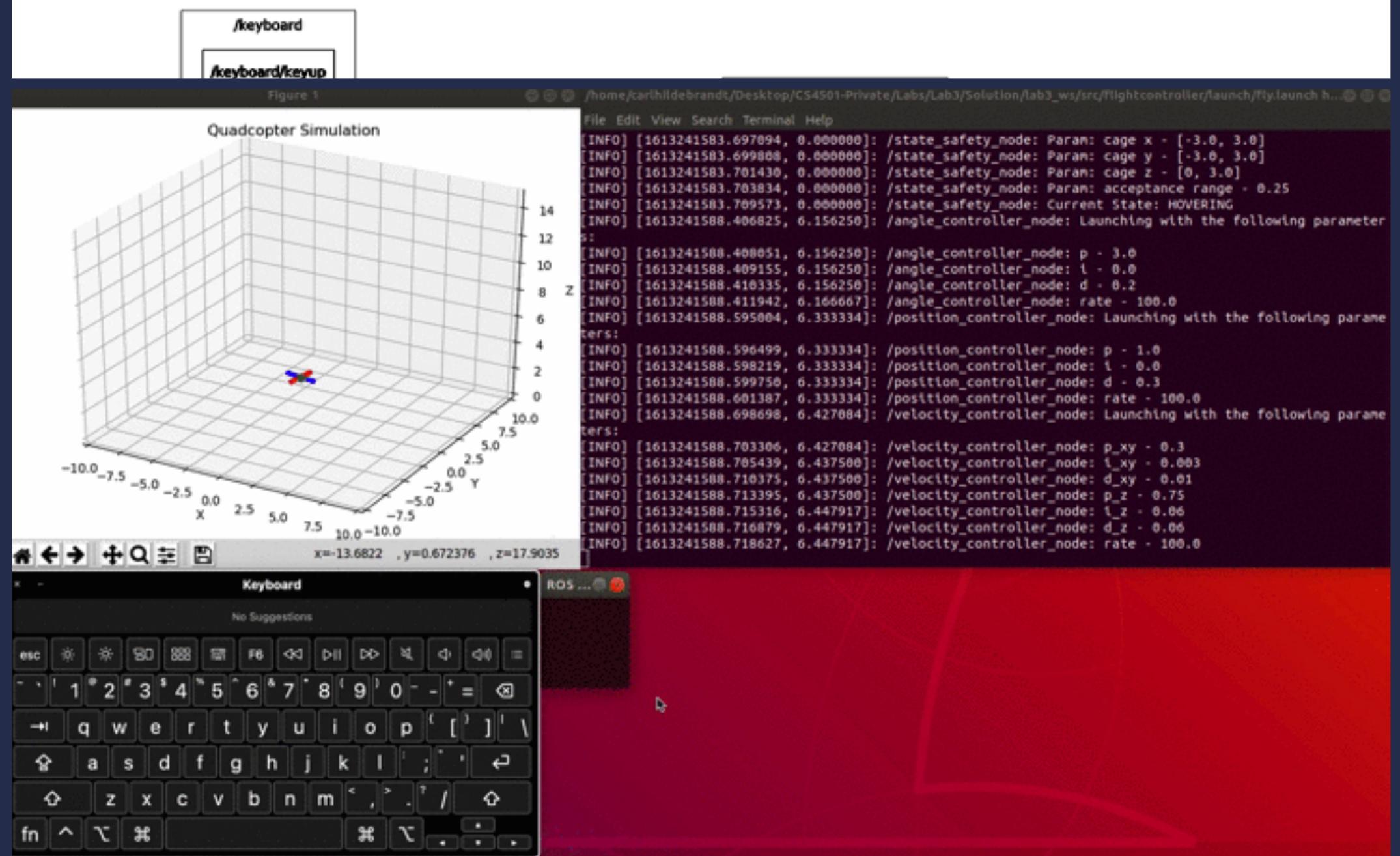
To learn and apply parameter servers, let's start by adding a verification state to forbid the quadrotor from flying outside of a virtual cage. Verifying that a waypoint is

Verifying that Waypoints are within Cage

Now let's adapt our FSM to include a verifying state. This verification state will verify the commanded position and make sure it is inside the cage before transitioning to a

Checkpoint 2

Launch the simulator and check that you have correctly created the state and safety node as well as changed the keyboard manager to publish the correct data. Your ROS computation graph should look like, as shown below. Take a screenshot of the ROS computation graph:



Starts highlighting how what is learned is implemented in real systems.

Starts with implementing a basic FSM
(I3 - Pair SE and Robotics topic)

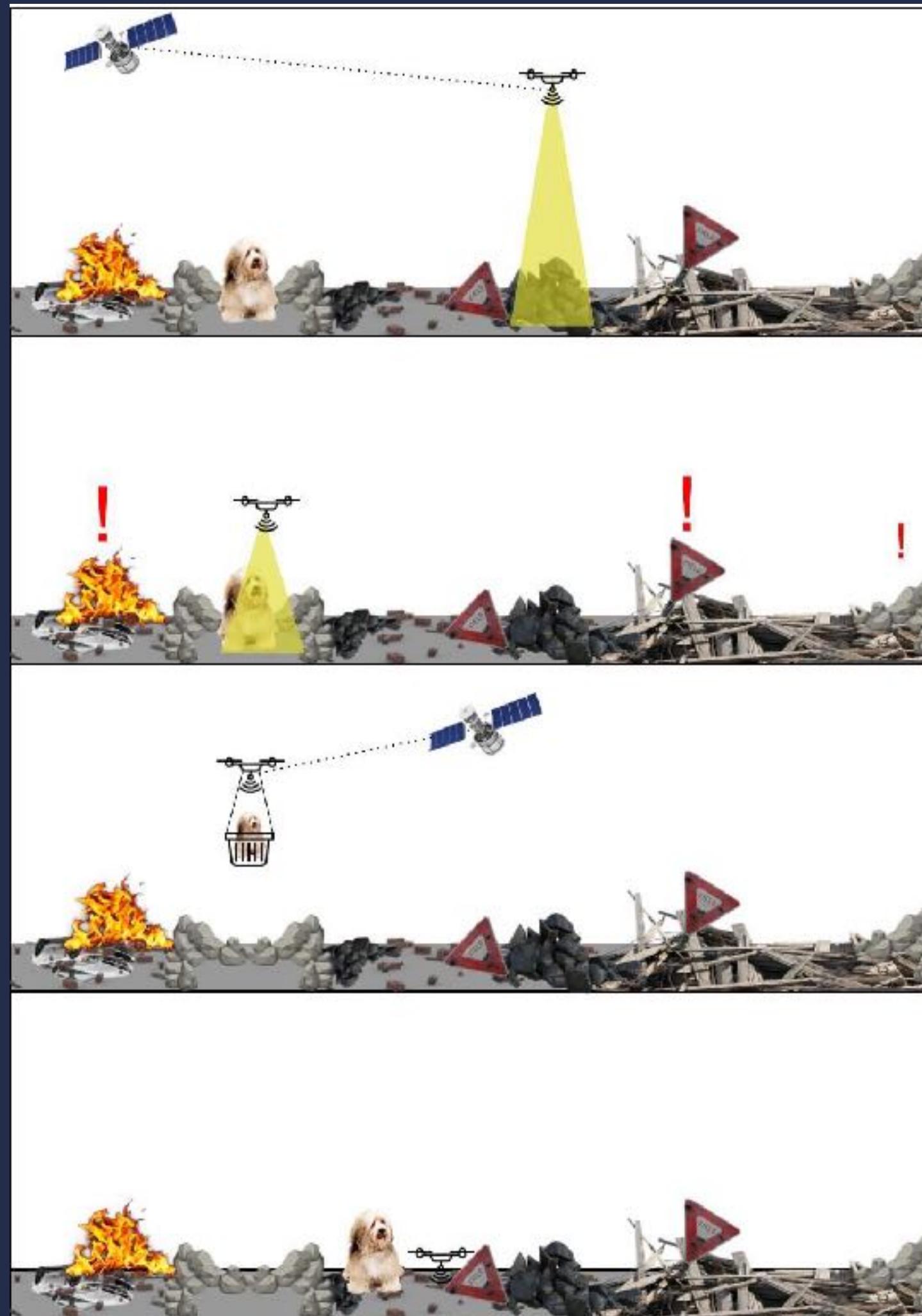
Highlight how ROS allows for looking information allowing for easy debugging
(I1 - Cover robotics fundamentals)

Checkpoint allowing reflection and discussion while allowing freedom to implement their own solution
(I4 - Students make design and implementation decisions)
(I5 - Demonstration, conversation, and checkpoints)

Add a verifying state that emphasizes developing code that is easily parameterizable allowing easy reuse
(I3 - Pair SE and Robotics topic)

Make the FSM slightly more complicated, allowing for more complex behavior.
(I7 - Incrementally build concepts)

Project



Locate itself

Locate object, avoiding obstacles

Collect the object

Return to safe area

Project



Lessons Learned

What worked well



What needs improvement



VS

What Worked Well



1. Pairing SE and robotics topics
2. Building flexibility into the course
3. Using different levels of abstraction
4. Incremental scaffolding of course material
5. Team structure and process
6. Demonstrating and reflecting during checkpoints

What Worked Well



1. Pairing SE and robotics topics
2. Building flexibility into the course
3. Using different levels of abstraction
4. Incremental scaffolding of course material
5. Team structure and process
6. Demonstrating and reflecting during checkpoints

Topic	Lab
Introduction	Lab-1: Set up and Basic ROS
Distinguishing Development Features	Lab-2: ROS processes, communication, and simulation environment
Software Machinery + Q1	Lab-3: Types and machines
Robot and world through sensors	Lab-4: Sensor filtering and fusion
Perception + Q2	... Lab-5: Perception though Analyzing Images
UVA Break Day	Invited Speaker
Controlling your robot	Lab-E: Robotics and Ethics
Making plans + Q3	Lab-6: Controlling and testing robots
Localization and navigation	Lab-7: Mapping and Motion Planning
Transformations	Lab-8: Transformations
Advanced Robotics + Q4	UVA Break Day
Project parameters	Project consult
Project check	Project consult
Project Presentations and Demos	Taking stock

What Worked Well



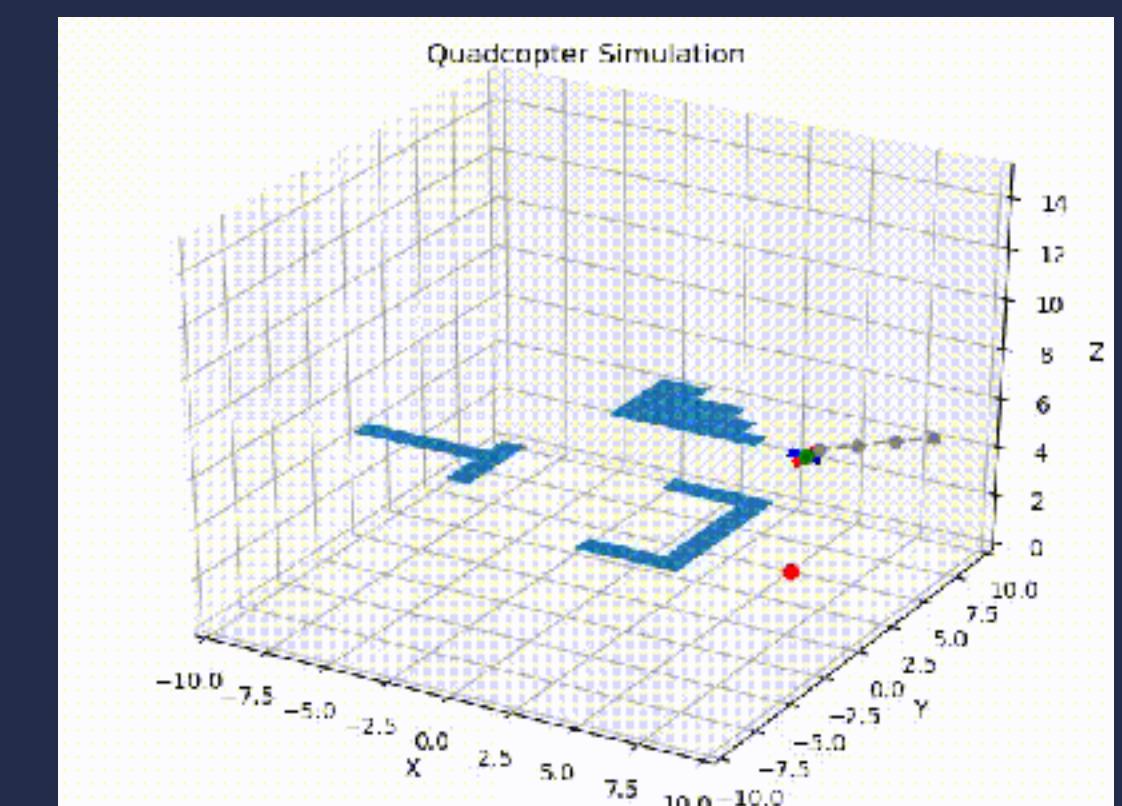
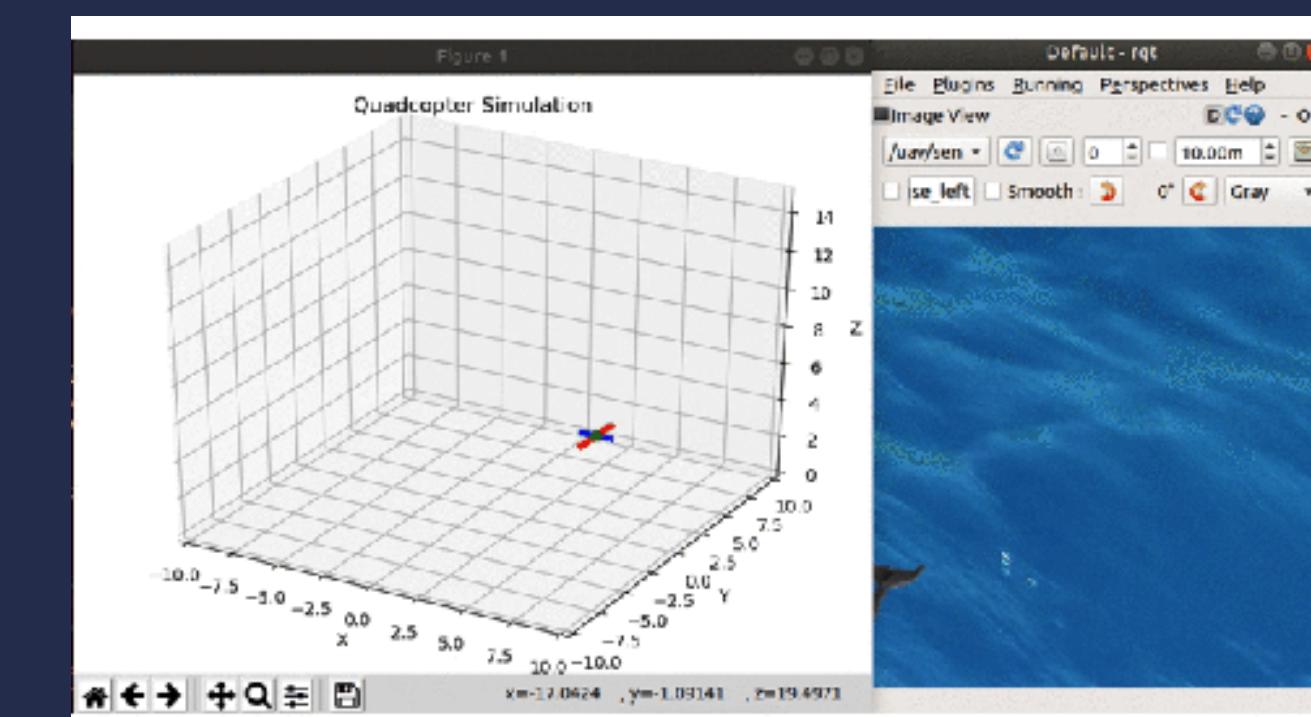
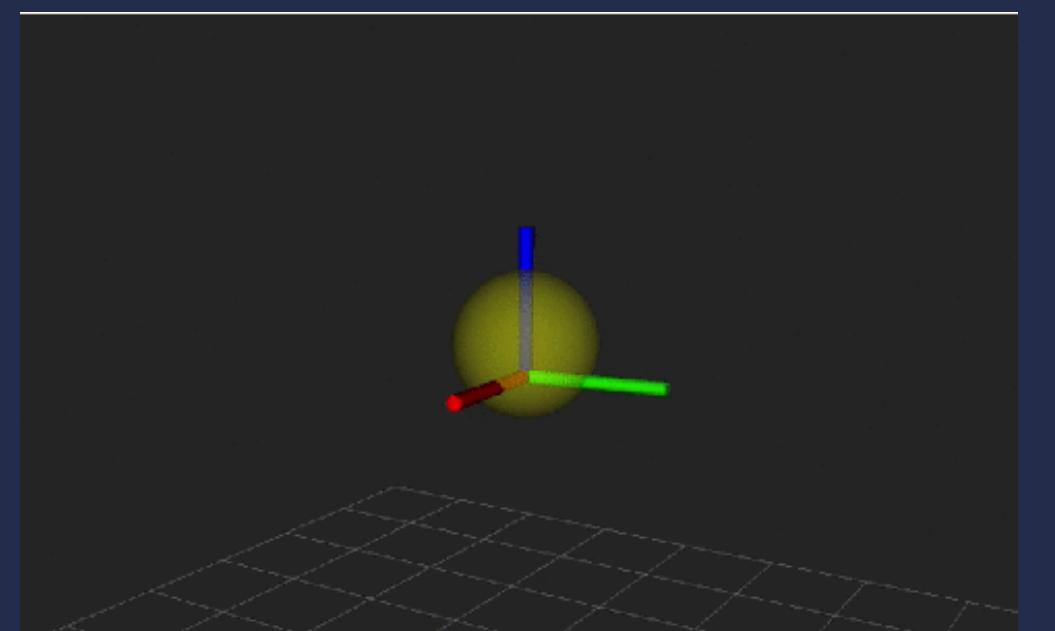
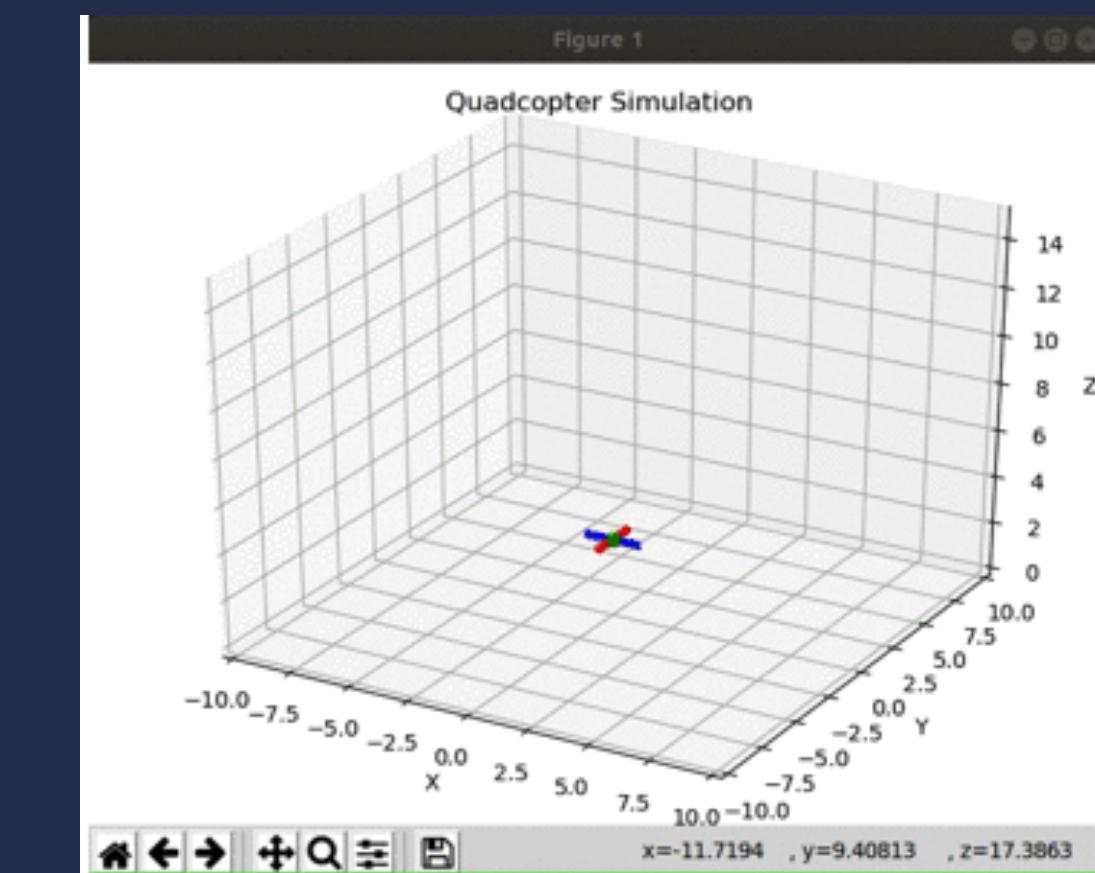
1. Pairing SE and robotics topics
2. Building flexibility into the course
3. Using different levels of abstraction
4. Incremental scaffolding of course material
5. Team structure and process
6. Demonstrating and reflecting during checkpoints

Topic	Lab
Introduction	Lab-1: Set up and Basic ROS
Distinguishing Development Features	Lab-2: ROS processes, communication, and simulation environment
Software Machinery + Q1	Lab-3: Types and machines
Robot and world through sensors	Lab-4: Sensor filtering and fusion
Perception + Q2	Lab-5: Perception though Analyzing Images
UVA Break Day	Invited Speaker
Controlling your robot	Lab-E: Robotics and Ethics
Making plans + Q3	Lab-6: Controlling and testing robots
Localization and navigation	Lab-7: Mapping and Motion Planning
Transformations	Lab-8: Transformations
Advanced Robotics + Q4	UVA Break Day
Project parameters	Project consult
Project check	Project consult
Project Presentations and Demos	Taking stock

What Worked Well



1. Pairing SE and robotics topics
2. Building flexibility into the course
3. Using different levels of abstraction
4. Incremental scaffolding of course material
5. Team structure and process
6. Demonstrating and reflecting during checkpoints



What Worked Well



1. Pairing SE and robotics topics
2. Building flexibility into the course
3. Using different levels of abstraction
4. Incremental scaffolding of course material
5. Team structure and process
6. Demonstrating and reflecting during checkpoints

Topic
Introduction
Distinguishing Development Features
Software Machinery + Q1
Robot and world through sensors
Perception + Q2
UVA Break Day
Controlling your robot
Making plans + Q3
Localization and navigation
Transformations
Advanced Robotics + Q4
Project parameters
Project check
Project Presentations and Demos



...

What Worked Well



1. Pairing SE and robotics topics
2. Building flexibility into the course
3. Using different levels of abstraction
4. Incremental scaffolding of course material
5. Team structure and process
6. Demonstrating and reflecting during checkpoints

Current Team

Sebastian Elbaum

Instructor



selbaum at virginia

Meriel Stein

Teaching Assistant



meriel at virginia

Trey Woodlief

Teaching Assistant



adw8dm at virginia

Carl Hildebrandt

Leads Labs



hildebrandt.carl at virginia

What Worked Well



1. Pairing SE and robotics topics

2. Building flexibility into the course

3. Using different levels of abstraction

4. Incremental scaffolding of course material

5. Team structure and process

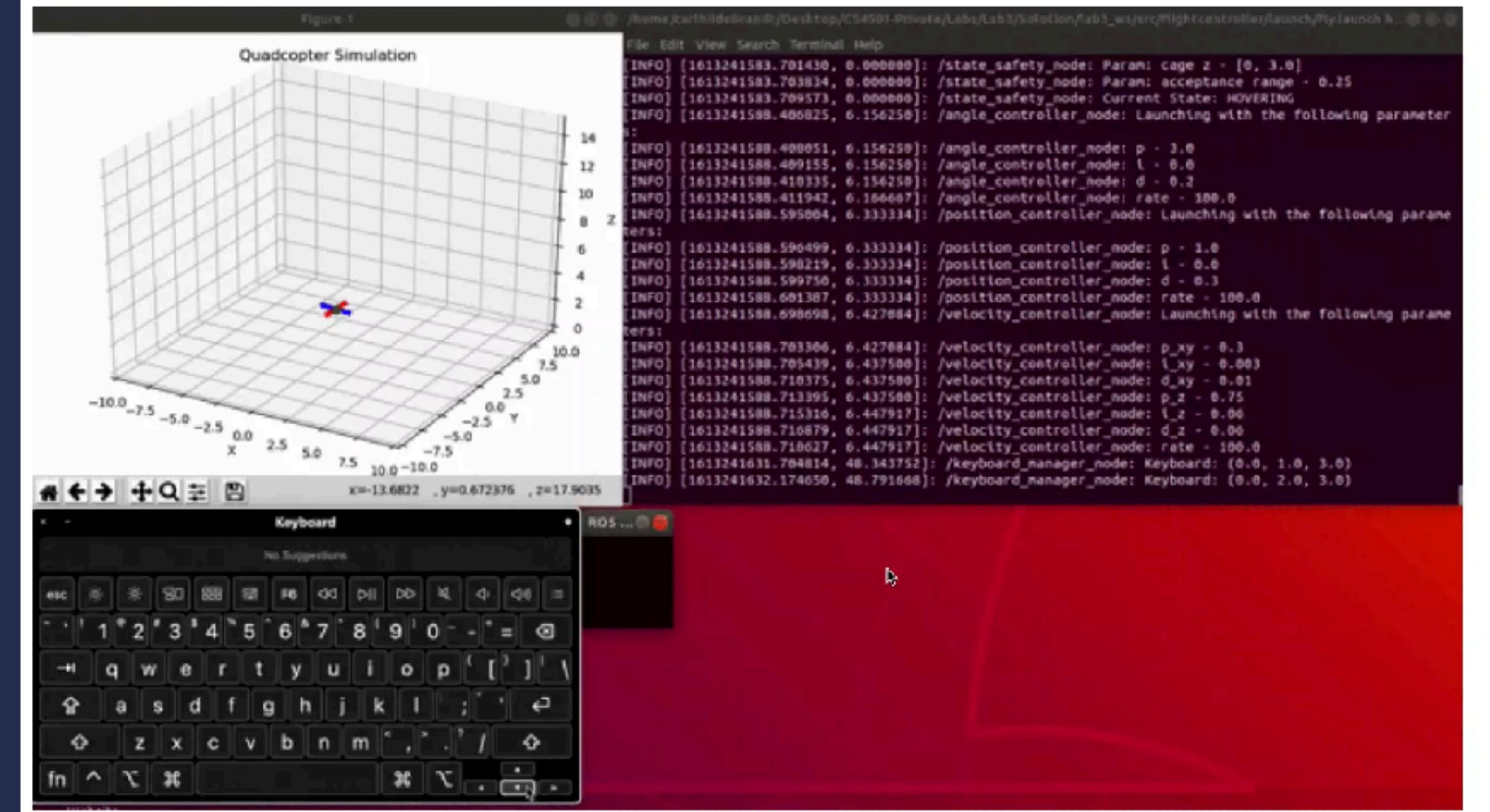
6. Demonstrating and reflecting during checkpoints

Checkpoint 2

1. What cases do the unit tests in `test_moving_averag.py` cover?
2. Take a screenshot of your RViz set up.
3. How does the new ball compare in size to the old one? Why?
4. How does the movement of the new ball compare to the old one? Why?
5. How does changing the window size affect the error?
6. What information do we lose by using moving average instead of individual measurements?

Checkpoint 4

Show the quadrotor flying inside the geofence area. First, send the drone a waypoint inside the geofence area. Second, send the drone a waypoint outside of the geofence area.



What Needs Improvement



1. Diversity of student machines presents a continual challenge
2. Requires identification of fundamental robotic topics and matching SE principles
3. Freedom of design and implementation requires more time for checking and discussion
4. Defining prerequisites for the course is challenging
5. Require a way to empirically assess the success of this course

What Needs Improvement



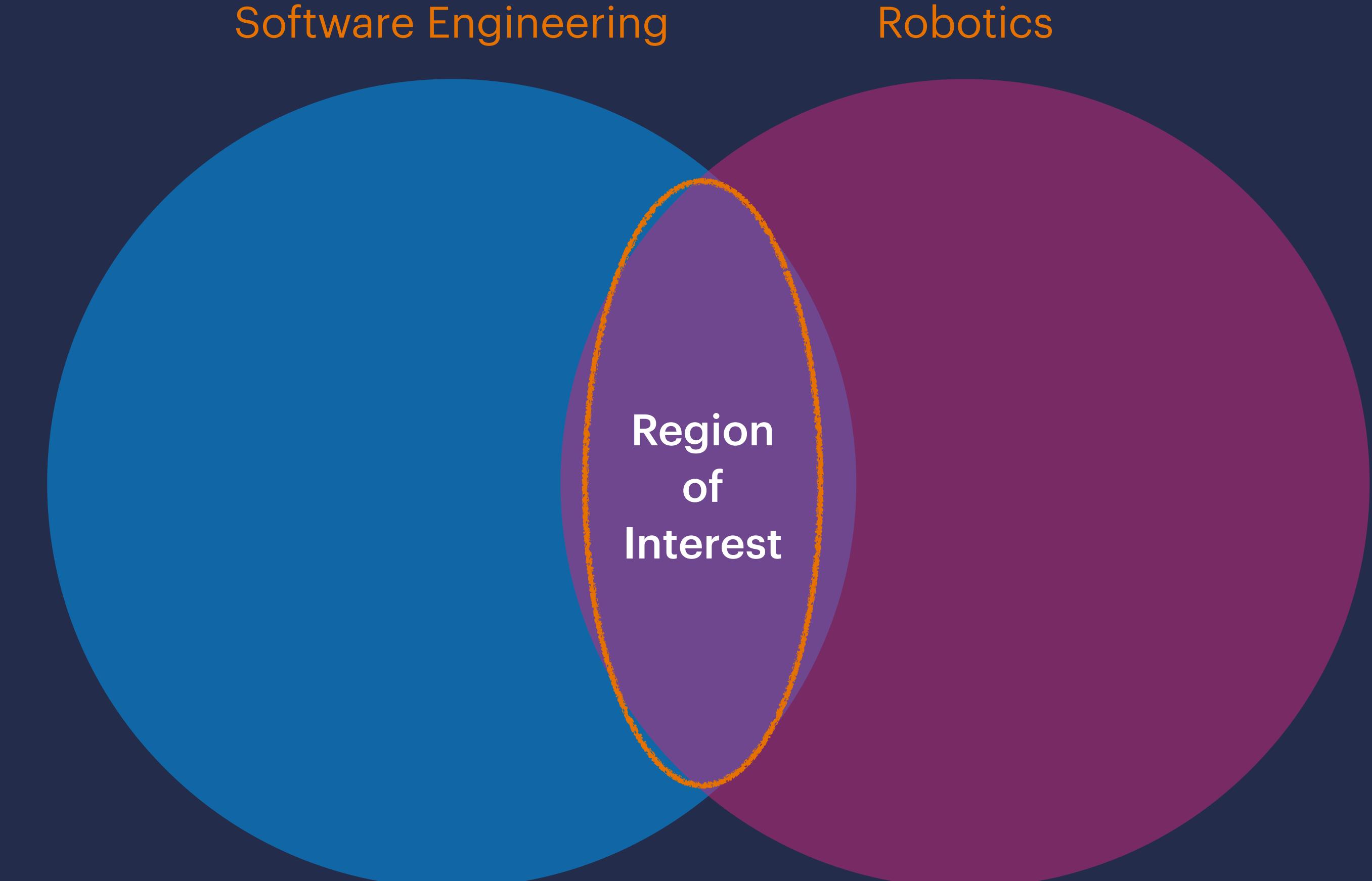
1. Diversity of student machines presents a continual challenge
2. Requires identification of fundamental robotic topics and matching SE principles
3. Freedom of design and implementation requires more time for checking and discussion
4. Defining prerequisites for the course is challenging
5. Require a way to empirically assess the success of this course



What Needs Improvement



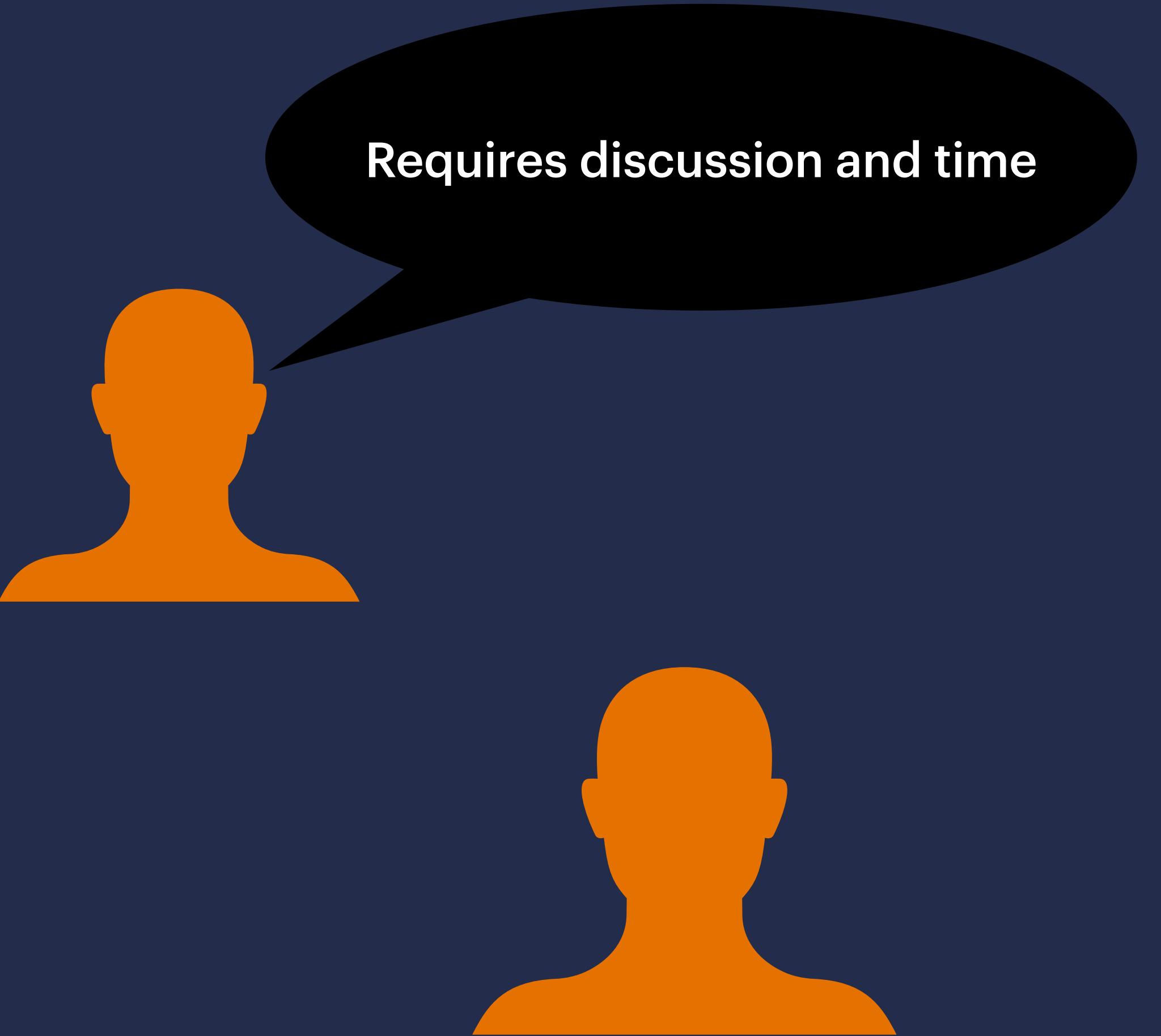
1. Diversity of student machines presents a continual challenge
2. Requires identification of fundamental robotic topics and matching SE principles
3. Freedom of design and implementation requires more time for checking and discussion
4. Defining prerequisites for the course is challenging
5. Require a way to empirically assess the success of this course



What Needs Improvement



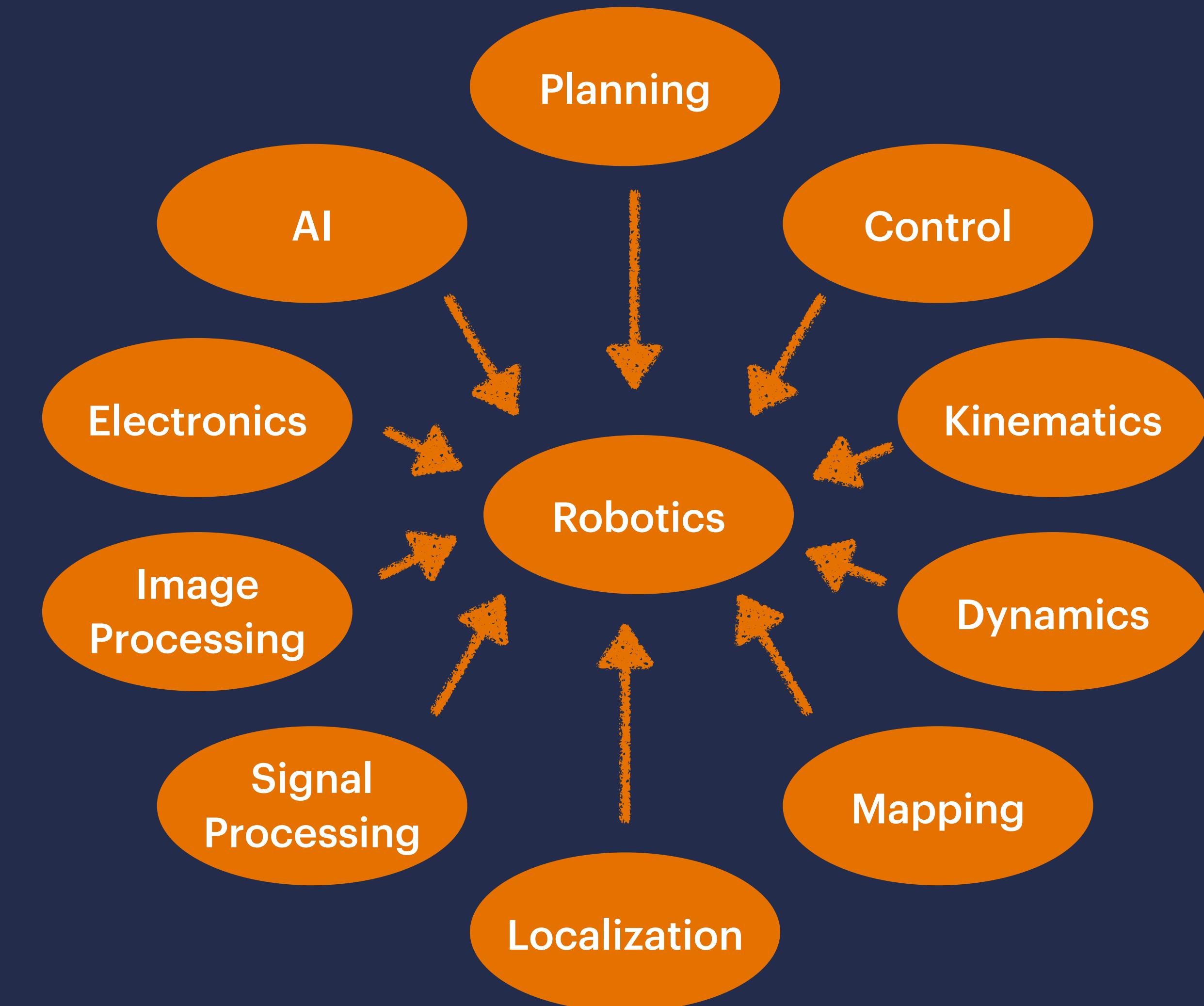
1. Diversity of student machines presents a continual challenge
2. Requires identification of fundamental robotic topics and matching SE principles
3. Freedom of design and implementation requires more time for checking and discussion
4. Defining prerequisites for the course is challenging
5. Require a way to empirically assess the success of this course



What Needs Improvement



1. Diversity of student machines presents a continual challenge
2. Requires identification of fundamental robotic topics and matching SE principles
3. Freedom of design and implementation requires more time for checking and discussion
4. Defining prerequisites for the course is challenging
5. Require a way to empirically assess the success of this course



What Needs Improvement



1. Diversity of student machines presents a continual challenge
2. Requires identification of fundamental robotic topics and matching SE principles
3. Freedom of design and implementation requires more time for checking and discussion
4. Defining prerequisites for the course is challenging
5. Require a way to empirically assess the success of this course



Impact



This year we have 75 students!

4 Ended up in Robotics Industry



"During the interview, I talked about your course a great deal"
"[I talked about] how path planning differs in this scenario versus a UGV or drone type system"
"I grew a lot throughout it [the course]"

"I actually used some concepts we learned about in class in my interviews, and I think it helped a lot!"
"I think it was a great way to understand the fundamentals of robotics!"

Conclusion

Introduced a course aiming at equipping students with a unique understanding of the challenges in developing the software underlying robotic systems and a set of tools to address those challenges

Conclusion



Introduced a course aiming at equipping students with a unique understanding of the challenges in developing the software underlying robotic systems and a set of tools to address those challenges

The screenshot shows a web browser displaying the course website for "Robotics for Software Engineers" (CS 4501 - Spring 2021). The page includes sections for "Team", "Goal and Scope", and "Class location and time". A message from the instructor, Sebastian Elbaum, is visible in the "Goal and Scope" section.

Robotics for Software Engineers
CS 4501 - Spring 2021

This course is part of our ongoing effort to bridge the gap between software engineering and robotics. If you are a student and find value in the lectures and in the labs please drop us an email so that we can get an idea of the impact this effort is having and how we can improve it. If you are a faculty member interested in teaching a course like this, reach out to us as we have supplementary material and hard earned experiences that might be helpful. Thank you! - Sebastian Elbaum.

Team

- Sebastian Elbaum - Instructor, sebaum@virginia.edu
- Trey Woodlief - Teaching Assistant, adw8alm@virginia.edu
- Meriel Stein - Teaching Assistant, merielas@virginia.edu
- Carl Hildebrandt - Leads Lab, hildebrandtcar@virginia.edu

This is going to be a great class but the semester is likely to be another weird one with remote learning and hybrid strategies of different types, so let's be patient with each other, let's be honest with each other and let's strive to learn as much as possible within our means. - Sebastian Elbaum

Goal and Scope

Developing software for robot systems is challenging as they must sense, actuate, and represent the physical world. Sensing the physical world is usually noisy, actuating in and on the world is often inaccurate, and the knowledge and representation of the world is incomplete and uncertain. In this class we will explore software engineering approaches to cope with those challenges. You will learn to use domain-specific abstractions, architectures, libraries, and validation approaches and tools to safely perform robot activities like motion, navigation, perception, planning, and interaction. The expectation is that this course will open up new career options in robotics for our students.

Class location and time

- Tuesday and Thursday from 2:00PM to 3:30PM
- Class will be online with most lectures on Tuesdays and labs on Thursdays

Office Hours

- Trey Woodlief: Mondays 9AM-11AM
- Meriel Stein: Thursdays 9AM-11AM
- Sebastian Elbaum: Friday 9AM-10AM or by email request