Preparing Software Engineers to Develop Robot Systems

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

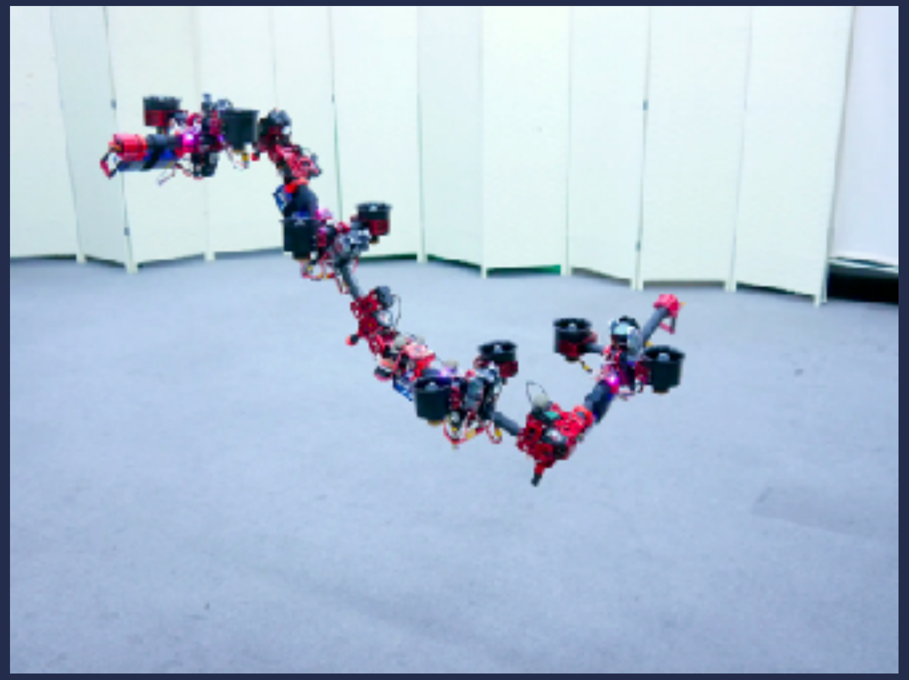
The University of Virginia

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



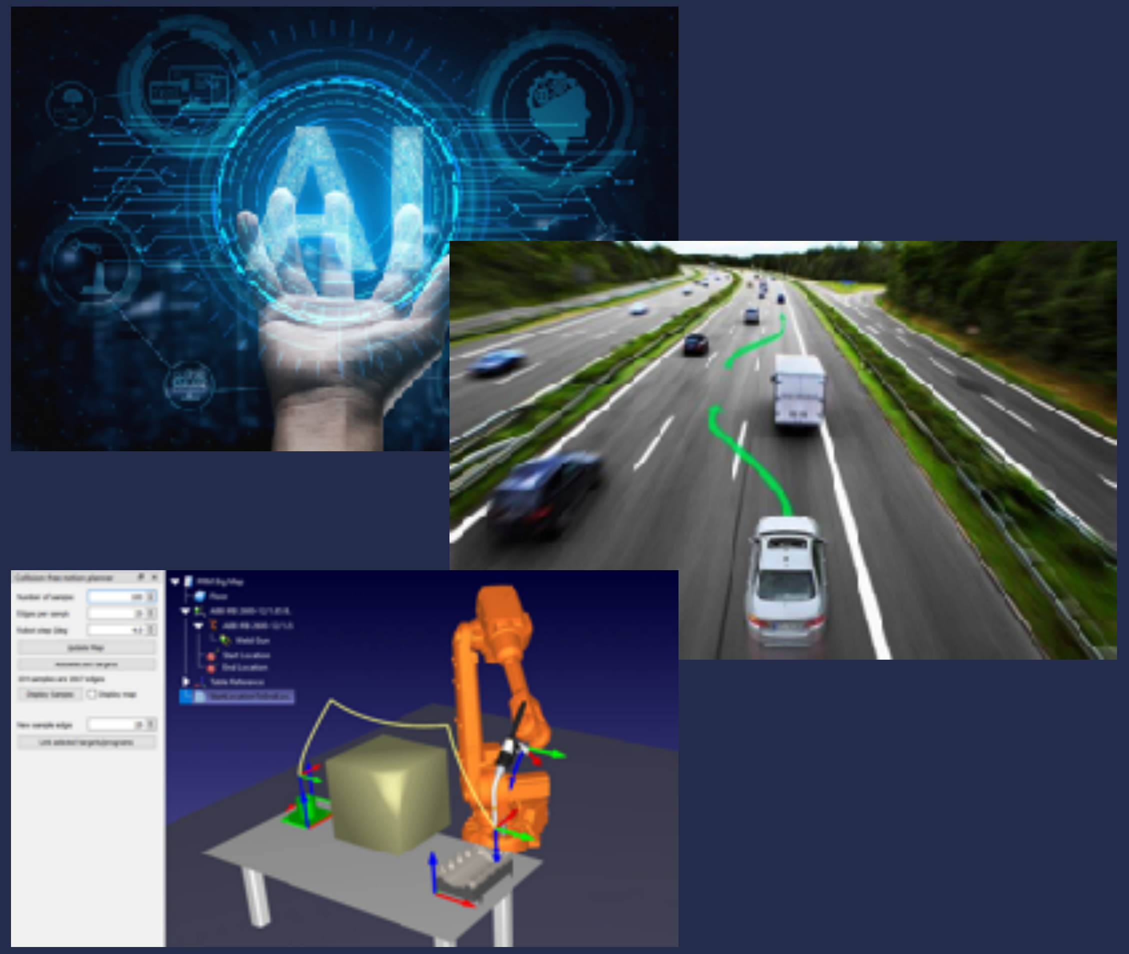
Robotics

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

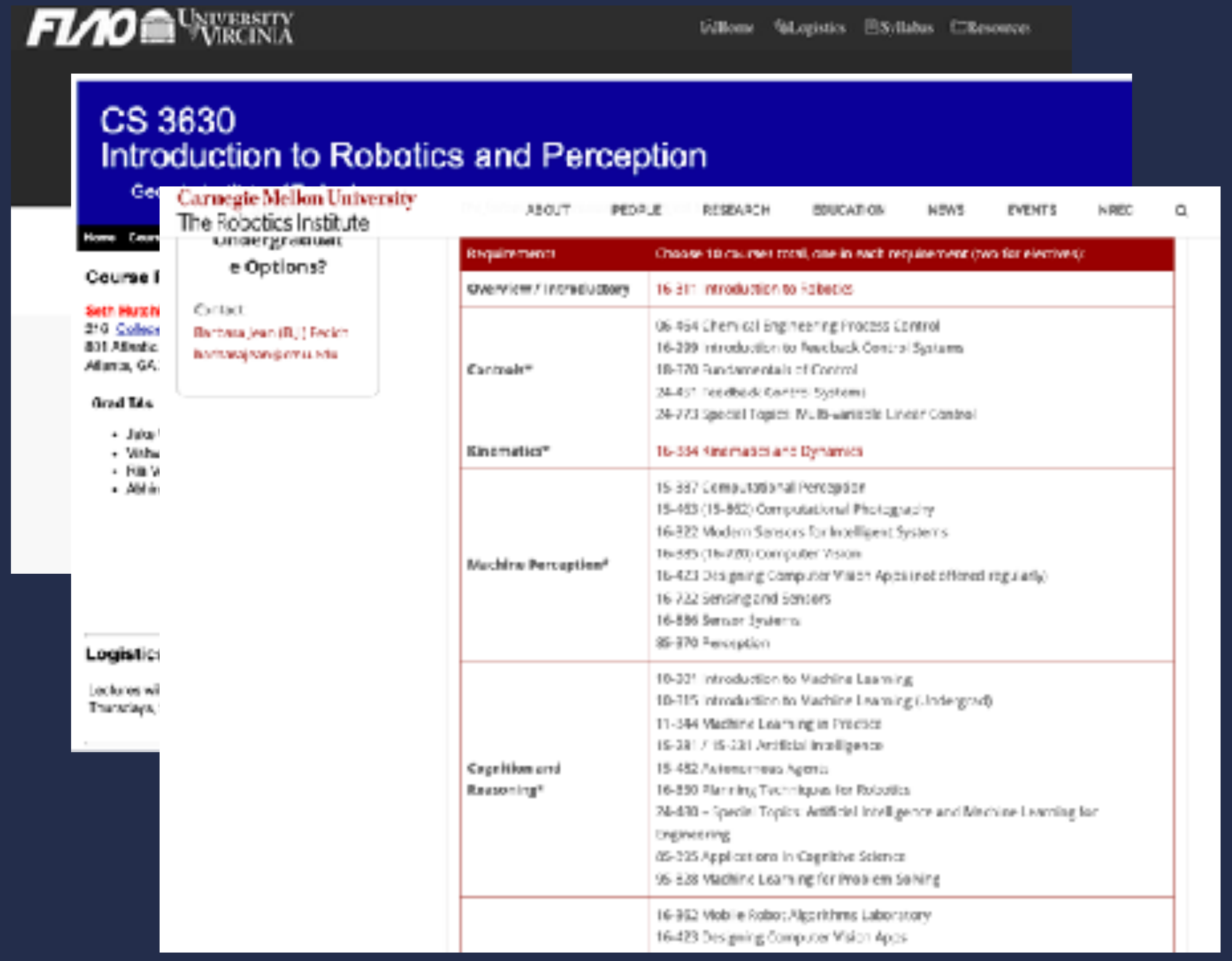
Traditional CS Curricula

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

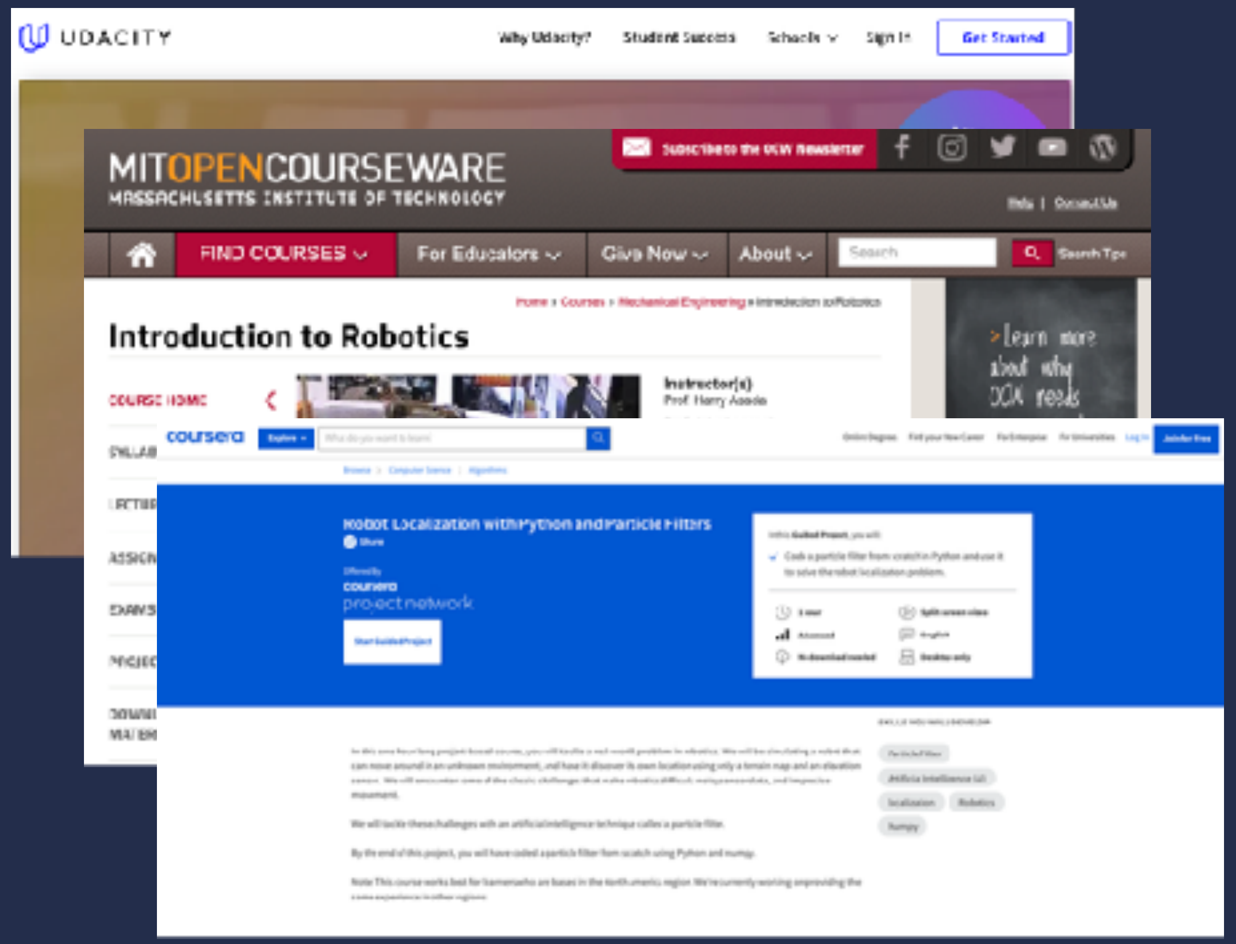
Specialized Graduate-Level Courses



CPS/Embedded Systems Courses



Massive Open Online Courses

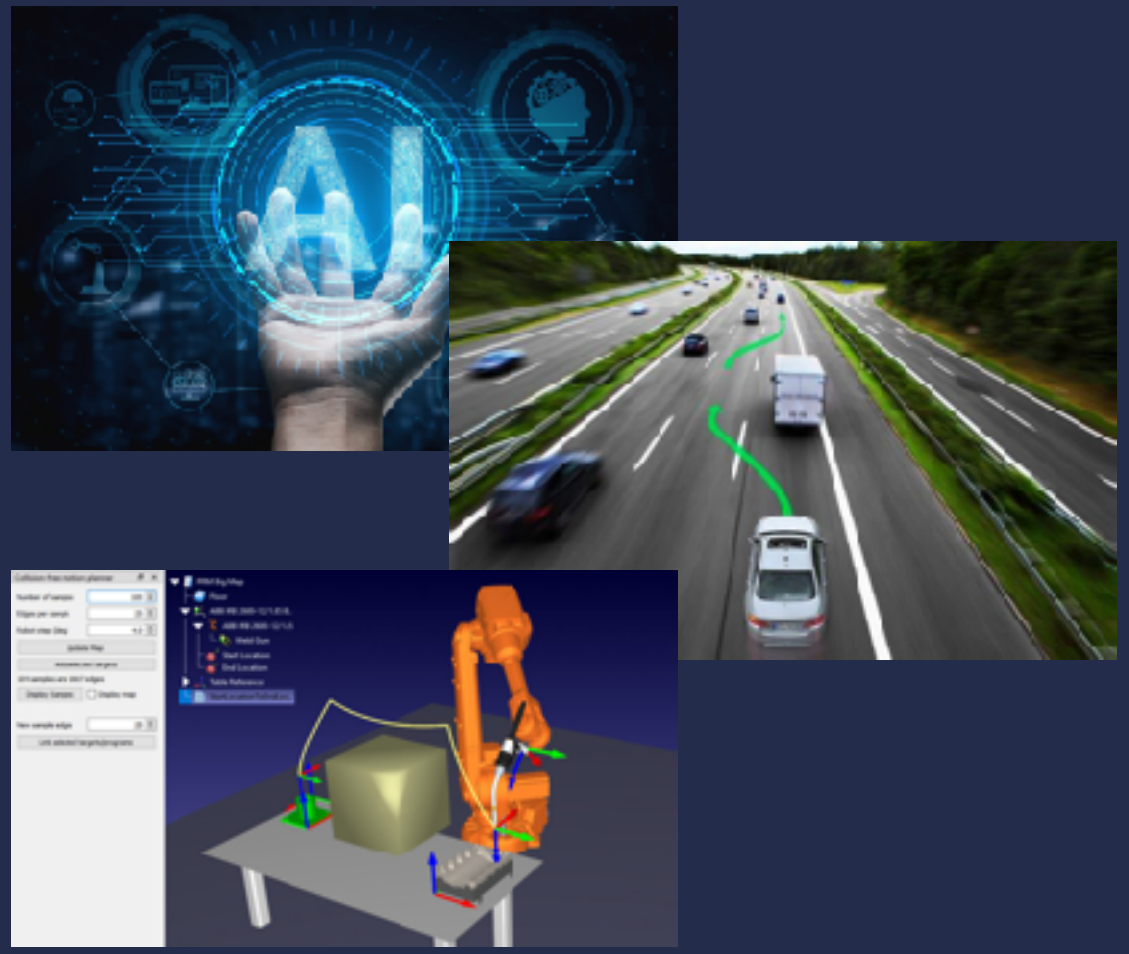


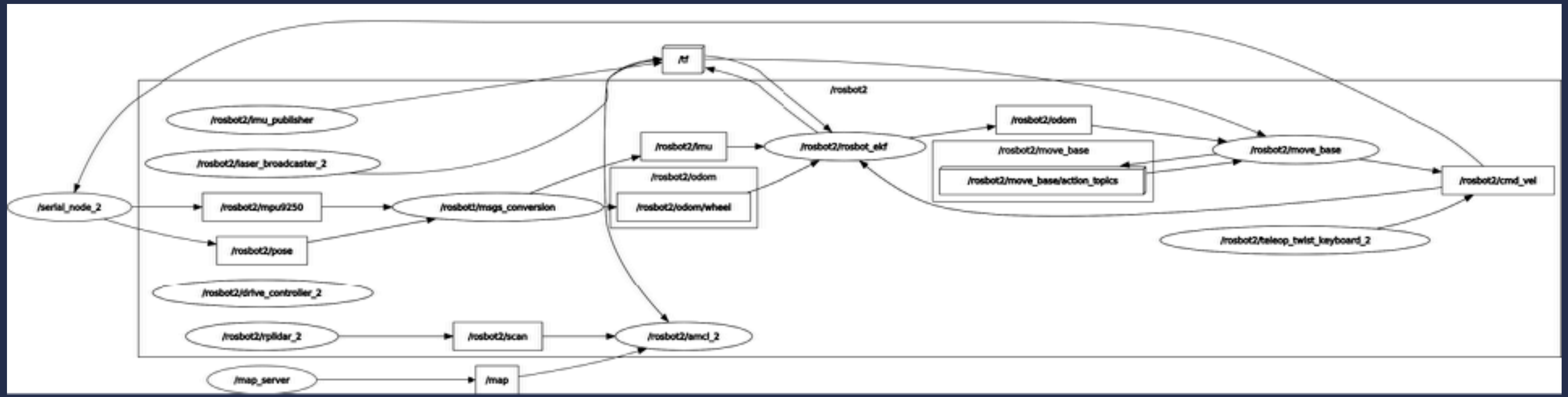
Traditional SE Courses

Traditional CS Curricula

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

Specialized Graduate-Level Courses





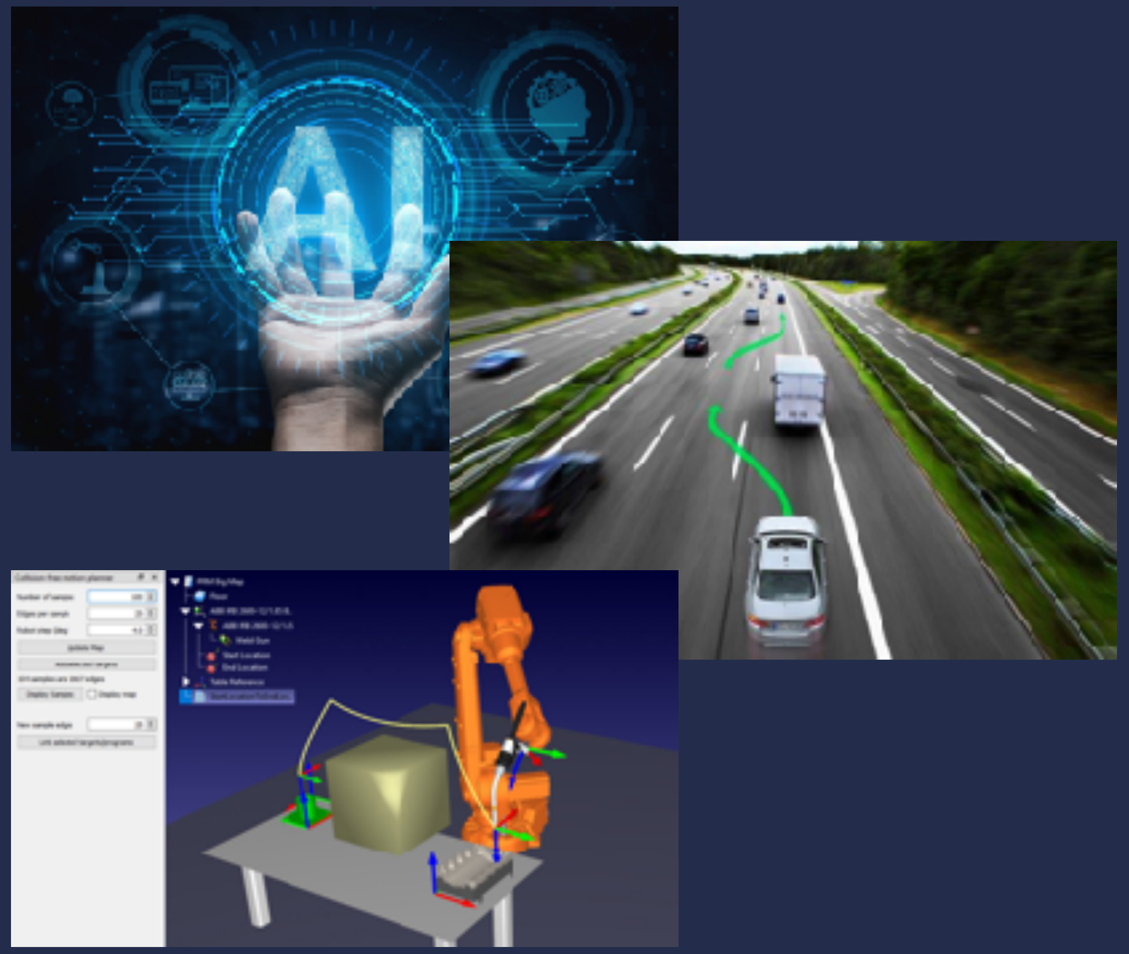
Overlooks that robotics heavily relies on software.

Assumes students are familiar with software engineering.

Traditional CS Curricula

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

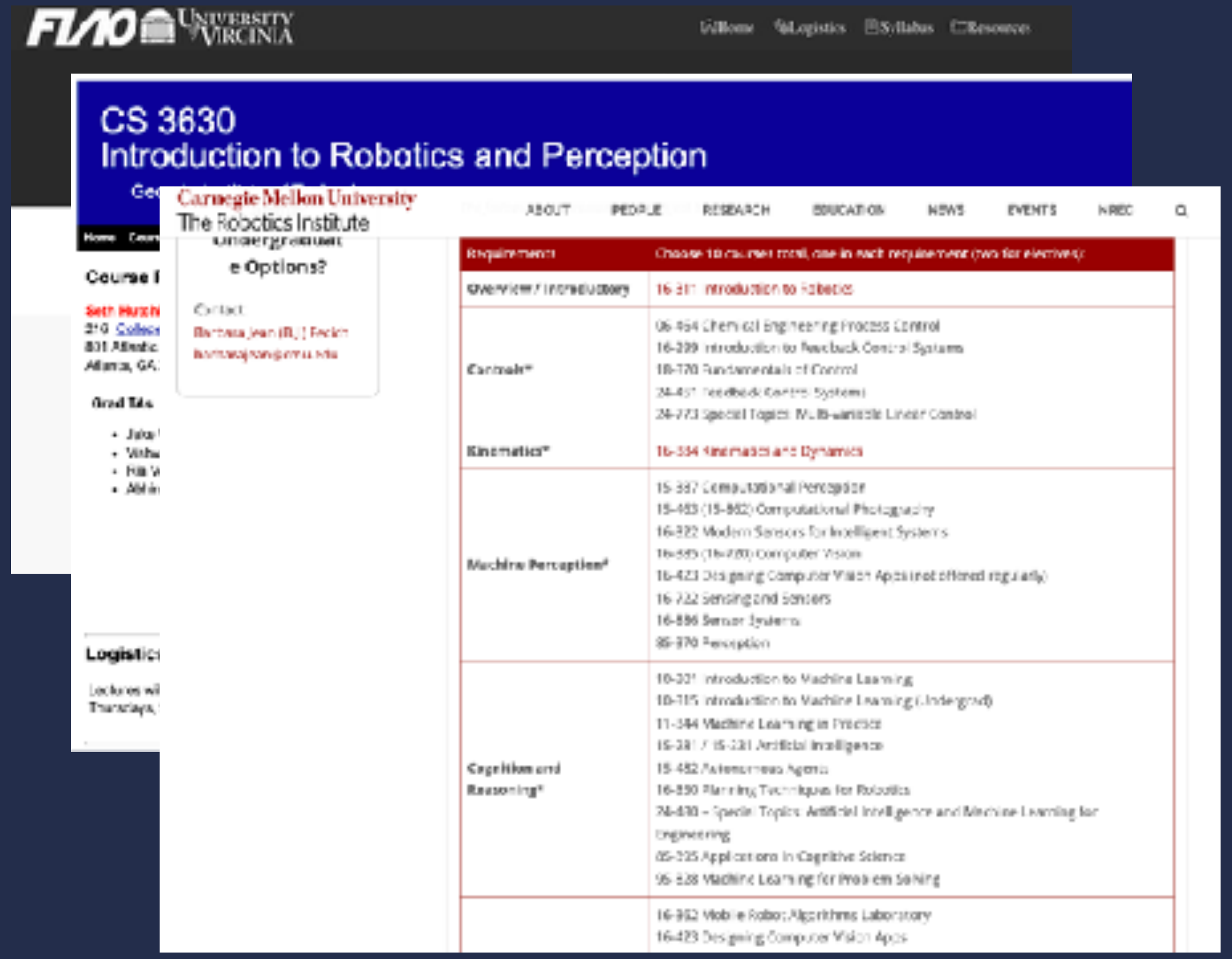
Specialized Graduate-Level Courses



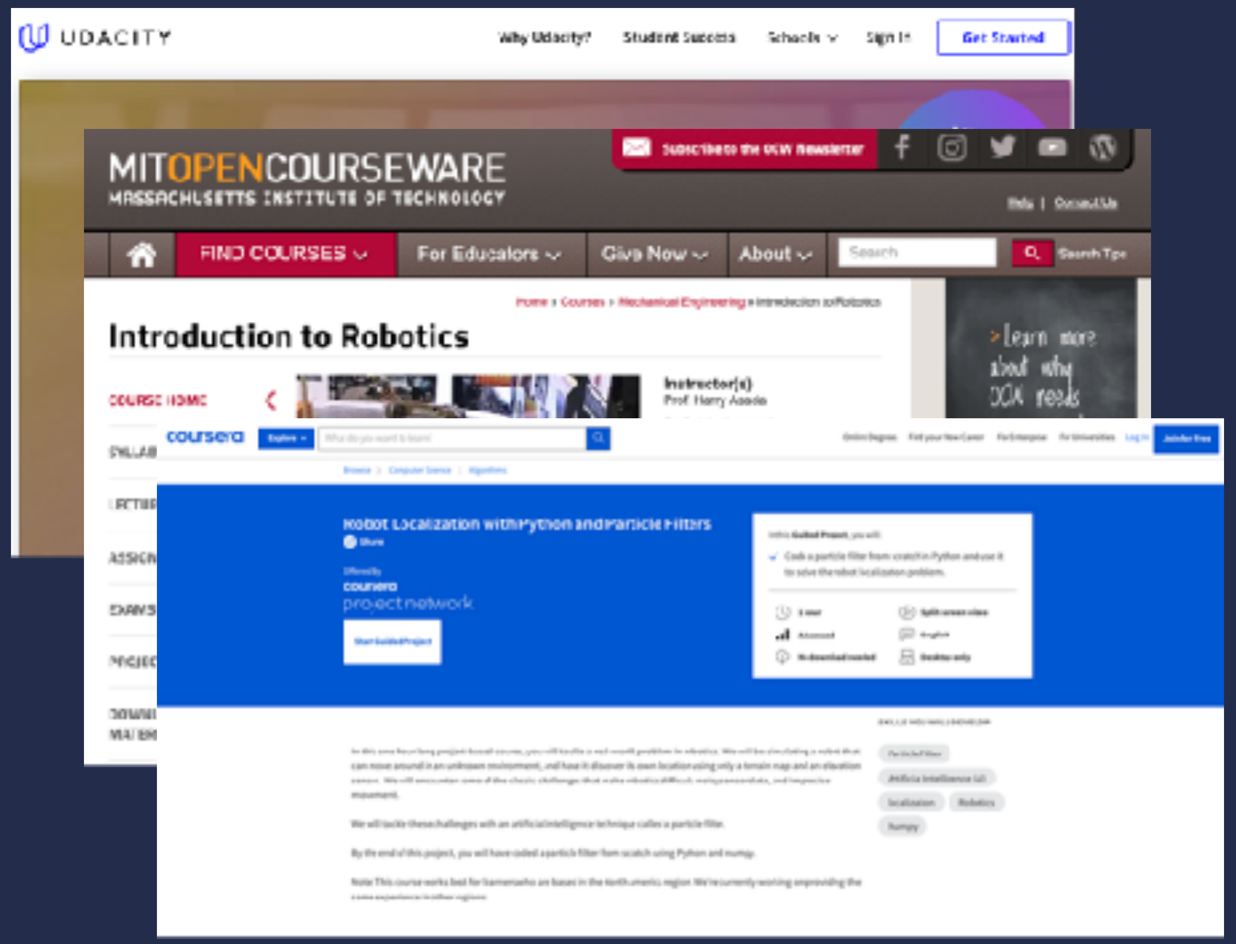
Overlooks that robotics heavily relies on software.

Assumes students are familiar with software engineering.

CPS/Embedded Systems Courses



Massive Open Online Courses

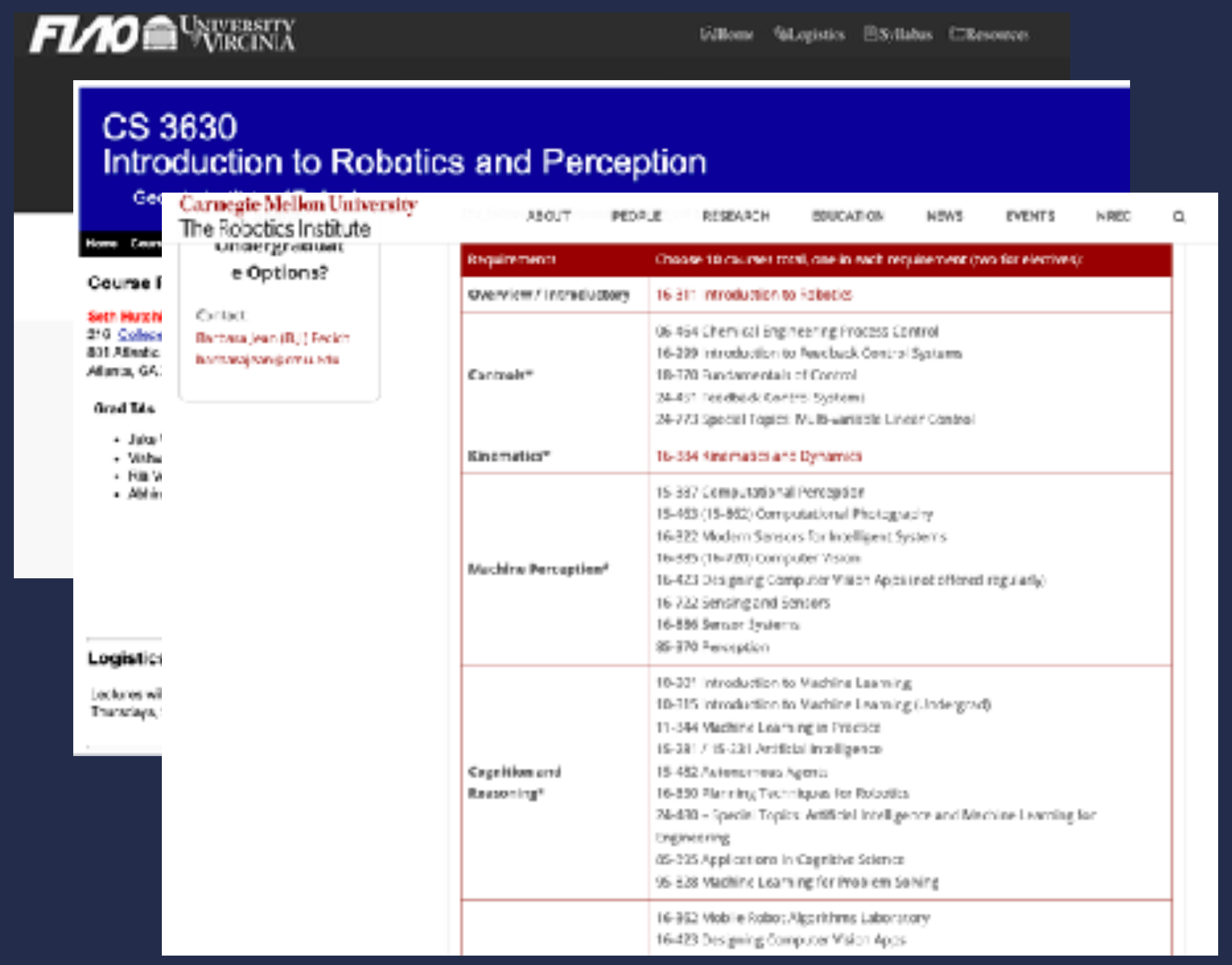


Traditional SE Courses

Traditional CS Curricula

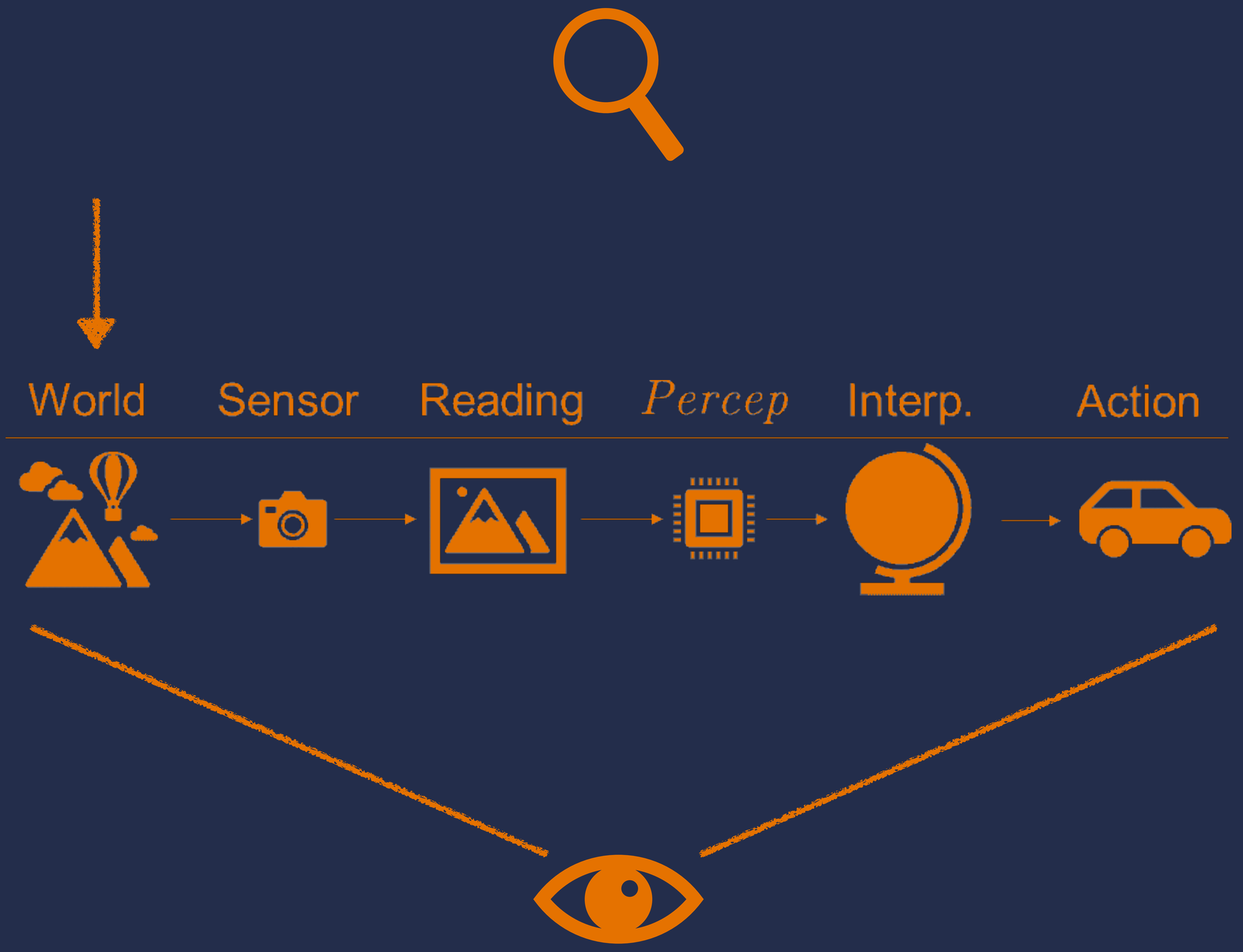
Preparing our future software engineers for this has been met primarily through:

CPS/Embedded Systems Courses



Tends to focus on particular aspects of robot pipeline.

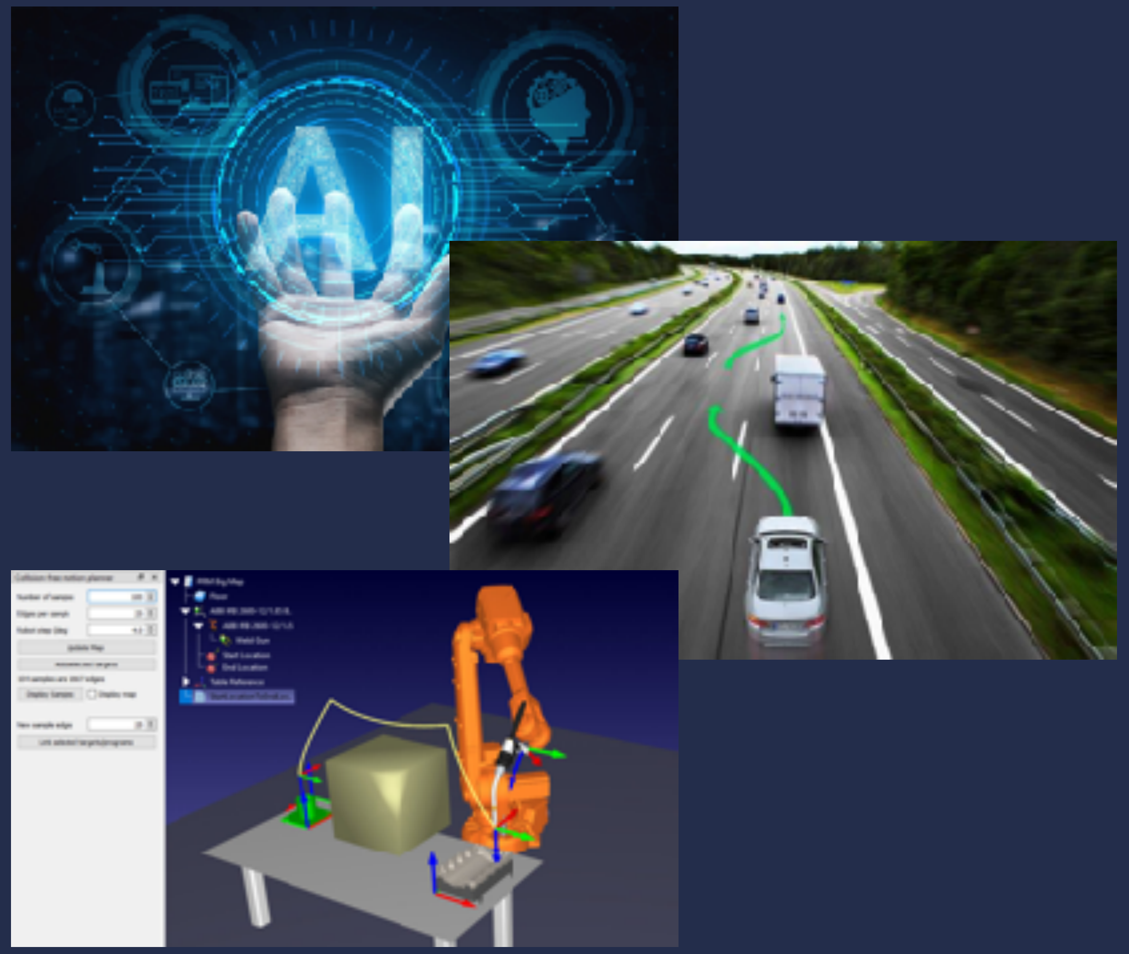
Misses opportunities to discuss broader crosscutting issues.



Traditional CS Curricula

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

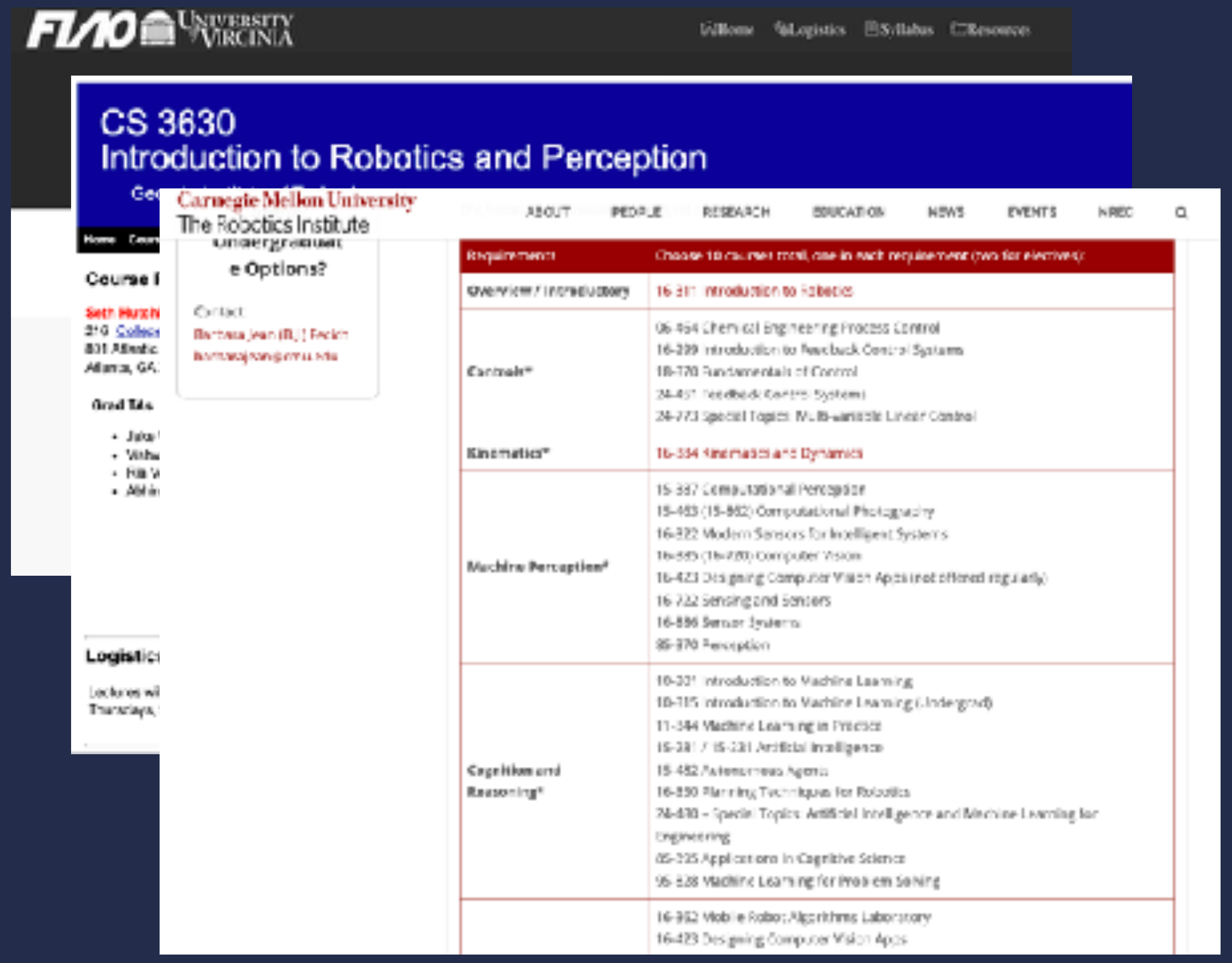
Specialized Graduate-Level Courses



Overlooks that robotics heavily relies on software.

Assumes students are familiar with software engineering.

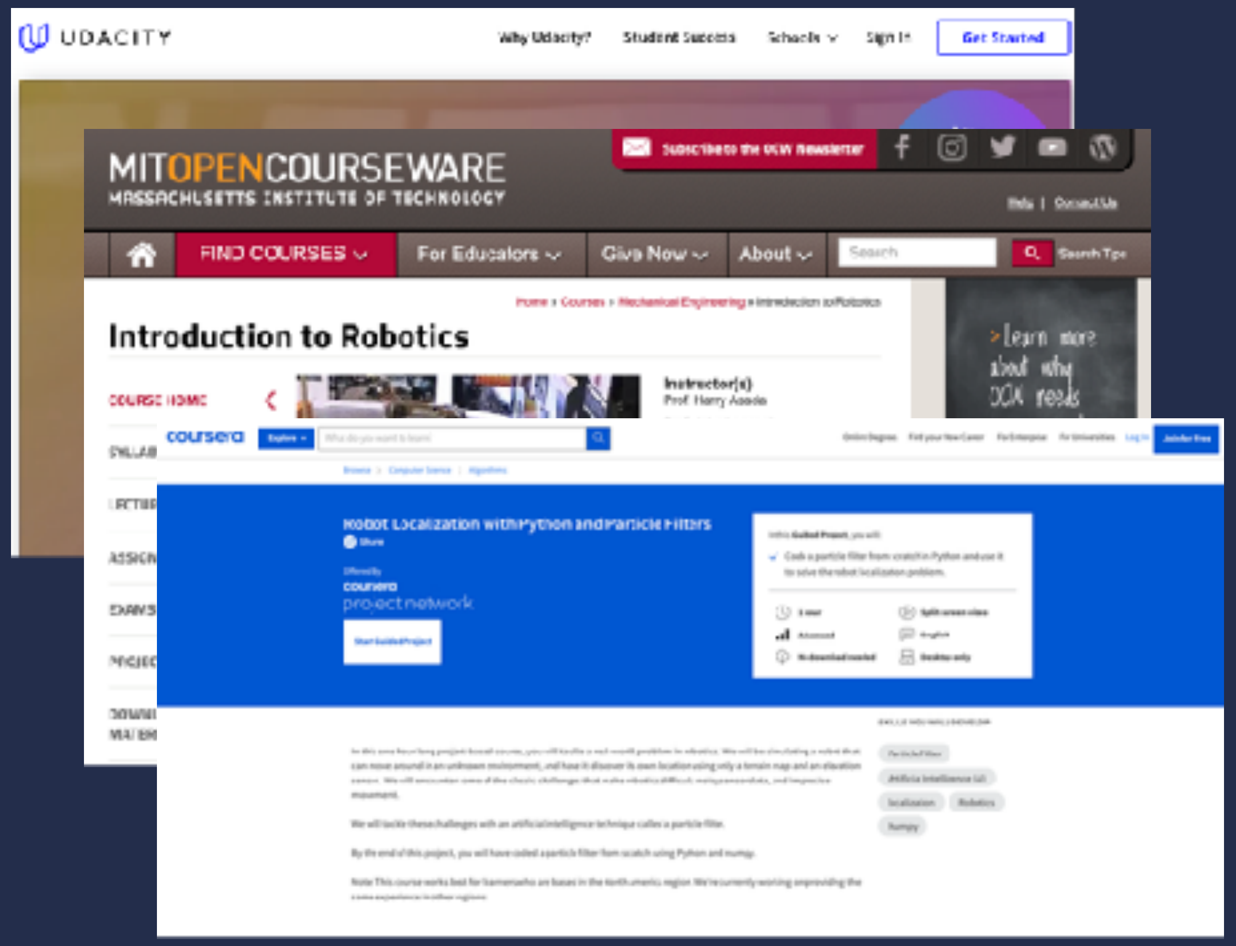
CPS/Embedded Systems Courses



Tends to focus on particular aspects of robot pipeline.

Misses opportunities to discuss broader crosscutting issues.

Massive Open Online Courses

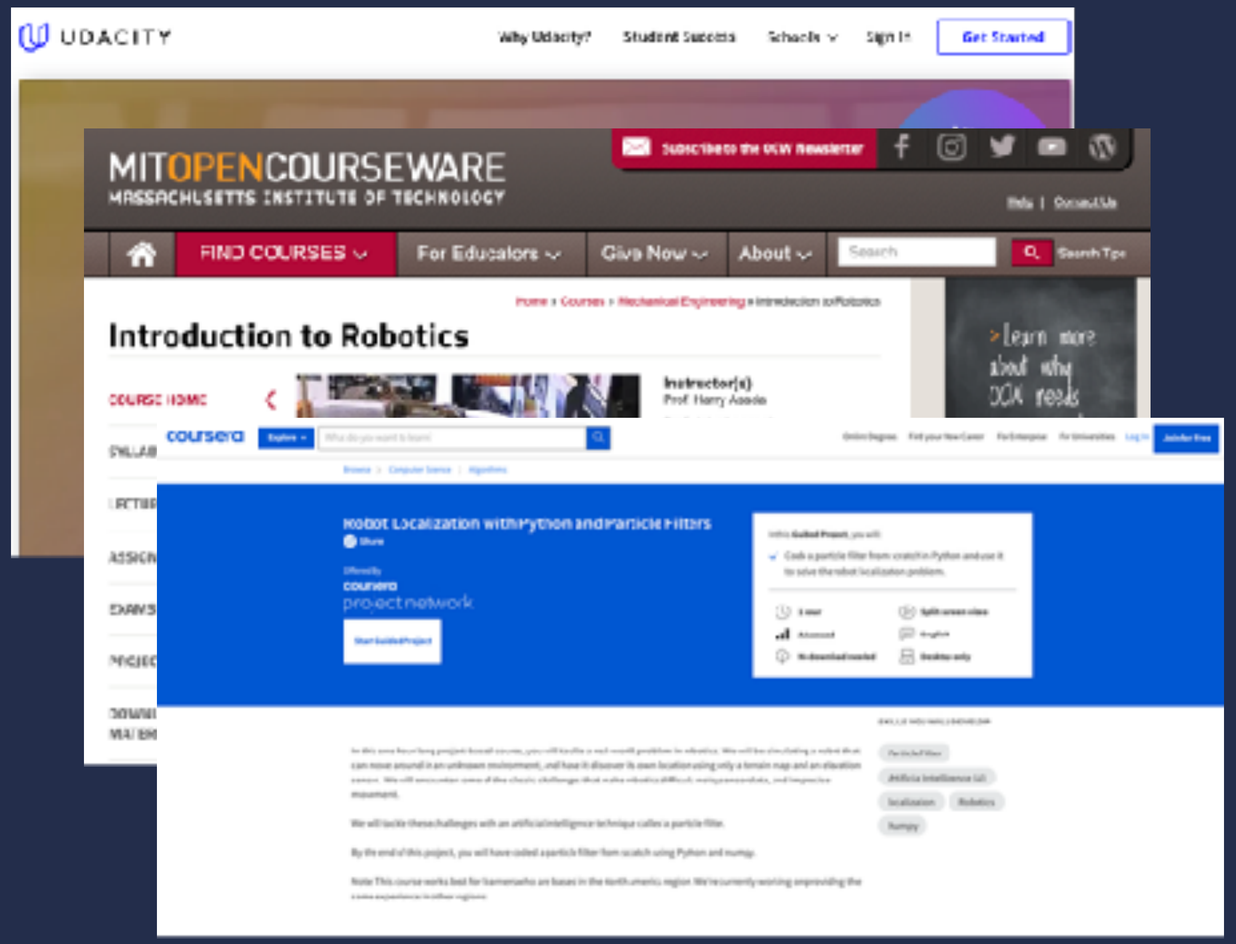


Traditional SE Courses

Traditional CS Curricula

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

Massive Open Online Courses



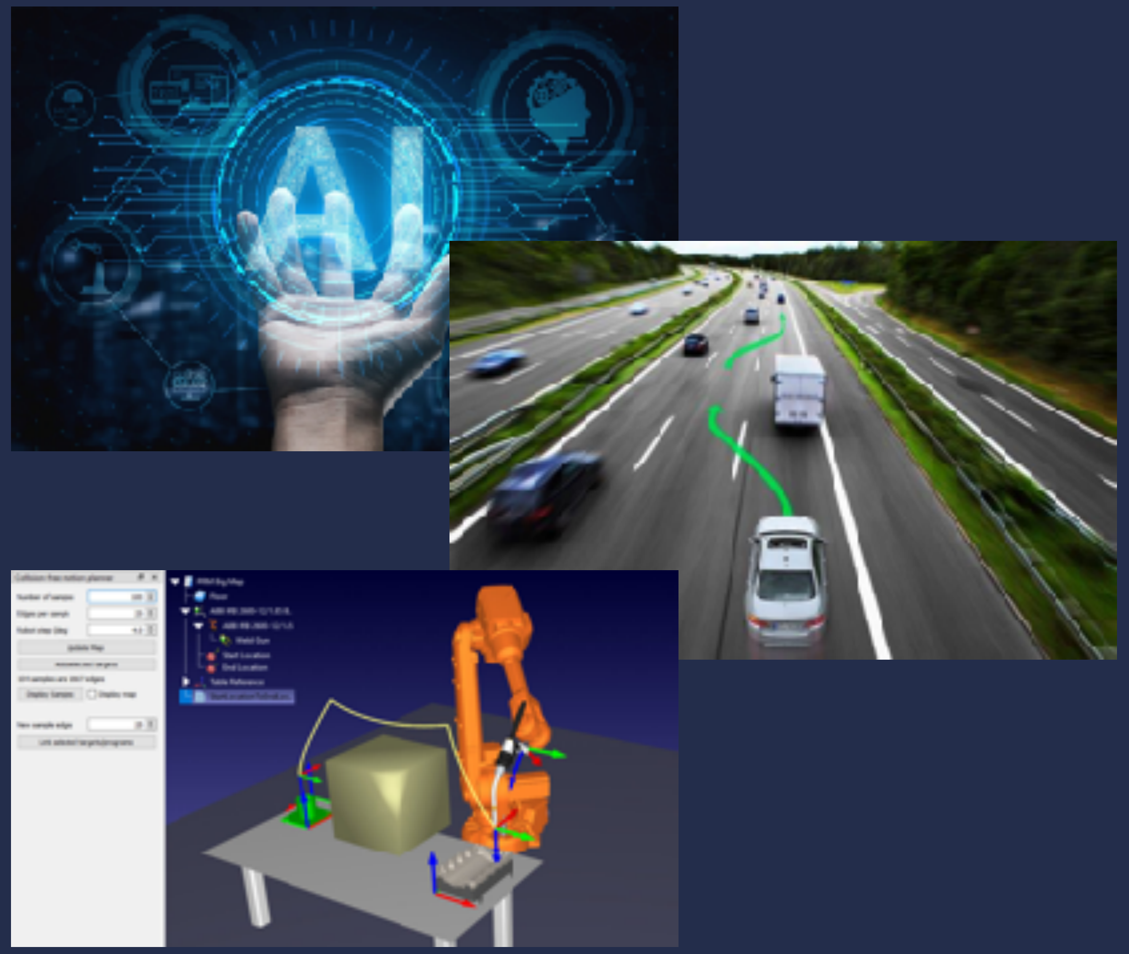
Aims for a breadth of student applicants.

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:

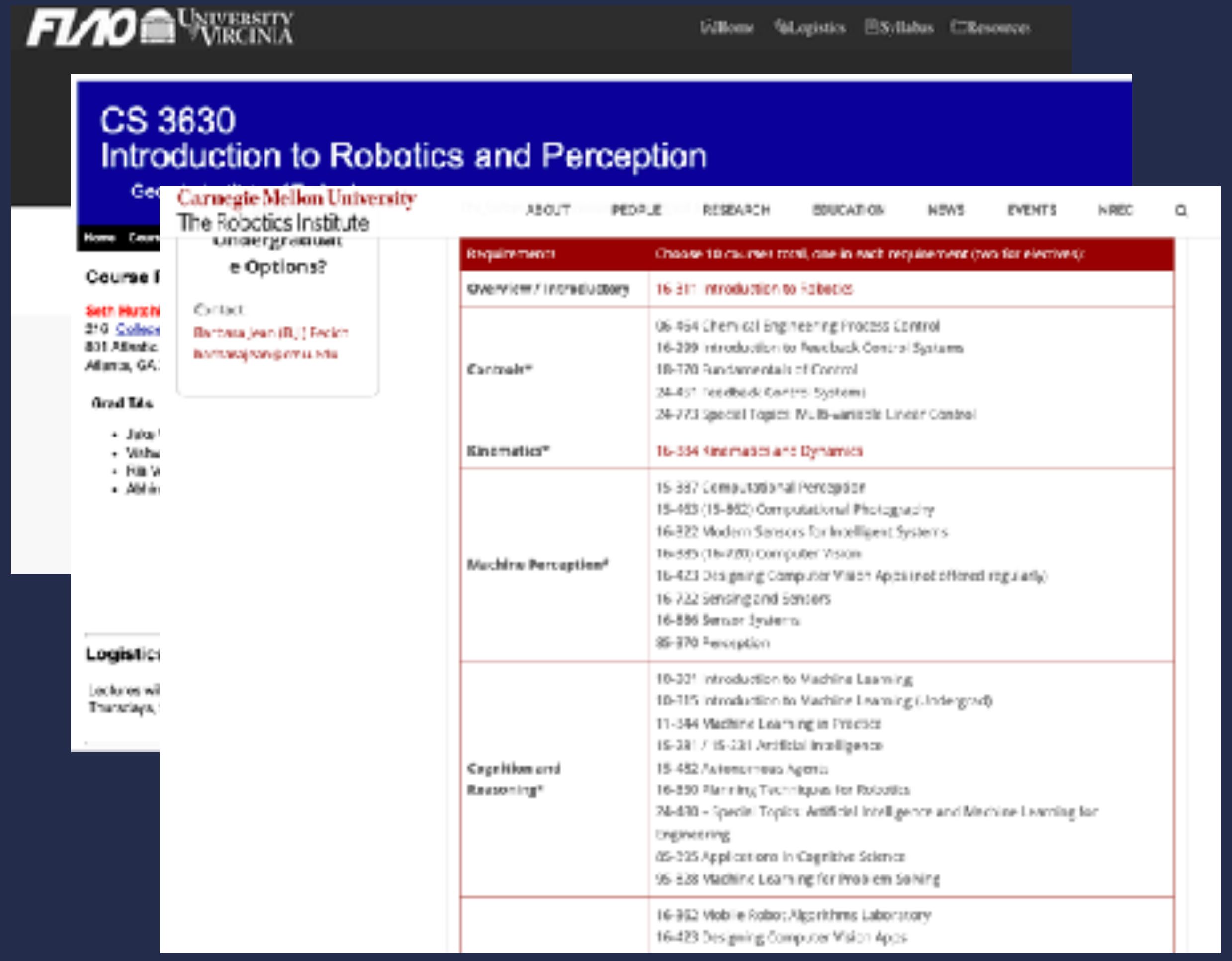
Specialized Graduate-Level Courses



Overlooks that robotics heavily relies on software.

Assumes students are familiar with software engineering.

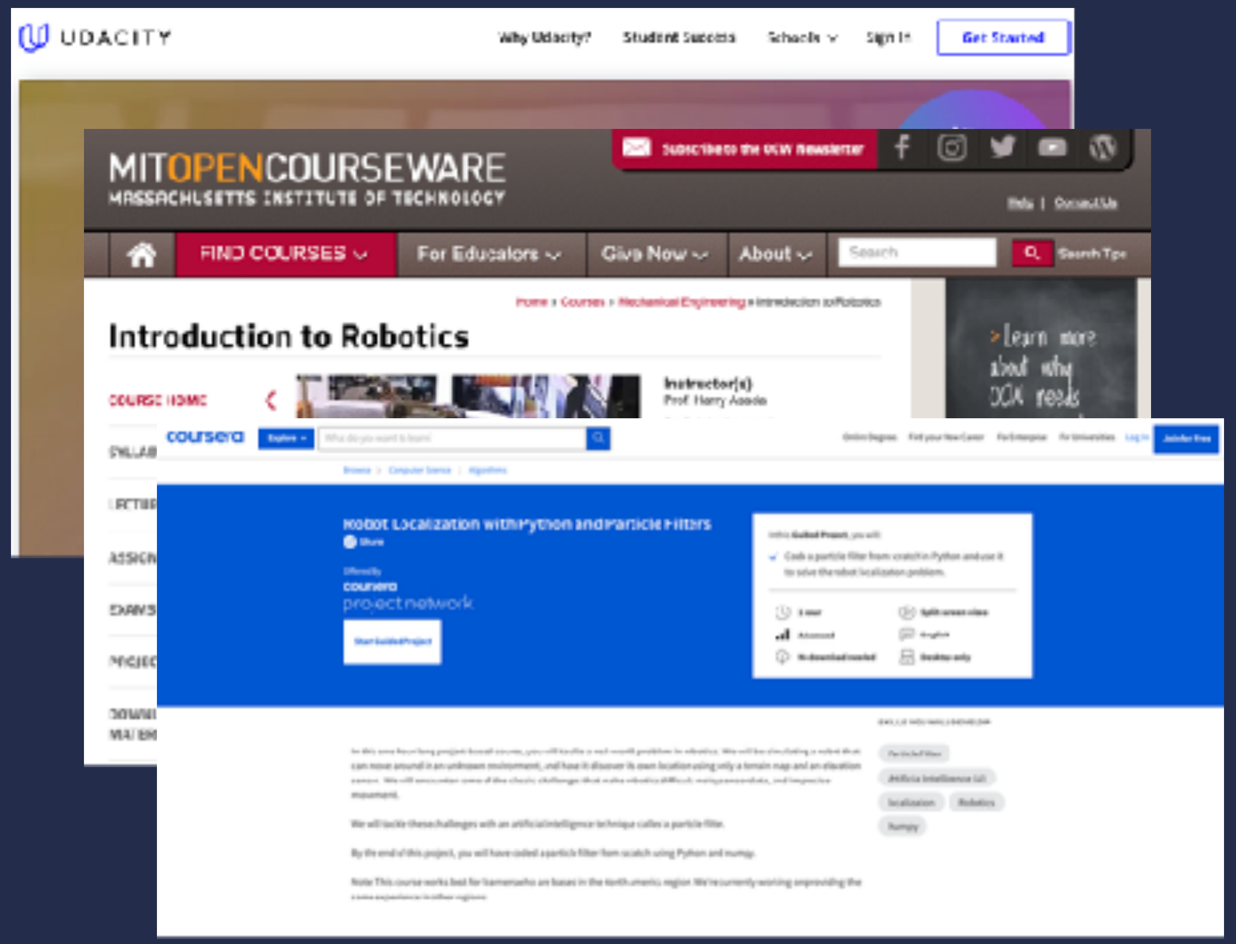
CPS/Embedded Systems Courses



Tends to focus on particular aspects of robot pipeline.

Misses opportunities to discuss broader crosscutting issues.

Massive Open Online Courses



Aims for a breadth of student applicants.

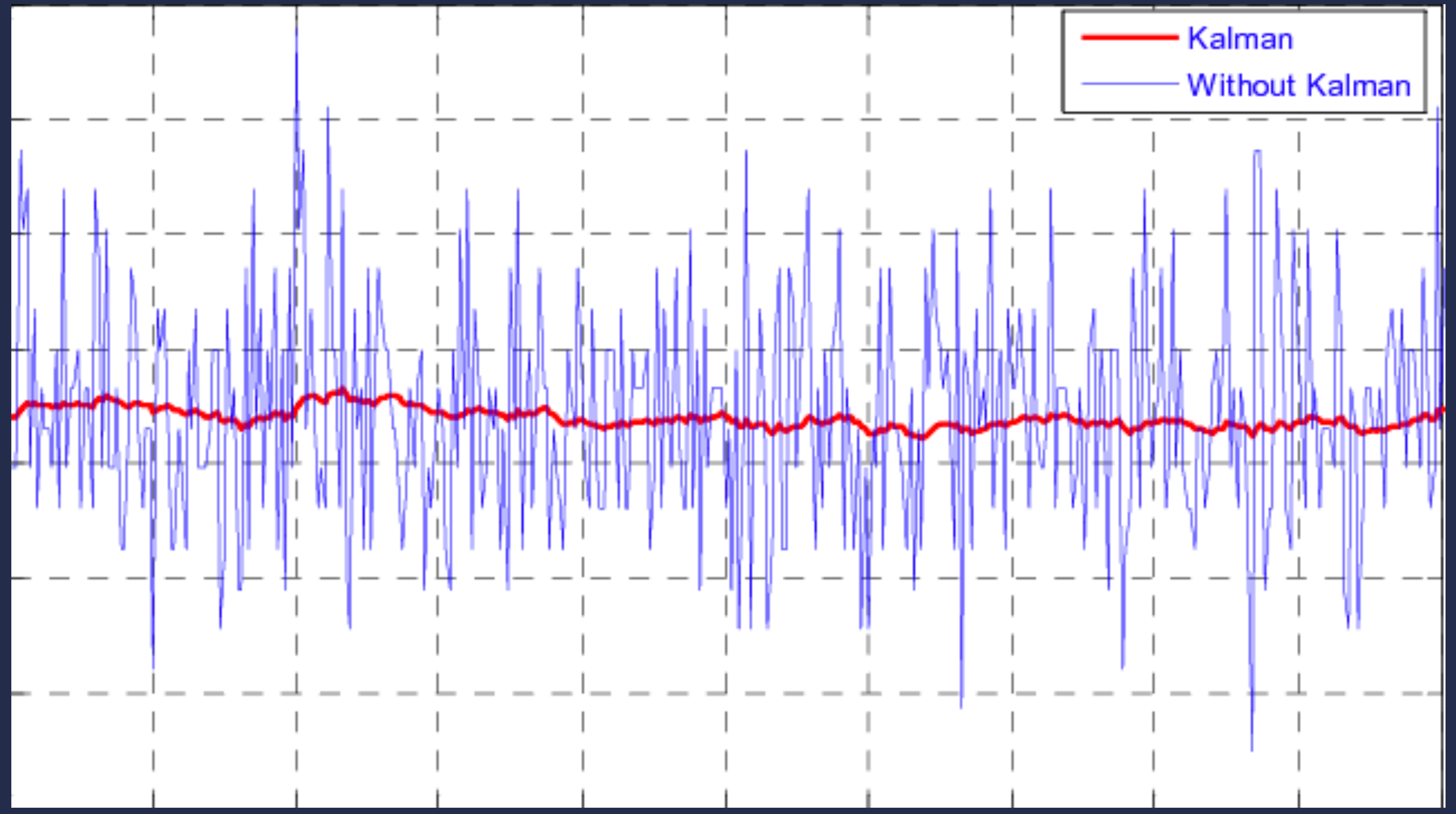
No prerequisites resulting in students that may lack fundamental software engineering principles.

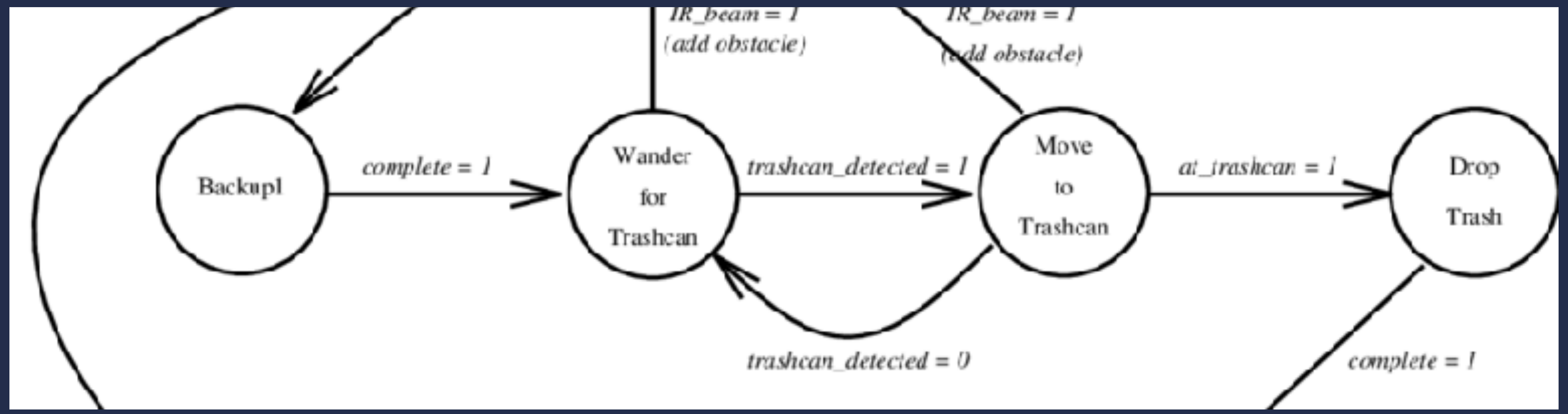
Traditional SE Courses

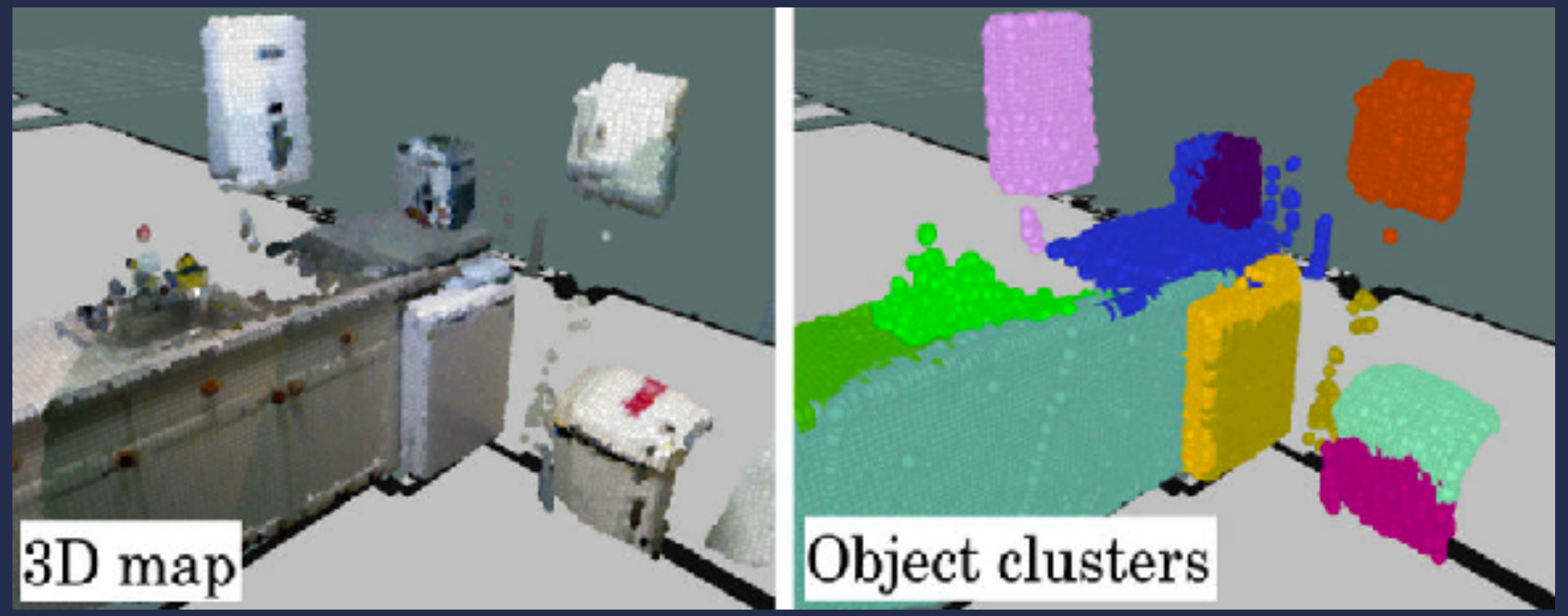
Traditional CS Curricula

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

Traditional
SE Courses







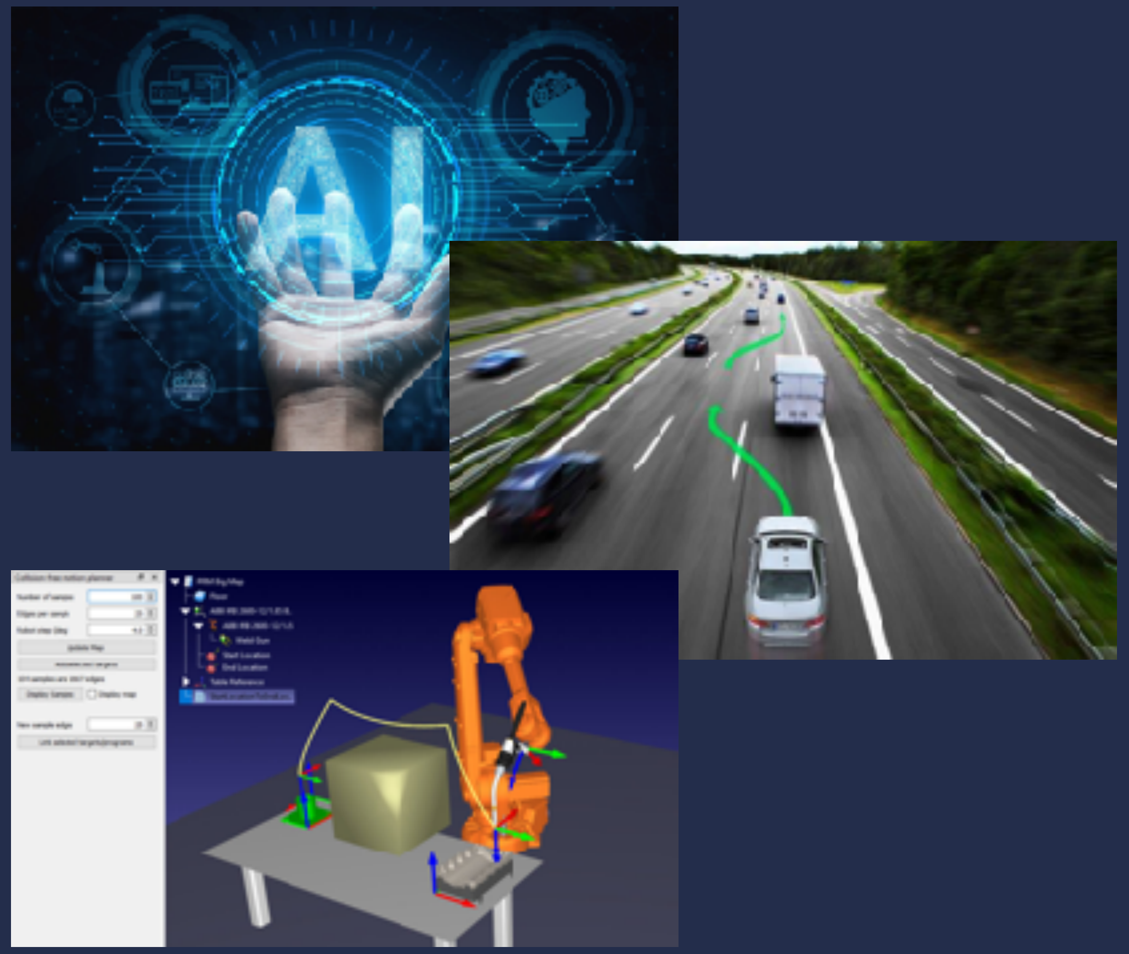
Does not handle aspects specific to robotic systems.

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:

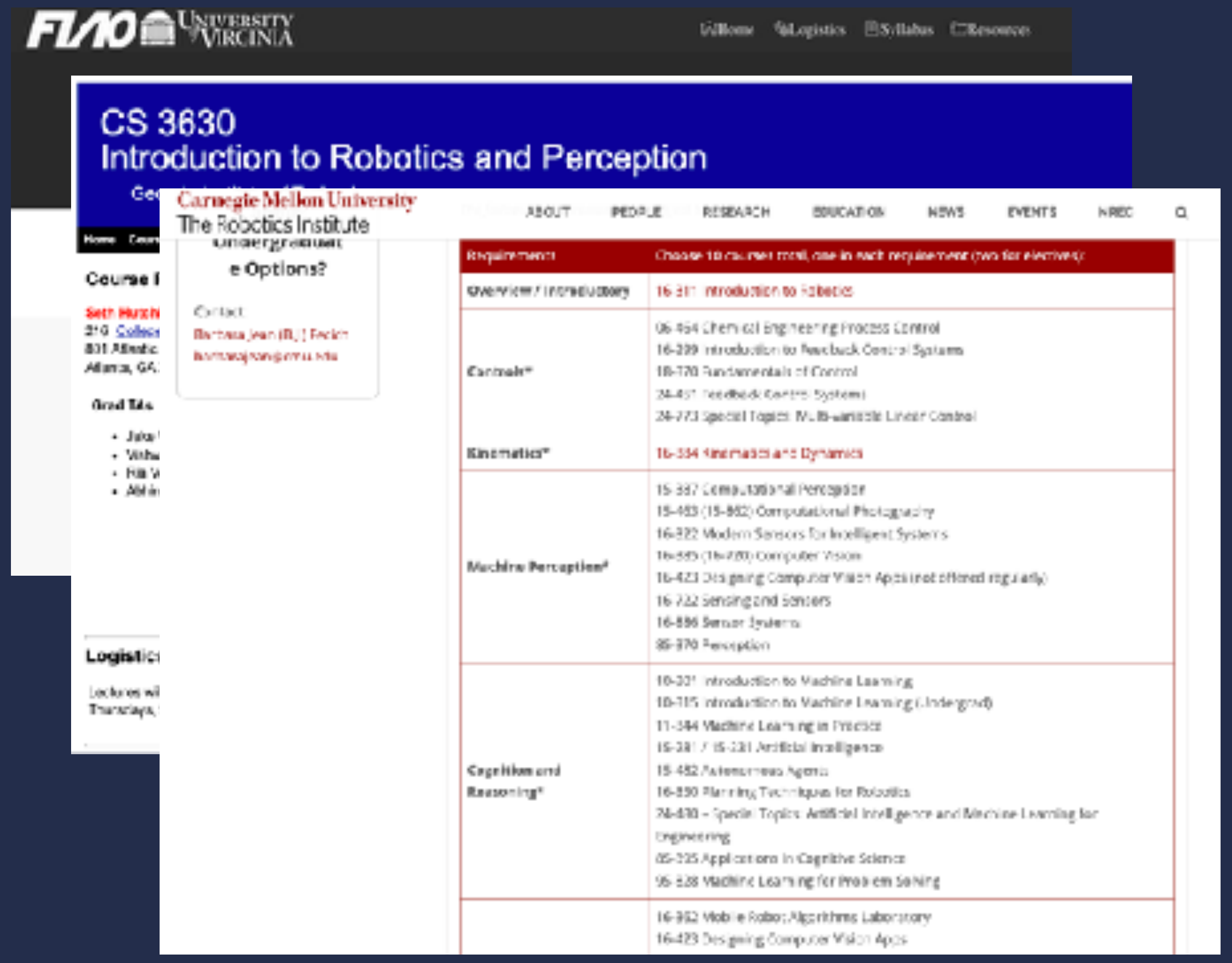
Specialized Graduate-Level Courses



Overlooks that robotics heavily relies on software.

Assumes students are familiar with software engineering.

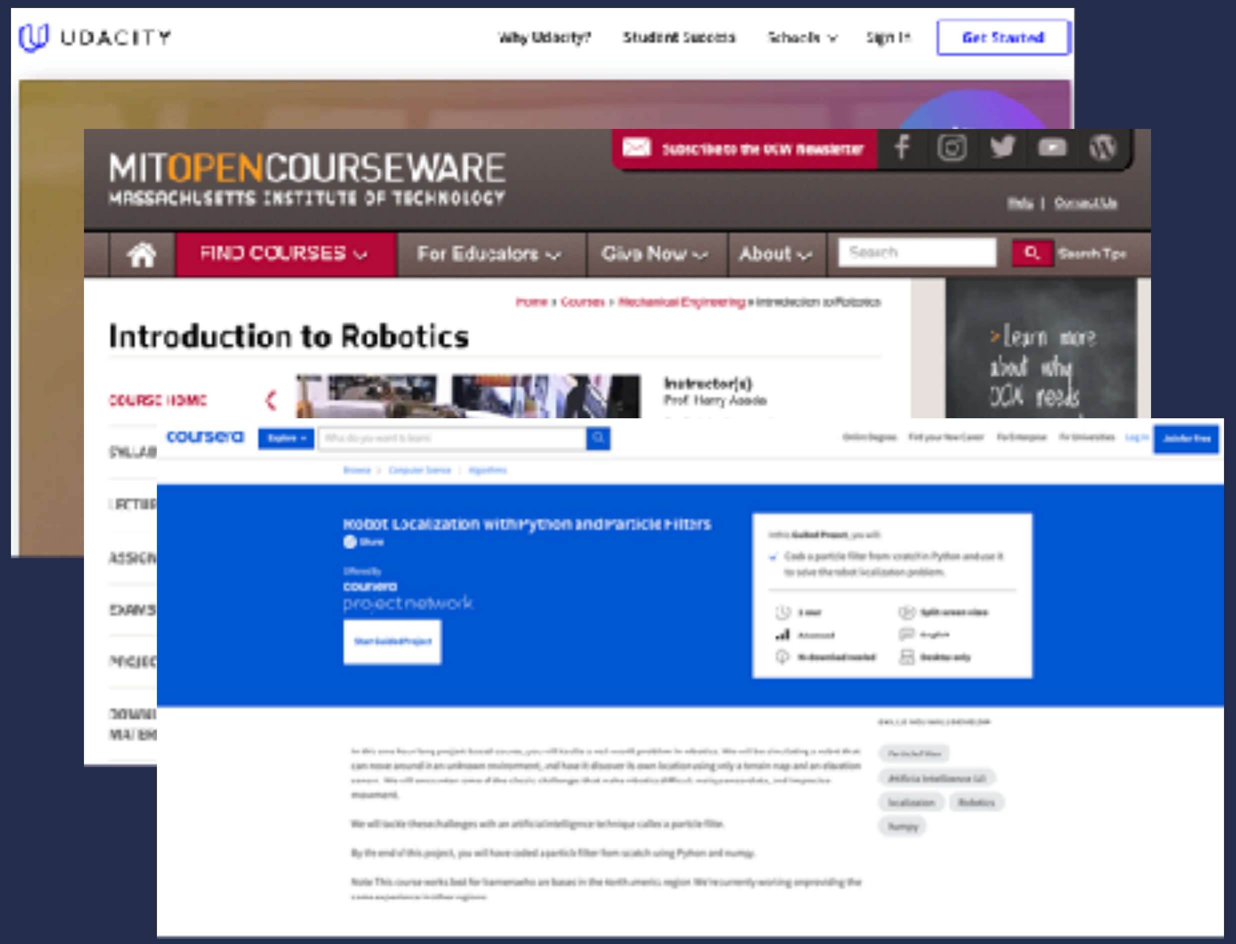
CPS/Embedded Systems Courses



Tends to focus on particular aspects of robot pipeline.

Misses opportunities to discuss broader crosscutting issues.

Massive Open Online Courses



Aims for a breadth of student applicants.

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

Traditional SE Courses

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

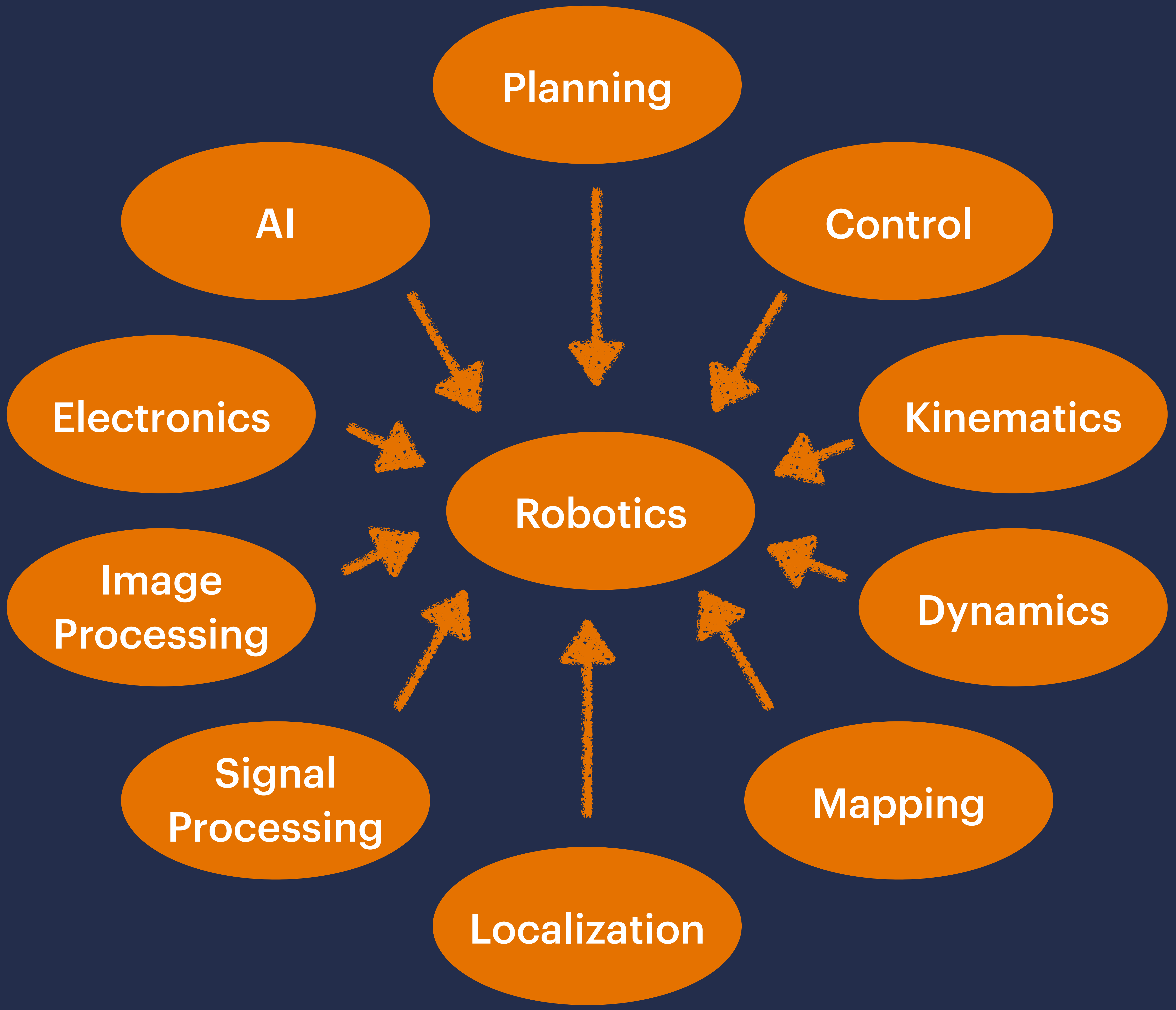
Challenges

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



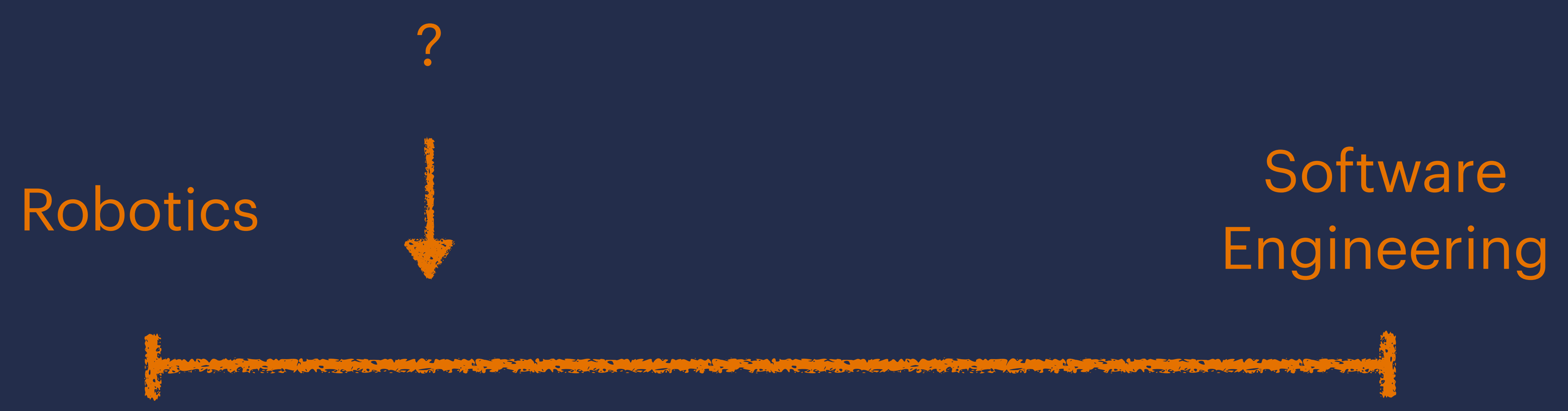
Challenges

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



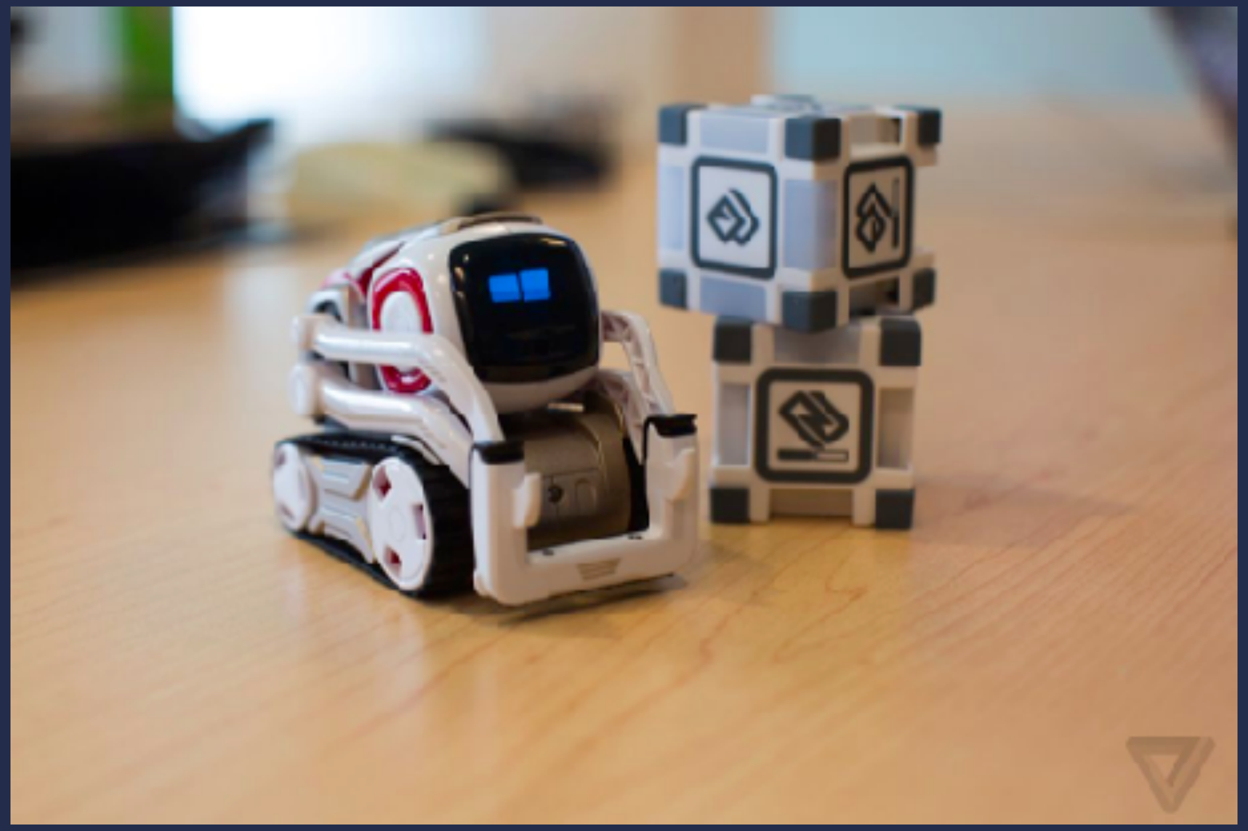
Challenges

1. Multidisciplinary and rapidly expanding field

2. How to distribute the emphasis between robotics and software engineering

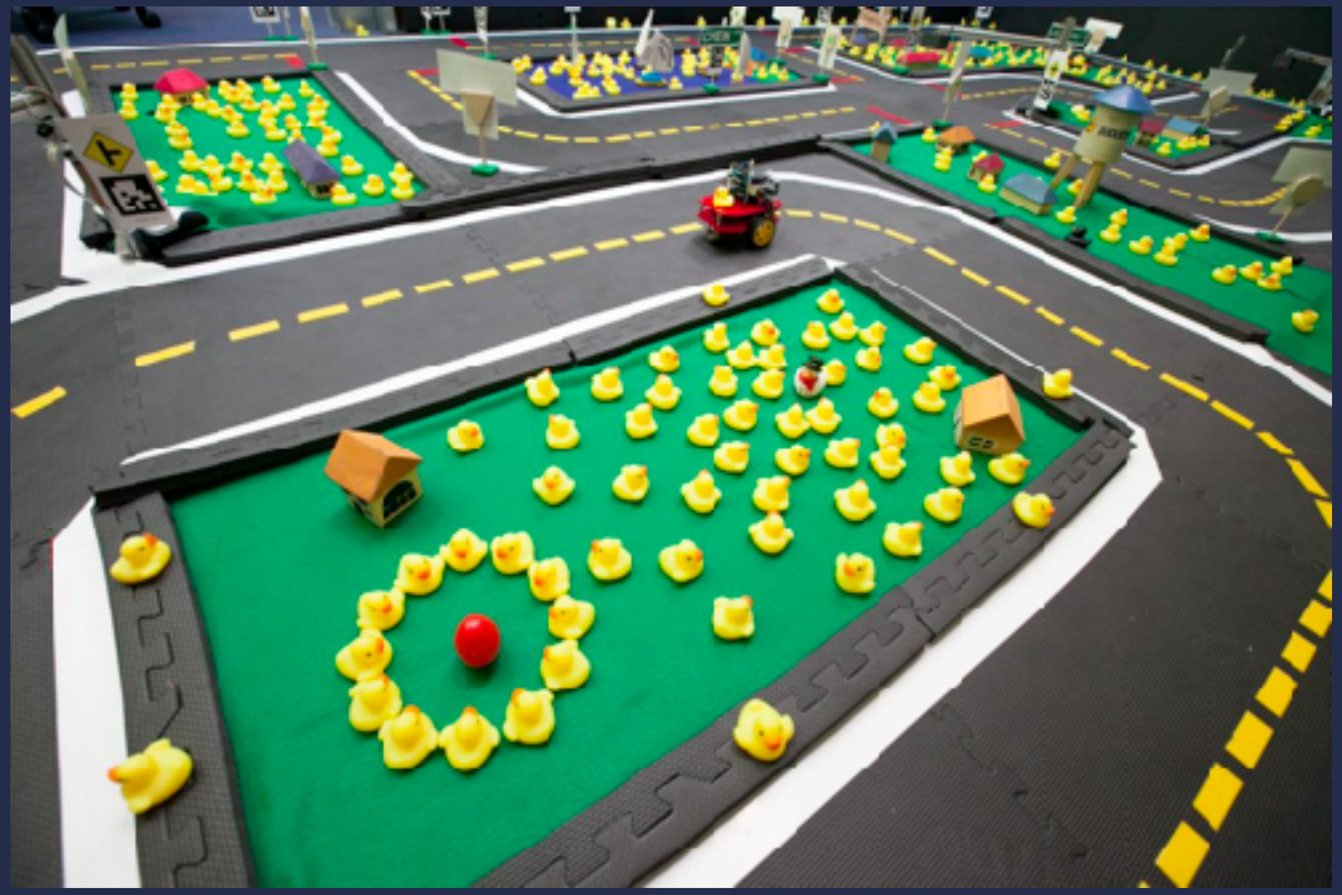
3. No available integrated platform

4. Robotics courses can require significant upfront investment in equipment



Anki Cosmo

Duckytown



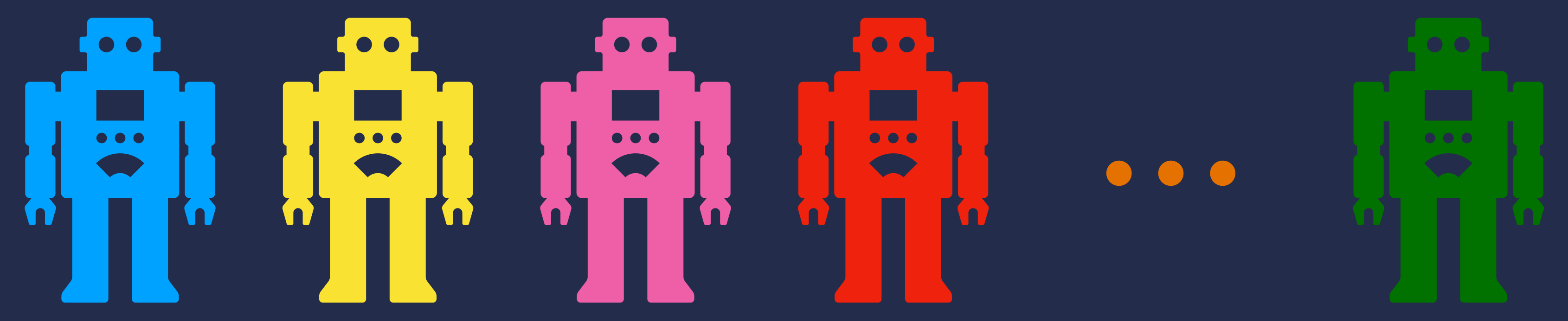
Challenges

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

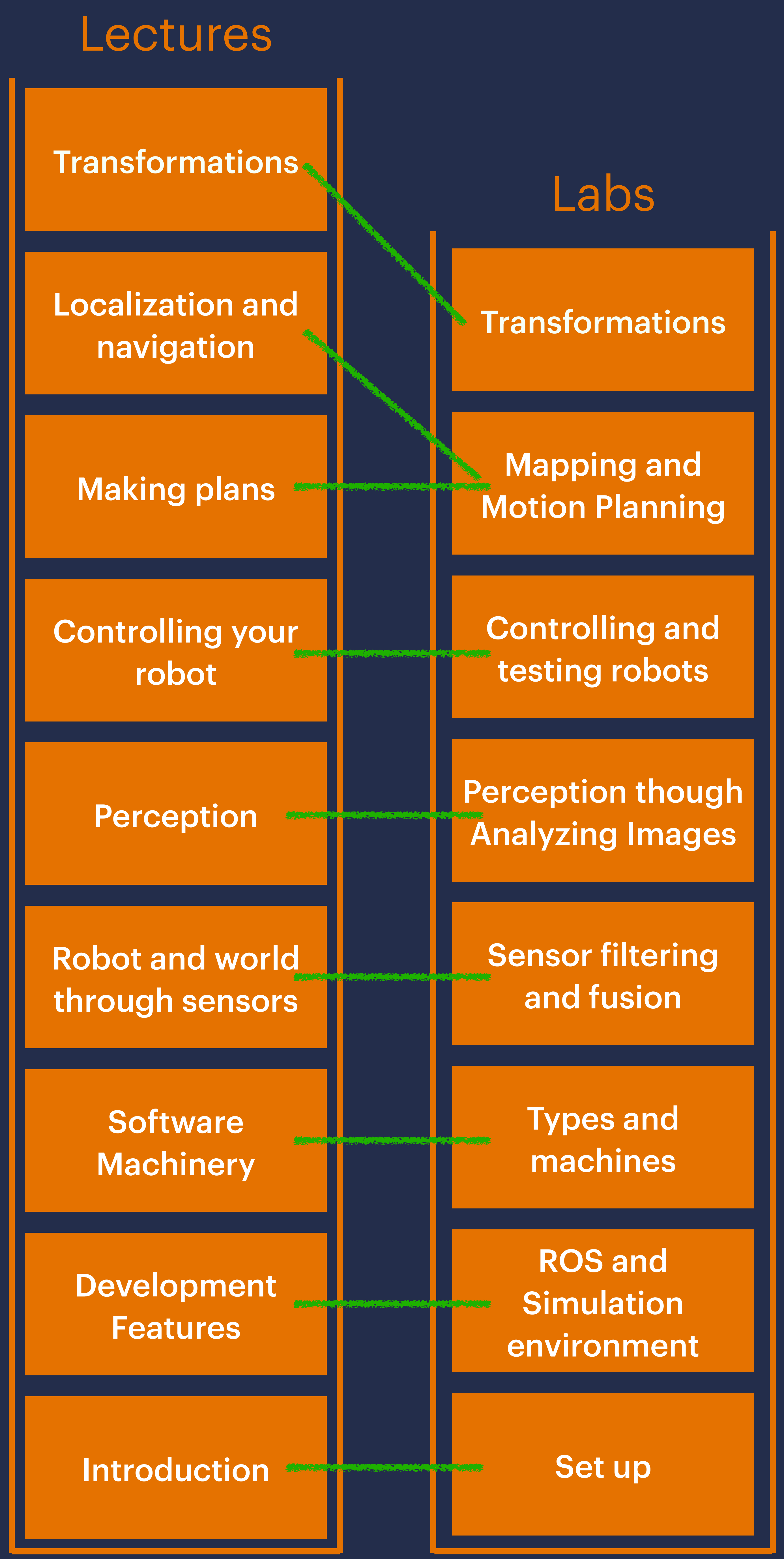
No available integrated platform

	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



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

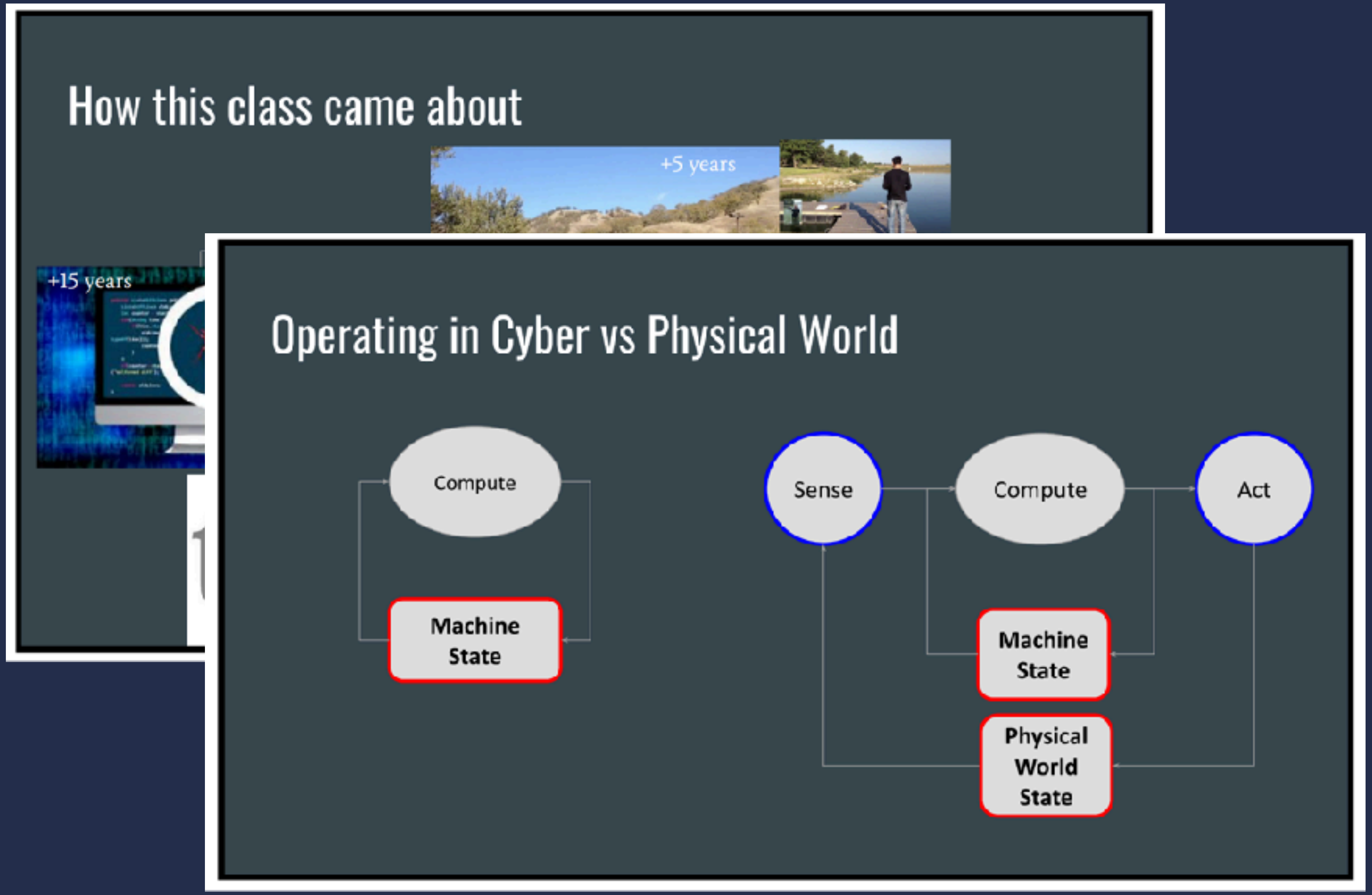
Lecture Lab Pairing

Labs

Lectures

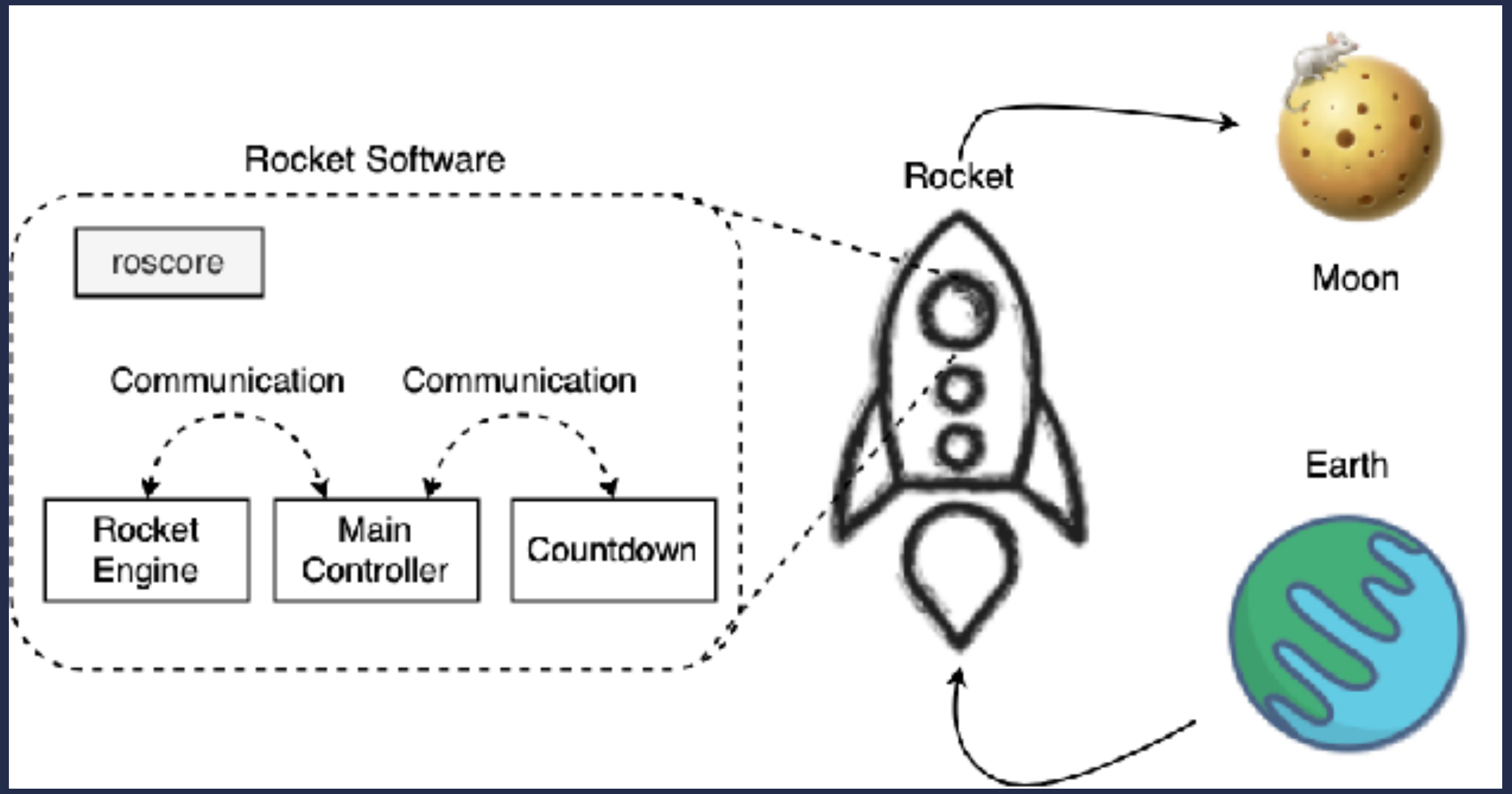
Lecture Lab Pairing

Labs



Introducing robotics

Introduction to ROS



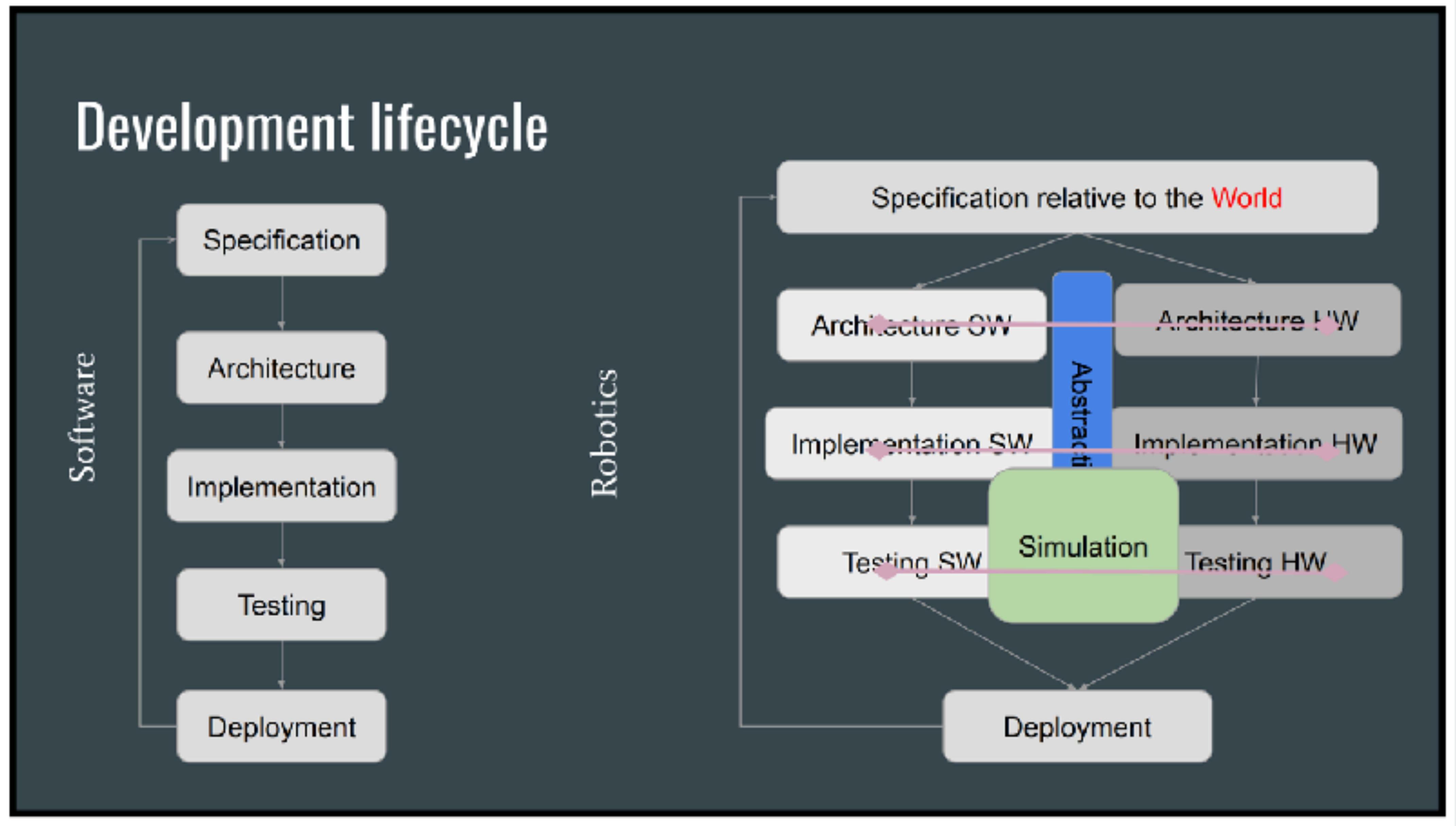
Introduction

Set up

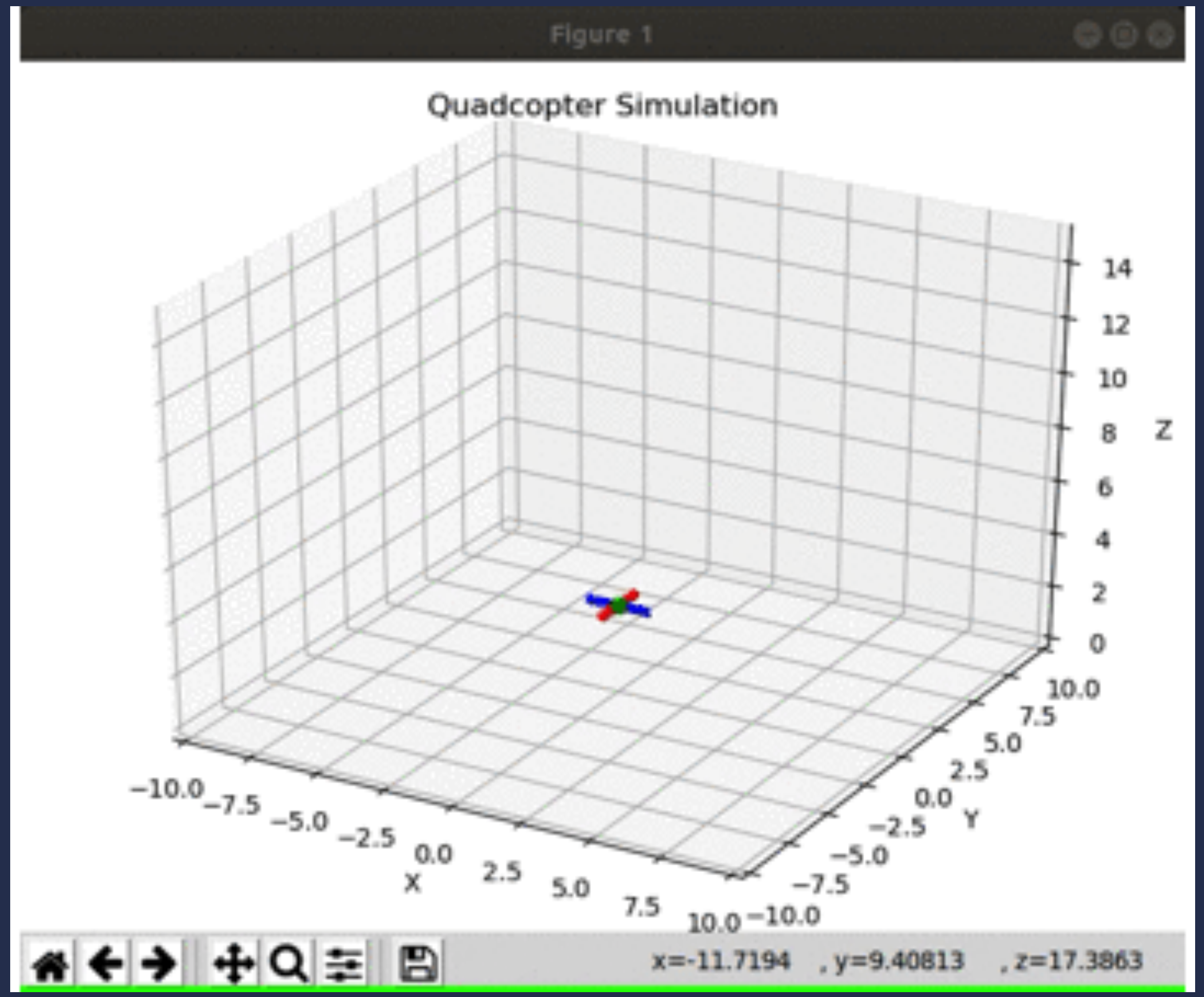
Lecture Lab Pairing

Lectures

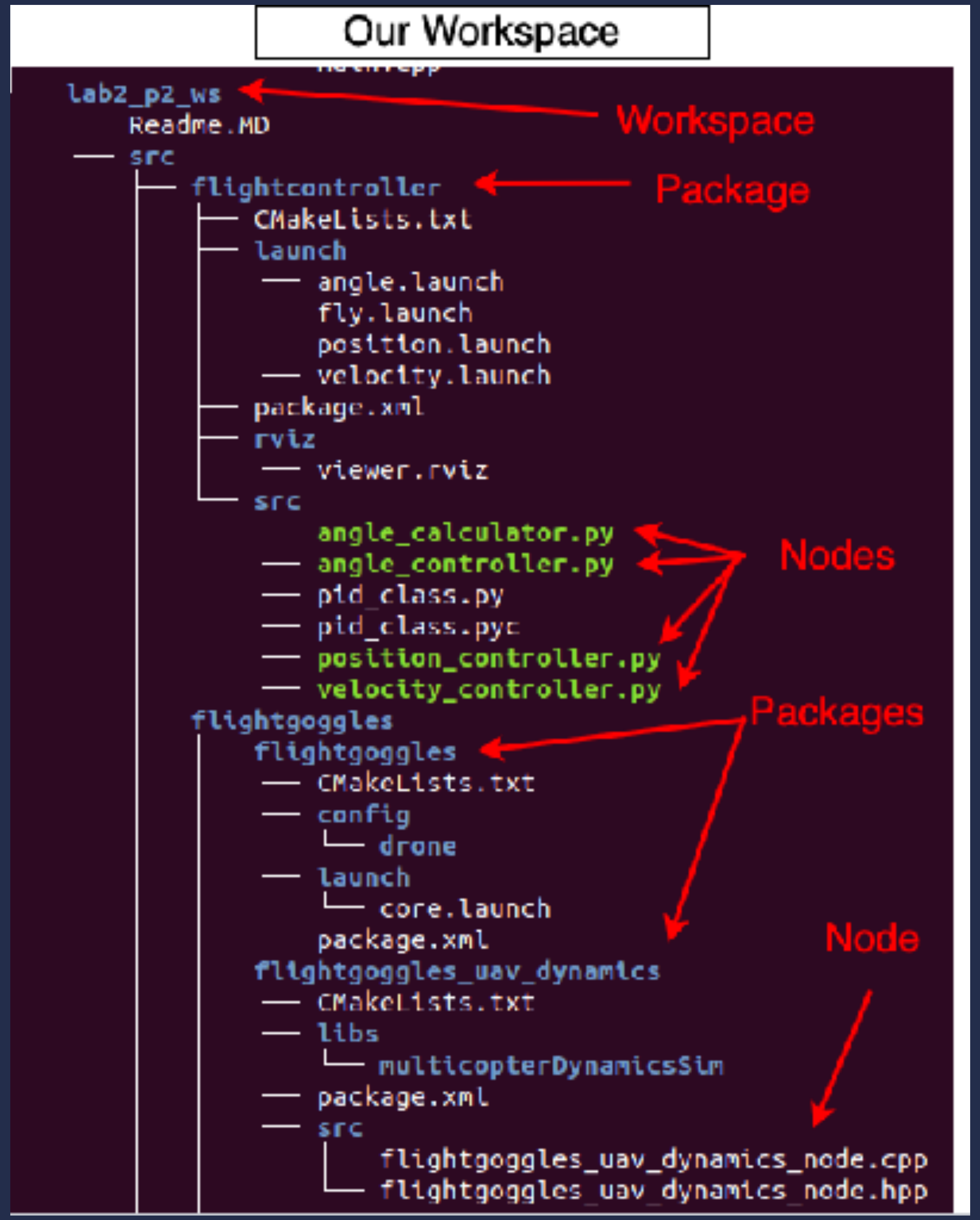
- Development Features
- Introduction



Describing the development lifecycle



Introducing the simulation

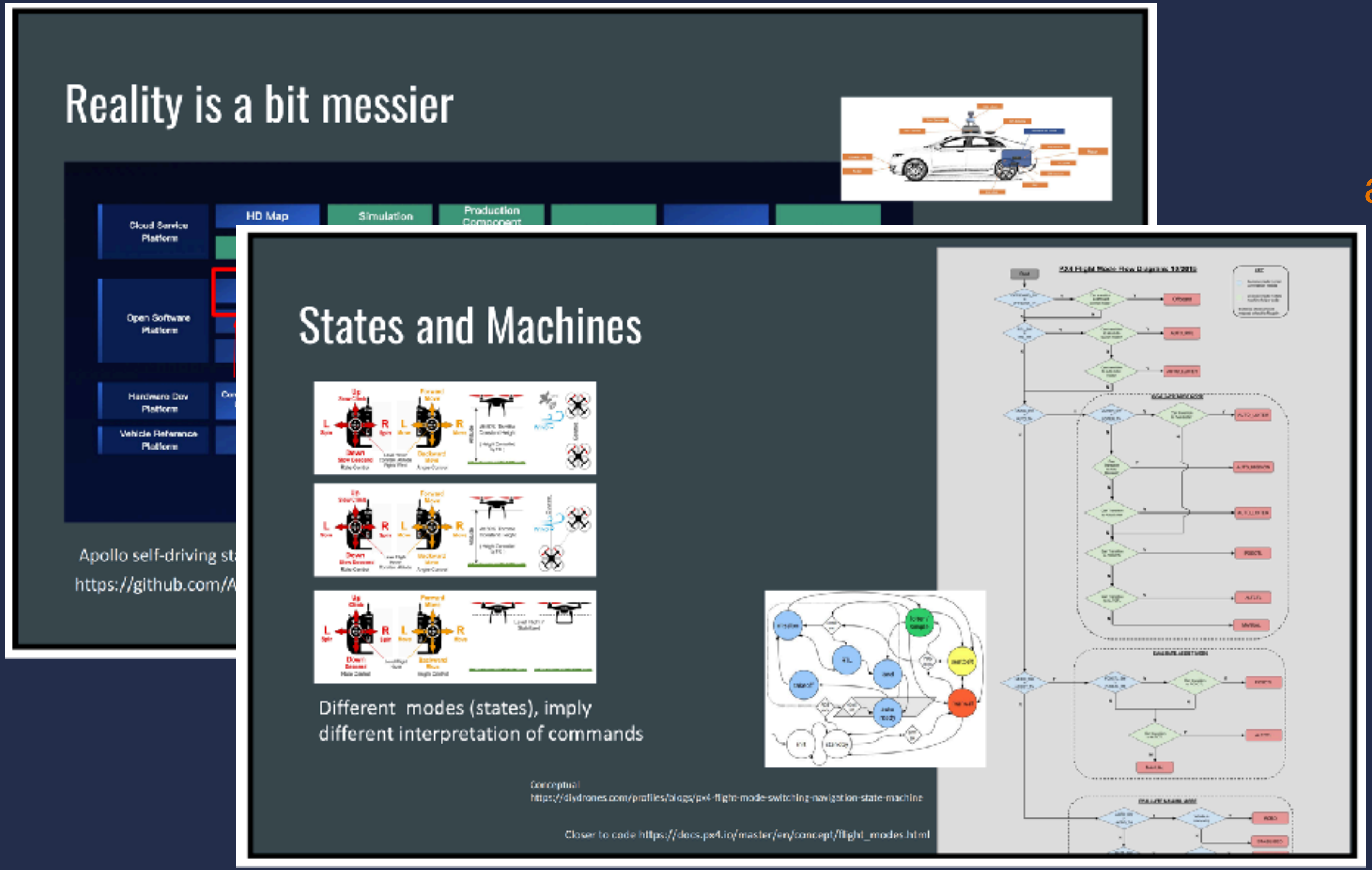


Labs

- ROS and Simulation environment
- Set up

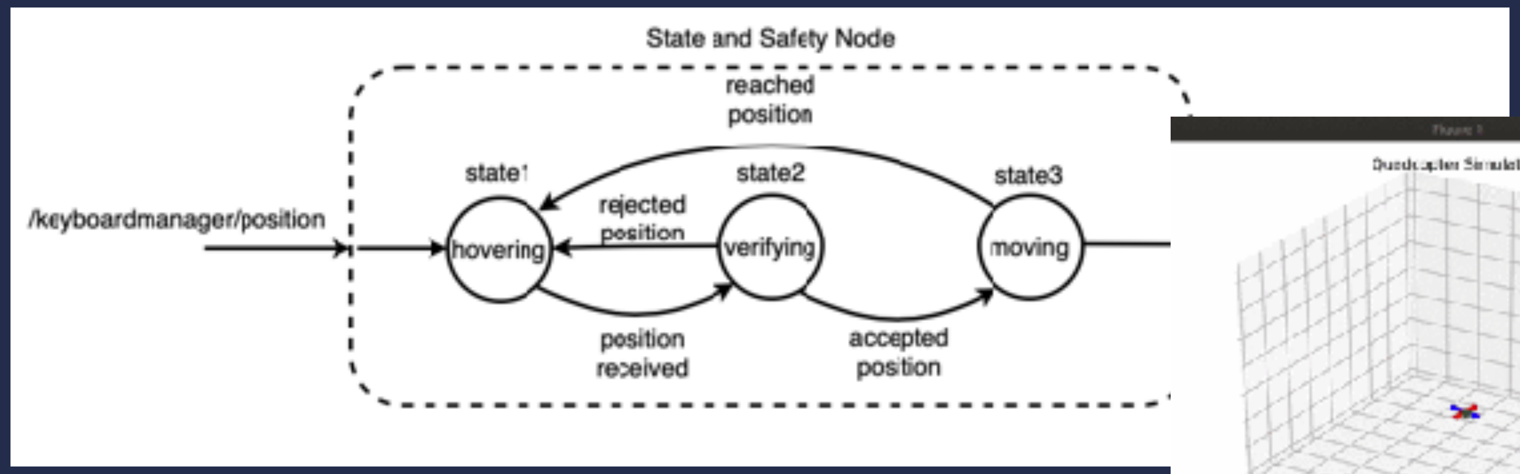
Lecture Lab Pairing

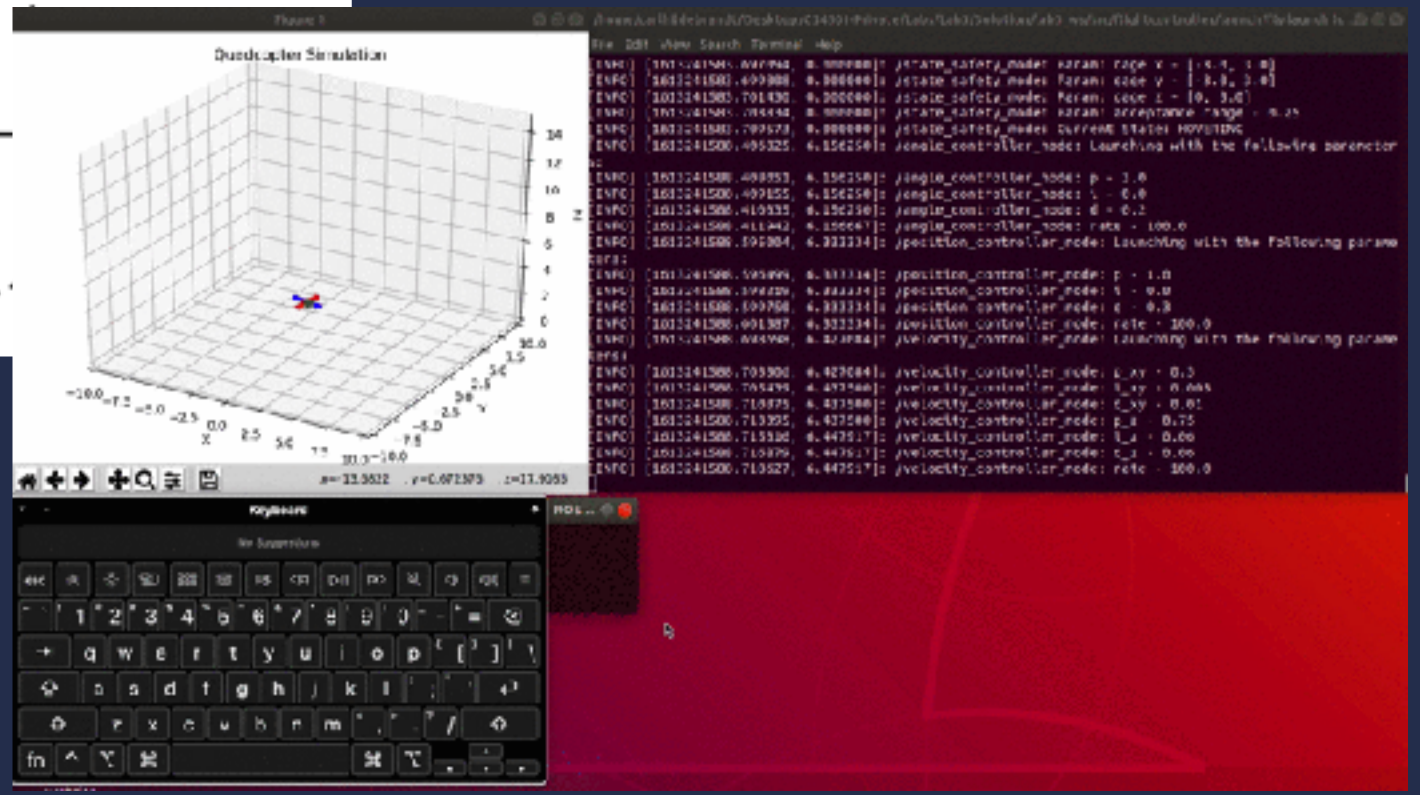
Lectures



Describing how systems are implemented in reality

Introducing state machines





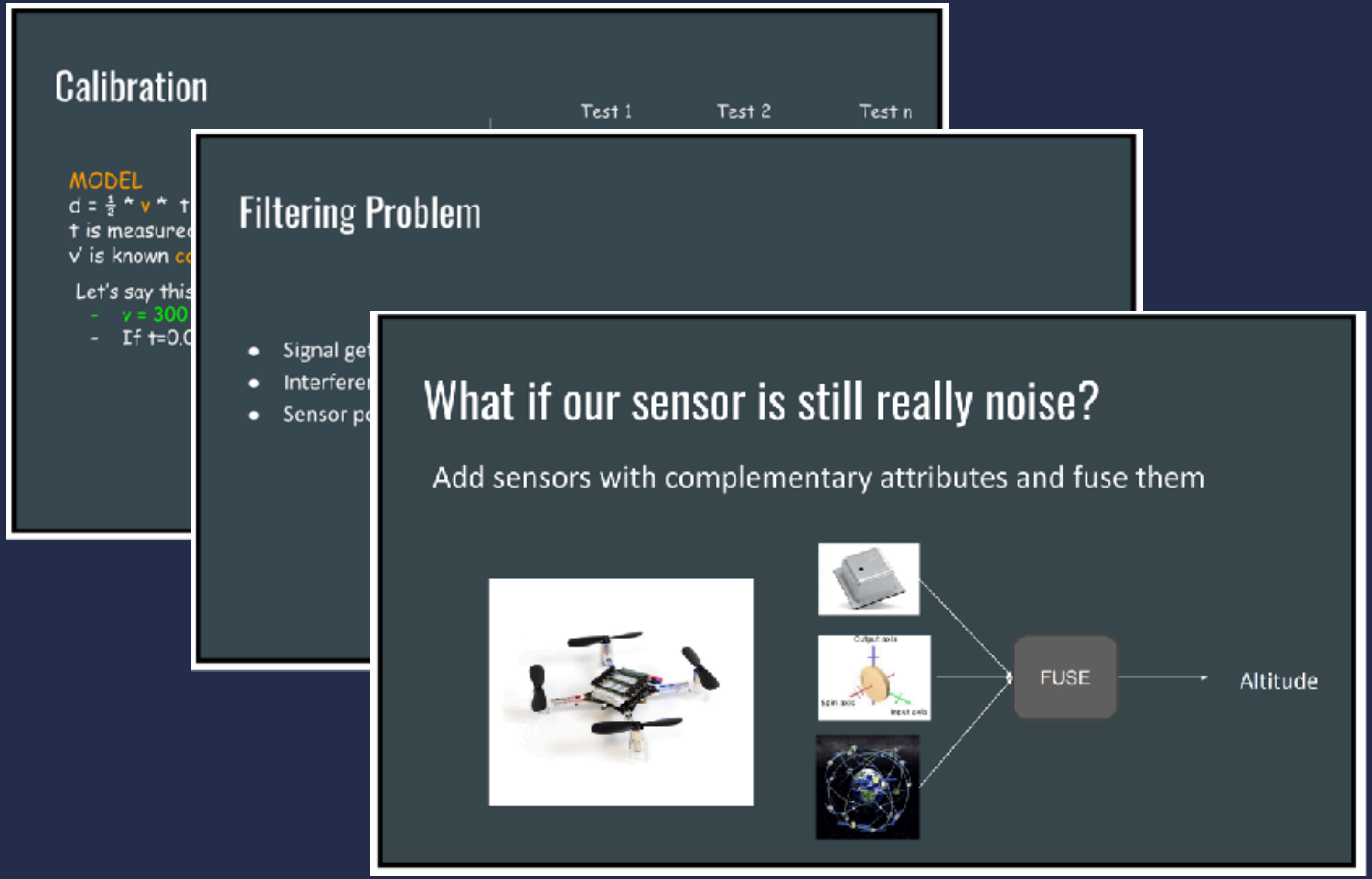
Using state machines to create a keyboard controller

Labs

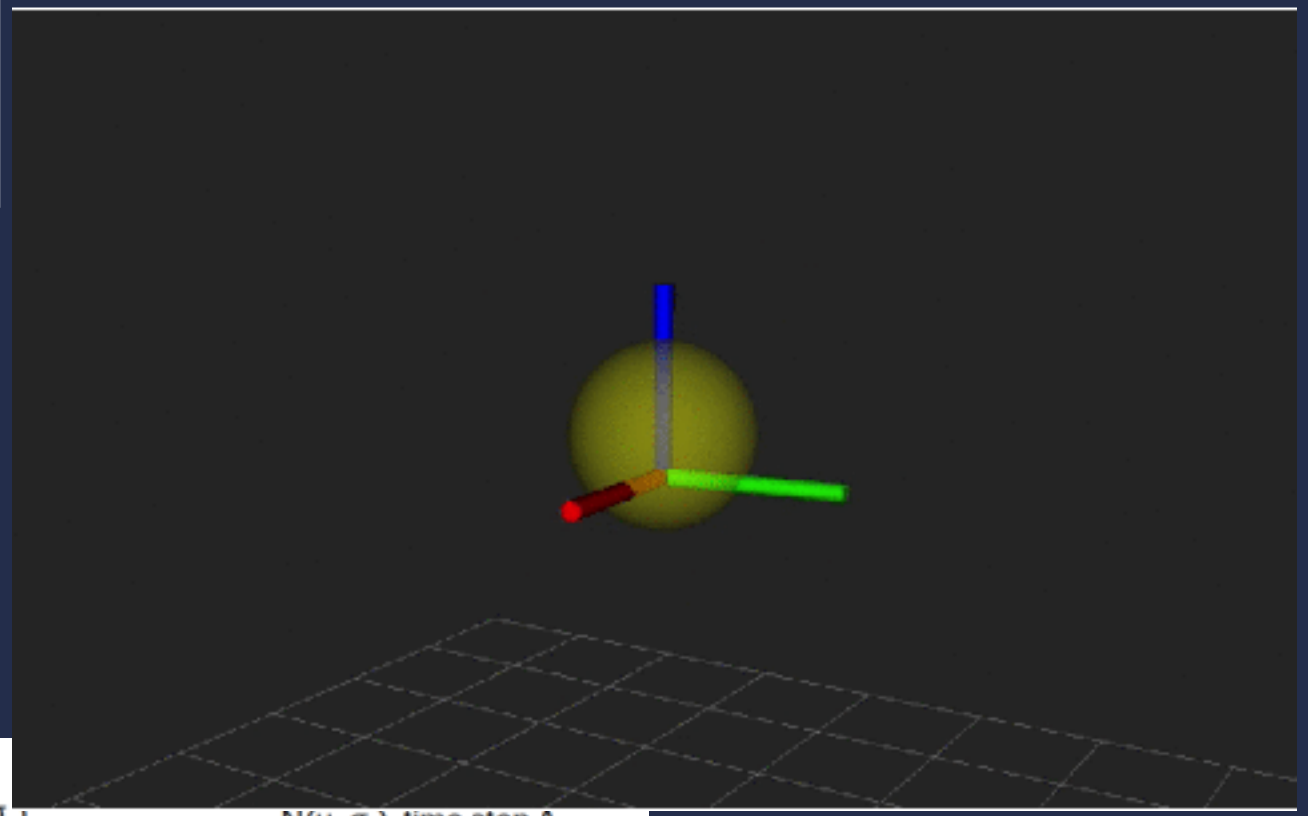
Lectures	Labs
Software Machinery	Types and machines
Development Features	ROS and Simulation environment
Introduction	Set up

Lecture Lab Pairing

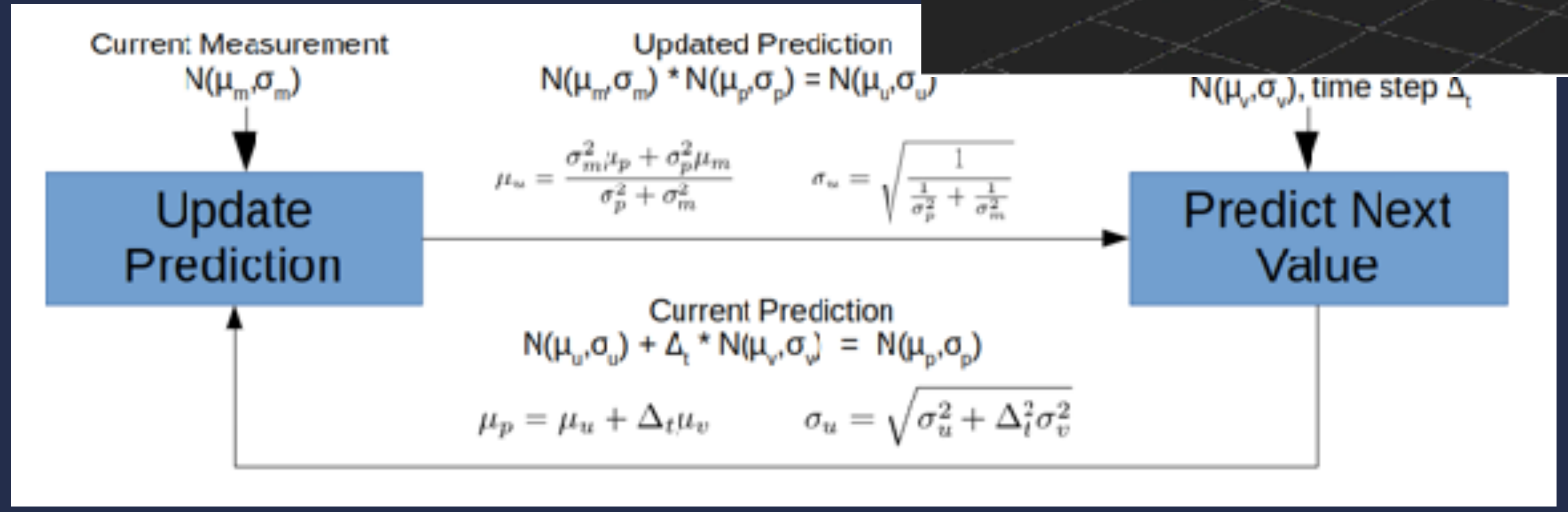
Lectures



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



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



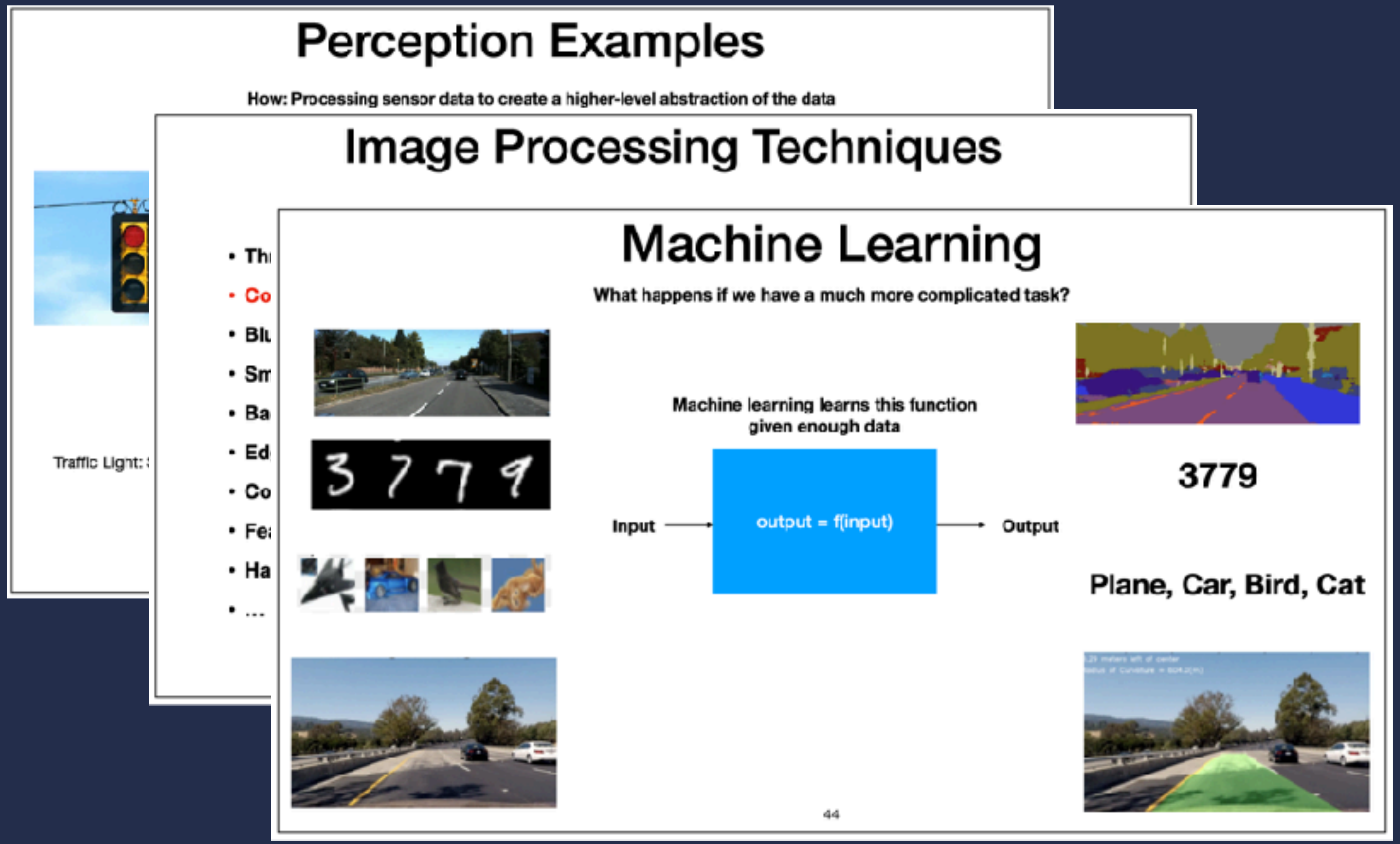
Labs

Lectures	Labs
Robot and world through sensors	Sensor filtering and fusion
Software Machinery	Types and machines
Development Features	ROS and Simulation environment
Introduction	Set up

Lecture Lab Pairing

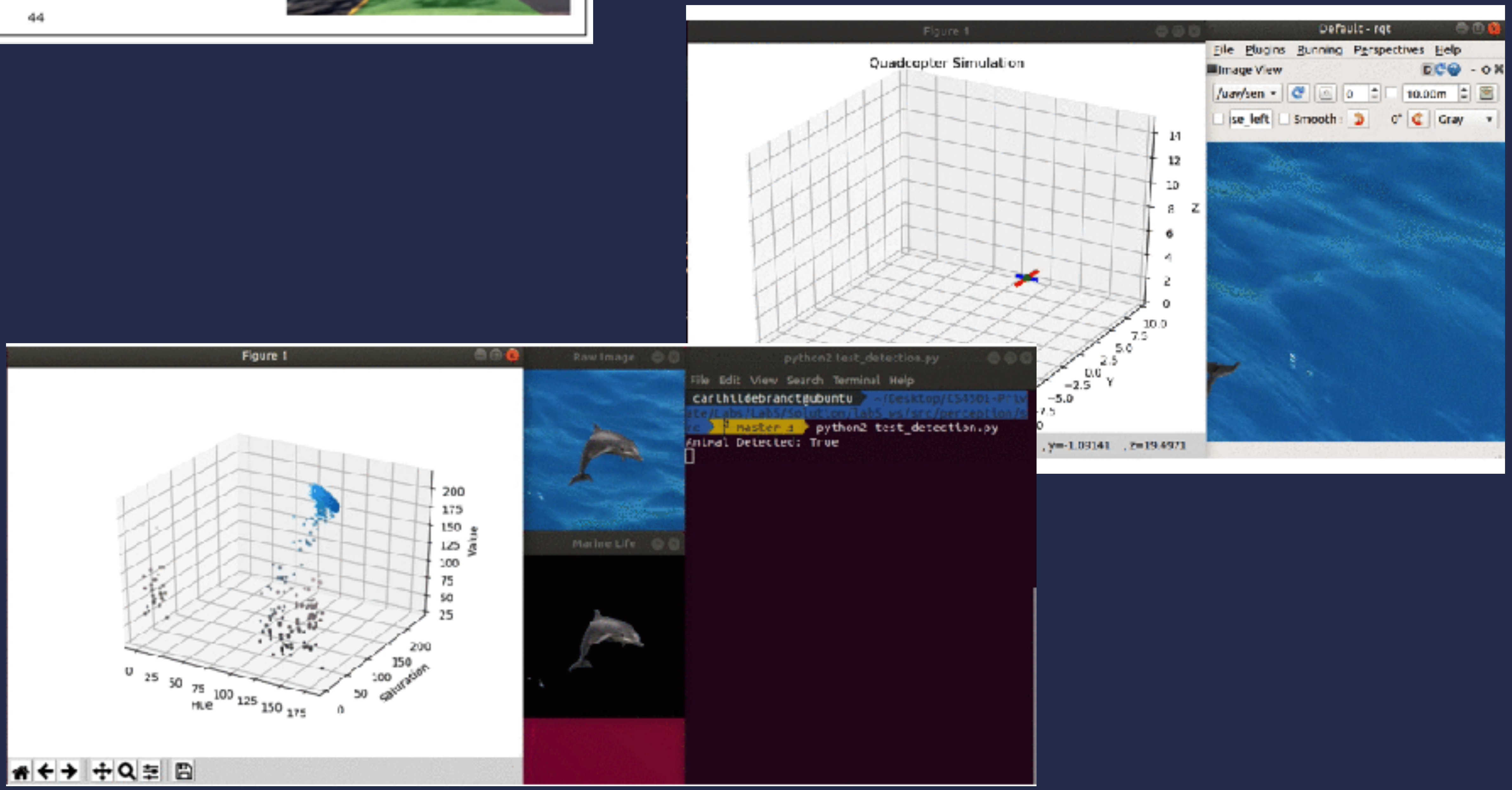
Lectures

- Perception
- Robot and world through sensors
- Software Machinery
- Development Features
- Introduction



Introducing different types of perception

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



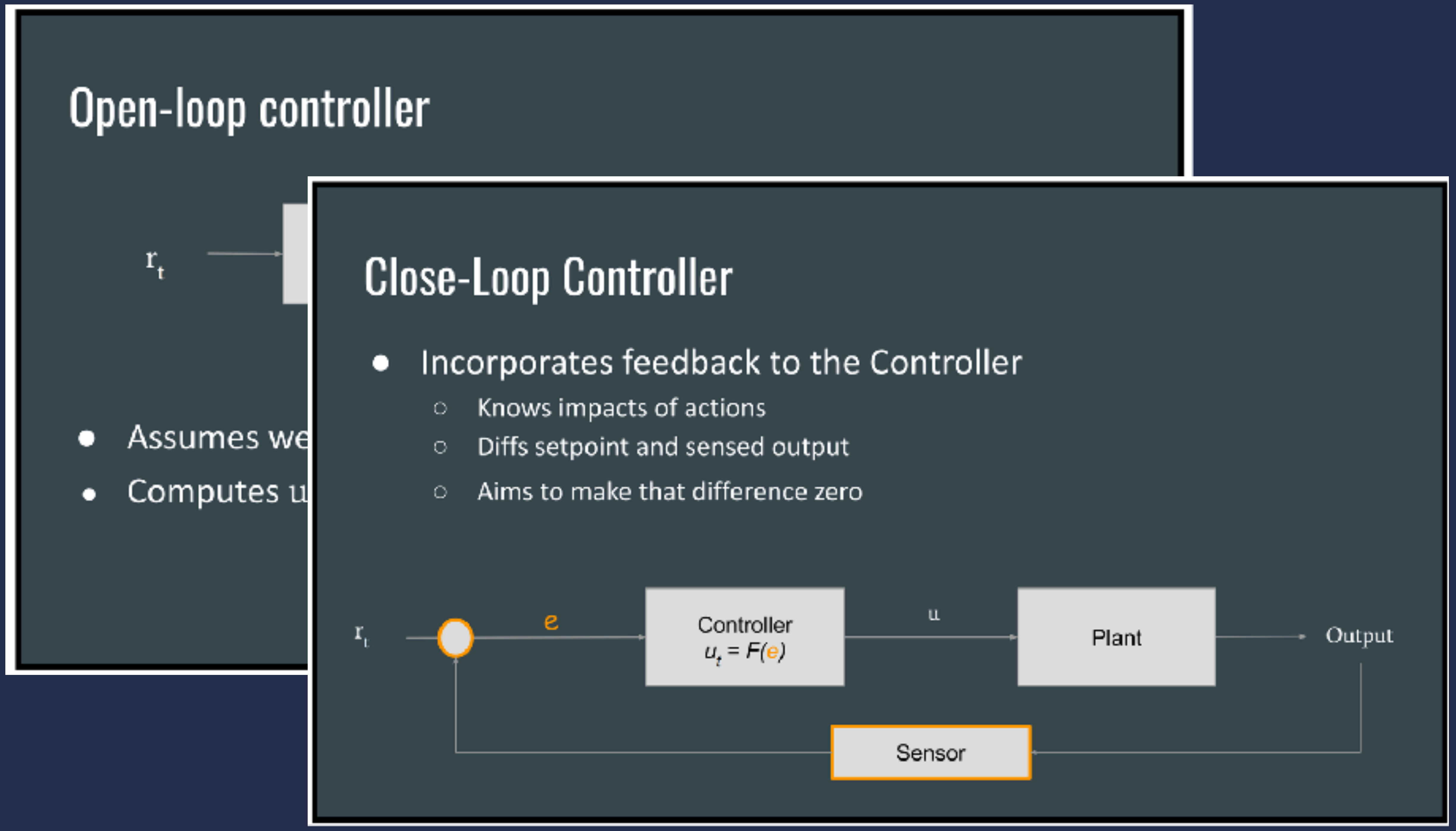
Labs

- Perception though Analyzing Images
- Sensor filtering and fusion
- Types and machines
- ROS and Simulation environment
- Set up

Lecture Lab Pairing

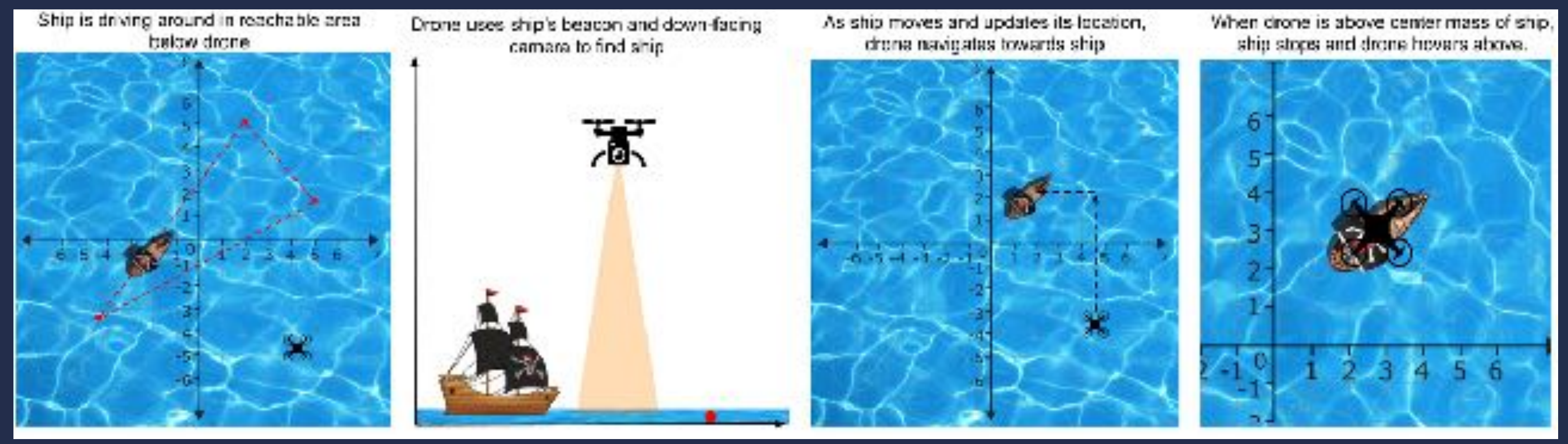
Lectures

- Controlling your robot
- Perception
- Robot and world through sensors
- Software Machinery
- Development Features
- Introduction



Introducing different control schemes

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



```
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')
```

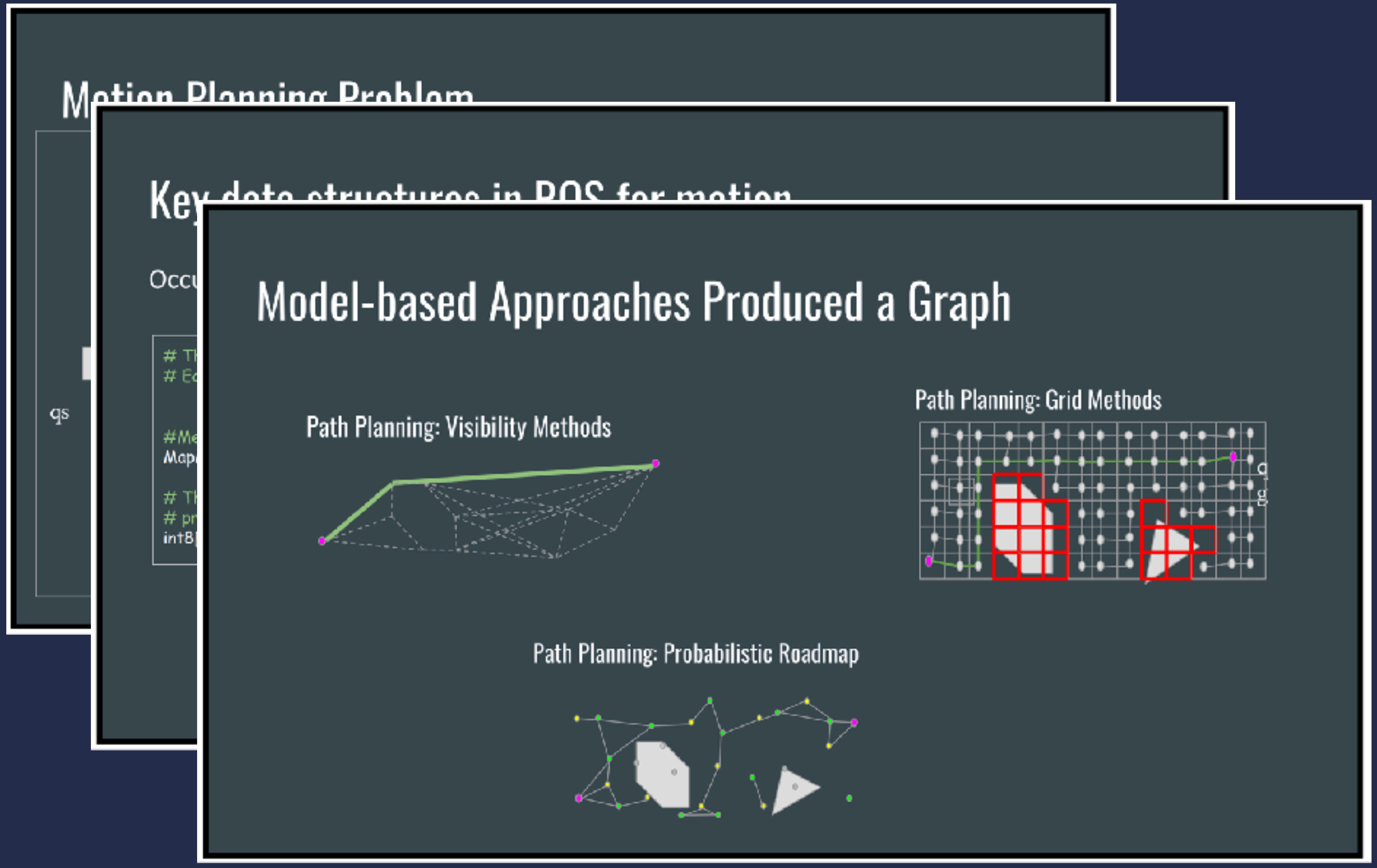
Labs

- Controlling and testing robots
- Perception though Analyzing Images
- Sensor filtering and fusion
- Types and machines
- ROS and Simulation environment
- Set up

Lecture Lab Pairing

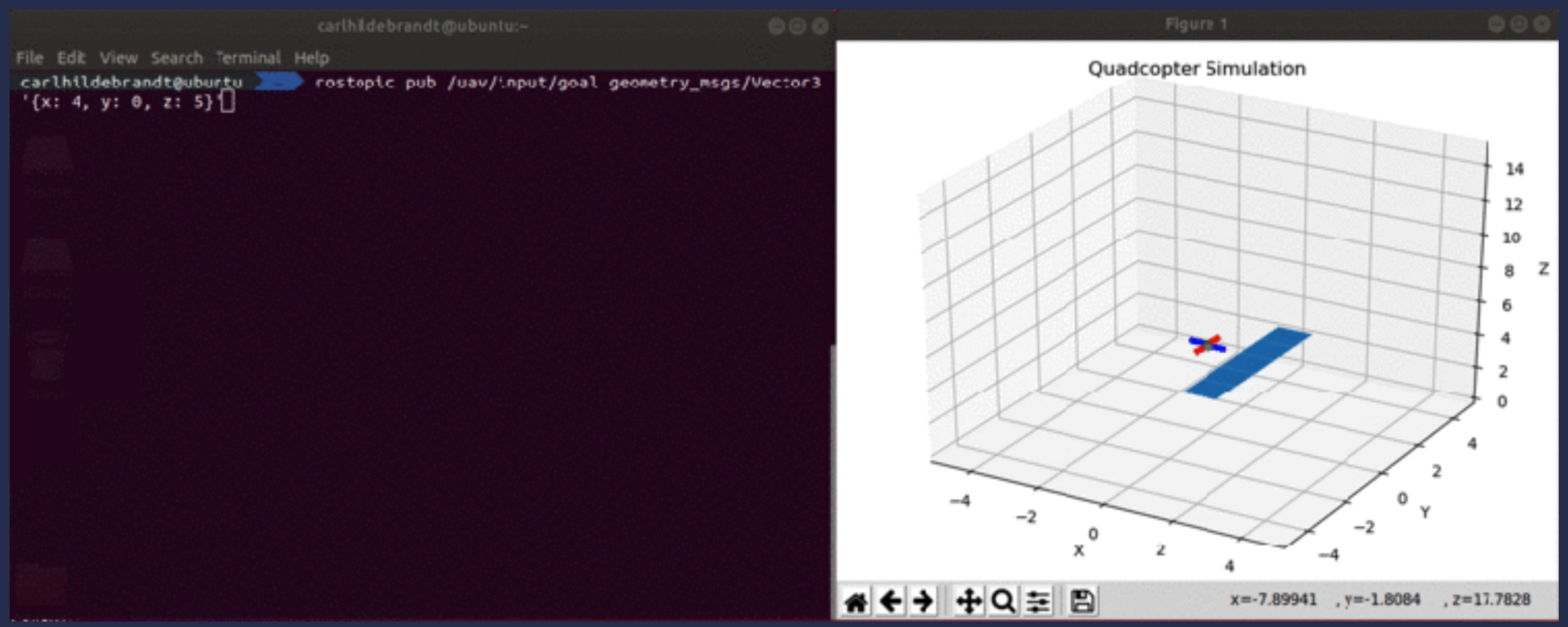
Lectures

- Localization and navigation
- Making plans
- Controlling your robot
- Perception
- Robot and world through sensors
- Software Machinery
- Development Features
- Introduction



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

Implementing mapping and path planning in simulation



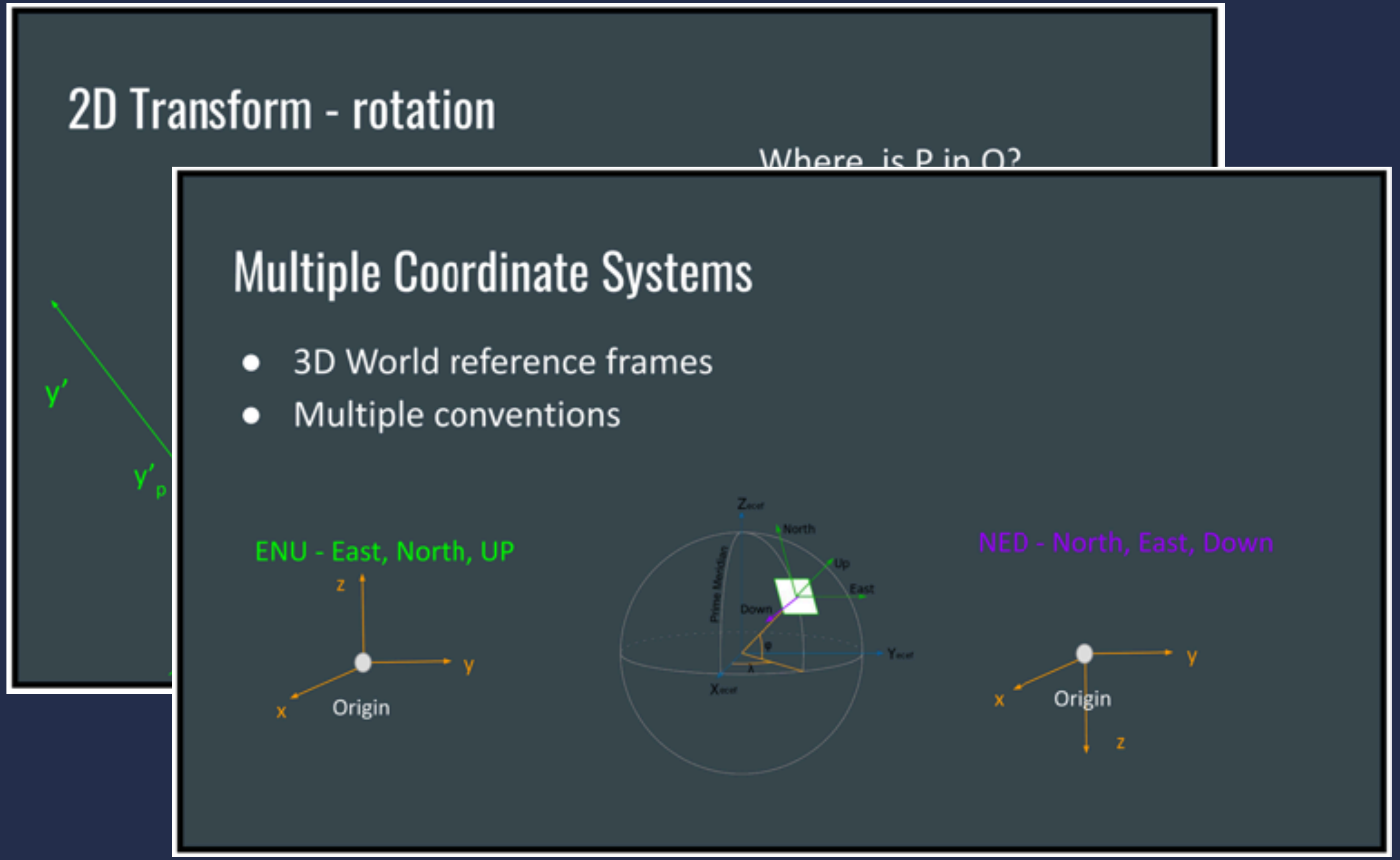
Labs

- 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

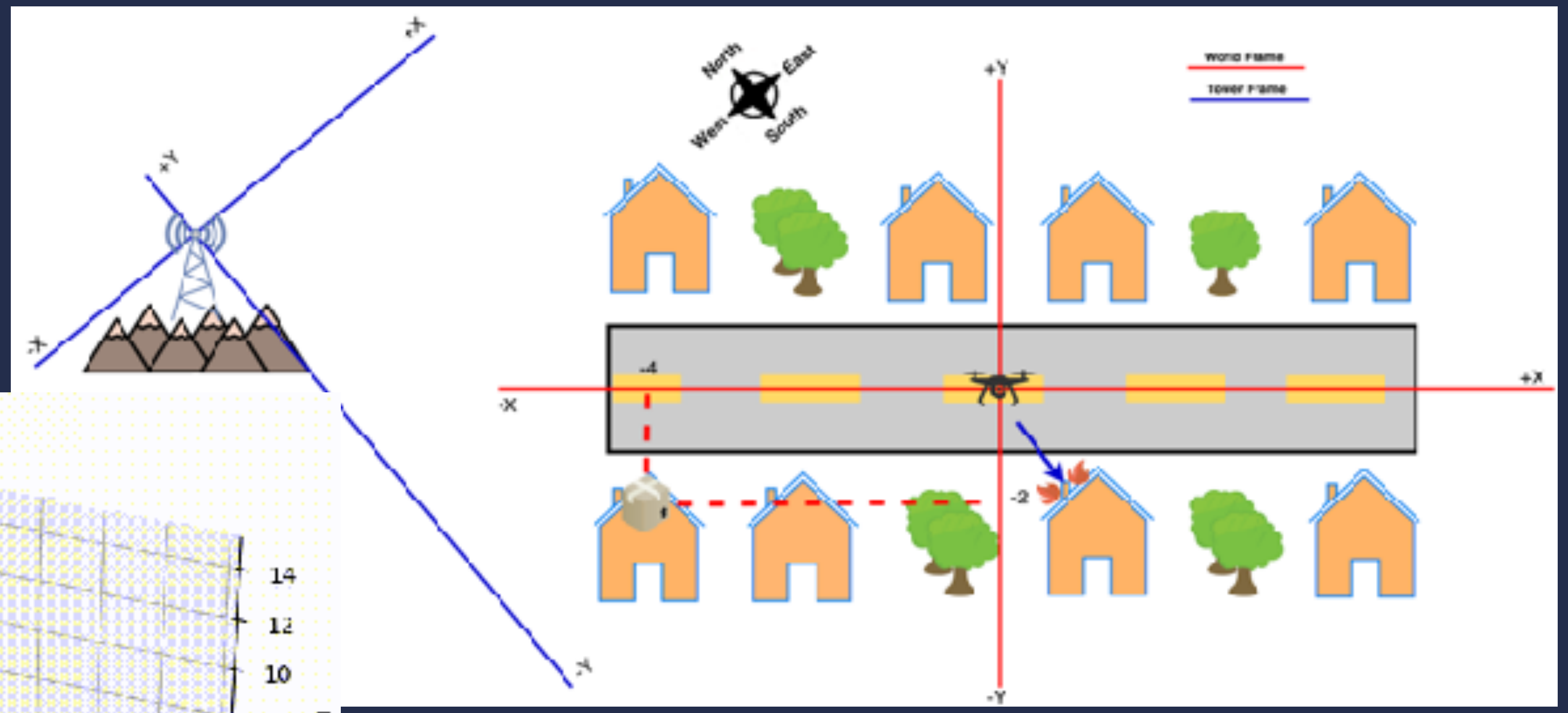
Lecture Lab Pairing

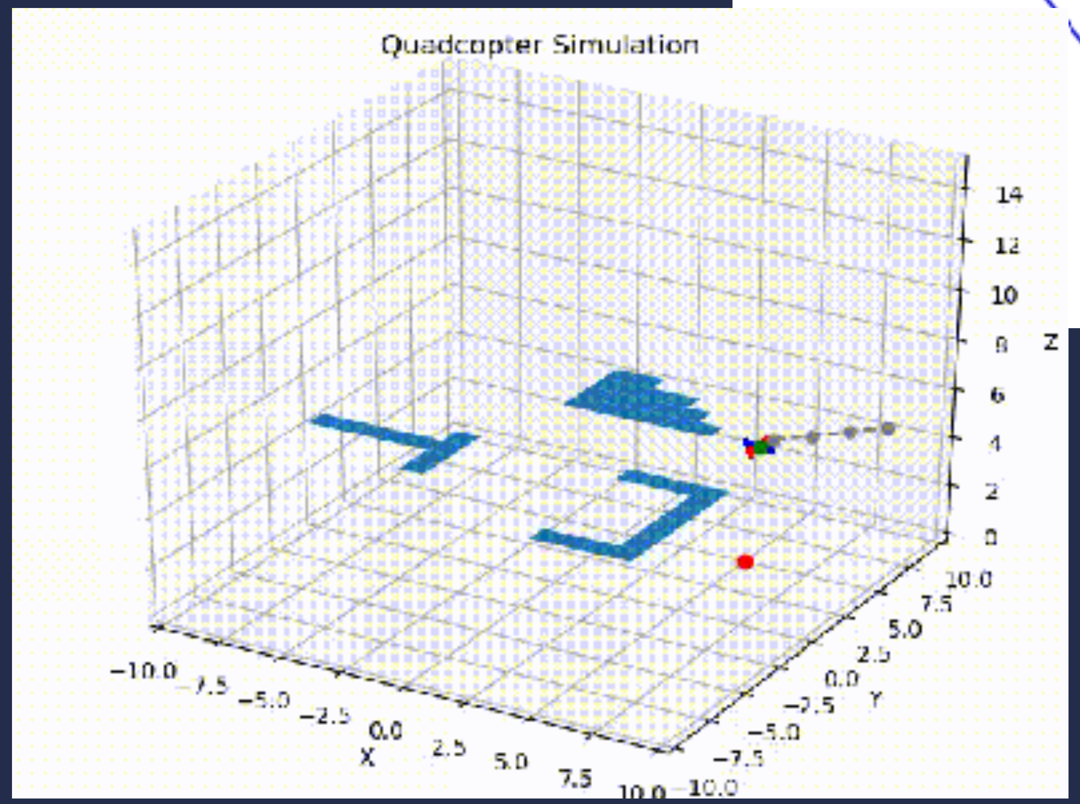
Lectures

- Transformations
- Localization and navigation
- Making plans
- Controlling your robot
- Perception
- Robot and world through sensors
- Software Machinery
- Development Features
- Introduction



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 though Analyzing Images
- Sensor filtering and fusion
- Types and machines
- ROS and Simulation environment
- Set up

Crosscutting Issues

Industry perspective: Guest Speaker

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

Ethics Lab

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

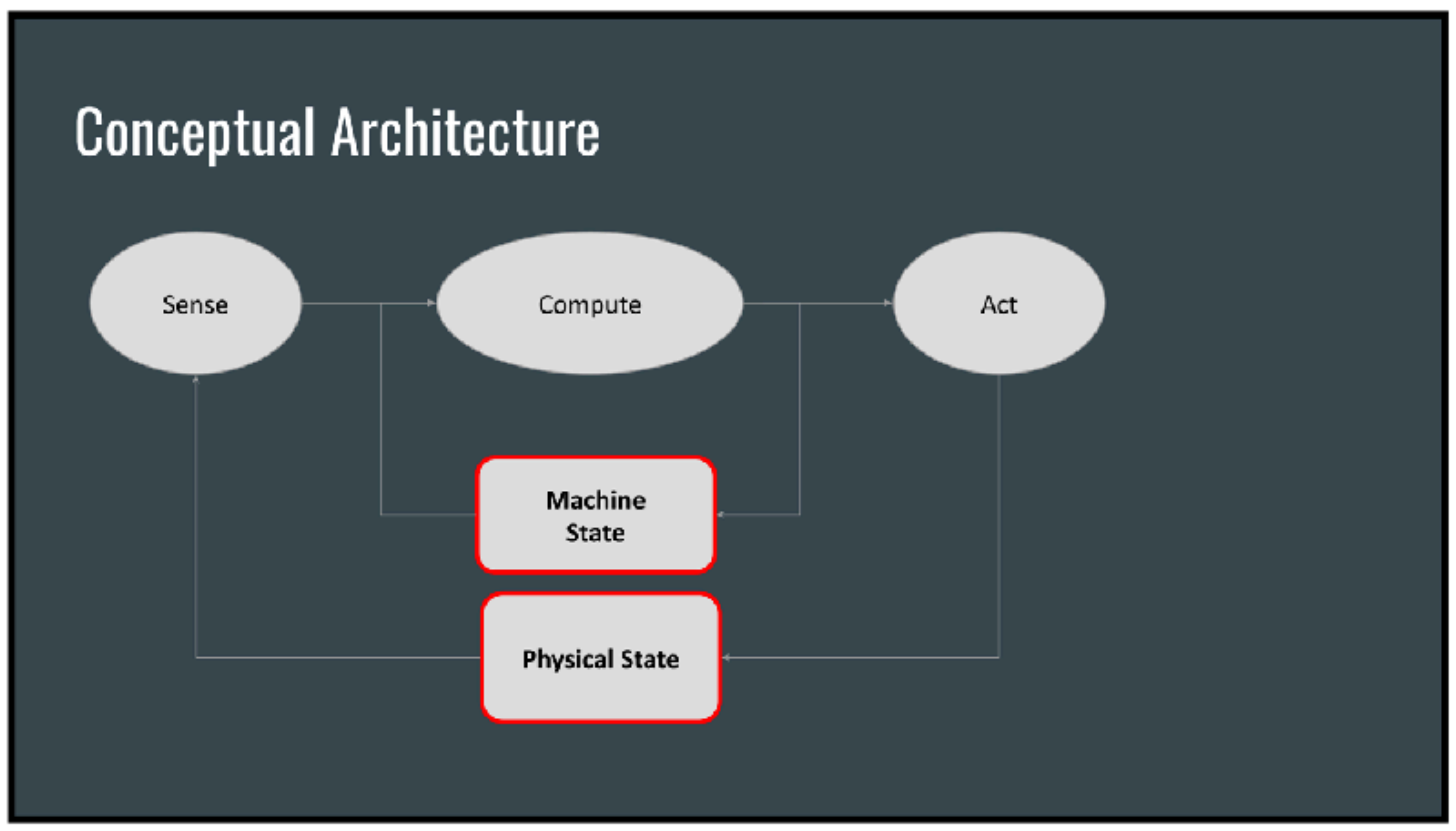
We apply these principles to bring several innovations to this course

Innovations

We apply these principles to bring several innovations to this course

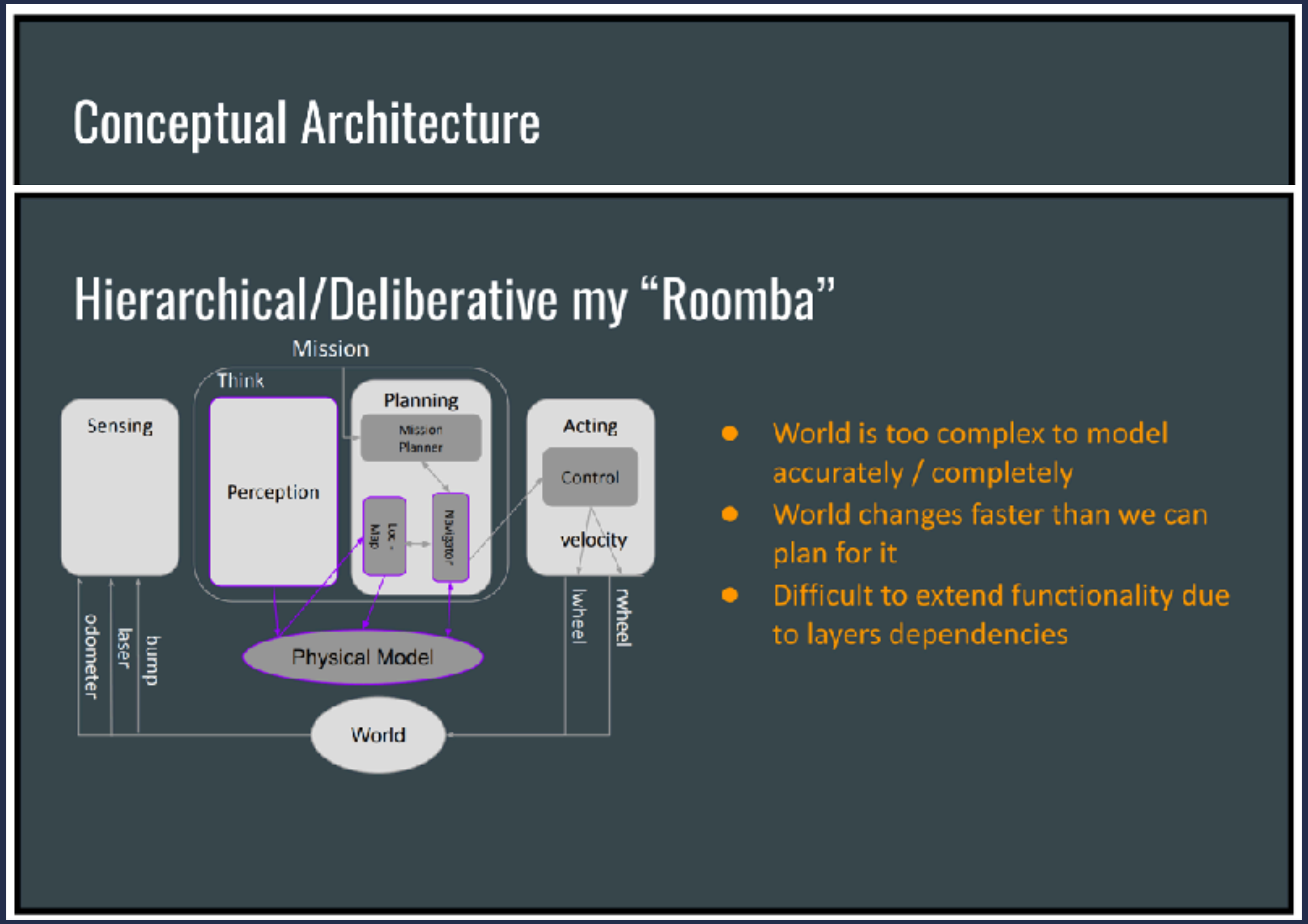
	Innovation	P1	P2	P3	P4	P5
I1	Cover robotics fundamentals	✓	✓			
I2	Offer different levels of abstraction	✓			✓	
I3	Pair SE and Robotics topics throughout the course	✓	✓			
I4	Enable students to make design and implementation decisions in labs and project			✓		✓
I5	Make use of demonstration, conversation, 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

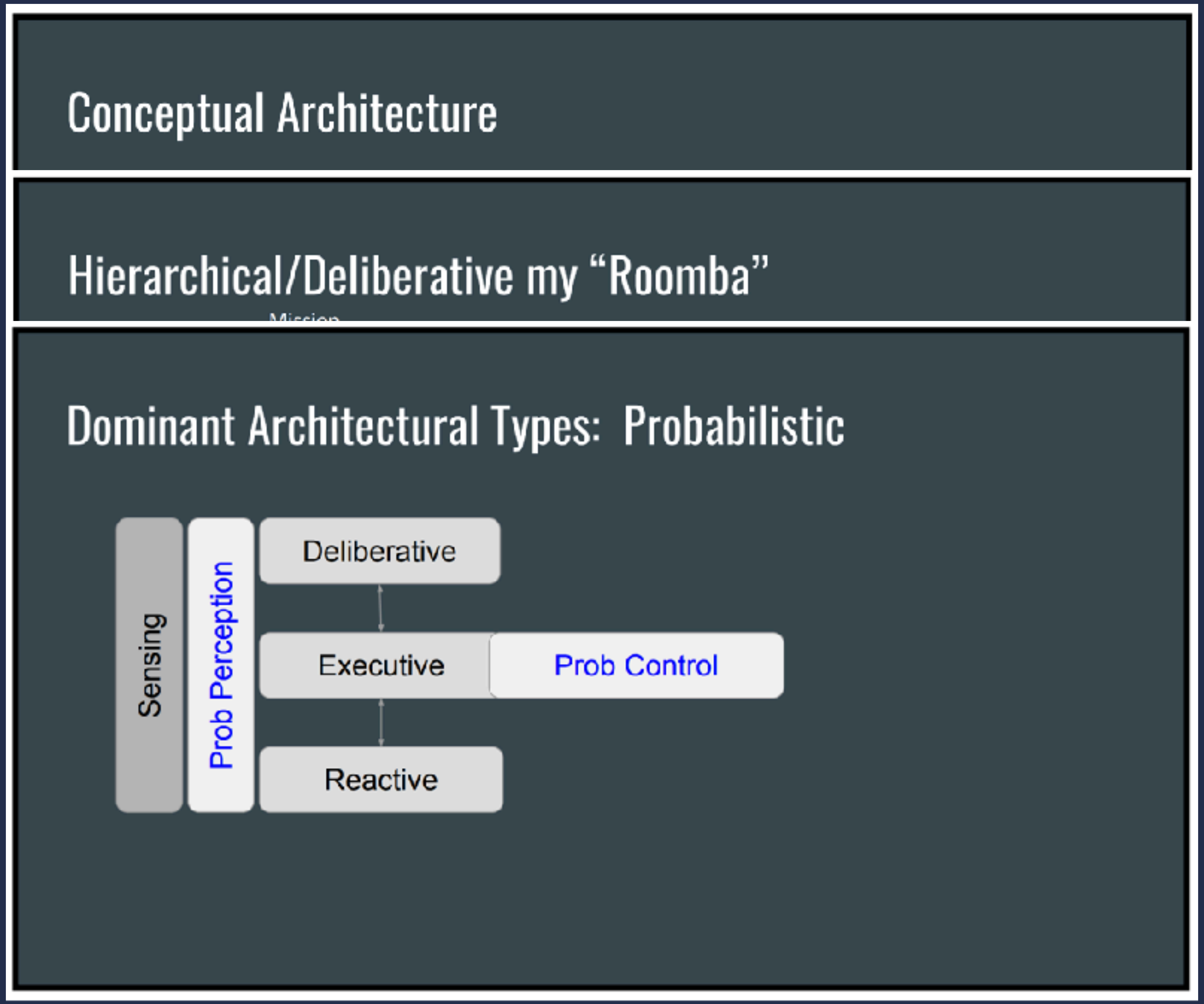
Example Lecture



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

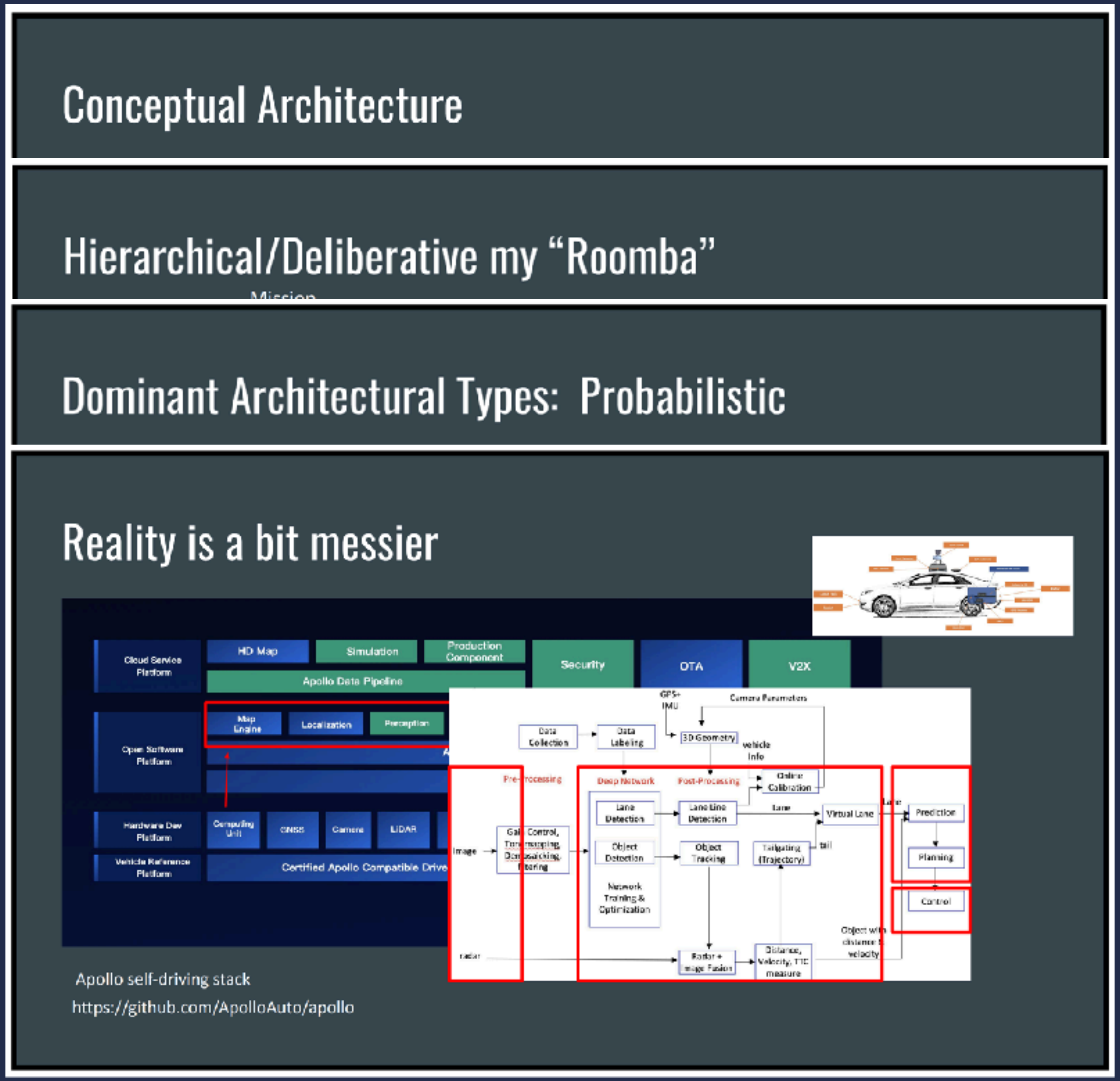


Aim: introduce the fundamental concepts related to robotics architecture and modeling machinery in robotics

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

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

Example Lecture



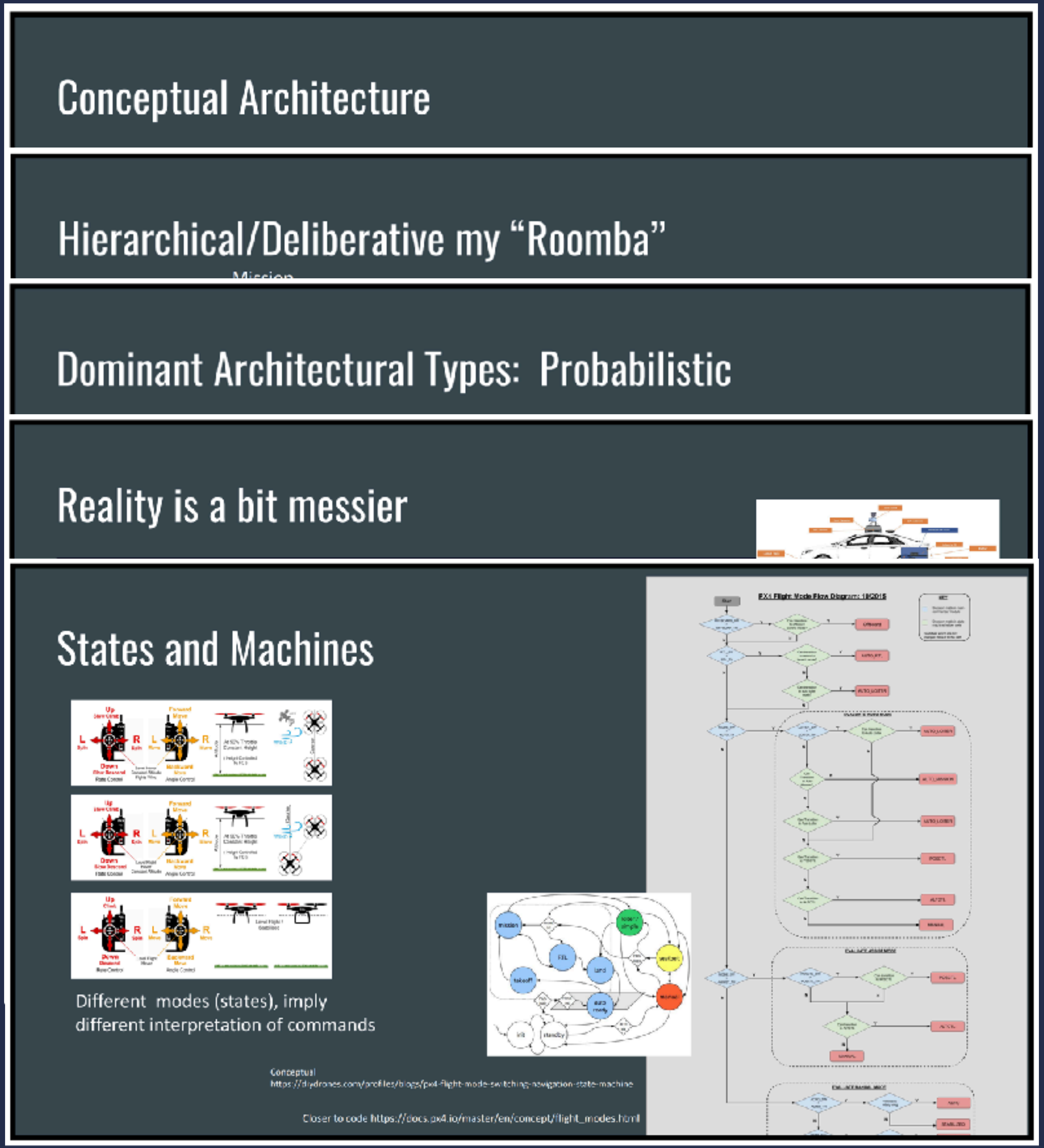
Aim: introduce the fundamental concepts related to robotics architecture and modeling machinery in robotics

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)

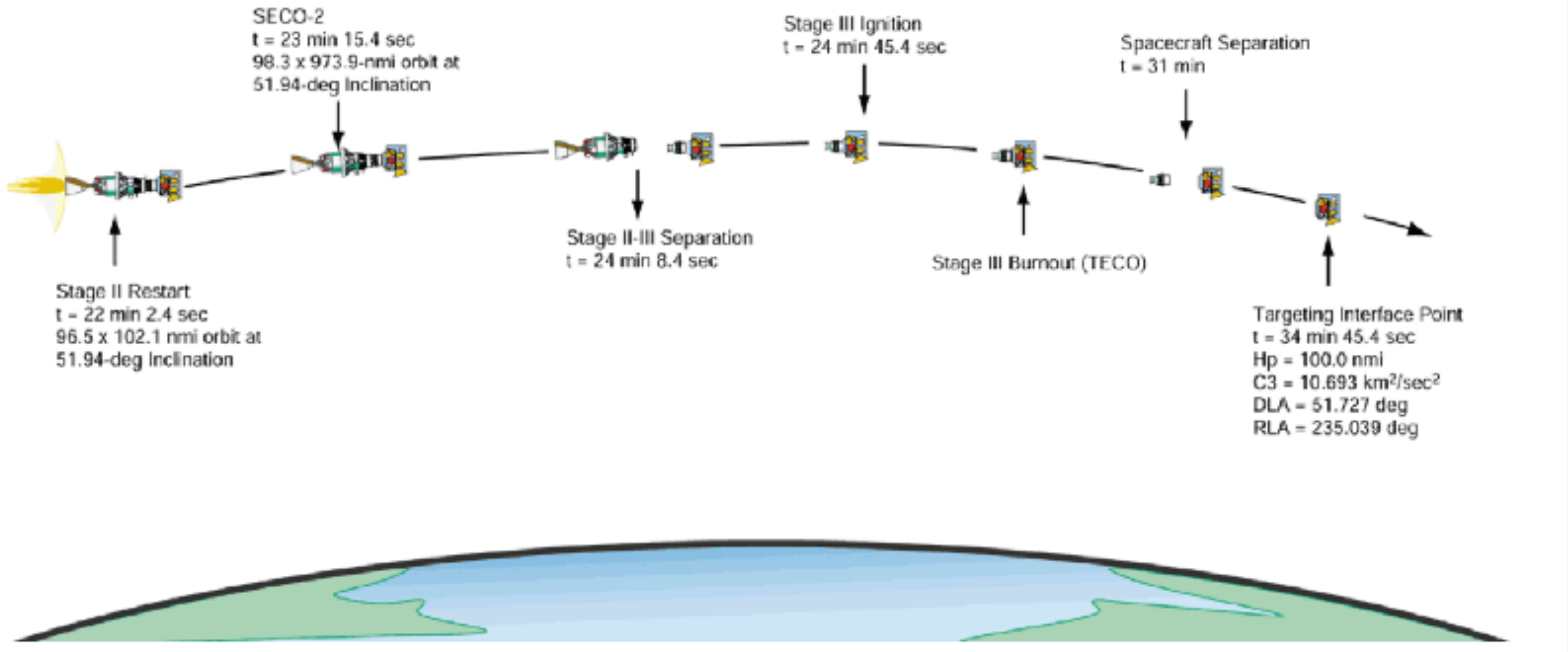
Introduce FSMs
-what types of states they can encode
-how they can assist in understanding the real world
-how they are represented in code
-how to scale them up to develop robotic systems
(I3 - Pair SE and Robotics topic)

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.

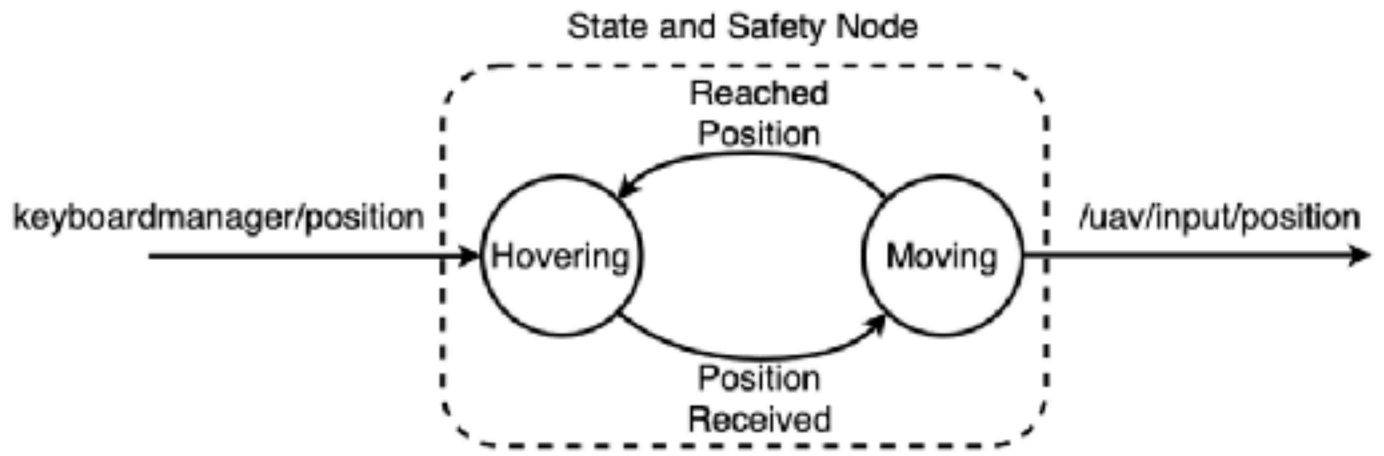
```
# Change to lab directory
$ cd ~/Desktop/CS4501-Labs/
# Clone the code
$ git pull
```

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

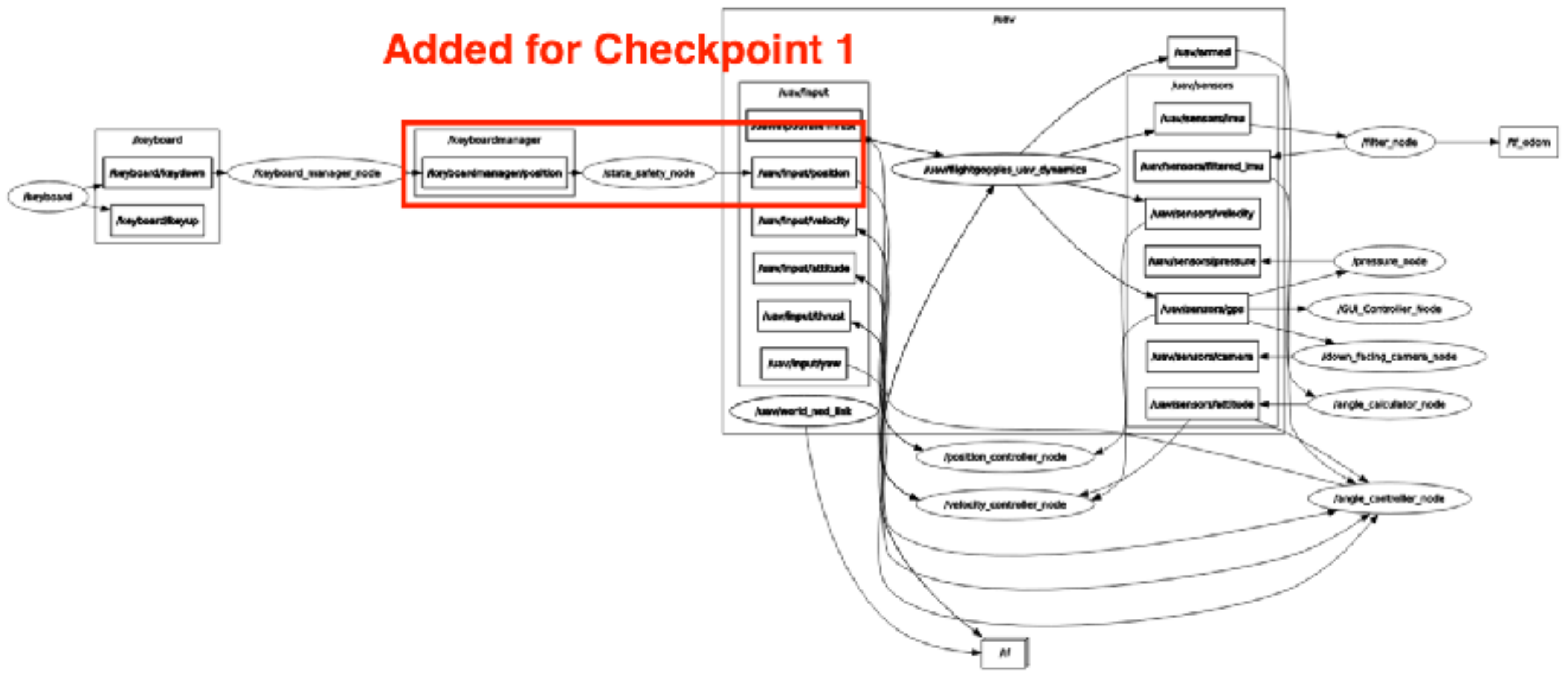
Create the State and Safety Node

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 quadrotor 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`

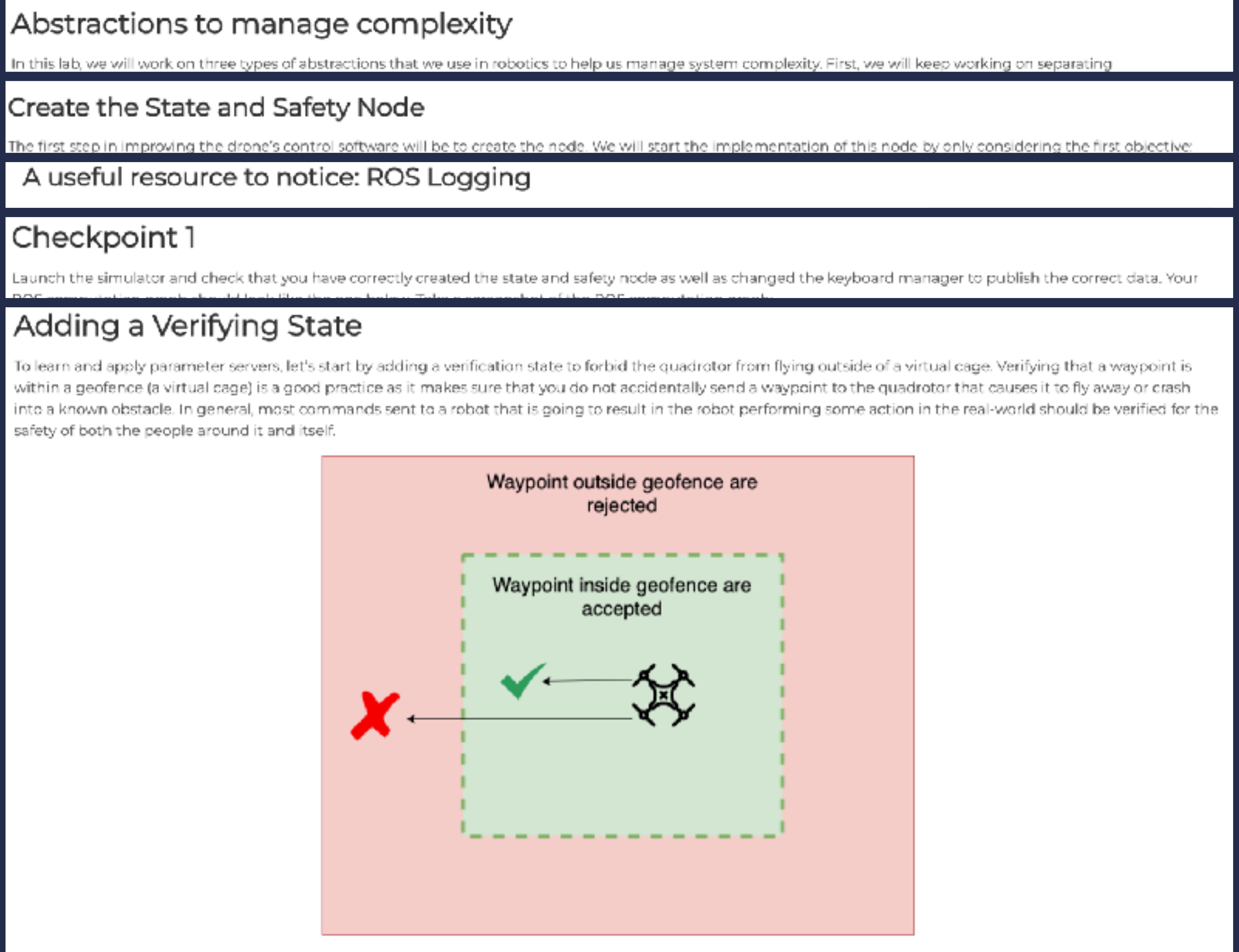
Starts highlighting how what is learned is implemented in real systems.

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

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.

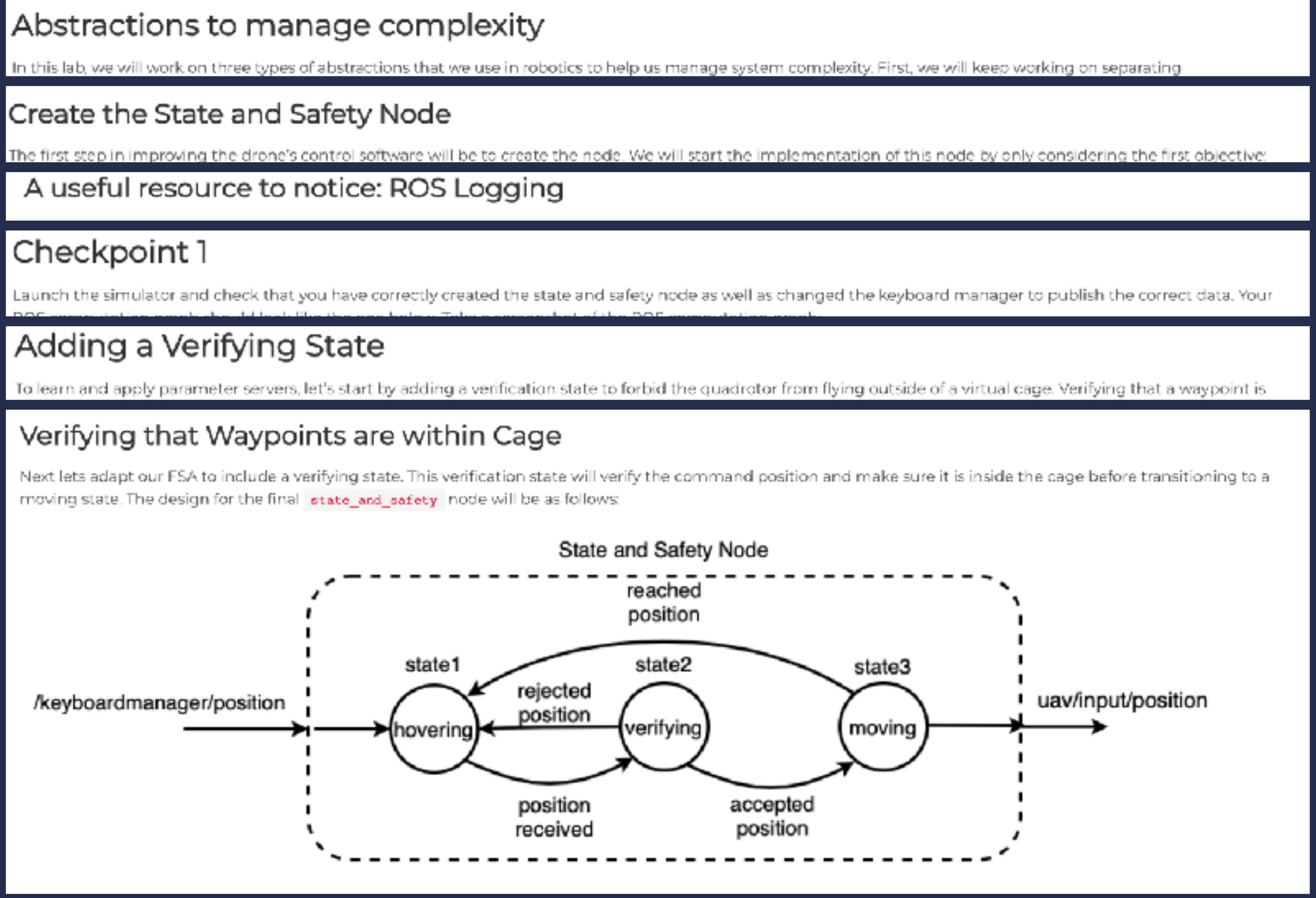
Starts with implementing a basic FSM
(I3 - Pair SE and Robotics topic)

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

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 lets 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

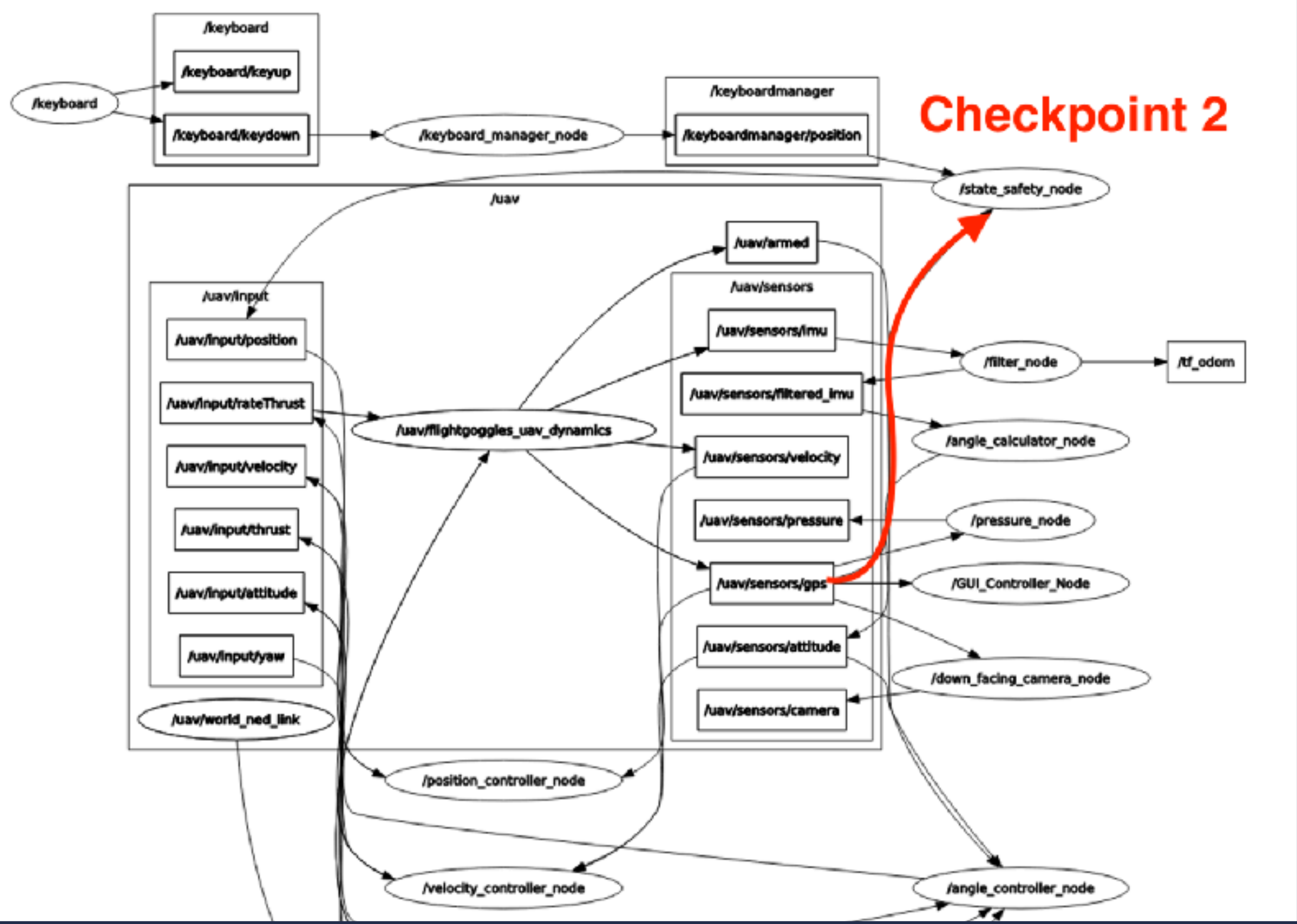
Adding a Verifying State

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

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)

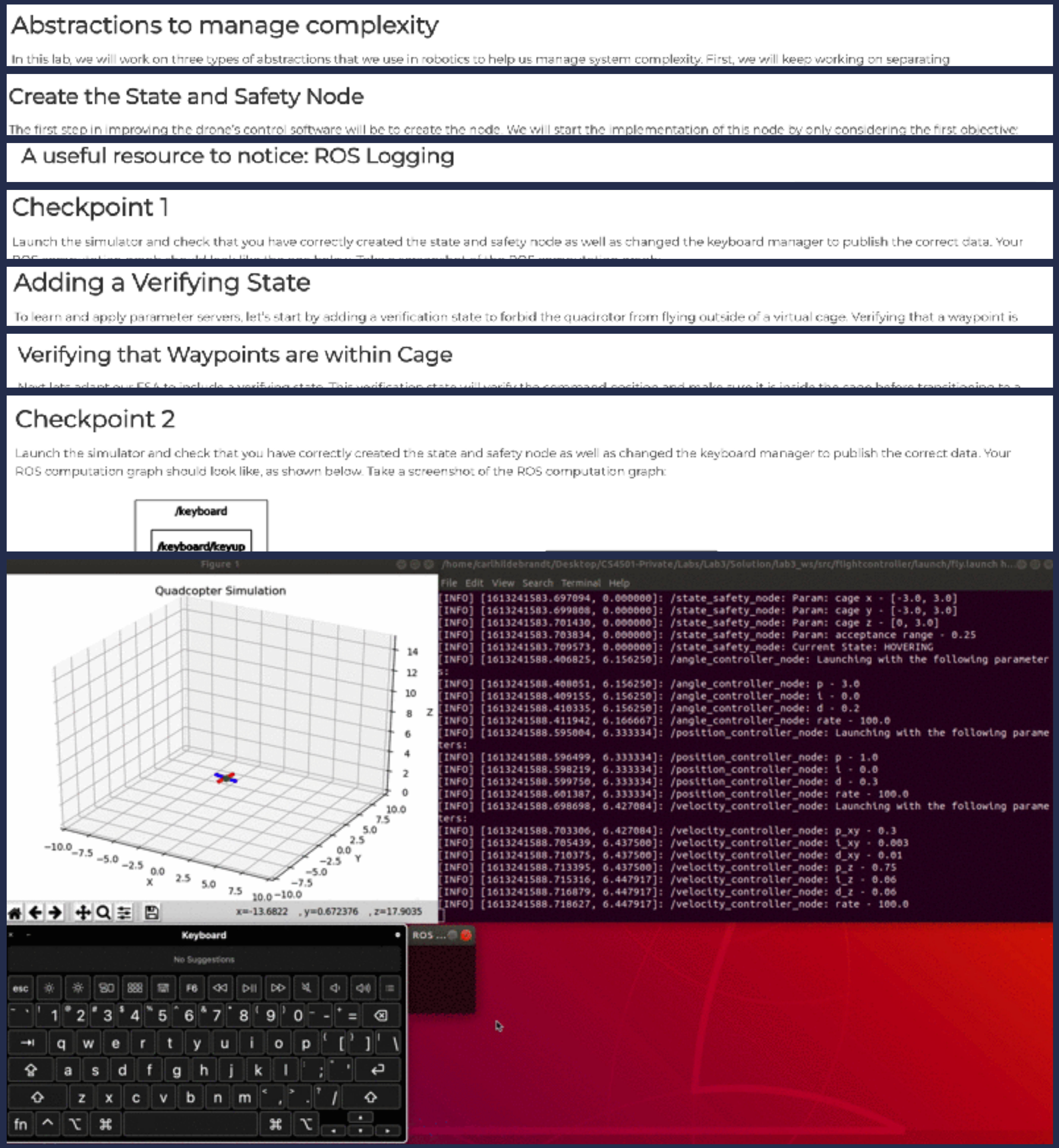
Highlight how ROS allows for looking information allowing for easy debugging
(I1 - Cover robotics fundamentals)

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



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

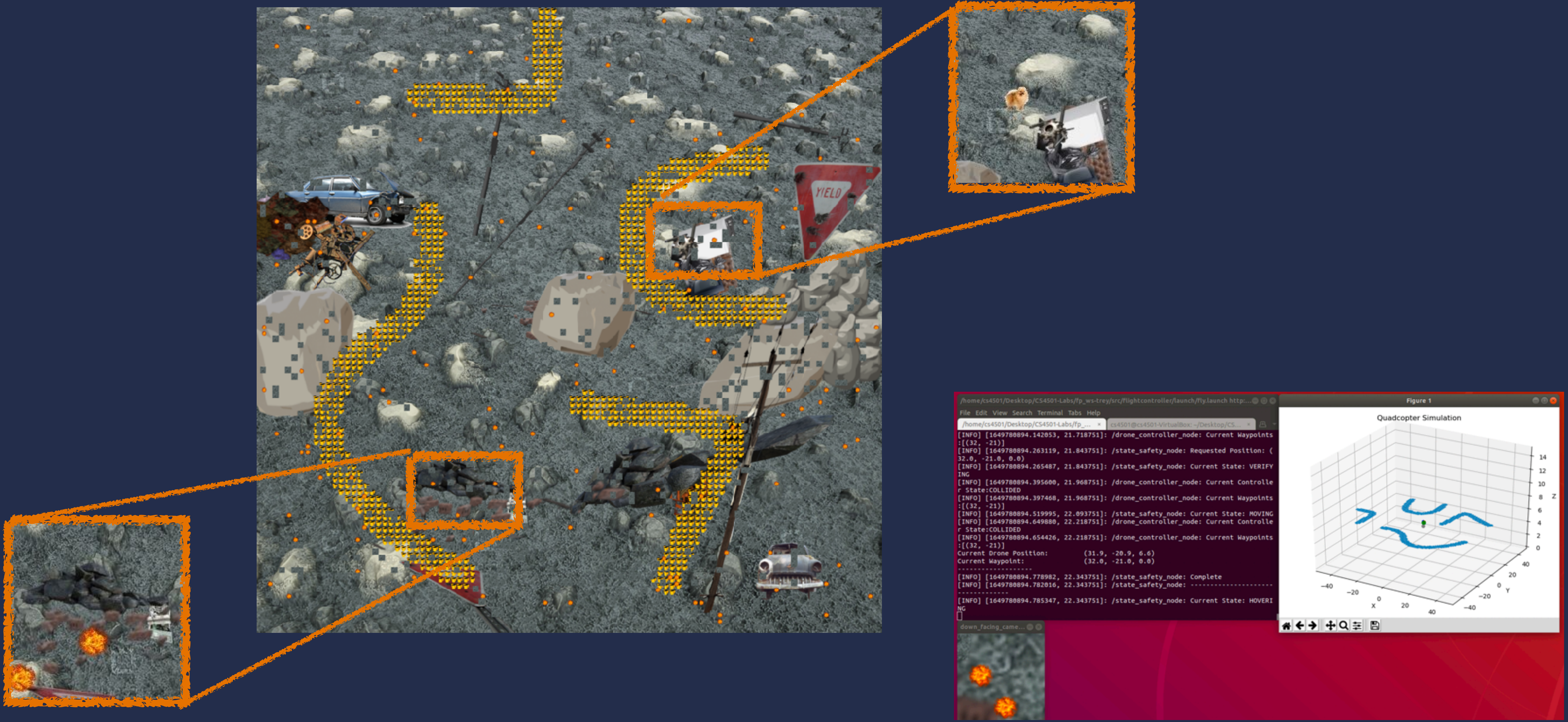
Locate itself

Locate object, avoiding obstacles

Collect the object

Return to safe area

Project



YIELD
/home/cs4501/Desktop/CS4501-Labs/fp_ws-trey/src/flightcontroller/launch/fly.launch http:...
File Edit View Search Terminal Tabs Help
/home/cs4501/Desktop/CS4501-Labs/fp_...
cs4501@cs4501-VirtualBox: ~/Desktop/CS...
[INFO] [1649780894.142053, 21.718751]: /drone_controller_node: Current Waypoints :[(32, -21)]
[INFO] [1649780894.263119, 21.843751]: /state_safety_node: Requested Position: (32.0, -21.0, 0.0)
[INFO] [1649780894.265487, 21.843751]: /state_safety_node: Current State: VERIFYING
[INFO] [1649780894.395600, 21.968751]: /drone_controller_node: Current Controller State:COLLIDED
[INFO] [1649780894.397468, 21.968751]: /drone_controller_node: Current Waypoints :[(32, -21)]
[INFO] [1649780894.519995, 22.093751]: /state_safety_node: Current State: MOVING
[INFO] [1649780894.649880, 22.218751]: /drone_controller_node: Current Controller State:COLLIDED
[INFO] [1649780894.654426, 22.218751]: /drone_controller_node: Current Waypoints :[(32, -21)]
Current Drone Position: (31.9, -20.9, 6.6)
Current Waypoint: (32.0, -21.0, 0.0)
[INFO] [1649780894.778982, 22.343751]: /state_safety_node: Complete
[INFO] [1649780894.782016, 22.343751]: /state_safety_node: --------------------------------
[INFO] [1649780894.785347, 22.343751]: /state_safety_node: Current State: HOVERING
down_facing_came...
Figure 1
Quadcopter Simulation

Lessons Learned

What worked well

VS

What needs improvement

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

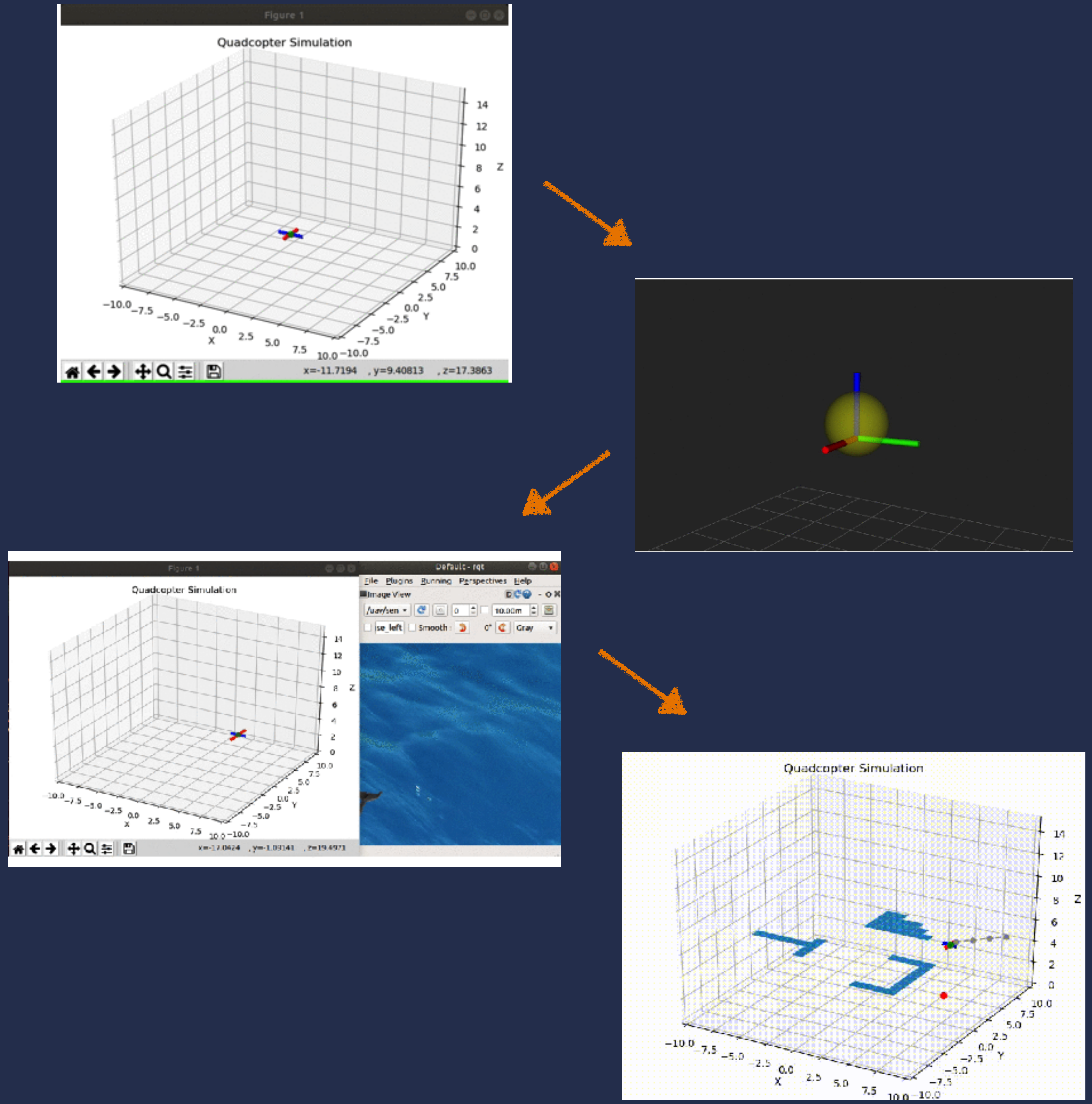
What Worked Well 👍

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

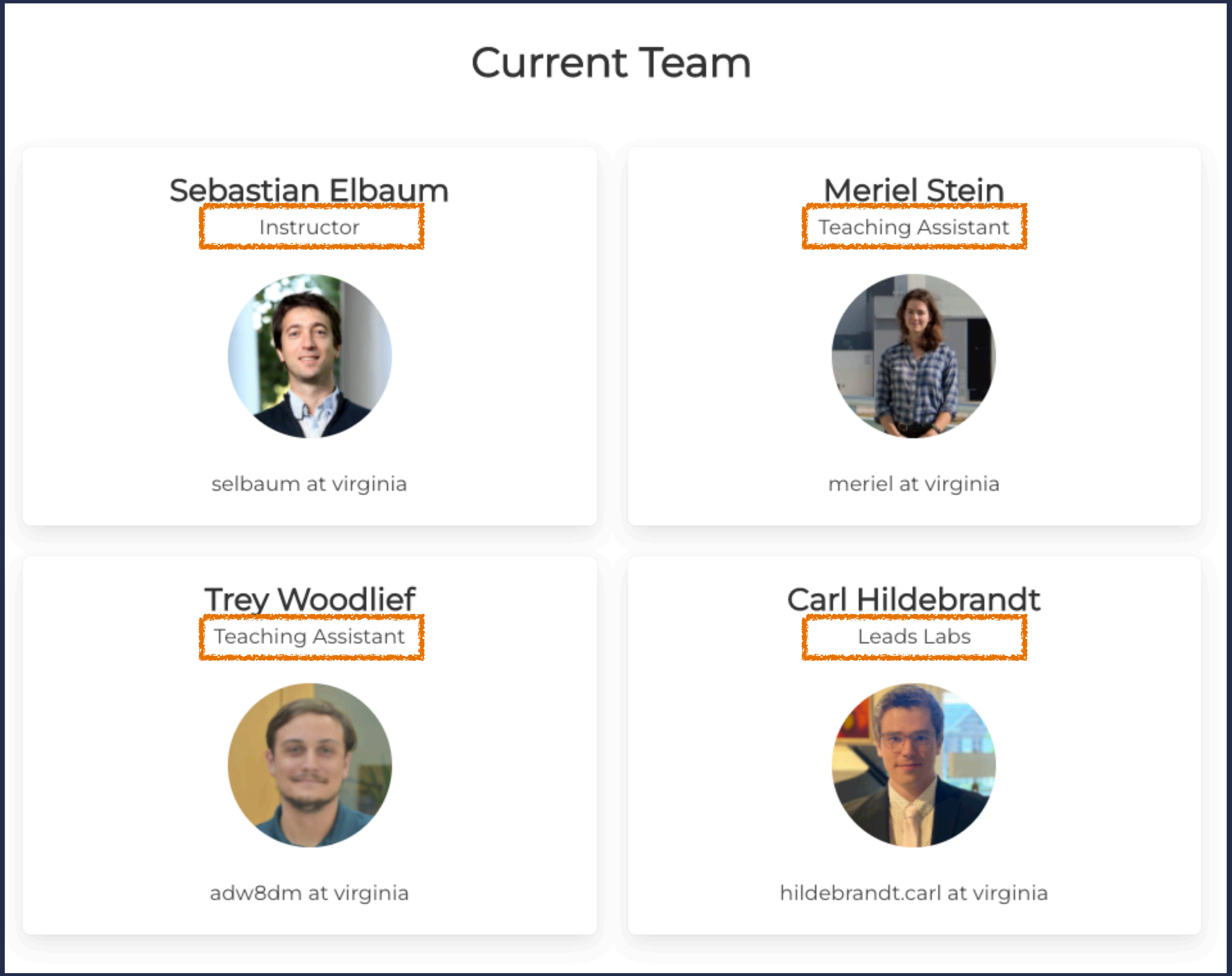
1. Pairing SE and robotics topics
2. Building flexibility into the course
3. Using different levels of abstraction
4. Incremental scaffolding of course material
5. Team structure and process
6. Demonstrating and reflecting during checkpoints

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



What Worked Well

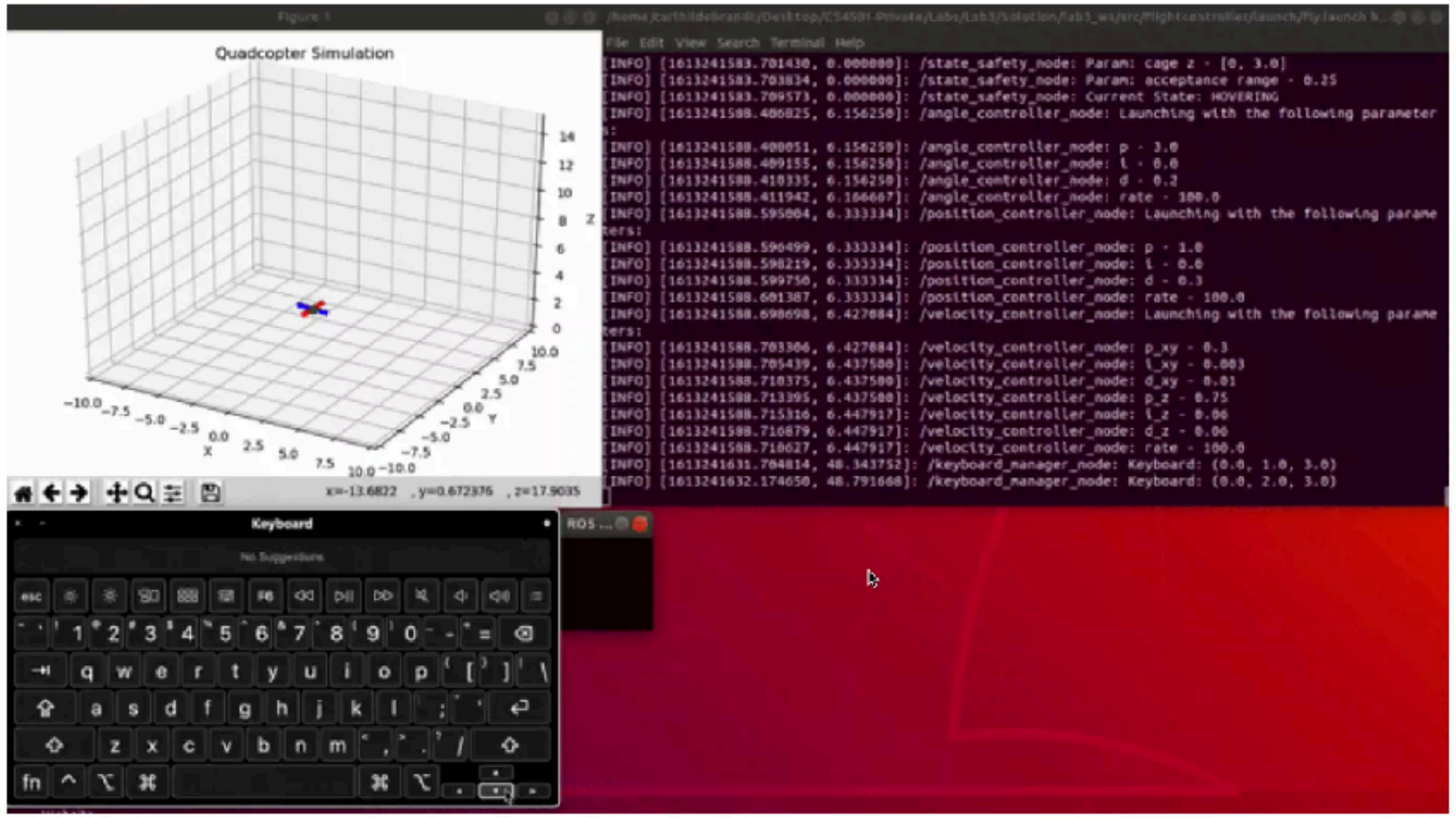
1. Pairing SE and robotics topics
2. Building flexibility into the course
3. Using different levels of abstraction
4. Incremental scaffolding of course material
5. Team structure and process
6. Demonstrating and reflecting during checkpoints

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

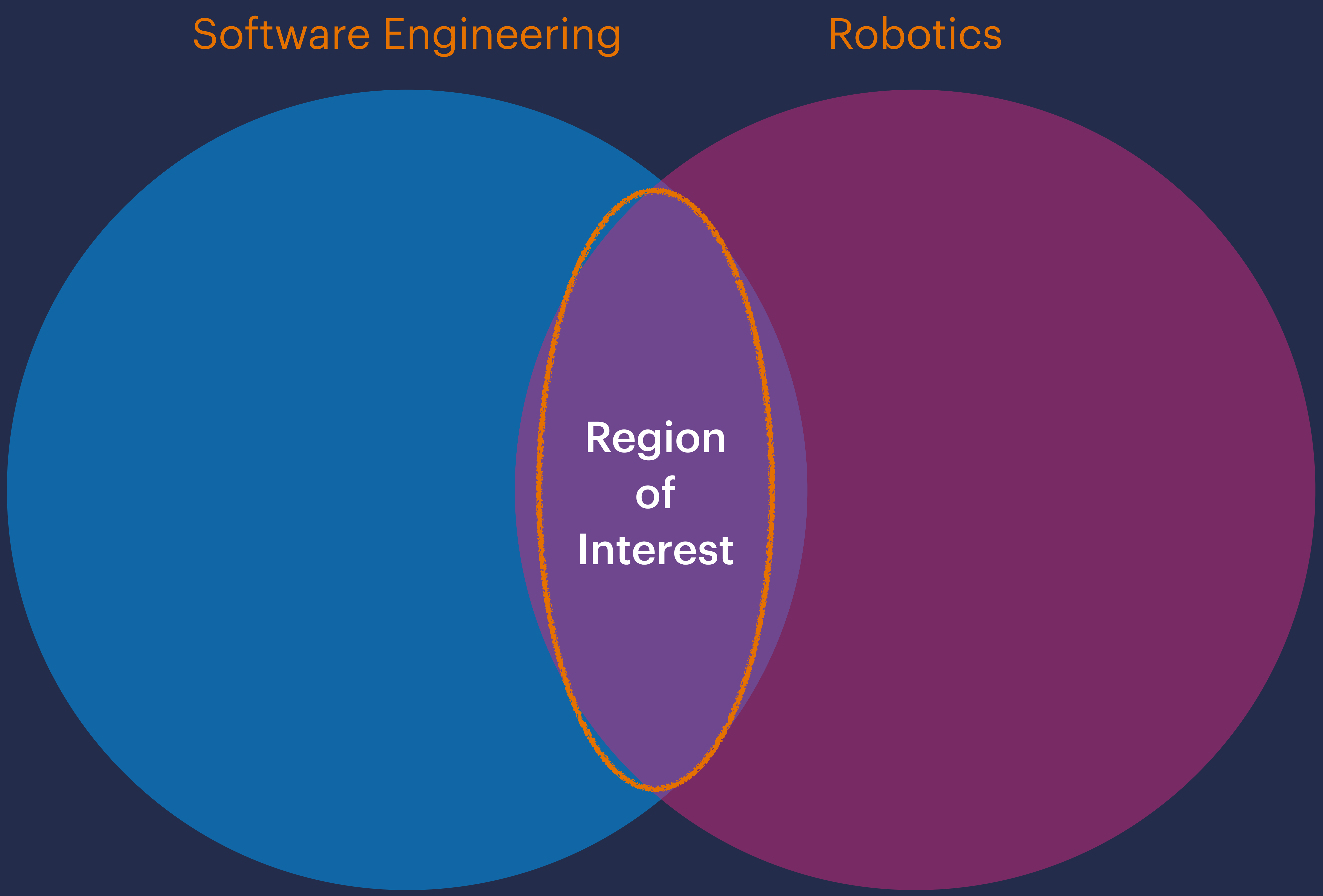
5. Require a way to empirically assess the success of this course

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

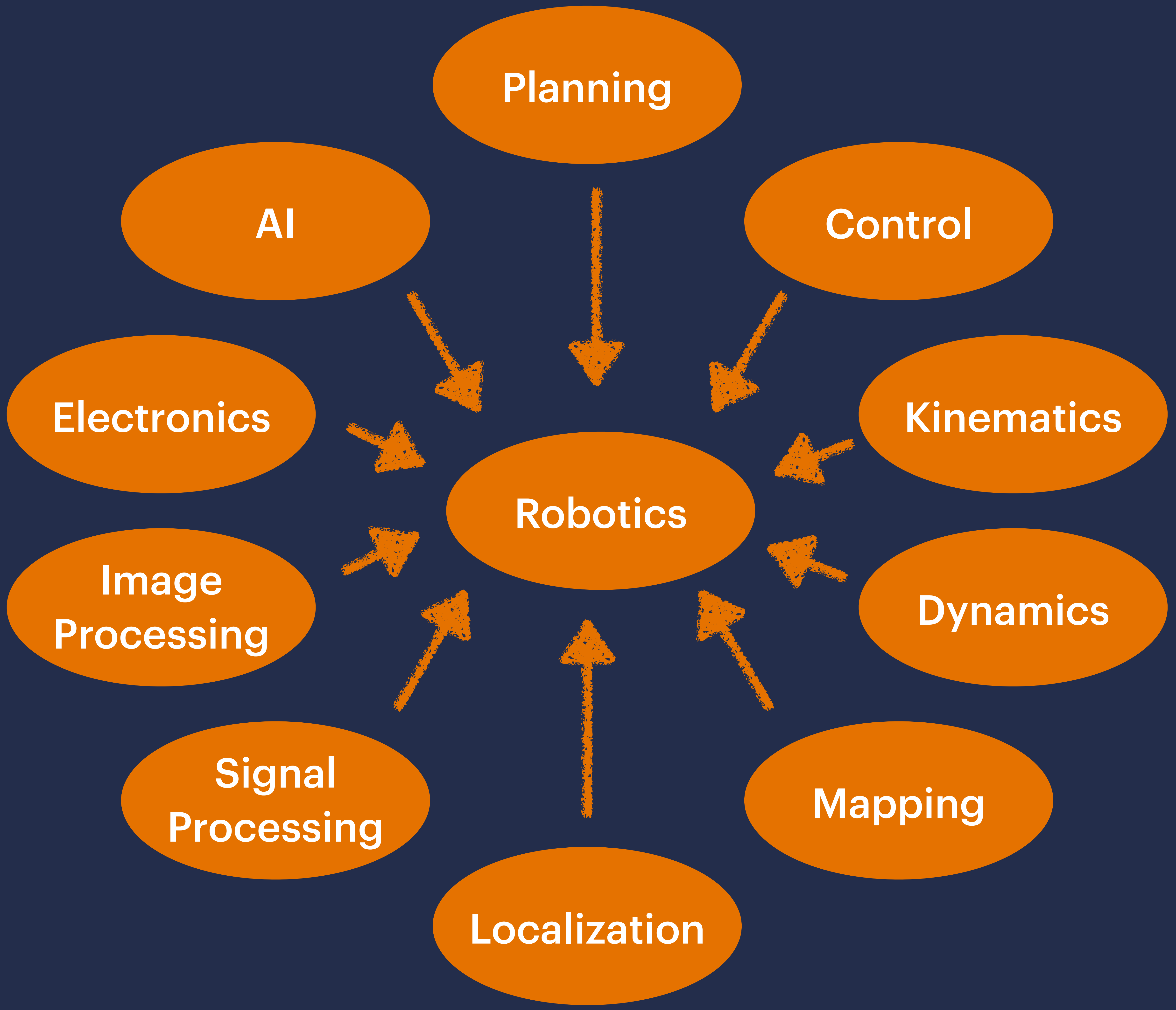
1. Diversity of student machines presents a continual challenge

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

4 Ended up in Robotics Industry

Class of 26 Students

"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]"

This year we have 75 students!

"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