



# 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](mailto:hildebrandt.carl@virginia.edu)



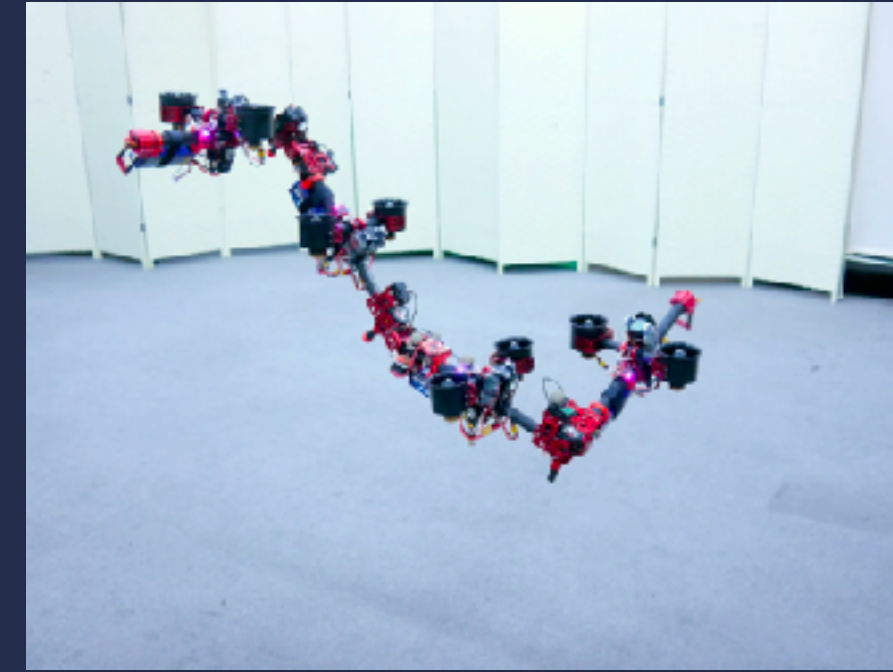
<https://less-lab-uva.github.io/CS4501-Website/>





# 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:

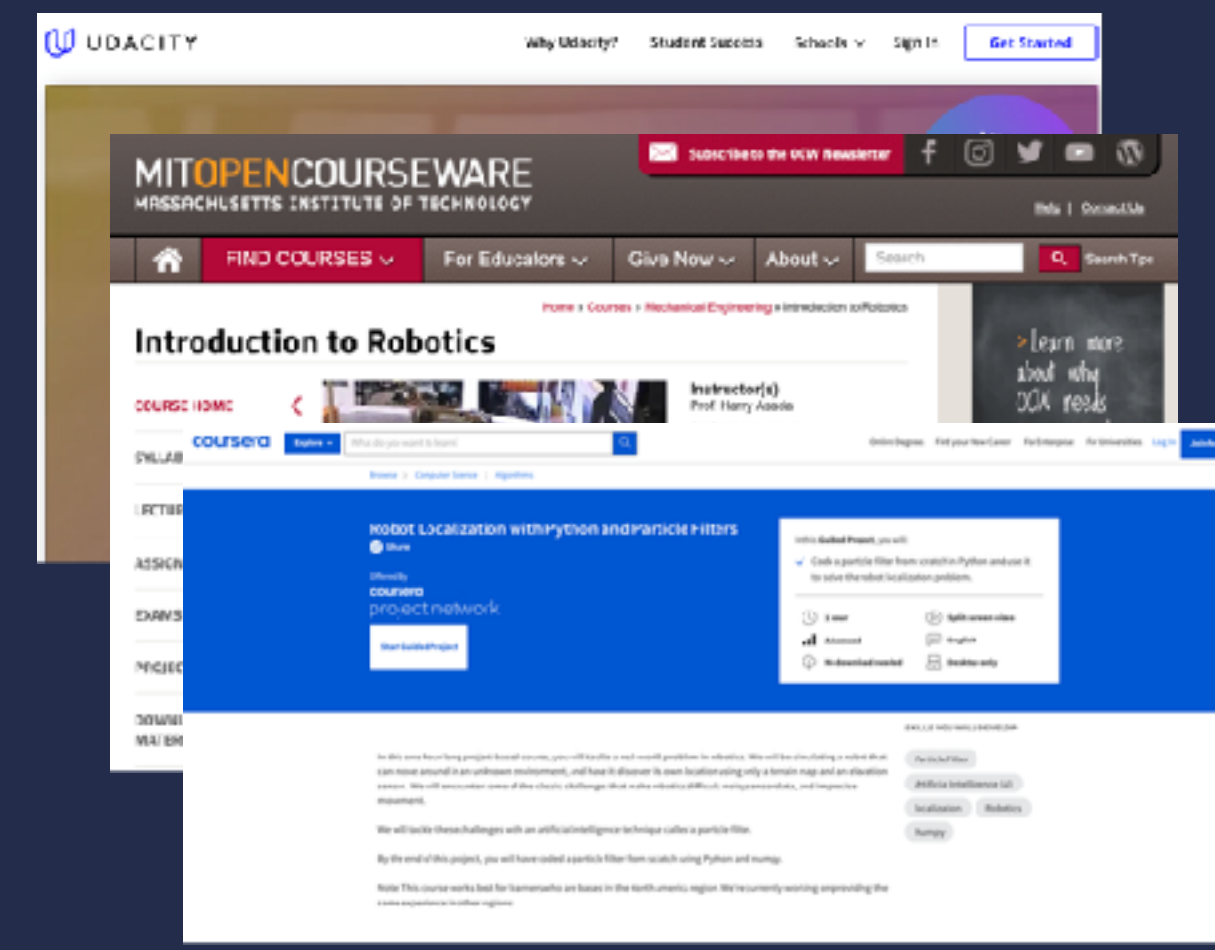
## Specialized Graduate-Level Courses



## CPS/Embedded Systems Courses



## Massive Open Online Courses



## Traditional SE Courses

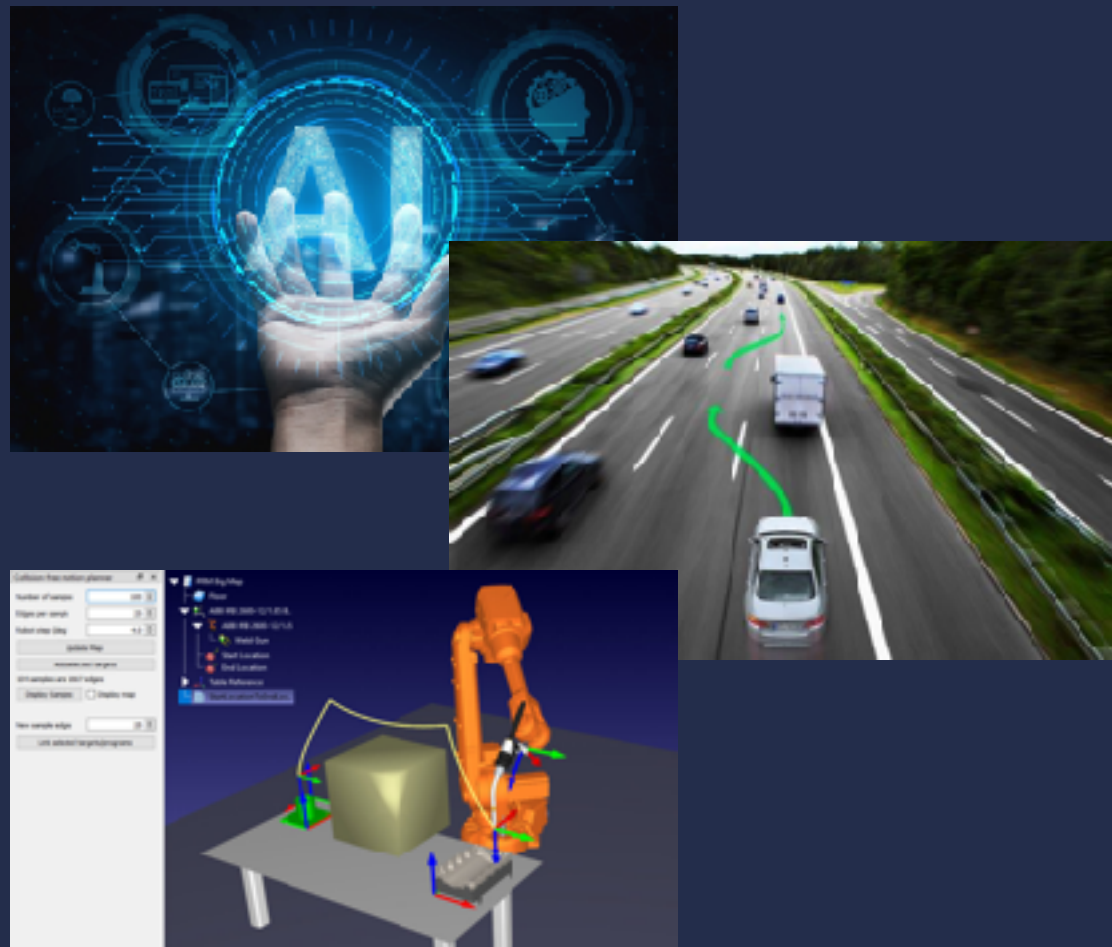




# Traditional CS Curricula

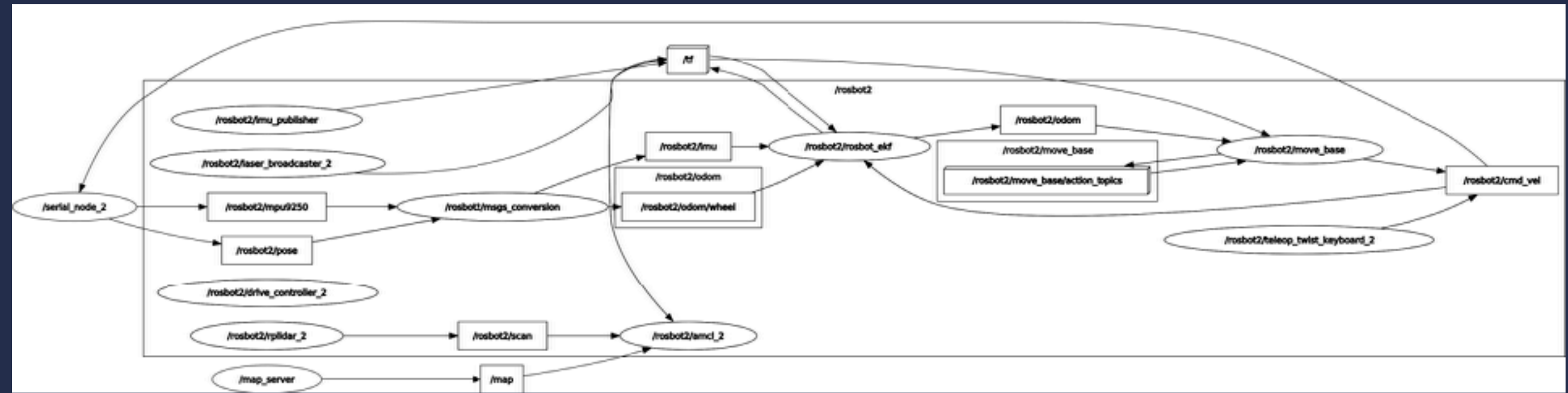
Preparing our future software engineers for this has been met primarily through:

## Specialized Graduate-Level Courses



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

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



```
252 }
253 }
254 }
255 function updatePhotoDescription() {
256   if (descriptions.length > (page * 9) + (currentImageSubsting() - 1))
257     document.getElementById("bigimageDesc").innerHTML = descriptions[page * 9 + currentImageSubsting() - 1];
258 }
259 }
260 }
261 function updateAllImages() {
262   var i = 1;
263   while (i < 10) {
264     var elementId = "foto" + i;
265     var elementIdBig = "bigimage" + i;
```



# Traditional CS Curricula

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

Specialized Graduate-Level Courses



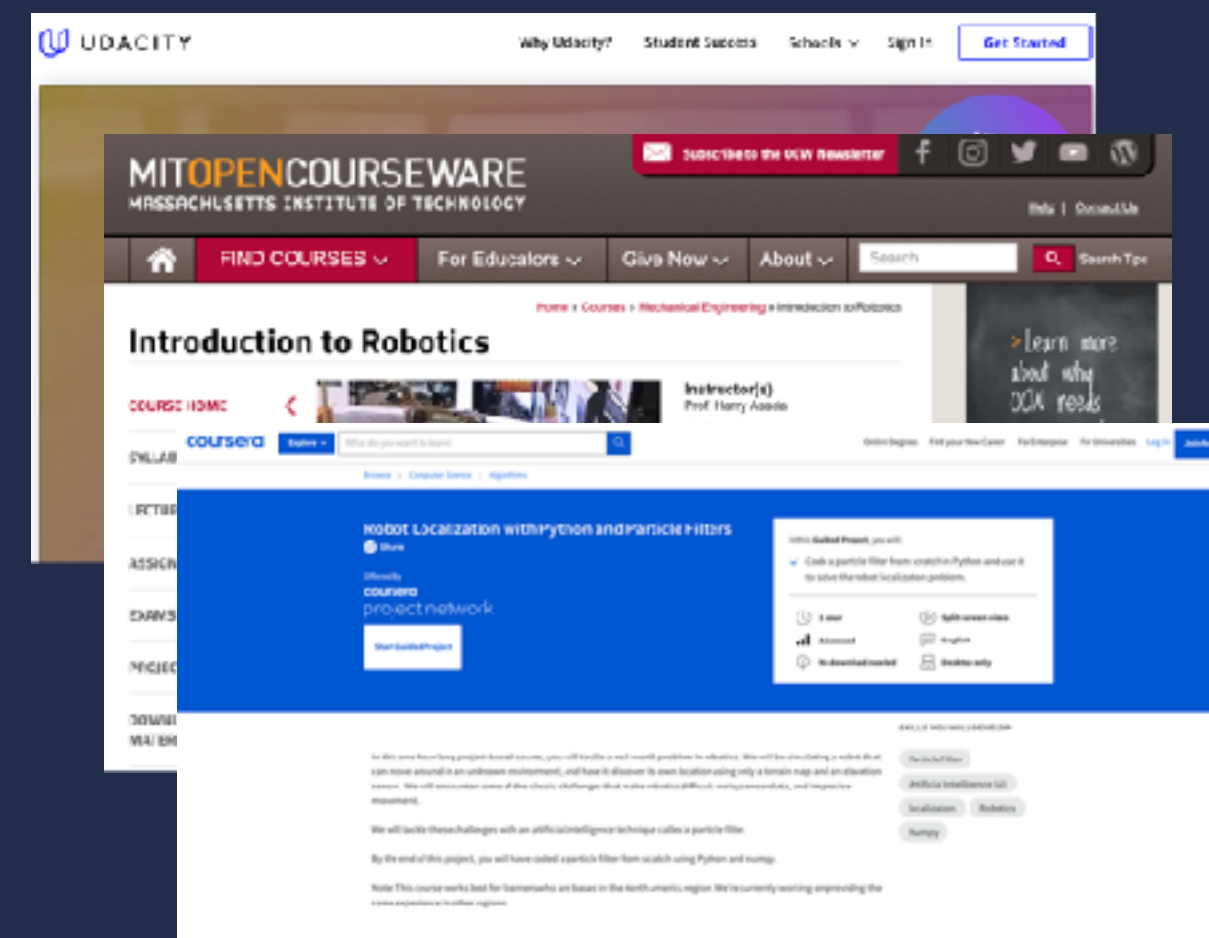
Overlooks that **robotics** heavily **relies on software**.

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

CPS/Embedded Systems Courses



Massive Open Online Courses



Traditional SE Courses

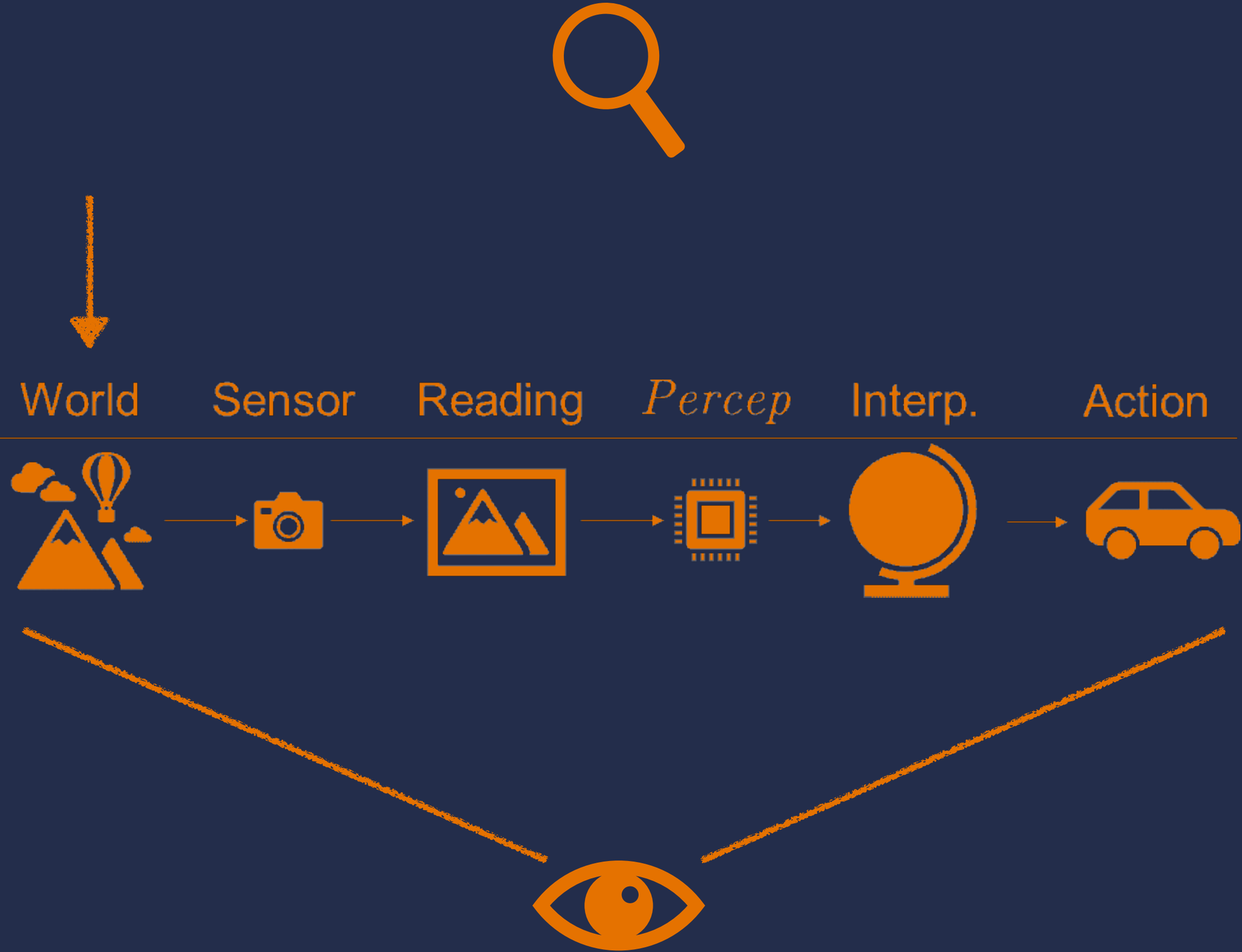
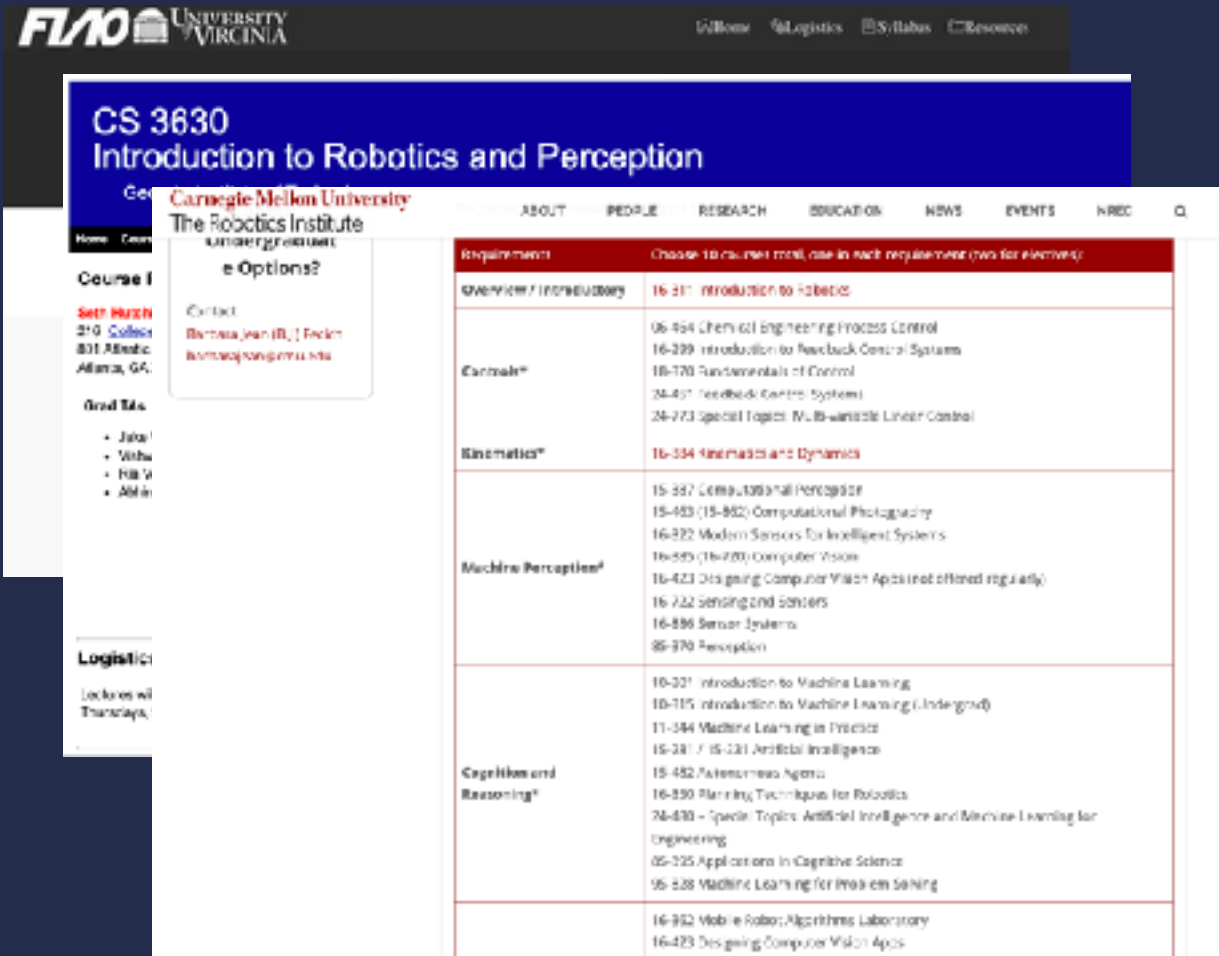




# Traditional CS Curricula

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**.

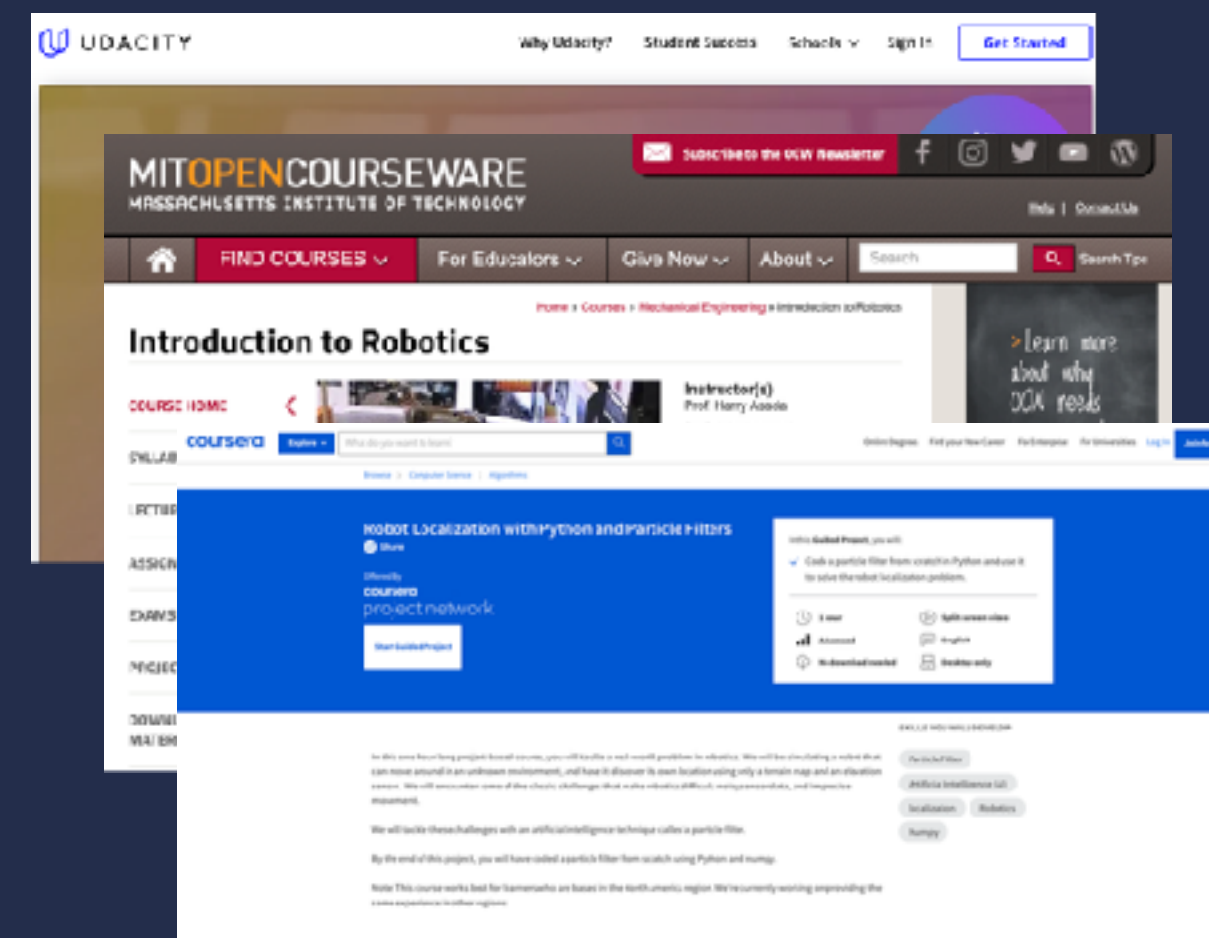
## CPS/Embedded Systems Courses



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

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

## Massive Open Online Courses



## Traditional SE Courses



### 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 hails 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 -

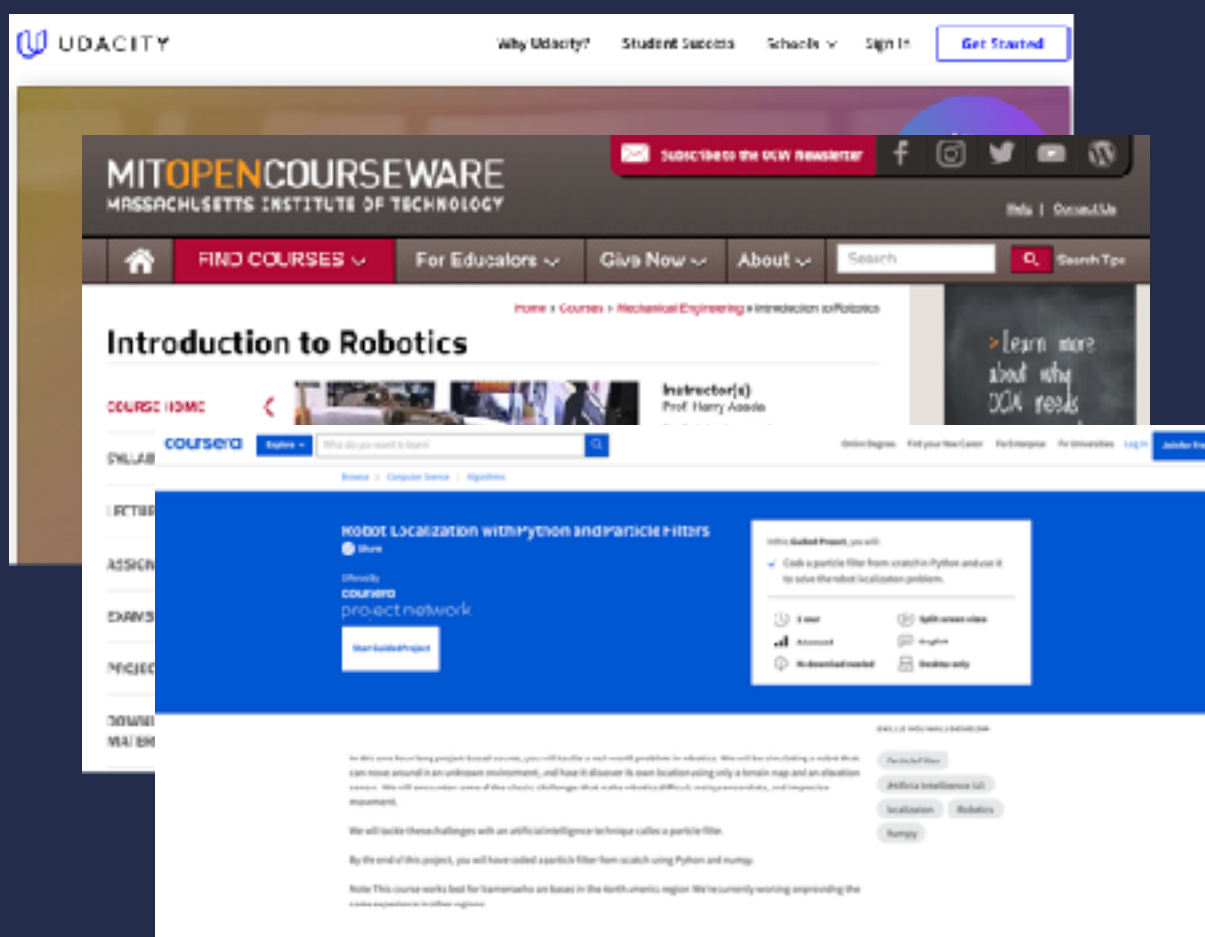
Create innovations for computer games, business applications, operating systems, network control systems, and much more.



# 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**.

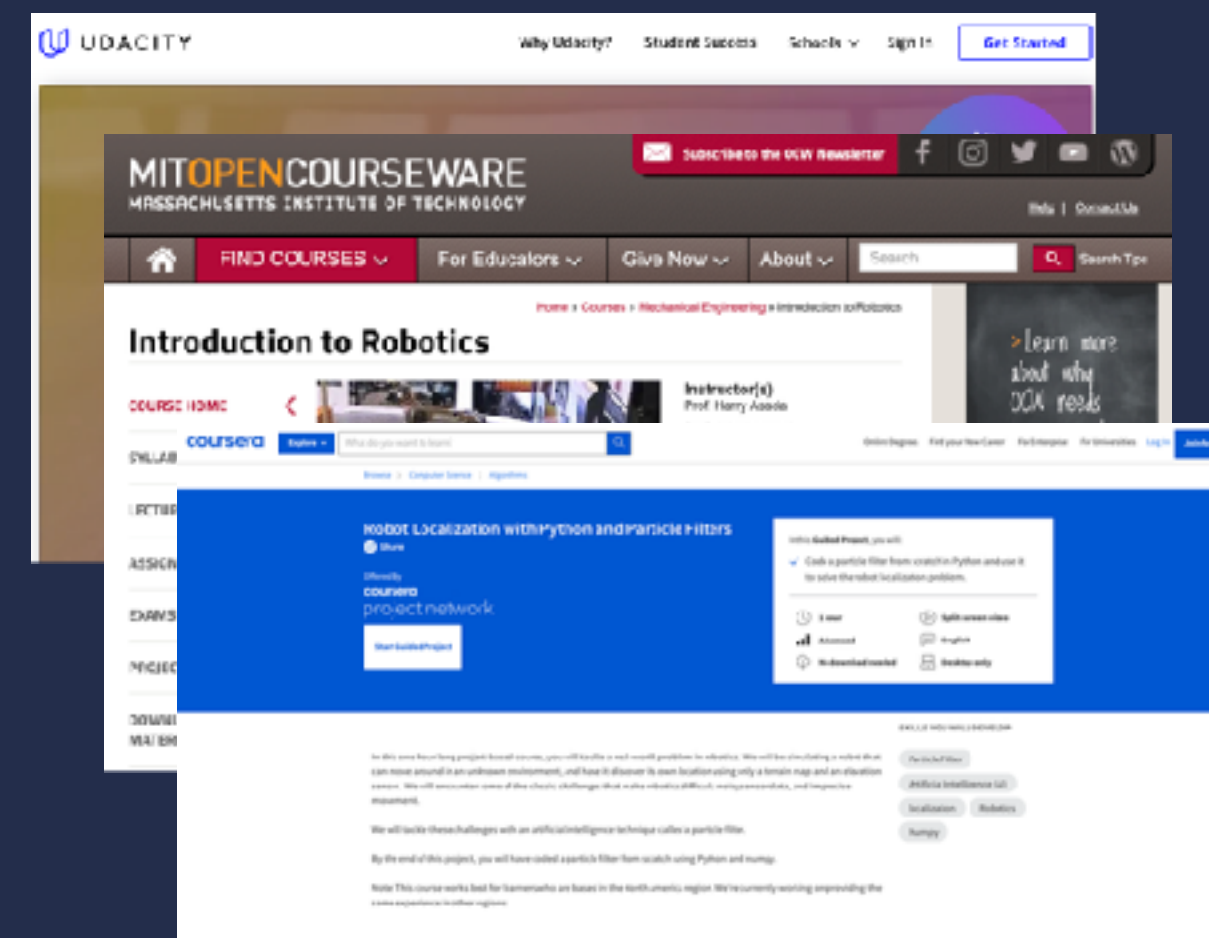
## CPS/Embedded Systems Courses



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



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

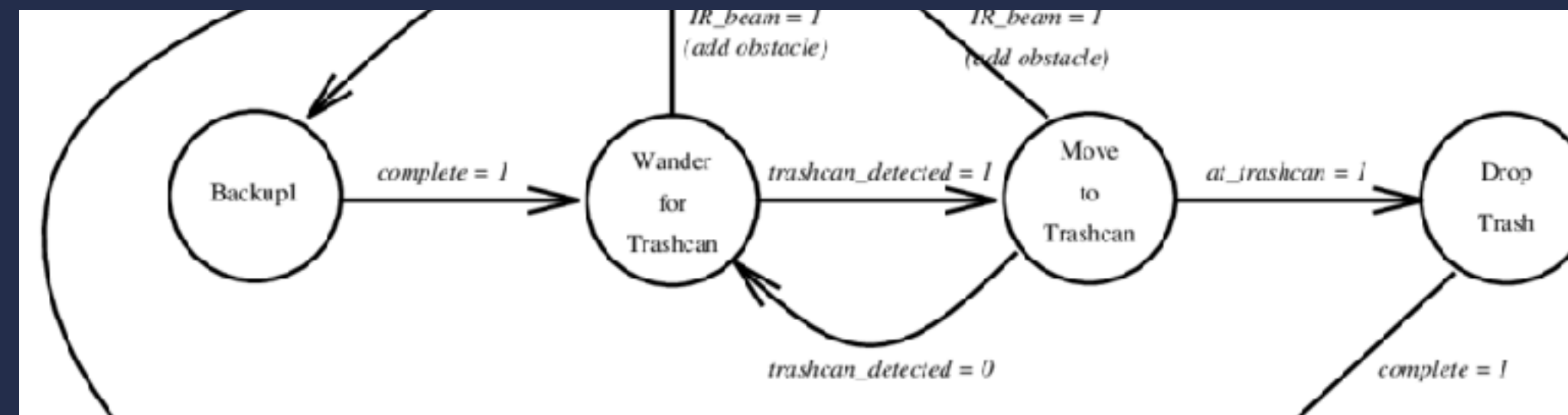
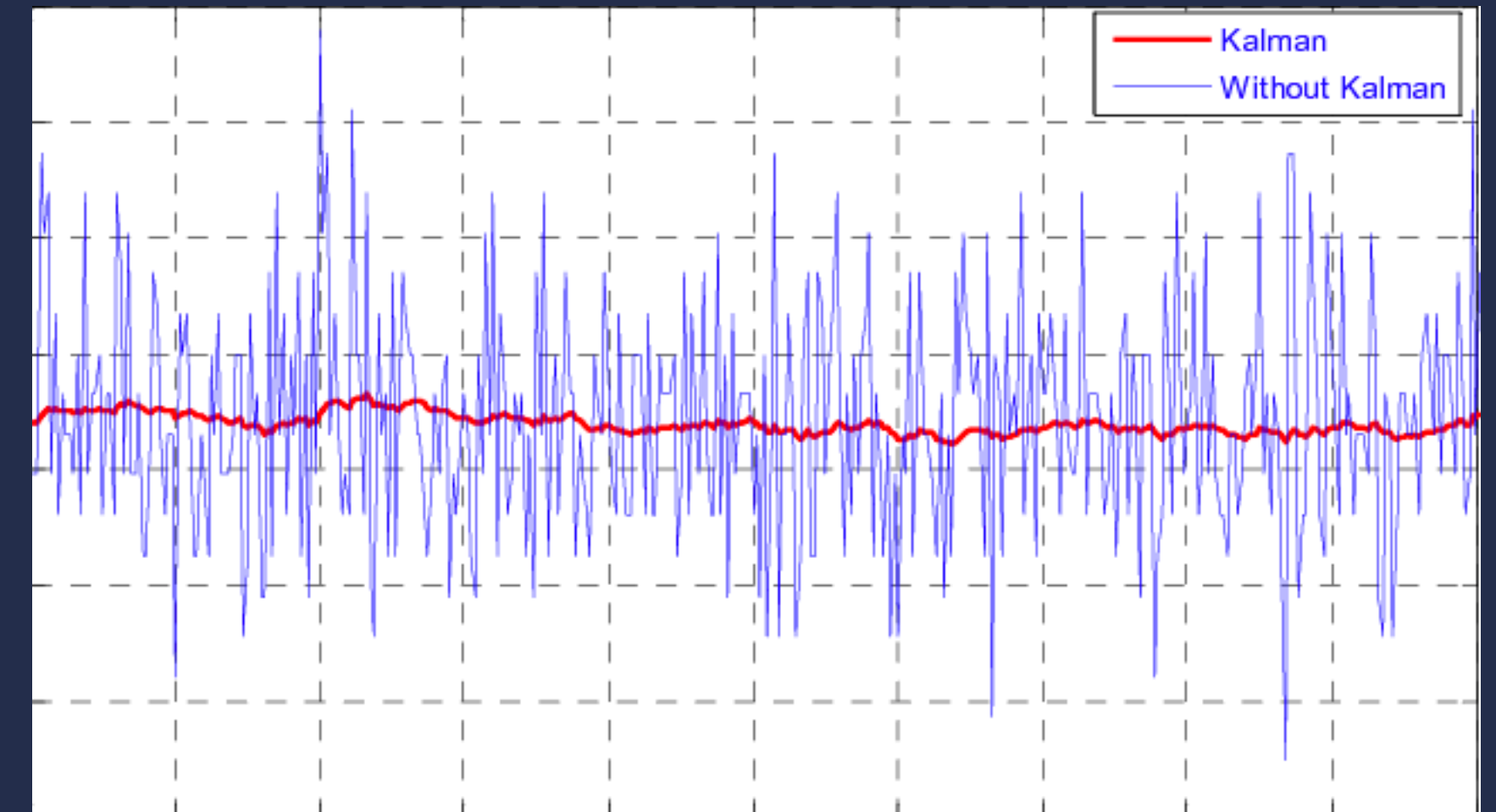
**- Improve software -**



# Traditional CS Curricula

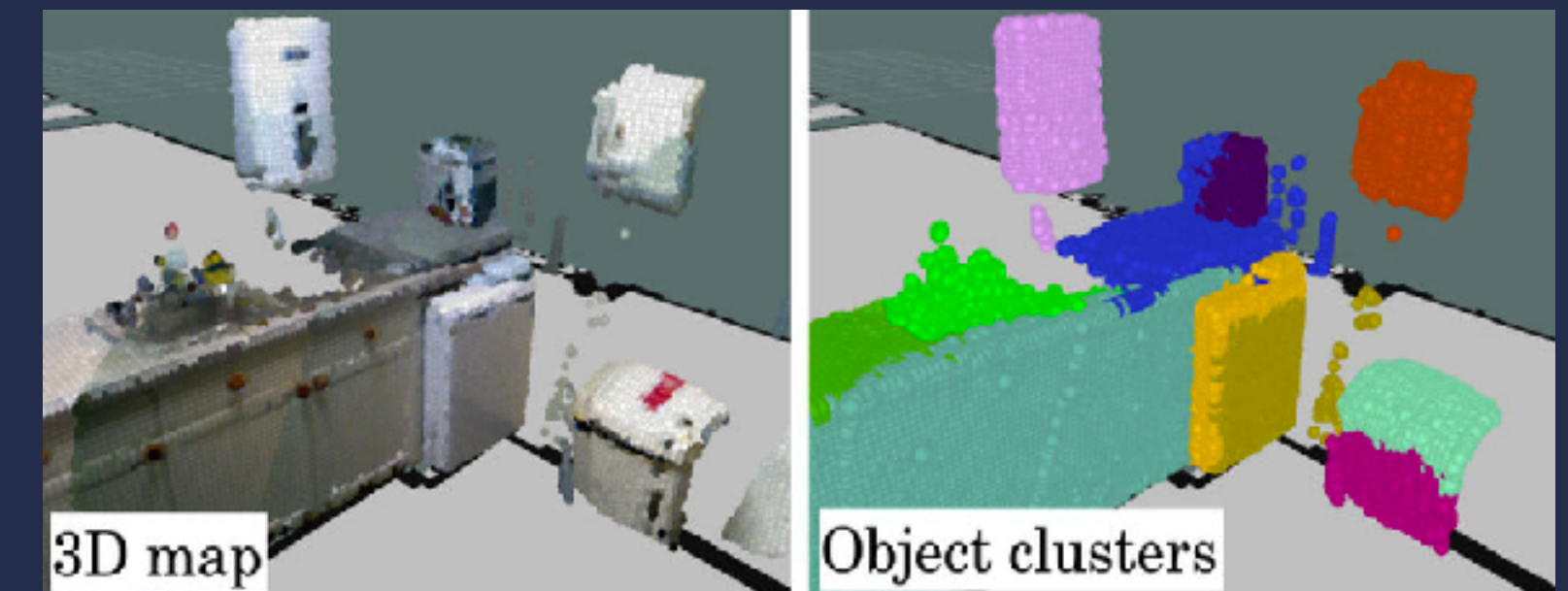
Preparing our future software engineers for this has been met primarily through:

## 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**.

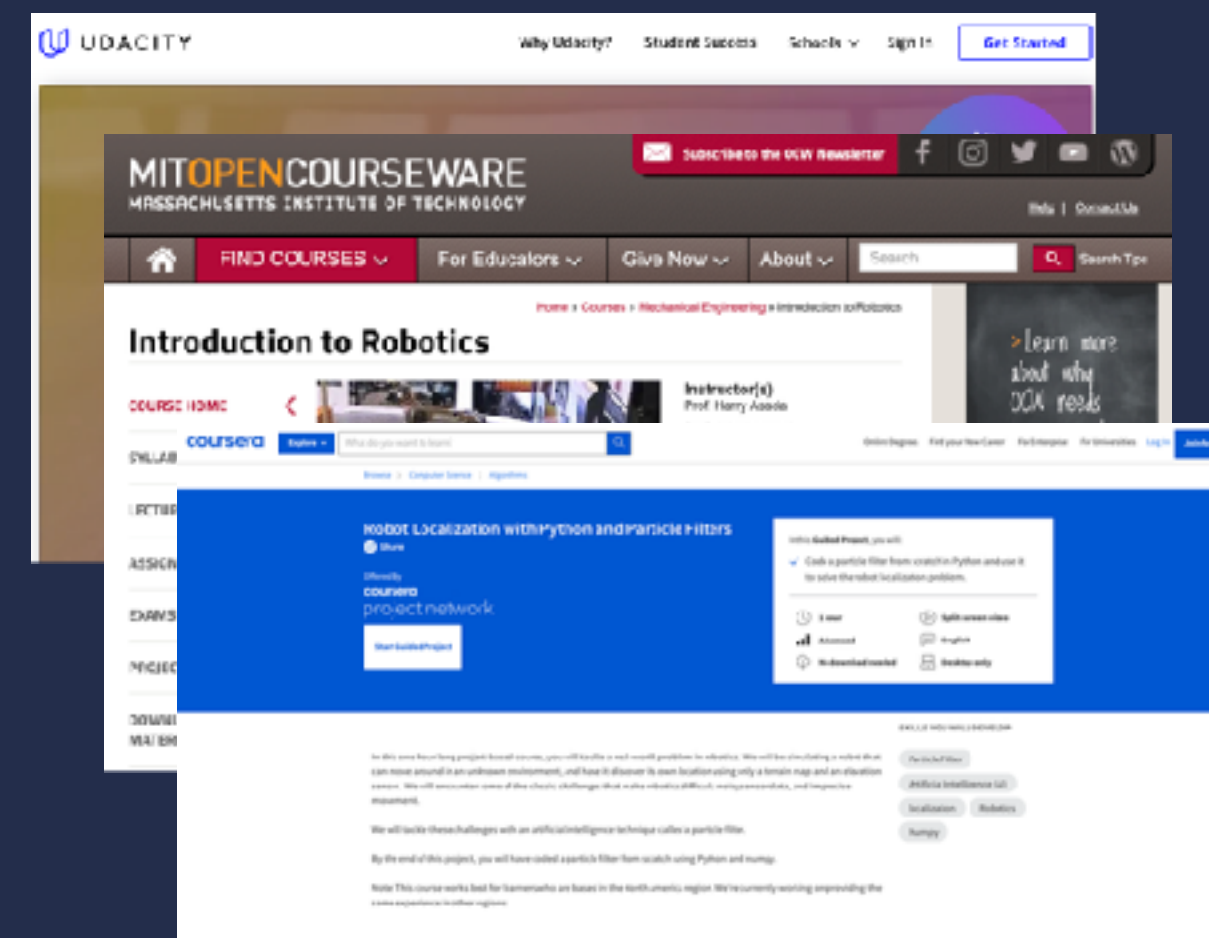
## CPS/Embedded Systems Courses



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**



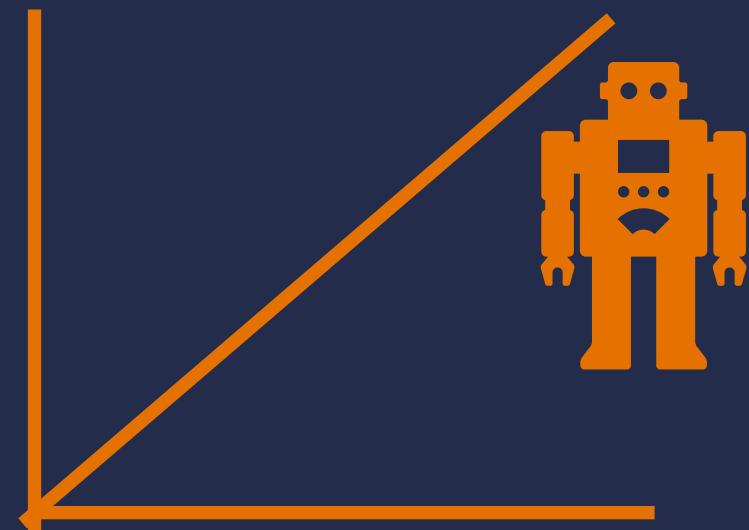
# Whats New?

Link between Software Engineering and Robotics

Specifications



Testing



Uncertainty  
representation



Design patterns



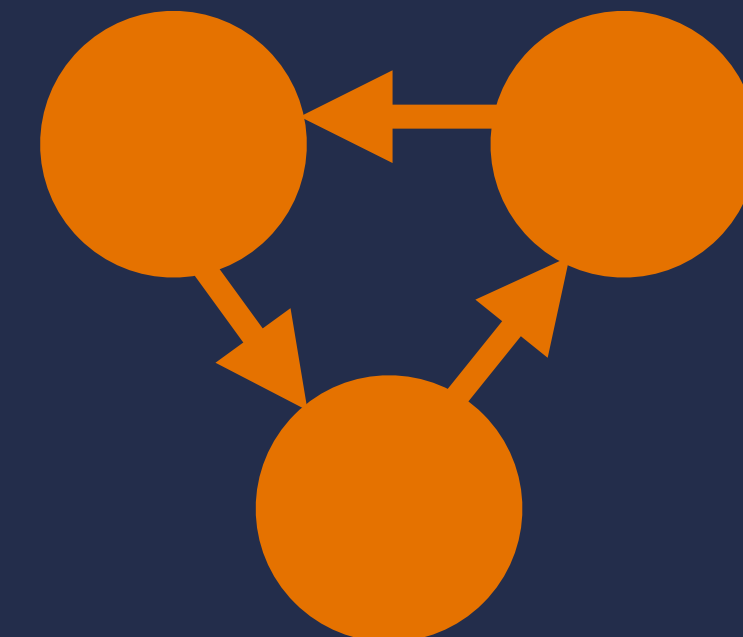
Reuse



Abstractions



States



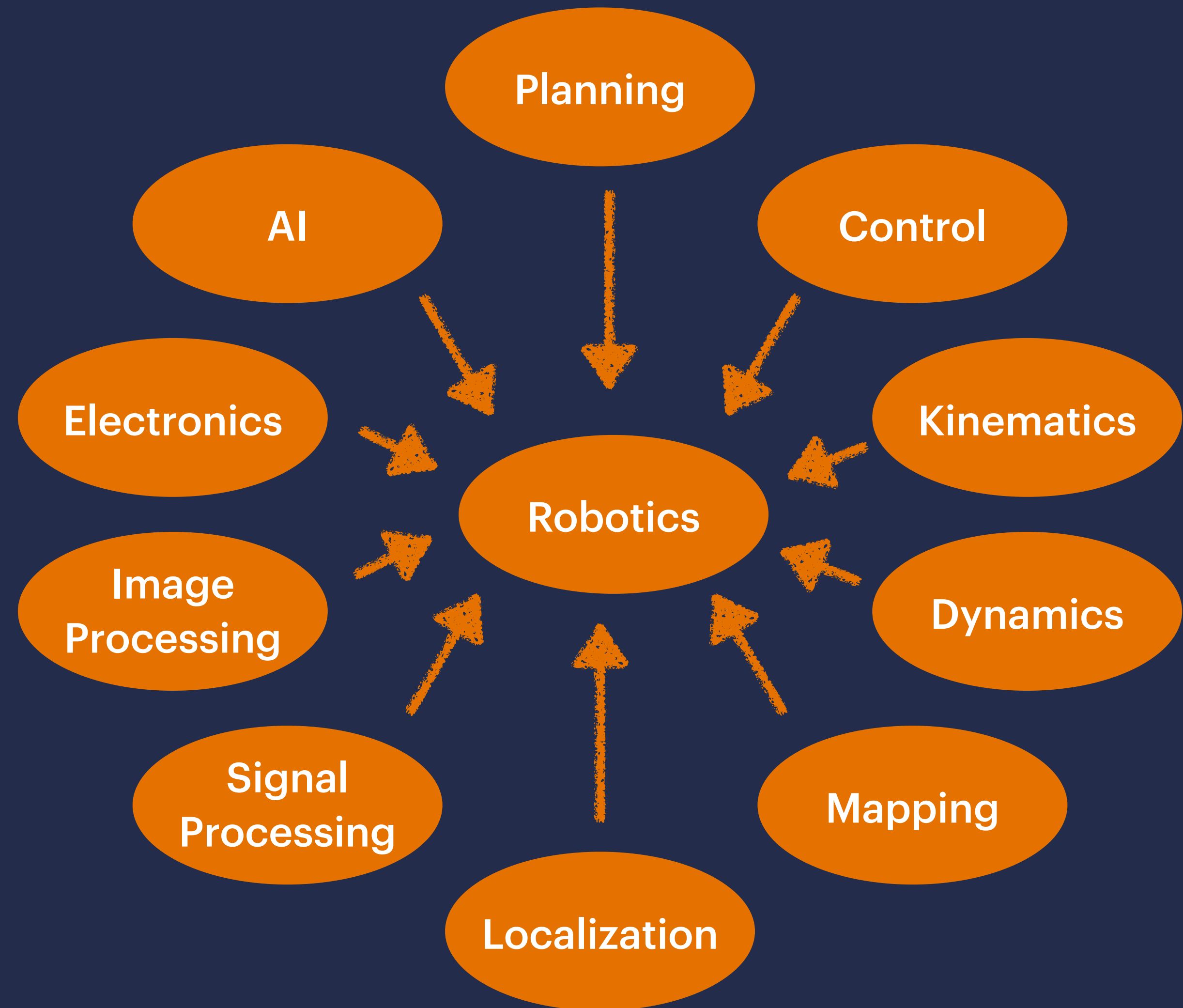


# Challenges

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

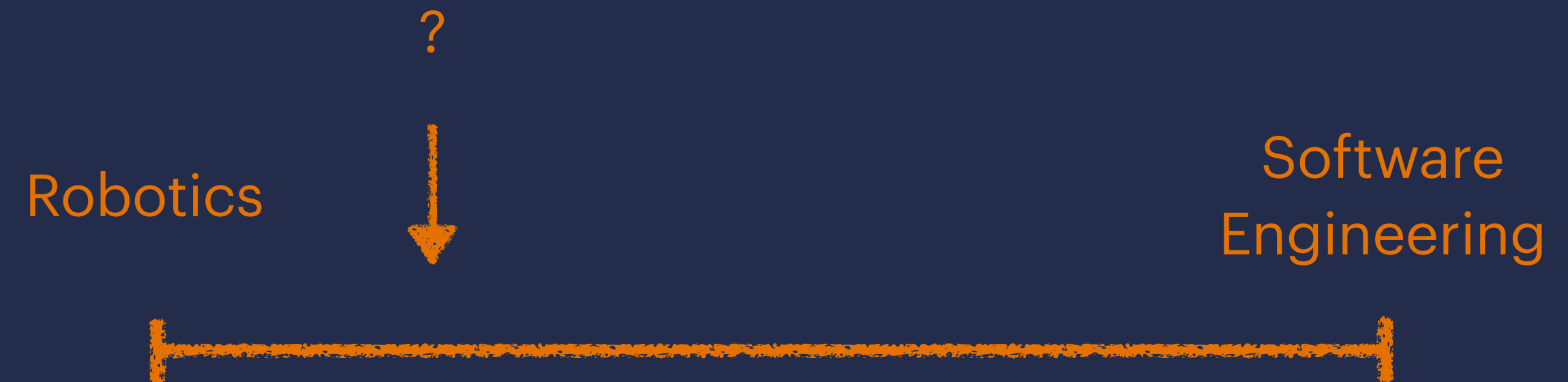
1. Multidisciplinary and rapidly expanding field
2. How to distribute the emphasis between robotics and software engineering
3. No available integrated platform
4. Robotics courses can require significant upfront investment in equipment





# 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



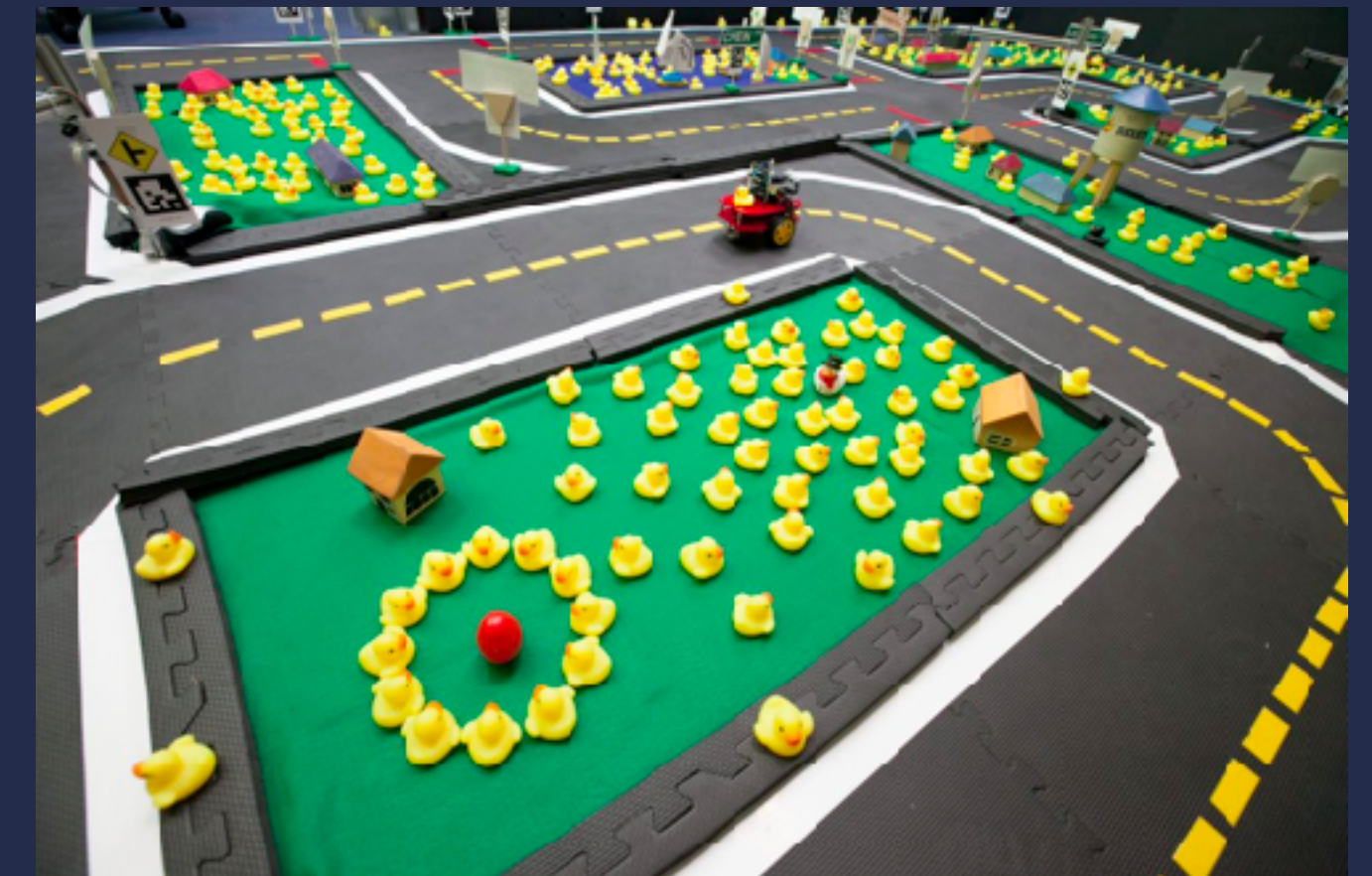
# Challenges

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

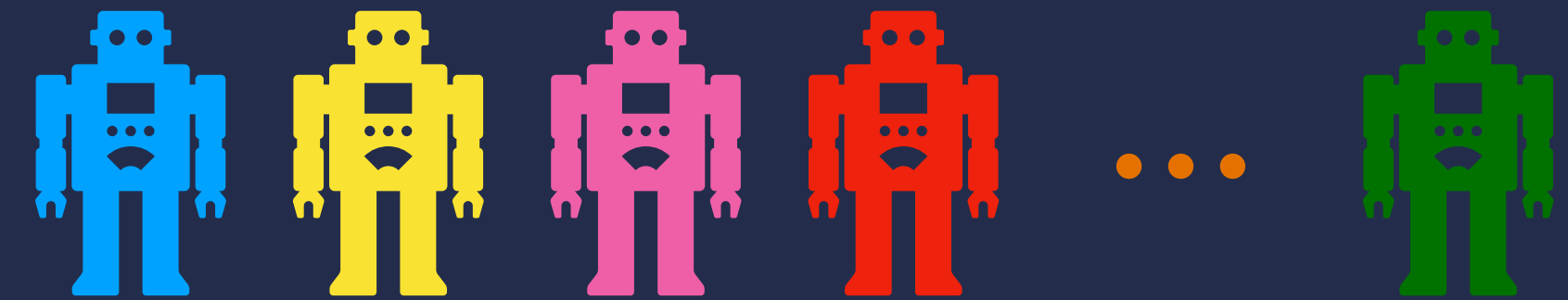
Duckytown





# 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

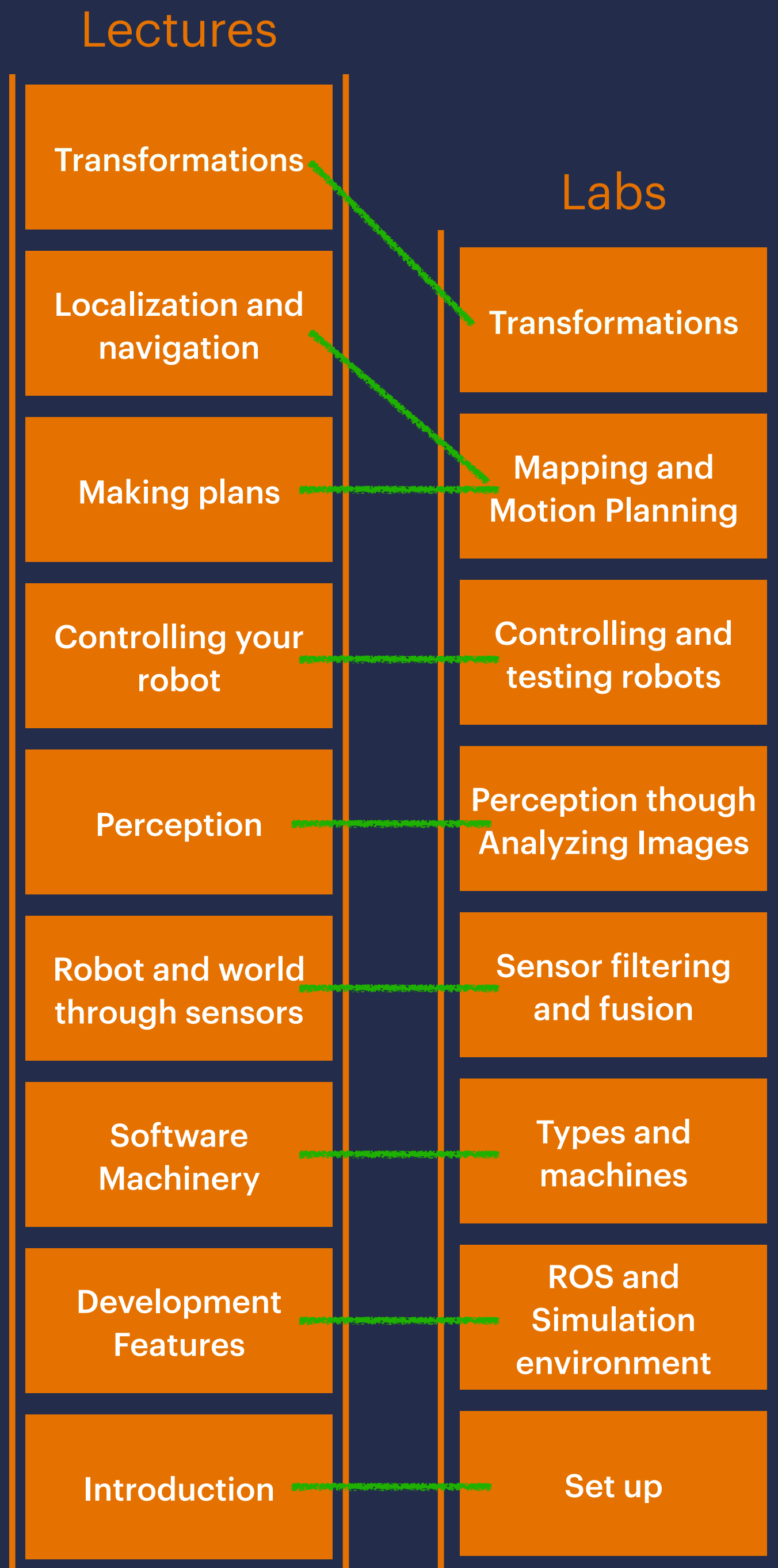
# Principles

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

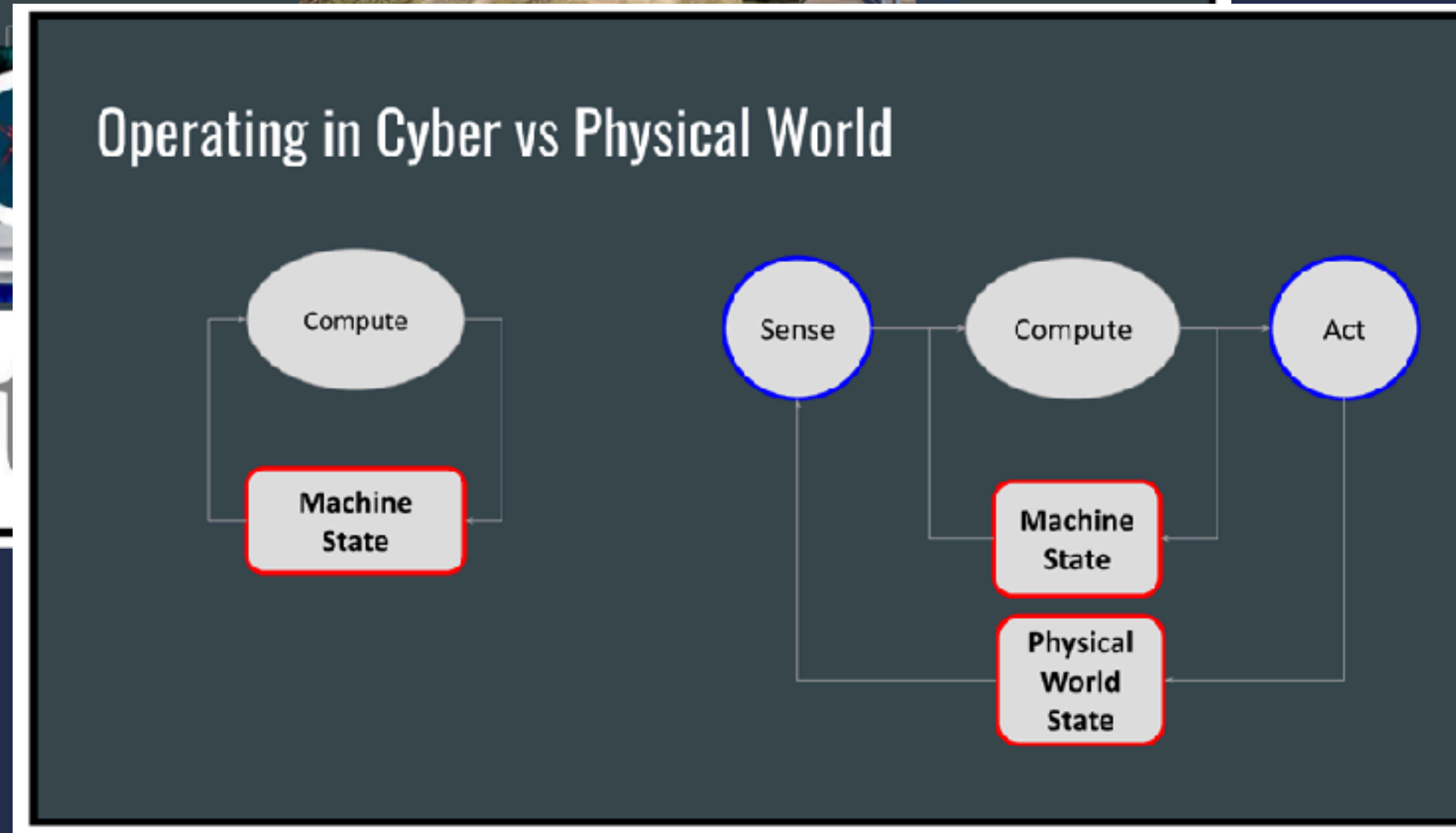
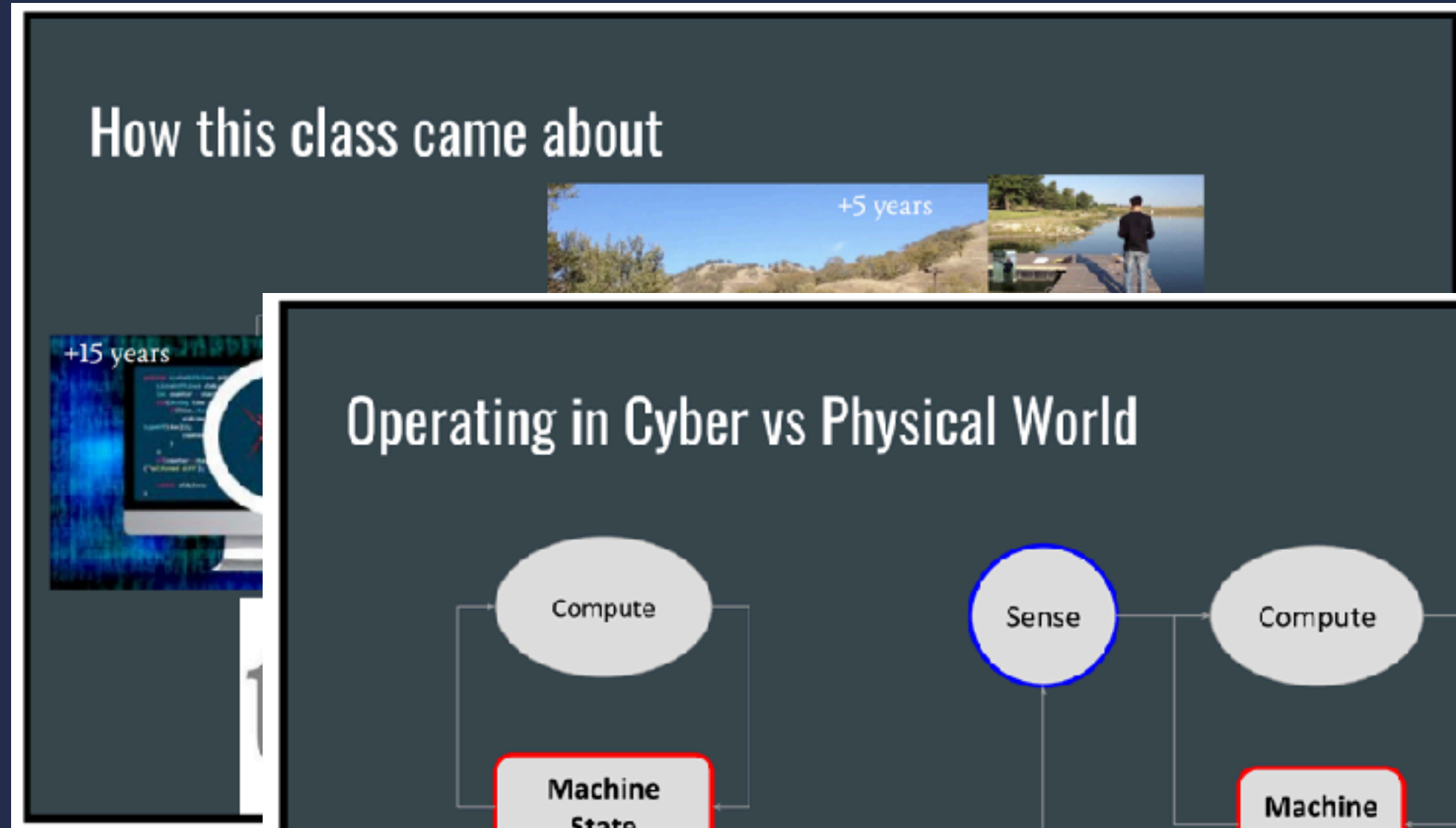
Lectures

# Lecture Lab Pairing

Labs

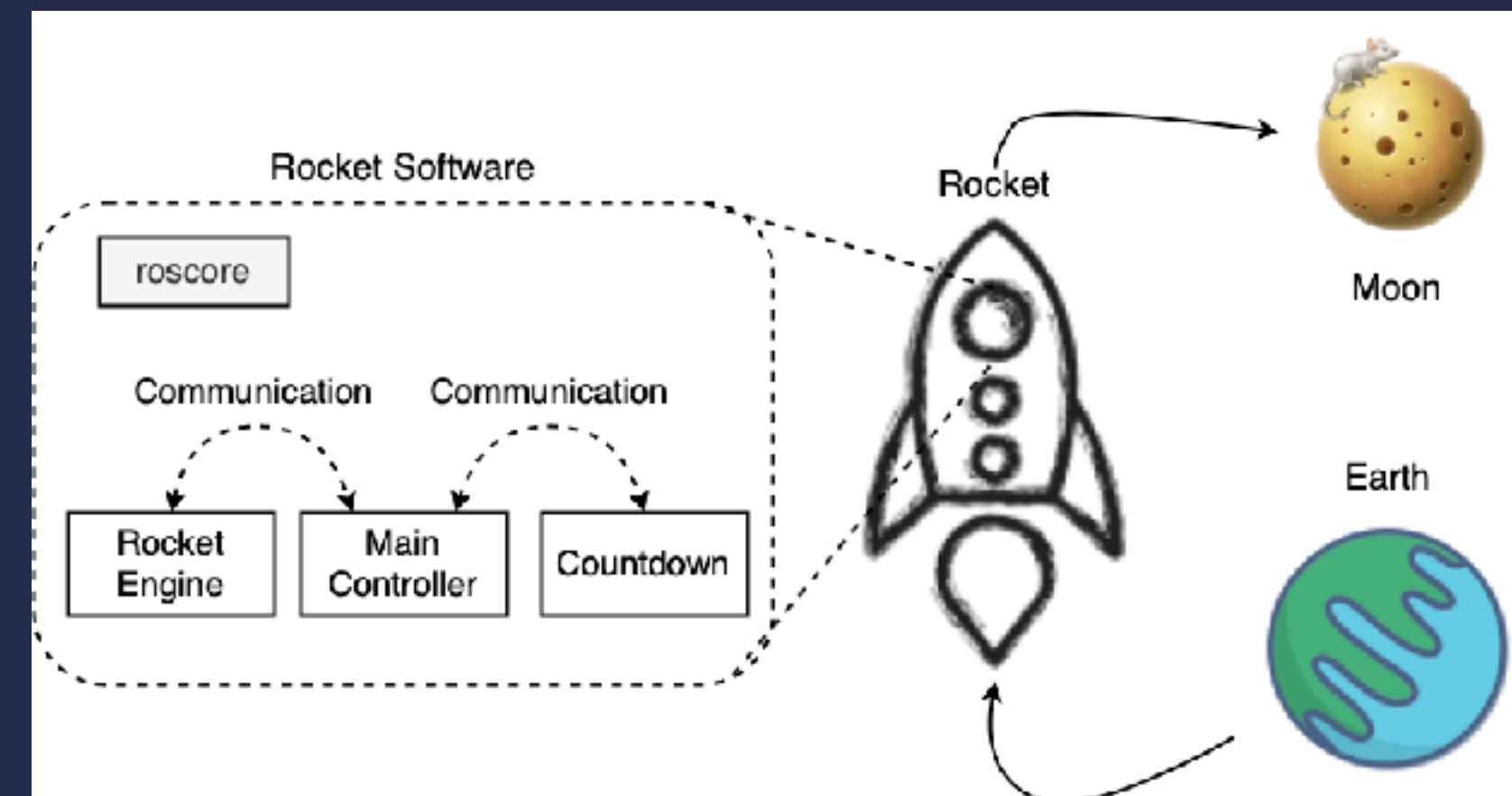


# Lecture Lab Pairing

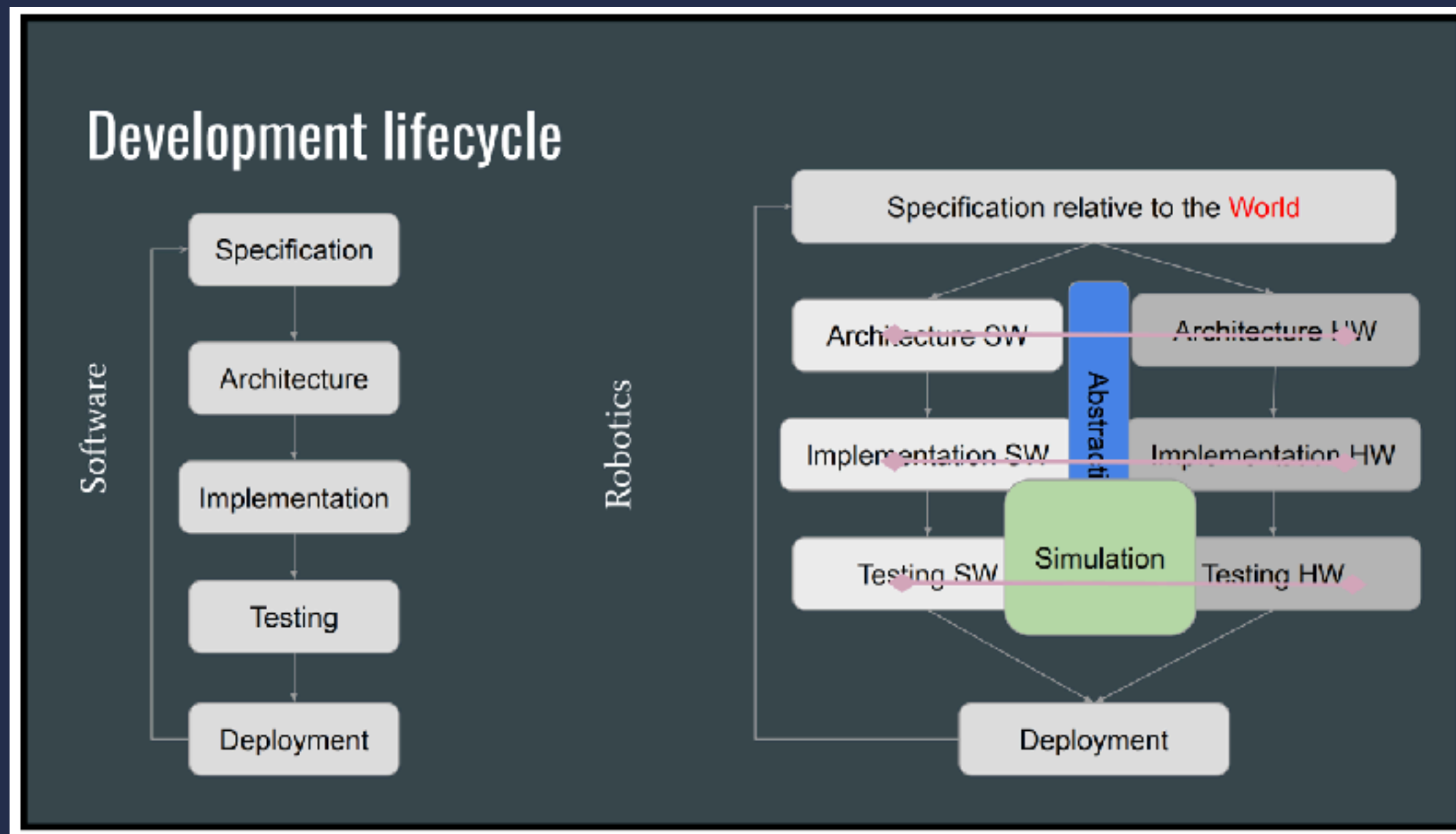


Introducing robotics

Introduction to ROS

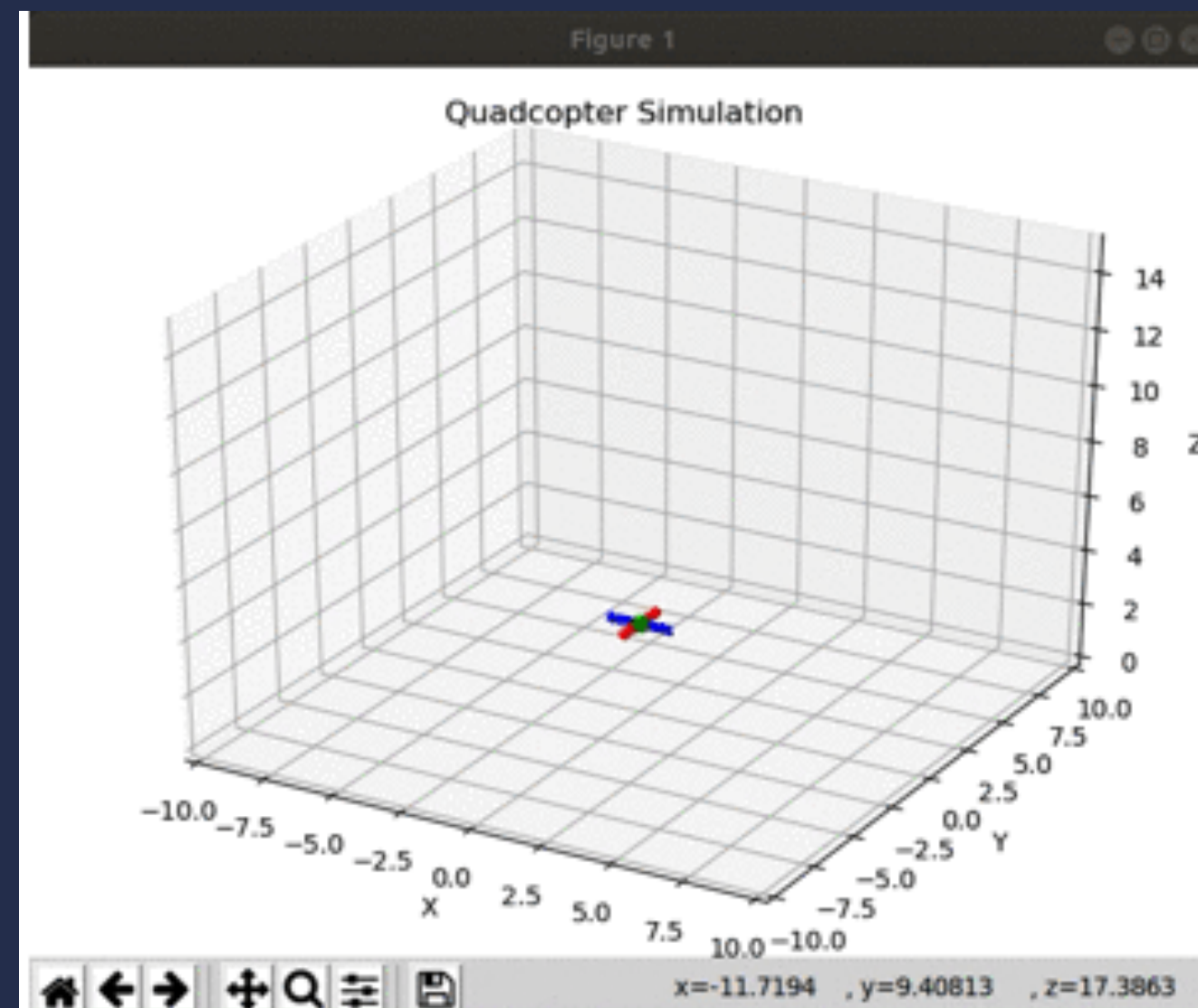


# Lecture Lab Pairing



Describing the development lifecycle

Introducing the simulation



```

Our Workspace
├── Lab2_p2_ws
│   ├── README.md
│   └── src
│       ├── flightcontroller
│       │   ├── CMakeLists.txt
│       │   ├── launch
│       │   │   ├── angle.launch
│       │   │   ├── fly.launch
│       │   │   ├── position.launch
│       │   │   └── velocity.launch
│       │   ├── package.xml
│       │   ├── rviz
│       │   │   └── viewer.rviz
│       │   └── src
│       │       ├── angle_calculator.py
│       │       ├── angle_controller.py
│       │       ├── pid_class.py
│       │       ├── pid_class.pyc
│       │       ├── position_controller.py
│       │       └── velocity_controller.py
│       └── flightgoggles
│           ├── flightgoggles
│           │   ├── CMakeLists.txt
│           │   ├── config
│           │   │   └── drone
│           │   ├── launch
│           │   │   ├── core.launch
│           │   │   └── package.xml
│           │   ├── flightgoggles_uav_dynamics
│           │   │   ├── CMakeLists.txt
│           │   │   ├── libs
│           │   │   │   └── multicopterDynamicsSim
│           │   │   ├── package.xml
│           │   │   └── src
│           │   │       ├── flightgoggles_uav_dynamics_node.cpp
│           │   │       └── flightgoggles_uav_dynamics_node.hpp

```

ROS and Simulation environment

Set up

Development Features

Introduction





# Lecture Lab Pairing

**Calibration**

Test 1    Test 2    Test n

**MODEL**  
 $d = \frac{1}{2} * v * t$   
 $t$  is measured  
 $v$  is known constant  
 Let's say this  
 -  $v = 300$   
 - If  $t=0.0$

**Filtering Problem**

- Signal get
- Interfere
- Sensor pe

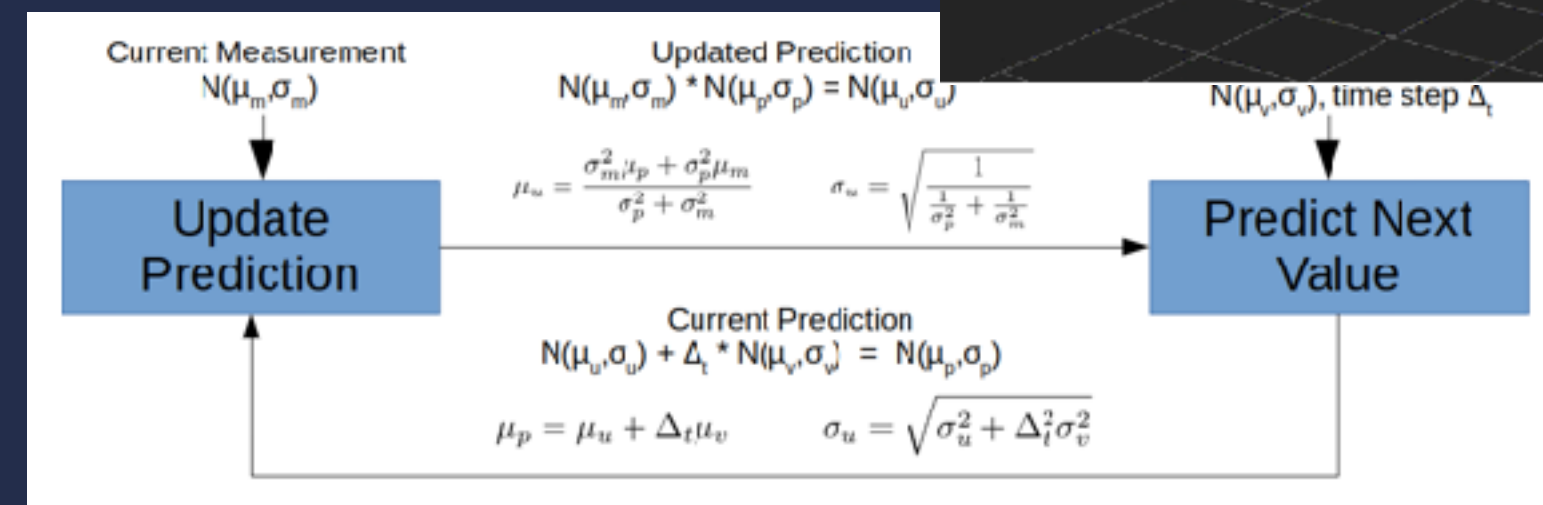
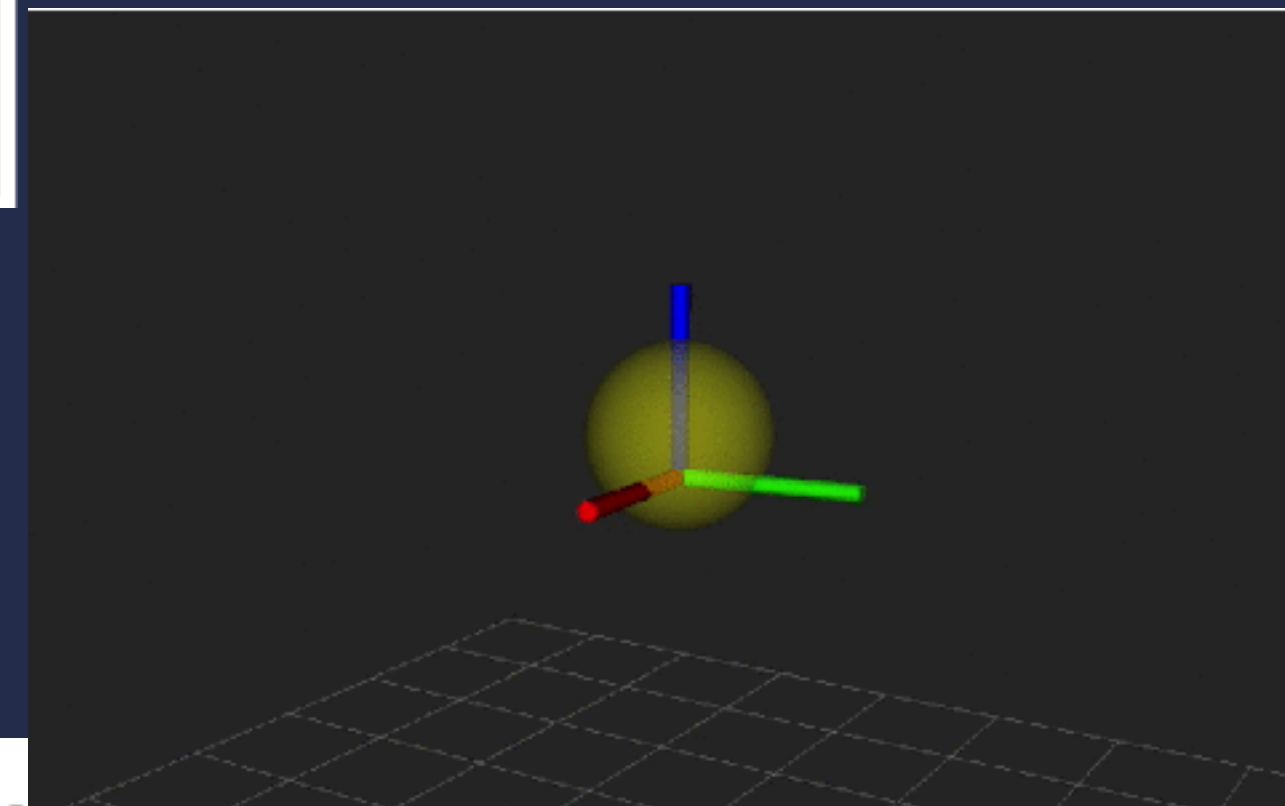
**What if our sensor is still really noisy?**

Add sensors with complementary attributes and fuse them

The diagram shows a drone on the left. In the center, three boxes represent different sensors: a barometric pressure sensor, a GPS receiver, and a satellite. Arrows from these three boxes point to a central box labeled 'FUSE'. An arrow from the 'FUSE' box points to the word 'Altitude'.

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

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



Robot and world through sensors

Software Machinery

Development Features

Introduction

Sensor filtering and fusion

Types and machines

ROS and Simulation environment

Set up



# Lecture Lab Pairing

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

**Image Processing Techniques**

- Thi
- Co
- Blt
- Str
- Ba
- Ed
- Co
- Fei
- Ha
- ...

**Machine Learning**  
What happens if we have a much more complicated task?

Machine learning learns this function given enough data

Input →  $output = f(input)$  → Output

3779

Plane, Car, Bird, Cat

Introducing different types of perception

Perception

Robot and world through sensors

Software Machinery

Development Features

Introduction

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

Perception though Analyzing Images

Sensor filtering and fusion

Types and machines

ROS and Simulation environment

Set up



# Lecture Lab Pairing

## Open-loop controller

- Assumes we
- Computes u

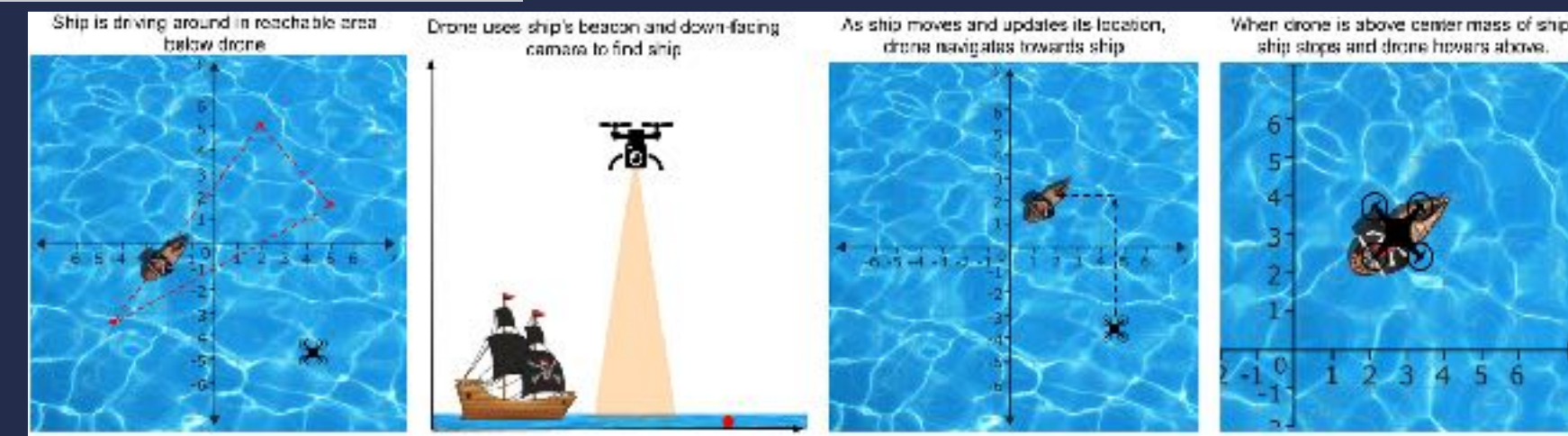
## Close-Loop Controller

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



Introducing different control schemes

Using the control to implement a drone that follows a ship. Testing 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')
```

Controlling your robot

Perception

Robot and world through sensors

Software Machinery

Development Features

Introduction

Controlling and testing robots

Perception though Analyzing Images

Sensor filtering and fusion

Types and machines

ROS and Simulation environment

Set up

# Lecture Lab Pairing

Localization and navigation

Making plans

Controlling your robot

Perception

Robot and world through sensors

Software Machinery

Development Features

Introduction

Motion Planning Problem

Key data structures in ROS for motion

Occu

```
# Tr
# Ed
# Me
Map
# Tr
# pr
int8
```

### 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

```
carlhildebrandt@ubuntu:~$ rostopic pub /uav/input/goal geometry_msgs/Vector3 '{x: 4, y: 0, z: 5}'
```

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

Quadcopter Simulation

# 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



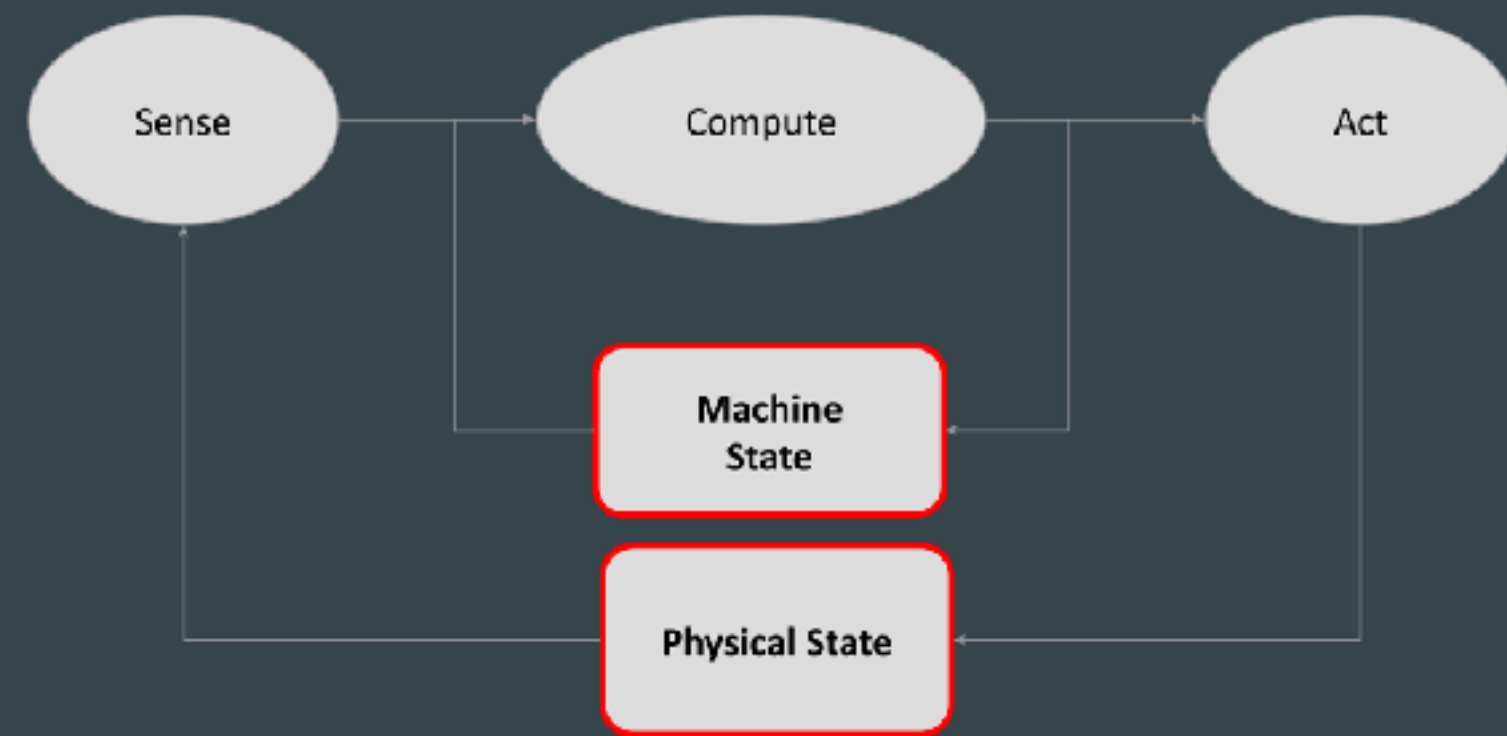
# Innovations

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

## Conceptual Architecture



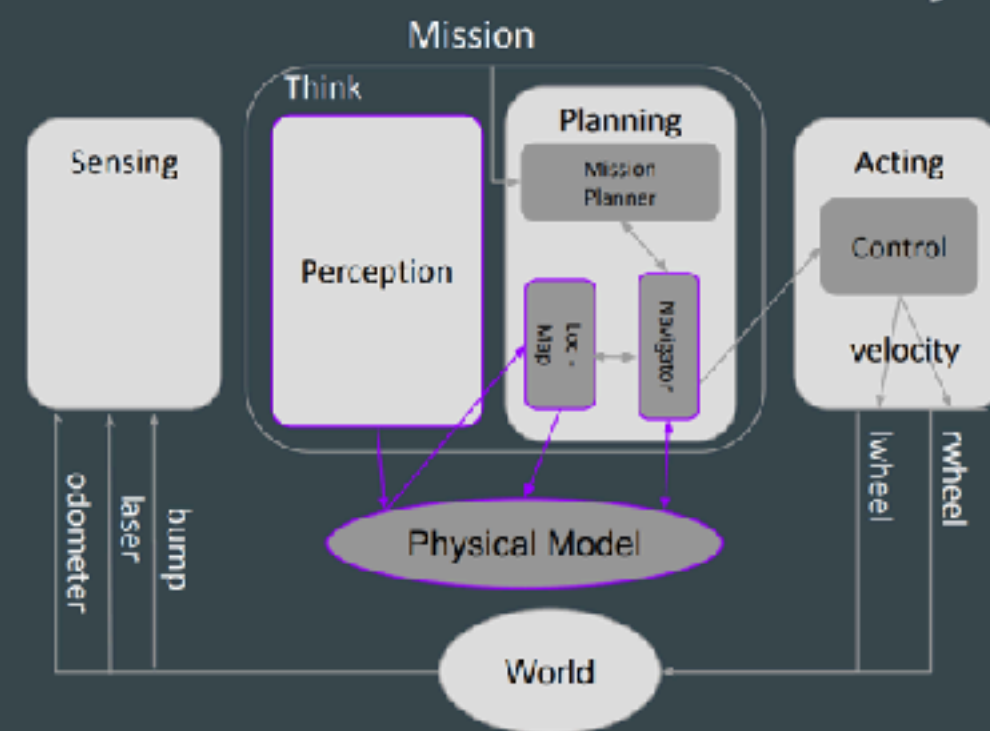
Aim: introduce the fundamental concepts related to robotics architecture and modeling machinery in robotics



# Example Lecture

## Conceptual Architecture

### Hierarchical/Deliberative my "Roomba"



- World is too complex to model accurately / completely
- World changes faster than we can plan for it
- Difficult to extend functionality due to layers dependencies

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

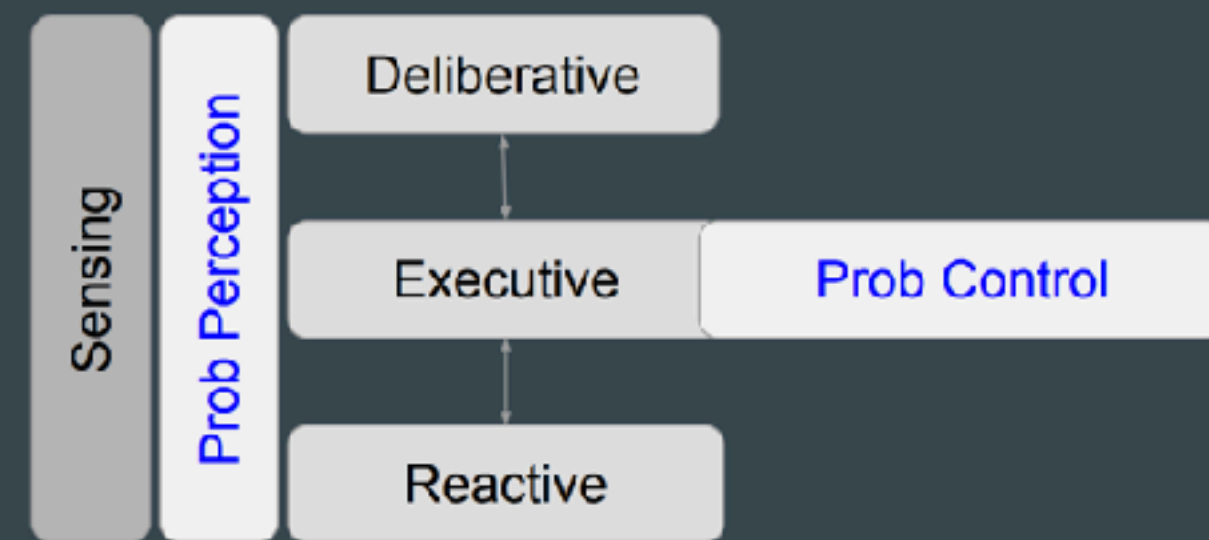
← Begins with the basic conceptual architecture of robotics  
(1 - Cover robotics fundamentals)

# Example Lecture

Conceptual Architecture

Hierarchical/Deliberative my "Roomba"  
Mission

Dominant Architectural Types: Probabilistic



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)

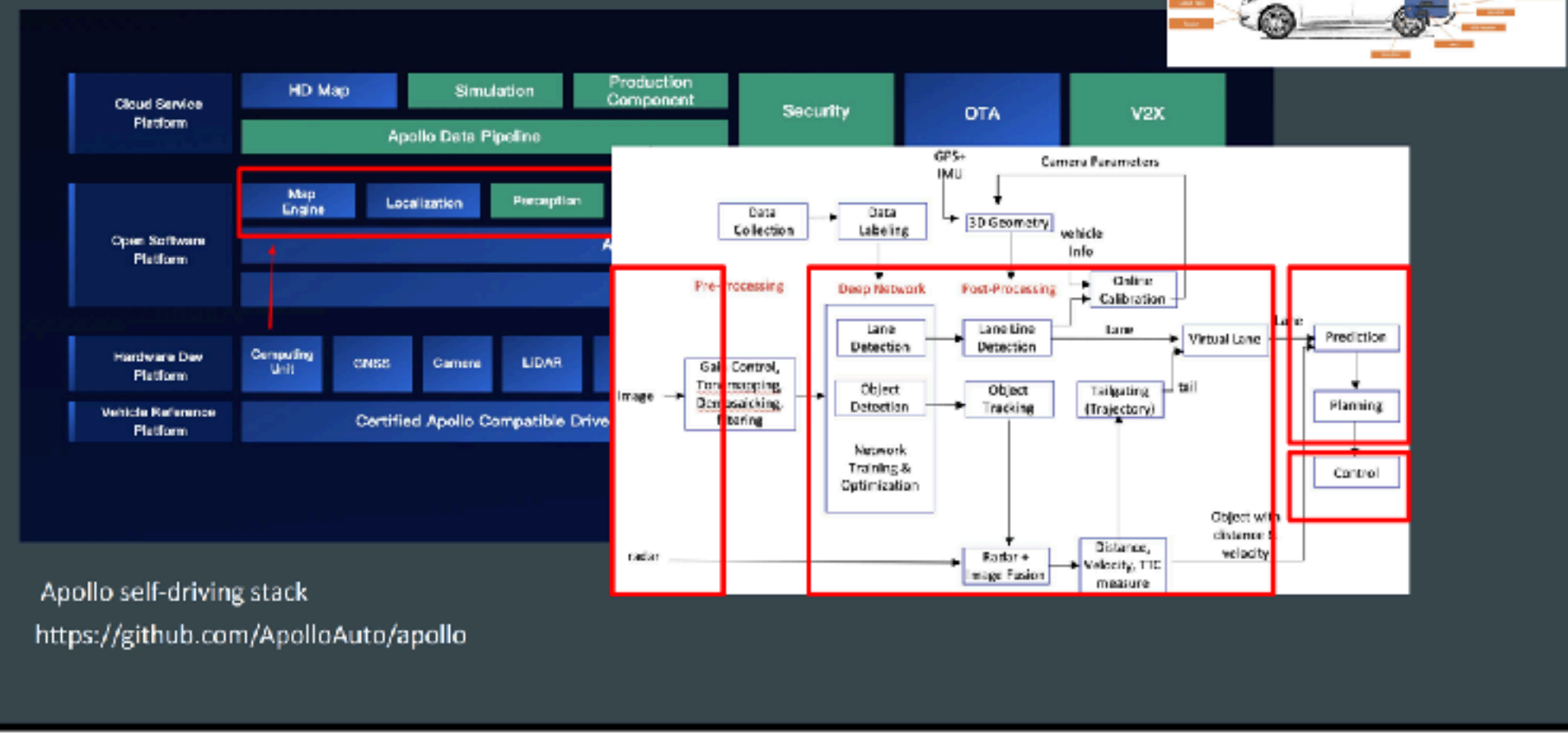
# Example Lecture

Conceptual Architecture

Hierarchical/Deliberative my "Roomba"

Dominant Architectural Types: Probabilistic

Reality is a bit messier



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

Conceptual Architecture

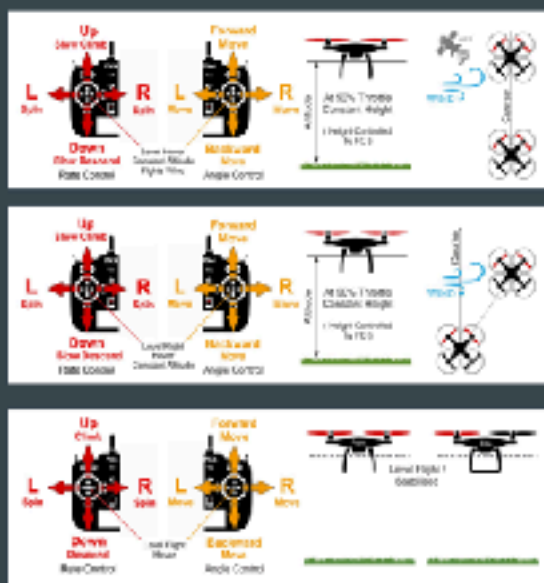
Hierarchical/Deliberative my "Roomba"  
Mission

Dominant Architectural Types: Probabilistic

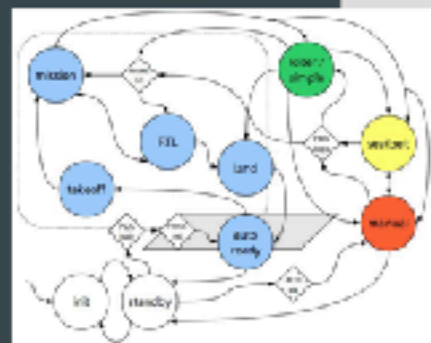
Reality is a bit messier



States and Machines



Different modes (states), imply different interpretation of commands



Conceptual  
<https://drones.com/prof1ez/blog/px4-flight-mode-switching-navigation-state-machine>

Close to code [https://docs.px4.io/master/en/concept/flight\\_modes.html](https://docs.px4.io/master/en/concept/flight_modes.html)

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)

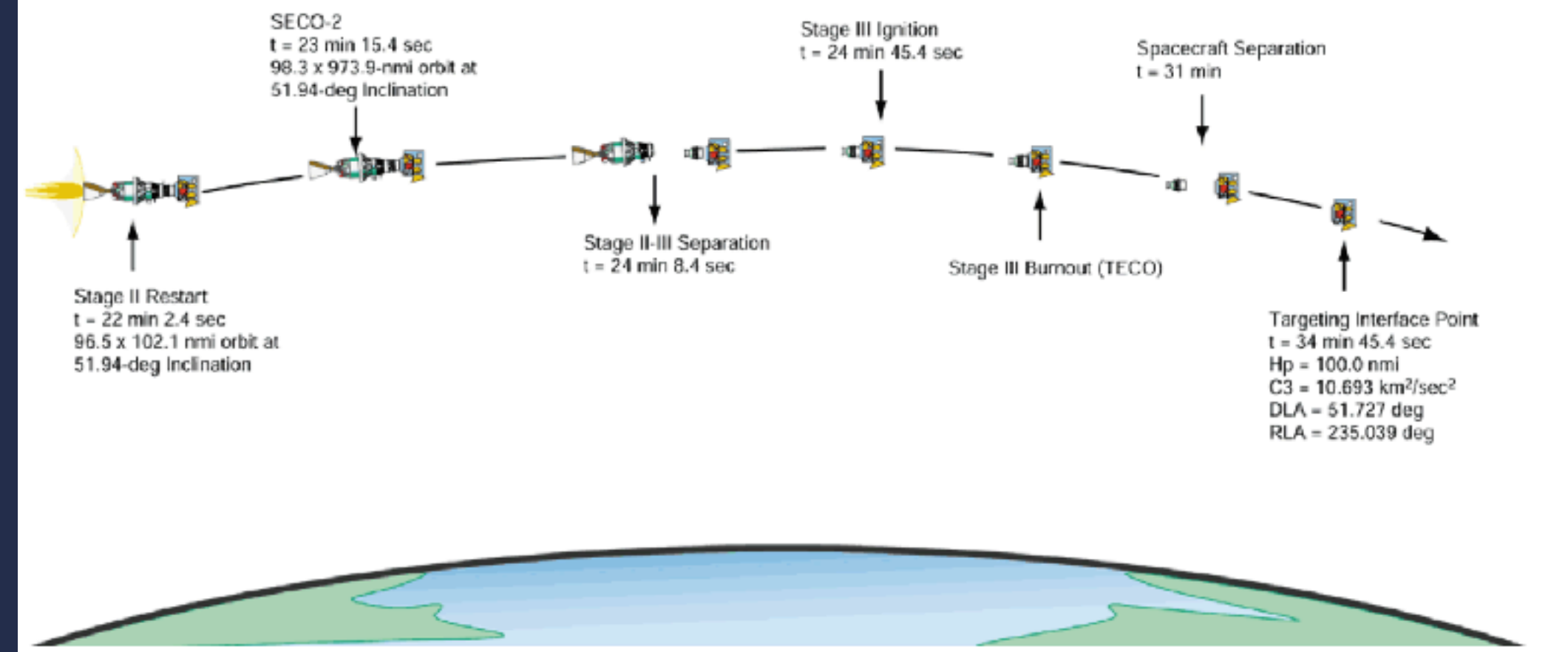


# Example Lab

## 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 stages, 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.

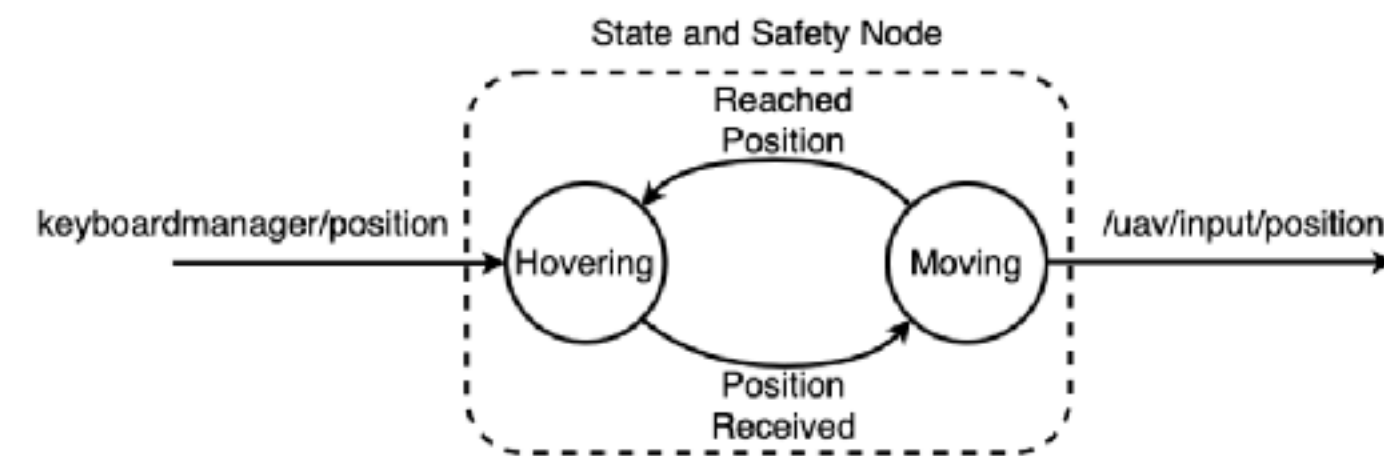
```
# Change to lab directory
$ cd ~/Desktop/C84501-Labs/
# Clone the code
$ git pull
```

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: `RUR(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

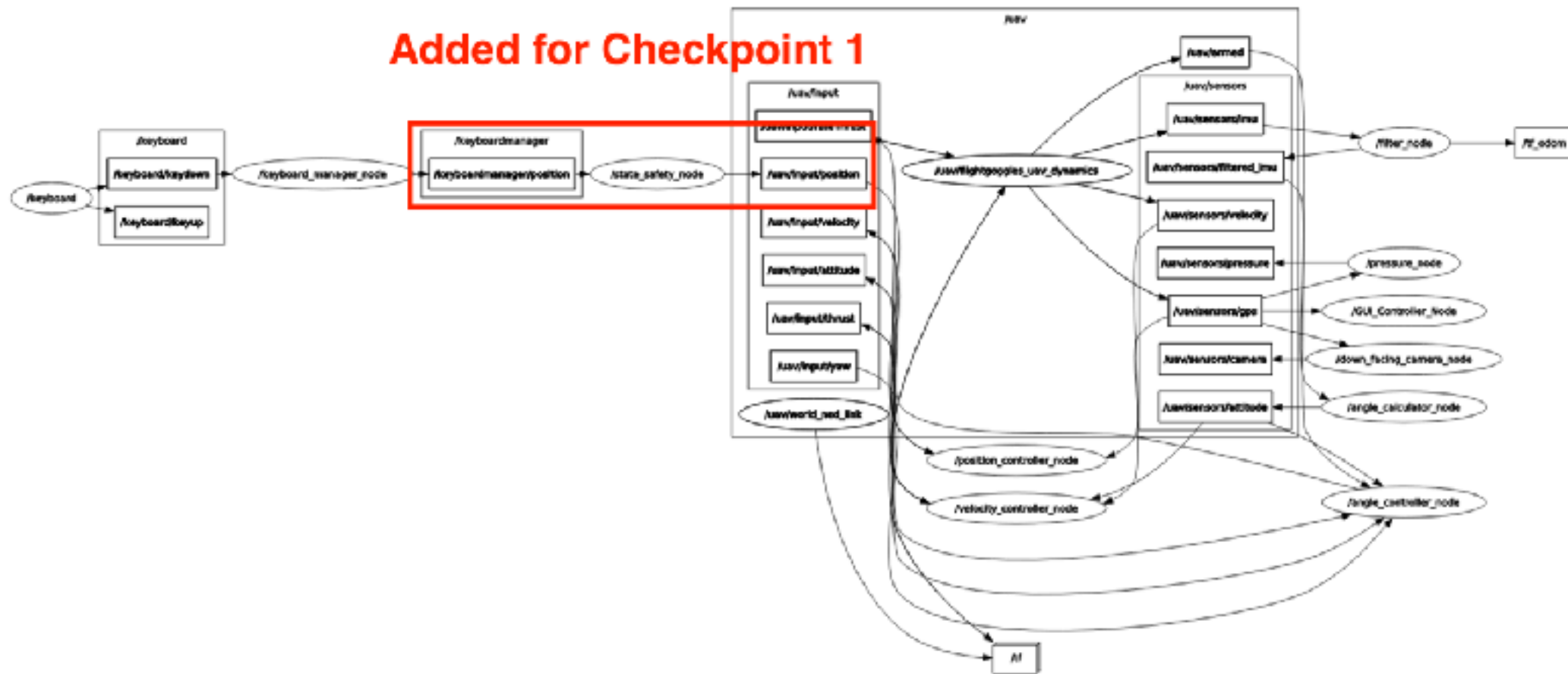
**Create the State and Safety Node**

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  
(13 - Pair SE and Robotics topic)

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

Checkpoint allowing reflection and discussion while allowing freedom to implement their own solution  
(14 - Students make design and implementation decisions)  
(15 - 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:

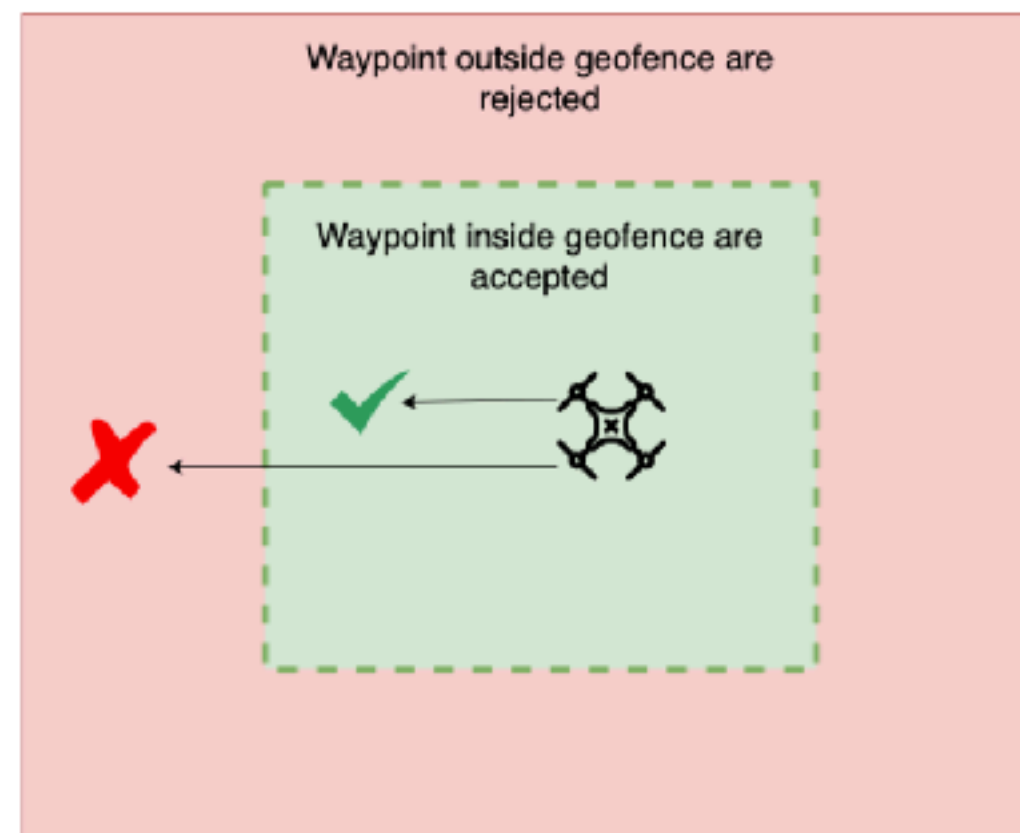
## A useful resource to notice: ROS Logging

## 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 console 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 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

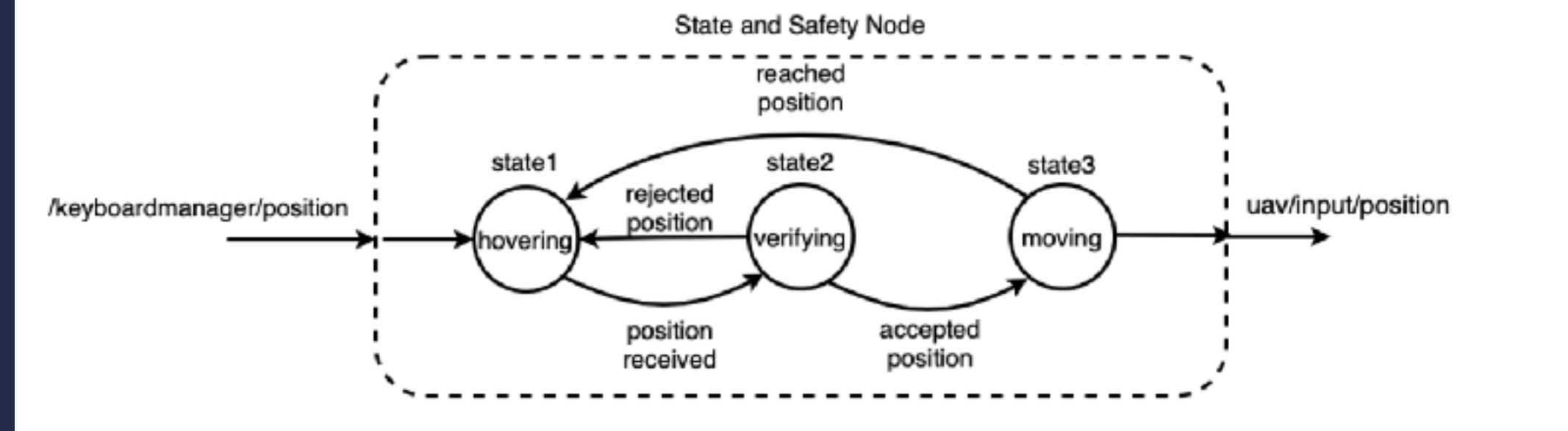
Launch the simulator and check that you have correctly created the state and safety node as well as changed the keyboard manager to publish the correct data. Your ROS environment should be able to find the node. To demonstrate that the ROS environment is working:

## 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

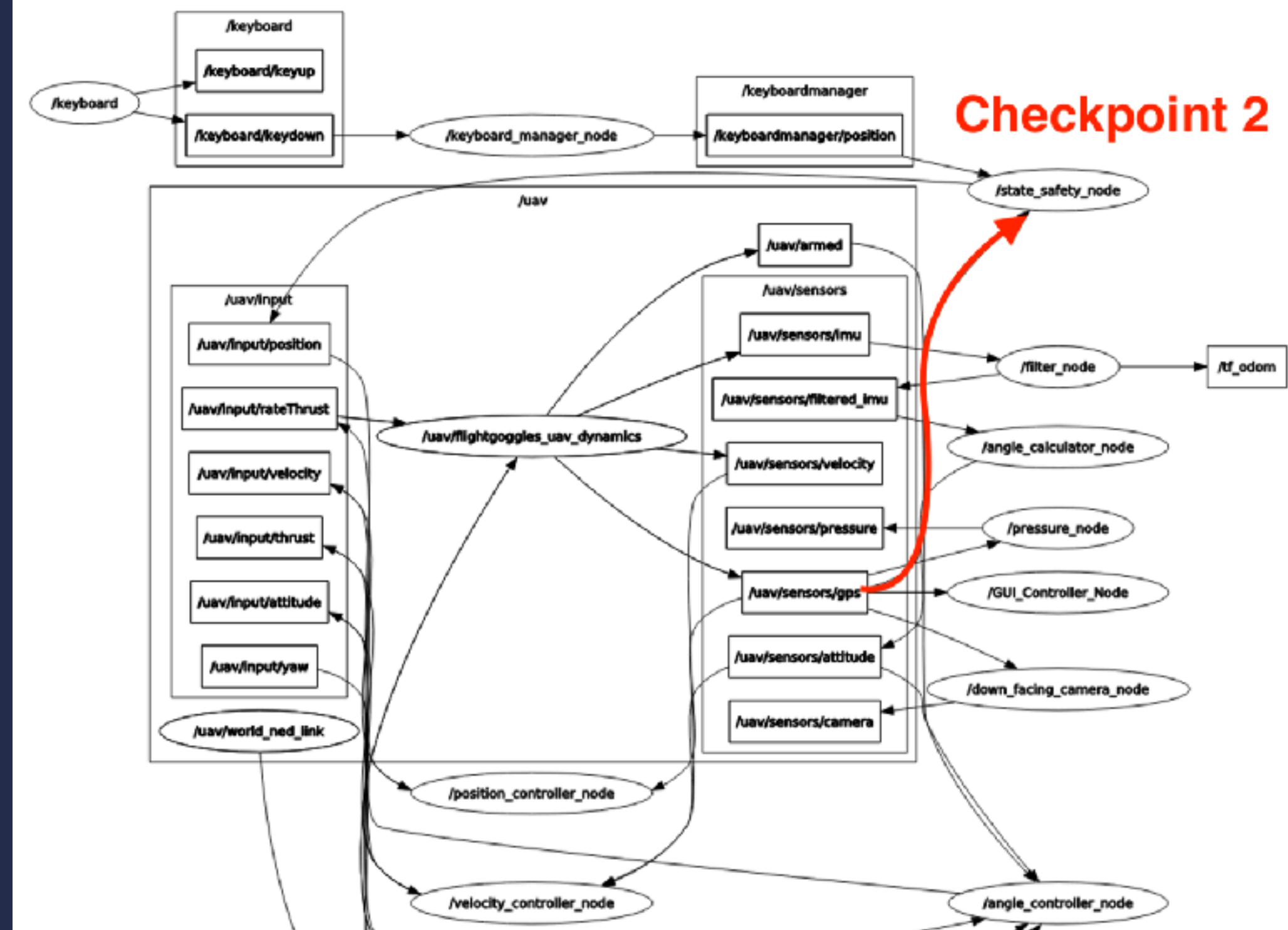
To learn and apply parameter servers, let's start by adding a verification state to forbid the quadrotor from flying outside of a virtual cage. Verifying that a waypoint is

## Verifying that Waypoints are within Cage

Next, let's adapt our FSM to include a verifying state. This verification state will verify the proposed 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  
(13 - Pair SE and Robotics topic)

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

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

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

Make the FSM slightly more complicated, allowing for more complex behavior.  
(17 - 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

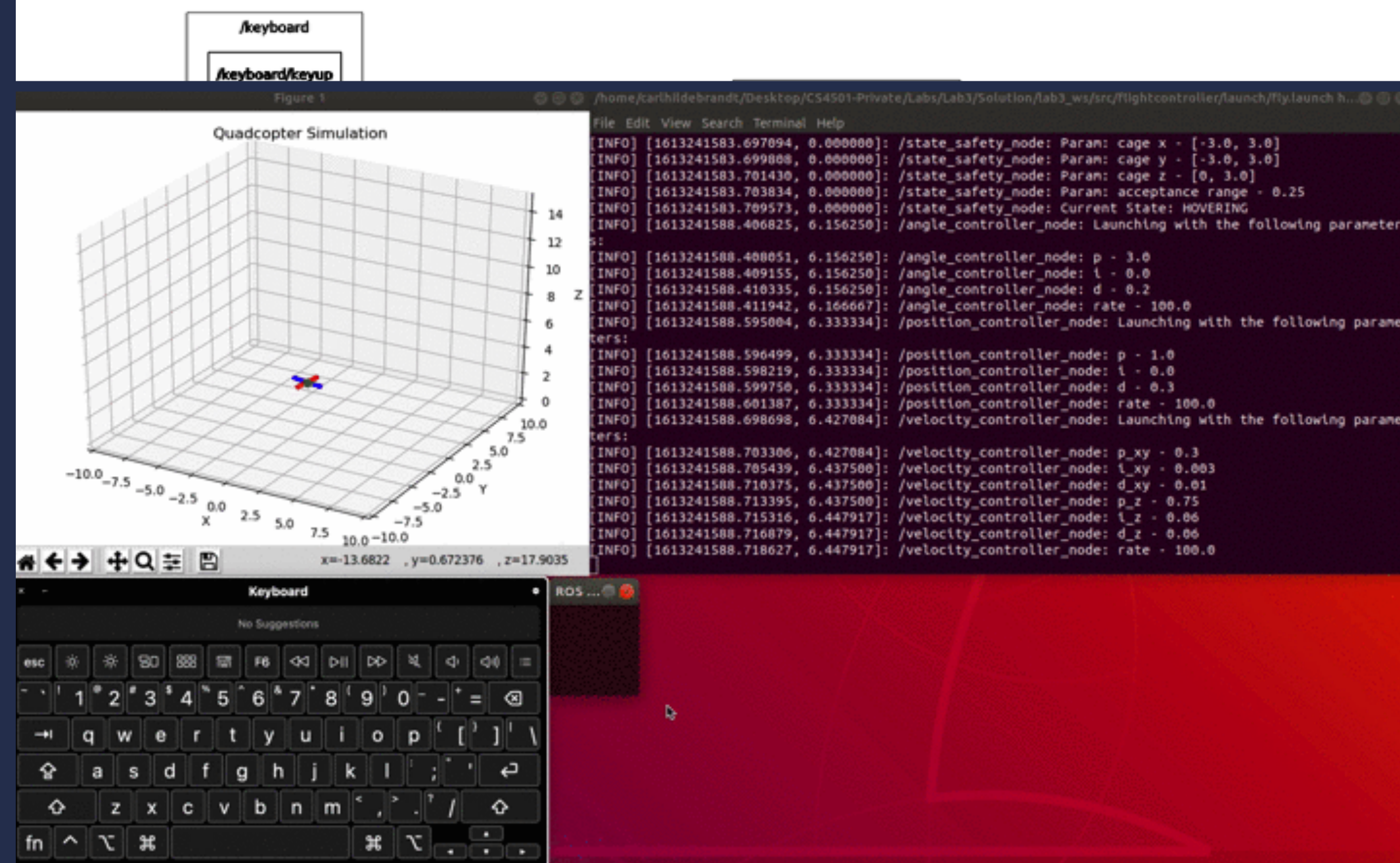
To learn and apply parameter servers, let's start by adding a verification state to forbid the quadrotor from flying outside of a virtual cage. Verifying that a waypoint is

## Verifying that Waypoints are within Cage

Next, let's adapt our CSA to include a verification state. This verification state will verify the proposed 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)

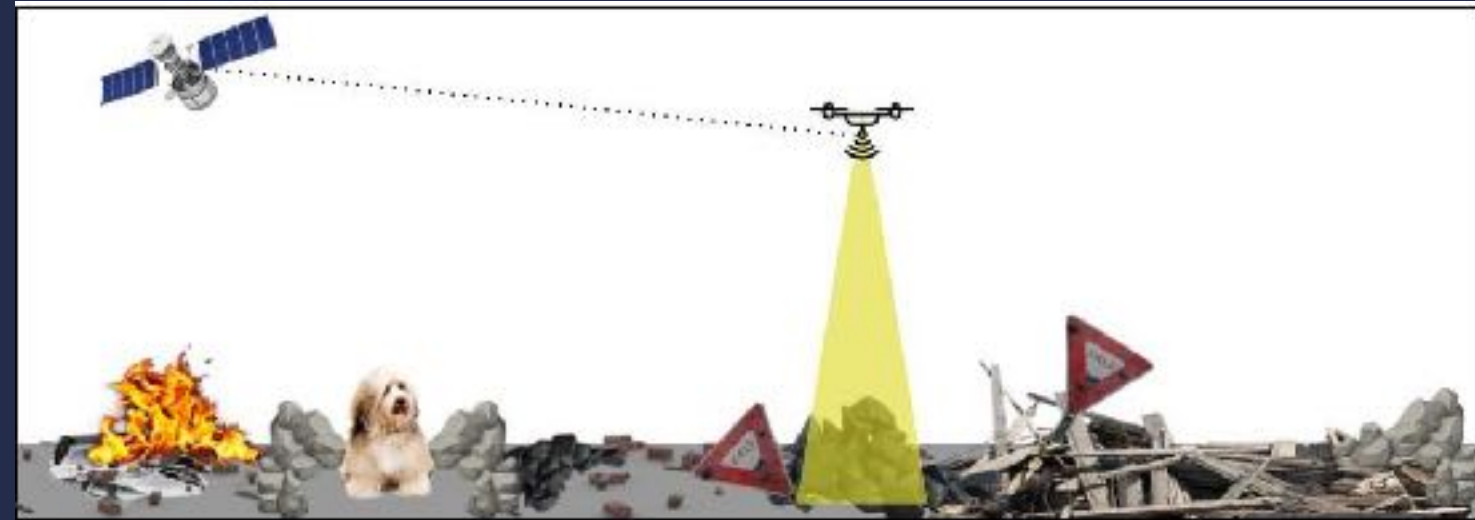
Checkpoint allowing reflection and discussion while allowing freedom to implement their own solution  
(I4 - Students make design and implementation decisions)  
(I5 - Demonstration, conversation, and checkpoints)

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)



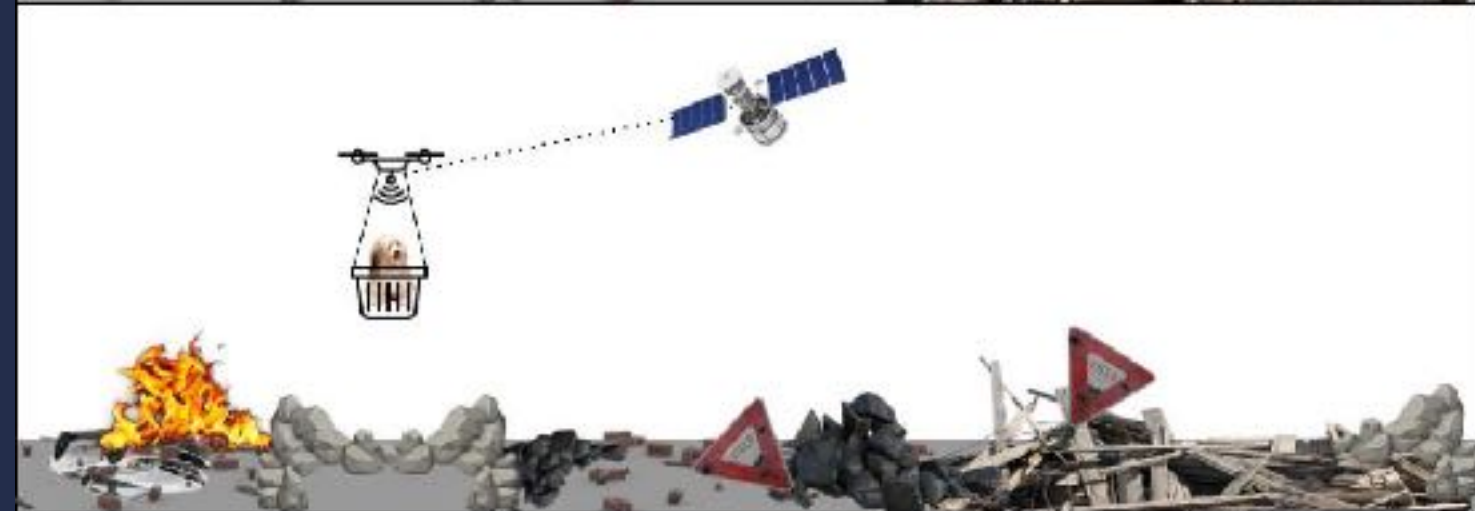
# Project



Locate itself



Locate object, avoiding obstacles



Collect the object



Return to safe area



# Project



```
/home/cs4501/Desktop/CS4501-Labs/tp_ws-trey/src/flightcontroller/launch/fly.launch http...
File Edit View Search Terminal Tabs Help
/home/cs4501/Desktop/CS4501-Labs/tp_ws-trey/src/flightcontroller/launch/fly.launch http...
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: VERIFY
ING
[INFO] [1649780894.395600, 21.968751]: /drone_controller_node: Current Controlle
r 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 Controlle
r 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: HOVERI
NG
down_facing_came...
```

Figure 1

Quadcopter Simulation

A 3D plot showing a quadcopter simulation. The plot is a 3D coordinate system with X, Y, and Z axes. The X and Y axes range from -40 to 40, and the Z axis ranges from 0 to 14. A blue path is shown, starting at the origin and moving in a series of loops and turns. A small green dot represents the current position of the drone. The plot is titled 'Quadcopter Simulation' and 'Figure 1'.



# 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 through 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



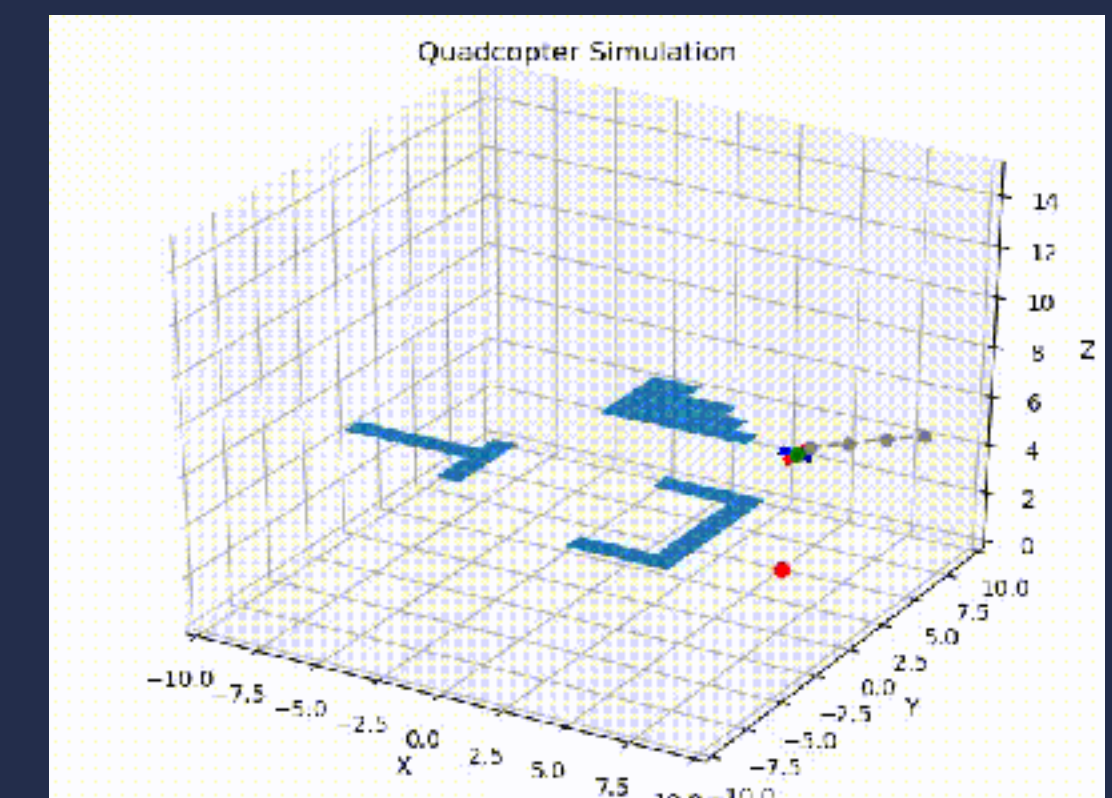
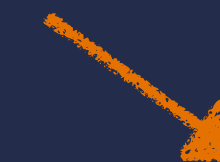
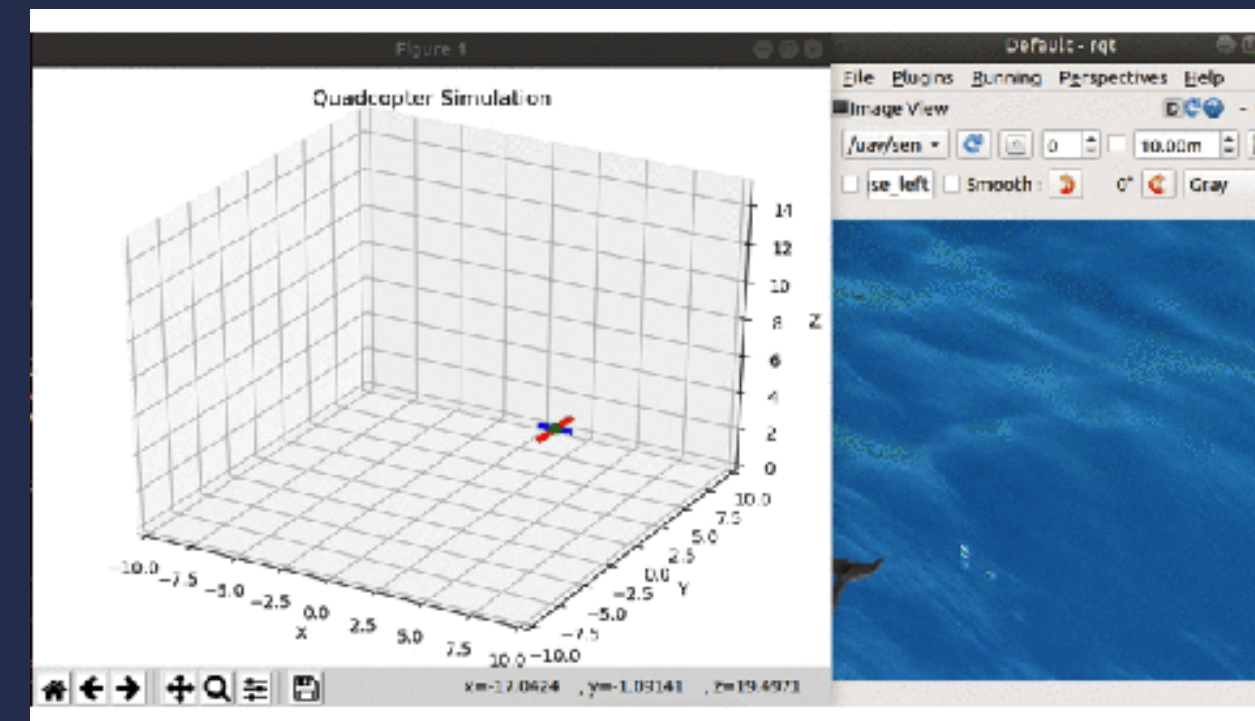
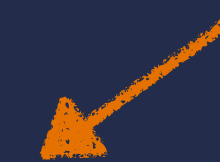
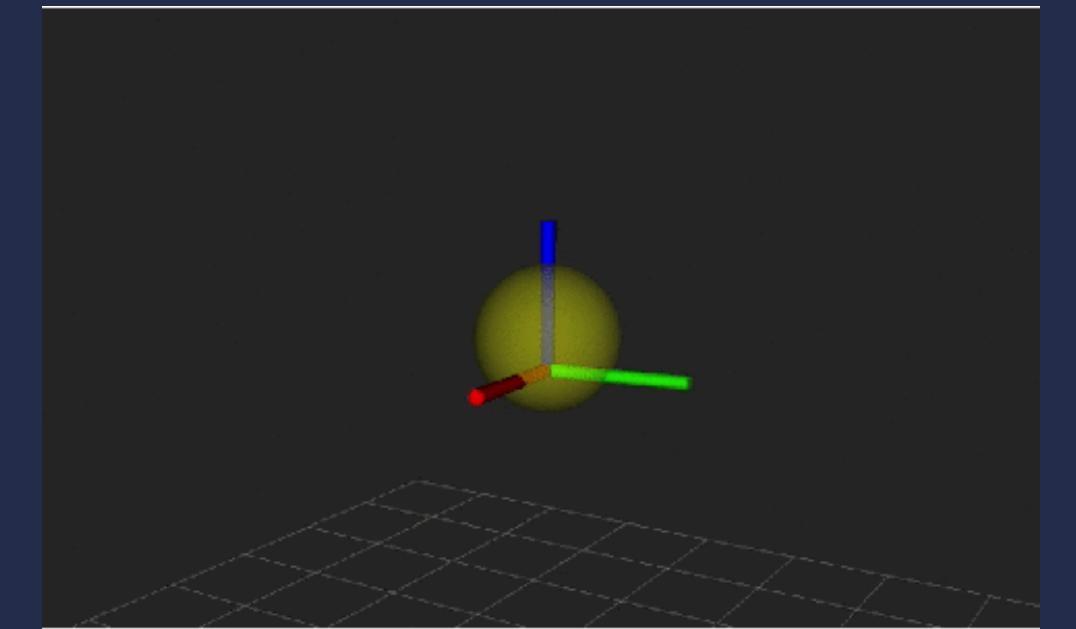
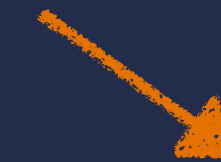
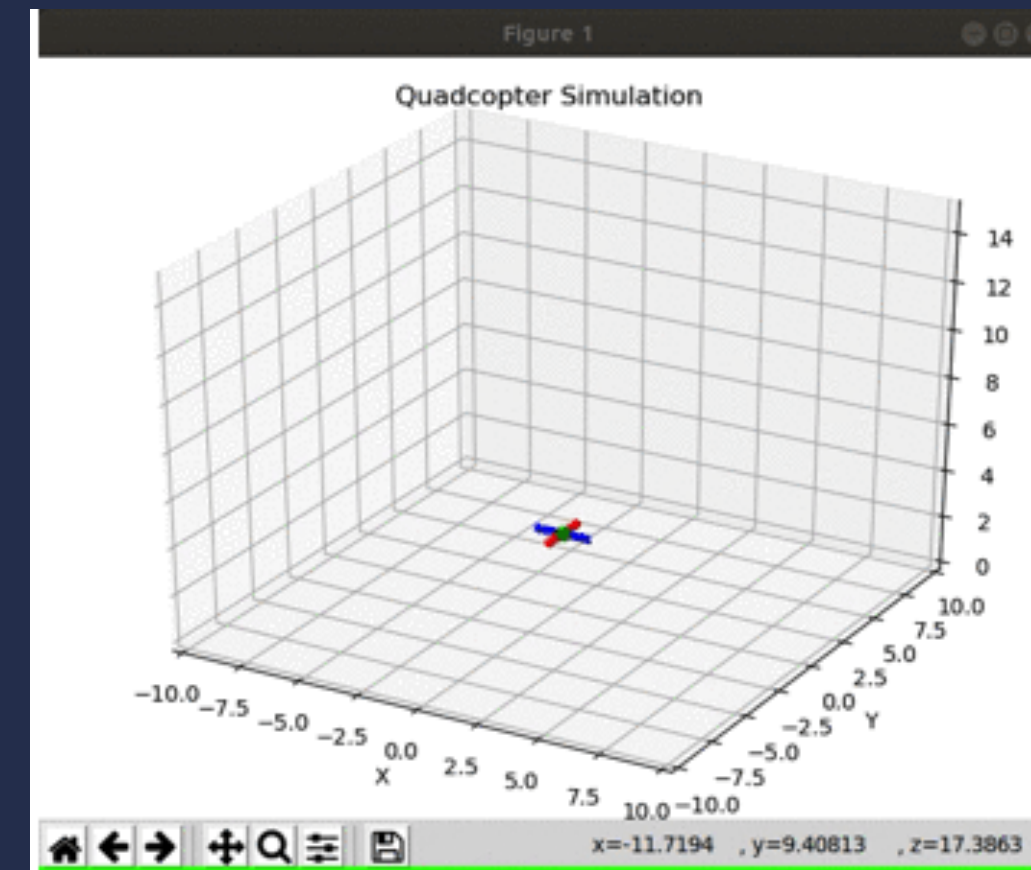
# What Worked Well

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 through 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

<p>Sebastian Elbaum Instructor</p>  <p>selbaum at virginia</p>	<p>Meriel Stein Teaching Assistant</p>  <p>meriel at virginia</p>
<p>Trey Woodlief Teaching Assistant</p>  <p>adw8dm at virginia</p>	<p>Carl Hildebrandt Leads Labs</p>  <p>hildebrandt.carl at virginia</p>



# What Worked Well

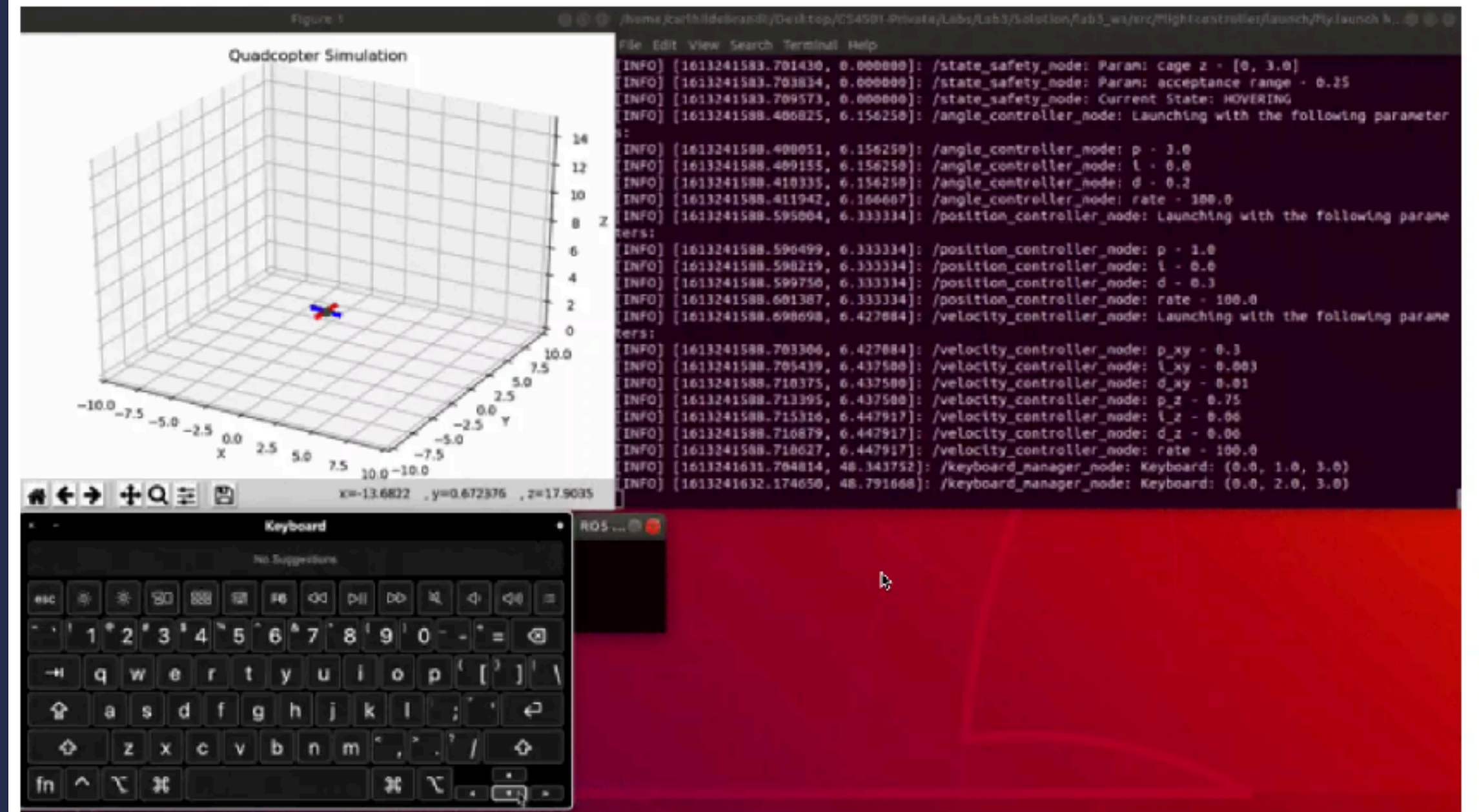
1. Pairing SE and robotics topics
2. Building flexibility into the course
3. Using different levels of abstraction
4. Incremental scaffolding of course material
5. Team structure and process
6. Demonstrating and reflecting during checkpoints

## 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.



The screenshot displays a ROS simulation environment. On the left, a 3D plot titled "Quadcopter Simulation" shows a red quadrotor flying within a wireframe geofence. The plot axes are labeled X, Y, and Z, with values ranging from -10.0 to 10.0. On the right, a terminal window shows ROS logs for various nodes, including /state\_safety\_node, /angle\_controller\_node, /position\_controller\_node, and /velocity\_controller\_node. The logs indicate the drone is in a HOVERING state and provide parameters for the controllers. A keyboard window is visible at the bottom, showing a standard QWERTY keyboard layout.

# 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

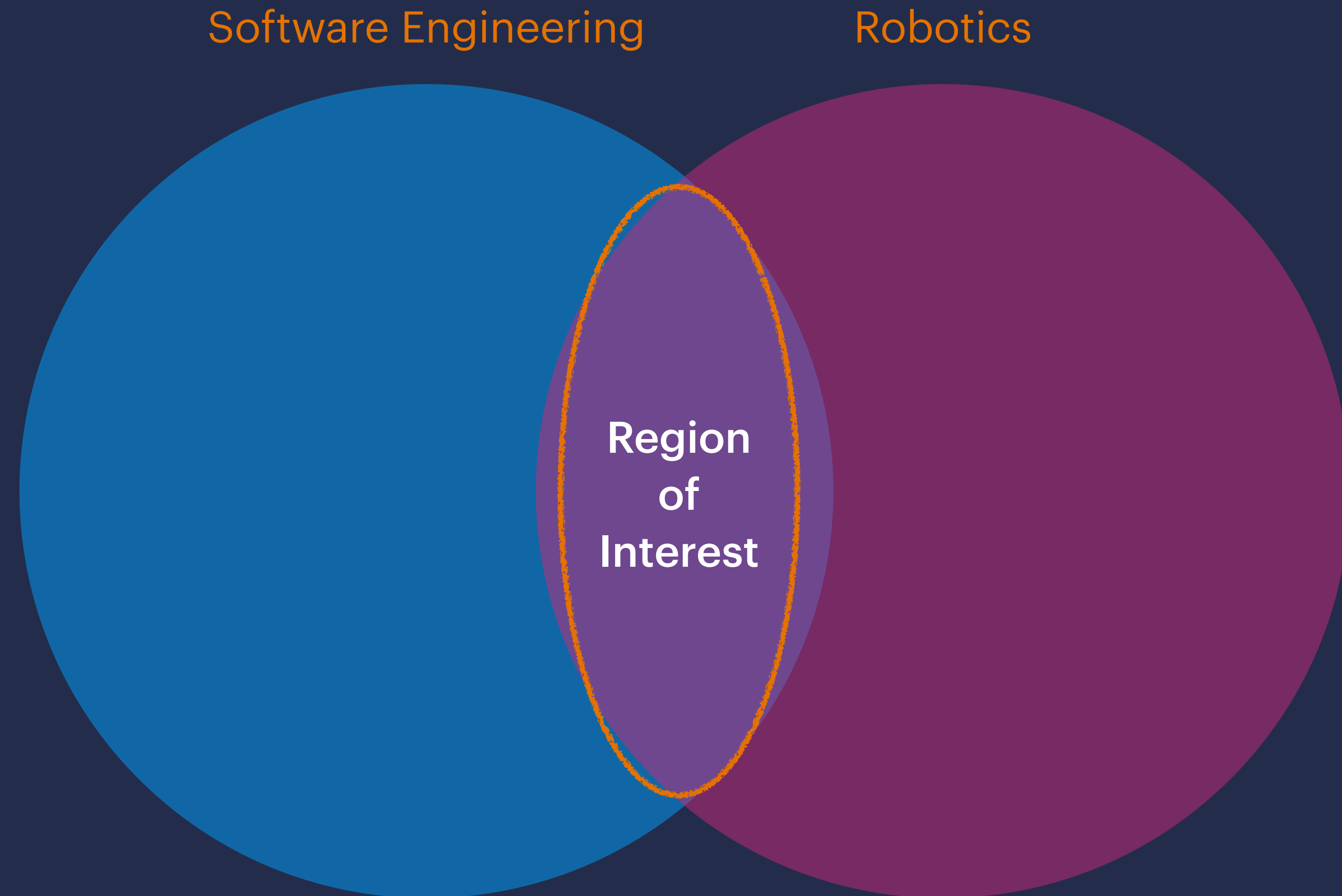




# What Needs Improvement



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



Requires discussion and time

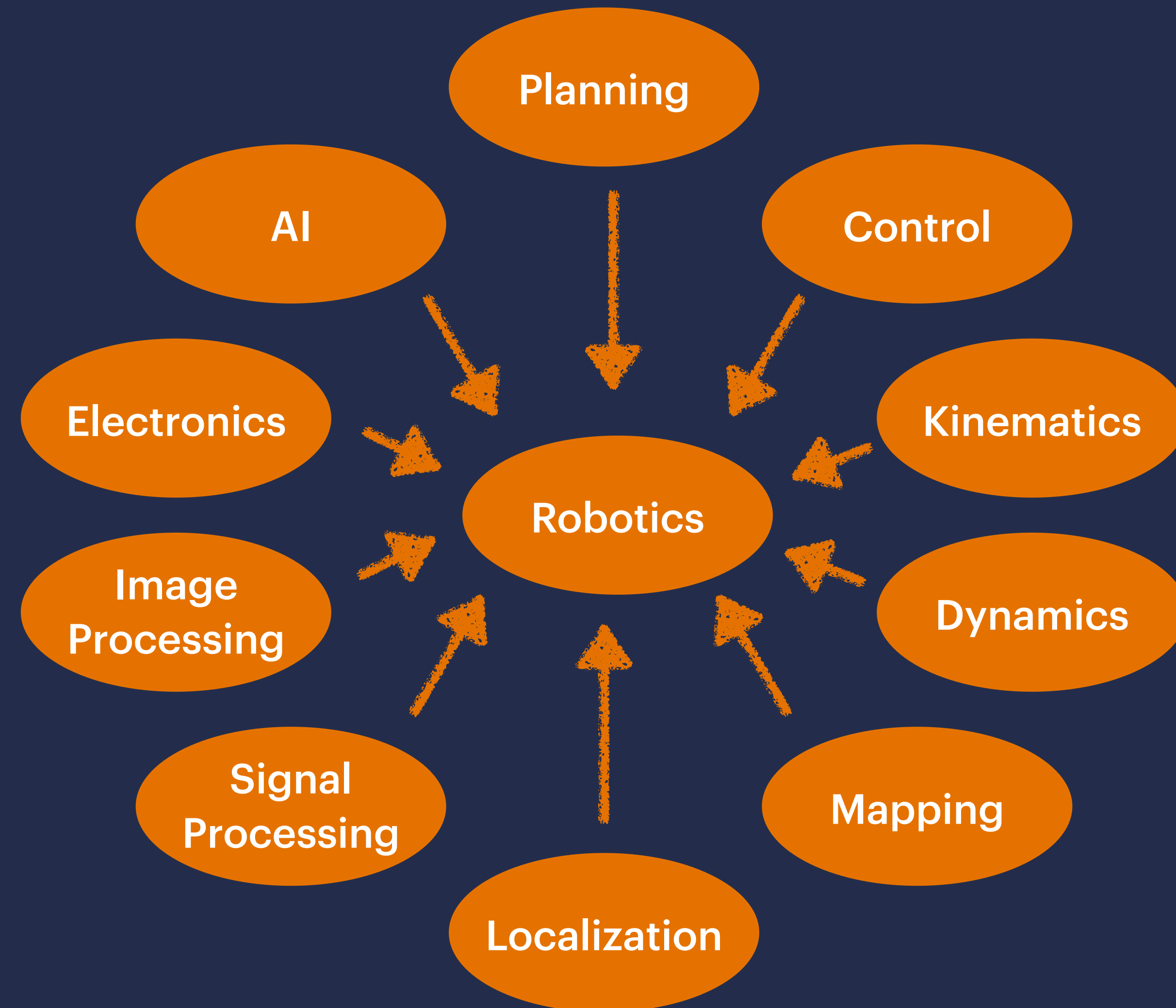




# What Needs Improvement



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

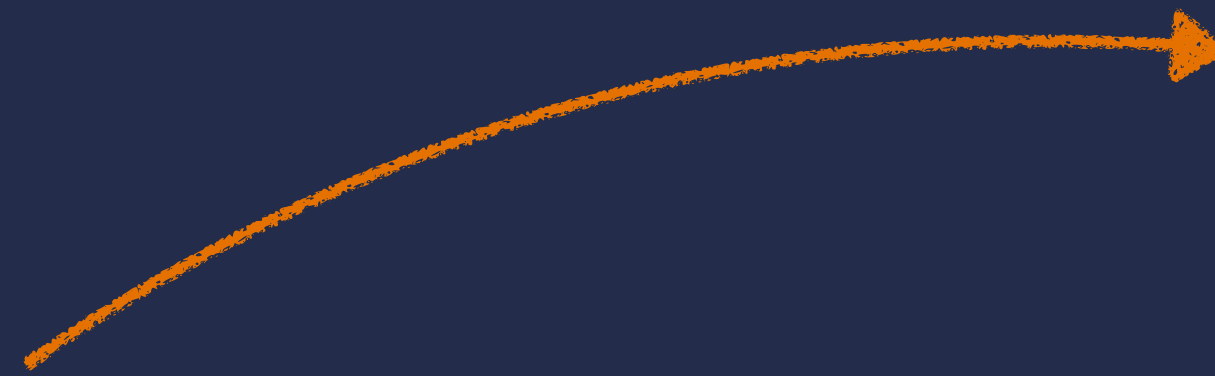


# Impact

4 Ended up in Robotics Industry



Class of 26 Students



This year we have 75 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]"

"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

A screenshot of a web browser displaying the course page for 'Robotics for Software Engineers CS 4501 - Spring 2021'. The page has a teal header with navigation links for 'L3400', 'LECTURES', 'LABS', and 'PROJECT', and a 'learn' button. The main content area includes a green box with a message about bridging the gap between software engineering and robotics, a 'Team' section listing instructors and teaching assistants, a blue box with a message about remote learning challenges, a 'Goal and Scope' section, a 'Class location and time' section, and an 'Office Hours' 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 lab 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, selbaum@virginia
- Trey Woodlief - Teaching Assistant, adw@em at virginia
- Meriel Stein - Teaching Assistant, meriel@virginia
- Carl Hildebrandt - Labs Lab, hildebrandt@car at virginia

This is going to be a great class but the semester is likely to be another weird one with remote learning and "hard struggles 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: Monday 5AM-11AM
- Meriel Stein: Thursdays 5AM-11AM
- Sebastian Elbaum: Friday 9AM-10AM or by email request