The Duckietown Instructor Manual

Contents

Welcome to the Duckietown instructor manual!

In this book we share the resources and experiences that may be useful for constructing and delivering classes using the Duckietown platform.

Who Should Read This Book?

This book is intended to be a resource for people who are looking to **build** a class that uses the Duckietown platform in some way.

In other words, this book is designed for:

- **teachers**, instructors or professors;
- **teaching assistants** (TAs) may also find the content here useful.

We recommend the materials specifically for **university or college** classes either at the **graduate** or **undergraduate** level.

It is not impossible to use Duckietown at high school or lower age/skill levels, but a key consideration in that case would be to think about the prerequisites.

You also need not be developing a class on *robot autonomy* necessarily. The platform can be used as a component of a course centered on **control systems**, **computer vision**, **planning**, **autonomy**, and many other topics.

How to Get Started

Ideally you are reading this book in advance of starting a class, so you have time to plan and prepare.

You might be considering teaching a complete Duckietown class, or integrating the Duckietown resources in a preexisting class.

If this is your case, proceed with the next sections in the order presented. You can use this manual to:

1. Gain insights on the technical resources offered by Duckietown;
2. Decide the scope of your class by reviewing sample curricula and classroom slides;
3. Read the course design considerations and understand what is needed or nice to have in setting up a class.

> **Note**
>
> If the beginning of your class is instead imminent, go directly to:
>
> - Teaching with Duckietown - Quick Start Guide.

How to Use This Book

You should be reading this section if you are considering teaching with Duckietown, and your class starts no sooner than in a few months.

The best time to read this book is:

- after gaining familiarity with the Duckietown platform, but
- before starting your first class.

To gain familiarity with Duckietown consider the following steps:

1. The student experience

We are all learners, and the starting point is getting a taste of the student experience.

You can do so, for free, by evaluating Duckietown in simulation, e.g., by signing up for the "Self-Driving Cars with Duckietown" massive open online course (MOOC).

Glance at the curriculum and go through the first few modules at least. You can consider this step complete once you:

- have tried at least one of the learning experiences (LXs), which are accessible independently of the MOOC in the Duckietown LX GitHub repository;
- made a submission to at least one challenge, and tracked your results on the Duckietown Challenges server;
- joined the Duckietown Slack and private Stack Overflow spaces (invitation required, see instructions on Slack).

2. The hands-on experience

Duckietown shines when you get real robots involved. We strongly contend that one cannot learn robotics properly without a robot. *Simulations are doomed to succeed.*

To proceed:

- get and build a Duckiebot (or Duckiedrone), and a small Duckietown;
  - Get a Duckiebot
  - Build instructions
- become familiar with the Duckietown Shell (`dts`) and the robot Dashboard GUI, paying particular attention to the software architecture and diagnostic tools provided.

You can consider this step complete once you have a Duckiebot working nominally, able to autonomously drive around a city loop by running the lane-following demo.

> **Note**
>
> We offer demo kits discount codes for teachers. Do not hesitate to reach out and ask for one at class-in-a-box@duckietown.com.

Plan your class

Once you have tried Duckietown in simulation, learned about learning experiences and submission evaluations, built, calibrated, operated and drove a Duckiebot autonomously; you should be ready to make more informed decisions in planning for your class.

In the next sections, we provide an overview of the resources available for teaching with Duckietown, and some tips and tricks to running a successful class gathered through combined decades of teaching experience.

Quickstart Guide

You should be reading this section if your class start is imminent, and you have no prior experience teaching with Duckietown. This is a quick step-by-step list action items for creating your class using Duckietown.

Before the course

- Decide on the hardware component of the class so that you can start obtaining it and becoming familiar with it as soon as possible. If you need a quick quote, or even just recommendations, reach out to us.
- Decide on the class composition, what prerequisites you will expect your students to have, the intended learning outcomes that you would like to achieve, and how you would like to structure your class.
- Decide on which material you would like to teach in your class. It could be helpful to look at sample curricula, depending on the level of course you plan to teach (i.e., high school, undergraduate, or graduate). You may also find it useful to look at all of our lecture slides and recordings for inspiration to help you build your curriculum.
- Summarize the above in a class Syllabus, and make a class website page to refer your students to. You can check an example syllabus here.

- Join the Duckietown Slack (and invite your staff to do so as well) to facilitate communication with the Duckietown staff.
- Decide how you are going to evaluate the students. We have a set of curated exercises, but you are also free to build your own. These exercises are automatically graded with an automated evaluation system but also can be run on real robot hardware. You may also want your students to do more involved projects.
- Someone in your course staff will want to develop some level of familiarity with the code.
- Decide whether you are going to teach the class in person or remotely. If you are planning on using hardware in person, start securing an appropriate space to build your Duckietown.
- It is good practice to have teaching assistants support your efforts during a Duckietown course. Depending on if hardware is being used and if student projects are part of your class, we recommend one teaching assistant for every 4-10 students (more TAs if hardware and projects).
- Other teaching staff: make a short list of Ph.D. students, Postdocs and potentially guests that can be invited for lectures (e.g., other Professors, industry practitioners, researchers, etc.). Each of these will be able to improve the overall experience of both the main course instructor and the students, while providing value to them as well. Read more about this is the Duckietown Professor's journey.

Once you have decided on these elements, you are several steps closer to being ready to get started.

The first lesson

At this point you have decided to teach the class, and your syllabus is online.

- Send an opening email to your perspective students roughly one week before the first lesson. Welcome them, thank them for pre-registering, link to the class page and introduce the Duckietown spirit. Tell them more information will be provided on the first day of class.
- If you have decided on a maximum number of participants for the class (generally a good idea, when possible), make sure you allow more than the upper limit on the first day. In our experience, even when pre-filtered, roughly 10% of the students will drop the class by week 2 or 3, once they realize it is not an easy credits course. Not having the planned number of students in the class might cause problems if you are planning on running student projects.
- Meet the perspective students on the first day of class and present your course. Especially if using hardware, make sure you communicate that this course will be challenging and probably more time-consuming than the assigned credits to the course would otherwise suggest. You want to "scare away" at this stage all those students who are not serious about putting in the needed time and effort to actually build competence in robotics.
- We suggest having perspective students (only those who joined the first lesson) fill in an introductory application form. Here is an example comprehensive introduction questionnaire refined over many class iterations. You will use this information to filter the actual students of the class. Provide a tight deadline to fill up this form, e.g., 24 or 48 hours after the first class.
- Evaluate the applications you received, and select the students who are actually accepted in the course.

During the Class

As your class unfolds, you will want to monitor progress. You may want to take a look at the common pitfalls for some ways that others who have used the platform have found that things can go wrong.

During the stress of the semester, it can be easy to lose sight of the class spirit. When things get busy, it may seem overwhelming to add these elements. But this element of the class can be a major factor in the students' overall appreciation of the course you are offering.

Also, should you have questions, there is plenty of support available.

After the Class

We recommend that you solicit feedback from your students, ideally after each learning experience / class module.

Also, *please* let us know how your class went. Your constructive feedback, whether positive or negative, helps us constantly improve the platform and deliver more value for the next generation of instructors and learners.

The Professor's journey

Duckietown was designed with multiple objectives in mind: to help professors deliver great classes, publish compelling papers, and engage their communities.

If you are reading this book, you are likely mostly motivated by the teaching use case, but a Professor often has many diverse objectives (such as teaching, research, and outreach), and aligning those objectives can be beneficial.

In this page, we provide some suggestions about how the Duckietown platform can be integrated more holistically into your lab, hopefully creating a positive feedback loop and bringing compounding value over time in the different aspects of professorial life.

To do so, consider the following "journey" (although there could be many others), that begins with gaining familiarity with the platform for the purposes of teaching but culminates in the use of the platform across a range of use cases including research and outreach.

Semester 0: Evaluate the platform

There are many freely available resources available to get started. It's a great idea for you and/or some people from your group to try out the software, hardware, and pedagogical materials before committing to any next step.

A great place to start could be to sign up for the Self-Driving Cars with Duckietown MOOC on EdX.

> **Tip**
>
> Evaluate Duckietown in simulation and on real hardware before committing.

Semester 1: Start some student projects

A great way to get more familiar with the platform before the stress of teaching a course could be to build a small city, get a few robots, and have students do some projects using it (e.g., for Bachelor's or Master's theses, or supporting Ph.D. students' research).

This serves multiple goals:

1. Mitigate technological risk: small details (space configuration, room lighting, networks, etc.) can cause big headaches. Get these out of the way before prime time.
2. Mitigate pedagogical risk: train a few good potential future teaching assistants (TAs) for your upcoming class.

> **Tip**
>
> Start small, one semester ahead of your first class. Start training your staff as future teaching assistants.

Semester 2: Teaching the first class

Teaching your first robotics class (with Duckietown or otherwise) is fun but it can also be somewhat daunting.

> **Tip**
>
> The best way to mitigate stress is reducing uncertainty.

Use your trained staff as TAs

There is great value is having teaching assistants support you in teaching a class with Duckietown, especially if they have previously gained familiarity with the hardware, software and teaching materials.

If allowed by your institution, don't shy away from the notion of student-TAs. Yes, you can have someone at the same time taking and (help) teaching the class!

Align goals and create win-win scenarios

Students learn best by doing. Organize students in the class in groups and propose projects for them to develop new autonomous behaviors, or expand upon existing ones. Assign each group a *mentor*, such as a Ph.D. student or postdoc, for best results.

Ph.D. students and Postdocs students in your lab will benefit from:

- Gaining mentoring experience;
- Having "free labor" (the students in the groups) to test aspects of their research. Have your (previously trained) Ph.D. students help you define the projects and mentor the student groups.

Bringing a thriving group of experts, all with aligned goals, to the class will create a positive environment for your students, who will:

1. Enjoy a high "instructors/student" ratio for this class, which will result in better course evaluations;
2. Have the opportunity, by excelling in the class, to get much more than just a good grade for the class. E.g., particularly successful projects could result in co-authorship in publications (led by your Ph.D. students and/or Postdocs).

Peer review and community engagement

Although we understand that grades are necessary, there are many ways to assess the learning outcomes of your students.

After all, there are no exams "in the real world", but there definitely is peer review.

If at all possible, we recommend organizing (well in advance) a final demonstration instead of a final exam where students showcase the outcomes of their projects to the general public.

This, in our experience, produces several positive outcomes:

- Students gain ownership of their work, resulting in improved learning outcomes;
- Forcing students to *show* their results working in the real world and explaining successes and failures to a less prepared audience is a great learning experience in itself;
- Robots and duckies attract attention! Notify your department PR team.

Select the next generation of talent

At the end of your class, identifying the best students and offering them to join your lab for the next semesters produces a strong line-up of candidate TAs for the next iteration of the course!

Semester 3: Expansion

As the skills of your team increase, it becomes a good time to expand your setup in size and complexity to unlock more complex scenarios.

There are several dimensions in which this expansion can happen, e.g.:

- Number of vehicles
- Type of vehicles (self-driving cars, quadcopters, smart city infrastructure)
- Size and complexity of the city topography

As the complexity grows, so do the opportunities for teaching more advanced content, and exploring different research directions.

Semester 4 and beyond: Contribute back to the community

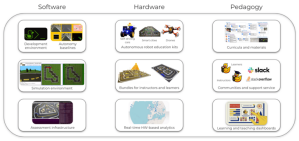
At this stage you will be able to develop new behaviors, create new learning experiences for students, and explore different research directions, while at the same time providing engaging opportunities for outreach.

As you grow into an expert user, we ask you to consider contributing back to the community. Examples of contributions could be:

- Share your learning experiences for other students and instructors worldwide to use.
- Have your students and staff provide constructive feedback to the community on how to improve the platform (many things are open source, so direct contributions are welcome too!).
- Join and become active members of the Duckietown community. Support others on Slack and Stack Overflow, as you and your students have been supported.
- Network with fellow instructors, set up mutually beneficial collaborations.
- Tell us about your publications using Duckietown, let us advertise them to bring you (and us) visibility.

Overview

Duckietown is a *platform*, i.e., a set of tools designed to simplify teaching and learning robot autonomy.



Software, hardware and pedagogical resources are integrated with each other to provide *joyful* learning experiences.

Once familiar with these tools, you will be able to develop new autonomous behaviors for Duckiebots, Duckiedrones and Duckietown Autolabs, and build new learning experiences for your students.

In this section, we list some of the existing components that could be useful to build a class.

Lecture Slides and Recordings

This section contains a compilation of lecture slides and recordings, representative of the content that can be taught with Duckietown.

For access to the editable Keynote or PowerPoint slides, please send us a message on the Duckietown Instructor Slack or at class-in-a-box@duckietown.com, with your institutional email.

Materials are currently available in the following subject areas:

Introductions to Duckietown

This section lists slides and recordings typically used to introduce Duckietown courses and topics.

Getting started

Introduction to Autonomous Vehicles

What are autonomous vehicles, and why are they important?

04:40

Autonomy and Automation

In this short video, we introduce some of the basic concepts and definitions, such as what is a robot, and what is the difference between autonomy and automation.

03:11

Autonomous Vehicles

In these slides, we start to focus more specifically on the problem of autonomous vehicles.

The Levels of Autonomy

The “Levels of Autonomy” define a taxonomy introduced by the US governmental body NHTSA. In this video, we present the details of this taxonomy.

05:36

Visions for Autonomous Vehicles

In connection with the levels introduced previously, different stakeholders view the pathway to achieving full self-driving cars differently. In this video, we present some of those viewpoints and consider their pros and cons.

06:21

The Challenges of Making Autonomous Vehicles

In these slides, we give an overview of some of the core challenges that developers of autonomous vehicles are facing.

Robots

Making Robots

In this video, we describe what are the fundamental building blocks of any robot, from both a hardware and software perspective. We also introduce the “Duckiebot” as an example of a differential drive robot.

05:49

Architectures in Robotics

One of the first and most fundamental choices in the design of the software for a robotic system is the choice of how to structure the architecture. In this series of videos, we present some options.

Sensorimotor Architecture

In some sense, a sensorimotor architecture is the simplest possible robot architecture that one could imagine - one where sensors are directly connected to the motors. To illustrate this idea we introduce “Braitenberg vehicles”, which also has an associated learning experience.

06:06

Stateful Architectures

In a stateful architecture, we now have more complex abstractions, such as perception, planning, and control. These pieces operate together in a feedback loop that can be used to control the robot.

03:56

The following slides cover sensorimotor and stateful architectures.

Logical and Physical Architectures

In this video, we explain the distinction between logical architectures, which specify the functionality of components and their interfaces, and physical architectures, which explain the details of how the components are implemented.

03:54

Testing

Testing Autonomous Vehicles

We discuss the different types of testing that are needed for developing autonomous cars.

Testing in Duckietown

We give some more concrete examples of how automated testing is performed in the Duckietown codebase.

Signal Processing

Modern Signal Processing

We discuss some basic concepts related to signal processing, such as latency, throughput, and frequency, as well as event-driven (asynchronous) vs. periodic (synchronous) approaches.

Networking

Basics of Networking for Robotics

Networking can be one of the most challenging aspects of working with robots. Having some understanding of basic networking concepts can save a lot of time and frustration.

Full Introductory Lectures

Introduction to Autonomous Vehicles

In this lecture, we present a broad view of the field of autonomous vehicles. This includes some technical content such as some important nomenclature that will be used later in the course, the broad building blocks and abstractions of an autonomous system. It also includes some non-technical content such as societal considerations related to self-driving cars, including ethical considerations, as well as the various business models for making self-driving cars a viable product.

We have found that it can be a good idea to devote some time to these larger questions before delving into the technical details for the remainder of the course.

Software Engineering

In this lecture, we combine the previous sections on networking, testing, and signal processing into one software engineering-themed lecture.

Why Duckietown?

We have this fun lecture about the origins of the Duckietown project that can be a nice way to start the semester and get students excited.

Tools

This section lists slides and recordings used to present some of the software tools that are helpful for students to understand.

Containerization

Containerization is a method of isolating software together with its dependencies. It is useful for reproducibility, transportability and standardization of interfaces. The containerization software used in Duckietown is called Docker.

Middlewares and the Robot Operating System (ROS)

ROS is a popular robotics “middleware”. It provides mechanisms for communication between components of a robot system, as well as debugging and many other features. ROS is sometimes affectionately referred to as the “McDonalds” of robotics. Duckietown robots, including Duckiebots, Duckiedrones and smart Duckietown infrastructure use ROS (and from mid 2024 will start supporting other middlewares, such as ROS2).

Git

Version Control with Git

Git is a tool for building collaborative software projects. Here we present some of the overarching principles and use-cases for Git, such as version control, repositories, branches, forks, push and pull, etc.

Videos

03:51

Slides

Modeling

This section lists slides and recordings typically used in introducing mathematical models that are relevant in Duckietown. We will see how these models are particularly useful for synthesizing controllers as well as building state estimators.

Robot Representations

Robots leverage *mathematical models* to create an “idea”, or representation, of the world around them, and place themselves within it.

These models are leveraged to quantify key features of the environment such as the presence of obstacles or viable paths, so to plan and execute actions aimed at achieving specific tasks.

Here, we introduce the most common tools and notions used to build robot representations.

06:02

The slides introduce groups as well as choices for world and robot representations.

For the accompanying exercise, please see Modeling and Control.

Modeling a Differential Drive Robot

A differential drive robot is a particular mobile robot hardware configuration that relies on two independently driven wheels to move in the environment.

Differential drive is a simple configuration, yet not trivial. In this section we derive a kinematic model for a differential drive robot, mapping the input voltages to the motors to the output pose evolution over time, under simplifying assumptions.

For the accompanying exercise, please see Modeling and Control.

Full Lecture on Modeling

In these slides, we develop a model of the Duckiebot that estimates how the voltages applied to the motors will cause the Duckiebot to move through space. This involves a model of the DC motors (very approximate) that converts voltages to torques, followed by a model of the dynamics of the robot that converts torques to accelerations and velocities, and finally a model of the robot kinematics that converts accelerations and velocities to position and orientation.

For the accompanying exercise, please see Modeling and Control.

Control Systems

This section lists slides and recordings typically used in introducing control-systems-related topics in Duckietown.

We leverage models to describe the physical systems and the reference trajectory that is computed by a motion planner, to synthesize a control policy.

The policies translate in actuation commands that make the robot move.

Introduction to Control Systems

Control systems is the science of making machines do what we, humans, want them to do, rather than what they would naturally be inclined to do.

In a more narrow definition, it is the logical component in the mind of the robot that transforms estimates of tracking errors into corrective actions, so to achieve the task set forth by the planner.

For the accompanying exercise, please see Modeling and Control.

Odometry

The etymology of the word odometry is “measurement of the path”, and it describes a way to use sensors, e.g., wheel encoders, data to create a dead-reckoning model to estimate the pose of the Duckiebot over time.

In other words, odometry describes one of the simplest approaches for a mobile robot to estimate the evolution of its position and orientation (pose) over time.

For the accompanying exercise, please see Modeling and Control.

PID Control

“Proportional, Integrative, Derivative” (PID) control is a feedback control architecture that leverages the output tracking error to drive the robot to a steady-state reference trajectory.

It is arguably the most successful controller of all times, definitely the most famous.

For the accompanying exercise, please see Modeling and Control.

Full Lecture on Control

The following slides cover an in-depth introduction to control systems, including fundamental concepts such as stability and robustness. We introduce some popular control schemes in detail, such as PID, pure pursuit, and model predictive control.

For the accompanying exercise, please see Modeling and Control.

Additional legacy learning activities are available on PI and LQR-I controller design and discrete-time implementation.

Computer Vision

This section lists slides and recordings used to present topics related to computer vision. Specifically, we cover the process of image creation in a camera, through the pinhole camera model and projective geometry, the calibration of a camera, and the process of image filtering. These materials could be considered a prerequisite for understanding Visual Perception.

Projective Geometry

We introduce the mathematical tools needed to understand how to model an image sensor. This includes a model of a camera, referred to as the “pinhole camera”, as well as the basics of projective geometry including homographies, perspective projection, and homogeneous coordinates.

Camera Calibration

Not every camera is the same, as not every Duckiebot is the same.

As a result, we need to execute a process of calibration to identify the parameters of a particular setup. These parameters fall into two categories: the *intrinsic* camera parameters pertain to the camera itself and how an image is formed, and the *extrinsic* camera parameters pertain to where the camera is mounted relative to the robot.

We describe methods for estimating these two classes of parameters.

Image Filtering

We can modify an image by designing a filter and convolving it with the original image. The way we design this filter will depend on how we would like to process the image. In this video, we describe this filtering process and describe a few import types of filters, such as ones that would be useful to help us detect lines or corners in an image.

Image Gradients

These slides cover the notion of image gradients:

Full Lectures on Computer Vision

Projective Geometry

This lectures introduces Projective Geometry, Camera Calibration, as well as more detail on homographies and perspective projections.

Image Processing

This lecture discusses Image Filtering in more detail, including edge detectors (with a particular focus on the Canny-edge detector), image features, corner detectors and scale-invariant descriptors (such as SIFT).

This material is supported by the learning experience related to “Visual Lane Servoing”.

Visual Perception

This section lists slides and recordings used to present topics related to visual perception.

This includes a definition of visual perception tasks, an introduction to neural and deep convolutional neural networks, and a more thorough investigation of the problems of object detection and place recognition (which is related to SLAM).

A prerequisite for understanding this material could be the basics of Computer Vision. Much of this material is supported by the learning experience related to “Object Detection”.

Introduction to Visual Perception

We start our treatment of visual perception by defining the specific visual perception tasks and discussing how each of them can be evaluated. Namely, we introduce the problems of image classification, image segmentation, instance segmentation, and object detection. We define precision and recall and introduce the precision-recall (PR) curve.

We also define intersection-over-union (IoU) and mean average precision (mAP), which are metrics used to evaluate object detection algorithms.

Introduction to Neural Networks

Neural networks have become a nearly ubiquitous tool in visual perception systems for robots.

This is due largely to two key properties that they hold: compositionality and differentiability.

Together, these allow us to compose the atomic building blocks (neurons) and learn parameters by propagating gradients.

We briefly cover the simplest neural network architecture, the multi-layer perceptron, and discuss how it can learn through a process called “stochastic gradient descent” (SGD).

Deep Convolutional Neural Networks

Since images are high-dimensional (there are lots of pixels!), it is impractical to have a fully connected neural network. Instead, we will learn image filters and apply them through convolution. If we chain several of these blocks together the result is a deep convolutional neural network, which we introduce here. We also discuss some details needed in the construction of these networks, such as padding, stride, pooling, and finally we introduce the residual which can more efficiently pass gradients enabling such models to be much deeper (more layers).

Object Detection

One of the most important visual perception tasks for autonomous driving is the detection of objects. Two classes of CNN-based approaches have emerged for tackling this problem: one-stage and two-stage. The two-stage approaches included an explicit step for generating region proposals, or areas of

the image that are likely to contain objects. In this video, we detail how these methods work and are evaluated. The object detection experience further explores how an object detection model can be built and integrated into an autonomous system.

Full Lectures on Visual Perception

Introduction to Machine Learning

These slides provide a basic introduction to machine learning, deep learning, neural networks and convolutional neural networks.

Visual Perception

These slides combine the sections: Introduction to Visual Perception, Introduction to Neural Networks, Deep Convolutional Neural Networks, and Object Detection, into one full-length lecture.

These are a slightly older version with mostly similar material but slightly different emphasis.

State Estimation

This section lists slides and recordings used to present topics related to state estimation.

These are methods for the robot to quantify its position and orientation (pose) in the world, and possibly also the state of the world around it in the case of simultaneous localization and mapping (SLAM).

In other words, estimation algorithms provide solutions for transforming sensor *data* into actionable *information*, useful to the robot to achieve its task.

We present materials related to how to represent the robot, as well as several different filters, which generate a state estimate based on the data that the robot is collecting (as well as some pre-built models and assumptions).

Finally, we present some methods related to robust optimization and SLAM. These materials are supported by the learning experience related to state estimation.

Probabilistic Representations

Before discussing the methods for estimating the state, we decide how to represent the robot state. This lecture expands on a lecture related to representations for building models to discuss different choices about how we might represent the state of the robot and the world around it in some mathematical way. This includes some basics about probability theory and an introduction to group theory and choices for robot and world representations.

Bayes Filter

The Bayes Filter, built with Bayes' theorem, represents the optimal estimate of the posterior distribution of the state.

Unfortunately, in practice, it is almost always impossible to compute.

Notwithstanding, this filter forms the basis for introducing several other filter families which make different assumptions to make the computation of the posterior more efficient.

Kalman Filter

The Kalman filter approximates the Bayes filter by assuming that the process and measurement models are linear and corrupted by additive Gaussian noise. Under these assumptions, the Kalman filter is optimal in the sense of minimizing the covariance of the estimation error.

In the case that the linearity assumption is violated, we can linearize the equations at each time step, which results in the extended Kalman filter (EKF).

Even though these assumptions seem severe, this filter is probably the most widely used in practice.

Note that there is also an activity that explains this approach in more detail in the state estimation exercise.

Particle Filter

The particle filter exists in many variants, but the fundamental idea is to approximate the Bayes Filter solution by evaluating it on a discrete set of weighted samples.

The particle filter approach is inexact in the sense of being unable to represent arbitrary distributions but makes the Bayes' equations computationally tractable.

Note there is also a notebook activity that explains this approach in more detail in the state estimation exercise.

Histogram Filter

The histogram filter approximates the Bayes Filter by discretizing the state space, making the solution not only tractable, but computationally efficient.

This is the approach taken in the state-estimation exercise.

Full Lecture on Filtering

These slides present the previous four filtering approaches.

Robust Estimation

Robust Fitting

We present a brief overview of the problem of finding a transformation, specifically a homography between two images, based on a set of correspondences, some of which may be incorrect. This is the famous Random Sample Consensus, or RANSAC, algorithm.

Full Lecture on Simultaneous Localization and Mapping

Introduction to SLAM

Simultaneous localization and mapping (SLAM) is the problem where a robot must use the sensor data that it is collecting to build a representation of the environment while concurrently placing itself within that representation.

In this lecture, we present first the mapping problem, assuming the robot knows where it is (i.e., is *localized*), then the localization problem assuming that the map is provided, and then finally the full joint SLAM problem.

We discuss three approaches to the SLAM problem, the first based on the Kalman filter, the second based on the particle filter, and the third a maximum a-posteriori optimization-based approach.

Planning

This section lists slides and recordings used to present topics related to robot planning.

Typically, this problem is formulated as operating at a higher level of abstraction than the problem of robot control. Nevertheless, a formal framework for viewing this problem is through the lens of optimal control.

This formulation is theoretically interesting, but often impractical unless significant approximations are made. An alternative is to formulate the planning problem as some kind of search over graphs. There are several different approaches to do this, but then we can leverage well-known methods of graph search to find the optimal plan (or a good approximation to it).

For the accompanying exercise, see Planning.

Motion Planning

We formulate the problem of motion planning and introduce several important definitions and concepts that will be used to analyze solutions to the motion planning problem.

Part 1: Motion Planning as Optimal Control

We formalize the motion planning problem as an optimal control problem.

Part 2: Geometric Primitives

We introduce some needed basic primitives such as paths and trajectories and present some basic classes of strategies commonly used to address the problem of motion planning.

Part 3: The Geometric Perspective

We further introduce the concepts of workspaces and configuration spaces and show how they can be used to find solutions to the motion planning problem.

Planning on Graphs

In many cases, we can reduce the motion planning problem to that of searching over a graph. Here we present some formalizations about the types of graphs we are going to use and what methods we can use to search over them.

Undirected and Directed Graphs

We start by introducing some basic graph theory concepts, such as vertices and edges and the distinction between directed and undirected edges.

Types of Graphs

We continue the discussion about the types and properties of graphs that we will use for motion planning.

Planning Using Graph Search

We formalize how the motion planning problem can be represented as a search over a graph, and we provide two popular approaches, Djiksra's algorithm and the A* algorithm, for how we can efficiently search the graph.

These slides summarize some of the concepts presented in the videos above:

Sampling-based Planning

In the previous section, we introduced how graphs can be searched. In this section, we discuss practical, and very commonly used strategies for building these graphs.

ince, in general, the configuration space is large and potentially high dimensional, we don't want to search it exhaustively. Instead, it turns out that we can use strategies to sample it and then construct a graph whose edges constitute collision-free paths.

Full Lecture on Planning

These slides present a full lecture on the subject of robot planning, including components of Motion Planning and Planning on Graphs as well as much more content related to classic path planning methods such as motion primitives, variational methods, potential fields, cell decomposition, visibility graphs, and sampling-based methods such as rapidly exploring random trees (RRT and RRT*) as well as an introduction to finite state automata.

End-to-End Learning Approaches

In this section, we depart from the traditional abstractions of perception, estimation, planning, and control and instead try to directly learn how to control an autonomous vehicle directly through data. We will primarily focus on two paradigms (although there are others): reinforcement learning (RL) and imitation learning (IL).

Markov Decision Processes

The mathematical formalism that we will use to develop in particular reinforcement learning algorithms is the Markov Decision Process, or MDP.

The MDP comprises a state representation (that satisfies the Markov property), an action space, a transition function, a reward function, and a discount factor. In the case that we are estimating the state through sensor data, this becomes a partially observable MDP, or POMDP. In this context, our objective is to find a good policy.

Policy Iteration

A general framework for determining a good policy for an MDP is to start by finding the *value function*, or the value associated with each state, or state-action pair, for that policy.

This indicates our estimate of the discounted return that we would obtain if we started in a given state and then followed the policy forever after. Policy iteration involves iterating between improving the policy and estimating the value function.

Q Learning

Through the Bellman equation, we can formulate a bootstrapping objective for estimating the value function (the Q-value function more specifically). This objective involves minimizing the temporal difference error. We discuss strategies for doing this, distinguishing between on-policy and off-policy approaches. Finally, if we have a high dimensional state, such as an image input, we show how we can use a neural network to estimate the state representation, a method called Deep Q Learning, or DQN.

Simulation and Sim-to-real

There are several reasons why training RL agents on real robot hardware is challenging. Training in a simulator and then applying the resulting model in the real world can be a good strategy to overcome some of these challenges. This transfer process is referred to as sim-to-real transfer.

In general, there are two classes of approaches. The first is to explicitly model the discrepancies between the simulated and real worlds. The second is to use the simulator and RL training scheme to learn a policy that is inherently robust to these differences.

Full Lecture on Reinforcement Learning

This lecture contains presents the material on Markov Decision Processes, Policy Iteration, Q Learning, and Simulation and Sim-to-real, as well as an additional section on policy gradients, an approach to learning an RL policy that does not explicitly estimate a value function.

This is another version of the slides on RL with more of a focus on model-based RL.

Imitation Learning

Learning a policy with RL can be very inefficient and require a lot of trials. An alternative is to learn directly from expert data.

This approach is referred to as “imitation learning” (IL).

Advanced Topics

In this section, we collect slides that cover more advanced topics (i.e., topics that are still very much the subject of active research).

These resources would likely be too advanced to include in an undergraduate course.

Multi-vehicle Planning

In this section, we consider the case where we are planning or coordinating more than one autonomous vehicle at a time, perhaps even in an entire fleet.

Multi-vehicle coordination

In this lecture, we formalize the multi-robot planning problem. We introduce some basic concepts in optimization and present some potential solutions for planning and coordinating with multiple vehicles.

Fleet-level Planning

In this lecture, we consider the optimization of an entire fleet of autonomous vehicles.

Safety

Safety is a crucial consideration for the integration of self-driving cars at scale. Ideally, we would be able to provide some kind of formal guarantee about the performance so that legislators and other stakeholders can *trust* the technology and move to regulate it.

Part 1 - Introduction to Safety

We start by introducing some basic concepts related to formal verification and safety.

Part 2 - Formal Methods for Safety

We provide a more rigorous treatment of using formal methods for safety.

Estimating Uncertainty

With the inclusion of more deep-learning-based models inside autonomous vehicle systems, the issue of safety becomes even more central, but also challenging.

One hope is to build deep learning models that are also able to output *calibrated* estimates of their confidence. This requires understanding the different types and sources of uncertainty in the output of deep-learning model.

Estimation from Motion Blur

Cameras are integrative sensors, as they sum up incoming light during the duration of the exposure. One central challenge in state estimation when using a camera is that the relative motion of the camera and the scene will induce motion blur in the resulting image. While motion blur is typically considered undesirable because it makes features of interest more difficult to detect, it is possible to use this blur as an asset rather than a nuisance.

If you would like to recommend a specific topic for new materials, do not hesitate to let us know by reaching out, or start building a Duckietown learning experience yourself.

Activities and Exercises: Learning Experiences (LXs)

> **Note**
>
> Duckietown learning experiences (LXs) a ready-to-go "weeks" of class. They include videos, notes, interactive activities, and exercises, and are integrated with the Duckietown technical infrastructure (simulator, hardware, evaluation infrastructure).

We define:

- "activities" as learning tasks to which solutions are provided. Activities are designed to be "tutorials" for specific topics.
- "exercises" as learning tasks to which solutions are not provided. Solutions to exercises are typically matched to "challenges", which provide evaluations of performance across engineering metrics. These evaluations can be leveraged to, e.g., automatically grade student assignments.
- Duckietown "learning activities" (LX): as standalone classes on specific topics, typically containing activities, exercises, videos, slides, quizzes and pointers to further reading. LXs can be thought of as a week of (university-level) classes.

Both activities and exercises are structured to include Jupyter Notebooks that introduce a concept followed by coding blocks. For the most part, the result is a piece of code that can be easily:

- Run in a simulation environment;
- Run on robot hardware;
- Submit for automatic evaluation.

The pedagogical goal in general is to explore some narrowly scoped component of the autonomy stack with everything else being "hidden" (or provided) so that the student may experience the impact of that component on the others. Whenever possible, the result of the exercise should be an "end-to-end" experience that makes the robot do something (e.g., move).

For a complete description of the technical components of the learning experiences (LXs) please refer to the Learning Experiences Manual. This includes information about the workflow for completing an exercise and the procedure for creating your own learning experience from scratch.

For help on how to create learning experiences, open a question on Stack Overflow with the tag `LX` (preferred), or ask on the #help-build-lxs channel on Slack.

MOOC Activities and Exercises

The Self-driving Cars with Duckietown Massive Online Open Class comprises the following learning experiences. Activities and exercises can be accessed independently of the MOOC at:

EXERCISE NAME	DESCRIPTION
Braitenberg	A very simple reactive control approach that is inspired by the repulsive and attractive forces
Modeling and Control	We build a kinematic model of the Duckiebot and build a simple PID controller using the feedback from the encoders
Object Detection	We train a deep neural network to detect objects and connect it to the control of the Duckiebot
Visual Lane Servoing	We use basic concepts of computer vision to build a reactive control that operates directly on the camera images
State Estimation	A somewhat more advanced exercise that takes the detections of the road markings and uses them to calculate an estimate of the robot's state
Collision Checker	We build an algorithm to detect if the robot will collide with it's environment by understanding its state and geometry
Planning	We explore algorithms that the Duckiebot can use to successfully navigate in a cluttered environment

Evaluation

An important consideration in the construction of a class will be how the students are going to be evaluated. We typically **do not** test students' knowledge through traditional exams and tests (although this would certainly be possible).

Instead, we recommend some combination of exercises and a course project. We also typically put an above-normal emphasis on student participation or being "Being a Good Citizen" in Duckietown.

Evaluating Learning Experiences

An automated evaluation system is available to ease the process of grading the learning experiences in simulation.

Fig. 1 The Duckietown "Challenges Server" is used for automated evaluation of exercise submissions

We have a collection of existing learning experiences that you can use in this way, or you can design your own. The design of the learning experiences includes the specification of metrics that will be used to evaluate the performance of the submission to the automated evaluation system.

We recommend that you decide on some sufficient performance and grade the submissions as either pass or fail, based on whether they have met these criteria. The reason for this is that, with careful tuning, the metrics can be optimized and overfit but this often comes with diminishing returns in terms of pedagogical value.

It is important for students to get a flavor of how the various parameters affect the result, but when they are spending many hours tuning things, they tend to become irritated, and rightfully so.

Also, note that we have plans to extend this capability to real robot evaluations in remote labs that we call Autolabs. Autolabs have been used, e.g., for the AI Driving Olympics, and are available in beta upon request.

Evaluating Projects

We usually advocate for students to submit written reports and progress presentations at regular intervals. Please see the course projects page for resources such as templates for these reports.

We often solicit the students themselves to provide feedback on each others documents and presentations, to expose learners to the notion of peer review, which we believe to be more formative than traditional top-down grading.

In our experience, it is important not to put too much weight on the final performance of the demo since this causes stress.

Evaluating Student Participation

One of the key factors leading to a successful class experience for everyone is the ability to cultivate a sense of community and cooperation.

Since many students are quite focused on optimizing their grades, it can be helpful to explicitly include some weight in the final evaluation for this.

A combination of quantitative and qualitative metrics can be used to evaluate the level of participation of the students in the class. It is important to be transparent about the specific way that students will be evaluated.

However, it is also important to express that these metrics are meant to be a guideline, or you may fall victim to Goodhart's law.

Some metrics that we have used in the past include:

- Quantity and quality of feedback given to fellow students on reports and presentations (see Evaluating Projects);
- Filing issues, fixing bugs, answering questions on Stack Overflow, making pull requests or other quantifiable contributions to the code infrastructure;
- Quality and quantity of involvement on the course communication board (Slack, etc.).
- It can be interesting to allow students to nominate others in the class that they feel have been particularly helpful. If following this approach, make sure to require a detailed justification otherwise students may tend to only nominate their friends. Any sort of academic dishonesty (e.g., providing biased peer review because of a conflict of interests) should be penalized.

Finally, there can be some part of the participation grade that is entirely subjective and up to the discretion of the professor and/or teaching assistants.

Duckietown Simulator

We use simulations for both development and evaluation.

Gym Duckietown

We have developed a simulation environment in OpenGL.

Fig. 2 A screenshot of our simple OpenGL based simulation environment

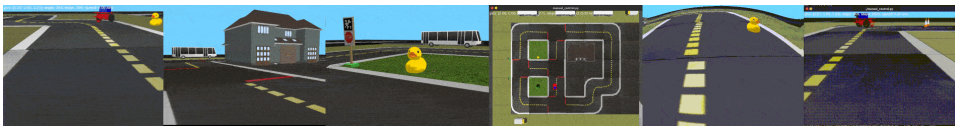
This simulator is used as a development environment in many of the learning experiences. To test exercises in simulation, one simply adds the `--sim` flag to the `dts code workbench` command:

```
dts code workbench --sim
```

For more details about the `dts code` API that is used with the exercises please refer to the Learning Experiences Manual.

The simulator may also be used directly. For example, this is done in the object detection exercise to automate the process of collecting labeled data to train the model.

For full details about how to use the simulator, please refer to the page on simulation in the developer manual.



> **Warning**
>
> If the simulator is used in standalone mode it is slightly different than the version of the simulator that is used for exercise evalution.

The reason for this is partly historical and partly practical. The Gym Duckietown simulation predates the automated evaluation infrastructure. It was originally used as a tool for training reinforcement learning agents. This is the reason that it adheres to the OpenAI Gym API. As such, the focus originally was on making the simulation extremely fast and lightweight, and this is the reason it was written in pure Python.

When we developed the Challenges infrastructure we determined that the simulation was not realistic enough since there were some missing pieces such as motion blur and momentum. For those very motivated to dig into the details, the extra pieces coded are in the challenge-aido_LF-simulator-gym repository.

DuckieMatrix

We are in the process of developing a higher-fidelity simulation environment that we refer to as the DuckieMatrix. This simulation is built using Unity.

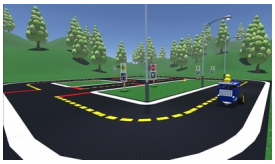


Fig. 3 The soon-to-be-released "DuckieMatrix" simulation environment

For anyone interested in getting a sneak peek, please refer to the DuckieMatrix Development book.

The Code Structure

In this section, we give a brief overview of how the code is structured at a high level, and where you can find various things.

We make extensive use of Docker in our infrastructure. This enables us to keep things compartmentalized. It means that we can take the same "agent", e.g., one that is built as part of a learning experience), and run it many different ways: as locally but in simulation, on an actual Duckiebot, or on a cloud server for evaluation.

> **See also**
>
> For more details, you may want to refer to code hierarchy page the Developer manual.

`dt-core`

The **`dt-core`** repository contains the code that runs the core autonomy stack for the Duckiebot. This includes all the needed components for the lane-following demo as well as some of their core components for more complicated robot behaviors.

Some of the learning-experiences leverage code from this repository, though it is somewhat "hidden" from the student doing the learning experience because it is part of the base docker image on which they are working.

`dt-ros-commons`

The **`dt-ros-commons`** repository is *upstream* of **`dt-core`**, in the sense that the Dockerfile that builds the `dt-core` image builds from the image that is created by [`dt-ros-commons`][dt-ros-commmons].

This repository contains ROS-related configurations and details.

For example, we created a parent class for all ROS nodes called **`dtros`**, and you will also find all of the message definitions for topics that ROS nodes use to communicate in this repository.

It is relatively unlikely that you should need to look into or understand in detail the code in this repository.

`dt-commons`

The dt-commons repository sits *upstream* of dt-ros-commons and contains Duckietown-specific configurations and libraries.

It is very unlikely that you will need to look at the code in this repository.

`dt-machine-learning-base-environment`

The dt-machine-learning-base-environment is *downstream* of dt-ros-commons but includes extra libraries and utilities for using the GPU and machine learning on the Jetson nano. For example, this is used in the object detection learning experience.

duckietown-shell

We have created the **`duckietown-shell`** to abstract away a lot of the details, particularly with respect to the way that Docker images are created and configured.

This is the code that runs the **`dts`** that is used extensively to do exercises, build code and documentation, run code and many other things.

It is very unlikely that you would need to understand the code in this repository. If you are interested to understand better what happens for a specific command that you run with `dts`, it is probably better to start with the implementation of that command in the **`duckietown-shell-commands`** repository.

duckietown-shell-commands

The **`duckietown-shell-commands`** repository contains the implementations of the commands that are run by the Duckietown shell (with `dts` in the command line).

For the specific implementations of how these commands are executed, you can refer to the subfolders in the repository. You also probably should not need to understand the details of how these commands work, but if you get an error you don't understand, it might be a good place to start debugging (in addition to also reporting the bug).

gym-duckietown

The **`gym-duckietown`** repository contains the code that implements the **`gym-duckietown`** simulator.

This simulator is written in OpenGL and Python to be as slim as possible, with a view of being useful for machine learning, as it adheres to the OpenAI Gym API.

It is also used to evaluate exercise submissions.

duckietown-lx

The **`duckietown-lx`** repository contains the starter code and notebooks for the learning experiences.

This is a repository that students will typically fork and use for doing the learning experiences.

duckietown-lx-recipes

The duckietown-lx-recipes repository contains the configurations for the exercises in the **`duckietown-lx`** repository. Unless you are interested in building your own learning experiences, you probably do not need to understand in detail the code in this repository.

duckietown-lx-solutions

The duckietown-lx-solutions repository contains solutions for the exercises in the **`duckietown-lx`** repository. This repository is private, if you would like to gain access, please contact us.

Duckietown Hardware

We are firm believers that you can't learn about robotics without a robot.

Our core objective in the design of the hardware was to provide a platform that is as simple and inexpensive as possible, but still able to provide a wide range of challenging and reproducible learning experiences.

> **See also**
>
> To acquire Duckietown hardware, visit the Duckietown online store.

Duckiebots

The Duckiebot is a small differentiable drive robot that is simple yet powerful. It is equipped with a camera, wheel encoders, as well as other sensors. It is powered by a custom battery that we have built to enable live diagnostics and advanced power management behaviors. The `DB21` family of Duckiebots use NVIDIA Jetson Nano for computation.

Fig. 4 The Duckiebot is a small-scale robot that we have developed for autonomy education and research.

The platform has gone through several iterations, for a full breakdown of the configurations see this page in the Duckiebot Operation Manual. Also, for much more detail about the components of the Duckiebot, as well as its operation, please refer to the Duckiebot Operation Manual.

Duckietown

To guarantee the reliable performance of the Duckiebot, we have also designed a custom environment, the Duckietown, for the Duckiebot to operate in.

Fig. 5 Duckiebots operate in Duckietowns, which can have aribitrary topologies as long as the appearance specifications are respected.

The environment is very tightly specified. For full details, visit the Duckietown Operation Manual.

You might be tempted to build a Duckietown out of off-the-shelf components. While it is totally possible, make sure to understand the underlying design criteria to prevent future headaches.

Duckiedrones

We have also more recently developed a flying robot, that we refer to as the Duckiedrone.

Duckiedrones are Raspberry Pi-powered, and are designed to introduce younger learners to autonomy.

For full details on the platform, visit the Duckiedrone Operation Manual.

Fig. 6 The Duckiedrone is a Raspberry-Pi based autonomous quadcopter.

Course materials based on the Duckiedrone have been developed, largely by Prof. Stefanie Tellex and her team at Brown University.

For details, check out the Introduction to Robotics with Drones book in the library.

Projects

Student projects can be an excellent way to cultivate good class cohesion and promote hands-on learning.

Fig. 7 Projects are a great way to learn by doing. Check out some examples of past projects.

In this page, we will outline the resources that are available to support student projects.

Project Structure

We have experimented with different models for project structure.

One model that has worked extremely well has been to create one large overarching objective, and then break it down into sub-projects. In this way, we can create a type of “startup company” atmosphere that promotes cooperation and collective learning. Often we will assign one group to be responsible for the integration of the work of the other groups.

A different but also very acceptable model is to allow students to work on smaller but more independent projects.

See Student Project Ideas for some concrete ideas for projects. These typically revolve around the idea of improving, expanding or creating new autonomous behaviors.

In either case, it may be a good idea to present some concrete project suggestions and then ask the students to fill out a form (such as this) to indicate their interest in the various topics.

Then, an assignment can be made that optimally allocates students to the topics that interest them the most.

Project Report Templates

If you decide to make the project groups submit progress reports, you may find the following templates a useful starting point for your students to structure their reports:

- **Preliminary Design Report Template**: Usually due about one-third of the way through the allotted period. The principal objective is for the team to converge on the topic that they will work on, how to define its success, and have some idea about its feasibility.
- **Critical Design Report Template**: Usually due about two-thirds of the way through the allotted period. The students should have converged on their project and the scope should be clear. In this document, they outline the implementation strategy and final demo/evaluation plan.
- **Final Report Template**: Due at the end of the allotted time. Should detail what was achieved and document the steps to reproduce it.

Development Workflow for Projects

Although you may exclusively use the **`dts code`** workflow for the exercises, you may find this limiting for more expansive projects. The `dts code` workflow is designed specifically for learning experiences that are quite narrowly scoped and specific.

In the context of projects, the scope may be larger and your students may need more flexibility. As a result, in general, we recommend that students get familiar with the **`DTProjects`** structure. In particular, we have project template repositories for different types of projects. This workflow uses the `dts devel` interface as opposed to `dts code`. A good place to start could be here.

To discuss this topic you can reach out on the #devel-dt-projects Slack channel.

Student Project Ideas

Depending on what topics you are covering as components of your class, possible project ideas may differ.

In general, projects can aim to improve on an existing capability within Duckietown, or add something new. If choosing to add something new, it could be a single component that exists within the larger structure that we have defined, or it could replace everything with some totally different approach (e.g., to use end-to-end machine learning).

> **Note**
>
> We keep a non-exhaustive collection of videos of past student projects on Vimeo. If you would like us to add your project to the list, reach out to us!

Improving Existing Components

A sound strategy for coming up with potential projects is to build upon existing behaviors. The main code repository that contains the algorithmic components that run on the Duckiebot is the dt-core (see The Code Structure for details on the code structure in Duckietown). Within it are packages that are part of the core lane following, or more advanced indefinite navigation, demos.

The core components of lane following include:

- Color correction with the “anti-Instagram” package
- Line detection
- Projection of line segments to the ground plane
- Fusion of the line segments with a histogram filter
- Feedback control to drive down the lane
- A simple finite-state machine

These implementations are meant to be for reference, and there certainly are other ways of implementing each of these blocks. A possible way to structure a project could be to replace or improve on one of these core components and then see the effect on the overall performance of the lane following behavior.

Additional components that contribute to the indefinite navigation (where a Duckiebot drives indefinitely in a city with intersections and traffic lights) include:

- Detection of fiducial markers called “Apriltags”
- Detection of flashing LEDs for intersection coordination
- Detection of the red stop line
- Traversing intersections
- Coordination at intersections
- A more complex configuration of the finite state machine

These represent a possible implementation of this indefinite navigation behavior but, as above, there could be improvements.

Projects of this type are possibly slightly more ambitious projects as these packages are less well-tested, and students should be more prepared to have to work independently to make things work well.

> **Tip**
>
> One idea could be to structure the entire suite of projects as having the objective to get one more complex behavior such as indefinite navigation to work.

Finally, other packages exist in the repository that worked at some point but probably haven't been tested in a while, such as:

- Vehicle detection
- Dead Reckoning

These may serve as starting points or inspiration, but the students will in all likelihood have to do significant testing or develop their replacement.

Hierarchy of potential autonomous behaviors

We note that a neat hierarchy of autonomous behaviors may be envisioned for Duckiebots driving in Duckietowns.

We briefly describe each behavior as source of potential inspiration for additional student projects. This is not a comprehensive list, as other behaviors may be conceived that don't even require city environments.

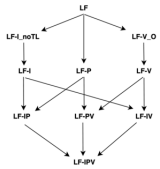


Fig. 8 Autonomous behaviors can build on each other in terms of complexity.

BEHAVIOR NAME	DESCRIPTION	CITY CONFIGURATION
Lane Following (`LF`)	A single Duckiebot drives indefinitely in a Duckietown without intersections.	City loop (without intersections).
`LF` with intersections and no traffic lights (`LF-I_noTL`)	A single Duckiebot drives indefinitely in a Duckietown with intersections. No intersection is equipped with traffic lights. The additional challenge here is introducing a finite state machine, having Duckiebots stop at intersections, read traffic signs, and navigate intersections before switching back to lane following mode.	City with intersections but no traffic lights.
`LF` with vehicles on opposite lanes (`LF-V_O`)	Same as `LF`, but with two Duckiebots on the map starting in opposite lanes. The additional challenge with respect to `LF` is to be robust to sensory perturbations caused by the lights of the vehicle on the opposite lane. Moreover, Duckiebots must at all times stay in their lane (strictly) to ensure success.	City loop (without intersections).
`LF` with intersections (`LF-I`)	A single Duckiebot drives indefinitely in a Duckietown with intersections, which may or may not be equipped with traffic lights. The additional challenge here is centralized coordination (LED detection and interpretation).	City with intersections (with or without traffic lights).
`LF` with pedestrians (`LF-P`)	A single Duckiebot navigates a city without intersections, detecting and avoiding (when possible) “pedestrians” (i.e., duckies). The challenge here is detecting objects, and planning around them.	City loop (without intersections), but with duckies in the road.
`LF` with other vehicles (`LF-V`)	Multiple Duckiebots drive indefinitely in a city without intersections. Duckiebots are allowed to be in the same lane. The challenge here is traffic management, i.e., detecting other Duckiebots and maintaining a safe distance from them.	City loop (without intersections, without duckies).
`LF` with intersections and pedestrians (`LF-IP`)	This challenge is similar to `LF-I`, with the additional complication of potentially having pedestrian inside intersection tiles.	City with intersections (with or without traffic lights), and duckies on the road.

BEHAVIOR NAME	DESCRIPTION	CITY CONFIGURATION
`LF` with pedestrians and other vehicles (`LF-PV`)	The union of `LF-P` and `LF-V`. Duckiebots must be able to detect and avoid both static and moving obstacles (duckies and Duckiebots, respectively).	City loop (without intersections), with duckies on the road.
`LF` with intersections and other vehicles (`LF-IV`)	Multiple Duckiebots navigate indefinitely in a city with intersection, equipped or not with traffic lights. The additional challenge here is dealing with decentralized coordination - i.e., introducing a protocol for having Duckiebots negatiate safe crossing of intersections.	City with intersections (with or without traffic lights), and other vehicles on the same or opposite lanes.
`LF` with intersections, pedestrians, and other vehicles (`LF-IPV`)	This is the ultimate challenge, where any number of Duckiebots can navigate indefinitely in any city configuration, with pedestrians.	All configurations allowed (with intersections, duckies on the road, and multiple vehicles in any lane).

New End-to-End Behaviors

Another different project idea is to replace all the autonomy stack in dt-core with an entirely different approach. To this end, the templates and baselines that were developed for the AI Driving Olympics may prove useful or at least a source of inspiration (although they may need some updating).

> **Note**
>
> Even more advanced behaviors end up becoming published research. You can find some examples as source of inspiration on the research papers page of our website.

Support

There are many ways to get help from the huge community of people who have used and continue to use Duckietown. We suggest the following options:

Community-based Support

> **Tip**
>
> Community-based support is the way to go if you do not need time-sensitive help.

- **Duckietown Slack**: We have a large user community on Slack. In general, we encourage you and your students to sign up and come and say hi. There are many people there willing to help with debugging problems, channels to coordinate development of new behaviors, join beta-testing programs, receive critical Duckietown updates (e.g., on occasional downtimes), etc.
- **Duckietown Stack Overflow**: If you sign up to Slack you will find instructions to join our dedicated Duckietown Stack Overflow space. This is a large repository of questions with accepted answers and, in general, is more archival and structured than the Slack community. You can find instructions on how to join the Duckietown Stack Overflow, e.g., in this Slack post. When facing a technical problem, this is always the first go-to place. Asking (and answering) questions here helps Duckietown become a better place.
- **Duckietown Instructor Slack**: We also host a separate Slack community dedicated only to instructors using Duckietown for education. If you would like to be invited to that community, please send a message to Liam Paull (on the main Duckietown Slack).

- **Email-based support**: In general, we do not have the resources to provide email-based support, as it does not scale. The notable exception is to order hardware component replacements. If you need a replacement or want to place an order, reach out to hardware@duckietown.com.

Priority support

> **Tip**
>
> Priority support is the way to go if you need time-sensitive help from Duckietown staff.

We also offer priority support that comes with certain benefits such as dedicated Slack channels for you and your teaching assistants. We include this level of support in the **Class-in-a-box** bundle. Please email us at info@duckietown.com if you would like more details.

Prerequisites

Before designing a Duckietown class, it is important to evaluate the prerequisites for instructors and students, as well as for supporting materials (space, equipment, infrastructure, etc.).

Instructor background

Aside from the obvious knowledge required to teach the material, it will be helpful if the (main) instructor considers the following.

Teaching Assistants

> **Attention**
>
> It is **strongly** recommended to recruit teaching assistants for your class, if possible.

Working with robots is more involved than a typical class, and will require more support (despite Duckietown's best efforts to minimize the amount of support needed wherever possible!).

In a fully-fledged Duckietown class, with student projects and one robot per child, we have found that a sweet spot is roughly one TA per student group (i.e., 3-5 students), and one TA dedicated to hardware and behind-the-scenes support.

> **Tip**
>
> While teaching assistants can be hired, hence motivated through money, we have found that there are other valuable forms of motivation, e.g.:
>
> - **Ph. D. students** can be excellent TAs, and leverage students groups in the class to implement their research on physical hardware;
> - **Postdoctoral researchers** will be happy to gain teaching experience and can support the teaching efforts by taking ownership of one or more modules (weeks) according to their expertise. Moreover, they can be valuable mentors to the students during projects (if applicable in your course).

Writing code in Python

Fig. 9 Duckietown's stack uses several technologies, some of which are considered prerequisites to a joyful Duckietown experience.

Python is a versatile programming language, amongst the most used in the world. Although it might not be the most performant for robotic applications, it strikes a good balance between simplicity to use and performance.

All the Duckietown notebooks and exercises as well as the code that runs the robot are written in Python.

> **Note**
>
> To help students debug their solutions, someone on staff should be quite familiar with coding in Python.

We also make extensive use of Jupyter notebooks, which are interactive Python files that can be run in the browser.

There are many resources out there to learn Python for free, e.g.,

- learnpython.org.

(This is just an example, we are not affiliated with this resource. Additional options: additional resources).

Version control with Git

The workflow that we propose for completing the exercises includes forking and cloning our `duckietown-lx` repository, as well as adding an upstream remote.

> **Tip**
>
> If you do not recognize terms such as *repository*, *clone*, *fork*, *merge* and in general are not familiar with Git, read the Duckietown pointers on using Git.

Using the Linux Command Line

Fig. 10 Linux terminal usage is streamlined by the Duckietown Shell.

While we have made significant progress in recent years to reduce the requirement that students interact directly with the Linux terminal by introducing the Duckietown Dashboard, some knowledge is still required at this stage.

To streamline operations that would require complex terminal commands, we created the Duckietown Shell. Yet, being familiar with the fundamentals of terminal usage (`ls`, `cat`, `cd`, etc.) is needed and other notions like the Secure shell (`ssh`) may be useful for debugging.

- To install the Duckietown Shell, follow the DTS installation instructions;
- For an introduction to `ssh`, see the Duckietown quick guide to `ssh`;
- for a general introduction to Linux, we recommend (while not being affiliated to) among other resources, the free Linux Journey.

Docker

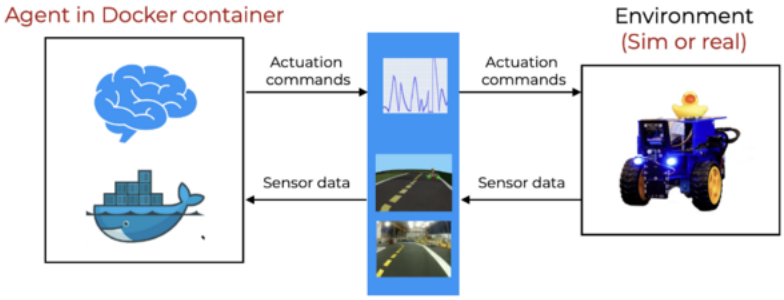


Fig. 11 Docker allows Duckietown to (a) work in very diverse environments reproducibly and (b) have seamless deployment of agents between real and virtual worlds.

Although at first glance it might seem redundant, we make extensive use of Docker under the hood.

Docker is a tool for containerization. In short, we package up code together with all of its dependencies into a "Docker image" which can be downloaded and run.

In this way, we can guarantee (if we do things right) that the code will run properly and reproducibly regardless of the specific computing environment.

We can also rigorously specify interfaces between containers, which enables portability in a very seamless way. This is how we can have one "agent" that can be run in many different ways, such as in the simulator, on the real robot, or in a cloud evaluation.

You should not need to know the details about how this works as we have made every effort to abstract Docker away, but some familiarity with the basics may reduce your anxiety about what is happening.

Robot Operating System (ROS)

Similarly to Docker, although we make some effort to abstract things away in Python libraries, several of the learning experiences leverage the "Robot operating system", which has become almost ubiquitous in robotics.

ROS is a "middleware" software that acts as a *glue* for the different components of the robot autonomy stack and provides tools for message passing, parameter tuning, and debugging among other things.

We have some specific resources for getting started with ROS in the context of Duckietown. It is worth getting familiar with these (and even potentially assigning them as homework assignments for the students).

Going forward, Duckietown will support different middleware solutions, such as ROS2. For additional information on this effort, or if you would like to help, please reach out to us on Slack or at info@duckietown.com.

Student background

What you require students to know before joining your class is very much a design decision that will guide how rapidly your course can progress.

In particular, it will affect the learning experiences and homework assignments that the students will be able to complete.

You may have no say in the matter, depending on your institution's dynamics, but we recommend that you at least ask students at the beginning of the course what their perceived proficiency in certain technical topics is.

Here are a few example course introduction forms for you to take inspiration from:

- Duckietown introduction survey - Google form
- Autonomous Vehicles class - Google form - University of Montreal

In general, your life will be easier if your students have some level of computer science background, and/or are very motivated to learn.

Student Technical Background

Specifics may vary significantly from department to department, e.g., mechanical engineering students might be stronger in control systems and weaker in coding with respect to their computer science peers.

Moreover, technical prerequisites will vary depending on the intended learning outcomes of the class, and in some sense on the level at which it is taught (undergraduate or graduate level).

Here are some **technical prerequisites** that should be considered as guidelines:

- Basic coding skills and tools:
  - **Linux/Ubuntu terminal interface**: the most complete way to interface with Duckietown is via terminal, so basic knowledge of Bash is required (`cd`, `ls`, `mkdir`, …). Using Linux (Ubuntu) typically comes as a shock to some instructors as well as learners, but we strongly recommend throwing your heart over the obstacle and start learning. A life is not enough to learn *everything* there is to know in Linux and we provide step-by-step instructions as well an "operating system", the Duckietown Shell (`dts`) to streamline everything;
  - **Python**: We are going to write "autonomy" code in Python;
  - **Git/GitHub**: We are going to pull, fork, push, branch repositories, etc.

Student Academic Background

You may also decide to cater the course material based on the students' previous exposure to subjects such as:

- Mathematics and Physics
  - Elements of **linear algebra**: matrices are used to represent coordinate systems;
  - Notions of **probability theory**: concepts like Bayes theorem, marginalization, probability distribution will be used to derive perception algorithms for the Duckiebot and Duckiedrone;
  - **Calculus** I: learners should be familiar with the notion of derivatives, ODEs, and ideally of their discrete equivalents (finite differences);
  - Fundamentals of **kinematics**: basic vector algebra, rotation fields, relations between position, velocity and acceleration are going to be used to derive equations of motion.

Space, infrastructure and equipment

Duckietown could be taught completely remotely, or in a traditional "in presence" fashion with frontal presentations only. Nonetheless,

> **Attention**
>
> the best part of teaching with Duckietown is the hands-on work that it facilitates.

To enable this experience, you will require some items.

Space

Space is an important consideration. You will need a space that is:

- indoors;
- large enough to accommodate an assembled Duckietown, built in compliance with Duckietown appearance specifications, and the working positions for students operating it;

- reserved for the duration of the class as reconstructing the Duckietown for every lab session will become tedious.

If there is insufficient space for all students in the class to work at the same time, an effective option is to set up a time-sharing system where students have designated preferential time slots.

When building Duckietown inside a room, you want to consider the following factors:

- Must haves:
  - **passive lighting control**: the goal is creating an environment that as much as possible has uniform, diffused, white lighting, as Duckiebots rely on vision for much of their operations. Varying lighting conditions (colored lights, shadows, time-varying illumination, etc.) will create a strong element of disturbance that will translate into frustration among students. Thick curtains near any window help block the variability of lighting from the sun, and normal ceiling lights may be augmented with properly placed additional light sources. Make sure to avoid reflections that may confuse the Duckiebots.
  - A broadband, appropriately set up internet connection;
- Nice to have:
  - **active lighting control**: installing tunable (in color and intensity) LED lights on the room ceiling will allow us to do great things like simulating night/day cycles, enabling many projects;
  - facilities to securely store and safely charge multiple robots at once;
  - coffee and tea machines!

> **Tip**
>
> Instruct everyone to remove their shoes before walking on a Duckietown, to prevent leaving dirt marks that will create visual elements of disturbance. Bonus points for wearing Duckiesocks!

Technical Infrastructure and Equipment

Additional equipment is required for things to function properly.

Computers

To interact with the robots, students will need a computer with a Ubuntu installation. We **strongly** recommend that this is a native installation (as opposed to a virtual machine), as virtual machines can cause weird and hard-to-debug problems when interacting with low-level components like network interfaces.

The recommended version of Ubuntu (as of January 2024) is Ubuntu 22.04. If the student does not already have this installed, then they can choose a dual-boot setup.

- Mac dual boot instruction
- Windows dual boot instructions

> **Attention**
>
> Every student should have a computer with a native Ubuntu installation:
>
> - **Minimum specifications**: Quad-core at 1.8Ghz, 4GB RAM, 60GB hard drive, GPU compatible with OpenGL 2.1+
> - **Recommended specifications**: Quad-core at 2.1Ghz, 8GB RAM, 120GB hard drive, GPU compatible with OpenGL 2.1+

Follow the instructions to set up the working environment once you have a working Ubuntu installation.

Other operating systems might work, but use them at your own risk. The Duckietown staff is unable to provide support for non-Ubuntu setups, but you can always leverage our community resources to work through your problems if you choose that path.

We are working to provide broader supported compatibility. If you would like additional information on this effort, or to support it, you can reach out to us on Slack.

Network

> **Tip**
>
> If working within a corporate network (e.g., in a University), read the following section carefully and prioritize starting a conversation with your IT people. It might be necessary to create a dedicated subnet for your lab to be compliant with typical corporate security guidelines.

We like to say in our classes that "**90% of problems in robotics come from networks**", as in all likelihood, this is the thing that is going to cause you the most headaches.

You will need a reliable internet connection for students to:

- Download things (such as Docker images) onto their computer;
- Download things (such as Docker images) onto their robot;
- To communicate with their robot.

The faster the better when it comes to the bandwidth of the connection.

We are going to up and download Gigabytes of data (exercises, activities, agent submissions, etc.) and if many students are all trying to do this at the same time, this could slow things down considerably.

Other than the speed, there are a few other technical requirements for your network:

1. Students will need to be able to SSH into their robots, therefore port 22 needs to be open on the router;
2. We make extensive use of the name-discovery tool called Avahi. This is needed so that we don't need to know the IP of the robot to be able to communicate with it. Instead, we can assign a unique name to each robot and then refer to it by name. For example:

   ```
   ping robot_name.local
   ```

3. You can log onto the network with simple credentials (not going through some weird browser authentication).

To do this, Avahi keeps track of all the mappings between the names and the IPs for us. **Some networks do not allow this** - in particular in academic labs, it is very common for network administrators to disable this feature.

As a result, we recommend that you (a) talk to your network administrator and (b) have a dedicated router for the lab that gets its uplink from the university network but that you can directly control. Note that typically this latter option will not be appreciated by your network administrator.

> **Tip**
>
> We have compiled some common problems and solutions that may be helpful. Having some working knowledge of WANs, LANs, Subnets, MAC addresses, DHCP servers, etc., may help along the way.

Fig. 12 Everyone is a learner in Duckietown! Students, Professors, Ph. D. students, Postdocs, and researchers alike find the appropriate challenge for their level.

Class Composition

The makeup of the student body has an impact on the course. Here are a few considerations based on our experience teaching with Duckietown in Universities across different geographies.

> **Note**
>
> We are assuming here that you are planning to teach a “stand-alone” Duckietown classroom, as opposed to integrating Duckietown as part of an existing course.

Class Size

A first important consideration is the class size, we advocate for **limiting enrollment** if possible. We have found that classes of 25-30 students, each with a robot of their own, are manageable.

There are several reasons for this:

1. Smaller classes are more amenable to community-building and the cultivation of the type of atmosphere that we are trying to create.
2. It can be difficult to support the technical challenges that some students may have. Particularly, if you are not assuming a lot of prerequisite technical knowledge, it may require a lot of time to get students up to speed with the basics.
3. In many cases, the difficulty of the course ramps upwards, and it is important that no student is left behind. Having a smaller class facilitates this objective.

Student Background

We have found that it is preferable to have as diverse a student body as possible. This refers to not only the demographics of the students but also their academic background. This is particularly true if you choose to teach a curriculum that is relatively high breadth such as the graduate-level Duckietown curricula.

It is useful to have students who are relatively knowledgeable about the covered topics, so that **each student feels like they have something to contribute**. This is is useful in helping to build a sense of community and strong and balanced student teams if doing projects.

Intended Learning Outcomes

It is important to converge on what you are expecting your students to learn by the end of the course: the intended learning outcomes (ILOs).

Here are some potential learning outcomes to consider:

The Use of Operational Tools

By the end of the course, students can use operational tools to:

- Build a Duckietown robot (Duckiebot, Duckiedrone), given the hardware and a set of instructions. Includes best practices on assembly, calibration and maintenance;
- Configure the software and network, and establish reliable connection between robots and base stations (e.g., the student laptop);
- Be comfortable using the Linux command line;
- Understand the design concepts of the Robotic Operating System (ROS);
- Understand the design concepts of Docker;
- Demonstrate the correct operation of their Duckiebot;
- Use standard tools for software development (e.g., source code repositories, branching and merging);
- Become familiar with the secure shell (`ssh`);
- Articulate the importance of environment (e.g., Duckietown city) element on the robust autonomous behavior of Duckietown robots.
- Design and deploy a robotic agent on a Duckietown robot.

Development Methods and Workflows

Master system development methods, such as:

- Develop ROS software modules and integrate them into the system;
- Utilize the best practices of system development, including test-driven and data-driven development;
- Familiarize with the dynamics of open-source development, including the challenges of integrating independently developed functionalities.

Mastery of Theory

Demonstrate, through the completion of learning experiences that are implemented both in simulation and on real robot hardware, an understanding of theory related to:

- Image processing, Bayesian filtering, localization;
- Navigation, modeling and control, inter-robot coordination;
- Integration of perception and control into complex behaviors;
- Effects of deployment of continuous time algorithms on computers.

Communication and Dissemination

Be able to communicate and effectively disseminate technical information, such as:

- Explaining design choices and trade-offs;
- Presenting project status updates;
- Writing reports describing motivations and approaches to solve problems;
- Documenting their work, by creating step-by-step instruction sets to enable future users to reproduce their results;
- Supporting their peers, and evaluating their work with principles of academic integrity (citing sources, declaration of conflicts of interest, etc.).

Course Structure

Duckietown may be used to teach in different ways.

Classical (Frontal) Teaching

We provide many resources in the form of lecture slides and recordings that can be useful for a traditional frontal teaching style.

In this setup, the instructor lectures and the students take notes and answer questions.

Flipped Classroom

An alternative method of structuring your course is to implement a flipped classroom. In this paradigm, students should be exposed to the material before the class, for example, by looking at the slides and watching the videos, and then time in class is spent helping students work on a task.

The Jupyter notebooks that accompany the learning experiences can be a great option for this type of approach.

A third natural option that we have used extensively is a hybrid of the two (classical and flipped). Particularly if your class sessions are long (longer than 45 minutes), research shows that students have a very difficult time focusing for this amount of time. So a good option can be to alternate between classical lecturing styles and then work together on the notebooks in small groups.

Project-based classes

Structuring learning around the achievement of a concrete goal in a small group is beneficial to ensuring engagement from all learners. For resources related to projects see Projects.

We typically find that groups of 3-5 students can effectively self-organize and collaborate to achieve a goal.

This requires oversight from the course staff. Some class time can be allocated to project meetings where the course staff goes to each group individually to assess progress and offer suggestions.

Common Pitfalls

On this page, we list some common difficulties that we have experienced and some possible methods to remedy or prevent them where possible.

Students dropping out

Unfortunately, sometimes students do drop out of the class. This can be inconvenient, particularly if project groups have already been created because it can leave one team deficient.

Additionally, the hardware (robot) allocated to that student may be partially constructed and it may not be possible to find another student to take the place of the one that has dropped out. It is important to consider in the design of the course that some students may drop out and that you should be able to handle such an occurrence.

Some strategies to minimize the probability of students dropping out include:

- **Starting slowly** - particularly with the hardware component. Some students will be more intimidated by hardware than others, and this can take time to overcome.

One of the most likely reasons a student may drop out is because they feel they have fallen behind and they have no hope of catching up. It is also possible that there could be some problems with some hardware (it is rare, but it happens) and this can further delay students and cause frustration.

This is the nature of working with real robots and should be embraced as much as possible.

- **Allocate as many teaching assistant resources as possible** - in particular at the beginning of the class. Ensure that your TAs are very familiar with the hardware and software before the beginning of the class so that they can effectively help debug problems.
- **Hide the details at the beginning** - particularly for students who have limited experience with the tools (See Student background). We have structured the learning experiences in this way deliberately. For example, the “Braitenberg” learning experience performs end-to-end control with a reactive approach that consists of simple matrix multiplication. The best approach is for the *tools* to progress in complexity at the same rate as the *complexity of the approaches* so that the necessity of the more complex tools can be appreciated.
- **Learning to solve problems** - in some cases, things won’t work properly and it will be because of a student’s error or lack of understanding of the material. It can be tempting for students to blame the parts of the system that are abstracted away and that they don’t understand before

considering that they have made a mistake. In our experience students tend to become subject to the fallacy of imitation, or “superstition”. I.e., “they had this problem and fixed by doing that thing” can lead to very frustrating rabbit holes. Presenting step-by-step debugging instructions could be useful to have students determine the actual original causes of their problems, and fix them.

- **Getting help** - in the case that there is a problem that does not seem to be related to the student’s code, it can cause frustration because the student’s solutions will fail, and they don’t know why. For common problems, refer to the “Troubleshooting Guides” in the Duckiebot operation manual. In case the course staff needs support, refer to Support.

Getting Student Feedback

In our experience, it is well worth the effort to get detailed feedback from your students at regular intervals during the course. We solicit more detailed feedback than our standard University feedback forms.

Here is an example Google form used to solicit feedback on all aspects of the course at its completion.

Duckietown Spirit

> **Note**
>
> Duckietown is a place of joy and relaxed introspection.

Last but not least, one of the things that can really set the Duckietown class is the spirit.

Some students are more attracted to this narrative than others, but regardless, it can serve as a tool for cultivating a strong sense of community around the class which inevitably results in a better overall experience for the students.

Below are some innovative aspects of the “spirit of Duckietown” that we try to embody.

Class Philosophy

The best engineers and scientists are the ones who have solid theoretical foundations, as well as practical experience in the domain of interest.

In autonomous robotics, it is important to get the “feeling” of what makes a robot work, and how the success or failure depends on subtle interaction between many hardware and software components.

To this end, it is necessary to study a complete system like Duckietown. The materials might seem simple, and the appearance might be playful, but the complexity of behaviors and representations is comparable to those of deployed robotic systems.

This class is a collaborative learning experience about modern robotics systems. The fundamental theme of the class is that embodied systems are a particular brand of AI system that has special real-world constraints. The only way to feel those constraints is to experiment with the physical system.

To this end, we like to tell students during the first class, that “Duckietown is a place of joy and relaxed introspection”, where typical stress factors, such as the final grade of the course, should be ignored. The laws of physics are harsh enough judges of our work for anyone to be actually worried about grades. This approach results in students spending a lot of time with their robots, inside their Duckietown, focusing only on making things work *in the way they should*.

Personalized experience

We advocate for a setup where each student gets their own personal robot to build and love. This starts with being able to name their robot - something that immediately establishes a connection between the student and their robot. We encourage students to customize their duckies and bots (being mindful of the potential technical repercussions).

Student code can live on

We strongly encourage you and your students to contribute back to the larger project. This could be through fixing bugs and making pull-requests, or by creating new and creative content or projects that we would be happy to showcase.

Many students tend to be extremely motivated by the possibility that, if they do an excellent job, their project or work could live on beyond just the specific course that they are taking. Students crave to be part of something bigger than the scope of a class.

The Duckies!

> **Attention**
>
> Duckies are entirely non-functional, purely decorative, *yet essential*.

Robots are typically thought of as dangerous, strong, fast, aggressive and unpredictable. We designed Duckiebots instead to be safe, weak and slow. Moreover, through the use of the duckie theme, they are perceived as curious, friendly and fun, breaking preconceptions about robotics: igniting curiosity and attracting broad attention. The Duckietown in our laboratory attracts plenty of attention, even when there are no robots.

We strongly recommend that everybody gets a Duckiebot (with duckies included) when they start the class, as part of a “box ceremony” where each student receives their robot.

Customizing your Duckietown

Your Duckietown is an opportunity to create something playful and fun. You may be surprised at the positive impact and impression that this has on your students. Your students may also become inspired to contribute.

The fusion of art and technology is a powerful tool for unleashing creativity. It is somewhat customary to include in each Duckietown non-functional elements, such as decorative buildings or a background “horizon” from the robot’s point of view, that recall landmarks of the city or region you are in.

Public Demonstration

Whenever possible, we try to end the course with a **public** demonstration. This has many potential benefits:

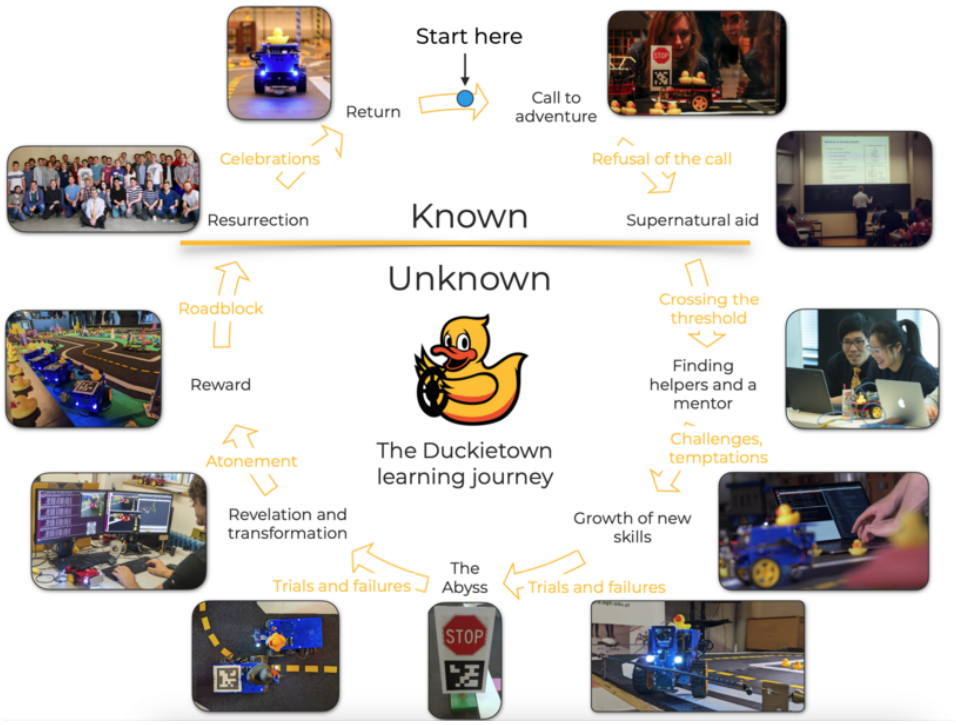
1. it acts as a strong motivation for students to produce something that they are proud of to show the world.
2. it teaches students how to communicate about their work to a more general audience (this is one possible intended learning outcome).
3. the pressure of the demo is an essential aspect of robotics - it is possible to make something work *once* and produce a video, but doing a public demonstration requires ensuring that it works *reliably every time*.
4. it teaches students how to operate under stressful conditions, a skill that is very useful in the real world.
5. it teaches students that evaluation comes from other individuals in the real world, and not from grades derived from tests.

The Hero Journey

Learning is an adventure, and in every proper adventure the hero has ups and downs, companions and mentors, moments of near-despair and eventually of cathartic celebration.

What will lead most of your students away is a drop in motivation coming from repeated failure. But “the master has failed more times than the novice has tried”, and we like to make our students aware of this from the get-go.

Sometimes we go as far as creating a story for them, inspired by the Hero’s Journey:



and here is how we introduce it in the massive open online course “Self-Driving Cars with Duckietown”:

The Duckietown Learning Journey

A word before proceeding further.

While cozy at your computer, you might be wondering what this section is about. This is a warning: there is a “storm” coming your way.

It is such a big and powerful storm that reasonable people would agree that staying home is the best way to safely avoid the inconvenience. We ask you instead to make preparations and step out of your dwelling. Set forth towards the storm!

Like the hero of your favorite mythological saga, you are, right here and right now, given a choice.

If you proceed further, you will take the first steps in a transformative journey, which might lead you to become a roboticist, a hacker, a software magician, or something else you will discover along the way.

In this quest, you will step from the comfort zone of what is known to you, in a journey through the unknown.

Like your favorite hero, you will be able to rely on “supernatural” aid provided by the beauty and perfection of mathematics.

Like your favorite hero, you will meet a mentor along the way. Duckietown will point you in the right direction and provide you with essential tools to succeed. But as with all hero adventures, your mentor will not walk the journey with you. This is your own learning adventure.

But if you look around, you will see that you are not alone in this journey. Many aspiring heroes are setting their first steps, too. Associate, work together, and help each other - because the storm is upon all. Assemble your party before marching forward, as there are many challenges along the path.

If you choose to cross the threshold of your comfort zone, you will begin your initiation quest.

During the first part of this quest, your path will lead downwards. You will face trials and failures which will test your resolve. It is through perseverance and patience that new skills are acquired and cultivated. Remember that “the master has failed more times than the novice has tried.”

Although you will be gathering new technical knowledge in this first part of your quest, you will come to feel overwhelmed. As you learn new things, the immensity of what you do not know becomes more evident, and it weighs. You will be tempted to quit; choose to assert your will and continue instead.

After these doubts, you will hit rock bottom. This place of the mind is known by many names: “the cave of the dragon”, “the supreme ordeal”, or the “abyss of death and rebirth”. Here you will (metaphorically) go through a process of death (nothing works, everything is broken, the course is terrible, and everybody complains) and rebirth.

Rebirth is the second part of the initiation quest, and it happens slowly, in a continuous process of revelations and transformation. As your coding skills start improving, and the “nuisances” of the real-world start to become clearer, things will start working. Slowly… but they will.

The final step of your quest to become a hero will be one of atonement. Your hard work will have produced imperishable fruits for you to banquet from and share at large. Only one last effort now separates you from the comforts of known territory, which you will reach as a person anew.

> Welcome to this Duckietown learning adventure.

Graduate level

In this section, we provide examples of graduate-level classes being taught at various universities.

Autonomous Vehicles (IFT 6757) at the University of Montreal / Mila

Professor Liam Paull at the University of Montreal and Mila has been using the Duckietown platform to teach a graduate course on robotics for a number of years.

This course is taught in a computer science department but is cross-listed to encourage students from a wide variety of backgrounds. The class size is typically limited to about 15-20, and students are selected through an application process. The exact format has varied over the years but the course has always had a strong project-based component.

Fig. 13 The IFT 6757 Autonomous Vehicles Class at the University of Montreal in Fall

2022

Fig. 14 The IFT 6757 Autonomous Vehicles Class at the University of Montreal in Fall

2018

Fig. 15 The IFT 6757 Autonomous Vehicles Class at the University of Montreal in Fall

2017

For details please visit the course website. In particular, the curricula for each past year of the class are:

- 2022
- 2021
- 2020
- 2019

which can all be used for inspiration. We would particularly recommend looking at the most recent version of the class because, naturally, it is refined and improved each year.

ETH Zurich

ETH Zurich has taught two different Duckietown classes through the years as part of the Master of Science in Robotics, Systems and Control in the Department of Mechanical and Process Engineering (D-MAVT).

For more information, please visit the website course.

If you would like your class to show up here, reach out to us with your class link at info@duckietown.com.

Undergraduate level

In this section, we provide examples of undergraduate-level classes being taught at various universities.

EECE 5560 at the University of Massachusetts Lowell

Prof. Paul Robinette has been teaching an undergraduate course in robotics at the University of Massachusetts Lowell for a number of years. The course focuses on the basics of robotic systems.

See full details of the courses including syllabus and assignments at the following links:

- EECE 5560 Fall 2021
- EECE 5560 Spring 2021
- EECE 5560 Fall 2022
- EECE 5560 Spring 2022
- EECE 5560 Fall 2023
- EECE 5560 Spring 2023

If you would like your class to show up here, reach out to us with your class link at info@duckietown.com.

High School level

Duckietown is designed for university-level education, so there are no readily available curricula we offer at this point.

Nonetheless, several high schools have used Duckietown.

If you are interested in exploring the possibility of using Duckietown in your class, robotics club, or summer camp, please reach out to us and we will be happy to discuss your particular case.

Self-Driving Cars with Duckietown (MOOC)

“Self-Driving Cars with Duckietown” is a massive open online course (MOOC): a completely remote course that can be attended by anyone for free.

- For general information, see our website;
- for full access to the course, enroll on edX.

You can watch the trailer of the course below.