Duckietown
Software Infrastructure for Autonomous Robotics
Hello Duckietown!

- Low-cost platform for autonomy education and research
- Robotics and machine learning for students of all ages
- Teaches concepts in perception, planning, and control with miniature autonomous vehicles in the classroom
Duckiebots and Deep Learning

- Duckiebots are battery-powered vehicles
  - Camera is the only sensor
  - Classic computer vision pipeline
    - Multiple stages of hand-designed image processing
    - Proportional integral derivative (PID) controller
    - Each robot needs to be individually calibrated

- Mila is a deep learning research lab
  - Students interested in applying DL & RL
  - Teach students to teach robots how to drive
A Tale of Two Duckietowns

● Gathering real data is real slow
  ○ Time consuming and tedious to collect
    ■ Real robots break down, drain batteries
  ○ Even worse if you need a diverse dataset
    ■ Need to vary lighting conditions, etc.

● Simulation is an appealing alternative...
  ○ Can produce arbitrary amounts of data
  ○ Easy to augment with generated data
  ○ Simplifies reproducibility and verifiability
Machine learning is a changin’

- Evolving tools, frameworks, and languages
- Evolving domain models and architectures
- Evolving hardware technology and platforms
Simulators and environments


Problem: Bias-Variance and Overfitting

<table>
<thead>
<tr>
<th></th>
<th>Low Variance (High Precision)</th>
<th>High Variance (Low Precision)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low Bias</td>
<td><img src="image" alt="Low Bias, Low Variance" /></td>
<td><img src="image" alt="Low Bias, High Variance" /></td>
</tr>
<tr>
<td></td>
<td><strong>Most Accurate</strong></td>
<td><strong>Least Accurate</strong></td>
</tr>
</tbody>
</table>
Software is a changin’

  https://medium.com/@karpathy/software-2-0-a64152b37c35

**Benefits**
- Computationally homogeneous
- Portability & runtime characteristics
- Predictable latency/accuracy tradeoffs
- Modularity, portability, agility

**(Current) Limitations**
- Low interpretability
- Unintuitive failure modes
- Software stack is immature
- Difficult to train and test
Robotics is a changin’
Idealized Agent

Environment

Reality

Agent

Optics

Sensor

ISP

CV Tasks

Control

Tuning Parameters
ROS / Duckietown Tools (rviz, rqt_*, sh)
Let's try to make installs more repeatable

Step 1. Partition hard drive
Step 2. Install Ubuntu
Step 3. Install ROS
Step 4. Install Python stuff
Step 5. Configure network
...
Step 98. source environment.sh
Step 99. catkin_make -C ...

$ dts init_sd_card
Containerization: A User Story

1. Type a short command, e.g. $ dts init_sd_card
2. Follow the installation wizard to flash an image.
3. Transfer flashed SD card to Duckiebot and boot.
4. Open a URL, e.g. http://duckiebot.local:9000/
5. Start or download a container, e.g. duckietown/rpi-duckiebot-base, duckietown/gym-duckietown-agent, duckietown/rpi-joystick-demo
6. Open a browser console and run, e.g. roslaunch joystick ...
Duckietown ROS Nodes
Reproducible Builds & Containerization

● Benefits of containerization:
  ○ Reproducible build and deployment artifacts
  ○ Specified, documented software environments
  ○ Reusable, multi-platform applications

● Disadvantages:
  ○ Learning curve for Docker containers
  ○ GUI applications and X11
  ○ Migration complexity
<table>
<thead>
<tr>
<th>Name</th>
<th>State</th>
<th>Quick actions</th>
<th>Stack</th>
<th>Image</th>
<th>IP Address</th>
<th>Published Ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>eager_lichterman</td>
<td>running</td>
<td></td>
<td></td>
<td>duckietown/software</td>
<td>172.17.0.3</td>
<td></td>
</tr>
<tr>
<td>portainer1.raa07xtrhxc6d9m...</td>
<td>running</td>
<td></td>
<td></td>
<td>portainer/portainer/linux-arm</td>
<td>172.17.0.2</td>
<td>9000:8000</td>
</tr>
</tbody>
</table>

Docker management web UI
Docker management web UI
Containerization: Deployment Models
Containerization: Classic (ROS) Stack

Demos and ROS nodes

Based on ARM32v7, ROS Kinetic Kame, Ubuntu Xenial Xerus, Python 2.7

Docker Client (via get.docker.com)

Lightweight base operating system

ARM-based single board computer (SBC)
Containerization: Laptop / Cloud Stack

- Demos and ROS nodes

- Based on ARM32v7, ROS Kinetic Kame, Ubuntu Xenial Xerus, Python 2.7

- Docker Client (via get.docker.com)

- Any major OS (Windows/MacOS/Deb)

- Any x86 compatible architecture will do
Lessons Learned

- Be careful with Docker inheritance
  - Rebuilds can play havoc on a large tree
- Use a versioning scheme from the outset
- Don’t try to over automate
  - There is still value in teaching manual commands
- Compile nodes and run tests in the build
  - Prevents changes from propagating downstream
- Utilize emulator tools for cross-building
- Don’t compile libraries unnecessarily
  - PiWheels et al. have precompile binaries for ARM
  - Long builds will slow down your development
- Utilize caching whenever possible
AI Driving Olympics

Three principal challenges:

- Lane following
- Lane following with obstacles
- Fleet management / mobility on demand

We evaluate your submission in a simulator and run it on a real Duckiebot!

Coming to NIPS 2018 and ICRA 2019. Register today at duckietown.org
Recap and future work

- Two groups of users: researchers and students
  - One wants reproducibility, both want user friendly tooling
- Need to facilitate comparability with baseline implementations
- Simplify deployment model to fleet with easy rollbacks
- Gracefully degrade services and exert precise control over QOS
- Docker helps us achieve this, but it is not a silver bullet for reproducibility
- By utilizing emulation, we can gradually deploy and fail early
- Hardware-in-the-loop testing can give us further predictability
Help Wanted: Hatchery, a ROS IDE

Learn more at [github.com/duckietown/hatchery](https://github.com/duckietown/hatchery)
Special Thanks

Rusi Hristov
Liam Paull
Andrea Censi