The Duckietown Book

The last version of this book and other documents are available at the URL
`http://book.duckietown.org/`

Table of contents

Part 1
The Duckietown project

CHAPTER 1
What is Duckietown?

1.1. Goals and objectives

Duckietown is a robotics educations and outreach effort.

The most tangible goal of the project is to provide a low-cost educational platform for learning autonomy, consisting of the Duckiebots, an autonomous robot, and the Duckietowns, the infrastructure in which the Duckiebots navigate.

However, we focus on the *learning experience* as a whole, by providing a set of modules teaching plans and other guides, as well as a curated role-play experience.

We have two targets:

1. For **instructors**, we want to create a "class-in-a-box" that allows to offer a modern and engaging learning experience. Currently, this is feasible at the advanced undergraduate and graduate level, though in the future we would like to present the platform as multi-grade experiences.
2. For **self-guided learners**, we want to create a "self-learning experience", that allows to go from zero knowledge of robotics to graduate-level understanding.

In addition, the Duckietown platform has been used as a research platform.

1.2. Learn about the Duckietown educational experience

This video is a Duckumentary about the first version of the class, during Spring 2016. The Duckumentary was shot by Chris Welch.

TODO: Add Duckumentary

Figure 1. The Duckumentary

See also this documentary by Red Hat:

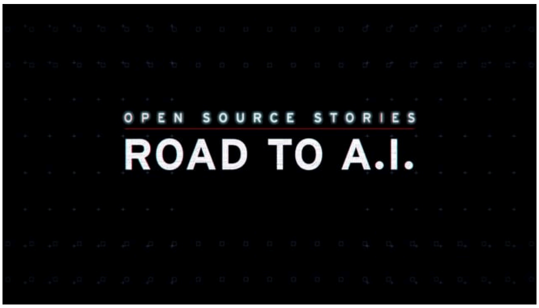

Figure 2. The road to autonomy

If you'd like to know more about the educational experience, [1] present a more formal description of the course design for Duckietown: learning objectives, teaching methods, etc.

1.3. Learn about the platform

The best way to get a sense of how the platform looks is to watch these videos. They show off the capabilities of the platform.

If you would like to know more, the paper [2] describes the Duckiebot and its software. (With 29 authors, we made the record for a robotics conference!)

TODO: add the video here that we showed at ICRA.

Can you do it by night?

Figure 3. Cool Duckietown by night

CHAPTER 2

Duckietown history and future

2.1. The beginnings of Duckietown

The original Duckietown class was at MIT in 2016.

Figure 4. Part of the first MIT class, during the final demo.

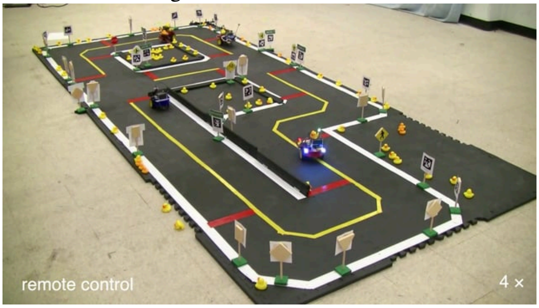

Figure 5. The need for autonomy

Figure 6. Advertisement

Figure 7. The elves of Duckietown

2.2. University-level classes in 2016

Later that year, the Duckietown platform was also used in these classes:

- NCTU 2016 - Prof. Nick Wang;
- RPI 2016 - Prof. John Wen;

Figure 8. Duckietown at NCTU in 2016

2.3. University-level classes in 2017

In 2017, these four courses will be taught together, with the students interacting among institutions:

- ETH Zürich 2017 - Prof. Emilio Frazzoli, Dr. Andrea Censi;
- University of Montreal, 2017 - Prof. Liam Paull;
- TTI/Chicago 2017 - Prof. Matthew Walter;
- National Chiao Tung University, Taiwan - Prof. Nick Wang's course;

Furthermore, the Duckietown platform is used also in the following universities:

- RPI (Jeff Trinkle)
- National Chiao Tung University, Taiwan - Prof. Yon-Ping Chen's *Dynamic system simulation and implementation*.
- Chosun University, Korea - Prof. Woosuk Sung's course;
- Petra Christian University, Indonesia - Prof. Resmana Lim's *Mobile Robot Design Course*

- National Tainan Normal University, Taiwan - Prof. Jen-Jee Chen's *Vehicle to Everything* (V2X) Course;
- Yuan Zhu University, Taiwan - Prof. Kan-Lin Hsiung's Control course;

2.4. Chile

TODO: to write

2.5. Duckietown High School

TODO: to write

CHAPTER 3
First steps

3.1. How to get started

If you are an instructor, please jump to Section 3.2.

If you are a self-guided learner, please jump to Section 3.3.

If you are a company, and interested in working with Duckietown, please jump to Section 3.4.

3.2. Duckietown for instructors

TODO: to write

3.3. Duckietown for self-guided learners

TODO: to write

3.4. Introduction for companies

TODO: to write

3.5. How to keep in touch

TODO: add link to Facebook

TODO: add link to Mailing list

TODO: add link to Slack?

3.6. How to contribute

TODO: If you want to contribute to the software...

TODO: If you want to contribute to the hardware...

TODO: If you want to contribute to the documentation...

TODO: If you want to contribute to the dissemination...

3.7. Frequently Asked Questions

1) General questions

Q: What is Duckietown?

Duckietown is a low-cost educational and research platform.

Q: Is Duckietown free to use?

Yes. All materials are released according to an open source license.

Q: Is everything ready?

Not quite! Please sign up to our mailing list to get notified when things are a bit more ready.

Q: How can I start?

See the section First Steps.

Q: How can I help?

If you would like to help actively, please email duckietown@mit.edu.

2) FAQ by students / independent learners

Q: I want to build my own Duckiebot. How do I get started?

TODO: to write

3) FAQ by instructors

Q: How large a class can it be? I teach large classes.

TODO: to write

Q: What is the budget for the robot?

TODO: to write

Q: I want to teach a Duckietown class. How do I get started?

Please get in touch with us at duckietown@mit.edu. We will be happy to get you started and sign you up to the Duckietown instructors mailing list.

Q: Why the duckies?

Compared to other educational robotics projects, the presence of the duckies is what makes this project stand out. Why the duckies?

We want to present robotics in an accessible and friendly way.

TODO: copy usual discussion from somewhere else.

TODO: add picture of kids with Duckiebots.

Chapter 4
Accounts

4.1. Complete list of accounts

Currently, Duckietown has the following accounts:

- Github: for source code, and issue tracking;
- Slack: a forum for wide communication;
- Twist: to be used for instructors coordination;
- Google Drive: to be used for instructors coordination, maintaining TODOs, etc;
- Dropbox Folders (part of Andrea's personal accounts): to be abandoned;
- Vimeo, for storing the videos;
- The `duckietown-teaching` mailing list, for low-rate communication with instructors;
- We also have a list of addresses, of people signed up on the website, that we didn't use yet;
- The Facebook page.

4.2. For other contributors

If you are an international contributor:

- Sign up on Slack, to keep up with the project.
- (optional) Get Github permissions if you do frequent updates to the repositories.

Part 2
Duckumentation documentation

CHAPTER 5

Contributing to the documentation ✎ 🔗

5.1. Where the documentation is ✎ 🔗

All the documentation is in the repository `duckietown/duckuments`.

The documentation is written as a series of small files in Markdown format.

It is then processed by a series of scripts to create this output:

- a publication-quality PDF;
- an online HTML version, split in multiple pages and with comments boxes.

5.2. Editing links ✎ 🔗

The simplest way to contribute to the documentation is to click any of the "✎" icons next to the headers.

They link to the "edit" page in Github. There, one can make and commit the edits in only a few seconds.

5.3. Comments ✎ 🔗

In the multiple-page version, each page also includes a comment box powered by a service called Disqus. This provides a way for people to write comments with a very low barrier. (We would periodically remove the comments.)

5.4. Installing the documentation system ✎ 🔗

In the following, we are going to assume that the documentation system is installed in `~/duckuments`. However, it can be installed anywhere.

We are also going to assume that you have setup a Github account with working public keys.

We are also going to assume that you have installed the `duckietown/software` in `~/duckietown`.

1) Dependencies (Ubuntu 16.04) ✎ 🔗

On Ubuntu 16.04, these are the dependencies to install:

```
$ sudo apt install libxml2-dev libxslt1-dev
$ sudo apt install libffi6 libffi-dev
$ sudo apt install python-dev python-numpy python-matplotlib
$ sudo apt install virtualenv
$ sudo apt install bibtex2html pdftk
```

2) Download the `duckuments` repo ✎ 🔗

Download the `duckietown/duckuments` repository in that directory:

```
$ git clone git@github.com:duckietown/duckuments ~/duckuments
```

3) Setup the virtual environment

Next, we will create a virtual environment using inside the `~/duckuments` directory. Change into that directory:

```
$ cd ~/duckuments
```

Create the virtual environment using `virtualenv`:

```
$ virtualenv --system-site-packages deploy
```

Other distributions: In other distributions you might need to use `venv` instead of `virtualenv`.

Activate the virtual environment:

```
$ source ~/duckuments/deploy/bin/activate
```

4) Setup the `mcdp` external repository

Make sure you are in the directory:

```
$ cd ~/duckuments
```

Clone the `mcdp` external repository, with the branch `duckuments`.

```
$ git clone -b duckuments git@github.com:AndreaCensi/mcdp
```

Install it and its dependencies:

```
$ cd ~/duckuments/mcdp
$ python setup.py develop
```

> Note: If you get a permission error here, it means you have not properly activated the virtual environment.

Other distributions: If you are not on Ubuntu 16, depending on your system, you might need to install these other dependencies:

```
$ pip install numpy matplotlib
```

5.5. Compiling the documentation

> **Check before you continue**
>
> Make sure you have deployed and activated the virtual environment. You can check this by checking which `python` is active:

```
$ which python
/home/user/duckuments/deploy/bin/python
```

Then:

```
$ cd ~/duckuments
$ make duckuments-dist
```

This creates the directory `duckuments-dist`, which contains another checked out copy of the repository, but with the branch `gh-pages`, which is the branch that is published by Github using the "Github Pages" mechanism.

Check before you continue

At this point, please make sure that you have these two `.git` folders:

```
~/duckuments/.git
~/duckuments/duckuments-dist/.git
```

To compile the docs, run `make clean compile`:

```
$ make clean compile
```

To see the result, open the file

```
./duckuments-dist/master/duckiebook/index.html
```

1) Incremental compilation

If you want to do incremental compilation, you can omit the `clean` and just use:

```
$ make compile
```

This will be faster. However, sometimes it might get confused. At that point, do `make clean`.

5.6. Troubleshooting compilation

Symptom: "Invalid XML"

Resolution: "Markdown" doesn't mean that you can put anything in a file. Except for the code blocks, it must be valid XML. For example, if you use "`>`" and "`<`" without quoting, it will likely cause a compile error.

Symptom: "Tabs are evil"

Resolution: Do not use tab characters. The error message in this case is quite helpful in telling you exactly where the tabs are.

Symptom: The error message contains `ValueError: Suspicious math fragment 'KEYMATHS000END-KEY'`

Resolution: You probably have forgotten to indent a command line by at least 4 spaces. The dollar in the command line is now being confused for a math formula.

5.7. The workflow to edit documentation.

This is the workflow:

1. Edit the Markdown in the `master` branch of the `duckuments` repository.
2. Run `make compile` to make sure it compiles.
3. Commit the Markdown and push on the `master` branch.

Done. A bot will redo the compilation and push the changes in the `gh-pages` branch.

Step 2 is there so you know that the bot will not encounter errors.

5.8. *Deploying the documentation

> **Note:** This part is now done by a bot, so you don't need to do it manually.

To deploy the documentation, jump into the `DUCKUMENTS/duckuments-dist` directory.

Run the command `git branch`. If the out does not say that you are on the branch `gh-pages`, then one of the steps before was done incorrectly.

```
$ cd $DUCKUMENTS/duckuments-dist
$ git branch
...
* gh-pages
...
```

Now, after triple checking that you are in the `gh-pages` branch, you can use `git status` to see the files that were added or modified, and simply use `git add`, `git commit` and `git push` to push the files to Github.

5.9. *Compiling the PDF version

> **Note:** The dependencies below are harder to install. If you don't manage to do it, then you only lose the ability to compile the PDF. You can do `make compile` to compile the HTML version, but you cannot do `make compile-pdf`.

1) Installing `nodejs`

Ensure the latest version (>6) of `nodejs` is installed.

Run:

```
$ nodejs --version
6.xx
```

If the version is 4 or less, remove `nodejs`:

```
$ sudo apt remove nodejs
```

Install `nodejs` using the instructions at this page.

Next, install the necessary Javascript libraries using `npm`:

```
$ cd $DUCKUMENTS
$ npm install MathJax-node jsdom@9.3 less
```

2) Troubleshooting `nodejs` installation problems

The only pain point in the installation procedure has been the installation of `nodejs` packages using `npm`. For some reason, they cannot be installed globally (`npm install -g`).

Do **not** use `sudo` for installation. It will cause problems.

If you use `sudo`, you probably have to delete a bunch of directories, such as: `~/duckuments/node_modules`, `~/.npm`, and `~/.node_modules`, if they exist.

3) Installing Prince

Install PrinceXML from this page.

4) Installing fonts

Copy the `~/duckuments/fonts` directory in `~/.fonts`:

```
$ mkdir -p ~/.fonts    # create if not exists
$ cp -R ~/duckuments/fonts ~/.fonts
```

and then rebuild the font cache using:

```
$ fc-cache -fv
```

5) Compiling the PDF

To compile the PDF, use:

```
$ make compile-pdf
```

This creates the file:

```
./duckuments-dist/master/duckiebook.pdf
```

Chapter 6
Features of the documentation writing system

The Duckiebook is written in a Markdown dialect. A subset of LaTeX is supported.

There are also some additional features that make it possible to create publication-worthy materials.

6.1. Markdown

The Duckiebook is written in a Markdown dialect.

➡ A tutorial on Markdown.

6.2. Embedded LaTeX

You can use $\LaTeX$ math, environment, and references. For example, take a look at

$$x^2 = \int_0^t f(\tau)\ \text{d}\tau$$

or refer to Proposition 1.

Proposition 1. (Proposition example) This is an example proposition: $2x = x + x$.

The above was written as in Listing 1.

```
You can use $\LaTeX$ math, environment, and references.
For example, take a look at

\[
    x^2 = \int_0^t f(\tau)\ \text{d}\tau
\]

or refer to [](#prop:example).

\begin{proposition}[Proposition example]\label{prop:example}
This is an example proposition: $2x = x + x$.
\end{proposition}
```

Listing 1. Use of LaTeX code.

For the LaTeX environments to work properly you *must* add a `\label` declaration inside. Moreover, the label must have a prefix that is adequate to the environment. For example, for a proposition, you must insert `\label{def:name}` inside.

The following table shows the list of the LaTeX environments supported and the label prefix that they need.

TABLE 1. LATEX ENVIRONMENTS AND LABEL PREFIXES

`definition`	`def:name`
`proposition`	`prop:name`
`remark`	`rem:name`
`problem`	`prob:name`
`theorem`	`thm:name`
`lemma`	`lem:name`

Examples of all environments follow.

Definition 1. Lorem

Proposition 2. Lorem

Remark 1. Lorem

Problem 1. Lorem

Example 1. Lorem

Theorem 1. Lorem

Lemma 1. Lorem

TODO: other LaTeX features supported

6.3. LaTeX symbols

The LaTeX symbols definitions are in a file called `docs/symbols.tex`.

Put all definitions there; if they are centralized it is easier to check that they are coherent.

6.4. Variables in command lines and command output

Use the syntax "`![name]`" for describing the variables in the code.

Example .

For example, to obtain:

```
$ ssh robot name.local
```

Use the following:

```
For example, to obtain:

    $ ssh ![robot name].local
```

Make sure to quote (with 4 spaces) all command lines. Otherwise, the dollar symbol confuses the LaTeX interpreter.

6.5. Character escapes

Use the string "`$`" to write the dollar symbol "`$`", otherwise it gets confused with LaTeX math materials. Also notice that you should probably use "USD" to refer to U.S.

dollars.

Other symbols to escape are shown in Table 2.

TABLE 2. SYMBOLS TO ESCAPE

use `$`	instead of `$`
use ```	instead of `` ` ``
use `&#lt;`	instead of `<`
use `&#gt;`	instead of `>`

6.6. Keyboard keys

Use the `kbd` element for keystrokes.

Example .

For example, to obtain:

> Press a then Ctrl-C.

use the following:

```
Press <kbd>a</kbd> then <kbd>Ctrl</kbd>-<kbd>C</kbd>.
```

6.7. Figures

For any element, adding an attribute called `figure-id` with value `fig:figure ID` or `tab:table ID` will create a figure that wraps the element.

For example:

```
<div figure-id="fig:figure ID">
    figure content
</div>
```

It will create HMTL of the form:

```
<div id='fig:code-wrap' class='generated-figure-wrap'>
    <figure id='fig:figure ID' class='generated-figure'>
        <div>
            figure content
        </div>
    </figure>
</div>
```

To add a caption, add an attribute `figure-caption`:

```
<div figure-id="fig:figure ID" figure-caption="This is my caption">
    figure content
</div>
```

Alternatively, you can put anywhere an element `figcaption` with ID `figure id:caption`:

```
<element figure-id="fig:figure ID">
    figure content
</element>

<figcaption id='fig:figure ID:caption'>
    This the caption figure.
</figcaption>
```

To refer to the figure, use an empty link:

```
Please see [](#fig:figure ID).
```

The code will put a reference to “Figure XX”.

6.8. Subfigures

You can also create subfigures, using the following syntax.

```
<div figure-id="fig:big">
    <figcaption>Caption of big figure</figcaption>

    <div figure-id="subfig:first">
        <figcaption>Caption 1</figcaption>
        <p>Content of first subfig</p>
    </div>

    <div figure-id="subfig:second">
        <figcaption>Caption 2</figcaption>
        <p>Content of second subfig</p>
    </div>
</div>
```

Content of first subfig

(a) Caption 1

Content of second subfig

(b) Caption 2

Figure 9. Caption of big figure

6.9. Shortcut for tables

The shortcuts `col2`, `col3`, `col4`, `col5` are expanded in tables with 2, 3, 4 or 5 columns.
The following code:

```
<col2 figure-id="tab:mytable" figure-caption="My table">
    <span>A</span>
    <span>B</span>
    <span>C</span>
    <span>D</span>
</col2>
```

gives the following result:

TABLE 3. MY TABLE

A	B
C	D

1) `labels-row1` and `labels-row1`

Use the classes `labels-row1` and `labels-row1` to make pretty tables like the following.

`labels-row1`: the first row is the headers.

`labels-col1`: the first column is the headers.

TABLE 4. USING CLASS="LABELS-COL1"

header A	B	C	1
header D	E	F	2
header G	H	I	3

TABLE 5. USING CLASS="LABELS-ROW1"

header A	header B	header C
D	E	F
G	H	I
1	2	3

6.10. Linking to documentation from inside and outside the documentation

1) Establishing names of headers

You give IDs to headers using the format:

```
### header title {#topic ID}
```

For example, for this subsection, we have used:

```
### Establishing names of headers {#establishing}
```

With this, we have given this header the ID "`establishing`".

2) Linking from the documentation to the documentation

You can use the syntax:

```
[](#topic ID)
```

to refer to the header.

You can also use some slightly more complex syntax that also allows to link to only the name, only the number or both (Table 6).

TABLE 6. SYNTAX FOR REFERRING TO SECTIONS.

`See [](#establishing).`

See Subsection 6.10.1

`See <a class="only_name" href="#establishing"></a>.`

See Establishing names of headers.

`See <a class="only_number" href="#establishing"></a>.`

See 6.10.1.

`See <a class="number_name" href="#establishing"></a>.`

See Subsection 6.10.1 - Establishing names of headers.

3) Linking to the documentation from outside the documentation

You are encouraged to put links to the documentation from the code or scripts.

To do so, use links of the form:

```
http://purl.org/dth/topic ID
```

Here "`dth`" stands for "Duckietown Help". This link will get redirected to the corresponding document on the website.

For example, you might have a script whose output is:

```
$ rosrun mypackage myscript
Error. I cannot find the scuderia file.
See: http://purl.org/dth/scuderia
```

When the user clicks on the link, they will be redirected to Section 110.2.

6.11. Embedding videos

It is possible to embed Vimeo videos in the documentation.

> Note: Do not upload the videos to your personal Vimeo account; they must all be posted to the Duckietown Engineering account.

This is the syntax:

```
<dtvideo src="vimeo:vimeo ID"/>
```

For example, this code:

```
<div figure-id="fig:example-embed">
    <figcaption>Cool Duckietown by night</figcaption>
    <dtvideo src="vimeo:152825632"/>
</div>
```

produces this result:

Figure 10. Cool Duckietown by night

Depending on the output media, the result will change:

- On the online book, the result is that a player is embedded.
- On the e-book version, the result is that a thumbnail is produced, with a link to the video;
- On the dead-tree version, a thumbnail is produced with a QR code linking to the video (TODO).

6.12. Bibliography

You need to have installed `bibtex2html`.

The system supports Bibtex files.

Place `*.bib` files anywhere in the directory.

Then you can refer to them using the syntax:

```
[](#bib:bibtex ID)
```

For example:

```
Please see [](#bib:siciliano07handbook).
```

Will result in:

> Please see [3].

CHAPTER 7

Documentation style guide

This chapter describes the conventions for writing the technical documentation.

7.1. General guidelines for technical writing

The following holds for all technical writing.

- The documentation is written in correct English.
- Do not say "should" when you mean "must". "Must" and "should" have precise meanings and they are not interchangeable. These meanings are explained in this document.
- "Please" is unnecessary in technical documentation.
  - ✖ "Please remove the SD card."
  - ✔ "Remove the SD card".
- Do not use colloquialisms or abbreviations.
  - ✖ "The pwd is `ubuntu`."
  - ✔ "The password is `ubuntu`."
  - ✖ "To create a ROS pkg…"
  - ✔ "To create a ROS package…"
- Python is capitalized when used as a name.
  - ✖ "If you are using python…"
  - ✔ "If you are using Python…"
- Do not use emojis.
- Do not use ALL CAPS.
- Make infrequent use of **bold statements**.
- Do not use exclamation points.

7.2. Style guide for the Duckietown documentation

- It's ok to use "it's" instead of "it is", "can't" instead of "cannot", etc.
- All the filenames and commands must be enclosed in code blocks using Markdown backticks.
  - ✖ "Edit the ~/.ssh/config file using vi."
  - ✔ "Edit the `~/.ssh/config` file using `vi`."
- Ctrl-C, ssh etc. are not verbs.
  - ✖ "Ctrl-C from the command line".
  - ✔ "Use Ctrl-C from the command line".
- Subtle humor and puns about duckies are encouraged.

7.3. Writing command lines

Use either "`laptop`" or "`duckiebot`" (not capitalized, as a hostname) as the prefix for the command line.

For example, for a command that is supposed to run on the laptop, use:

```
laptop $ cd ~/duckietown
```

It will become:

```
$ cd ~/duckietown
```

For a command that must run on the Duckiebot, use:

```
duckiebot $ cd ~/duckietown
```

It will become:

```
$ cd ~/duckietown
```

If the command is supposed to be run on both, omit the hostname:

```
$ cd ~/duckietown
```

7.4. Frequently misspelled words

- "Duckiebot" is always capitalized.
- Use "Raspberry Pi", not "PI", "raspi", etc.
- These are other words frequently misspelled: 5 GHz WiFi

7.5. Other conventions

When the user must edit a file, just say: "edit `/this/file`".

Writing down the command line for editing, like the following:

```
$ vi /this/file
```

is too much detail.

(If people need to be told how to edit a file, Duckietown is too advanced for them.)

7.6. Troubleshooting sections

Write the documentation as if every step succeeds.

Then, at the end, make a "Troubleshooting" section.

Organize the troubleshooting section as a list of symptom/resolution.

The following is an example of a troubleshooting section.

1) Troubleshooting

Symptom: This strange thing happens.

Resolution: Maybe the camera is not inserted correctly. Remove and reconnect.

> **Symptom:** This other strange thing happens.

Resolution: Maybe the plumbus is not working correctly. Try reformatting the plumbus.

Chapter 8
Knowledge graph

Note: This chapter describes something that is not implemented yet.

8.1. Formalization

1) Atoms

Definition 2. (Atom) An *atom* is a concrete resource (text, video) that is the smallest unit that is individually addressable. It is indivisible.

Each atom as a type, as follows:

```
text
  text/theory
  text/setup
  text/demo
  text/exercise
  text/reference
  text/instructor-guide
  text/quiz
video
  video/lecture
  video/instructable
  video/screencast
  video/demo
```

2) Semantic graph of atoms

Atoms form a directed graph, called “semantic graph”.

Each node is an atom.

The graph has four different types of edges:

- “**Requires**” edges describe a strong dependency: “You need to have done this. Otherwise it will not work.”
- “**Recommended**” edges describe a weaker dependency; it is not strictly necessary to have done that other thing, but it will significantly improve the result of this.
- “**Reference**” edges describe background information. “If you don’t know / don’t remember, you might want to see this”
- “**See also**” edges describe interesting materials for the interested reader. Completely optional; it will not impact the result of the current procedure.

3) Modules

A “module” is an abstraction from the point of view of the teacher.

Definition 3. (Module) A *module* is a directed graph, where the nodes are either atoms or other modules, and the edges can be of the four types described in Subsection 8.1.2.

Because modules can contain other modules, they allow to describe hierarchical contents. For example, a class module is a module that contains other modules; a “degree” is a module that contains “class” modules, etc.

Modules can overlap. For example, a “Basic Object Detection” and an “Advanced Object Detection” module might have a few atoms in common.

8.2. Atoms properties

Each atom has the following **properties**:

- An **ID** (alphanumeric + - and ‘_’). The ID is used for cross-referencing. It is the same in all languages.
- A **type**, as above.

There might be different **versions** of each atom. This is used primarily for dealing with translations of texts, different representations of the same image, Powerpoint vs Keynote, etc.

A version is a tuple of attributes.

The attributes are:

- **Language**: A language code, such as `en-US` (default), `zh-CN`, etc.
- **Mime type**: a MIME type.

Each atom version has:

- A **status** value: one of `draft`, `beta`, `ready`, `to-update` (Table 7).
- A human-readable **title**.
- A human-readable **summary** (1 short paragraph).

TABLE 7. STATUS CODES

`draft`	We just started working on it, and it is not ready for public consumption.
`beta`	Early reviewers should look at it now.
`ready`	The document is ready for everybody.
`to-update`	A new pass is needed on this document, because it is not up to date anymore.

8.3. Markdown format for text-like atoms

For the text-like resources, they are described in Markdown files.

The name of the file does not matter.

All files are encoded in UTF-8.

Each file starts with a `H1` header. The contents is the title.

The header has the following attributes:

1. The ID. (`{#ID}`)
2. The language is given by an attribute `lang` (`{lang=en-US}`).
3. The type is given by an attribute `type` (`{type=demo}`).
4. The status is given by an attribute `status` (`{status=draft}`).

Here is an example of a header with all the attributes:

```
# Odometry calibration {#odometry-calibration lang=en-US type='text/theory' status=ready}

This first paragraph will be used as the "summary" for this text.
```

Listing 2. `calibration.en.md`

And this is how the Italian translation would look like:

```
# Calibrazione dell'odometria {#odometry-calibration lang=it type='text/theory' status=draft}

Questo paragrafo sarà usato come un sommario del testo.
```

Listing 3. `calibration.it.md`

8.4. How to describe the semantic graphs of atoms

In the text, you describe the semantic graph using tags and IDs.

In Markdown, you can give IDs to sections using the syntax:

```
# Setup step 1  {#setup-step1}

This is the first setup step.
```

Then, when you write the second step, you can add a a semantic edge using the following.

```
# Setup step 2  {#setup-step2}

This is the second setup step.

Requires: You have completed the first step in [](#setup-step1).
```

The following table describes the syntax for the different types of semantic links:

TABLE 8. SEMANTIC LINKS

Requires	`Requires: You need to have done [](#setup-step).`
Recommended	`Recommended: It is better if you have setup Wifi as in [](#setup-wifi).`
Reference	`Reference: For more information about rostopic, see [](#rostopic).`
See also	`See also: If you are interested in feature detection, you might want to learn about [SIFT](#SIFT).`

8.5. How to describe modules

TODO: Define a micro-format for this.

Chapter 9
Translations

Note: This part is not implemented yet.

9.1. File organization

Translations are organized file-by-file.

For every file `name.md`, name the translated file `name.language code.md`, where the language code is one of the standard codes, and put it in the same directory.

For example, these could be a set of files, including a Chinese (simplified), italian, and Spanish translation:

```
representations.md
representations.zh-CN.md
representations.it.md
representations.es.md
```

The reason is that in this way you can check automatically from Git whether `representations.zh-CN.md` is up to date or `representations.md` has been modified since.

9.2. Guidelines for English writers

Here are some considerations for the writers of the original version, to make the translators' job easier.

It is better to keep files smallish so that (1) the translation tasks can feel approachable by translators; (2) it is easier for the system to reason about the files.

Name all the headers with short, easy identifiers, and never change them.

9.3. File format

All files are assumed to be encoded in UTF-8.

The header IDs should not be translated and should remain exactly the same. This will allow keeping track of the different translations.

For example, if this is the original version:

```
# Robot uprising {#robot-uprising}

Hopefully it will never happen.
```

Then the translated version should be:

```
# La rivolta dei robot {#robot-uprising}

Speriamo che non succeda.
```

PART 3

Operation manual - Duckiebot

CHAPTER 10
Duckiebot configurations

Here we define the different Duckiebot hardware configurations, and describe their functionalities. This is a good starting point if you are wondering what parts you should purchase to get started. Once you have decided which configuration best suits your needs, you can proceed to purchasing the components for a `C0+wjd` or `C1` Duckiebot.

10.1. Configuration list

- Configuration `C0`: Only camera and motors.
- Configuration `C0+w`: `C0`, plus an additional wireless adapter.
- Configuration `C0+j`: `C0`, plus an additional wireless joypad for remote control.
- Configuration `C0+d`: `C0`, plus an additional USB drive.
- Configuration `C1`: `C0+wjd`, plus LEDs and bumpers.

10.2. Configuration functionality

1) `C0`

This is the minimal configuration for a Duckiebot. It will be able to navigate a Duckietown, but not communicate with other Duckiebots. It is the configuration of choice for tight budgets or when operation of a single Duckiebot is more of interest than fleet behaviours.

TODO: Insert pic of assembled Duckiebot in `C0` configuration.

2) `C0+w`

In this configuration, the minimal `C0` version is augmented with a 5 GHz wireless adapter, which drastically improves connectivity. This feature is particularly useful in connection saturated enviroments, e.g., classrooms.

TODO: Insert pic of assembled Duckiebot in `C0+w` configuration.

3) `C0+j`

In this configuration, the minimal `C0` version is augmented with a 2.4 GHz wireless joypad, used for manual remote control of the Duckiebot. It is particularly useful for getting the Duckiebot our of tight spots or letting younger ones have a drive.

TODO: Insert pic of assembled Duckiebot in `C0+j` configuration.

4) `C0+d`

In this configuration, the minimal `C0` version is augmented with a USB flash hard drive. This drive is convenient for storing videos (logs) as it provides both extra capacity and faster data transfer rates than the microSD card in the Raspberry Pi. Moreover, it is easy to unplug it from the Duckiebot at the end of teh day and bring it over to a computer for downloading and analyzing stored data.

TODO: Insert pic of assembled Duckiebot in `C0+d` configuration.

5) `C0+wjd`

The upgrades of the minimal `C0` version are not mutually exclusive. We will refer to `C0+wjd` when any or all of the add-ons to the minimal version are considered.

TODO: Insert pic of assembled Duckiebot in `C0+wjd` configuration.

6) `C1`

This is the ultimate Duckiebot configuration and it includes the necessary hardware for controlling and placing 5 RGB LEDs on the Duckiebot. It is the necessary configuration to enable communication between Duckiebots, hence fleet behaviours (e.g., negotiating corssing an intersection).

TODO: Insert pic of assembled Duckiebot in `C1` configuration.

CHAPTER 11

Acquiring the parts for the Duckiebot `C0`

The trip begins with acquiring the parts. Here, we provide a link to all bits and pieces that are needed to build a Duckiebot, along with their price tag. If you are wondering what is the difference between different Duckiebot configurations, read this.

In general, keep in mind that:

- The links might expire, or the prices might vary.
- Shipping times and fees vary, and are not included in the prices shown below.
- Substitutions are OK for the mechanical components, and not OK for all the electronics, unless you are OK in writing some software.
- Buying the parts for more than one Duckiebot makes each one cheaper than buying only one.

> **Requires:** - Cost: USD 169 + Shipping Fees (minimal configuration `C0`) - Time: 15 days (average shipping for cheapest choice of components)
>
> Results: - A kit of parts ready to be assembled in a `C0` or `C0+wjd` configuration.
>
> Next Steps: - After receiving these components, you are ready to do some soldering before assembling your `C0` or `C0+wjd` Duckiebot.

TODO: Add a different "Tools" section in the table (e.g., solderer), or add in the resoruces beginning snippet; Differentiate pricing for bulk vs detail purchase (?)

11.1. Bill of materials

TABLE 9. BILL OF MATERIALS

Chassis	USD 20
Camera with 160-FOV Fisheye Lens	USD 22
Camera Mount	USD 8.50
300mm Camera Cable	USD 2
Raspberry Pi 3 - Model B	USD 35
Heat Sinks	USD 5
Power supply for Raspberry Pi	USD 7.50
16 GB Class 10 MicroSD Card	USD 10
Mirco SD card reader	USD 6
Stepper Motor HAT	USD 22.50
Stacking Header	USD 2.50/piece
Battery	USD 20
16 Nylon Standoffs (M2.5 12mm F 6mm M	USD 0.05/piece
4 Nylon Hex Nuts (M2.5)	USD 0.02/piece
4 Nylon Screws (M2.5x10)	USD 0.05/piece
2 Zip Ties (300x5mm)	USD 9
Wireless Adapter (5 GHz) (`C0+w`)	USD 20
Joypad (`C0+j`)	USD 10.50
Tiny 32GB USB Flash Drive (`C0+d`)	USD 12.50
Total for `C0` configuration	USD 159
Total for `C0+w` configuration	USD 179
Total for `C0+j` configuration	USD 169.50
Total for `C0+d` configuration	USD 171.50
Total for `C0+wjd` configuration	USD 212

TODO: modify to account for new USB to wires power solution.

11.2. Chassis

We selected the Magician Chassis as the basic chassis for the robot (Figure 11).

We chose it because it has a double-decker configuration, and so we can put the battery in the lower part.

The chassis pack includes the motors and wheels as well as the structural part.

The price for this in the US is about USD 15-30.

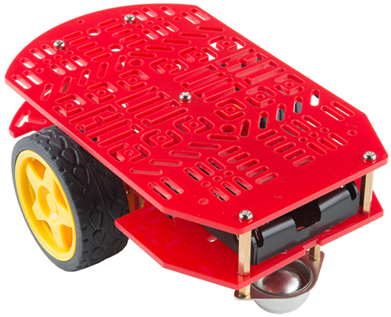

Figure 11. The Magician Chassis

11.3. Raspberry Pi 3 - Model B

The Raspberry Pi is the central computer of the Duckiebot. Duckiebots use Model B (Figure 12) (A1.2GHz 64-bit quad-core ARMv8 CPU, 1GB RAM), a small but powerful computer.

Figure 12. The Raspberry Pi 3 Model B

The price for this in the US is about USD 35.

1) Power Supply

We want a hard-wired power source (5VDC, 2.4A, Micro USB) to supply the Raspberry Pi (Figure 13).

Figure 13. The Power Supply

The price for this in the US is about USD 5-10.

2) Heat Sinks

The Raspberry Pi will heat up significantly during use. It is warmly recommended to add heat sinks, as in Figure 14. Since we will be stacking HATs on top of the Raspberry Pi with 15 mm standoffs, the maximum height of the heat sinks should be well below 15 mm. The chip dimensions are 15x15mm and 10x10mm.

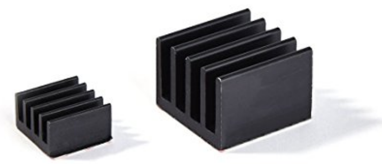

Figure 14. The Heat Sinks

3) Class 10 MicroSD Card

The MicroSD card (Figure 15) is the hard disk of the Raspberry Pi. 16 Gigabytes of capacity are sufficient for the system image.

Figure 15. The MicroSD card

4) Mirco SD card reader

A microSD card reader (Figure 16) is useful to copy the system image to a Duckiebot from a computer to the Raspberry Pi microSD card, when the computer does not have a native SD card slot.

Figure 16. The Mirco SD card reader

11.4. Camera

The Camera is the main sensor of the Duckiebot. All versions equip a 5 Mega Pixels 1080p camera with wide field of view (160°) fisheye lens (Figure 17).

Figure 17. The Camera with Fisheye Lens

1) Camera Mount

The camera mount (Figure 18) serves to keep the camera looking forward at the right angle to the road (looking slightly down). The front cover is not essential.

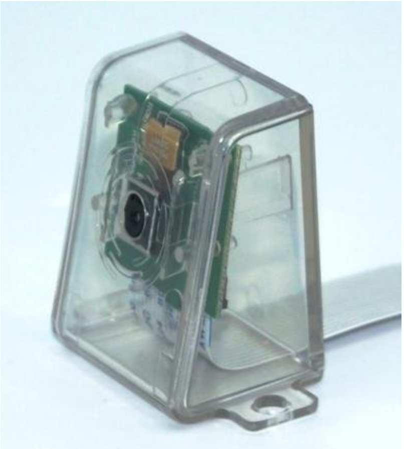

Figure 18. The Camera Mount

2) 300mm Camera Cable

A longer (300 mm) camera cable Figure 19 make assembling the Duckiebot easier, allowing for more freedom in the relative positioning of camera and computational stack.

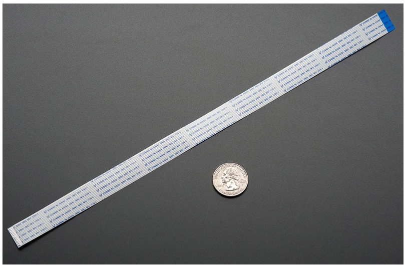

Figure 19. A 300 mm camera cable for the Raspberry Pi

11.5. DC Stepper Motor HAT

We use the DC Stepper motor HAT (Figure 26) to control the DC motors that drive the wheels. This item will require soldering to be functional.

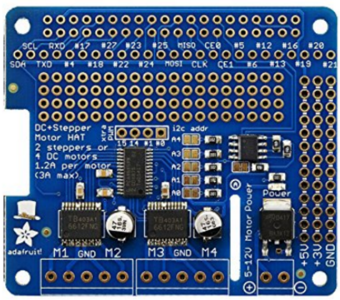

Figure 20. The Stepper Motor HAT

1) Stacking Headers

We use a long 20x2 stacking header (Figure 21) to connect the Raspberry Pi with the DC Stepper Motor HAT. This item will require soldering to be functional.

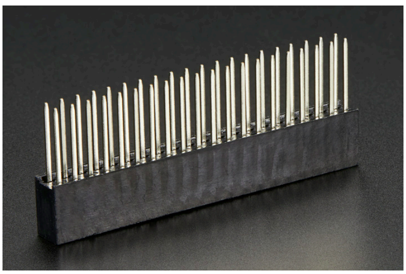

Figure 21. The Stacking Headers

11.6. Battery

The battery (Figure 22) provides power to the Duckiebot.

We choose this battery because it has a good combination of size (to fit in the lower deck of the Magician Chassis), high output amperage (2.4A and 2.1A at 5V DC) over two USB outputs, a good capacity (10400 mAh) at an affordable price (USD 20).

Figure 22. The Battery

11.7. Standoffs, Nuts and Screws

We use non electrically conductive standoffs (M2.5 12mm F 6mm M), nuts (M2.5), and screws (M2.5x10mm) to hold the Raspberry Pi to the chassis and the HATs stacked on top of the Raspberry Pi.

The Duckiebot requires 8 standoffs, 4 nuts and 4 screws.

Figure 23. Standoffs, Nuts and Screws

11.8. Zip Tie

Two 300x5mm zip ties are needed to keep the battery at the lower deck from moving around.

Figure 24. The zip ties

11.9. Configuration `C0-w`

1) Wireless Adapter (5 GHz)

The Edimax AC1200 EW-7822ULC 5 GHz WiFi adpater (Figure 25) boosts the connectivity of the Duckiebot, especially useful in busy Duckietowns (e.g., classroom).

Figure 25. The Edimax AC1200 EW-7822ULC wifi adapter

11.10. Configuration `C0-j`

1) Joypad

The joypad is used to manually remote control the Duckiebot. Any 2.4 GHz wireless controller (with a *tiny* USB dongle) will do.

The model linked in the table (Figure 26) does not include batteries (required: 2 AA 1.5V).

Figure 26. A Wireless Joypad

TODO: Add figure with 2 AA batteries

11.11. Configuration `C0-d`

1) Tiny 32GB USB Flash Drive

Iin configuration `C0+d`, the Duckiebot is equppied with a “external” hard drive (Figure 27). This add-on is very convenient to store logs during experiments and later port them to a workstation for analysis. It provides storage capacity and faster data transfer than the MicroSD card.

Figure 27. The Tiny 32GB USB Flash Drive

CHAPTER 12

Soldering boards for `C0`

> Assigned to: Shiying

> Resources necessary:
>
> > **Requires:** - Duckiebot `C0+wjd` parts. The acquisition process is explained in Chapter 11. The configurations are described in Chapter 10.
>
> > **Requires:** - Time: ??? minutes
>
> Results:
>
> - ...
>
> TODO: finish above

CHAPTER 13
Preparing the power cable for `C0`

In configuration `C0` we will need a cable to power the DC motor hat from the battery. The keen observer might have noticed that such cable was not included in the `C0` Duckiebot parts chapter. Here, we create this cable by splitting open any USB-A cable, identifying and stripping the power wires, and using them to power the DC motor HAT. If you are unsure about the definitions of the different Duckiebot configurations, read Chapter 10.

It is important to note that these instructions are relevant only for assembling a `C0+wjd` configuration Duckiebot. If you intend to build a `C1` configuration Duckiebot, you can skip these instructions.

> Resources necessary:
>
> > **Requires:** - One male USB-A to anything cable - a pair of scissors - a multimeter (only if you are not purchasing the suggested components)
>
> > **Requires:** - Time: 5 minutes
>
> Results: - One male USB-A to wires power cable

13.1. Step 1: Find a cable

To begin with, find a male USB-A to anything cable.

If you have purchased the suggested components listed in Chapter 11, you can use the longer USB cable contained inside the battery package (Figure 28), which will be used as an example in these instructions.

Figure 28. The two USB cables in the suggested battery pack.

Put the shorter cable back in the box, and open the longer cable (Figure 29)

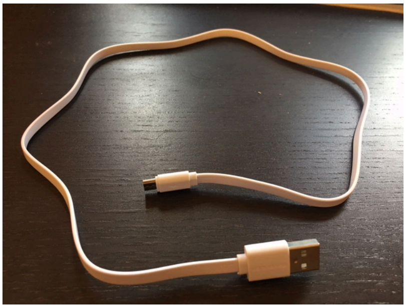

Figure 29. Take the longer cable, and put the shorter on back in the box.

13.2. Step 2: Cut the cable

> **Check before you continue**
> Make sure the USB cable is *unplugged* from any power source before proceeding.

Take the scissors and cut it (Figure 30) at the desired length from the USB-A port.

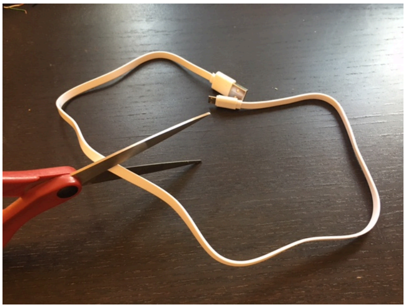

Figure 30. Cut the USB cable using the scissors.

The cut will look like in Figure 31.

Figure 31. A cut USB cable.

13.3. Step 3: Strip the cable

Paying attention not to get hurt, strip the external white plastic. A way to do so without damaging the wires is shown in Figure 32.

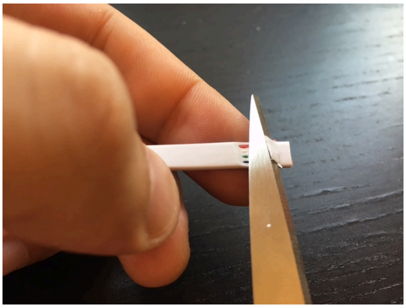

Figure 32. Stripping the external layer of the USB cable.

After removing the external plastic, you will see four wires: black, green, white and red (Figure 33).

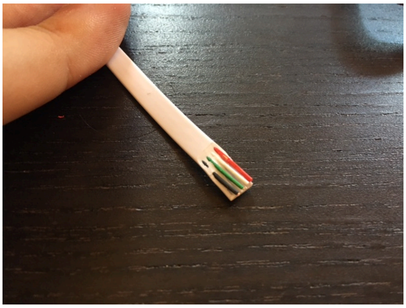

Figure 33. Under the hood of a USB-A cable.

Once the bottom part of the external cable is removed, you will have isolated the four wires (Figure 34).

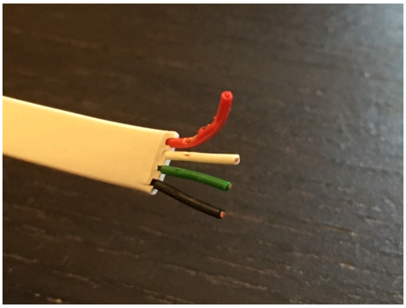

Figure 34. The four wires inside a USB-A cable.

13.4. Step 3: Strip the wires

> **Check before you continue**
> Make sure the USB cable is *unplugged* from any power source before proceeding.

Once you have isolated the wires, strip them, and use the scissors to cut off the data wires (green and white, central positions) (Figure 35).

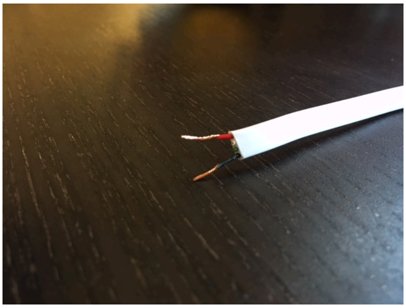

Figure 35. Strip the power wires and cut the data wires.

If you are not using the suggested cable, or want to verify which are the data and power wires, continue reading.

13.5. Step 4: Find the power wires

If you are using the USB-A cable from the suggested battery pack, black and red are the power wires and green and white are instead for data.

If you are using a different USB cable, or are curious to verify which the power wires are, continue reading here.

Plug the USB port inside a power source, e.g., the Duckiebot's battery. You can use some scotch tape to keep the cable from moving while probing the different pairs of wires with a multimeter. The voltage across the pair of power cables will be roughly twice the voltage between a power and data cable. The pair of data cables with have no voltage differential across them. If you are using the suggested Duckiebot battery as power source, you will measure around 5V across the power cables (Figure 36).

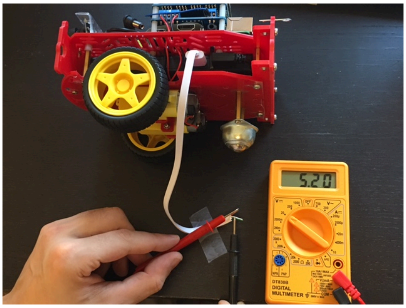

Figure 36. Finding which two wires are for power.

13.6. Step 5: Test correct operation

You are now ready to secure the power wires to the DC motor HAT power pins. To do so though, you need to have soldered the boards first. If you have not done so yet, read Chapter 12.

If you have soldered the boards already, you may test correct functionality of the newly crafted cable. Connect the battery with the DC motor HAT by making sure you plug the black wire in the pin labeled with a minus: `-` and the red wire to the plus: `+` (Figure 37).

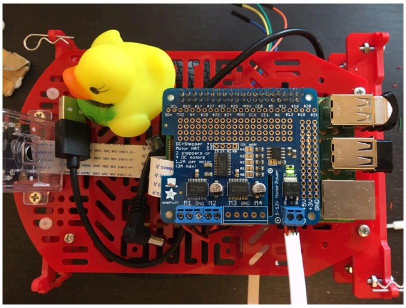

Figure 37. Connnect the power wires to the DC motor HAT

If the green LED turns on, the DC motor HAT is receving power.

CHAPTER 14
Assembling the Duckiebot `C0`

> Assigned to: Shiying

> **Requires:** - Duckiebot `C0+wjd` parts. The acquisition process is explained in Chapter 11.
>
> - Having soldered the `C0+wjd` parts. The soldering process is explained in Chapter 12.
> - Having prepared the power cable. The power cable preparation is explained in Chapter 13.
> - Time: about ??? minutes.
>
> TODO: estimate time.
>
> Results:
>
> - An assembled Duckiebot in configuration `C0+wjd`.

Shiying: here will be the instruction about assembling the Duckiebot.

Chapter 15
Reproducing the image

> Assigned to: Andrea

These are the instructions to reproduce the Ubuntu image that we use.

Please note that the image is already available, so you don't need to do this manually.

However, this documentation might be useful if you would like to port the software to a different distribution.

> Resources necessaries:
>
> > **Requires:** Internet connection to download the packages. Requires: A PC running any Linux with an SD card reader. Requires: Time: about 20 minutes.
>
> Results:
>
> - A baseline Ubuntu Mate 16.04.2 image with updated software.

15.1. Download and uncompress the Ubuntu Mate image

Download the image from the page

https://ubuntu-mate.org/download/

The file we are looking for is:

```
filename: ubuntu-mate-16.04.2-desktop-armhf-raspberry-pi.img.xz
    size: 1.2 GB
  SHA256: dc3afcad68a5de3ba683dc30d2093a3b5b3cd6b2c16c0b5de8d50fede78f75c2
```

After download, run the command `sha256sum` to make sure you have the right version:

```
$ sha256sum ubuntu-mate-16.04.2-desktop-armhf-raspberry-pi.img.xz
dc3afcad68a5de3ba683dc30d2093a3b5b3cd6b2c16c0b5de8d50fede78f75c2
```

If the string does not correspond exactly, your download was corrupted. Delete the file and try again.

Then decompress using the command `xz`:

```
$ xz -d ubuntu-mate-16.04.2-desktop-armhf-raspberry-pi.img.xz
```

15.2. Burn the image to an SD card

Next, burn the image on to the SD card.

→ This procedure is explained in Section 95.2.

1) Verify that the SD card was created correctly

Remove the SD card and plug it in again in the laptop.

Ubuntu will mount two partitions, by the name of `PI_ROOT` and `PI_BOOT`.

2) Installation

Boot the disk in the Raspberry Pi.

Choose the following options:

```
language: English
username: ubuntu
password: ubuntu
hostname: duckiebot
```

Choose the option to log in automatically.

Reboot.

3) Update installed software

The WiFi was connected to airport network `duckietown` with password `quackquack`.

Afterwards I upgraded all the software preinstalled with these commands:

```
$ sudo apt update
$ sudo apt dist-upgrade
```

Expect `dist-upgrade` to take quite a long time (up to 2 hours).

15.3. Raspberry Pi Config

The Raspberry Pi is not accessible by SSH by default.

Run `raspi-config`:

```
$ sudo raspi-config
```

choose "3. Interfacing Options", and enable SSH,

We need to enable the camera and the I2C bus.

choose "3. Interfacing Options", and enable camera, and I2C.

Also disable the graphical boot

15.4. Install packages

Install these packages.

Etckeeper:

```
$ sudo apt install etckeeper
```

Editors / shells:

```
$ sudo apt install -y vim emacs byobu zsh
```

Git:

```
$ sudo apt install -y git git-extras
```

Other:

```
$ sudo apt install htop atop nethogs iftop
$ sudo apt install aptitude apt-file
```

Development:

```
$ sudo apt install -y build-essential libblas-dev liblapack-dev libatlas-base-dev gfortran
libyaml-cpp-dev
```

Python:

```
$ sudo apt install -y python-dev ipython python-sklearn python-smbus
$ sudo apt install -y python-termcolor
$ sudo pip install scipy --upgrade
```

I2C:

```
$ sudo apt install -y i2c-tools
```

15.5. Install Edimax driver

First, mark the kernel packages as not upgradeable:

```
$ sudo apt-mark hold raspberrypi-kernel raspberrypi-kernel-headers
raspberrypi-kernel set on hold.
raspberrypi-kernel-headers set on hold
```

Then, download and install the Edimax driver from this repository.

```
$ git clone git@github.com:duckietown/rtl8822bu.git
$ cd rtl8822bu
$ make
$ sudo make install
```

15.6. Install ROS

Install ROS.

➡ The procedure is given in Section 107.1.

15.7. Wireless configuration (old version)

This is the old version.

There are two files that are important to edit.

The file `/etc/network/interfaces` should look like this:

```
# interfaces(5) file used by ifup(8) and ifdown(8)
# Include files from /etc/network/interfaces.d:
#source-directory /etc/network/interfaces.d

auto wlan0

# The loopback network interface
auto lo
iface lo inet loopback

# Wireless network interface
allow-hotplug wlan0
iface wlan0 inet dhcp
wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet dhcp
```

The file `/etc/wpa_supplicant/wpa_supplicant.conf` should look like this:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
ssid="duckietown"
psk="quackquack"
proto=RSN
key_mgmt=WPA-PSK
pairwise=CCMP
auth_alg=OPEN
}
network={
   key_mgmt=NONE
}
```

15.8. Wireless configuration

The files that describe the network configuration are in the directory

```
/etc/NetworkManager/system-connections/
```

This is the contents of the connection file `duckietown`, which describes how to connect to the `duckietown` wireless network:

```
[connection]
id=duckietown
uuid=e9cef1bd-f6fb-4c5b-93cf-cca837ec35f2
type=wifi
permissions=
secondaries=
timestamp=1502254646

[wifi]
mac-address-blacklist=
mac-address-randomization=0
mode=infrastructure
ssid=duckietown

[wifi-security]
group=
key-mgmt=wpa-psk
pairwise=
proto=
psk=quackquack

[ipv4]
dns-search=
method=auto

[ipv6]
addr-gen-mode=stable-privacy
dns-search=
ip6-privacy=0
method=auto
```

This is the file

```
/etc/NetworkManager/system-connections/create-5ghz-network
```

Contents:

```
[connection]
id=create-5ghz-network
uuid=7331d1e7-2cdf-4047-b426-c170ecc16f51
type=wifi
# Put the Edimax interface name here:
interface-name=wlx74da38c9caa0 - to change
permissions=
secondaries=
timestamp=1502023843

[wifi]
band=a
# Put the Edimax MAC address here
mac-address=74:DA:38:C9:CA:A0 - to change
mac-address-blacklist=
mac-address-randomization=0
mode=ap
seen-bssids=
ssid=duckiebot-not-configured

[ipv4]
dns-search=
method=shared

[ipv6]
addr-gen-mode=stable-privacy
dns-search=
ip6-privacy=0
method=ignore
```

Note that there is an interface name and MAC address that need to be changed on each PI.

15.9. SSH server config

This enables the SSH server:

```
$ sudo systemctl enable ssh
```

15.10. Create swap Space

Do the following:

Create an empty file using the `dd` (device-to-device copy) command:

```
$ sudo dd if=/dev/zero of=/swap0 bs=1M count=512
```

This is for a 512 MB swap space.

Format the file for use as swap:

```
$ sudo mkswap /swap0
```

Add the swap file to the system configuration:

```
$ sudo vi /etc/fstab
```

Add this line to the bottom:

```
/swap0 swap swap
```

Activate the swap space:

```
$ sudo swapon -a
```

15.11. Passwordless sudo

First, make `vi` the default editor, using

```
$ sudo update-alternatives --config editor
```

and then choose `vim.basic`.

Then run:

```
$ sudo visudo
```

And then change this line:

```
%sudo   ALL=(ALL:ALL) ALL
```

into this line:

```
%sudo   ALL=(ALL:ALL) NOPASSWD:ALL
```

15.12. Clean up

You can use the command `dpigs` to find out which packages take lots of space.

```
$ sudo apt install wajig  debian-goodies
```

Either:

```
$ wajig large
$ dpigs -H -n 20
```

Stuff to remove:

```
$ sudo apt remove thunderbird
$ sudo apt remove libreoffice-\*
$ sudo apt remove openjdk-8-jre-headless
$ sudo apt remove fonts-noto-cjk
$ sudo apt remove brasero
```

At the end, remove extra dependencies:

```
$ sudo apt autoremove
```

And remove the `apt` cache using:

```
$ sudo apt clean
```

The total size should be around 6.6GB.

15.13. Ubuntu user configuration

1) Groups

You should make the `ubuntu` user belong to the `i2c` and `input` groups:

```
$ sudo adduser ubuntu i2c
$ sudo adduser ubuntu input
```

: forgot to add to aug20 image:

```
$ sudo adduser ubuntu video
```

You may need to do the following (but might be done already through `raspi-config`):

```
$ sudo udevadm trigger
```

2) Basic SSH config

Do the basic SSH config.

→ The procedure is documented in Section 97.3.

Note: this is not in the aug10 image.

3) Passwordless SSH config

Add `.authorized_keys` so that we can all do passwordless SSH.

The key is at the URL

```
https://www.dropbox.com/s/pxyou3qy1p8m4d0/duckietown_key1.pub?dl=1
```

Download to `.ssh/authorized_keys`:

```
$ curl -o .ssh/authorized_keys URL above
```

4) Shell prompt

Add the following lines to `~ubuntu/.bashrc`:

```
echo ""
echo "Welcome to a duckiebot!"
echo ""
echo "Reminders:"
echo ""
echo "1) Do not use the user 'ubuntu' for development - create your own user."
echo "2) Change the name of the robot from 'duckiebot' to something else."
echo ""

export EDITOR=vim
```

15.14. Check that all required packages were installed

At this point, before you copy/distribute the image, create a user, install the software, and make sure that `what-the-duck` does not complain about any missing package.

(Ignore `what-the-duck` 's errors about things that are not set up yet, like users.)

15.15. Creating the image

You may now want to create an image that you can share with your friends. They will think you are cool because they won't have to duplicate all of the work that you just did. Luckily this is easy. Just power down the duckiebot with:

```
$ sudo shutdown -h now
```

and put the SD card back in your laptop.

➡ The procedure of how to burn an image is explained in Section 95.2; except you will invert the `if` and `of` destinations.

You may want to subsequently shrink the image, for example if your friends have smaller SD cards than you.

➡ The procedure of how to shrink an image is explained in Section 95.3.

15.16. Some additions since last image to add in the next image

Note here the additions since the last image was created.

Create a file

```
/etc/duckietown-image.yaml
```

Containing these lines

```
base: Ubuntu 16.04.2
date: DATE
comments: |
    any comments you have
```

So that we know which image is currently in used.

Install `ntpdate`:

```
$ sudo apt install ntpdate
```

> **Note:** We should install Git LFS on the Raspberry Pi, but so far AC did not have any luck. See Section 105.1.

CHAPTER 16

Installing Ubuntu on laptops

Assigned to: Andrea

Before you prepare the Duckiebot, you need to have a laptop with Ubuntu installed.

> **Requires:** A laptop with free disk space.
>
> **Requires:** Internet connection to download the Ubuntu image.
>
> **Requires:** About ??? minutes .
>
> TODO: estimate time
>
> Results:
>
> - A laptop ready to be used for Duckietown.

16.1. Install Ubuntu

Install Ubuntu 16.04.2.

→ For instructions, see for example this online tutorial.

On the choice of username: During the installation, create a user for yourself with a username different from `ubuntu`, which is the default. Otherwise, you may get confused later.

16.2. Install useful software

Use `etckeeper` to keep track of the configuration in `/etc`:

```
$ sudo apt install etckeeper
```

Install `ssh` to login remotely and the server:

```
$ sudo apt install ssh
```

Use `byobu`:

```
$ sudo apt install byobu
```

Use `vim`:

```
$ sudo apt install vim
```

Use `htop` to monitor CPU usage:

```
$ sudo apt install htop
```

Additional utilities for `git`:

```
$ sudo apt install git git-extras
```

Other utilities:

```
$ sudo apt install avahi-utils ecryptfs-utils
```

16.3. Install ROS

Install ROS on your laptop.

→ The procedure is given in Section 107.1.

16.4. Other suggested software

1) Redshift

This is Flux for Linux. It is an accessibility/lab safety issue: bright screens damage eyes and perturb sleep [4].

Install redshift and run it.

```
$ sudo apt install redshift-gtk
```

Set to "autostart" from the icon.

16.5. Installation of the duckuments system

Optional but very encouraged: install the duckuments system. This will allow you to have a local copy of the documentation and easily submit questions and changes.

→ The procedure is documented in Section 5.4.

16.6. Passwordless `sudo`

Set up passwordless `sudo`.

→ This procedure is described in Section 15.11.

16.7. SSH and Git setup

1) Basic SSH config

Do the basic SSH config.

→ The procedure is documented in Section 97.3.

2) Create key pair for `username`

Next, create a private/public key pair for the user; call it `username@robot name`.

➡ The procedure is documented in Section 97.5.

3) Add `username`'s public key to Github

Add the public key to your Github account.

➡ The procedure is documented in Section 106.3.

If the step is done correctly, this command should succeed:

```
$ ssh -T git@github.com
```

4) Local Git setup

Set up Git locally.

➡ The procedure is described in Section 104.3.

CHAPTER 17

Duckiebot Initialization

Assigned to: Andrea

Requires: An SD card of dimensions at least 32 GB.

Requires: A computer with an internet connection, an SD card reader, and 35 GB of free space.

Requires: An assembled Duckiebot in configuration `D17-C0`. This is the result of Chapter 14.

Result:

- A Duckiebot that is ready to use.

What does it mean “ready to use”?.

17.1. Acquire and burn the image

On the laptop, download the compressed image at this URL:

https://www.dropbox.com/s/1p4am7erdd9e53r/duckiebot-RPI3-AC-aug10.img.xz?dl=1

The size is 2.5 GB.

You can use:

```
$ curl -o duckiebot-RPI3-AC-aug10.img.xz URL above
```

Uncompress the file:

```
$ xz -d -k duckiebot-RPI3-AC-aug10.img.xz
```

This will create a file of 32 GB in size.

To make sure that the image is downloaded correctly, compute its hash using the program `sha256sum`:

```
$ sha256sum duckiebot-RPI3-AC-aug10.img
2ea79b0fc6353361063c89977417fc5e8fde70611e8afa5cbf2d3a166d57e8cf  duckiebot-ac-aug10.img
```

Compare the hash that you obtain with the hash above. If they are different, there was some problem in downloading the image.

Next, burn the image on disk.

→ The procedure of how to burn an image is explained in Section 95.2.

17.2. Turn on the Duckiebot

Put the SD Card in the Duckiebot.

Turn on the Duckiebot by connecting the power cable to the battery.

TODO: Add figure

17.3. Connect the Duckiebot to a network

You can login to the Duckiebot in two ways:

1. Through an Ethernet cable.
2. Through a `duckietown` WiFi network.

In the worst case, you can use an HDMI monitor and a USB keyboard.

1) Option 1: Ethernet cable

Connect the Duckiebot and your laptop to the same network switch.

Allow 30 s - 1 minute for the DHCP to work.

2) Option 2: Duckietown network

The Duckiebot connects automatically to a 2.4 GHz network called "`duckietown`" and password "`quackquack`".

Connect your laptop to the same wireless network.

17.4. Ping the Duckiebot

To test that the Duckiebot is connected, try to ping it.

The hostname of a freshly-installed duckiebot is `duckiebot-not-configured`:

```
$ ping duckiebot-not-configured.local
```

You should see output similar to the following:

```
PING duckiebot-not-configured.local (X.X.X.X): 56 data bytes
64 bytes from X.X.X.X: icmp_seq=0 ttl=64 time=2.164 ms
64 bytes from X.X.X.X: icmp_seq=1 ttl=64 time=2.303 ms
...
```

17.5. SSH to the Duckiebot

Next, try to log in using SSH, with account `ubuntu`:

```
$ ssh ubuntu@duckiebot-not-configured.local
```

The password is `ubuntu`.

By default, the robot boots into Byobu.

Please see Chapter 103 for an introduction to Byobu.

Not sure it's a good idea to boot into Byobu.

17.6. (For D17-C1) Configure the robot-generated network

`D17-0+w` The Duckiebot in configuration `D17-C0+w` can create a WiFi network.

It is a 5 GHz network; this means that you need to have a 5 GHz WiFi adapter in your laptop.

First, make sure that the Edimax is correctly installed. Using `iwconfig`, you should see four interfaces:

```
$ iwconfig
wlxAABBCCDDEEFFGG  unassociated  Nickname:"rtl8822bu"

...

lo        no wireless extensions.

enxb827eb1f81a4  no wireless extensions.

wlan1     IEEE 802.11bgn  ESSID:"duckietown"

...
```

Make note of the name `wlxAABBCCDDEEFFGG`.

Look up the MAC address using the command:

```
$ ifconfig wlxAABBCCDDEEFFGG
wlx74da38c9caa0 Link encap:Ethernet  HWaddr AA:BB:CC:DD:EE:FF:GG
```

Then, edit the connection file

```
/etc/NetworkManager/system-connections/create-5ghz-network
```

Make the following changes:

- Where it says `interface-name=...`, put "`wlxAABBCCDDEEFFGG`".
- Where it says `mac-address=...`, put "`AA:BB:CC:DD:EE:FF:GG`".
- Where it says `ssid=duckiebot-not-configured`, put "`ssid=robot name`".

Reboot.

At this point you should see a new network being created named "`robot name`".

You can connect with the laptop to that network.

If the Raspberry Pi's network interface is connected to the `duckietown` network and to the internet, the Raspberry Pi will act as a bridge to the internet.

17.7. Setting up wireless network configuration

This part should not be necessary anymore

The Duckiebot is configured by default to connect to a wireless network with SSID `duckietown`. If that is not your SSID then you will need to change the configuration.

You can add a new network by editing the file:

```
/etc/wpa_supplicant/wpa_supplicant.conf
```

You will see a block like the following:

```
network={
  ssid="duckietown"
  scan_ssid=1
  psk="quackquack"
  priority=10
}
```

Add a new one with your SSID and password.

This assumes you have a roughly similar wireless network setup - if not then you might need to change some of the other attributes.

17.8. Update the system

Next, we need to update to bring the system up to date.

Use these commands

```
$ sudo apt update
$ sudo apt dist-upgrade
```

17.9. Give a name to the Duckiebot

It is now time to give a name to the Duckiebot.

These are the criteria:

- It should be a simple alphabetic string (no numbers or other characters like "-", "_", etc.) .
- It will always appear lowercase.
- It cannot be a generic name like "`duckiebot`", "`robot`" or similar.

From here on, we will refer to this string as "`robot name`". Every time you see `robot name`, you should substitute the name that you chose.

17.10. Change the hostname

We will put the robot name in configuration files.

> **Note:** Files in `/etc` are only writable by `root`, so you need to use `sudo` to edit them. For example:

```
$ sudo vi filename
```

Edit the file

```
/etc/hostname
```

and put "`robot name`" instead of `duckiebot-not-configured`.

Also edit the file

```
/etc/hosts
```

and put "`robot name`" where `duckiebot-not-configured` appears.

The first two lines of `/etc/hosts` should be:

```
127.0.0.1   localhost
127.0.1.1   robot name
```

Note: there is a command `hostname` that promises to change the hostname. However, the change given by that command does not persist across reboots. You need to edit the files above for the changes to persist.

Note: Never add other hostnames in `/etc/hosts`. It is a tempting fix when DNS does not work, but it will cause other problems subsequently.

Then reboot the Raspberry Pi using the command

```
$ sudo reboot
```

After reboot, log in again, and run the command `hostname` to check that the change has persisted:

```
$ hostname
robot name
```

17.11. Expand your filesystem

If your SD card is larger than the image, you'll want to expand the filesystem on your robot so that you can use all of the space available. Achieve this with:

```
$ sudo raspi-config --expand-rootfs
```

and then reboot

```
$ sudo shutdown -r now
```

once rebooted you can test whether this was successful by doing

```
$ df -lh
```

the output should give you something like:

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/root        29G  7.8G   21G  28% /
devtmpfs        427M     0  427M   0% /dev
tmpfs           432M  316K  431M   1% /dev/shm
tmpfs           432M   12M  420M   3% /run
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           432M     0  432M   0% /sys/fs/cgroup
/dev/mmcblk0p1   63M   21M   43M  34% /boot
tmpfs            87M   24K   87M   1% /run/user/1000
/dev/sda1        29G  5.3G   24G  19% /media/ubuntu/44A7-9E91
```

You should see that the Size of your `/dev/sda1` partition is "close" to the side of your SD card.

17.12. Create your user

You must not use the `ubuntu` user for development. Instead, you need to create a new user.

Choose a user name, which we will refer to as `username`.

To create a new user:

```
$ sudo useradd -m username
```

Make the user an administrator by adding it to the group `sudo`:

```
$ sudo adduser username sudo
```

Make the user a member of the group `input` and `i2c`

```
$ sudo adduser username input
$ sudo adduser username video
$ sudo adduser username i2c
```

Set the shell `bash`:

```
$ sudo chsh -s /bin/bash andrea
```

To set a password, use:

```
$ sudo passwd username
```

At this point, you should be able to login to the new user from the laptop using the password:

```
$ ssh username@robot name
```

Next, you should repeat some steps that we already described.

1) Basic SSH config

Do the basic SSH config.

→ The procedure is documented in Section 97.3.

2) Create key pair for `username`

Next, create a private/public key pair for the user; call it `username@robot name`.

→ The procedure is documented in Section 97.5.

3) Add `username`'s public key to Github

Add the public key to your Github account.

→ The procedure is documented in Section 106.3.

If the step is done correctly, this command should succeed:

```
$ ssh -T git@github.com
```

4) Local Git configuration

- This procedure is in Section 104.3.

5) Set up the laptop-Duckiebot connection

Make sure that you can login passwordlessly to your user from the laptop.

- The procedure is explained in Section 97.6. In this case, we have: `local` = laptop, `local-user` = your local user on the laptop, `remote` = `robot name`, `remote-user` = `username`.

If the step is done correctly, you should be able to login from the laptop to the robot, without typing a password:

```
$ ssh username@robot name
```

6) Some advice on the importance of passwordless access

In general, if you find yourself:

- typing an IP
- typing a password
- typing `ssh` more than once
- using a screen / USB keyboard

it means you should learn more about Linux and networks, and you are setting yourself up for failure.

Yes, you “can do without”, but with an additional 30 seconds of your time. The 30 seconds you are not saving every time are the difference between being productive roboticists and going crazy.

Really, it is impossible to do robotics when you have to think about IPs and passwords...

17.13. Other customizations

If you know what you are doing, you are welcome to install and use additional shells, but please keep Bash as be the default shell. This is important for ROS installation.

For the record, our favorite shell is ZSH with `oh-my-zsh`.

17.14. Hardware check: camera

Check that the camera is connected using this command:

```
$ vcgencmd get_camera
supported=1 detected=1
```

If you see `detected=0`, it means that the hardware connection is not working.

You can test the camera right away using a command-line utility called `raspistill`.

Use the `raspistill` command to capture the file `out.jpg`:

```
$ raspistill -t 1 -o out.jpg
```

Then download `out.jpg` to your computer using `scp` for inspection.

➡ For instructions on how to use `scp`, see Subsection 99.1.1.

1) Troubleshooting

Symptom: `detected=0`

Resolution: If you see `detected=0`, it is likely that the camera is not connected correctly.

If you see an error that starts like this:

```
mmal: Cannot read camera info, keeping the defaults for OV5647
...
mmal: Camera is not detected. Please check carefully the camera module is installed correctly.
```

then, just like it says: "Please check carefully the camera module is installed correctly.".

CHAPTER 18

Software setup and RC remote control

Assigned to: Andrea

Requires: Laptop configured, according to Chapter 16.

Requires: You have configured the Duckiebot. The procedure is documented in Chapter 17.

Requires: You have created a Github account and configured public keys, both for the laptop and for the Duckiebot. The procedure is documented in Chapter 106.

Results:

- You can run the joystick demo.

18.1. Clone the Duckietown repository

Clone the repository in the directory `~/duckietown`:

```
$ git clone git@github.com:duckietown/Software.git ~/duckietown
```

For the above to succeed you should have a Github account already set up.

It should not ask for a password.

1) Troubleshooting

Symptom: It asks for a password.

Resolution: You missed some of the steps described in Chapter 106.

Symptom: Other weird errors.

Resolution: Probably the time is not set up correctly. Use `ntpdate` as above:

```
$ sudo ntpdate -u us.pool.ntp.org
```

18.2. Set up ROS environment on the Duckiebot

All the following commands should be run in the `~/duckietown` directory:

```
$ cd ~/duckietown
```

Now we are ready to make the workspace. First you need to source the baseline ROS environment:

```
$ source /opt/ros/kinetic/setup.bash
```

Then, build the workspace using:

```
$ catkin_make -C catkin_ws/
```

☀ *For more information about `catkin_make`, see Section 107.6.*

> **Note:** there is a known bug, for which it fails the first time on the Raspberry Pi. Try again; it will work.

18.3. Add your vehicle to the scuderia file

Add your vehicle to the scuderia file.

→ See Section 110.2.

18.4. Test that the joystick is detected

Plug the joystick receiver in one of the USB port on the Raspberry Pi.

To make sure that the joystick is detected, run:

```
$ ls /dev/input/
```

and check if there is a device called `js0` on the list.

Check before you continue

Make sure that your user is in the group `input` and `i2c`:

```
$ groups
username sudo input i2c
```

If `input` and `i2c` are not in the list, you missed a step. Ohi ohi! You are not following the instructions carefully!

→ Consult again Section 17.12.

To test whether or not the joystick itself is working properly, run:

```
$ jstest /dev/input/js0
```

Move the joysticks and push the buttons. You should see the data displayed change according to your actions.

18.5. Run the joystick demo

SSH into the Raspberry Pi and run the following from the `duckietown` directory:

```
$ cd ~/duckietown
$ source environment.sh
```

The `environment.sh` setups the ROS environment at the terminal (so you can use commands like `rosrun` and `roslaunch`).

Now make sure the motor shield is connected.

Run the command:

```
$ roslaunch duckietown joystick.launch veh:=robot name
```

If there is no "red" output in the command line then pushing the left joystick knob controls throttle - right controls steering.

This is the expected result of the commands:

left joystick up	forward
left joystick down	backward
right joystick left	turn left (positive yaw)
right joystick right	turn right (negative yaw)

It is possible you will have to unplug and replug the joystick or just push lots of buttons on your joystick until it wakes up. Also make sure that the mode switch on the top of your joystick is set to "X", not "D".

Is all of the above valid with the new joystick?

Close the program using Ctrl-C.

1) Troubleshooting

Symptom: The robot moves weirdly (e.g. forward instead of backward).

Resolution: The cables are not correctly inserted. Please refer to the assembly guide for pictures of the correct connections. Try swapping cables until you obtain the expected behavior.

Resolution: Check that the joystick has the switch set to the position "x".And the mode light should be off.

Symptom: The left joystick does not work.

Resolution: If the green light on the right to the "mode" button is on, click the "mode" button to turn the light off. The "mode" button toggles between left joystick or the cross on the left.

Symptom: The robot does not move at all.

Resolution: The cables are disconnected.

Resolution: The program assumes that the joystick is at `/dev/input/js0`. In doubt, see Section 18.4.

18.6. The proper shutdown procedure for the Raspberry Pi

Generally speaking, you can terminate any `roslaunch` command with Ctrl-C.

To completely shutdown the robot, issue the following command:

```
$ sudo shutdown -h now
```

Then wait 30 seconds.

Warning: If you disconnect the power before shutting down properly using `shutdown`, the system might get corrupted.

Then, disconnect the power cable, at the **battery end**.

Warning: If you disconnect frequently the cable at the Raspberry Pi's end, you might damage the port.

CHAPTER 19

Reading from the camera

Requires: You have configured the Duckiebot. The procedure is documented in Chapter 17.

Requires: You know the basics of ROS (launch files, `roslaunch`, topics, `rostopic`).

TODO: put reference

Results:

- You know that the camera works under ROS.

19.1. Check the camera hardware

It might be useful to do a quick camera hardware check.

➡ The procedure is documented in Section 17.14.

19.2. Create two windows

On the laptop, create two Byobu windows.

➡ A quick reference about Byobu commands is in Chapter 103.

You will use the two windows as follows:

- In the first window, you will launch the nodes that control the camera.
- In the second window, you will launch programs to monitor the data flow.

Note: You could also use multiple *terminals* instead of one terminal with multiple Byobu windows. However, using Byobu is the best practice to learn.

19.3. First window: launch the camera nodes

In the first window, we will launch the nodes that control the camera.

Activate ROS:

```
$ source environment.sh
```

Run the launch file called `camera.launch`:

```
$ roslaunch duckietown camera.launch veh:=robot name
```

At this point, you should see the red LED on the camera light up continuously.

In the terminal you should not see any red message, but only happy messages like the following:

```
...
[INFO] [1502539383.948237]: [/robot name/camera_node] Initialized.
[INFO] [1502539383.951123]: [/robot name/camera_node] Start capturing.
[INFO] [1502539384.040615]: [/robot name/camera_node] Published the first image.
```

☀ *For more information about `roslaunch` and "launch files", see* *Section 107.3.*

19.4. Second window: view published topics

Switch to the second window.

Activate the ROS environment:

```
$ source environment.sh
```

1) List topics

You can see a list of published topics with the command:

```
$ rostopic list
```

☀ *For more information about `rostopic`, see* *Section 107.5.*

You should see the following topics:

```
/robot name/camera_node/camera_info
/robot name/camera_node/image/compressed
/robot name/camera_node/image/raw
/rosout
/rosout_agg
```

2) Show topics frequency

You can use `rostopic hz` to see the statistics about the publishing frequency:

```
$ rostopic hz /robot name/camera_node/image/compressed
```

On a Raspberry Pi 3, you should see a number close to 30 Hz:

```
average rate: 30.016
    min: 0.026s max: 0.045s std dev: 0.00190s window: 841
```

3) Show topics data

You can view the messages in real time with the command `rostopic echo`:

```
$ rostopic echo /robot name/camera_node/image/compressed
```

You should see a large sequence of numbers being printed to your terminal. That's the "image" — as seen by a machine.

If you are Neo, then this already makes sense. If you are not Neo, in Chapter 21, you will learn how to visualize the image stream on the laptop using `rviz`.

use Ctrl-C to stop `rostopic`.

CHAPTER 20
RC control launched remotely

Assigned to: Andrea

Requires: You can run the joystick demo from the Raspberry Pi. The procedure is documented in Chapter 18.

Results:

- You can run the joystick demo from your laptop.

20.1. Two ways to launch a program

ROS nodes can be launched in two ways:

1. “local launch”: log in to the Raspberry Pi using SSH and run the program from there.
2. “remote launch”: run the program directly from a laptop.

Which is better when is a long discussion that will be done later. Here we set up the “remote launch”.

TODO: draw diagrams

20.2. Download and setup `Software` repository on the laptop

As you did on the Duckiebot, you should clone the `Software` repository in the `~/duckietown` directory.

➡ The procedure is documented in Section 18.1.

Then, you should build the repository.

➡ This procedure is documented in Section 18.2.

20.3. Edit the `machines` files on your laptop

You have to edit the `machines` files on your laptop, as you did on the Duckiebot.

➡ The procedure is documented in Section 18.3.

20.4. Start the demo

Now you are ready to launch the joystick demo remotely.

Check before you continue

Make sure that you can login with SSH **without a password**. From the laptop, run:

```
$ ssh username@robot name.local
```

If this doesn't work, you missed some previous steps.

Run this *on the laptop*:

```
$ source environment.sh
$ roslaunch duckietown joystick.launch veh:=robot name
```

You should be able to drive the vehicle with joystick just like the last example. Note that remotely launching nodes from your laptop doesn't mean that the nodes are running on your laptop. They are still running on the Raspberry Pi in this case.

☀ *For more information about `roslaunch`, see Section 107.3.*

20.5. Watch the program output using `rqt_console`

Also, you might have notice that the terminal where you launch the launch file is not printing all the printouts like the previous example. This is one of the limitation of remote launch.

Don't worry though, we can still see the printouts using `rqt_console`.

On the laptop, open a new terminal window, and run:

```
$ export ROS_MASTER_URI=http://robot name.local:11311/
$ rqt_console
```

AC: I could not see any messages in `rqt_console` - not sure what is wrong.

You should see a nice interface listing all the printouts in real time, completed with filters that can help you find that message you are looking for in a sea of messages.

You can use Ctrl-C at the terminal where `roslaunch` was executed to stop all the nodes launched by the launch file.

☀ *For more information about `rqt_console`, see Section 107.2.*

20.6. Troubleshooting

Symptom: `roslaunch` fails with an error similar to the following:

```
remote[robot name.local-0]: failed to launch on robot name:

Unable to establish ssh connection to [username@robot name.local:22]:
Server u'robot name.local' not found in known_hosts.
```

Resolution: You have not followed the instructions that told you to add the `HostKeyAlgorithms` option. Delete `~/.ssh/known_hosts` and fix your configuration.

➡ The procedure is documented in Section 97.3.

CHAPTER 21
RC+camera remotely

Assigned to: Andrea

Requires: You can run the joystick demo remotely. The procedure is documented in Chapter 20.

Requires: You can read the camera data from ROS. The procedure is documented in Chapter 19.

Requires: You know how to get around in Byobu. You can find the Byobu tutorial in Chapter 103.

Results:

- You can run the joystick demo from your laptop and see the camera image on the laptop.

21.1. Assumptions

We are assuming that the joystick demo in Chapter 20 worked.

We are assuming that the procedure in Chapter 19 succeeded.

We also assume that you terminated all instances of `roslaunch` with Ctrl-C, so that currently there is nothing running in any window.

21.2. Terminal setup

On the laptop, this time create **four** Byobu windows.

➡ A quick reference about Byobu commands is in Chapter 103.

You will use the four windows as follows:

- In the first window, you will run the joystick demo, as before.
- In the second window, you will launch the nodes that control the camera.
- In the third window, you will launch programs to monitor the data flow.
- In the fourth window, you will use `rviz` to see the camera image.

TODO: Add figures

21.3. First window: launch the joystick demo

In the first window, launch the joystick remotely using the same procedure in Section 20.4.

```
$ source environment.sh
$ roslaunch duckietown joystick.launch veh:=robot name
```

You should be able to drive the robot with the joystick at this point.

21.4. Second window: launch the camera nodes

In the second window, we will launch the nodes that control the camera.

The launch file is called `camera.launch`:

```
$ source environment.sh
$ roslaunch duckietown camera.launch veh:=robot name
```

You should see the red led on the camera light up.

21.5. Third window: view data flow

Open a third terminal on the laptop.

You can see a list of topics currently on the `ROS_MASTER` with the commands:

```
$ source environment.sh
$ export ROS_MASTER_URI=http://robot name.local:11311/
$ rostopic list
```

You should see the following:

```
/diagnostics
/robot name/camera_node/camera_info
/robot name/camera_node/image/compressed
/robot name/camera_node/image/raw
/robot name/joy
/robot name/wheels_driver_node/wheels_cmd
/rosout
/rosout_agg
```

21.6. Fourth window: visualize the image using `rviz`

Launch `rviz` by using these commands:

```
$ source environment.sh
$ source set_ros_master.sh robot name
$ rviz
```

☀ *For more information about `rviz`, see* *Section 107.4**.*

In the `rviz` interface, click "Add" on the lower left, then the "By topic" tag, then select the "Image" topic by the name

```
/robot name/camera_node/image/compressed
```

Then click "ok". You should be able to see a live stream of the image from the camera.

21.7. Proper shutdown procedure

To stop the nodes: You can stop the node by pressing Ctrl-C on the terminal where `roslaunch` was executed. In this case, you can use Ctrl-C in the terminal where you launched the `camera.launch`.

You should see the red light on the camera turn off in a few seconds.

Note that the `joystick.launch` is still up and running, so you can still drive the vehicle with the joystick.

Chapter 22
Interlude: Ergonomics

Assigned to: Andrea

So far, we have been spelling out all commands for you, to make sure that you understand what is going on.

Now, we will tell you about some shortcuts that you can use to save some time.

Note: in the future you will have to debug problems, and these problems might be harder to understand if you rely blindly on the shortcuts.

Results:

- You will know about some useful shortcuts.

22.1. `set_ros_master.sh`

Instead of using:

```
$ export ROS_MASTER_URI=http://robot name.local:11311/
```

You can use the "`set_ros_master.sh`" script in the repo:

```
$ source set_ros_master.sh robot name
```

Note that you need to use `source`; without that, it will not work.

22.2. SSH aliases

Instead of using

```
$ ssh username@robot name.local
```

You can set up SSH so that you can use:

```
$ ssh my-robot
```

To do this, create a host section in `~/.ssh/config` with the following contents:

```
Host my-robot
    User username
    Hostname robot name.local
```

Here, you can choose any other string in place of "`my-robot`".

Note that you **cannot** do

```
$ ping my-robot
```

You haven't created another hostname, just an alias for SSH.

However, you can use the alias with all the tools that rely on SSH, including `rsync` and `scp`.

Chapter 23
Wheel calibration

> **Assigned to:** Andrea

Chapter 24
Camera calibration

Chapter 25

Taking a log

> **Assigned to:** Andrea

PART 4

Operation manual - Duckietowns

CHAPTER 26
Duckietown parts

Duckietowns are the cities where Duckiebots drive. Here, we provide a link to all bits and pieces that are needed to build a Duckietown, along with their price tag. Note that while the topography of the map is highly customable, we recommend using the components listed below. Before purchasing components for a Duckietown, read Chapter 30 to understand how Duckietowns are built.

In general, keep in mind that:

- The links might expire, or the prices might vary.
- Shipping times and fees vary, and are not included in the prices shown below.
- Substitutions are probably not OK, unless you are OK in writing some software.

> **Requires:** Cost (per m^2): USD ?? + Shipping Fees Requires: Time: ?? days (average shipping time)
>
> Results: A kit of parts ready to be assembled in a Duckietown.
>
> Next Steps: Assemblying a Duckietown.

TODO: Figure out costs

26.1. Bill of materials

TABLE 10. BILL OF MATERIALS FOR DUCKIETOWN

Duckies	USD 17/100 pieces
Floor Mats	USD 37.5/6 pieces (24 sqft)
Duct tape - Red	USD 8.50/roll
Duct tape - White	USD 8.50/roll
Duct tape - Yellow	USD 8/roll
Traffic signs	USD 18.50/13 pieces
Total for **Duckietown**/m^2	USD ??

TODO: Add suggestions for “small”, “medium”, “big” towns as a function of m^2 and supported bots

26.2. Duckies

Duckies (Figure 38) are essential yet non functional.

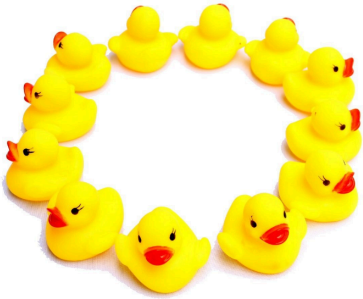

Figure 38. The Duckies

26.3. Floor Mats

The floor mats (Figure 39) are the ground on which the Duckiebots drive.

We choose these mats because they have desirable surface properties, are modular, and have the right size to be street segments. Each square is (~61x61cm) and can connect on every side of other squares. There are 6 mats in each package.

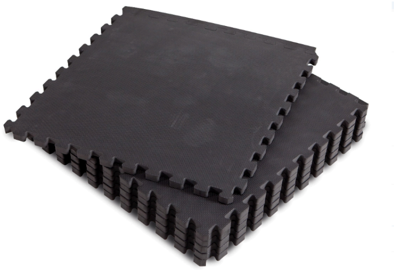

Figure 39. The Floor Mats

Each mat can be a segment of road: straight, a curve, or an intersection (3, or 4 way). To design your Duckietown, see Chapter 30.

26.4. Duck Tape

We use duck (duct) tape of different colors (Figure 40) for defining the roads and their signals. White indicates the road boundaries, yellow determines lane boundaries and red are stop signs.

The white and red tape we use are 2 inches wide, while the yellow one is 1 inch wide.

Figure 40. The Duck Tapes

To verify how much tape you need for each road segment type, see Chapter 30.

26.5. Traffic Signs

Traffic signs (Figure 41) inform Duckiebots on the map of Duckietown, allowing them to make driving decisions.

Figure 41. The Signs

Depending on the chose road topograhy, the number of necessary road signal will vary. To design your Duckietown, see Chapter 30.

CHAPTER 27
Traffic lights Parts

Traffic lights regulate intersections in Duckietown. Here, we provide a link to all bits and pieces that are needed to build a traffic light, along with their price tag. You will need one traffic per either three, or four way intersections. The components listed below meet the appearence specifications described in Chapter 30.

In general, keep in mind that:

- The links might expire, or the prices might vary.
- Shipping times and fees vary, and are not included in the prices shown below.
- Substitutions are probably OK, if you are willing to write some software.

> **Requires:** - Cost: USD ?? + Shipping Fees - Time: ?? days (average shipping time)
>
> Results: - A kit of parts ready to be assembled in a traffic light.
>
> Next Steps: - Assemblying a traffic light.

TODO: Estimate time and costs

27.1. Bill of materials

TABLE 11. BILL OF MATERIALS FOR TRAFFIC LIGHT

Raspberry Pi	USD ??
4 LEDs	USD ??
Wires	USD ??
Total for Traffic Light	USD ??

TODO: Complete table

27.2. Raspberry Pi

(Figure 42) are essential yet non functional.

Figure 42. The placeholder

Chapter 28
Duckietown Assembly

> **Assigned to:** Shiying

Chapter 29
Traffic lights Assembly

> **Assigned to:** Shiying

CHAPTER 30
The Duckietown specification

> Assigned to: Liam?

30.1. Topology

1) Topology constraints

30.2. Signs placement

Part 5

Operation manual - Duckiebot with LEDs

Chapter 31

Acquiring the parts for the Duckiebot `C1`

Upgrading your `C0+wjd` configuration to `C1` starts here, with purchasing the necessary components. We provide a link to all bits and pieces that are needed to build a `C1` Duckiebot, along with their price tag. If you are wondering what is the difference between different Duckiebot configurations, read Chapter 10.

In general, keep in mind that:

- The links might expire, or the prices might vary.
- Shipping times and fees vary, and are not included in the prices shown below.
- Buying the parts for more than one Duckiebot makes each one cheaper than buying only one.
- A few components in this configuration are custom designed, and might be trickier to obtain.

> Requires: - A Duckiebot in `C0+wjd` configuration. - Cost: USD 77 + Bumpers manufacturing solution - Time: 21 Days (LED board manufacturing and shipping time)
>
> Results: - A kit of parts ready to be assembled in a `C1` configuration Duckiebot.
>
> Next Steps: - After receiving these components, you are ready to do some soldering before assembling your `C1` Duckiebot.

31.1. Bill of materials

Table 12. Bill of materials

LEDs	USD 10
LED HAT	USD 28.20 for 3 pieces
Power Cable	USD 7.80
20 Female-Female Jumper Wires (300mm)	USD 8
Male-Male Jumper Wire (150mm)	USD 1.95
PWM/Servo HAT	USD 17.50
Bumpers	TBD (custom made)
40 pin female header	USD 1.50
5 4 pin female header	USD 0.60/piece
2 16 pin male header	USD 0.61/piece
12 pin male header	USD 0.48/piece
3 pin male header	USD 0.10/piece
2 pin female shunt jumper	USD 2/piece
5 200 Ohm resistors	USD 0.10/piece
10 130 Ohm resistors	USD 0.10/piece
Total for `C0+wjd` configuration	USD 212
Total for `C1` components	USD 77 + Bumpers
Total for `C1` configuration	USD 299+Bumpers

TODO: add links to Bumpers: (a) bumper design files; (b) one-click purchasing option (?)

31.2. LEDs

The Duckiebot is equipped with 5 RGB LEDs (Figure 43). LEDs can be used to signal to other Duckiebots, or just make *fancy* patterns.

The pack of LEDs linked in the table above holds 10 LEDs, enough for two Duckiebots.

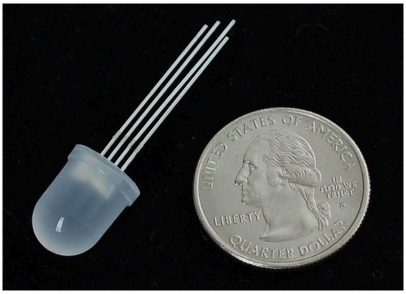

Figure 43. The RGB LEDs

1) LED HAT

The LED HAT (Figure 44) provides an interface for our RGB LEDs and the computational stack. This board is a daughterboard for the Adafruit 16-Channel PWM/Servo HAT, and enables connection with additional gadgets such as ADS1015 12 Bit 4 Channel ADC, Monochrome 128x32 I2C OLED graphic display, and Adafruit 9-DOF IMU Breakout - L3GD20H+LSM303. This item will require soldering.

This board is custom degined and can only be ordered in minimum runs of 3 pieces. The price scales down quickly with quantity, and lead times may be significant, so it is better to buy these boards in bulk.

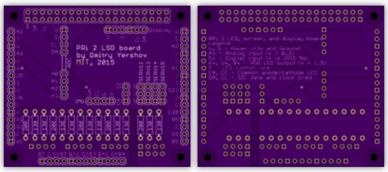

Figure 44. The LED HAT

2) PWM/Servo HAT

The PWM/Servo HAT (Figure 45) mates to the LED HAT and provides the signals to control the LEDs, without taking computational resources away from the Rasperry Pi itself. This item will require soldering.

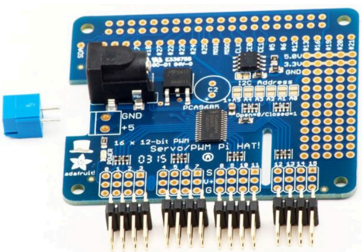

Figure 45. The PWM-Servo HAT

3) Power Cable

To power the PWM/Servo HAT from the battery, we use a short (30cm) angled male USB-A to 5.5/2.1mm DC power jack cable (Figure 46).

Figure 46. The 30cm angled USB to 5.5/2.1mm power jack cable.

4) Male-Male Jumper Wires

The Duckiebot needs one male-male jumper wire (Figure 47) to power the DC Stepper Motor HAT from the PWM/Servo HAT.

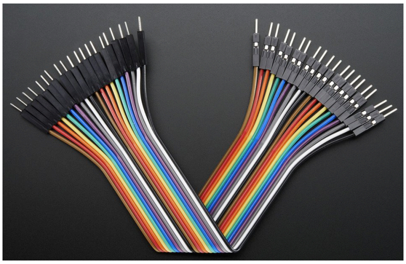

Figure 47. Premier Male-Male Jumper Wires

5) Female-Female Jumper Wires

20 Female-Female Jumper Wires (Figure 48) are necessary to connect 5 LEDs to the LED HAT.

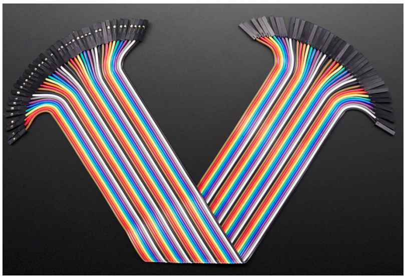

Figure 48. Premier Female-Female Jumper Wires

31.3. Bumpers

These bumpers are designed to keep the LEDs in place and are therefore used only in configuration `C1`. They are custom designed parts, so they must be produced and cannot be bought. We used laser cutting facilities. Our design files are available [here].

TODO: add links to .sldprt files once confirmed final version

Figure 49. The Bumpers

31.4. Headers, resistors and jumper

Upgrading `C0+wjd` to `C1` requires several electrical bits: 5 of 4 pin female header, 2 of 16 pin male headers, 1 of 12 pin male header, 1 of 3 pin male header, 1 of 2 pin female shunt jumper, 5 of 200 Ohm resistors and finally 10 of 130 Ohm resistors.

These items require soldering.

Figure 50. The Headers

Figure 51. The Resistors

TODO: Missing figures.

CHAPTER 32
Soldering boards for `C1`

Assigned to: Shiying

Resources necessary:

Requires: - Duckiebot `C1` parts. The acquisition process is explained in Chapter 31. The configurations are described in Chapter 10.

Requires: - Time: ??? minutes

Results:

- ...

TODO: finish above

Chapter 33
Assembling the Duckiebot C1

> Assigned to: Shiying

> **Requires:** Duckiebot `C1` parts. The acquisition process is explained in Chapter 31.
>
> **Requires:** Soldering `C1` parts. The soldering process is explained in Chapter 32.
>
> **Requires:** Time: about ??? minutes.
>
> **TODO:** estimate time.
>
> Results:
>
> - An assembled Duckiebot in configuration `C1`.

Shiying: here will be the instruction about assembling the Duckiebot.

Chapter 34
`C1` (LEDs) setup

> **Assigned to:** Andrea

Part 6
Theory chapters

These are the theory chapters.

CHAPTER 35

Chapter template

> **Assigned to:** Jacopo

Theory chapters benefit from a standardized exposition. Here, we define the template for these chapters.

TODO: Define new classes for:

- 'required-reading' (must read - red color),
- 'suggested-reading' (best if read - orange)
- 'additional-reading' (for curious users - green)
- 'example' (pop-up box in soothing color (blue?) similar to 'check': stop and think about this for a second). Keep in mind that: (a) color blind people might be confused by red/green. Use additional visual cue (thick/dashed/dotted boundary boxes?); (b) book could be printed in b/w only.

35.1. Example Title: PID control

Start with a brief introduction of the discussed topic, describing its place in the bigger picture, justifying the reading constraints/guidelines below. Write it as if the reader knew the relevant terminology. For example:

PID control is the simplest approach to making a system behave in a desired way rather than how it would naturally behave. It is simple because the measured output is directly feedbacked, as opposed to, e.g., the system's states. The control signal is obtained as a weighted sum of the tracking error (*P*roportional term), its integral over time (*I*ntegrative term) and its instantaneous derivative (*D*erivative term), from which the appellative of PID control. The tracking error is defined as the instantaneous difference between a reference and a measured system output.

Check before you continue

Required Reading: Insert here a list of topics and suggested resources related to *necessary* knowledge in order to understand the content presented. Example:

If you are not familiar with the terminology above (system, plant, output, reference, ...), you must read: autonomy overview

If you are not familiar with how to obtain a system, you must read: - basic kinematics - basic dynamics. - linear algebra - State space representations - Linear Time Invariant Systems - ...

Check before you continue

Suggested Reading: Insert here a list of topics and suggested resources related to *recommended* knowledge in order to better understand the content presented. Example:

If you want to know more about the subtleties of PID control, you can read the following:

- Definitions of Stability, Performances and Robustness: reference-7, ...
- observability/detectability and controllability/reachability: reference-1, reference-2, ...
- Discrete time PID: reference-4,
- Bode diagrams: reference-5, ...
- Nyquist plots: reference-6, ...
- [...]

35.2. Problem Definition

In this section we crisply define the problem object of this chapter. It serves as a very brief recap of exactly what is needed from previous atoms as well. E.g.

Let:

$$\begin{aligned} \dot{\boldsymbol{x}}_t &= \boldsymbol{A}\boldsymbol{x}_t + \boldsymbol{B}\boldsymbol{u}_t \\ \boldsymbol{y} &= \boldsymbol{C}\boldsymbol{x}_t + \boldsymbol{D}\boldsymbol{u}_t \end{aligned} \tag{1}$$

be the LTI model of the Duckiebot's plant, with $\boldsymbol{x} \in \mathcal{X}$, $\boldsymbol{y} \in \mathbb{R}^p$ and $\boldsymbol{u} \in \mathbb{R}^m$. We recall (Duckiebot Modeling) that:

$$\begin{aligned} A &= \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{bmatrix} \\ B &= \begin{bmatrix} b_1 & \dots & b_m \end{bmatrix}^T \\ C &= \begin{bmatrix} c_1 & \dots & c_p \end{bmatrix} \\ D &= 0. \end{aligned}$$

[...]

TODO: fix uncentered dot in $\dot{\boldsymbol{x}}_t$

as shown in (Figure 52).

Figure 52. A classical block diagram for PID control. We like to use a lot of clear figures in the Duckiebook.

35.3. Introduced Notions

1) Section 1: title-1 (e.g.: Definitions)

- Definition 1: A reference signal $\tilde{y}_t \in \mathcal{L}_2(\mathcal{T})$ is ...

> **Check before you continue**
> Insert 'random' checks to keep the reader's attention up: if you can't be woken up in the middle of the night and rememeber the definition of $\mathcal{L}_2(\cdot)$, read: another math text

- Definition 2: [...]

2) Section 2: title-2 (e.g.: Output feedback)

Now that we know what we're talking about, lets get in the meat of the problem. Here is what is happening:

[...math, math, oh math...]

3) Section 3: title-3 (e.g.: Tuning the controller)

Introduce the 'synthesys through attempts' methodology (a.k.a. tweak until death)

4) Section 4: title-4 (e.g.: Performance Metrics)

How do we know if the PID controller designed above is doing well? We need to define some performance metrics first:

Overshoot, Module at resonance, Settling Time, Rising Time

...

> **Check before you continue**
> This is a TBD 'example' class application:
>
> For example, when a Duckiebot 'overshoots', it means that [...] and the following will happen [...].

5) Section N: title-N (e.g.: Saving the world with PID)

And this is how to save the world with PID...

35.4. Examples

This section serves as a collection of theoretical and practical examples that can clarify part or all of the above.

1) Theoretical Examples

More academic examples

T-Example 1:

Immagine a spring-mass-damper system...

T-Example M:

[...]

2) Implementation Examples

More Duckiebot related examples

I-Example 1:

I-Example M:

[...]

TODO: It might make sense to decouple the examples from the the theory (as we are doing for the exercises), as an example may be explanatory for different theory chapters.

35.5. Pointers to Exercises

Here we just add references to the suggested exercises, defined in the appropriate exercise chapters.

35.6. Conclusions

- What did we do? (recap)
- What did we find? (analysis)
- Why is it useful? (synthesis)
- Final Conclusions (what have we learned)

35.7. Next Steps

Strong of this new knowledge (what have we learned), we can now [...].

Check before you continue

Further Reading: insert here reference resources for the interested reader:

- learn all there is to know about PID: Isidori-1
- become a linear algebra master: Matrix cookbook
- [...]

35.8. References

The external references mentioned in this chapter should be listed here.

[1] Crazy math paper with lots of arguable math, *W. L. Mathematics*, Nasty Journal, vol. 1, pg. 1-99, 2001.

[2] Isidori-1, ...

[3] Matrix cookbook

TODO: How to implement scalable difficulty? Suggestion: lets start from the graduate level that we need. We will then create seperate files for the ‘undergrad’ and ‘high-school’ versions, simplfying the ‘graduate’ level files.

TODO: auto-compilation of references section

TODO: add “next” in top right corner of every page

TODO: add “click to enlarge pic” functionality

Chapter 36
Symbols and conventions

Assigned to: Andrea

36.1. Conventions

If x is a function of time, use x_t rather than $x(t)$.

✖ Consider the function $x(t)$.

✔ Consider the function x_t.

36.2. Table of symbols

Here are some useful symbols.

Table 13. Spaces

command	result	
`\SOthree`	$\mathbf{SO}(3)$	Rotation matrices
`\SEthree`	$\mathbf{SE}(3)$	Euclidean group
`\SEtwo`	$\mathbf{SE}(2)$	Euclidean group
`\setwo`	$\mathbf{se}(2)$	Euclidean group algebra

States and poses:

Table 14. Poses and states

command	result	
`\pose`	$\boldsymbol{q}_t \in \mathbf{SE}(2)$	Pose of the robot in the plane
`\state_t \in \statesp`	$\boldsymbol{x}_t \in \mathcal{X}$	System state (includes the pose, and everything else)

Chapter 37
Linear algebra

> **Assigned to:** Jacopo

Chapter 38
Probability basics

> Assigned to: Liam?

Chapter 39
Dynamics

> **Assigned to:** Jacopo

TODO: this is a repetition of Chapter 47.

Chapter 40
Coordinate systems

Assigned to: Falcon

TODO: Provide a basic description of coordinate systems. The description of general coordinate systems should be brief, with the majority of the text focusing on Cartesian coordinate systems (2D and 3D).

CHAPTER 41
Autonomy overview

> Assigned to: Liam

41.1. Perception, planning, control

Chapter 42
Autonomy architectures

> **Assigned to:** Andrea

42.1. Contracts

```
API: Types, messages
Latency, frequency
computation
semantics
reliability / probability
```

CHAPTER 43
Representations

> Assigned to: Matt

Check before you continue

Required Reading: The following assumes a working familiarity with 2D and 3D Cartesian coordinate systems. If you are not familiar with Cartesian coordinate systems, please read the chapter on coordinate systems.

Discuss:

- Introduction to the notion of *state* as a sufficient statistic that represents the agent (robot) and environment.
- Define notion of *static* and *dynamic* states.
- Provide examples of robot and environment states.

Notes:

- The state should not express information that is not necessary for the task that the robot is performing.

43.1. Preliminaries

Some/all of the following could be simplified or omitted and instead refer readers to reference material.

Discuss basics associated with

- Coordinate systems (moved to preliminaries: coordinate systems)
- Reference frames
- Transformations

43.2. Robot Representations

Define the notion of:

- *pose* for mobile robots;
- *configuration* for manipulators
- robot and joint velocities

Discuss specific robot state representation for Duckietown.

43.3. Environment Representations

Discuss:

- Difference between topological and metric environment representations;
- Details of topological representation;
- Common metric representations, notably feature-based maps and gridmaps;

1) Duckietown Environment Representation

Discuss specific environment representation for Duckietown.

Chapter 44
Software architectures and middlewares

> **Assigned to:** Andrea

Chapter 45
Modern signal processing

> Assigned to: Andrea

Chapter 46
Basic Kinematics

> **Assigned to:** Jacopo

Chapter 47
Basic Dynamics

> **Assigned to:** Jacopo

Chapter 48
Odometry Calibration

> **Assigned to:** Jacopo

Chapter 49
Computer vision basics

> Assigned to: Matt

Chapter 50
Illumination invariance

> **Assigned to:** Matt

Chapter 51
Line Detection

> Assigned to: Matt

Chapter 52
Feature extraction

> Assigned to: Matt

Chapter 53
Place recognition

> Assigned to: Matt

Chapter 54
Filtering 1

> **Assigned to:** Liam

Chapter 55
Filtering 2

> **Assigned to:** Liam

CHAPTER 56
Mission planning

> Assigned to: ETH

CHAPTER 57
Planning in discrete domains

> Assigned to: ETH

Chapter 58
Motion planning

> Assigned to: ETH

Chapter 59
RRT

> Assigned to: ETH

Chapter 60
Feedback control

> **Assigned to:** Jacopo

Chapter 61
PID Control

> **Assigned to:** Jacopo

Chapter 62
MPC Control

> **Assigned to:** Jacopo

Chapter 63
Object detection

> **Assigned to:** Nick and David

Chapter 64
Object classification

> **Assigned to:** Nick and David

Chapter 65
Object tracking

> **Assigned to:** Nick and David

Chapter 66

Reacting to obstacles

> **Assigned to:** Jacopo

Chapter 67
Semantic segmentation

> **Assigned to:** Nick and David

Chapter 68
Text recognition

> Assigned to: Nick

Chapter 69

SLAM - Problem formulation

> **Assigned to:** Liam

Chapter 70
SLAM - Broad categories

> **Assigned to:** Liam

Chapter 71
VINS

> **Assigned to:** Liam

CHAPTER 72

Advanced place recognition

> Assigned to: Liam

Chapter 73
Fleet level planning (placeholder)

> Assigned to: ETH

Chapter 74
Fleet level planning (placeholder)

> Assigned to: ETH

CHAPTER 75
Bibliography

[1] Jacopo Tani, Liam Paull, Maria Zuber, Daniela Rus, Jonathan How, John Leonard, and Andrea Censi. **Duckietown: an innovative way to teach autonomy.** In *EduRobotics 2016*. Athens, Greece, December 2016. pdf

[2] Liam Paull, Jacopo Tani, Heejin Ahn, Javier Alonso-Mora, Luca Carlone, Michal Cap, Yu Fan Chen, Changhyun Choi, Jeff Dusek, Daniel Hoehener, Shih-Yuan Liu, Michael Novitzky, Igor Franzoni Okuyama, Jason Pazis, Guy Rosman, Valerio Varricchio, Hsueh-Cheng Wang, Dmitry Yershov, Hang Zhao, Michael Benjamin, Christopher Carr, Maria Zuber, Sertac Karaman, Emilio Frazzoli, Domitilla Del Vecchio, Daniela Rus, Jonathan How, John Leonard, and Andrea Censi. **Duckietown: an open, inexpensive and flexible platform for autonomy education and research.** In *IEEE International Conference on Robotics and Automation (ICRA)*. Singapore, May 2017. pdf

[3] Bruno Siciliano and Oussama Khatib. *Springer Handbook of Robotics*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[4] Tosini, G., Ferguson, I., Tsubota, K. *Effects of blue light on the circadian system and eye physiology*. Molecular Vision, 22, 61–72, 2016 (online).

Part 7
Exercises

These are the exercises.

CHAPTER 76
ROS Exercises

Progression of ROS skills:

76.1. Parameters

- Reading parameters
- Dynamic modification of parameters

76.2. Running from a log

- Running from a log

76.3. Unit tests

- Unit tests
- Integration with ROS tests

76.4. Analytics

- Measure the latency and frequency of the node
- Measure the latency of another node

76.5. Visualization

- Making a plot displayed using topic
- Using ROS parameters

Chapter 77

Line detection

Chapter 78

Data processing

Chapter 79

Git and conventions

PART 8
Software reference

This part describes things that you should know about UNIX/Linux environments.

Documentation writers: please make sure that every command used has a section in these chapters.

Chapter 80
Ubuntu packaging with APT

80.1. `apt install`

TODO: to write

80.2. `apt update`

TODO: to write

1) `apt dist-upgrade`

TODO: hold back packages

80.3. `apt-key`

TODO: to write

80.4. `apt-mark`

TODO: to write

80.5. `add-apt-repository`

TODO: to write

80.6. `wajig`

TODO: to write

80.7. `dpigs`

TODO: to write

Chapter 81
GNU/Linux general notions

> Assigned to: Andrea

81.1. Background reading

- UNIX
- Linux
- free software; open source software.

Chapter 82
Every day Linux

82.1. `cd`

TODO: to write

82.2. `sudo`

TODO: to write

82.3. `ls`

TODO: to write

82.4. `cp`

TODO: to write

82.5. `mkdir`

TODO: to write

82.6. `touch`

TODO: to write

82.7. `reboot`

TODO: to write

82.8. `shutdown`

TODO: to write

82.9. `rm`

TODO: to write

Chapter 83
Users

83.1. `passwd`

TODO: to write

CHAPTER 84

UNIX tools

84.1. `cat`

TODO: to write

84.2. `tee`

TODO: to write

84.3. `truncate`

TODO: to write

Chapter 85
Linux disks and files

85.1. `fdisk`

TODO: to write

85.2. `mount`

TODO: to write

85.3. `umount`

TODO: to write

85.4. `losetup`

TODO: to write

85.5. `gparted`

TODO: to write

85.6. `dd`

TODO: to write

85.7. `sync`

TODO: to write

85.8. `df`

TODO: to write

Chapter 86
Other administration commands

86.1. `visudo`

TODO: to write

86.2. `update-alternatives`

TODO: to write

86.3. `udevadm`

TODO: to write

86.4. `systemctl`

TODO: to write

Chapter 87
Make

87.1. `make`

TODO: to write

Chapter 88
Python-related tools

88.1. `virtualenv`

TODO: to write

88.2. `pip`

TODO: to write

Chapter 89
Raspberry-PI commands

89.1. `raspi-config`

TODO: to write

89.2. `vcgencmd`

TODO: to write

89.3. `raspistill`

TODO: to write

89.4. `jstest`

TODO: to write

89.5. `swapon`

TODO: to write

89.6. `mkswap`

TODO: to write

Chapter 90
Users and permissions

90.1. `chmod`

TODO: to write

90.2. `groups`

TODO: to write

90.3. `adduser`

TODO: to write

90.4. `useradd`

TODO: to write

Chapter 91
Downloading

91.1. `curl`

TODO: to write

91.2. `wget`

TODO: to write

91.3. `sha256sum`

TODO: to write

91.4. `xz`

TODO: to write

Chapter 92
Shells and environments

92.1. `source`

TODO: to write

92.2. `which`

TODO: to write

92.3. `export`

TODO: to write

Chapter 93
Other misc commands

93.1. `pgrep`

TODO: to write

93.2. `npm`

TODO: to write

93.3. `nodejs`

TODO: to write

93.4. `ntpdate`

TODO: to write

93.5. `chsh`

TODO: to write

93.6. `echo`

TODO: to write

93.7. `sh`

TODO: to write

93.8. `fc-cache`

TODO: to write

Chapter 94
Linux resources usage

94.1. Measuring CPU usage using `htop`

You can use `htop` to monitor CPU usage.

```
$ sudo apt install htop
```

TODO: to write

94.2. Measuring I/O usage using `iotop`

Install using:

```
$ sudo apt install iotop
```

TODO: to write

94.3. How fast is the SD card?

➡ Section 95.1.

Chapter 95
SD Cards tools

95.1. Testing SD Card and disk speed

Test SD Card (or any disk) speed using the following commands, which write to a file called `filename`.

```
$ dd if=/dev/zero of=filename bs=500K count=1024
$ sync
$ echo 3 | sudo tee /proc/sys/vm/drop_caches
$ dd if=filename of=/dev/null bs=500K count=1024
$ rm filename
```

Note the `sync` and the `echo` command are very important.

Example results:

```
524288000 bytes (524 MB, 500 MiB) copied, 30.2087 s, 17.4 MB/s
524288000 bytes (524 MB, 500 MiB) copied, 23.3568 s, 22.4 MB/s
```

That is write 17.4 MB/s, read 22 MB/s.

95.2. How to burn an image to an SD card

> Requires:
>
> - A blank SD card.
> - An image file to burn.
> - An Ubuntu computer with an SD reader.
>
> Results:
>
> - A burned image.

1) Finding your device name for the SD card

First, find out what is the device name for the SD card.

Insert the SD Card in the slot.

Run the command:

```
$ sudo fdisk -l
```

Find your device name, by looking at the sizes.

For example, the output might contain:

```
Disk /dev/mmcblk0: 14.9 GiB, 15931539456 bytes, 31116288 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

In this case, the device is `/dev/mmcblk0`. That will be the `device` in the next commands.

You may see `/dev/mmcblk0pX` or a couple of similar entries for each partition on the card, where `X` is the partition number. If you don't see anything like that, take out the SD card and run the command again and see what disappeared.

2) Unmount partitions

Before proceeding, unmount all partitions.

Run `df -h`. If there are partitions like `/dev/mmcblk0pn`, then unmount each of them. For example:

```
$ sudo umount /dev/mmcblk0p1
$ sudo umount /dev/mmcblk0p2
```

3) Burn the image

Now that you know that the device is `device`, you can burn the image to disk.

Let the image file be `image file`.

Burn the image using the command `dd`:

```
$ sudo dd of=device if=image file status=progress bs=4M
```

> Note: Use the name of the device, without partitions. i.e., `/dev/mmcblk0`, not `/dev/mmcblk0pX`.

95.3. How to shrink an image

> Requires:
>
> - An image file to burn.
> - An Ubuntu computer.
>
> Results:
>
> - A shrunk image.

> Note: Majority of content taken from here

We are going to use the tool `gparted` so make sure it's installed

```
$ sudo apt install gparted
```

Let the image file be `image file`. Run the command:

```
$ sudo fdisk -l image file
```

It should give you something like:

```
Device                        Boot  Start      End   Sectors  Size Id Type
duckiebot-RPI3-LP-aug15.img1         2048   131071    129024   63M  c W95 FAT32 (LBA)
duckiebot-RPI3-LP-aug15.img2       131072 21219327  21088256 10.1G 83 Linux
```

Take note of the start of the Linux partition (in our case 131072), let's call it `start`. Now we are going to mount the Linux partition from the image:

```
$ sudo losetup /dev/loop0 imagename.img -o $((start*512))
```

and then run `gparted`:

```
$ sudo gparted /dev/loop0
```

In `gparted` click on the partition and click “Resize” under the “Partition” menu. Resize drag the arrow or enter a size that is equal to the minimum size plus 20MB

Note: This didn’t work well for me - I had to add much more than 20MB for it to work.

Click the “Apply” check mark. *Before* closing the final screen click through the arrows in the dialogue box to find a line such a “`resize2fs -p /dev/loop0 1410048K`”. Take note of the new size of your partition. Let’s call it `new size`.

Now remove the loopback on the second partition and setup a loopback on the whole image and run `fdisk`:

```
$ sudo losetup -d /dev/loop0
$ sudo losetup /dev/loop0 image file
$ sudo fdisk /dev/loop0

Command (m for help): enter d
Partition number (1,2, default 2): enter 2
Command (m for help): enter n
Partition type
p   primary (1 primary, 0 extended, 3 free)
e   extended (container for logical partitions)
Select (default p): enter p
Partition number (2-4, default 2): enter 2
First sector (131072-62521343, default 131072): start
Last sector, +sectors or +size{K,M,G,T,P} (131072-62521343, default 62521343): +new sizeK
```

Note: on the last line include the `+` and the `K` as part of the size.

```
Created a new partition 2 of type 'Linux' and of size 10.1 GiB.

Command (m for help): enter w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Re-reading the partition table failed.: Invalid argument

The kernel still uses the old table. The new table will be used at the next reboot or after
you run partprobe(8) or kpartx(8).
```

Disregard the final error.

You partition has now been resized and the partition table has been updated. Now we will remove the loopback and then truncate the end of the image file:

```
$ fdisk -l /dev/loop0
```

```
Device        Boot  Start      End  Sectors  Size Id Type
/dev/loop0p1         2048   131071   129024   63M  c W95 FAT32 (LBA)
/dev/loop0p2       131072 21219327 21088256 10.1G 83 Linux
```

Note down the end of the second partition (in this case 21219327). Call this `end`.

```
$ sudo losetup -d /dev/loop0
$ sudo truncate -s $(((end+1)*512)) image file
```

You now have a shrunken image file. A further idea is to compress it:

```
$ xz image file
```

Chapter 96
Networking tools

> Assigned to: Andrea

> Preliminary reading:
>
> - Basics of networking, including
>   - what are IP addresses
>   - what are subnets
>   - how DNS works
>   - how `.local` names work
>   - ...
>
> ➡ (ref to find).

TODO: to write

Make sure that you know:

96.1. `hostname`

TODO: to write

96.2. Visualizing information about the network

1) `ping`: are you there?

TODO: to write

2) `ifconfig`

TODO: to write

```
$ ifconfig
```

Chapter 97
Accessing computers using SSH

> Assigned to: Andrea

97.1. Background reading

TODO: to write

- Encryption
- Public key authentication

97.2. Installation of SSH

This installs the client:

```
$ sudo apt install ssh
```

This installs the server:

TODO: to write

This enables the server:

TODO: to write

97.3. Local configuration

The SSH configuration as a client is in the file

```
~/.ssh/config
```

Create the directory with the right permissions:

```
$ mkdir ~/.ssh
$ chmod 0700 ~/.ssh
```

Then add the following lines:

```
HostKeyAlgorithms ssh-rsa
```

The reason is that Paramiko, used by `roslaunch`, does not support the ECSDA keys.

97.4. How to login with SSH and a password

To log in to a remote computer `remote` with user `remote-user`, use:

```
$ ssh remote-user@remote
```

1) Troubleshooting

Symptom: “Offending key error”.

If you get something like this:

```
Warning: the ECDSA host key for ... differs from the key for the IP address '... '

Offending key for IP in /home/user/.ssh/known_hosts:line
```

then remove line `line` in `~/.ssh/known_hosts`.

97.5. Creating an SSH keypair

This is a step that you will repeat twice: once on the Duckiebot, and once on your laptop.

The program will prompt you for the filename on which to save the file.

Use the convention

```
/home/username/.ssh/username@host name
/home/username/.ssh/username@host name.pub
```

where:

- `username` is the current user name that you are using (`ubuntu` or your chosen one);
- `host name` is the name of the host (the Duckiebot or laptop);

An SSH key can be generated with the command:

```
$ ssh-keygen -h
```

The session output will look something like this:

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/username/.ssh/id_rsa):
```

At this point, tell it to choose this file:

```
/home/username/.ssh/username@host name
```

Then:

```
Enter passphrase (empty for no passphrase):
```

Press enter; you want an empty passphrase.

```
Enter same passphrase again:
```

Press enter.

```
Your identification has been saved in /home/username/.ssh/username@host name
Your public key has been saved in /home/username/.ssh/username@host name.pub
The key fingerprint is:
XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX username@host name
The key's randomart image is:
+--[ RSA 2048]----+
|            .    |
|      o   o  .   |
|     o = o   . o |
|      B . .   * o|
|       S o    0  |
|        o o  . E|
|         o o  o  |
|          o  +   |
|            .. . |
+-----------------+
```

Note that the program created two files.

The file that contains the private key is

```
/home/username/.ssh/username@host name
```

The file that contains the public key has extension `.pub`:

```
/home/username/.ssh/username@host name.pub
```

Next, tell SSH that you want to use this key.

Make sure that the file `~/.ssh/config` exists:

```
$ touch ~/.ssh/config
```

Add a line containing

```
IdentityFile PRIVATE_KEY_FILE
```

(using the filename for the private key).

Check that the config file is correct:

```
$ cat ~/.ssh/config
...
IdentityFile PRIVATE_KEY_FILE
...
```

97.6. How to login without a password

> Assumptions:
>
> - You have two computers, called "`local`" and "`remote`", with users "`local-user`" and "`remote-user`".
> - The two computers are on the same network.

- You have created a keypair for `local-user` on `local`.
  - ➡ This procedure is described in Section 97.5.

Results:

- From the `local` computer, `local-user` will be able to log in to `remote` computer without a password.

First, connect the two computers to the same network, and make sure that you can ping `remote` from `local`:

```
local $ ping remote.local
```

Do not continue if you cannot do this successfully.

If you have created a keypair for `local-user`, you will have a public key in this file on the `local` computer:

```
/home/local-user/.ssh/local-user@local.pub
```

This file is in the form:

```
ssh-rsa long list of letters and numbers local-user@local
```

You will have to copy the contents of this file on the `remote` computer, to tell it that this key is authorized.

On the `remote` computer, edit or create the file:

```
/home/remote-user/.ssh/authorized_keys
```

and add the entire line as above containing the public key.

Now, from the `local` computer, try to log in into the `remote` one:

```
local $ ssh remote-user@remote
```

This should succeed, and you should not be asked for a password.

97.7. Fixing SSH Permissions

Sometimes, SSH does not work because you have the wrong permissions on some files.

In doubt, these lines fix the permissions for your `.ssh` directory.

```
$ chmod 0700 ~/.ssh
$ chmod 0700 ~/.ssh/*
```

97.8. ssh-keygen

TODO: to write

Chapter 98

Wireless networking in Linux

98.1. `iwconfig`

TODO: to write

98.2. `iwlist`

1) Getting a list of WiFi networks

What wireless networks do I have around?

```
$ sudo iwlist interface scan | grep SSID
```

2) Do I have 5 GHz?

Does the interface support 5 GHz channels?

```
$ sudo iwlist interface freq
```

Example output:

```
wlx74da38c9caa0  20 channels in total; available frequencies :
  Channel 01 : 2.412 GHz
  Channel 02 : 2.417 GHz
  Channel 03 : 2.422 GHz
  Channel 04 : 2.427 GHz
  Channel 05 : 2.432 GHz
  Channel 06 : 2.437 GHz
  Channel 07 : 2.442 GHz
  Channel 08 : 2.447 GHz
  Channel 09 : 2.452 GHz
  Channel 10 : 2.457 GHz
  Channel 11 : 2.462 GHz
  Channel 36 : 5.18 GHz
  Channel 40 : 5.2 GHz
  Channel 44 : 5.22 GHz
  Channel 48 : 5.24 GHz
  Channel 149 : 5.745 GHz
  Channel 153 : 5.765 GHz
  Channel 157 : 5.785 GHz
  Channel 161 : 5.805 GHz
  Channel 165 : 5.825 GHz
  Current Frequency:2.437 GHz (Channel 6)
```

Note that in this example only *some* 5Ghz channels are supported (36, 40, 44, 48, 149, 153, 157, 161, 165); for example, channel 38, 42, 50 are not supported. This means that you need to set up the router not to use those channels.

Chapter 99
Moving files between computers

99.1. SCP

TODO: to write

1) Download a file with SCP

TODO: to write

99.2. RSync

TODO: to write

Chapter 100
VIM

> Assigned to: Andrea

To do quick changes to files, especially when logged remotely, we suggest you use the VI editor, or more precisely, VIM (“VI iMproved”).

100.1. External documentation

- A VIM tutorial.

100.2. Installation

Install like this:

```
$ sudo apt install vim
```

100.3. `vi`

TODO: to write

100.4. Suggested configuration

Suggested `~/.vimrc`:

```
syntax on
set number
filetype plugin indent on
highlight Comment ctermfg=Gray
autocmd FileType python set complete isk+=.,(
```

100.5. Visual mode

TODO: to write

100.6. Indenting using VIM

Use the `>` command to indent.

To indent 5 lines, use `5` `>` `>`.

To mark a block of lines and indent it, use `V`.

For example, use `V` `J` `J` `>` to indent 3 lines.

Use `<` to dedent.

Chapter 101
Atom

TODO: to write

Chapter 102
Eclipse

TODO: to write

102.1. Installing LiClipse

TODO: to write

Chapter 103
Byobu

> Assigned to: Andrea

You need to learn to use Byobu. It will save you much time later.
(Alternatives such as GNU Screen are fine as well.)

103.1. Advantages of using Byobu

TODO: To write

103.2. Installation

On Ubuntu, install using:

```
$ sudo apt install byobu
```

103.3. Documentation

- *See the screencast on the website http://byobu.co/.*

103.4. Quick command reference

You can change the escape sequence from Ctrl-A to something else by using the configuration tool that appears when you type F9.

Commands to use windows:

Table 15. Windows

	Using function keys	Using escape sequences
Create new window	F2	Ctrl-A then C
Previous window	F3	
Next window	F4	
Switch to window		Ctrl-A then a number
Close window	F6	
Rename window		Ctrl-A then ,

Commands to use panes (windows split in two or more):

Table 16. Commands for panes

	Using function keys	Using escape sequences
Split horizontally	Shift-F2	Ctrl-A then |
Split vertically	Ctrl-F2	Ctrl-A then %
Switch focus among panes	Ctrl-↑↓←→	Ctrl-A then one of ↑↓←→
Break pane		Ctrl-A then !

Other commands:

TABLE 17. OTHER

	Using function keys	Using escape sequences
Help		Ctrl-A then ?
Detach		Ctrl-A then D

103.5. Commands on OS X

Scroll up and down using fn option ↑ and fn option ↓.

Highlight using alt

Chapter 104
Source code control with Git

> Assigned to: Andrea

104.1. Background reading

TODO: to write

- Git
- GitFlow

104.2. Installation

The basic Git program is installed using

```
$ sudo apt install git
```

Additional utilities for `git` are installed using:

```
$ sudo apt install git-extras
```

This include the `git-ignore` utility.

104.3. Setting up global configurations for Git

This should be done twice, once on the laptop, and later, on the robot.

These options tell Git who you are:

```
$ git config --global user.email "email"
$ git config --global user.name  "full name"
```

Also do this, and it doesn't matter if you don't know what it is:

```
$ git config --global push.default simple
```

104.4. Git tips

1) Shallow clone

You can clone without history with the command:

```
$ git clone --depth 1 repository URL
```

104.5. Git troubleshooting

1) Problem 1: https instead of ssh:

The symptom is:

```
$ git push
Username for 'https://github.com':
```

Diagnosis: the `remote` is not correct.

If you do `git remote` you get entries with `https:`:

```
$ git remote -v
origin  https://github.com/duckietown/Software.git (fetch)
origin  https://github.com/duckietown/Software.git (push)
```

Expectation:

```
$ git remote -v
origin  git@github.com:duckietown/Software.git (fetch)
origin  git@github.com:duckietown/Software.git (push)
```

Solution:

```
$ git remote remove origin
$ git remote add origin git@github.com:duckietown/Software.git
```

2) Problem 1: `git` push complains about upstream

The symptom is:

```
fatal: The current branch branch name has no upstream branch.
```

Solution:

```
$ git push --set-upstream origin branch name
```

104.6. `git`

TODO: to write

Chapter 105
Git LFS

This describes Git LFS.

105.1. Generic installation instructions

See instructions at:

> https://git-lfs.github.com/

105.2. Ubuntu 16 installation (laptop)

Following these instructions, run the following:

```
$ sudo add-apt-repository ppa:git-core/ppa
$ curl -s https://packagecloud.io/install/repositories/github/git-lfs/script.deb.sh | sudo bash
$ sudo apt update
$ sudo apt install git-lfs
```

105.3. Ubuntu 16 Mate installation (Raspberry Pi 3)

> Note: unresolved issues.

The instructions above do not work.

Following this, the error that appears is that golang on the Pi is 1.6 instead it should be 1.7.

1) Troubleshooting

> Symptom: The binary files are not downloaded. In their place, there are short "pointer" files.

If you have installed LFS after pulling the repository and you see only the pointer files, do:

```
$ git lfs pull --all
```

Chapter 106
Setup Github access

Assigned to: Andrea

This chapter describes how to create a Github account and setup SSH on the robot and on the laptop.

106.1. Create a Github account

Our example account is the following:

```
Github name: greta-p
E-mail: greta-p@duckietown.com
```

Create a Github account (Figure 53).

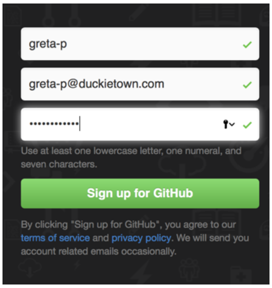

Figure 53

Go to your inbox and verify the email.

106.2. Become a member of the Duckietown organization

Give the administrators your account name. They will invite you.

Accept the invitation to join the organization that you will find in your email.

106.3. Add a public key to Github

You will do this procedure twice: once for the public key created on the laptop, and later with the public key created on the robot.

> Requires:
>
> - A public/private keypair already created and configured.
>   - This procedure is explained in Section 97.5.
>
> Result:
>
> - You can access Github using the key provided.

Go to settings (Figure 54).

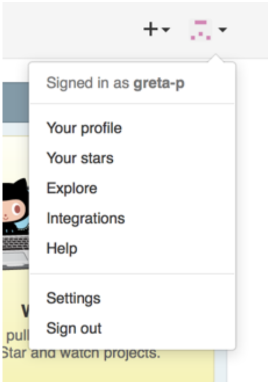

Figure 54

Add the public key that you created:

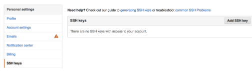

Figure 55

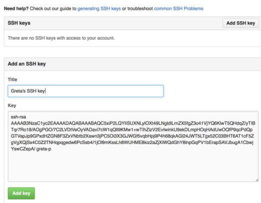

Figure 56

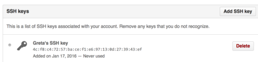

Figure 57

To check that all of this works, use the command

```
$ ssh -T git@github.com
```

The command tries to connect to Github using the private keys that you specified. This is the expected output:

```
Warning: Permanently added the RSA host key for IP address 'ip address' to the list of known
hosts.
Hi username! You've successfully authenticated, but GitHub does not provide shell access.
```

If you don't see the greeting, stop.

Repeat what you just did for the Duckiebot on the laptop as well, making sure to change the name of the file containing the private key.

Chapter 107
ROS installation and reference

Assigned to: Liam

107.1. Install ROS

This part installs ROS. You will run this twice, once on the laptop, once on the robot.

The first commands are copied from this page.

Tell Ubuntu where to find ROS:

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/
sources.list.d/ros-latest.list'
```

Tell Ubuntu that you trust the ROS people (they are nice folks):

```
$ sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key
421C365BD9FF1F717815A3895523BAEEB01FA116
```

Fetch the ROS repo:

```
$ sudo apt update
```

Now install the mega-package `ros-kinetic-desktop-full`.

```
$ sudo apt install ros-kinetic-desktop-full
```

There's more to install:

```
$ sudo apt install
ros-kinetic-{tf-conversions,cv-bridge,image-transport,camera-info-manager,theora-image-transport,joy,image-
```

Note: Do not install packages by the name of `ros-X`, only those by the name of `ros-kinetic-X`. The packages `ros-X` are from another version of ROS.

: not done in aug20 image:

Initialize ROS:

```
$ sudo rosdep init
$ rosdep update
```

107.2. `rqt_console`

TODO: to write

107.3. `roslaunch`

TODO: to write

107.4. `rviz`

TODO: to write

107.5. `rostopic`

TODO: to write

1) `rostopic hz`

TODO: to write

2) `rostopic echo`

TODO: to write

107.6. `catkin_make`

TODO: to write

107.7. `rosrun`

TODO: to write

107.8. `rostest`

TODO: to write

107.9. `rospack`

TODO: to write

107.10. `rosparam`

TODO: to write

107.11. `rosdep`

TODO: to write

107.12. roswtf

TODO: to write

107.13. rosbag

```
$ rosbag reindex bag file
```

107.14. roscore

TODO: to write

107.15. Troubleshooting ROS

Symptom: `computer is not in your SSH known_hosts file`

See this thread. Remove the `known_hosts` file and make sure you have followed the instructions in Section 97.3.

107.16. Other materials about ROS.

- *A gentle introduction to ROS*

Part 9
Software development guide

This part is about how to develop software for the Duckiebot.

CHAPTER 108

Python

108.1. Background reading

- Python
- Python tutorial

108.2. Python virtual environments

Install using:

```
$ sudo apt install virtualenv
```

108.3. Useful libraries

```
matplotlib
seaborne
numpy
panda
scipy
opencv
...
```

Chapter 109
Duckietown code conventions

109.1. Python

1) Tabs

Never use tabs in Python file.

The tab characters are evil in Python code. Please be *very* careful in changing them.

Do *not* use a tool to do it (e.g. "Convert tabs to spaces"); it will get it wrong.

✔ checked by `what-the-duck`.

2) Spaces

Indentation is 4 spaces.

3) Line lengths

Lines should be below 85 characters.

✔ `what-the-duck` report those above 100 characters.

This is just a symptom of a bigger problem.

The problem here is that you do not do how to program well, therefore you create programs with longer lines.

Do not go and try to shorten the lines; the line length is just the symptom. Rather, ask somebody to take a look at the code and tell you how to make it better.

4) Encoding lines

All files have an encoding declared.

```
# -*- coding: utf-8 -*-
```

5) Sha-bang lines

Executable files start with:

```
#!/usr/bin/env python
```

6) Comments

Comments refer to the next line.

Comments, bad:

```
from std_msgs.msg import String # This is my long comment
```

Comments, better:

```
# This is my long comment
from std_msgs.msg import String
```

CHAPTER 110
Configuration

This chapter explains what are the assumptions about the configuration.

While the “Setup” parts are “imperative” (do this, do that); this is the “declarative” part, which explains what are the properties of a correct configuration (but it does not explain how to get there).

he tool `what-the-duck` (Section 156.1) checks some of these conditions. If you make a change from the existing conditions, make sure that it gets implemented in `what-the-duck` by filing an issue.

110.1. Environment variables

You need to have set up the variables in Table 18.

TABLE 18. ENVIRONMENT VARIABLES USED BY THE SOFTWARE

variable	reasonable value	contains
`DUCKIETOWN_ROOT`	`~/duckietown`	`Software` repository
`DUCKIEFLEET_ROOT`	`~/duckiefleet`	A repository that contains `scuderia.yaml` and other team-specific configuration.
`DUCKIETOWN_DATA`	`~/duckietown-data`	Contains data for unit tests (Dropbox folder)

1) Duckietown root directory `DUCKIETOWN_ROOT`

TODO: to write

2) Duckiefleet directory `DUCKIEFLEET_ROOT`

For Fall 2017, this is the the repository `duckiefleet-fall2017`.

For self-guided learners, this is an arbitrary repository to create.

110.2. The scuderia file

In the `${DUCKIEFLEET_ROOT}` directory, there needs to exist a file called:

```
${DUCKIEFLEET_ROOT}/scuderia.yaml
```

The file must contain YAML entries of the type:

```
robot-name:
    username: username
    owner_duckietown_id: owner duckietown ID
```

A minimal example is in Listing 4.

```
emma:
  username: andrea
  owner_duckietown_id: censi
```

Listing 4. Minimal scuderia file

Explanations of the fields:

- `robot name`: the name of the robot, also equal to the host name.
- `username`: the name of the Linux user on the robot, from which to run programs.
- `owner_duckietown_id`: the owner's globally-unique Duckietown ID.

110.3. The `machines` file

The `machines` file is created using:

```
$ rosrun duckietown create-machines-file
```

110.4. People database

> Assigned to: Andrea

TODO: Describe the people database; this is the evolution of the yaml files

1) The globally-unique Duckietown ID

This is a globally-unique ID for people in the Duckietown project.

It is equal to the Slack username.

Chapter 111
Node configuration mechanisms

TODO: Where the config files are, how they are used.

Chapter 112
Minimal ROS node - `pkg_name`

> Assigned to: Andrea

This document outline the process of writing a ROS package and nodes in Python.

To follow along, it is recommend that you duplicate the `pkg_name` folder and edit the content of the files to make your own package.

112.1. The files in the package

1) `CMakeLists.txt`

We start with `CMakeLists.txt`.

Every ROS package needs a file `CMakeLists.txt`, even if you are just using Python code in your package.

☀ *documentation about CMakeLists.txt*

For a Python package, you only have to pay attention to the following parts.

The line:

```
project(pkg_name)
```

defines the name of the project.

The `find_package` lines:

```
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  duckietown_msgs # Every duckietown packages must use this.
  std_msgs
)
```

You will have to specify the packages on which your package is dependent.

In Duckietown, most packages depend on `duckietown_msgs` to make use of the customized messages.

The line:

```
catkine_python_setup()
```

tells `catkin` to setup Python-related stuff for this package.

☀ *ROS documentation about `setup.py`*

2) `package.xml`

The file `package.xml` defines the meta data of the package.

Catkin makes use of it to flush out the dependency tree and figures out the order of compiling.

Pay attention to the following parts.

`<name>` defines the name of the package. It has to match the project name in `CMakeLists.txt`.

`<description>` describes the package concisely.

`<maintainer>` provides information of the maintainer.

`<build_depend>` and `<run_depend>`. The catkin packages this package depends on. This usually match the `find_package` in `CMakeLists.txt`.

3) setup.py

The file `setup.py` configures the Python modules in this package.

The part to pay attention to is

```
setup_args = generate_distutils_setup(
    packages=['pkg_name'],
    package_dir={'': 'include'},
)
```

The `packages` parameter is set to a list of strings of the name of the folders inside the `include` folder.

The convention is to set the folder name the same as the package name. Here it's the `include/pkg_name` folder.

You should put ROS-independent and/or reusable module (for other packages) in the `include/pkg_name` folder.

Python files in this folder (for example, the `util.py`) will be available to scripts in the `catkin` workspace (this package and other packages too).

To use these modules from other packages, use:

```
from pkg_name.util import *
```

112.2. Writing a node: `talker.py`

Let's look at `src/talker.py` as an example.

ROS nodes are put under the `src` folder and they have to be made executable to function properly.

→ You use `chmod` for this; see Section 90.1.

1) Header

Header:

```
#!/usr/bin/env python
import rospy
# Imports module. Not limited to modules in this package.
from pkg_name.util import HelloGoodbye
# Imports msg
from std_msgs.msg import String
```

The first line, `#!/usr/bin/env python`, specifies that the script is written in Python.

Every ROS node in Python must start with this line.

The line `import rospy` imports the `rospy` module necessary for all ROS nodes in Python.

The line `from pkg_name.util import HelloGoodbye` imports the class `HelloGoodbye` defined in the file `pkg_name/util.py`.

Note that you can also include modules provided by other packages, if you specify the dependency in `CMakeLists.txt` and `package.xml`.

The line `from std_msgs.msg import String` imports the `String` message defined in the `std_msgs` package.

Note that you can use `rosmsg show std_msgs/String` in a terminal to lookup the definition of `String.msg`.

2) Main

This is the main file:

```
if __name__ == '__main__':
    # Initialize the node with rospy
    rospy.init_node('talker', anonymous=False)

    # Create the NodeName object
    node = Talker()

    # Setup proper shutdown behavior
    rospy.on_shutdown(node.on_shutdown)

    # Keep it spinning to keep the node alive
    rospy.spin()
```

The line `rospy.init_node('talker', anonymous=False)` initializes a node named `talker`.

Note that this name can be overwritten by a launch file. The launch file can also push this node down namespaces. If the `anonymous` argument is set to `True` then a random string of numbers will be append to the name of the node. Usually we don't use anonymous nodes.

The line `node = Talker()` creates an instance of the Talker object. More details in the next section.

The line `rospy.on_shutdown(node.on_shutdown)` ensures that the `node.on_shutdown` will be called when the node is shutdown.

The line `rospy.spin()` blocks to keep the script alive. This makes sure the node stays alive and all the publication/subscriptions work correctly.

112.3. The `Talker` class

We now discuss the `Talker` class in `talker.py`.

1) Constructor

In the constructor, we have:

```
self.node_name = rospy.get_name()
```

saves the name of the node.

This allows to include the name of the node in printouts to make them more informative. For example:

```
rospy.loginfo("[%s] Initializing." % (self.node_name))
```

The line:

```
self.pub_topic_a = rospy.Publisher("~topic_a", String, queue_size=1)
```

defines a publisher which publishes a `String` message to the topic `~topic_a`. Note that the `~` in the name of topic under the namespace of the node. More specifically, this will actually publishes to `talker/topic_a` instead of just `topic_a`. The `queue_size` is usually set to 1 on all publishers.

➡ For more details see `rospy` overview: publisher and subscribers.

The line:

```
self.sub_topic_b = rospy.Subscriber("~topic_b", String, self.cbTopic)
```

defines a subscriber which expects a `String` message and subscribes to `~topic_b`. The message will be handled by the `self.cbTopic` callback function. Note that similar to the publisher, the `~` in the topic name puts the topic under the namespace of the node. In this case the subscriber actually subscribes to the topic `talker/topic_b`.

It is strongly encouraged that a node always publishers and subscribes to topics under their `node_name` namespace. In other words, always put a `~` in front of the topic names when you defines a publisher or a subscriber. They can be easily remapped in a launch file. This makes the node more modular and minimizes the possibility of confusion and naming conflicts. See [the launch file section][howto-launch-file] for how remapping works.

The line

```
self.pub_timestep = self.setupParameter("~pub_timestep", 1.0)
```

Sets the value of `self.pub_timestep` to the value of the parameter `~pub_timestep`. If the parameter doesn't exist (not set in the launch file), then set it to the default value `1.0`. The `setupParameter` function also writes the final value to the parameter server. This means that you can `rosparam list` in a terminal to check the actual values of parameters being set.

The line:

```
self.timer = rospy.Timer(rospy.Duration.from_sec(self.pub_timestep), self.cbTimer)
```

defines a timer that calls the `self.cbTimer` function every `self.pub_timestep` seconds.

2) Timer callback

Contents:

```
def cbTimer(self,event):
    singer = HelloGoodbye()
    # Simulate hearing something
    msg = String()
    msg.data = singer.sing("duckietown")
    self.pub_topic_name.publish(msg)
```

Everyt ime the timer ticks, a message is generated and published.

3) Subscriber callback

Contents:

```
def cbTopic(self,msg):
    rospy.loginfo("[%s] %s" %(self.node_name,msg.data))
```

Every time a message is published to `~topic_b`, the `cbTopic` function is called. It simply prints the messae using `rospy.loginfo`.

112.4. Launch File

You should always write a launch file to launch a node. It also serves as a documentation on the I/O of the node.

Let's take a look at `launch/test.launch`.

```
<launch>
    <node name="talker" pkg="pkg_name" type="talker.py" output="screen">

        <param name="~pub_timestep" value="0.5"/>

        <remap from="~topic_b" to="~topic_a"/>
    </node>
</launch>
```

For the `<node>`, the `name` specify the name of the node, which overwrites `rospy.init_node()` in the `__main__` of `talker.py`. The `pkg` and `type` specify the package and the script of the node, in this case it's `talke.py`.

Don't forget the `.py` in the end (and remember to make the file executable through `chmod`).

The `output="screen"` direct all the `rospy.loginfo` to the screen, without this you won't see any printouts (useful when you want to suppress a node that's too talkative.)

The `<param>` can be used to set the parameters. Here we set the `~pub_timestep` to `0.5`. Note that in this case this sets the value of `talker/pub_timestep` to `0.5`.

The `<remap>` is used to remap the topic names. In this case we are replacing `~topic_b` with `~topic_a` so that the subscriber of the node actually listens to its own publisher. Replace the line with

```
<remap from="~topic_b" to="talker/topic_a"/>
```

will have the same effect. This is redundant in this case but very useful when you want to subscribe to a topic published by another node.

112.5. Testing the node

First of all, you have to `catkin_make` the package even if it only uses Python. `catkin` makes sure that the modules in the include folder and the messages are available to the whole workspace. You can do so by

```
$ cd ${DUCKIETOWN_ROOT}/catkin_ws
$ catkin_make
```

Ask ROS to re-index the packages so that you can auto-complete most things.

```
$ rospack profile
```

Now you can launch the node by the launch file.

```
$ roslaunch pkg_name test.launch
```

You should see something like this in the terminal:

```
... logging to /home/username/.ros/log/d4db7c80-b272-11e5-8800-5c514fb7f0ed/roslaunch-robot name-15961.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is 1GB.

started roslaunch server http://robot name.local:33925/

SUMMARY
========

PARAMETERS
 * /rosdistro: $ROS_DISTRO
 * /rosversion: 1.11.16
 * /talker/pub_timestep: 0.5

NODES
  /
    talker (pkg_name/talker.py)

auto-starting new master
process[master]: started with pid [15973]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to d4db7c80-b272-11e5-8800-5c514fb7f0ed
process[rosout-1]: started with pid [15986]
started core service [/rosout]
process[talker-2]: started with pid [15993]
[INFO] [WallTime: 1451864197.775356] [/talker] Initialzing.
[INFO] [WallTime: 1451864197.780158] [/talker] ~pub_timestep = 0.5
[INFO] [WallTime: 1451864197.780616] [/talker] Initialzed.
[INFO] [WallTime: 1451864198.281477] [/talker] Goodbye, duckietown.
[INFO] [WallTime: 1451864198.781445] [/talker] Hello, duckietown.
[INFO] [WallTime: 1451864199.281871] [/talker] Goodbye, duckietown.
[INFO] [WallTime: 1451864199.781486] [/talker] Hello, duckietown.
[INFO] [WallTime: 1451864200.281545] [/talker] Goodbye, duckietown.
[INFO] [WallTime: 1451864200.781453] [/talker] Goodbye, duckietown.
```

Open another terminal and run:

```
$ rostopic list
```

You should see

```
/rosout
/rosout_agg
/talker/topic_a
```

In the same terminal, run:

```
$ rosparam list
```

You should see the list of parameters, including `/talker/pub_timestep`.

You can see the parameters and the values of the `talker` node with

```
$ rosparam get /talker
```

112.6. Documentation

You should document the parameters and the publish/subscribe topic names of each node in your package. The user should not have to look at the source code to figure out how to use the nodes.

112.7. Guidelines

- Make sure to put all topics (publish or subscribe) and parameters under the namespace of the node with `~`. This makes sure that the IO of the node is crystal clear.
- Always include the name of the node in the printouts.
- Always provide a launch file that includes all the parameters (using `<param>`) and topics (using `<remap>`) with each node.

CHAPTER 113
ROS package verification

This chapter describes formally what makes a conforming ROS package in the Duckietown software architecture.

113.1. Naming

☐ For exercises packages, the name of the package must be `package_handle`.

113.2. `package.xml`

☐ There is a `package.xml` file.

✔ Checked by `what-the-duck`.

113.3. Messages

☐ The messages are called ....

113.4. Readme file

☐ There is a `README.md` file

✔ Checked by `what-the-duck`.

113.5. Launch files

☐ there is the first launch file

113.6. Test files

TODO: to write

Chapter 114
Creating unit tests with ROS

PART 10
Duckietown system

This part describes the Duckietown algorithms and system architecture.

We do not go in the software details. The implementation details have been already talked about at length in Part 9.

We do give links to the ROS packages implementing the functionality.

Chapter 115
Teleoperation

TODO: add video here

115.1. Implementation

Drivers:

- Chapter 157 - Package `adafruit_drivers`
- Chapter 167 - Package `pi_camera`

Operator interface:

- Chapter 162 - Package `joy_mapper`

115.2. Camera

TODO: to write

115.3. Actuators

TODO: to write

115.4. IMU

TODO: to write

CHAPTER 116

Parallel autonomy

TODO: to write

Chapter 117
Lane control

TODO: video here

117.1. Implementation

Perception:

- Chapter 158 - Package anti_instagram
- Chapter 161 - Package ground_projection
- Chapter 166 - Package line_detector, Chapter 165 - Package line_detector2
- Chapter 164 - Package lane_filter

Control:

- Chapter 163 - Package lane_control
- Chapter 159 - Package car_supervisor
- Chapter 160 - Package dagu_car

Chapter 118
Indefinite navigation

TODO: add video here

118.1. Implementation

The packages involved in this functionality are:

- Chapter 168 - Package `apriltags_ros`
- Chapter 169 - Package `fsm`
- Chapter 170 - Package `indefinite_navigation`
- Chapter 171 - Package `intersection_control`
- Chapter 172 - Package `navigation`

> Note: we don't discuss the details of the packages here; we just give pointers to them.

Chapter 119
Planning

TODO: add video here

119.1. Implementation

The packages involved in this functionality are:

- Chapter 175 - Package `localization`
- Chapter 174 - Package `duckietown_description`

> **Note:** we don't discuss the details of the packages here; we just give pointers to them.

Chapter 120
Coordination

TODO: add video here

120.1. Implementation

- Chapter 176 - Package `led_detection`
- Chapter 177 - Package `led_emitter`
- Chapter 178 - Package `led_interpreter`
- Chapter 179 - Package `led_joy_mapper`
- Chapter 181 - Package `traffic_light`
- Chapter 180 - Package `rgb_led`

Part 11
Fall 2017

This is the first time that a class is taught jointly across 3 continents!

There are 4 universities involved in the joint teaching for the term:

- ETH Zürich (ETHZ), with instructors Emilio Frazzoli, Andrea Censi, Jacopo Tani.
- University of Montreal (UdeM), with instructor Liam Paull.
- TTI-Chicago (TTIC), with instructor Matthew Walter.
- National C T University (NCTU), with instructor Nick Wang.

This part of the Duckiebook describes all the information that is needed by the students of the four institutions.

CHAPTER 121
General remarks

Assigned to: Andrea

121.1. The rules of Duckietown

The first rule of Duckietown

The first rule of Duckietown is: you don't talk about Duckietown, *using email*.

Instead, we use a communication platform called Slack.

There is one exception: inquiries about "meta" level issues, such as course enrollment and other official bureaucratic issues can be communicated via email.

The second rule of Duckietown

The second rule of Duckietown is: be kind and respectful, and have fun.

The third rule of Duckietown

The third rule of Duckietown is: read the instructions carefully.

Do not blindly copy and paste.

Only run a command if you know what it does.

121.2. Synchronization between classes

At ETHZ, UdeM, TTIC, the class will be more-or-less synchronized. The materials are the same; there is some slight variation in the ordering.

Moreover, there will be some common groups for the projects.

The NCTU class is undergraduate level. Students will learn slightly simplified materials. They will not collaborate directly with the classes.

121.3. Accounts for students

To participate in Duckietown, students must use two accounts: Slack and Github.

1) Slack

You need a Slack account, for team discussion and organization.

TODO: Sign up link here:

TODO: Account naming convention

2) Github

TODO: Account naming convention

- A Github account;
- Membership in the Duckietown organization.

121.4. Accounts for all instructors and TAs

As an instructor/TA for the Fall 2017 class, in addition to the accounts above, these are two more accounts that you need.

1) Twist

Twist is used for class organization (such as TAs, logistics);

TODO:

2) Google docs

Google Docs is used to maintain TODOs and other coordination materials.

TODO: how to be authorized?

In particular:

- This is the schedule: XXX
- This is the calendar in which to annotate everything: XXX

CHAPTER 122
Additional information for ETH Zürich students

> Assigned to: Andrea

This section describes information specific for ETH Zürich students.

TODO: to write

1) Website

> All really important information, such as deadlines, is in the authoritative website:

2) Duckiebox distribution

TODO: to write

3) Lab access

TODO: To write

4) The local TAs

TODO: to write

Chapter 123
Additional information for UdeM students

> **Assigned to:** Liam

TODO: to write

Chapter 124
Additional information for TTIC students

> Assigned to: Matt

TODO: to write

Chapter 125
Additional information for NCTU students

> Assigned to: Nick

TODO: to write

Chapter 126
Milestone: ROS node working

Chapter 127

Homework: Take and process a log

CHAPTER 128

Milestone: Calibrated robot

Chapter 129
Homework: Camera geometry

Chapter 130

Milestone: Illumination invariance

Chapter 131
Homework: Place recognition

Chapter 132

Milestone: Lane following

Chapter 133
Homework: localization

Chapter 134
Milestone: Navigation

Chapter 135

Homework: group forming

Chapter 136

Milestone: Ducks in a row

Chapter 137
Homework: Comparison of PID

Chapter 138

Homework: RRT

Chapter 139

Caffe tutorial

Chapter 140
Milestone: Object Detection

CHAPTER 141

Homework: Object Detection

Chapter 142

Milestone: Semantic perception

Chapter 143
Homework: Semantic perception

CHAPTER 144

Milestone: Reacting to obstacles

Chapter 145
Homework: Reacting to obstacles

Chapter 146
Milestone: SLAM demo

Chapter 147
Homework: SLAM

Chapter 148

Milestone: fleet demo

Chapter 149

Homework: fleet

Chapter 150

Project proposals

Chapter 151
Template of a project

151.1. Checklist for students

- Have a Github account. See Chapter 106. See name conventions (TODO).
- Be part of the Duckietown Github organization. You are sure only when you commit and push one change to one of our repositories.
- Be part of the Duckietown Slack. See name conventions (TODO).

151.2. Checklist for TAs

- Be signed up on

Part 12

Packages - Infrastructure

TODO: to write

Chapter 152
Package `duckietown`

TODO: to write

CHAPTER 153
Package duckietown_msgs

TODO: to write

Chapter 154
Package easy_node

easy_node is a framework to make it easier to create and document ROS nodes. It allows a *declarative approach* to declaring subscriptions, publishers, and parameters.

The user can directly describe what are the subscription, the publishers, the parameters in a YAML file. The framework then takes care of calling the necessary boilerplate ROS commands for subscribing, publishing, etc.

In addition, easy_node can also create the Mardkown documentation from the YAML file.

Using easy_node allows to cut 40%-50% of the code required for programming a node. For an example, see the package line_detector2, which contains a re-implementation of line_detector using the new framework.

154.1. Transition plan

The plan is to first use easy_node just for documentation of the nodes; then, later, convert all the nodes to use it.

154.2. YAML file format

If you have a node with the name my node, implemented in the file my node.py you must create a file by the name my node.easy_node.yaml somewhere in the package.

The YAML file must contain 4 sections, each of which is a dictionary.

This is the smallest example of an empty configuration:

```
parameters:
subscriptions:
publishers:
contracts:
```

1) Configuring parameters

This is the syntax:

```
parameters:
    name parameter:
        type: type
        desc: description
        default: default value
```

where:

- ![type] is one of float, int, bool, str.
- ![description] is a description that will appear in the documentation.
- The optional field default gives a default value for the parameter.

For example:

```
parameters:
    k_d:
        type: float
        desc: The derivative gain for $\theta$.
        default: 1.02
```

2) Describing publishers and subscriptions

The syntax for describing subscribers is:

```
subscriptions:
    name subscription:
        topic: topic name
        type: message type
        desc: description

        queue_size: queue size
        latch: latch
```

where:

- `topic name` is the name of the topic to subscribe.
- `message type` is a ROS message type name, such as `sensor_msgs/Joy`.
- `description` is a Markdown description string.
- `queue size`, `latch` are optional parameters for ROS publishing/subscribing functions.

The syntax for describing publishers is similar.

Example:

```
subscriptions:
    segment_list:
        topic: ~segment_list
        type: duckietown_msgs/SegmentList
        desc: Line detections
        queue_size: 1

publishers:
    lane_pose:
        topic: ~lane_pose
        type: duckietown_msgs/LanePose
        desc: Estimated pose
        queue_size: 1
```

3) Describing contracts

This is not implemented yet. The idea is to have a place where we can describe constraints such as:

- “This topic must publish at least at 30 Hz.”
- “Panic if you didn’t receive a message for 2 seconds.”
- “The maximum latency for this is 0.2 s”

Then, we can implement all these checks once and for all in a proper way, instead of relying on multiple broken implementations

154.3. Automatic docs generation

Generate the docs for each node using this command:

```
$ rosrun easy_node generate_docs.py
```

Chapter 155
Duckietown ROS Gudieline

155.1. Node and Topics

In the source code, a node must only publish/subscribe to private topics.

In `rospy`, this means that the topic argument of `rospy.Publisher` and `rospy.Subscriber` should always have a leading `~`. ex: `~wheels_cmd`, `~mode`.

In `roscpp`, this means that the node handle should always be initialized as a private node handle by supplying with a `"~"` agrument at initialization. Note that the leading "~" must then be obmitted in the topic names of. ex:

```
ros::NodeHandle nh_("~");
sub_lineseglist_ = nh_.subscribe("lineseglist_in", 1, &GroundProjection::lineseglist_cb, this);
pub_lineseglist_ = nh_.advertise<duckietown_msgs::SegmentList> ("lineseglist_out", 1);
```

155.2. Parameters

All the parameters of a node must be private parameters to that node.

All the nodes must write the value of the parameters being used to the parameter server at initialization. This ensures transparence of the parameters. Note that the `get_param(name,default_value)` does not write the default value to the parameter server automatically.

The default parameter of `pkg_name/node_name` should be put in `~/duckietown/catkin_ws/src/duckietown/config/baseline/pkg_name/node_name/dafault.yaml`. The elemental launch file of this node should load the parameter using `<rosparam>`.

155.3. Launch file

Each node must have a launch file with the same name in the `launch` folder of the package. ex: `joy_mapper.py` must have a `joy_mapper.launch`. These are referred to as the elemental launch files.

Each elemental launch file must only launch one node.

The elemental launch file should put the node under the correct namespace through the `veh` arg, load the correct configuration and parameter file throught `config` and `param_file_name` args respectively. `veh` must not have a default value. This is to ensure the user to always provide the `veh` arg. `config` must be default to `baseline` and `param_file_name` must be default to `default`.

When a node can be run on the vehicle or on a laptop, the elemental launch file should provide a `local` arg. When set to true, the node must be launch on the launching machine, when set to false, the node must be launch on a vehicle throught the `mahcine` attribute.

A node should always be launched by calling its corresponding launch file instead of using `rosrun`. This ensures that the node is put under the correct namespace and all the necessary parameters are provided.

Do not use `<remapp>` in the elemental launch files.

Do not use `<param>` in the elemental launch files.

Chapter 156
Package `what_the_duck`

`what-the-duck` is a program that tests *dozens* of configuration inconsistencies that can happen on a Duckiebot.

156.1. What the duck

To use it, first compile the repository, and then run:

```
$ ./what-the-duck
```

156.2. Adding more tests to `what-the-duck`

The idea is to add to `what-the-duck` all the tests that can be automated.

The documentation about to do that is not ready yet.

156.3. Tests already added

Here is the list of tests already added:

```
✓  Camera is detected
✓  Scipy is installed
✓  sklearn is installed
✓  Date is set correctly
✓  Not running as root
✓  Not running as ubuntu
✓  Member of group sudo
✓  Member of group input
✓  Member of group video
✓  Member of group i2c
✓  ~/.ssh exists
✓  ~/.ssh permissions
✓  ~/.ssh/config exists
✓  SSH option HostKeyAlgorithms is set
✓  At least one key is configured.
✓  ~/.ssh/authorized_keys exists
✓  Git configured
✓  Git email set
✓  Git name set
✓  Git push policy set
✓  Edimax detected
✓  The hostname is configured
✓  /etc/hosts is sane
✓  Correct kernel version
✓  Messages are compiled
✓  Shell is bash
✓  Working internet connection
✓  Github configured
✓  Joystick detected
✓  Environment variable DUCKIETOWN_ROOT
✓  ${DUCKIETOWN_ROOT} exists
✓  Environment variable DUCKIETOWN_FLEET
✓  ${DUCKIETOWN_FLEET} exists
✓  ${DUCKIETOWN_FLEET}/scuderia.yaml exists
✓  ${DUCKIETOWN_FLEET}/scuderia.yaml is valid
✓  machines file is valid
✓  Wifi network configured
✓  Python: No CamelCase
✓  Python: No tab chars
✓  Python:  No half merges
```

156.4. List of tests to add

Please add below any configuration test that can be automated:

- Editor is set to `vim`.
- Syntax on in `~/.vimrc`
- They put the right MAC address in the network configuration
- Ubuntu user is in group video, input, i2c (even if run from other user.)
- There is at least X.YGB of free disk space.

- If the SD is larger than 8GB, the disk has been resized.

Part 13
Packages - Lane control

TODO: to write

Chapter 157

Package adafruit_drivers

TODO: to write

Chapter 158
Package anti_instagram

TODO: to write

158.1. Unit tests integrated with rostest

Unit tests are integrated with [rostest][#rostest].

To run manually, use:

```
$ rostest anti_instagram antiinstagram_correctness_test.test
$ rostest anti_instagram antiinstagram_stub_test.test
$ rostest anti_instagram antiinstagram_performance_test.test
```

158.2. Unit tests needed external files

These are other unittest that require the logs in DUCKIETOWN_DATA:

```
$ rosrun anti_instagram annotations_test.py
```

158.3. Node anti_instagram_node

1) Parameters

Parameter `publish_corrected_image`: `bool`; default value: `False`

Whether to compute and publish the corrected image.

2) Subscriptions

Subscription `image`: topic `~uncorrected_image` (`CompressedImage`)

This is the compressed image to read.

Subscription `click`: topic `~click` (`BoolStamped`)

Activate the calibration phase with this switch.

3) Published topics

Publisher `image`: topic `~corrected_image` (`Image`)

The corrected image.

Publisher `health`: topic `~colorSegment` (`AntiInstagramHealth`)

The health of the process.

Publisher `transform`: topic `~transform` (`AntiInstagramTransform`)

The computed transform.

Chapter 159
Package car_supervisor

TODO: to write

Chapter 160

Package dagu_car

TODO: to write

Chapter 161
Package ground_projection

TODO: to write

Chapter 162

Package `joy_mapper`

TODO: to write

162.1. Testing

To test run:

1) connect joystick 2) `roslaunch launch/joy_mapper_test.launch` 3) the robot should move when you push buttons

162.2. Dependencies

- `rospy`
- `sensor_msgs`: for the `Joy.msg`
- `duckietown_msgs`: for the `CarControl.msg`

162.3. Node: joy_mapper.py

This node takes a `sensor_msgs/Joy.msg` and converts it to a `duckietown_msgs/CarControl.msg`. It publishes at a fixed interval with a zero-order hold.

1) Parameters

Parameter `v_gain`: `float`; default value: `0.41`

TODO: Missing description for entry "`v_gain`".

Parameter `omega_gain`: `float`; default value: `8.3`

TODO: Missing description for entry "`omega_gain`".

Parameter `bicycle_kinematics`: `int`; default value: `0`

TODO: Missing description for entry "`bicycle_kinematics`".

Parameter `simulated_vehicle_length`: `float`; default value: `0.18`

TODO: Missing description for entry "`simulated_vehicle_length`".

Parameter `steer_angle_gain`: `int`; default value: `1`

TODO: Missing description for entry "`steer_angle_gain`".

2) Subscriptions

Subscription `joy`: topic `joy` (`Joy`)

The `Joy.msg` from `joy_node` of the `joy` package. The vertical axis of the left stick maps to speed. The horizontal axis of the right stick maps to steering.

3) Published topics

Publisher `avoidance`: topic `~start_avoidance` (`BoolStamped`)

TODO: Missing description for entry "`avoidance`".

Publisher `car_cmd`: topic `~car_cmd` (`Twist2DStamped`)

TODO: Missing description for entry "`car_cmd`".

Publisher `joy_override`: topic `~joystick_override` (`BoolStamped`)

TODO: Missing description for entry "`joy_override`".

Publisher `parallel_autonomy`: topic `~parallel_autonomy` (`BoolStamped`)

TODO: Missing description for entry "`parallel_autonomy`".

Publisher `e_stop`: topic `wheels_driver_node/emergency_stop` (`BoolStamped`)

TODO: Missing description for entry "`e_stop`".

Publisher `anti_instagram`: topic `anti_instagram_node/click` (`BoolStamped`)

TODO: Missing description for entry "`anti_instagram`".

CHAPTER 163

Package `lane_control`

TODO: to write

163.1. `lane_controller_node`

> **Note:** there is some very funny business inside. It appears that `k_d` and `k_theta` are switched around.

1) Parameters

Parameter `k_theta`: `float`

Proportional gain for θ.

Parameter `theta_thres`: `float`

Maximum desired θ.

Parameter `d_offset`: `float`

A configurable offset from the lane position.

Parameter `d_thres`: `float`

Cap for error in d.

Parameter `v_bar`: `float`

Nominal linear velocity (m/s).

Parameter `k_d`: `float`

Propertional gain for d.

2) Subscriptions

Subscription `lane_reading`: topic `~lane_pose` (`LanePose`)

TODO: Missing description for entry "`lane_reading`".

3) Published topics

Publisher `car_cmd`: topic `~car_cmd` (`Twist2DStamped`)

TODO: Missing description for entry "`car_cmd`".

Chapter 164
Package `lane_filter`

> Assigned to: Liam

TODO: to write

164.1. `lane_filter_node`

1) Parameters

Parameter `peak_val`: `float`; default value: `10.0`

TODO: Missing description for entry "`peak_val`".

Parameter `l_max`: `float`; default value: `2.0`

TODO: Missing description for entry "`l_max`".

Parameter `lanewidth`: `float`; default value: `0.4`

TODO: Missing description for entry "`lanewidth`".

Parameter `mean_d_0`: `float`; default value: `0.0`

TODO: Missing description for entry "`mean_d_0`".

Parameter `min_max`: `float`; default value: `0.3`

Expressed in nats.

Parameter `d_max`: `float`; default value: `0.5`

TODO: Missing description for entry "`d_max`".

Parameter `use_distance_weighting`: `bool`; default value: `False`

For use of distance weighting (dw) function.

Parameter `linewidth_white`: `float`; default value: `0.04`

TODO: Missing description for entry "`linewidth_white`".

Parameter `cov_omega`: `float`; default value: `0.01`

Angular velocity "input".

> which units?

Parameter `use_max_segment_dist`: `bool`; default value: `False`

For use of maximum segment distance.

Parameter `delta_d`: `float`; default value: `0.02`

(meters)

Parameter `min_segs`: `int`; default value: `10`

For use of minimum segment count.

Parameter `phi_min`: `float`; default value: `-1.5707`

TODO: Missing description for entry "`phi_min`".

Parameter `sigma_d_0`: `float`; default value: `0.0`

TODO: Missing description for entry "`sigma_d_0`".

Parameter `phi_max`: `float`; default value: `1.5707`

TODO: Missing description for entry "`phi_max`".

Parameter `zero_val`: `float`; default value: `1.0`

TODO: Missing description for entry "`zero_val`".

Parameter `delta_phi`: `float`; default value: `0.0`

(radians)

Parameter `l_peak`: `float`; default value: `1.0`

TODO: Missing description for entry "`l_peak`".

Parameter `cov_v`: `float`; default value: `0.5`

Linear velocity "input".

which units?

Parameter `sigma_phi_mask`: `float`; default value: `0.05`

TODO: Missing description for entry "`sigma_phi_mask`".

Parameter `sigma_d_mask`: `float`; default value: `0.05`

TODO: Missing description for entry "`sigma_d_mask`".

Parameter `mean_phi_0`: `float`; default value: `0.0`

TODO: Missing description for entry "`mean_phi_0`".

Parameter `sigma_phi_0`: `float`; default value: `0.0`

TODO: Missing description for entry "`sigma_phi_0`".

Parameter `d_min`: `float`; default value: `-0.7`

TODO: Missing description for entry "`d_min`".

Parameter `linewidth_yellow`: `float`; default value: `0.02`

TODO: Missing description for entry "`linewidth_yellow`".

Parameter `use_min_segs`: `bool`; default value: `False`

For use of minimum segment count.

Parameter `use_propagation`: `bool`; default value: `False`

For propagation.

Parameter `max_segment_dist`: `float`; default value: `1.0`

For use of maximum segment distance.

2) Subscriptions

Subscription `velocity`: topic `~velocity` (`Twist2DStamped`)

TODO: Missing description for entry "`velocity`".

Subscription `segment_list`: topic `~segment_list` (`SegmentList`)

TODO: Missing description for entry "`segment_list`".

3) Published topics

Publisher `belief_img`: topic `~belief_img` (`Image`)

TODO: Missing description for entry "`belief_img`".

Publisher `switch`: topic `~switch` (`BoolStamped`)

TODO: Missing description for entry "`switch`".

Publisher `lane_pose`: topic `~lane_pose` (`LanePose`)

TODO: Missing description for entry "`lane_pose`".

Publisher `entropy`: topic `~entropy` (`Float32`)

TODO: Missing description for entry "`entropy`".

Publisher `in_lane`: topic `~in_lane` (`BoolStamped`)

TODO: Missing description for entry "`in_lane`".

CHAPTER 165

Package `line_detector2`

This is a re-implementation of the package `line_detector` using the new facilities provided by `easy_node`.

165.1. Testing the line detector using visual inspection

The following are instructions to test the line detector from bag files.

You can run from a bag with the following:

```
$ roslaunch line_detector line_detector2_bag veh:=vehicle bagin:=bag in bagout:=bag out
verbose:=true
```

Where:

- `bag in` is the **absolute path** of the input bag.
- `vehicle` is the name of the vehicle that took the log.
- `bag out` is the **absolute path** if the output bag.

Note: you always need to use absolute paths for bag files.

You can let this run for a few seconds, then stop using Ctrl-C.

You can then inspect the result using:

```
$ roscore &
$ rosbag play -l bag out
$ rviz &
```

In `rviz` click "add", click "by topic" tab, expand "`line_detector`" and click "`image_with_lines`".

Observe on the result that:

1. There are *lots* of detections.
2. Predominantly white detections (indicated in black) are on white lines, yellow detections (shown in blue) are on blue lines, and red detections (shown in green) are on red lines.

These are some sample logs on which to try:

```
$ wget -O 160122-manual1_ferrari.bag https://www.dropbox.com/s/8bpi656j7qox5kv?dl=1

https://www.dropbox.com/s/vwznjke4xvnhi9o/160122_manual2-ferrari.bag?dl=1

https://www.dropbox.com/s/y7ulj198punj0mp/160122_manual3_corner-ferrari.bag?dl=1

https://www.dropbox.com/s/d4n9otmlans4i62/160122-calibration-good_lighting-tesla.bag?dl=1
```

Sample output:

```
From cmd:roslaunch duckietown camera.launch veh:=${VEHICLE_NAME}
[INFO] [WallTime: 1453839555.948481] [LineDetectorNode] number of white lines = 14
[INFO] [WallTime: 1453839555.949102] [LineDetectorNode] number of yellow lines = 33
[INFO] [WallTime: 1453839555.986520] [LineDetectorNode] number of white lines = 18
[INFO] [WallTime: 1453839555.987039] [LineDetectorNode] number of yellow lines = 34
[INFO] [WallTime: 1453839556.013252] [LineDetectorNode] number of white lines = 14
[INFO] [WallTime: 1453839556.013857] [LineDetectorNode] number of yellow lines = 29
[INFO] [WallTime: 1453839556.014539] [LineDetectorNode] number of red lines = 2
[INFO] [WallTime: 1453839556.047944] [LineDetectorNode] number of white lines = 18
[INFO] [WallTime: 1453839556.048672] [LineDetectorNode] number of yellow lines = 28
[INFO] [WallTime: 1453839556.049534] [LineDetectorNode] number of red lines = 2
[INFO] [WallTime: 1453839556.081400] [LineDetectorNode] number of white lines = 13
[INFO] [WallTime: 1453839556.081944] [LineDetectorNode] number of yellow lines = 34
[INFO] [WallTime: 1453839556.082479] [LineDetectorNode] number of red lines = 1
```

The output from `rviz` looks like Figure 58.

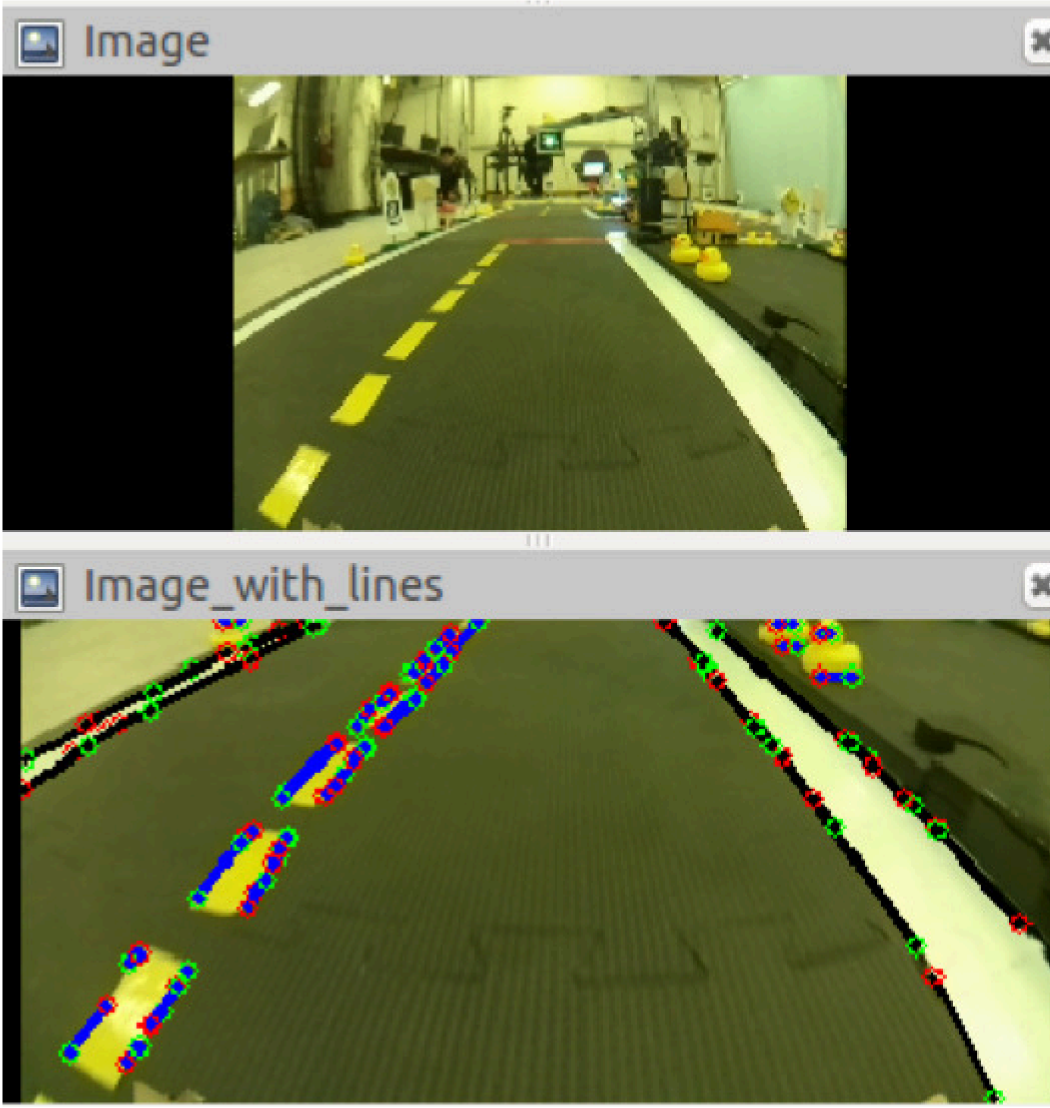

Figure 58

165.2. Quantitative tests

TODO: Something more quantitative (to be filled in by Liam or Hang)

165.3. `line_detector_node2`

This is a rewriting of `line_detector_node` using the EasyNode framework.

1) Parameters

Parameter `top_cutoff`: `int`; default value: `None`

This parameter decides how much of the image we should cut off. This is a performance improvement.

Parameter `detector`: not known

This parameter defines the line detector instantiation. It consists of an array of two elements, where the first is a classname or function, and the second is a dictionary of parameters to pass to the function.

Parameter `img_size`: not known

TODO: Missing description for entry "`img_size`".

Parameter `verbose`: `bool`; default value: `True`

Whether the node is verbose or not. If set to `True`, the node will write timing statistics to the log.

2) Subscriptions

Subscription `switch`: topic `~switch` (`BoolStamped`)

This is a switch that allows to control the activity of this node. If the message is true, the node becomes active. If false, it switches off. The node starts as active.

Subscription `image`: topic `~image` (`CompressedImage`)

This is the compressed image to read. Note that it takes a long time to simply decode the image JPG.

> Note: The data is processed *asynchronously* in a different thread.

Subscription `transform`: topic `~transform` (`AntiInstagramTransform`)

The anti-instagram transform to apply. See Chapter 158.

3) Published topics

Publisher `color_segment`: topic `~colorSegment` (`Image`)

TODO: Missing description for entry "`color_segment`".

Publisher `edge`: topic `~edge` (`Image`)

TODO: Missing description for entry "`edge`".

Publisher `segment_list`: topic `~segment_list` (`SegmentList`)

TODO: Missing description for entry "`segment_list`".

Publisher `image_with_lines`: topic `~image_with_lines` (`Image`)

TODO: Missing description for entry "`image_with_lines`".

CHAPTER 166

Package `line_detector`

This package is being replaced by the cleaned-up version, `line_detector2`. Do not write documentation here.

However, at the moment, all the launch files still call this one.

Chapter 167

Package pi_camera

TODO: to write

Part 14
Packages - Indefinite navigation

TODO: to write

Chapter 168

Package apriltags_ros

AprilTags for ROS.

Chapter 169
Package `fsm`

TODO: to write

Chapter 170
Package indefinite_navigation

TODO: to write

Chapter 171

Package `intersection_control`

TODO: to write

Chapter 172
Package navigation

TODO: to write

Chapter 173

Package stop_line_filter

TODO: to write

PART 15
Packages - Localization and planning

TODO: to write

Chapter 174
Package `duckietown_description`

TODO: to write

Chapter 175
Package `localization`

TODO: to write

PART 16

Packages - Coordination

TODO: to write

Chapter 176
Package led_detection

TODO: to write

176.1. LED detector

Pick your favourite duckiebot as the observer-bot. Refer to it as robot name for this step. If you are in good company, this can be tried on all the available duckiebots. First, activate the camera on the observer-bot:

```
$ roslaunch duckietown camera.launch veh:=robot name
```

In a separate terminal, fire up the LED detector and the custom GUI by running:

```
$ roslaunch led_detector LED_detector_with_gui.launch veh:=robot name
```

Note: to operate without a GUI:

```
$ roslaunch led_detector LED_detector.launch veh:=robot name
```

The LED_detector_node will be launched on the robot, while LED_visualizer (a simple GUI) will be started on your laptop. Make sure the camera image from the observer-bot is visualized and updated in the visualizer (tip: check that your camera cap is off).

Hit on Detect and wait to trigger a detection. This will not have any effect if LED_detector_node is not running on the duckiebot (it is included in the above launch file). After the capture and processing phases, the outcome will look like:

The red numbers represent the frequencies directly inferred from the camera stream, while the selected detections with the associated signaling frequencies will be displayed in green. You can click on the squares to visualize the brightness signals and the Fourier amplitude spectra of the corresponding cells in the video stream. You can also click on the camera image to visualize the variance map.

176.2. Unit tests

To run the unit tests for the LED detector, you need to have the F23 rosbags on you hard disk. These bag files should be synced from [this dropbox link] (https://www.dropbox.com/sh/5kx8qwgttu69fhr/AAASLpOVjV5r1xpzeW7xWZh_a?dl=0).

For the test to locate the bag files, you should have the `DUCKIETOWN_DATA` environment variable set, pointing to the location of you duckietown-data folder. This can be achieved by:

```
$ export DUCKIETOWN_DATA=local-path-to-duckietown-data-folder
```

All the available tests are specified in file `all_tests.yaml` in the scripts/ folder of the package led_detection in the duckietown ROS workspace. To run these, use the command:

```
$ rosrun led_detection unittests.py algorithm name-of-test
```

Currently, `algorithm` can be either ‘baseline’ or ‘LEDDetector_plots’ to also display the plot in the process.

To run all test with all algorithms, execute:

```
$ rosrun led_detection unittests.py '*' '*'
```

More in general:

```
$ rosrun led_detection unittests.py tests algorithms
```

where:

- `tests` is a comma separated list of algorithms. May use “*”.
- `algorithms` is a comma separated list of algorithms. May use “*”.

The default algorithm is called “`baseline`”, and its tests are invoked using:

```
$ rosrun led_detection <script> '*' 'baseline'
```

Chapter 177
Package `led_emitter`

The coordination team will use 3 signals: CAR_SIGNAL_A, CAR_SIGNAL_B, CAR_SIGNAL_C. To test the LED emitter with your joystick, run the following command:

```
$ roslaunch led_joy_mapper led_joy_with_led_emitter_test.launch veh:=robot name
```

This launches the joy controller, the mapper controller, and the led emitter nodes. You should not need to run anything external for this to work. Use the joystick buttons A, B and C to change your duckiebot's LED's blinking frequency.

Button A broadcasts signal CAR_SIGNAL_A (2.8hz), button B broadcasts signal CAR_SIGNAL_B (4.1hz), and button CAR_SIGNAL_C (Y on the controller) broadcasts signal C(5hz).The LB button will make the LEDs all white, the RB button will make some LEDs blue and some LEDs green, and the logitek button (middle button) will make the LEDs all red

Repeat this for each vehicle at the intersection that you wish to be blinking. Use previous command replacing `robot name` the names of the vehicles and try command different blinking patterns on different duckiebots.

(optional tests) For a grasp of the low level LED emitter, run:

```
$ roslaunch led_emitter led_emitter_node.launch veh:=robot name
```

You can then publish to the topic manually by running the following command in another screen on the duckiebot:

```
$ rostopic pub /robot name/led_emitter_node/change_to_state std_msgs/Float32 float-value
```

Where `float-value` is the desired blinking frequency, e.g. 1.0, .5, 3.0, etc. If you wish to run the LED emitter test, run the following:

```
$ roslaunch led_emitter led_emitter_node_test.launch veh:=robot name
```

This will cycle through frequencies of 3.0hz, 3.5hz, and 4hz every 5 seconds. Once done, kill everything and make sure you have joystick control as described above.

Chapter 178
Package `led_interpreter`

TODO: to write

Chapter 179
Package `led_joy_mapper`

TODO: to write

Chapter 180
Package `rgb_led`

TODO: to write

180.1. Demos

To test the traffic light:

```
$ rosrun rgb_led blink trafficlight4way
```

Fancy test:

```
$ rosrun rgb_led blink trafficlight4way
```

To do other tests:

```
$ rosrun rgb_led blink
```

Chapter 181
Package traffic_light

TODO: to write

Part 17
Packages - Additional functionality

TODO: to write

PART 18

Packages - Templates

These are templates.

CHAPTER 182

Package `pkg_name`

The package `pkg_name` is a template for ROS packages.
For the tutorial, see Chapter 112.

182.1. Status

Given an honest assessment of the status of this package.

Chapter 183
Package rostest_example

TODO: to write

Part 19

Packages - Convenience

TODO: to write

Chapter 184
Package `duckietown_demos`

TODO: to write

Chapter 185
Package `duckietown_unit_test`

TODO: to write

Part 20

Packages - To sort

We need to decide where these packages go.

Chapter 186
Package `adafruit_imu`

186.1. Testing

To test run:

TODO

186.2. Dependencies

- `rospy`
- `sensor_msgs`: for the `Joy.msg`
- `duckietown_msgs`: for the `CarControl.msg`

186.3. Node `adafruit_imu`

This node reads sensor data from adafruit IMU and publishes it to `sensor_msgs.Imu` and `sensor_msgs.MagneticField`.

1) Parameters

- `~pub_timestep`: Time steps (in seconds) between publishings of `CarControl` msgs. Default to 0.02 (50 Hz).

2) Publish Topics

- `~adafruit_imu`: sensor_msgs.Imu

`Imu.angular_velocity`: Vector3 of angular velocity vector.

`Imu.linear_acceleration`: Vector3 of linear acceleration.

- `~adafruit_mag`: sensor_msgs.MagneticField

`MagneticField.magnetic_field`: Vector3 of magnetic field.

3) Services

None

Chapter 187
Package duckie_rr_bridge

TODO: to write

CHAPTER 188

Package `duckiebot_visualizer`

TODO: to write

Chapter 189
Package duckietown_logs

TODO: to write.

Until we fix the dependencies:

```
sudo pip install SystemCmd==1.2 ros_node_utils==1.0 ConfTools==1.8 QuickApp==1.2.2

sudo apt-get install -y  mplayer mencoder

sudo add-apt-repository ppa:mc3man/trusty-media
sudo apt-get update
sudo apt-get install -y ffmpeg gstreamer0.10-ffmpeg
```

Chapter 190

Package `bag_stamper`

TODO: to write

Chapter 191
Package `kinematics`

TODO: to write

Chapter 192
Package visual_odometry

TODO: to write

Chapter 193

Package `mdoap`

TODO: to write

Chapter 194
Package parallel_autonomy

TODO: to write

Chapter 195
Package scene_segmentation

TODO: to write

Chapter 196
Package `veh_coordinator`

TODO: to write

Chapter 197
Package `vehicle_detection`

TODO: to write

Chapter 198

Package visual_odometry_line

TODO: to write

PART 21

Packages - Failed projects

These packages are abandoned failed projects.

Chapter 199

Package mouse_encoder

Use a mouse as encoder. Requires read permission to `/dev/input/mice`.

199.1. Publish Topic

- `mouse_encoder/tick`: `geometry_msg/Point` message with number of ticks in the x and y direction.

199.2. Parameters

- `~dev_path`: Default to `/dev/input/mice`. Point to the device path of the mouse.

199.3. Getting access to `/dev/input/mice`.

- Create a group named `input`

```
$ sudo groupadd input
```
```
* Add yourself to the `input` group
```
```
$ sudo adduser your_user_name input
```

- Log out and log back in for the change to take effect
- Put all devices under `/dev/input/` into the `input` group to grant the group read/write permission. Can be done by adding a file name `99-pure-data.rules` under `/etc/udev/rules.d` with the following line:

```
SUBSYSTEM=="input", GROUP="input", MODE="660"
```

- Reboot for the rule to take effect.

CHAPTER 200

Package `simcity` - Map Editor Version 0.1

All ./ references point to duckietown(Software)/catkin_ws/src/simcity A good reference for duckietown packages in general is catkin_ws/src/pkg_name/howto.md

200.1. How to run the map editor

(V0.1)

Inside ./launch is basic_map_tiler.launch. Ensure that the map file's path is correct for your machine. We start simcity, given a specific map file. We also start rviz, ROS' common gui.

200.2. How to edit the map

(V0.1)

The map is contained in ./maps as a YAML file. map.yaml is a small example of some circular streets, and censi_map.yaml is the map of the duckietown currently up and running.

200.3. What am I looking at, anyway?

(V0.1)

The magenta arrows point in the direction of traffic. These arrows are lines indicating where traffic flows.

200.4. What else is there to do?

(V0.1)

Lots of things. This part is mostly for rmata (1/11/16)

(1) Beautify it. Arrows are straight and ugly. Roads in duckietown can be curved, have not-so-subtle lane markings, stop signs, grass, and potentially cones and duckies.....

(2) Make it interactive. MarkerArrays consist of Markers. What does an InteractiveMarker consist of?

(3) Establish consistency and validation when adding a tile to a map. This would involve: - checking node positions at adjacent tiles - multiplying the sparse lane matrices and checking that number of lanes is consistent - having a perhaps separate node receive messages from the interactive server, or the basic_map_tiler node, and doing these computations for each change

Chapter 201

Package `slam`

TODO: to write

Chapter 202
Package street_name_detector

TODO: to write

Page left blank