Part A

Fall 2017

Welcome to the Fall 2017 Duckietown experience.

Unit A-1

The Fall 2017 Duckietown experience

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.

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 other classes.

1.1. 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.

UNIT A-2

First Steps in Duckietown

2.1. Onboarding Procedure

Welcome aboard! We are so happy you are joining us at Duckietown!

This is your onboarding procedure. Please read all the steps and then complete all the steps.

If you do not follow the steps in order, you will suffer from unnecessary confusion.

1) Github sign up

If you don't already have a Github account, sign up now.

➡ Github signup page

Please use your full name when it asks you. Ideally, the username should be something like `FirstLast` or something that resembles your name.

When you sign up, use your university email. This allows to claim an educational discount that will be useful later.

2) Questionnaire

Next, fill in this questionnaire:

Preliminary Student Questionnaire

Zurich: Please fill in questionnaire by Tuesday, September 26, 15:00 (extended from original deadline of 12:00).

Point of contact: if you have problems with this step, please contact Jacopo Tani <tanij@ethz.ch>.

3) Accept invite to Github organization Duckietown

After we receive the questionnaire, we will invite you to the Duckietown organization. You need to accept the invite; until you do, you are not part of the Duckietown organization and can't access our repositories.

The invite should be waiting for you at this page.

4) Accept the invite to Slack

After we receive the questionnaire, we will invite you to Slack.

The primary mode of online confabulation between staff and students is Slack, a team communication forum that allows the community to collaborate in making Duckietown awesome.

(Emails are otherwise forbidden, unless they relate to a private, university-based administrative concern.)

We will send you an invite to Slack. Check your inbox.

If after 24 hours from sending the questionnaire you haven't received the invite, contact HR representative Kirsten Bowser <akbowser@gmail.com>.

What is Slack? More details about Slack are available here (master). In particular, remember to disable email notifications.

Slack username. When you accept your Slack invite, please identify yourself with first and last names followed by a "-" and your institution.

example Andrea Censi - Zurich

Slack picture. Please add a picture (relatively professional, with duckie accessories encouraged).

Slack channels. A brief synopsis of all the help-related Slack channels is here: Unit A-9 - Slack Channels.

Check out all the channels in Slack, and add yourself to those that pertain or interest you. Be sure to introduce yourself in the General channel.

2.2. (optional) Add Duckietown Engineering Linkedin profile

This is an optional step.

If you wish to connect with the Duckietown alumni network, on LinkedIn you can join the company "Duckietown Engineering", with the title "Vehicle Autonomy Engineer in training". Please keep updated your Linkedin profile with any promotions you might receive in the future.

2.3. Laptops

If you do not have access to a laptop that meets the following requirements, please post a note in the channel `#help-laptops`.

You need a laptop with these specifications:

- Linux Ubuntu 16.04 installed natively (dual boot), not in a virtual machine. See Subsection 2.3.1 - Can I use a virtual machine instead of dual booting? below for a discussion of the virtual machine option.
- A WiFi interface that supports 5 GHz wireless networks. If you have a 2.4 GHz WiFi, you will not be able to comfortably stream images from the robot; moreover, you will need to adapt certain instructions.
- Minimum 50 GB of free disk space in addition to the OS. Ideally you have 200 GB+. This is for storing and processing logs.

There are no requirements of having a particularly good GPU, or a particularly good CPU. You will be developing code that runs on a Raspberry PI. Any laptop bought in the last 3 years should be powerful enough. However, having a good CPU / lots of RAM makes it faster to run regression tests.

1) Can I use a virtual machine instead of dual booting?

Running things in a virtual machine is possible, but **not supported**.

This means that while there is a way to make it work (in fact, Andrea develops in a VMWare virtual machine on OS X), we cannot guarantee that the instructions will work on a virtual machine, and, most importantly, the TAs will *not* help you debug those problems.

The issues that you will encounter are of two types.

- There are performance issues. For example, 3D acceleration might not work in the virtual machine.
- Most importantly, there are network configuration issues. These come up late in the class, when you start connecting the laptop to the Duckiebot. At that point, ROS makes certain assumptions about subnets, that might not be satisfied by your virtual machine configuration. At that point, you need to be relatively skilled to fix it.

So, the required skill here is not "being able to install Ubuntu on a virtual machine", but rather "Being able to debug network problems involving multiple real/virtual networks and multiple real/virtual adapters".

Here's a quiz: do these commands look familiar to you?

```
$ route add default gw 192.168.1.254 eth0
$ iptables -A FORWARD -o eth1 -j ACCEPT
```

If so, then things will probably work ok for you.

Otherwise, we strongly suggest that you use dual booting instead of a virtual machine.

2.4. Next steps for people in Zurich

1) Get acquainted with class journal and class logistics

At this point, you should be all set up, able to access our Github repositories, and, most important of all, able to ask for help on Slack.

You can now get acquainted to the class journal, to know the next steps.

➡ Unit A-10 - Zürich branch diary

Also, in this page, we will collect the logistics information (lab times, etc.).

➡ Unit A-3 - Logistics for Zürich branch

2) Make sure you can edit the Duckuments

To receive your Duckiebox on Wednesday Sep 27, you need to prove to be able to edit the Duckuments successfully.

➡ See the instructions in this section.

If you can't come on Wednesday, please contact one of the TAs.

2.5. Next steps for people in Chicago

TODO: to write

Unit A-3
Logistics for Zürich branch

This section describes information specific to Zürich.

1) The local staff

These are the local TAs:

- Shiying Li (shili@student.ethz.ch)
- Ercan Selçuk (ercans@student.ethz.ch)
- Miguel de la Iglesia Valls (dmiguel@student.ethz.ch)
- Harshit Khurana (hkhurana@student.ethz.ch)
- Dzenan Lapandic (ldzenan@student.ethz.ch)
- Marco Erni (merni@ethz.ch)

Please contact them on Slack, rather than email.

Also feel free to contact the TAs in Montreal and Chicago.

3.1. HR

Feel free to contact Ms. Kirsten Bowser (akbowser@gmail.com) if you have problems regarding accounts, permissions, etc.

3.2. Website / class journal

During the term, we are not going to update the website.

Rather, all important information, such as deadlines, is in the class journal.

➡ Unit A-10 - Zürich branch diary

3.3. Duckiebox

The point of contact for Duckiebox distribution is Shiying Li.

3.4. Duckietown room access

The local Duckietown room is ML J 44.2.

TODO: write opening hours and rules

+ comment
double check the room number -AC

3.5. Extra spaces

There will be extra lab space available.

Space-time coordinates TBD.

UNIT A-4

Logistics for Montréal branch

This unit contains all necessary info specific for students at Univeristé de Montréal.

4.1. Website

This is the official course website. It contains links to the the syllabus and description and other important info.

4.2. Class Schedule

The authoritative class schedule will be tracked in Unit A-11 - Montréal branch diary. This will contain all lecture material, homeworks, checkoffs, and labs.

4.3. Lab Access

> The lab room for the class is 2333 in Pavillion André-Aisenstadt. The code for the door is XXX. Please do not distribute the code for the door, we are trying to limit access to this room as much as possible.

4.4. The Local Staff

The TA for the class is Florian Golemo. All communications with the course staff should happen through Slack.

The instructor is Prof. Liam Paull, whose office is 2347 Pavillion André-Aisenstadt.

4.5. Storing Your Robot

It is preferable that you keep your robot for the semester. However, if you do not have a secure location where you can store it, we can store it for you in Room XXX in Pavillion André-Aisenstadt. However, you will have to ask Prof. Liam Paull to access or store your robot there each time since we cannot give out access to this space to the students in the class.

UNIT A-5

Logistics for Chicago branch

Assigned to: Matt

This section describes information specific to TTIC and U. Chicago students.

1) Website

The course website provides a copy of the syllabus, grading information, and details on learning objectives.

2) Class Schedule

Classes take place on Mondays and Wednesdays from 9am-11am in TTIC Room 530. In practice, each class will be divided into an initial lecture period (approximately one hour), followed by a lab session.

The class schedule is maintained as part of the TTIC Class Diary, which includes details on lecture topics, links to slides, etc.

3) Lab Access

Duckietown labs will take place at TTIC in the robotics lab on the 4th floor.

Note: TTIC and U. Chicago students in Matthew Walter's research group use the lab as their exclusive research and office space. It also houses several robots and hardware to support them. Please respect the space when you use it: try not to distract lab members while they are working and please don't touch the robots, sensors, or tools.

4) The Local LAs

Duckietown is a collaborative effort involving close interaction among students, TAs, mentors, and faculty across several institutions. The local learning assistants (LAs) at TTIC are:

- Andrea F. Daniele (afdaniele@ttic.edu)
- Falcon Dai (dai@ttic.edu)
- Jon Michaux (jmichaux@ttic.edu)

UNIT A-6

Git usage guide for Fall 2017

6.1. Repositories

These are the repositories we use.

1) Software

The Software repository is the main repository that contains the software.
The URL to clone is:

```
git@github.com:duckietown/Software.git
```

In the documentation, this is referred to as `DUCKIETOWN_ROOT`.
During the first part of the class, you will only read from this repository.

2) Duckiefleet

The duckiefleet-fall2017 repository contains the data specific to this instance of the class.
The URL to clone is:

```
git@github.com:duckietown/duckiefleet-fall2017.git
```

In the documentation, this is referred to as `DUCKIEFLEET_ROOT`.
You will be asked to write to this repository, to update the robot DB and the people DB, and for doing exercises.

3) Duckuments

The Duckuments repository is the one that contains this documentation.
The URL to clone is:

```
git@github.com:duckietown/duckuments.git
```

Everybody is encouraged to edit this documentation!
In particular, feel free to insert comments.

4) Lectures

The lectures repository contains the lecture slides.
The URL to clone is:

```
git@github.com:duckietown/lectures.git
```

Students are welcome to use this repository to get the slides, however, please note that this is a space full of drafts.

5) Exercises

The exercises repository contains the solution to exercises.
The URL to clone is:

```
git@github.com:duckietown/XX-exercises.git
```

Only TAs have write permissions to this repository.

6.2. Git policy for homeworks

Homeworks will require you to write and submit coding exercises. They will be submitted using git. Since we have a university plagiarism policy (UdeM's) we have to protect students work before the deadline of the homeworks. For this reason we will follow these steps for homework submission:

1. Go here and file a request at the bottom "Request a Discount" then enter your institution email and other info.
2. Go to duckiefleet-fall2017
3. Click "Fork" button in the top right
4. Choose your account if there are multiple options
5. Click on the Settings tab
6. Under "Teams", click the "X" in the right for the section for "Fall 2017 Students". You will get a popup asking you to confirm. Confirm.

Now you need to point the remote of your `duckiefleet-fall2017` to your new local private repo. To do, from inside your already previously cloned `duckiefleet-fall2017` repo do:

```
$ git remote set-url origin git@github.com:GIT_USERNAME/duckiefleet-fall2017.git
```

Let's also add an `upstream` remote that points back to the original duckietown repo:

```
$ git remote add upstream git@github.com:duckietown/duckiefleet-fall2017.git
```

If you type

```
$ git remote -v
```

You should now see:

```
origin  git@github.com:GIT_USERNAME/duckiefleet-fall2017.git (fetch)
origin  git@github.com:GIT_USERNAME/duckiefleet-fall2017.git (push)
upstream  git@github.com:duckietown/duckiefleet-fall2017.git (fetch)
upstream  git@github.com:duckietown/duckiefleet-fall2017.git (push)
```

Now the next time you push (without specifying a remote) you will push to your local private repo.

1) Duckiefleet file structure

You should put your homework files in folder at:

```
DUCKIEFLEET_ROOT/homeworks/XX_homework_name/YOUR_ROBOT_NAME
```

Some homeworks might not require ROS, they should go in a subfolder called `scripts`. ROS homeworks should go in packages which are generated using the process described here: Minimal ROS node - `pkg_name` (master). For an example see `DUCKIEFLEET_ROOT/homeworks/01_data_processing/shamrock`.

> **Note:** To make your ROS packages findable by ROS you should add a symlink from your `duckietown/catkin_ws/src` directory to `![$DUCKIEFLEET_ROOT`

2) To submit your homework

When you are ready to submit your homework, you should do **create a tag** and **tag the Fall 2017 instructors/TAs group** to let us know that your work is complete. This can be done through the command line or through the github web interface:

Command line:

```
git tag XX_homework_name -m"@duckietown/fall-2017-instructors-and-tas homework complete"
git push origin --tags
```

Through github:

1. Click on the "Releases" tab
2. Click "Create a new Release"
3. Add a version (e.g. 1.0)
4. Release title put `XX_homework_name`
5. In the text box put "@duckietown/fall-2017-instructors-and-tas homework complete"
6. Click "Publish release"

You may make as many releases as you like before the deadline.

3) Merging things back

Once all deadlines have passed for all institutions, we can merge all the homeworks. We will ask to create a "Pull Request" from your private repo.

1. In your private duckiefleet-fall2017 repo, click the "New pull request button".
2. Click "Create pull request" green button
3. The 4 drop down menus at the top should be left to right: (`base fork: duckietown/duckiefleet-fall2017`, `base: master`, `head fork: YOUR_GIT_NAME/duckiefleet-fall2017`, `compare: YOUR_BRANCH`)
4. Leave a comment if you like and click "Create pull request" green button below.
5. At some point a TA or instructor will either merge or leave you a comment.

6.3. Git policy for project development

Different than the homeworks, development for the projects will take place in the `Software` repo since plagiarism is not an issue here. The process is:

1. Create a branch from master

2. Develop code in that branch (note you may want to branch your branches. A good idea would be to have your own “master”, e.g. “your_project-master” and then do pull requests/merges into that branch as things start to work)
3. At the end of the project submit a pull request to master to merge your code. It may or may not get merged depending on many factors.

UNIT A-7

Organization

The following roster shows the teaching staff.

Andrea Censi

Liam Paull

Jacopo Tani

First Last

Emilio Frazzoli

First Last

First Last

First Last

First Last

First Last

First Last

First Last

First Last

First Last

First Last

Greta Paull

Kirsten Bowser

Staff: To add yourself to the roster, or to change your picture, add a YAML file and a jpg file to the `duckiefleet-fall2017` repository. in the `people/staff` directory.

7.1. The Activity Tracker

Link to Activity Tracker

The sheet called "Activity Tracker" describes specific tasks that you must do in a certain sequence. Tasks include things like "assemble your robot" or "sign up on Github".

The difference between the Areas sheet and the Task sheet is that the Task sheet contains tasks that you have to do once; instead, the Areas sheet contains ongoing activities.

In this sheet, each task is a row, and each person is a column. There is one column for each person in the class, including instructors, TAs, mentors, and students.

You have two options:

- Only use the sheet as a reference;
- Use the sheet actively to track your progress. To do this, send a message to Kirsten with your gmail address, and add yourself.

Each task in the first column is linked to the documentation that describes how to perform the task.

The colored boxes have the following meaning:

- Grey: not ready. This means the task is not ready for you to start yet.
- Red: not started. The person has not started the task.
- Blue: in progress. The person is doing the task.
- Yellow: blocked. The person is blocked.
- Green: done. The person is done with the task.
- n/a: the task is not applicable to the person. (Certain tasks are staff-only.)

7.2. The Areas sheet

Please familiarize yourself with this spreadsheet and bookmark it in your browser.

The sheet called "Areas" describes the points of contact for each part of this experience. These are the people that can offer support. In particular, note that we list two points of contact: one for America, and one for Europe. Moreover, there is a link to a Slack channel, which is the place where to ask for help. (We'll get you started on Slack in just a minute.)

UNIT A-8

Getting and giving help

8.1. *Who* to ask for help

1) Primary points of contact

The organization chart (Section 7.2 - The Areas sheet) lists the primary contact for each area.

2) Point of contacts for specific documents

Certain documents have specific points of contacts, listed at the top. These override the listing in the organization chart.

8.2. *How* to ask for help

The ways that we will support each other will depend on the type of situation. Here we will enumerate the different cases. Try to figure out which case is the most appropriate and act accordingly. These are ordered roughly in order of increasing severity.

1) Case: You find a mistake in the documentation

Action: Please fix it.

The goal for the instructions is that anybody is able to follow them. Last year, we managed to have two 15-year-old students reproduce the Duckiebot from instructions.

> ➡ How to edit the documentation is explained in Part D - Duckumentation documentation. In particular, the notation on how to insert a comment is explained in Section 3.3 - Notes and questions.

Note that because we use Git, we can always keep track of changes, and there is no risk of causing damage.

If you encounter typos, feel free to edit them directly.

Feel free to add additional explanations.

One thing that is very delicate is dealing with mistakes in the instructions.

A few times the following happened: there is a sequence of commands `cmd1;cmd2;cmd3` and `cmd2` has a mistake, and `cmd2b` is the right one, so that the sequence of commands is `cmd1;cmd2b;cmd3`. In those situations we first just corrected the command `cmd2`.

However, that created a problem: now half of the students had used `cmd1;cmd2;cmd3` and half of the students had used `cmd1;cmd2b;cmd3`: the states had diverged. Now chaos might arise, because there is the possibility of "forks".

Therefore, if a mistaken instruction is found, rather than just fixing the mistake, please add an addendum at the end of the section.

For example: "Note that instruction `cmd2` is wrong; it should be `cmd2b`. To fix this, please enter then command `cmd4`".

Later, when everybody has gone through the instructions, the mistake is fixed and the addendum is deleted.

2) Case: You find the instructions unclear and you need clarification

Action: Ask for clarification on the appropriate Slack channel. For a list of slack channels that could be helpful see Unit A-9 - Slack Channels. Once the ambiguity is clarified to your satisfaction, either you or the appropriate staff member should update the documentation if appropriate. For instructions on this see Part D - Duckumentation documentation.

3) Case: You understand the instructions but you are blocked for some reason

Action: This is more serious than the previous. Open an issue on the `duckiefleet-fall2017` github page. Once the issue is resolved, either you or the appropriate staff member should update the documentation if appropriate. For instructions on this see Part D - Duckumentation documentation.

4) Case: You are having a technical issue related to building the documentation

Action: Open an issue on the `duckuments` github page and provide all necessary information to reproduce it.

5) Case: You have found a well-defined defect in the software.

Action: open an issue on the `Software` repository github page and provide all necessary information for reproducing the bug.

UNIT A-9

Slack Channels

This page describes all of the helpful Slack channels and their purposes so that you can figure out where to get help.

9.1. Channels

You can also easily join the ones that you are interested in by clicking the links in this message.

TABLE 9.1. DUCKIETOWN SLACK CHANNELS

Channel	Purpose
`help-accounts`	Info about necessary accounts, such as Slack, Github, etc.
`help-assembly`	Help putting your robot together
`help-camera-calib`	Help doing the intrinsic and extrinsic calibration of your camera
`help-duckuments`	Help compiling the online documentation
`help-git`	Help with git
`help-infrastructure`	Help with software infrastructure, such as Makefiles, unit tests, continuous integration, etc.
`help-laptops`	Help getting your laptop setup with Ubuntu 16.04
`help-parts`	Help getting the parts for the robot or replacement parts if you broke something
`help-robot-setup`	Help getting the robot setup to do basic things like be driven with a joystick
`help-ros`	Help with the Robot Operating System (ROS)
`help-wheel-calib`	Help doing your odometry calibration

+ comment

Note that we can link directly to the channels. (See list in the org sheet.) -AC

UNIT A-10

Zürich branch diary

10.1. Wed Sep 20: Welcome to Duckietown!

This was an introduction meeting.

1) Material presented in class

These are the slides we showed:

- PDF
- Keynote (huge)

2) Feedback form

Please help us making the experience better by providing feedback (can be anonymous)

3) Pointers to reading materials

Read about Duckietown's history and watch the Duckumentary.

➡ Part C - The Duckietown project

Start learning about Git and Github.

➡ Source code control with Git (master)

Montreal, Chicago? What's happening?

➡ Unit A-1 - The Fall 2017 Duckietown experience

10.2. Monday Sep 25: Introduction to autonomy

1) Material presented in class

These are the slides we presented:

a - Logistics: Keynote, PDF.

a - Autonomous Vehicles: Keynote, PDF.

c - Autonomous Mobility on Demand: Keynote, PDF.

d - Plan for the next months: Keynote, PDF.

2) Feedback form

Please help us making the experience better by providing feedback (can be anonymous)

3) Questions and answers

Q: MyStudies is not updated; I am still on the waiting list. What should I do?

Answer: Nothing. Don't worry, if you have received the onboarding email, you are in the class, even if you still appear in the waiting list. We will figure this out with the de-

partment.

Q: What version of Linux do I need to install?

Answer: 16.04.* (16.04.03 is the latest at time of this writing)

Q: Do I need to install OpenCV, ROS, etc.?

Answer: Not necessary. We will provide instructions for those steps.

Q: My laptop is not ready. I'm having problems installing Linux on a partition.

Answer: Don't worry, take a Duckie, and, take a breath. We have time to fix every issue. Start by asking for help in the #help-laptops channel in Slack. We will address the outstanding issues in the next classes.

Q: How much space do I need on my Linux partition?

Answer: At least 50 GB; 200 GB are recommended for easy processing of data (logs) later in the course. If you have less space (say ~100GB), it might be wise to acquire an external hard drive to use as storage.

Q: Are there going to be Linux training sessions?

Answer: Maybe. We didn't plan for it, but it seems that there is a need. Subject to figuring out the logistics, we might organize an extra "lab" session or produce a support video.

10.3. Monday Sep 25, late at night: Onboarding instructions

At some late hour of the night, we sent out the onboarding instructions.

➡ Section 2.1 - Onboarding Procedure

Please complete the onboarding questionnaire by Tuesday, September 26, 15:00.

10.4. Wednesday Sep 27: Duckiebox distribution, and getting to know each other

Today we distribute the Duckieboxes and we name the robots. In other words, we perform the *Duckiebox ceremony*.

- getting to know each other;
- naming the robots;
- distribute the Duckieboxes.

> **Note:** If you cannot make it to this class for the Duckiebox distribution, please inform the TA, to schedule a different time.

1) Preparation, step 1: choose a name for your robot

Before arriving to class, you must think of a name for your robot.

Here are the constraints:

- The name must work as a hostname. It needs to start with a letter, contains only letters and numbers, and no spaces or punctuation.
- It should be short, easy to type. (You'll type it a lot.)
- It cannot be your own name.
- It cannot be a generic name like "robot", "duckiebot", "car". It cannot contain brand names.

2) Preparation, step 2: prepare a brief elevator pitch

As members of the same community, it is important to get to know a little about each other, so to know who to rely on in times of need.

During the Duckiebox distribution ceremony, you will be asked to walk up to the board, write your name on it, and introduce yourself. Keep it very brief (30 seconds), and tell us:

- what is your professional background and expertise / favorite subject;
- what is the name of your robot;
- why did you choose to name your robot in that way.

You will then receive a Duckiebox from our senior staff, a simple gesture but of sempiternal glory, for which you have now become a member of the Duckietown community. This important moment will be remembered through a photograph. (If in the future you become a famous roboticist, we want to claim it's all our merit.)

Finally, you will bring the Duckiebox to our junior staff, who will apply labels with your name and the name of the robot. They will also give you labels with your robot name for future application on your Duckiebot.

3) Feedback form

Please help us making the experience better by providing feedback (can be anonymous)

4) Material presented in class

- Duckiebot parts

10.5. Thursday Sep 26: Misc announcements

We created the channel `#ethz-chitchat` for questions and other random things, so that we can leave this channel `#ethz-announcements` only for announcements.

We sent the final list to the Department; so hopefully in a couple of days the situation on MyStudies is regularized.

The "lab" time on Friday consists in an empty room for you to use as you wish, for example to assembe the robots together. In particular, it's on the same floor of the Duckietown room and the IDSC lab.

The instructions for assembling the Duckiebots are here. Note that you don't have to do the parts that we did for you: buying the parts, soldering the boards, and reproducing the image.

Expected progress: We are happy if we see everybody reaching up to Unit I-12 - RC+camera remotely by Monday October 9. You are encouraged to start very early; it's likely that you will not receive much help on Sunday October 8...

10.6. Oct 02 (Mon)

Assigned to: XXX

- Autonomy Architectures
- System Architecture Basics
- representations - knowledge representation: tasks, goal

10.7. Oct 04 (Wed)

- Software architecture and middleware.
- Intro to ROS
- Networking
- Modern signal distribution and processing: periodic vs event-based processing; latency/frequency etc.

Assigned to: XXX

10.8. Oct 09 (Mon)

Assigned to: XXX

10.9. Oct 11 (Wed)

Assigned to: XXX

10.10. Oct 16 (Mon)

Assigned to: XXX

10.11. Oct 18 (Wed)

Assigned to: XXX

10.12. Oct 23 (Mon)

Assigned to: XXX

10.13. Oct 25 (Wed)

Assigned to: XXX

(Andrea traveling)

10.14. Oct 30 (Mon)

Assigned to: XXX

10.15. Nov 01 (Wed)

Assigned to: XXX

10.16. Nov 06 (Mon)

Assigned to: XXX

10.17. Nov 08 (Wed)

Assigned to: XXX

10.18. Nov 13 (Mon)

Assigned to: XXX

10.19. Nov 15 (Wed)

Assigned to: XXX

10.20. Nov 20 (Mon)

Assigned to: XXX

10.21. Nov 22 (Wed)

Assigned to: XXX

10.22. Nov 27 (Mon)

Assigned to: XXX

10.23. Nov 29 (Wed)

Assigned to: XXX

10.24. Dec 04 (Mon)

Assigned to: XXX

10.25. Dec 06 (Wed)

Assigned to: XXX

10.26. Dec 11 (Mon)

Assigned to: XXX

10.27. (Template for every lecture) Date: Topic

Assigned to: Name of TA

1) Preparation

Things that the students should do before class.

2) Material presented in class

Link to PDF and Keynote/Powerpoint materials.

3) Pointers to reading materials

Links to the units mentioned in the slides, and additional materials.

4) Questions and answers

Write here the FAQs that students have following the lecture or instructions.

10.28. (Template for every lecture) Date: Topic

> Assigned to: Name of TA

1) Preparation

Things that the students should do before class.

2) Material presented in class

Link to PDF and Keynote/Powerpoint materials.

3) Pointers to reading materials

Links to the units mentioned in the slides, and additional materials.

4) Questions and answers

Write here the FAQs that students have following the lecture or instructions.

Unit A-11

Montréal branch diary

11.1. Wed Sept 6

Class (11:30)

Slides:

- Duckietown History Future (keynote) (pdf)
- Duckietown Intro (keynote) (pdf)
- Autonomy Overview (keynote) (pdf)
- Autonomous Vehicles (keynote) (pdf)

Book materials:

- Part C - The Duckietown project
- Unit G-1 - Autonomous Vehicles [draft]
- Unit G-2 - Autonomy overview

11.2. Friday Sept 8

Acceptance emails sent

11.3. Sun Sept 10

- Onboarding email sent to accepted students

11.4. Mon Sept 11

Class (10:30)

> **Note:** This class we are meeting in rm. 2333 Pavillion André Aisenstadt.

- Logistics
- The Duckiebook
- Slack (Unit A-9 - Slack Channels)
- Git repos (Unit A-6 - Git usage guide for Fall 2017)
- How to get and give help (Unit A-8 - Getting and giving help)
- Grading scheme
- Student/Staff Introductions
- Duckiebox distribution.
- Go through the Duckiebox parts
- Unit B-1 - Checkoff: Duckiebot Assembly and Configuration initiated.

Deadline: **Mon Sept. 25**

11.5. Wed Sept 13

Class canceled. Continue working on Unit B-1 - Checkoff: Duckiebot Assembly and Configuration.

11.6. Mon Sept 18

Class (10:30 - 11:30)

- General discussion (how are things going? Sorry I was away last week.. anyone need anything?)
- Intro to robotics - Modern robotic systems
- The robot as a system - System architectures
- Decomposing the robotics sytems into smaller pieces - autonomy architectures
- Agreeing on the language that the different pieces “talk” - representations
- Background on basic probability theory?

Slides:

- Modern Robotic Systems (keynote) (pdf)
- System Architecture (keynote) (pdf)
- Autonomy Architectures (keynote) (pdf)
- Representations for Robotics (keynote) (pdf)

Book Materials:

- Modern Robotic Systems
- System Architecture Basics (master)
- Autonomy Architectures

- Representations
- Probability Basics

Lab (11:30 - 12:30)

- Liam will be in 2333 setting up network and building Duckietown.

11.7. Wed Sept 20

Class (11:30 - 12:30)

- Robotics middlewares - what are they and basic concepts
- Introduction to the Robot Operating System (ROS)

Slides:

- Software Architectures (keynote) (pdf)
- Introduction to ROS (keynote) (pdf)

Book Materials:

- Introduction to ROS (master)
- Middlewares (master)

Lab (12:30 - 1:30)

Homeworks and Checkoffs:

- Unit B-1 - Checkoff: Duckiebot Assembly and Configuration deadline extended to **Monday Sept 25**. Submit by clicking here.

11.8. Mon Sept 25

Class (10:30-11:30)

- Microelectronics (pptx) (pdf)
- Modern Signal Processing (keynote) (pdf)
- Intro to Networking (keynote) (pdf)

Book Materials (still rough drafts, will be completed soon):

Lab (11:30 - 12:30)

- Liam will be in 2333
- Any final help needed for Unit B-1 - Checkoff: Duckiebot Assembly and Configuration
- Finalize Duckietown map setup

Homeworks and Checkoffs:

- Unit B-1 - Checkoff: Duckiebot Assembly and Configuration due by 11pm.
- New checkoff: “Taking a log” will be linked later today (due Monday Oct 2)
- New homework: “Data processing” will be linked later today (due Monday Oct 2)

11.9. Wed Sept 27

Class (11:30 - 12:30) in Z-305

- USB drive distribution
- New checkoff announced: Unit B-2 - Checkoff: Take a Log
- New homework announced: Unit B-3 - Homework: Data Processing
- Let’s review the git policy for homeworks: Section 6.2 - Git policy for homeworks

- Announcement regarding activity spreadsheet which is now embedded here: Section 7.1 - The Activity Tracker. Feel free to just look or send Kirsten your gmail on Slack and she will give you access to it.

Slides:

- Duckiebot Modeling (pptx) (pdf)

Book Material:

- Coordinate systems (master)
- Reference frames (master)
- Unit G-6 - Duckiebot modeling

11.10. Mon Oct 2

Class (10:30 - 11:30) in Z-210

Slides:

- Computer vision basics: (keynote) (pdf)

Book Material:

- Unit G-7 - Computer vision basics [draft]
- Unit G-8 - Camera geometry [draft]
- Unit G-9 - Camera calibration [draft]
- Unit G-10 - Image filtering [draft]

Lab (11:30 - 12:30) in AA 2333

11.11. Wed Oct 4

Class (11:30 - 12:30) in Z-305

- Computer Vision - Illumination invariance

Lab (12:30 - 1:30) in AA 2333

> Note: Checkoff and Homework due at 11pm

UNIT A-12

Chicago branch Diary

Classes take place on Mondays and Wednesdays from 9am-11am in TTIC Room 530.

12.1. Monday September 25: Introduction to Duckietown

1) Lecture Content

- Duckietown Course Intro (Keynote, PDF)

2) Feedback Form

Please help us making the experience better by providing feedback (can be anonymous)

3) Reading Material

- Part C - The Duckietown project
- Unit G-1 - Autonomous Vehicles [draft]
- Unit G-2 - Autonomy overview

12.2. Tuesday September 26: Onboarding

Tonight, we sent out the onboarding instructions.

➡ Section 2.1 - Onboarding Procedure

Please complete the onboarding questionnaire by Thursday, September 27, 5:00pm CT

12.3. Wednesday, September 27: Duckiebox Ceremony

Welcome to Duckietown! This lecture constitutes the *Duckiebox ceremony* during which we will distribute your Duckieboxes and ask you to name your yet-to-be built Duckiebots!!! We will also discuss logistics related to the course, but we admit that that isn't as exciting.

1) Lecture Content

- The Duckiebook
- Slack (Unit A-9 - Slack Channels)
- Git repos (Unit A-6 - Git usage guide for Fall 2017)
- How to get and give help (Unit A-8 - Getting and giving help)
- Unit B-1 - Checkoff: Duckiebot Assembly and Configuration initiated
- Getting to know one another
- Naming your robots
- Distribute the Duckieboxes!

As you name your Duckiebots, please consider the following constraints:

- The name must work as a hostname. It needs to start with a letter, contains only letters and numbers, and no spaces or punctuation.
- It should be short, easy to type. (You'll type it a lot.)
- It cannot be your own name.
- It cannot be a generic name like "robot", "duckiebot", "car". It cannot contain brand names.

2) Feedback Form

Please help us making the experience better by providing feedback (can be anonymous)

12.4. (Template for every lecture) Date: Topic

What is this lecture all about?

1) Preparation

Things that the students should do before class.

2) Lecture Content

Link to PDF and Keynote/Powerpoint materials.

3) Feedback Form

Please help us making the experience better by providing feedback (can be anonymous)

4) Reading Material

Links to the units mentioned in the slides, and additional materials.

5) Questions and Answers

FAQs that students have following the lecture or instructions.

UNIT A-13

Guide for TAs

13.1. Dramatis personae

These are the TAs.

At ETHZ:

- Shiying Li (shili@student.ethz.ch)
- Ercan Selçuk (ercans@student.ethz.ch)
- Miguel de la Iglesia Valls (dmiguel@student.ethz.ch)
- Khurana Harshit (hkhurana@student.ethz.ch)
- Lapandic Dzenan (ldzenan@student.ethz.ch)
- Marco Erni (merni@ethz.ch)

At TTIC:

- Andrea F. Daniele (afdaniele@ttic.edu)
- Falcon Dai (dai@ttic.edu)
- Jon Michaux (jmichaux@ttic.edu)

At Montreal:

- Florian Golemo (fgolemo@gmail.com)

13.2. First steps

Here are the first steps for the TAs.

Note that many of these are not sequential and can be done in parallel.

1) Learn about Duckietown

Read about Duckietown's history; watch the Duckumentary.

➡ Part C - The Duckietown project

2) Online accounts

You have to set up:

- A personal Github account
- A Twist account
- A Slack account
- A Google Docs account (Gmail address)

Send an email to Kirsten Bowser (akbowser@gmail.com), with your GMail address and your Github account. She will give you further instructions.

Point of contact: Kirsten Bowser

3) Install Ubuntu

Install Ubuntu 16.04 on your laptop, and then install ROS, Atom, Liclipse, etc.

➡ Unit I-6 - Installing Ubuntu on laptops

4) Duckuments

Install the Duckuments system, so you can edit these instructions.

➡ Part D - Duckumentation documentation.

Point of contact: Andrea

5) Learn about Git and Github

Start learning about Git and Github. You don't have to read the entirety of the following references now, but keep them "on your desk" for later reference.

➡ Good book

➡ Git Flow

Point of contact: Liam?

6) Continuous integration

Understand the continuous integration system.

➡ Documentation on continuous integration (master).

Point of contact: Andrea

7) Duckiebot building

Build your Duckiebot according to the instructions.

➡ Part I - Operation manual - Duckiebot

Point of contact: Shiying (ETH)

Point of contact: ??? (UdeM)

Point of contact: ??? (TTIC)

As you read the instructions, keep open the Duckuments source, and note any discrepancies. You must note any unexpected thing that is not predicted from the instruction. If you don't understand anything, please note it.

The idea is that dozens of other people will have to do the same after you, so improving the documentation is the best use of your time, and it is much more efficient than answering the same question dozens of times.

8) Other documentation outside of the Duckuments

We have the following four documents outside of the duckuments:

1. Organization chart: This is where we assign areas of responsibility.
2. Lecture schedule
3. Checkoff spreadsheet
4. The big TODO list: Where we keep track of things to do.

UNIT A-14

Guide for mentors [draft]

This section has been removed because it is in status 'draft'. If you are feeling adventurous, you can read it on master.

To disable this behavior, and completely hide the sections, set the parameter show_removed to false in fall2017.version.yaml.

UNIT A-15

Project proposals [draft]

This section has been removed because it is in status 'draft'. If you are feeling adventurous, you can read it on master.

To disable this behavior, and completely hide the sections, set the parameter show_removed to false in fall2017.version.yaml.

UNIT A-16

Template of a project [draft]

This section has been removed because it is in status 'draft'. If you are feeling adventurous, you can read it on master.

To disable this behavior, and completely hide the sections, set the parameter show_removed to false in fall2017.version.yaml.

PART B

Fall 2017 Checkoffs and Homeworks

UNIT B-1

Checkoff: Duckiebot Assembly and Configuration

The first job is to get your Duckiebot put together and up and running.

1.1. Pick up your Duckiebox

Slack channel: #help-parts

There is a checklist inside. You should go through the box and ensure that the parts that are supposed to be in it actually are inside.

If you are missing something, contact your local responsible and ask for help on Slack in the appropriate channel.

These sections describe the parts that are in your box.

- Unit I-2 - Acquiring the parts for the Duckiebot `DB17-wjd`
- Unit J-1 - Acquiring the parts for the Duckiebot `DB17-lc`

1.2. Soldering your boards

Depending on how kind your instructors/TAs are, you may have to solder your boards.

- Soldering boards for `DB17` (master)
- Unit J-2 - Soldering boards for `DB17-l` [draft]

1.3. Assemble your Robot

Slack channel: #help-assembly

You are ready to put things together now.

- Unit I-4 - Assembling the Duckiebot `DB17`, `DB17-jwd`
- Unit J-4 - Bumper Assembly [draft]
- Unit J-3 - Assembling the Duckiebot `DB17-wjdlc` [draft]

1.4. Optional: Reproduce the SD Card Image

If you are very inexperienced with Linux/Unix/networking etc, then you may find it a valuable experience to reproduce the SD card image to "see how the sausage is made".

➡ Reproducing the image (master)

1.5. Setup your laptop

Slack channel: #help-laptops

The only officially supported OS is Ubuntu 16.04. If you are not running this OS it is recommended that you make a small partition on your hard drive and install the OS.

Related parts of the book are:

➡ How to make a partition (master) if you want to make a partition

➡ Unit I-6 - Installing Ubuntu on laptops

1.6. Make your robot move

Slack channel: #help-robot-setup

Now you need to clone the software repo and run things to make your robot move.

First initialize the robot:

➡ Unit I-7 - Duckiebot Initialization

Then get it to move!

➡ Unit I-9 - Software setup and RC remote control

+ comment

there is a deadline at this link, but each institution will have different deadlines.
Might be a source of entropy -JT

Unit B-2

Checkoff: Take a Log

Knowledge and activity graph

Requires: Unit B-1 - Checkoff: Duckiebot Assembly and Configuration

Results: A verified log in rosbag format updoaded to Dropbox

Slack channel: #help-logging

Montreal deadline: Oct 4, 11:00pm

2.1. Mount your USB drive

We will log to the USB drive that you were given.

➡ Mounting USB drives (master)

2.2. Take a Log

Take a 5min log as you drive in Duckietown (For Montreal this is rm. 2333).

➡ Unit I-15 - Taking a log for detailed instructions.

2.3. Verify your log

➡ Unit I-16 - Verify a log for detailed instructions.

2.4. Upload the log

Upload the log here

UNIT B-3

Homework: Data Processing

KNOWLEDGE AND ACTIVITY GRAPH

Requires: Unit B-2 - Checkoff: Take a Log
Requires: ROS installation and reference (master)
Results: Ability to perform basic operations on images
Results: Build your first ROS package and node
Results: Ability to process imagery live

Slack channel: #help-data-processing
Montreal deadline: Oct 4, 11:00pm

3.1. Follow the git policy for homeworks

➡ Section 6.2 - Git policy for homeworks

for instructions on how the homework should be submitted.

3.2. Exercise: Basic image operations

Complete Unit E-1 - Exercise: Basic image operations, adult version

3.3. Exercise: Log decimation

Complete Unit E-2 - Exercise: Bag in, bag out

3.4. Exercise: Instagram filters

Complete Unit E-3 - Exercise: Instagram filters

3.5. Exercise: Live Instagram

Complete Unit E-4 - Exercise: Live Instagram

Call your package `dt-instagram-live_ROBOT_NAME` and call your node `dt-instagram-live_ROBOT_NAME`

When you are done, take a 5min log (See Unit I-15 - Taking a log)in Duckietown (2333 in Montreal) and upload here

PART C
The Duckietown project

UNIT C-1
What is Duckietown?

1.1. Goals and objectives

Duckietown is a robotics education and outreach effort.

The most tangible goal of the project is to provide a low-cost educational platform for learning about autonomous systems, consisting of lectures and other learning material, the Duckiebot autonomous robots, and the Duckietowns, which constitute the infrastructure in which the Duckiebots navigate.

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 people 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 provide a platform that can be adapted to a range of different grade and experience levels.
2. For **self-guided learners**, we want to create a "self-learning experience" that allows students to go from having zero knowledge of robotics to a graduate-level understanding.

In addition, the Duckietown platform is also suitable for research.

1.2. Learn about the Duckietown educational experience

The video in Figure 1.1 is the "Duckumentary", a documentary about the first version of the class, during Spring 2016.

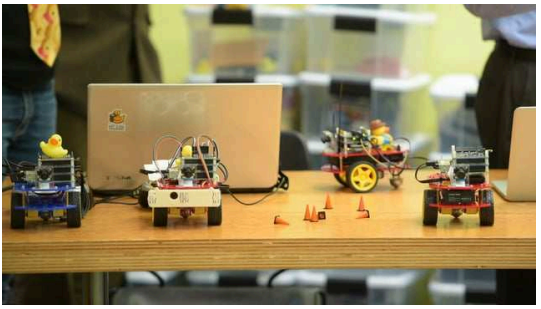


Figure 1.1. The Duckumentary, created by Chris Welch.

The video in Figure 1.2 is a documentary created by Red Hat on the current developments in self-driving cars.

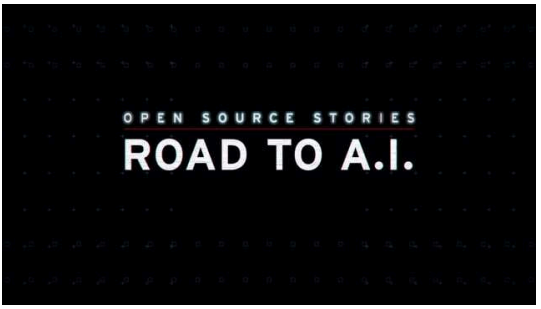


Figure 1.2. The road to autonomy

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 video in Figure 1.3 shows some of the functionality of the platform.

If you would like to know more, the paper [2] describes the Duckiebot and its software. (With 30 authors, we made the record for a robotics conference!)

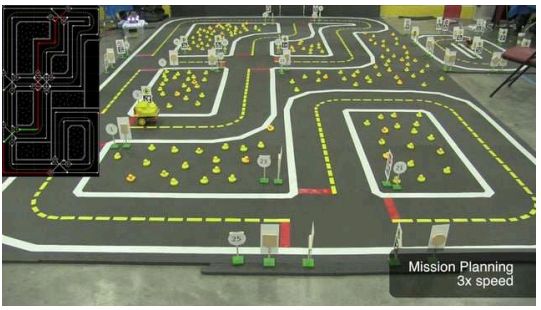


Figure 1.3. Duckietown functionality

UNIT C-2

Duckietown history and future

2.1. The beginnings of Duckietown

The original Duckietown class was at MIT in 2016 (Figure 2.1).

Figure 2.1. Part of the first MIT class, during the final demo.

Duckietown was built by elves (Figure 2.2).

Figure 2.2. The elves of Duckietown

These are some advertisement videos we used.

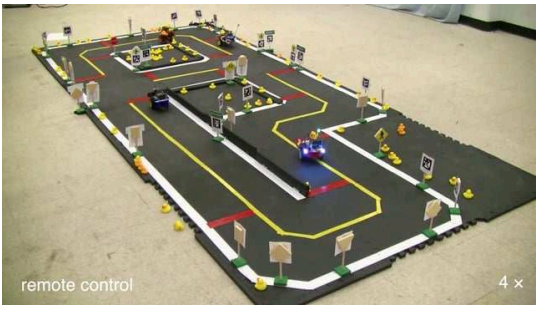


Figure 2.3. The need for autonomy

Figure 2.4. Advertisement

Figure 2.5. Cool Duckietown by night

2.2. University-level classes in 2016

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

- National Chiao Tung University 2016 (master), Taiwan - Prof. Nick Wang;
- Tsinghua University (master), People's Republic of China - Prof. (Samuel) Qing-Shan Jia's *Computer Networks with Applications* course;
- Rennselaer Polytechnic Institute 2016 (master) - Prof. John Wen;

Figure 2.6. 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 (master) - Prof. Emilio Frazzoli, Dr. Andrea Censi;
- University of Montreal, 2017 (master) - Prof. Liam Paull;
- TTI/Chicago 2017 (master) - Prof. Matthew Walter; and
- National Chiao Tung University, Taiwan - Prof. Nick Wang.

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

- Rennselaer Polytechnic Institute (Jeff Trinkle)
- National Chiao Tung University, Taiwan - Prof. Yon-Ping Chen's *Dynamic system simulation and implementation* course.
- 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; and
- Yuan Zhu University, Taiwan - Prof. Kan-Lin Hsiung's Control course.

2.4. Chile

TODO: to write

2.5. Duckietown High School

1) Introduction

DuckietownHS is inspired by the Duckietown project and targeted for high schools. The goal is to build and program duckiebots capable of moving autonomously on the streets of Duckietown. The technical objectives of DuckietownHS are simplified compared to those of the Duckietown project intended for universities so it is perfectly suited to the technical knowledge of the classes involved. The purpose is to create self-driving DuckiebotHS vehicles which can make choices and move autonomously on the streets of Duckietown, using sensors installed on the vehicles and special road signs positioned within Duckietown.

Once DuckiebotHS have been assembled and programmed to meet the specifications contained in this document and issued by the "customer" Perlatecnica, special missions and games will be offered for DuckiebotHS. The participants can also submit their own missions and games.

Just like the university project, DuckietownHS is an open source project, a role-playing game, a means to raise awareness on the subject and a learning experience for everyone involved. The project is promoted by the non-profit organization Perlatecnica based in Italy.

2) Purpose

The project has two main purposes:

- It is a course where students and teachers take part in a role play and they take the typical professional roles of an engineering company. They must design and implement a Duckietown responding to the specifications of the project, assemble DuckiebotHS (DBHS), and develop the software that will run on them. The deliverables of the project will be tutorials, how-to, source code, documentation, binaries and images and them will be designed and manufactured according to the procedures of the DTE.
- In respect of that mentioned above, special missions and games for DBHS will be introduced by the "customer" Perlatecnica.

3) Perlatecnica's role

Perlatecnica assumes the role of the customer and commissions the Duckietown Engineering company to design and construct the Duckietown and DuckiebotHS. It will provide all necessary product requirements and will assume the responsi-

bility to validate the compliancy of all deliverables to the required specifications.

4) The details of the project

The project consists in the design and realization of DuckiebotHS and DuckietownHS. They must have the same characteristics as the city of the University project as far as the size and color of the delimiting roadway bands is concerned but with a different type of management of the traffic lights system that regulates the passage of DuckiebotHS at intersections. The DuckietownHS (DTHS) and DuckiebotHS (DBHS) are defined in the documentation and there is little room for the DTE to make its own choices in terms of design. The reason for this is that the DBHS produced by the different DTE's need to be identical from a hardware point of view so that the software development makes the difference.

5) Where to start

The purchase of the necessary materials is the first step to take. For both DTHS and DBHS a list of these materials is provided with links to possible sellers. Even though Amazon is typically indicated as a seller this is nothing more than an indication to facilitate the purchase for those less experienced. It is left to the individual DTE to choose where to buy the required parts. It is allowed to buy and use parts that are not on the list but this is not recommended as they will make the Duckiebot unfit to enter in official competitions. When necessary an assembly tutorial will be provided together with the list of materials. Once the DTHS city and the DBHS robots have been assembled, the next step will be the development of the software for the running of both the city and the DuckiebotHS. The city and the Duckiebot run on a board based on a microcontroller STM32 from STMicroelectronics the Nucleo F401RE that will be programmed via the online development environment mbed. Perlatecnica will not release any of the official codes necessary for the navigation of the DuckiebotHS as these are owned by the DTE who developed them. The full standard document is available on the project official web site.

Each DTE may release the source code under a license Creative Commons CC BY-SA 4.0.

6) The first mission of the Duckiebot

Once you have completed the assembly of all the parts that make up the Duckietown and DuckiebotHS you should start programming the microcontroller so that the Duckiebot can move independently.

The basic mission of the DuckiebotHS is to move autonomously on the roads respecting the road signs and traffic lights, choosing a random journey and without crashing into other DuckiebotHS.

For the development of the code, there are no architectural constraints, but we recommend proceeding with order and to focus primarily on its major functions and not on a specific mission.

The main functions are those of perception and movement.

Moving around in DuckietownHS, the DuckiebotHS will have to drive on straight roads, make 90 degree curves while crossing an intersection but also make other unexpected curves. While doing all this the Duckiebot can be supported by a gyro-

scope that provides guidance to the orientation of the vehicle.

UNIT C-3

Duckietown classes [draft]

This section has been removed because it is in status 'draft'. If you are feeling adventurous, you can read it on master.

To disable this behavior, and completely hide the sections, set the parameter show_removed to false in fall2017.version.yaml.

UNIT C-4

First steps [draft]

This section has been removed because it is in status 'draft'. If you are feeling adventurous, you can read it on master.

To disable this behavior, and completely hide the sections, set the parameter show_removed to false in fall2017.version.yaml.

PART D

Duckumentation documentation

UNIT D-1

Contributing to the documentation

1.1. 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;
- a one-page version.

1.2. 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.

1.3. 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.

- Basic SSH config (master).
- Key pair creation (master).
- Adding public key on Github (master).

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 the `~/duckuments` directory:

```
$ git lfs clone --depth 100 git@github.com:duckietown/duckuments ~/duckuments
```

Here, note we are using `git lfs clone` – it's much faster, because it downloads the Git LFS files in parallel.

If it fails, it means that you do not have Git LFS installed. See Git LFS (master).

The command `--depth 100` tells it we don't care about the whole history.

3) Setup the virtual environment

Next, we will create a virtual environment using inside the `~/duckuments` directory. Make sure you are running Python 2.7.x. Python 3.x is not supported at the moment.

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 `vir-`

`tualenv`.

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
```

You also should run:

```
$ pip install -U SystemCmd==2.0.0
```

1.4. Compiling the documentation (updated Sep 12)

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
```

To compile the master versions of the docs, run:

```
$ make master-clean master
```

To see the result, open the file

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

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

```
$ make master
```

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

1) Compiling the Fall 2017 version only (introduced Sep 12)

To compile the Fall 2017 versions of the docs, run:

```
$ make fall2017-clean fall2017
```

To see the result, open the file

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

For incremental compilation, use:

```
$ make fall2017
```

2) Single-file compilation

There is also the option to compile one single file.

To do this, use:

```
$ ./compile-single path to .md file
```

This is the fastest way to see the results of the editing; however, there are limitations:

- no links to other sections will work.
- not all images might be found.

1.5. The workflow to edit documentation (updated Sep 12)

This is the basic workflow:

1. Create a branch called `yourname-branch` in the `duckuments` repository.
2. Edit the Markdown in the `yourname-branch` branch.
3. Run `make master` to make sure it compiles.
4. Commit the Markdown and push on the `yourname-branch` branch.
5. Create a pull request.
6. Tag the group `duckietown/gardeners`.

➡ Create a pull request from the command-line using hub (master).

1.6. Reporting problems

First, see the section Unit D-7 - Markduck troubleshooting for common problems and their resolution.

Please report problems with the duckuments using the `duckuments` issue tracker. If it is urgent, please tag people (Andrea); otherwise these are processed in batch mode every few days.

If you have a problem with a generated PDF, please attach the offending PDF.

If you say something like "This happens for Figure 3", then it is hard to know which figure you are referencing exactly, because numbering changes from commit to commit.

If you want to refer to specific parts of the text, please commit all your work on your branch, and obtain the name of the commit using the following commands:

```
$ git -C ~/duckuments rev-parse HEAD      # commit for duckuments
$ git -C ~/duckuments/mcdp rev-parse HEAD # commit for mcdp
```

UNIT D-2

Basic Markduck guide

The Duckiebook is written in Markduck, a Markdown dialect.

It supports many features that make it possible to create publication-worthy materials.

2.1. Markdown

The Duckiebook is written in a Markdown dialect.

➡ A tutorial on Markdown.

2.2. 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.

2.3. Character escapes

Use the string "`&#36;`" 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 2.1.

TABLE 2.1. SYMBOLS TO ESCAPE

use `&#36;`	instead of $
use ```	instead of `
use `&#lt;`	instead of <
use `&#gt;`	instead of >

2.4. 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>.
```

2.5. 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).
```

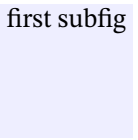
The code will put a reference to “Figure XX”.

2.6. Subfigures

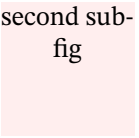
You can also create subfigures, using the following syntax.

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

    <div figure-id="subfig:first" figure-caption="Caption 1">
        <p style='width:5em;height:5em;background-color:#eef'>first subfig</p>
    </div>

    <div figure-id="subfig:second" figure-caption="Caption 2">
        <p style='width:5em;height:5em;background-color:#fee'>second subfig</p>
    </div>
</div>
```

This is the result:



(a) Caption 1

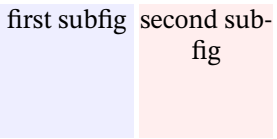


(b) Caption 2

Figure 2.1. Caption of big figure

By default, the subfigures are displayed one per line.

To make them flow horizontally, add `figure-class="flow-subfigures"` to the external figure `div`. Example:



(a) Caption 1 (b) Caption 2

Figure 2.2. Caption of big figure

2.7. 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 2.2. 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 2.3. USING CLASS="LABELS-COL1"

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

TABLE 2.4. USING CLASS=''LABELS-ROW1''

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

2.8. Linking to 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) How to name IDs - and why it's not automated

Some time ago, if there was a section called

```
## My section
```

then it would be assigned the ID "my-section".

This behavior has been removed, for several reasons.

One is that if you don't see the ID then you will be tempted to just change the name:

```
## My better section
```

and silently the ID will be changed to "my-better-section" and all the previous links will be invalidated.

The current behavior is to generate an ugly link like "autoid-209u31j".

This will make it clear that you cannot link using the PURL if you don't assign an ID.

Also, I would like to clarify that all IDs are *global* (so it's easy to link stuff, without thinking about namespaces, etc.).

Therefore, please choose descriptive IDs, with at least two IDs.

E.g. if you make a section called

```
## Localization  {#localization}
```

that's certainly a no-no, because "localization" is too generic.

✔

```
## Localization {#intro-localization}
```

Also note that you don't *need* to add IDs to everything, only the things that people could link to. (e.g. not subsubsections)

3) 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 2.5).

TABLE 2.5. SYNTAX FOR REFERRING TO SECTIONS.

```
See [](#establishing).
```

See Subsection 2.8.1 - Establishing names of headers

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

See Establishing names of headers.

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

See 2.8.1.

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

See Subsection 2.8.1 - Establishing names of headers.

4) 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 The “scuderia” (vehicle database) (master).

UNIT D-3

Special paragraphs and environments

3.1. Special paragraphs tags

The system supports parsing of some special paragraphs.

> **Note:** some of these might be redundant and will be eliminated. For now, I am documenting what is implemented.

1) Special paragraphs must be separated by a line

A special paragraph is marked by a special prefix. The list of special prefixes is given in the next section.

There must be an empty line before a special paragraph; this is because in Markdown a paragraph starts only after an empty line.

This is checked automatically, and the compilation will abort if the mistake is found.

For example, this is invalid:

```
See: this book
See: this other book
```

This is correct:

```
See: this book

See: this other book
```

Similarly, this is invalid:

```
Author: author
Maintainer: maintainer
```

and this is correct:

```
Author: author

Maintainer: maintainer
```

2) Todos, task markers

```
TODO: todo
```

TODO: todo

```
TOWRITE: towrite
```

To write: towrite

```
Task: task
```

Task: task

```
Assigned: assigned
```

> **Assigned to:** assigned

3) Notes and remarks

```
Remark: remark
```

> **Remark:** remark

```
Note: note
```

> **Note:** note

```
Warning: warning
```

Warning: warning

4) Troubleshooting

```
Symptom: symptom
```

> **Symptom:** symptom

```
Resolution: resolution
```

Resolution: resolution

5) Guidelines

```
Bad: bad
```

✖ bad

```
Better: better
```

✔ better

6) Questions and answers

```
Q: question
```

Q: question

```
A: answer
```

Answer: answer

7) Authors, maintainers, Point of Contact

```
Maintainer: maintainer
```

Maintainer: maintainer

```
Author: author
```

Author: author

```
Point of Contact: Point of Contact name
```

Point of contact: Point of Contact name

```
Slack channel: slack channel name
```

Slack channel: slack channel name

8) References

```
See: see
```

➡ see

```
Reference: reference
```

➡ reference

```
Requires: requires
```

Requires: requires

```
Results: results
```

Results: results

```
Next steps: next steps
```

Next: next steps

```
Recommended: recommended
```

Recommended: recommended

```
See also: see also
```

☀ see also

3.2. Other `div` environments

For these, note the rules:

- You must include `markdown="1"`.
- There must be an empty line after the first `div` and before the closing `/div`.

1) Example usage

```
<div class='example-usage' markdown="1">

This is how you can use `rosbag`:

    $ rosbag play log.bag

</div>
```

example

This is how you can use `rosbag`:

```
$ rosbag play log.bag
```

2) Check

```
<div class='check' markdown="1">

Check that you didn't forget anything.

</div>
```

Check before you continue
Check that you didn't forget anything.

3) Requirements

```
<div class='requirements' markdown="1">

List of requirements at the beginning of setup chapter.

</div>
```

KNOWLEDGE AND ACTIVITY GRAPH

List of requirements at the beginning of setup chapter.

3.3. Notes and questions

There are three environments: "comment", "question", "doubt", that result in boxes that can be expanded by the user.

These are the one-paragraph forms:

```
Comment: this is a comment on one paragraph.
```

+ comment
this is a comment on one paragraph.

```
Question: this is a question on one paragraph.
```

+ question
this is a question on one paragraph.

```
Doubt: I have my doubts on one paragraph.
```

+ doubt
I have my doubts on one paragraph.

These are the multiple-paragraphs forms:

```
<div class='comment' markdown='1'>
A comment...

A second paragraph...
</div>
```

+ comment
A comment...
A second paragraph...

```
<div class='question' markdown='1'>
A question...

A second paragraph...
</div>
```

+ question
A question...
A second paragraph...

```
<div class='doubt' markdown='1'>
A question...

Should it not be:

    $ alternative command

A second paragraph...
</div>
```

+ doubt
A question...
Should it not be:

```
$ alternative command
```

A second paragraph...

Using LaTeX constructs in documentation

KNOWLEDGE AND ACTIVITY GRAPH

> **Requires:** Working knowledge of LaTeX.

4.1. 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 example.

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

The above was written as in Listing 4.1.

```
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 4.1. 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{prop:name}` inside.

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

TABLE 4.1. 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.

example

```
\begin{definition}   \label{def:lorem}
Lorem
\end{definition}
```

Definition 1. Lorem

```
\begin{proposition}   \label{prop:lorem}
Lorem
\end{proposition}
```

Proposition 2. Lorem

```
\begin{remark}   \label{rem:lorem}
Lorem
\end{remark}
```

Remark 1. Lorem

```
\begin{problem}    \label{prob:lorem}
Lorem
\end{problem}
```

Problem 1. Lorem

```
\begin{example}   \label{exa:lorem}
Lorem
\end{example}
```

Example 1. Lorem

```
\begin{theorem}   \label{thm:lorem}
Lorem
\end{theorem}
```

Theorem 1. Lorem

```
\begin{lemma}   \label{lem:lorem}
Lorem
\end{lemma}
```

Lemma 1. Lorem

```
I can also refer to all of them:
[](#def:lorem),
[](#prop:lorem),
[](#rem:lorem),
[](#prob:lorem),
[](#exa:lorem),
[](#thm:lorem),
[](#lem:lorem).
```

I can also refer to all of them: Definition 1 - , Proposition 2 - , Remark 1 - , Problem 1 - , Example 1 - , Theorem 1 - , Lemma 1 - .

4.2. LaTeX equations

We can refer to equations, such as \eqref{eq:one}:
\begin{equation} 2a = a + a \label{eq:one}\tag{1} \end{equation}
This uses `align` and contains \eqref{eq:two} and \eqref{eq:three}.
\begin{align} a &= b \label{eq:two}\tag{2} \\ &= c \label{eq:three}\tag{3} \end{align}

```
We can refer to equations, such as \eqref{eq:one}:

\begin{equation}
    2a = a + a          \label{eq:one}
\end{equation}

This uses `align` and contains  \eqref{eq:two} and \eqref{eq:three}.

\begin{align}
    a &= b       \label{eq:two} \\
      &= c       \label{eq:three}
\end{align}
```

Note that referring to the equations is done using the syntax `\eqref{eq:name}`, rather than `[](#eq:name)`.

4.3. 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.

4.4. Bibliography support

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].

4.5. Embedding Latex in Figures through SVG

KNOWLEDGE AND ACTIVITY GRAPH

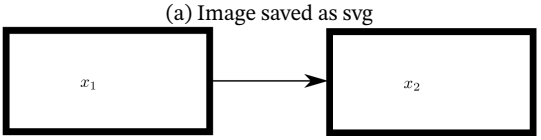
> **Requires:** In order to compile the figures into PDFs you need to have Inkscape installed. Instructions to download and install Inkscape are here.

To embed latex in your figures, you can add it directly to a file and save it as `filename.svg` file and save anywhere in the `/docs` directory.
You can run:

```
$ make process-svg-figs
```

And the SVG file will be compiled into a PDF figure with the LaTeX commands properly interpreted.
You can then include the PDF file in a normal way (Section 2.5 - Figures) using `filename.pdf` as the filename in the `<img>` tag.

(a) Image saved as svg



(b) Image as PDF after processing

Figure 4.1. Embedding LaTeX in images

It can take a bit of work to get the positioning of the code to appear properly on the figure.

UNIT D-5

Advanced Markduck guide

5.1. 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"/>
```

example

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 5.1. 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).

5.2. `move-here` tag

If a file contains the tag `move-here`, the fragment pointed by the `src` attribute is moved at the place of the tag.

This is used for autogenerated documentation.

Syntax:

```
# Node `node`

<move-here src='#package-node-autogenerated'/>
```

5.3. Comments

You can insert comments using the HTML syntax for comments: any text between "`<!--`" and "`-->`" is ignored.

```
# My section

<!-- this text is ignored -->

Let's start by...
```

5.4. Referring to Github files

You can refer to files in the repository by using:

```
See [this file](github:org=org,repo=repo,path=path,branch=branch).
```

The available keys are:

- `org` (required): organization name (e.g. `duckietown`);
- `repo` (required): repository name (e.g. `Software`);
- `path` (required): the filename. Can be just the file name or also include directories;
- `branch` (optional) the repository branch; defaults to `master`;

For example, you can refer to [the file `pkg_name/src/subscriber_node.py`](github:org=duckietown,repo=Software,path=pkg_name/src/subscriber_node.py) by using the following syntax:

```
See [this file](github:org=duckietown,repo=Software,path=pkg_name/src/subscriber_node.py)
```

You can also refer to a particular line:

This is done using the following parameters:

- `from_text` (optional): reference the first line containing the text;
- `from_line` (optional): reference the line by number;

For example, you can refer to [the line containing "Initialize"](github:org=duckietown,repo=Software,path=pkg_name/src/subscriber_node.py,from_text=Initialize) of `pkg_name/src/subscriber_node.py` by using the following syntax:

```
For example, you can refer to [the line containing "Initialize"][link2]
of `pkg_name/src/subscriber_node.py` by using the following syntax:

[link2]: github:org=duckietown,repo=Software,path=pkg_name/src/
subscriber_node.py,from_text=Initialize
```

You can also reference a range of lines, using the parameters:

- `to_text` (optional): references the final line, by text;
- `to_line` (optional): references the final line, by number.

You cannot give `from_text` and `from_line` at the same time. You cannot give a `to_...` without the `from_...`.

For example, [this link refers to a range of lines](github:org=duckietown,repo=Software,path=pkg_name/src/subscriber_node.py,from_text=Initialize,to_text=Finalize): click it to see how Github highlights the

lines from “Initialize” to “spin”.

This is the source of the previous paragraph:

```
For example, [this link refers to a range of lines][interval]: click it to see how Github
highlights the lines from ”Initialize” to ”spin”.

[interval]: github:org=duckietown,repo=Software,path=pkg_name/src/
subscriber_node.py,from_text=Initialize,to_text=spin
```

5.5. Putting code from the repository in line

In addition to referencing the files, you can also copy the contents of a file inside the documentation.

This is done by using the tag `display-file`.

For example, you can put a copy of `pkg_name/src/subscriber_node.py` using:

```
<display-file src=”
    github:org=duckietown,
    repo=Software,
    path=pkg_name/src/subscriber_node.py
”/>
```

and the result is the following automatically generated listing:

```
#!/usr/bin/env python
import rospy

# Imports message type
from std_msgs.msg import String

# Define callback function
def callback(msg):
    s = ”I heard: %s” % (msg.data)
    rospy.loginfo(s)

# Initialize the node with rospy
rospy.init_node('subscriber_node', anonymous=False)

# Create subscriber
subscriber = rospy.Subscriber(”topic”, String, callback)

# Runs continuously until interrupted
rospy.spin()
```

Listing 5.2. subscriber_node.py

If you use the `from_text` and `to_text` (or `from_line` and `to_line`), you can actually display part of a file. For example:

```
<display-file src="
    github:org=duckietown,
    repo=Software,
    path=pkg_name/src/subscriber_node.py,
    from_text=Initialize,
    to_text=spin
    "/>
```

creates the following automatically generated listing:

```
# Initialize the node with rospy
rospy.init_node('subscriber_node', anonymous=False)

# Create subscriber
subscriber = rospy.Subscriber("topic", String, callback)

# Runs continuously until interrupted
rospy.spin()
```

Listing 5.3. subscriber_node.py

UNIT D-6

*Compiling the PDF version

This part describes how to compile 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`.

6.1. 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
```

1) 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.

6.2. Installing Prince

Install PrinceXML from this page.

6.3. 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
```

6.4. Compiling the PDF

To compile the PDF, use:

```
$ make compile-pdf
```

This creates the file:

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

UNIT D-7

Markduck troubleshooting

7.1. Changes don't appear on the website

For these issues, see Unit D-8 - The Duckuments bot.

7.2. Troubleshooting errors in the compilation process

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.

7.3. Common mistakes with Markdown

Here are some common mistakes encountered.

1) Not properly starting a list

There must be an empty line before the list starts.

This is correct:

```
I want to learn:

- robotics
- computer vision
- underwater basket weaving
```

This is incorrect:

```
I want to learn:
- robotics
- computer vision
- underwater basket weaving
```

and it will be rendered as follows:

I want to learn: - robotics - computer vision - underwater basket weaving

UNIT D-8

The Duckuments bot

Note: This is an advanced section mainly for Liam.

8.1. Documentation deployment

The book is published to a different repository called `duckuments-dist` and from there published as `book.duckietown.org`.

8.2. Understand what's going on

There is a bot, called `frankfurt.co-design.science`, which is an AWS machine (somewhere in Frankfurt).

Every minute, it tries to do the following:

1. It checks out the last version of `mcdp`;
2. It checks out the last version of `duckuments`;
3. It compiles the various versions;
4. It uploads the results to a repository called `duckuments-dist`.

This process takes 4 minutes for an incremental change, and about 10-15 minutes for a big change, such as a change in headers IDs, which implies re-checking all cross-references.

8.3. Logging

There are logs you can access to see what's going on.

The high-level compilation log tells you in what phase of the cycle the bot is. Scroll to the bottom.

Ideally what you want to see is something like the following:

```
Starting
Mon Sep 11 10:49:04 CEST 2017
  succeded html
  succeded fall 2017
  succeded upload
  succeded split
  succeded html upload
  succeded PDF
  succeded PDF upload
Mon Sep 11 10:54:21 CEST 2017
Done.
```

This shows that the compilation took 5 minutes.

Every two hours you will see something like this:

```
automatic-compile-cleanup killing everything
```

and the next iteration will take longer because it starts from scratch.

The last log is a live version of the compilation log. This might not be tremendously informative because it is very verbose.

8.4. Debugging Github Pages problems

Sometimes, it's Github Pages that lags behind.

To check this, the bot also makes available the compilation output as a website called `book2.duckietown.org`. You can take any URL starting with `book.duckietown.org`, put `book2`, and you will see what is on the server.

This can identify if the problem is Github.

UNIT D-9

Documentation style guide

This chapter describes the conventions for writing the technical documentation.

9.1. 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.

9.2. Style guide for the Duckietown documentation

- The English version of the documentation is written in American English. Please note that your spell checker might be set to British English.

✖ behaviour

✔ behavior

✖ localisation

✔ localization

- 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.

9.3. 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
```

9.4. Frequently misspelled words

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

9.5. 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.)

9.6. 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.

UNIT D-10

Learning in Duckietown [draft]

This section has been removed because it is in status 'draft'. If you are feeling adventurous, you can read it on master.

To disable this behavior, and completely hide the sections, set the parameter show_removed to false in fall2017.version.yaml.

UNIT D-11

Knowledge graph [draft]

This section has been removed because it is in status 'draft'. If you are feeling adventurous, you can read it on master.

To disable this behavior, and completely hide the sections, set the parameter show_removed to false in fall2017.version.yaml.

UNIT D-12

Translations [draft]

This section has been removed because it is in status 'draft'. If you are feeling adventurous, you can read it on master.

To disable this behavior, and completely hide the sections, set the parameter show_removed to false in fall2017.version.yaml.

PART E

Exercises

UNIT E-1

Exercise: Basic image operations, adult version

Assigned to: Andrea Daniele

1.1. Skills learned

- Dealing with exceptions.
- Using exit conditions.
- Verification and unit tests.

1.2. Instructions

Implement the program `dt-image-flip` specified in the following section.

This time, we specify exactly what should happen for various anomalous conditions. This allows to do automated testing of the program.

1.3. `dt-image-flip` specification

The program `image-ops` expects exactly two arguments: a filename (a JPG file) and a directory name.

```
$ dt-image-flip file outdir
```

If the file does not exist, the script must exit with error code `2`.

If the file cannot be decoded, the script must exit with error code `3`.

If the file exists, then the script must create:

- `outdir/regular.jpg`: a copy of the initial file
- `outdir/flip.jpg`: the file, flipped vertically.
- `outdir/side-by-side.jpg`: the two files, side by side.

If any other error occurs, the script should exit with error code `99`.

(a) The original picture.

(b) The output `flip.jpg`

(c) The output `side-by-side.jpg`

Figure 1.1. Example input-output for the program `image-ops`.

1.4. Useful APIs

1) Images side-by-side

+ comment
Good explanation, but shouldn't it go in the previous exercise? -AC

An image loaded using the OpenCV function `imread` is stored in memory as a NumPy array. For example, the image shown above (Figure 1.1a - The original picture.) will be represented in memory as a NumPy array with shape `(96, 96, 3)`. The first dimension indicates the number of pixels along the `Y-axis`, the second indicates the number of pixels along the `X-axis` and the third is known as *number of channels* (e.g., **R**ed, **G**reen, and **B**lue).

+ comment
Are we sure it is RGB and not BGR? -AC

NumPy provides a utility function called `concatenate` that joins a sequence of arrays along a given axis.

1.5. Testing it works with `image-ops-tester`

We provide a script called `image-ops-tester` that can be used to make sure that you wrote a conforming `dt-image-flip`. The script `image-ops-tester` is in the directory `/exercises/dt-image-flip/image-ops-tester` in the `duckietown/duckuments` repository.

Use it as follows:

```
$ image-ops-tester candidate-program
```

If the script cannot be found, `image-ops-tester` will return 1.

`image-ops-tester` will return 0 if the program exists and conforms to the specification (Section 1.3 - `dt-image-flip` specification).

If it can establish that the program is not good, it will return 11.

Bottom line

Things that are not tested are broken.

UNIT E-2

Exercise: Bag in, bag out

Assigned to: Andrea Daniele

2.1. Skills learned

- Processing the contents of a bag to produce another bag.

2.2. Instructions

Implement the program `dt-bag-decimate` as specified below.

2.3. Specification of `dt-bag-decimate`

The program `dt-bag-decimate` takes as argument a bag filename, an integer value greater than zero, and an output bag file:

```
$ dt-bag-decimate "input bag" n "output bag"
```

The output bag contains the same topics as the input bag, however, only 1 in `n` messages are written. (If `n` is 1, the output is the same as the input.)

2.4. Useful new APIs

1) Create a new Bag

In ROS, a new bag can be created by specifying the mode `w` (i.e., write) while instantiating the class `rosbag.Bag`.

For example:

```
from rosbag import Bag
new_bag = Bag('./output_bag.bag', mode='w')
```

Visit the documentation page for the class `rosbag.Bag` for further information.

2) Write message to a Bag

A ROS bag instantiated in *write* mode accepts messages through the function `write()`.

2.5. Check that it works

To check that the program works, you can compute the statistics of the data using the program `dt-bag-analyze` that you have created in Exercise: Simple data analysis from a bag (master).

You should see that the statistics have changed.

UNIT E-3

Exercise: Instagram filters

Assigned to: Andrea Daniele

3.1. Skills learned

- Image pixel representation;

- Image manipulation;
- The idea that we can manipulate operations as objects, and refer to them (higher-order computation);
- The idea that we can compose operations, and sometimes the operations do commute, while sometimes they do not.

3.2. Instructions

Create `dt-instagram` as specified below.

3.3. Specification for `dt-instagram`

Write a program `dt-instagram` that applies a list of filters to an image.

The syntax to invoke the program is:

```
$ dt-instagram image in filters image out
```

where:

- `image in` is the input image;
- `filters` is a string, which is a colon-separated list of filters;
- `image out` is the output image.

The list of filters is given in Subsection 3.3.1 - List of filters.

For example, the result of the command:

```
$ dt-instagram image.jpg flip-horizontal:grayscale out.jpg
```

is that `out.jpg` contains the input image, flipped and than converted to grayscale.

Because these two commute, this command gives the same output:

```
$ dt-instagram image.jpg grayscale:flip-horizontal out.jpg
```

1) List of filters

Here is the list of possible values for the filters, and their effect:

- `flip-vertical`: flips the image vertically
- `flip-horizontal`: flips the image horizontally
- `grayscale`: Makes the image grayscale
- `sepia`: make the image sepia

3.4. Useful new APIs

1) User defined filters

In OpenCV it is possible to define custom filters and apply them to an image. A linear filter (e.g., sepia) is defined by a linear 9-dimensional kernel. The `sepia` filter is defined as:

$$ K_{sepia} = \begin{bmatrix} 0.272 & 0.534 & 0.131 \newline 0.349 & 0.686 & 0.168 \newline 0.393 & 0.769 & 0.189 \end{bmatrix} $$

A linear kernel describing a color filter defines a linear transformation in the color space. A transformation can be applied to an image in OpenCV by using the function `transform()`.

UNIT E-4

Exercise: Live Instagram

Assigned to: Andrea Daniele

4.1. Skills learned

- Live image processing

4.2. Instructions

You may find useful: Minimal ROS node - `pkg_name` (master). That tutorial is about listening to text messages and writing back text messages. Here, we apply the same principle, but to images.

Create a ROS node that takes camera images and applies a given operation, as specified in the next section, and then publishes it.

4.3. Specification for the node `dt-live-instagram-ROBOT_NAME_node`

Create a ROS node `dt-live-instagram-ROBOT_NAME_node` that takes reads a parameter called `filter`, where the filter is something from the list Subsection 3.3.1 - List of filters.

You should launch your camera and joystick with

```
$ make demo-joystick-camera
```

Then launch your node with

```
$ roslaunch dt-live-instagram_ROBOT_NAME dt_live-instagram_ROBOT_NAME_node
filter:=filter
```

This program should do the following:

- Subscribe to the camera images, by finding a topic that is called `.../compressed`. Call the name of the topic `topic`.
- Publish to the topic `topic/filters/compressed` a stream of images where the filter are applied to the image.

4.4. Check that it works

```
$ rqt_image_view
```

and look at `topic/filters/compressed`

PART F
Theory Background

UNIT F-1
Probability basics

In this chapter we give a brief review of some basic probabilistic concepts. For a more in-depth treatment of the subject we refer the interested reader to a textbook such as [19].

1.1. Random Variables

The key underlying concept in probabilistic theory is that of an *event*, which is the output of a random trial. Examples of an event include the result of a coin flip turning up HEADS or the result of rolling a die turning up the number "4".

Definition 2. (Random Variable) A (either discrete or continuous) variable that can take on any value that corresponds to the feasible output of a random trial.

For example, we could model the event of flipping a fair coin with the random variable X. We write the probability that X takes HEADS as $p(X=\text{HEADS})$. The set of all possible values for the variable X is its *domain*, $\aset{X}$. In this case, $$ \aset{X}=\{\text{HEADS},\text{TAILS}\}. $$ Since X can only take one of two values, it is a *binary* random variable. In the case of a die roll, $$ \aset{X}=\{1,2,3,4,5,6\}, $$ and we refer to this as a *discrete* random variable. If the output is real value or a subset of the real numbers, e.g., $\aset{X} = \reals$, then we refer to X as a *continuous* random variable.

Consider once again the coin tossing event. If the coin is fair, the have $p(X=HEADS)=p(X=TAILS)=0.5$. Here, the function $p(x)$ is called the *probability mass function* or pmf. The pmf is shown in Figure 1.1.

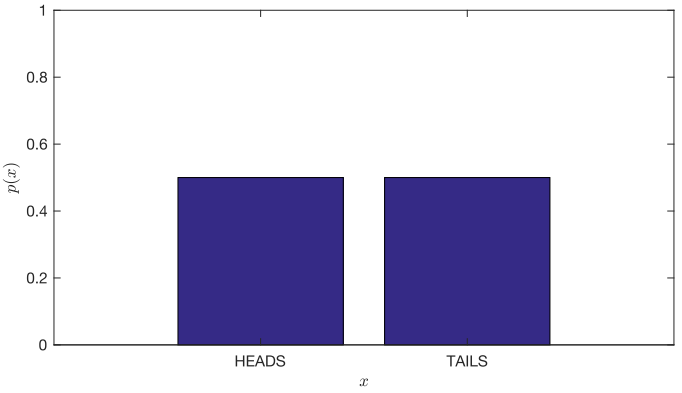


Figure 1.1. The pmf for a fair coin toss

Here are some very important properties of $p(x)$: - $0\leq p(x) \leq (1)$ - $\sum_{x\in\aset{X}}=1$

In the case of a continuous random variable, we will call this function $f(x)$ and call it a *probability density function*, or pdf.

In the case of continuous RVs, technically the $p(X=x)$ for any value x is zero since $\aset{X}$ is infinite. To deal with this, we also define another important function, the *cumulative density function*, which is given by $F(x) \triangleq p(X\leq x)$, and now we can define $f(x) \triangleq \frac{d}{dx}F(x)$. A pdf and corresponding cdf are shown in Figure 1.2 (This happens to be a Gaussian distribution, defined more precisely in Subsection 1.1.8 - The Gaussian Distribution.).

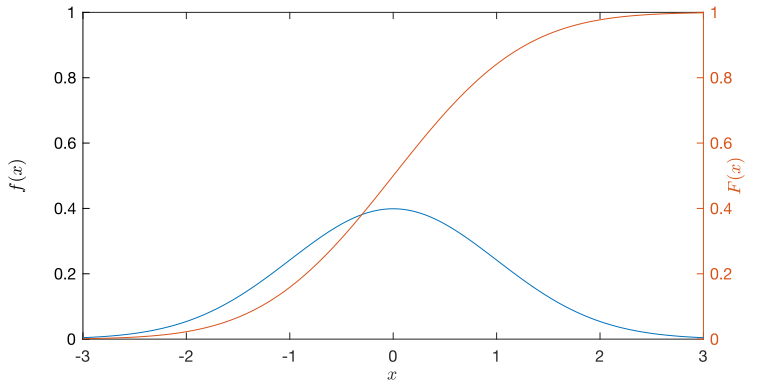


Figure 1.2. The continuous pdf and cdf

1) Joint Probabilities

If we have two different RVs representing two different events X and Y, then we represent the probability of two distinct events $x \in \aset{X}$ and $y \in \mathcal{Y}$ both happening, which we will denote as following: $p(X=x \; \text{AND} \; Y=y) = p(x,y)$. The function $p(x,y)$ is called *joint distribution*.

2) Conditional Probabilities

Again, considering that we have to RVs, X and Y, imagine these two events are

linked in some way. For example, X is the numerical output of a die roll and Y is the binary even-odd output of the same die roll. Clearly these two events are linked since they are both uniquely determined by the same underlying event (the rolling of the die). In this case, we say that the RVs are *dependent* on one another. In the event that we know one of events, this gives us some information about the other. We denote this using the following *conditional distribution* $p(X=x \; \text{GIVEN} \; Y=y) \triangleq p(x|y)$.

> **Check before you continue**
>
> Write down the conditional pmf for the scenario just described assuming an oracle tells you that the die roll is even. In other words, what is p(x|\text{EVEN})?
>
> (Warning: if you think this is very easy that's good, but don't get over-confident.)

The joint and conditional distributions are related by the following (which could be considered a definition of the joint distribution):

\begin{equation} p(x,y) = p(x|y)p(y) \label{eq:joint}\tag{1} \end{equation}

and similarly, the following could be considered a definition of the conditional distribution:

\begin{equation} p(x|y) = \frac{p(x,y)}{p(y)} \; \text{if} \; p(y) \gt{} 0 \label{eq:condition}\tag{2} \end{equation}

In other words, the conditional and joint distributions are inextricably linked (you can't really talk about one without the other).

If two variables are *independent*, then the following relation holds: $p(x,y)=p(x)p(y)$.

3) Bayes' Rule

Upon closer inspection of \eqref{eq:joint}, we can see that the choice of which variable to condition upon is completely arbitrary. We can write:

$$ p(y|x)p(x) = p(x,y) = p(x|y)p(y) $$

and then after rearranging things we arrive at one of the most important formulas for mobile robotics, Bayes' rule:

\begin{equation} p(x|y) = \frac{p(y|x)p(x)}{p(y)} \label{eq:bayes}\tag{3} \end{equation}

Exactly why this formula is so important will be covered in more detail in later sections (TODO), but we will give an initial intuition here.TODO

Consider that the variable X represents something that we are trying to estimate but cannot observe directly, and that the variable Y represents a physical measurement that relates to X. We want to estimate the distribution over X given the measurement Y, $p(x|y)$, which is called the *posterior* distribution. Bayes' rule lets us to do this. For every possible state, you take the probability that this measurement could have been generated, $p(y|x)$, which is called the *measurement likelihood*, you multiply it by the probability of that state being the true state, $p(x)$, which is called the *prior*, and you normalize over the probability of obtaining that

measurement from any state, $p(y)$, which is called the *evidence*.

> **Check before you continue**
>
> From Wikipedia: Suppose a drug test has a 99% true positive rate and a 99% true negative rate, and that we know that exactly 0.5% of people are using the drug. Given that a person's test gives a positive result, what is the probability that this person is actually a user of the drug.
>
> Answer: $\approx$ 33.2%. This answer should surprise you. It highlights the power of the *prior*.

4) Marginal Distribution

If we already have a joint distribution $p(x,y)$ and we wish to recover the single variable distribution $p(x)$, we must *marginalize* over the variable Y. The involves summing (for discrete RVs) or integrating (for continuous RVs) over all values of the variable we wish to marginalize:

\begin{align} p(x) &= \sum_{\mathcal{Y}} p(x,y) f(x) &= \int p(x,y) dy \end{align}

This can be thought of as projecting a higher dimensional distribution onto a lower dimensional subspace. For example, consider Figure 1.3, which shows some data plotted on a 2D scatter plot, and then the marginal histogram plots along each dimension of the data.

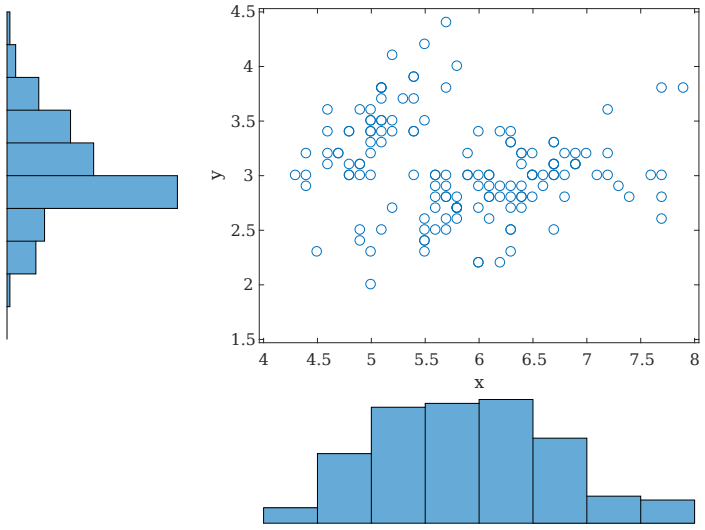


Figure 1.3. A 2D joint data and 2 marginal 1D histogram plots

Marginalization is an important operation since it allows us to reduce the size of our state space in a principled way.

5) Conditional Independence

Two RVs, X and Y may be correlated, we may be able to encapsulate the depen-

dence through a third random variable Z. Therefore, if we know Z

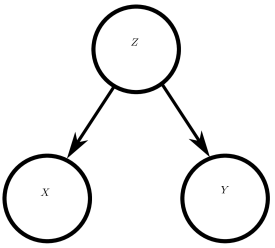


Figure 1.4. A graphical representation of the conditional independence of X and Y given Z

+ comment

Is there a discussion of graphical models anywhere? Doing a good job of sufficiently describing graphical models and the dependency relations that they express requires careful thought. Without it, we should refer readers to a graphical models text (e.g., Koller and Friedman, even if it is dense)

6) Moments

The nth moment of an RV, X, is given by $E[X^n]$ where $E[]$ is the expection operator with:

$$ E[f(X)] = \sum_{\aset{X}} x \, f(x) $$ in the discrete case and $$ E[f(X)] = \int x\, f(x) dx $$ in the continuous case.

The 1st moment is the *mean*, $\mu_X=E[X]$.

The nth central moment of an RV, X is given by $E[(X-\mu_X)^n]$. The second central moment is called the *covariance*, $\sigma^2_X=E[(X-\mu_X)^2]$.

7) Entropy

Definition 3. The *entropy* of an RV is a scalar measure of the uncertainty about the value the RV.

A common measure of entropy is the *Shannon entropy*, whose value is given by

\begin{equation} H(X)=-E[\log_2 p(x)] \label{eq:shannon}\tag{4} \end{equation}

This measure originates from communication theory and literally represents how many bits are required to transmit a distribution through a communication channel. For many more details related to information theory we recommend [20].

As an example, we can easily write out the Shannon entropy associated with a binary RV (e.g. flipping a coin) as a function of the probability that the coin turns up heads (call this p):

\begin{equation} H(X) = -p\log_2 p - (1-p)\log_2 (1-p) \label{eq:binary_entropy}\tag{5} \end{equation}

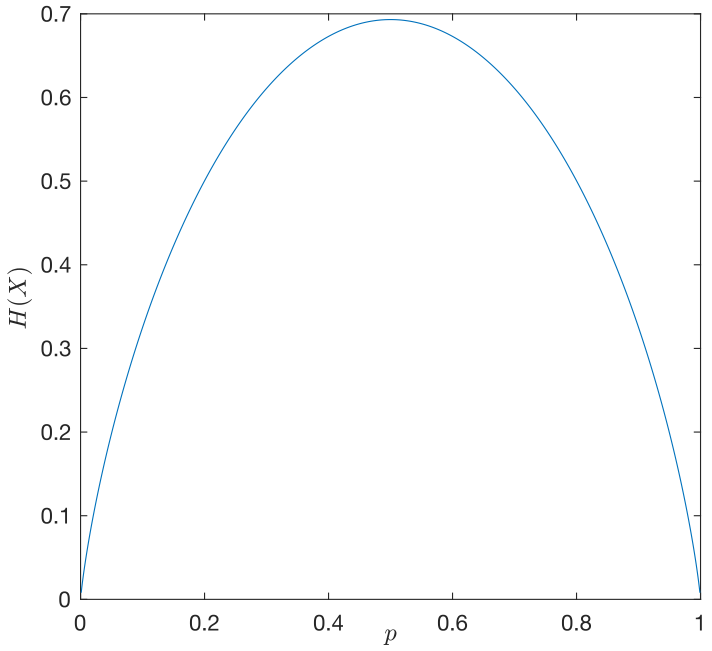


Figure 1.5. The Shannon entropy of a binary RV X

Notice that our highest entropy (uncertainty) about the outcome of the coin flip is when it is a fair coin (equal probability of heads and tails). The entropy decays to 0 as we approach $p=0$ and $p=1$ since in these two cases we have no uncertainty about the outcome of the flip. It should also be clear why the function is symmetrical around the $p=0.5$ value.

8) The Gaussian Distribution

In mobile robotics we use the Gaussian, or normal, distribution a lot.

+ comment
The banana distribution is the official distribution in robotics! - AC

+ comment
The banana distribution is Gaussian! http://www.roboticsproceedings.org/rss08/p34.pdf - LP

The 1-D Gaussian distribution pdf is given by:

\begin{equation} \mathcal{N}(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi \sigma^2}}e^{-\frac{1}{2\sigma^2}(x-\mu)^2} \label{eq:gaussian1D}\tag{6} \end{equation}

where μ is called the *mean* of the distribution, and σ is called the *standard deviation*. A plot of the 1D Gaussian was previously shown in Figure 1.2.

We will rarely deal with the univariate case and much more often deal with the multi-variate Gaussian:

\begin{equation} \mathcal{N}(\state|\bmu,\bSigma) = \frac{1}{(2*\pi)^{D/2}|\bSigma|^{1/2}}\exp[-\frac{1}{2}(\state-\bmu)^T\bSigma^{-1}(\state - \bmu)] \la-

bel{eq:gaussianND}\tag{7} \end{equation}

The value from the exponent: $(\state-\bmu)^T\bSigma^{-1}(\state - \bmu)$ is sometimes written $||\state - \bmu||_\bSigma$ and is referred to as the *Mahalanobis distance* or *energy norm*.

Mathematically, the Gaussian distribution has some nice properties as we will see. But is this the only reason to use this as a distribution. In other words, is the assumption of Gaussianicity a good one?

There are two very good reasons to think that the Gaussian distribution is the "right" one to use in a given situation.

1. The *central limit theorem* says that, in the limit, if we sum an increasing number of independent random variables, the distribution approaches Gaussian
2. It can be proven (TODO:ref) that the Gaussian distribution has the maximum entropy subject to a given value for the first and second moments. In other words, for a given mean and variance, it makes the *least* assumptions about the other moments.

Exercise: derive the formula for Gaussian entropy

UNIT F-2

Linearity and Vectors [draft]

This section has been removed because it is in status 'draft'. If you are feeling adventurous, you can read it on master.

To disable this behavior, and completely hide the sections, set the parameter show_removed to false in fall2017.version.yaml.

PART G

Introduction to autonomy

UNIT G-1

Autonomous Vehicles [draft]

This section has been removed because it is in status 'draft'. If you are feeling adventurous, you can read it on master.

To disable this behavior, and completely hide the sections, set the parameter show_removed to false in fall2017.version.yaml.

UNIT G-2

Autonomy overview

Assigned to: Liam

This unit introduces some basic concepts ubiquitous in autonomous vehicle naviga-

tion.

2.1. Basic Building Blocks of Autonomy

The minimal basic backbone processing pipeline for autonomous vehicle navigation is shown in Figure 2.1.

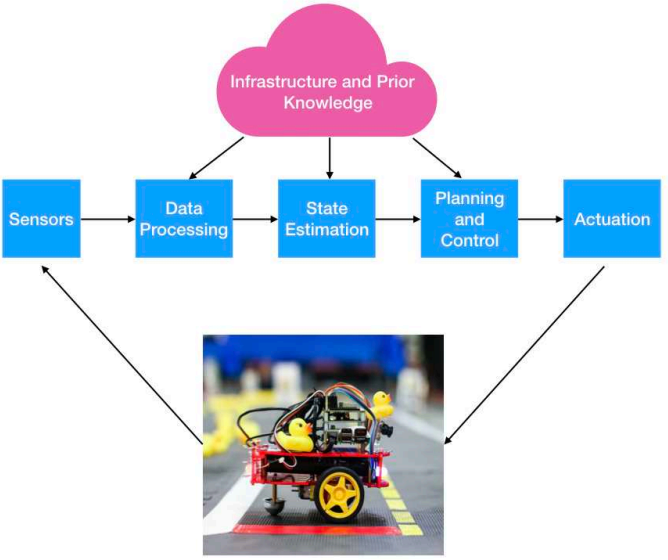


Figure 2.1. The basic building blocks of any autonomous vehicle

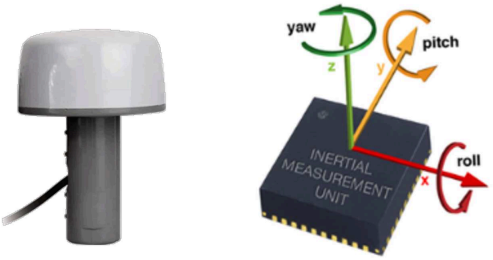
For an autonomous vehicle to function, it must achieve some level of performance for all of these components. The level of performance required depends on the *task* and the *required performance*. In the remainder of this section, we will discuss some of the most basic options. In the next section we will briefly introduce some of the more advanced options that are used in state-of-the-art autonomous vehicles.

1) Sensors

(a) Velodyne 3D Laser Scanner

(b) Camera

(c) Automotive Radar



(d) GPS Receiver (e) Inertial Measurement Unit

Figure 2.2. Some common sensors used for autonomous navigation

Definition 4. (Sensor) A *sensor* is a device that or mechanism that is capable of generating a measurement of some external phyiscal quantity

In general, sensors have two major types. *Passive* sensors generate measurements without affecting the environment that they are measuring. Examples include inertial sensors, odometers, GPS receivers, and cameras. *Active* sensors emit some form of energy into the environment in order to make a measurement. Examples of this type of sensor include Light Detection And Ranging (LiDAR), Radio Detection And Ranging (RaDAR), and Sound Navigation and Ranging (SoNAR). All of these sensors emit energy (from different spectra) into the environment and then detect some property of the energy that is reflected from the environment (e.g., the time of flight or the phase shift of the signal)

2) Raw Data Processing

The raw data that is input from a sensor needs to be processed in order to become useful and even understandandable to a human.

First, **calibration** is usually required to convert units, for example from a voltage to a physical quantity. As a simple example consider a thermometer, which measures temperature via an expanding liquid (usually mercury). The calibration is the known mapping from amount of expansion of liquid to temperature. In this case it is a linear mapping and is used to put the markings on the thermometer that make it useful as a sensor.

We will distiguish between two fundamentally types of calibrations.

Definition 5. (Intrinsic Calibration) An *intrinsic calibration* is required to determine sensor-specific paramaters that are internal to a specific sensor.

Definition 6. (Extrinsic Calibration) An *extrinsic calibration* is required to determine the external configuration of the sensor with respect to some reference frame.

> **Check before you continue**
> For more information about reference frames check out Reference frames (master)

Calibration is very important consideration in robotics. In the field, the most advanced algorithms will fail if sensors are not properly calibrated.

Once we have properly calibrated data in some meaningful units, we often do some preprocessing to reduce the overall size of the data. This is true particularly for sensors

that generate a lot of data, like cameras. Rather than deal with every pixel value generated by the camera, we will process an image to generate feature-points of interest. In "classical" computer vision many different feature descriptors have been proposed (Harris, BRIEF, BRISK, SURF, SIFT, etc), and more recently Convolutional Neural Networks (CNNs) are being used to learn these features.

The important property of these features is that they should be as easily to associate as possible across frames. In order to achieve this, the feature descriptors should be invariant to nuissance parameters.

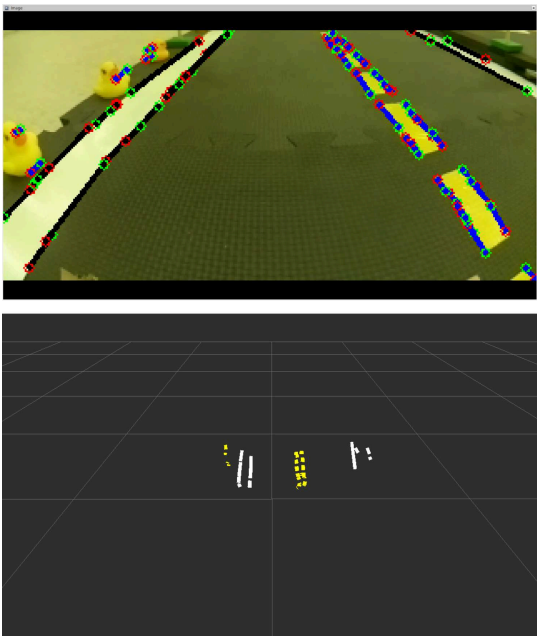


Figure 2.3. Top: A raw image with feature points indicated. Bottom: Lines projected onto ground plane using extrinsic calibration and ground projections

3) State Estimation

Now that we have used our sensors to generate a set of meaningful measurements, we need to combine these measurements together to produce an estimate of the underlying hidden *state* of the robot and possibly to environment.

Definition 7. (State) The state $\state_t \in \statesp$ is a *sufficient statistic* of the environment, i.e. it contains all sufficient information required for the robot to carry out its task in that environment. This can (and usually does) include the *configuration* of the robot itself.

What variables are maintained in the statespace $\statesp$ depends on the problem at hand. For example we may just be interested in a single robot's configuration in the plane, in which case $\state_t \equiv \pose_t$. However, in other cases, such as simultaneous localization and mapping, me may also be tracking the map in the state space.

According to Bayesian principles, any system parameters that are not fully known and deterministic should be maintained in the state space.

In general, we do not have direct access to values in $\state$, instead we rely on our (noisy) sensor measurements to tell us something about them, and then we *infer* the

values.

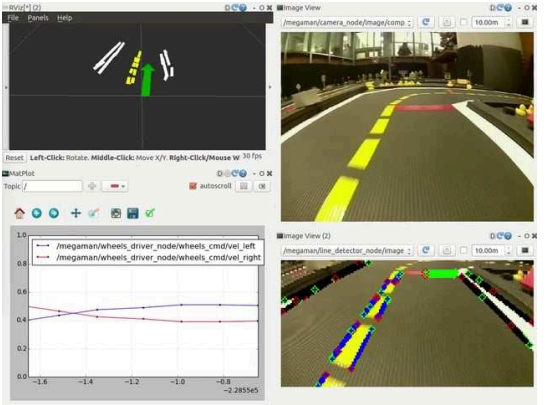


Figure 2.4. Lane Following in Duckietown. *Top Right*: Raw image; *Bottom Right*: Line detections; *Top Left*: Line projections and estimate of robot position within the lane (green arrow); *Bottom Left*: Control signals sent to wheels.

The animation in Figure 2.4 shows the lane following procedure. The output of the state estimator produces the **green arrow** in the top left pane.

4) Navigation and Planning

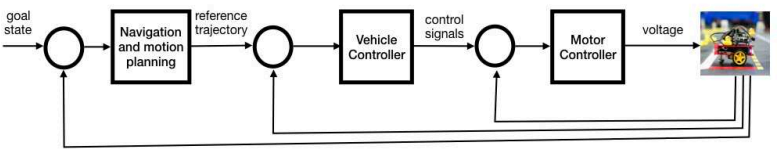


Figure 2.5. An example of nested control loops

In general we decompose the task of controlling an autonomous vehicle into a series of **nested control loops**.

The loops are called nested since the output of the outer loop is used as the reference input to the inner loop. An example is shown in Figure 2.5.

> **Recommended:** If Figure 2.5 is **VERY** mysterious to you, then you may want to have a quick look in a basic feedback control textbook. For example [21] or [22].

In this case we show three loops. At the outer loop, some goal state is provided. The actual state of the robot is used as the feedback. The controller is the block labeled `Navigation and Motion Planning`. The job of this block is generate a **feasible path** from the current state to the goal state. This is executed in **configuration space** rather than the state space (although these two spaces may happen to be the same they are fundamentally conceptually different.

Figure 2.6. Navigation in Duckietown

5) Control

The next inner loop of the nested controller in Figure 2.5 is the `Vehicle Controller`, which takes as input the reference trajectory generated by the `Navigation and Motion Planning` block and the current configuration of the robot, and uses the error between the two to generate a control signal.

The most basic feedback control law (See Feedback control (master) is called PID (for proportional, integral, derivative) which will be discussed in PID Control (master). For an excellent introduction to this control policy see Figure 2.7.

Figure 2.7. Controlling Self Driving Cars

We will also investigate some more advanced non-linear control policies such as Model Predictive Control (master), which is an optimization based technique.

6) Actuation

The very innermost control loop deals with actually tracking the correct voltage to be sent to the motors. This is generally executed as close to the hardware level as possible. For example we have a `Stepper Motor HAT` See the parts list.

7) Infrastructure and Prior Information

In general, we can make the autonomous navigation a simpler one by exploiting existing structure, infastructure, and contextual prior knowledge.

Infrastructure example: Maps or GPS satellites

Structure example: Known color and location of lane markings

Contextual prior knowledge example: Cars tend to follow the *Rules of the Road*

2.2. Advanced Building Blocks of Autonomy

The basic building blocks enable static navigation in Duckietown. However, many oth-

er components are necessary for more realistic scenarios.

1) Object Detection

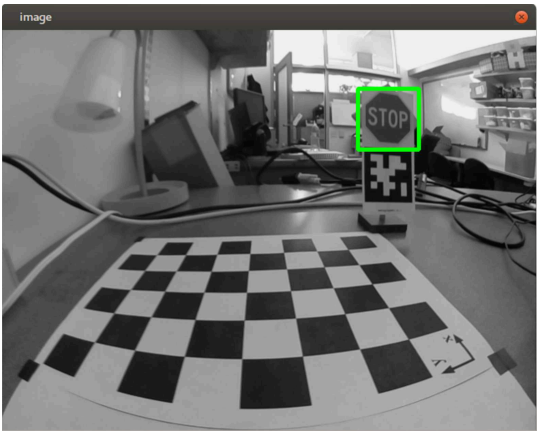


Figure 2.8. Advanced Autonomy: Object Detection

One key requirement is the ability to detect objects in the world such as but not limited to: signs, other robots, people, etc.

2) SLAM

The simultaneous localization and mapping (SLAM) problem involves simultaneously estimating not only the robot state but also the **map** at the same time, and is a fundamental capability for mobile robotics. In autonomous driving, generally the most common application for SLAM is actual in the map-building task. Once a map is built then it can be pre-loaded and then used for pure localization. A demonstration of this in Duckietown is shown in Figure 2.9.

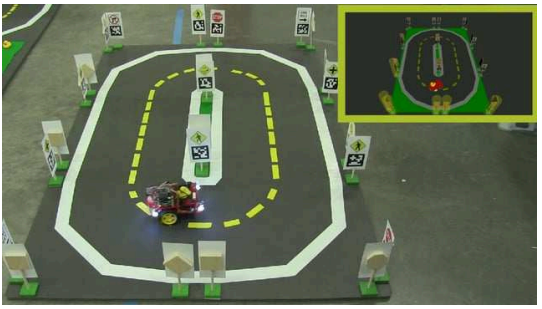


Figure 2.9. Localization in Duckietown

3) Other Advanced Topics

Other topics that will be covered include:

- Visual-inertial navigation (VINS)
- Fleet management and coordination
- Scene segmentation
- Deep perception
- Text recognition

UNIT G-3

Modern Robotic Systems [draft]

This section has been removed because it is in status 'draft'. If you are feeling adventurous, you can read it on master.

To disable this behavior, and completely hide the sections, set the parameter show_removed to false in fall2017.version.yaml.

UNIT G-4

Autonomy architectures [draft]

This section has been removed because it is in status 'draft'. If you are feeling adventurous, you can read it on master.

To disable this behavior, and completely hide the sections, set the parameter show_removed to false in fall2017.version.yaml.

UNIT G-5

Representations [draft]

This section has been removed because it is in status 'draft'. If you are feeling adventurous, you can read it on master.

To disable this behavior, and completely hide the sections, set the parameter show_removed to false in fall2017.version.yaml.

UNIT G-6

Duckiebot modeling

Assigned to: Jacopo

TODO: put prettier figures

Obtaining a mathematical model of the Duckiebot is important in order to (a) understand its behavior and (b) design a controller to obtain desired behaviors and performances, robustly.

The Duckiebot is a differential drive mobile robot, where the actuators are DC motors. By applying different torques to the wheels a Duckiebot can turn, go straight (same torque to both wheels) or stay still (no torque to both wheels). This driving configuration is referred to as *differential drive*.

In this section we will first derive the kinematic and dynamic models of a Duckiebot, comprehensive of the model of its actuators (DC motors).

Different methods can be followed to obtain the Duckiebot model, namely the Lagrangian or Newton-Euler, we choose to describe the latter as it arguably provides a

clearer physical insight. Showing the equivalence of these formulations is an interesting exercise, and the reader may refer to [27] for more insight. Another useful resource for modeling of a Duckiebot may be found here [28].

KNOWLEDGE AND ACTIVITY GRAPH

Requires:k:reference_frames (master) (inertial, body), k:intro-transformations (master) (Cartesian, polar)

Requires: k:intro-kinematics (master)

Requires: k:intro-dynamics (master)

Suggested: k:intro-ODEs-to-LTIsys (master)

Results: k:diff-drive-robot-model

6.1. Preliminaries

We first briefly recapitulate on the (reference frames)[#reference-frames] that we will use to model the Duckiebot, with the intent of introducing the notation used throughout this chapter.

To describe the behavior of a Duckiebot two reference frames will be used:

- An *inertial* frame: a fixed two dimensional "global" reference system that spans the plane on which the Duckiebot moves. We will denote its axis as $\{X_I, Y_I\}$.
- A *body* (or "robot") frame: a local reference frame fixed with respect to the robot, centered in the midpoint (A) of the axis between the wheels. The x axis points in the direction of the front of the robot, and the y axis lies along the axis between the wheels, so to form a right handed reference system. We denote the robot body frame with $\{x_r, y_r\}$.

Note: The robot is assumed to be a rigid body, symmetric, and x_r coincides with axis of symmetry.

Note: Quantities described with respect to the inertial or robot frames are denoted as $((\cdot)^I)$ and $((\cdot)^r)$ respectively.

- The center of mass $C^I = (x_c, y_c)$ of the robot is on the x_r axis, at a distance c from A, i.e., ($C^r = (c, 0)$);
- x^r forms an *orientation angle* θ with the local horizontal;
- the wheels are assumed to be identical, with diameter equal to $2R$;
- the distance between the wheels is denoted as $2L$.

The position of the robot with respect to the inertial frame is completely characterized by:

$$\avec{q^I} = \left(\begin{array}{c} x_A \\ y_A \\ \theta \end{array} \right),$$

and it is always possible to switch representation of a vector X from the robot to inertial frames through:

$$\label{eq:mod-r2i}\tag{5} \avec{X^I} = \amat{R}(\theta)\avec{X^r}.$$

$\amat{R}(\theta)$ is an orthogonal rotation matrix defined by:

$$\label{eq:mod-rot-mat}\tag{6} \amat{R}(\theta) = \left[\begin{array}{ccc} \cos\theta & -\sin \theta & 0 \\ \sin\theta & cos\theta & 0 \\ 0 & 0 & 1

\end{array} \right]. \end{align}

+ comment

LP. This above is incorrect. Should be \begin{align} \label{eq:mod-rot-mat}\tag{7} \amat{R}(\theta) = \left[\begin{array}{ccc} \cos\theta & -\sin \theta & 0 \\ \sin\theta & cos\theta & 0 \\ x_A & y_y & 1 \end{array} \right]. \end{align}

Note: Remember that the orthogonality condition implies that $\amat{R}^T(\theta)\amat{R}(\theta) = \amat{R}(\theta)\amat{R}^T(\theta) = \amat{I}$, hence: $$ \label{eq:mod-orthogonality-cond}\tag{1} \amat{R}^T(\theta) = \amat{R}^{-1}(\theta) $$

Note: $\amat{R}(\theta)$ is not a function of time, but only of the orientation. Hence, by denoting time derivatives as $\dot{(\cdot)}$ we can obtain the relation between velocities in the two reference systems: \begin{align} \label{eq:mod-r2i-dot}\tag{8} \avec{\dot X^I} = \amat{R}(\theta)\avec{\dot X^r}. \end{align}

Figure 6.1 summarizes the notations introduced.

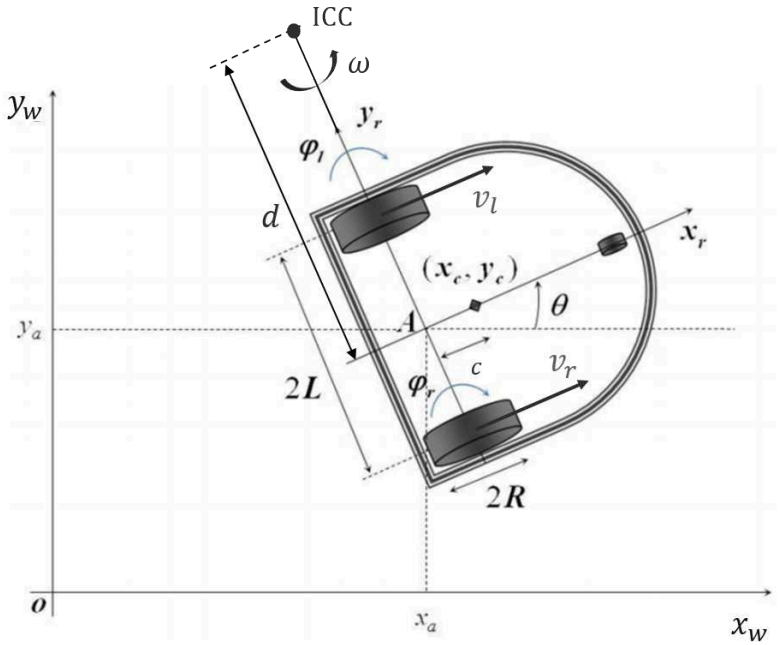


Figure 6.1. Relevant notations for modeling a differential drive robot

6.2. Kinematics

In this section we derive the kinematic model of a differential drive mobile platform.

1) Differential drive robot kinematic constraints

The kinematic constraints are derived from two assumptions:

- *No lateral slipping motion*: the robot cannot move sideways, but only in the direction of motion, i.e., its lateral velocity in the robot frame is zero, i.e.: $$ \label{eq:mod-no-lat-slip-constraint-r}\tag{2} \dot y_A^r = 0. $$

By inverting \eqref{eq:mod-r2i}, this constraint can be expressed through the inertial frame variables, yielding:

$$ \label{eq:mod-no-lat-slip-constraint-i}\tag{3} \dot y_A \cos \theta -\dot x_A \sin \theta = 0. $$

- *Pure rolling*: the wheels never slips or skids (Figure 6.2). Hence, letting $\dot \varphi_{l}, \dot \varphi_{r}$ be the angular velocities of the left and right wheels respectively, the velocity of the ground contact point P is given by:

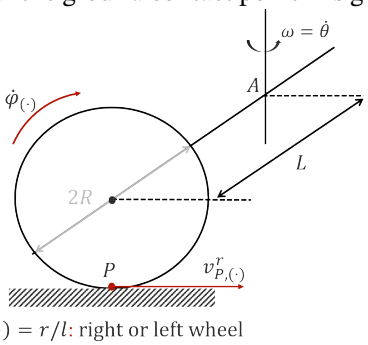


Figure 6.2. Pure rolling (no slipping) kinematic constraint

\begin{align} \label{eq:mod-pure-rolling}\tag{9} \left\{ \begin{array}{ll} v^r_{P,r} &= R \dot \varphi_{r}\\ v^r_{P,l} &= R \dot \varphi_{l} \end{array} \right. \end{align}

+ comment

LP - the overloading of r in these equations is very confusing

Recalling that the robot is assumed to be a rigid body, the velocity of point P in the inertial frame can be expressed as the sum of the translational velocity $\avec{v_A}$ and that of the rotating field $\avec{w_P^I} = L \dot \theta$ due to the robot's rotation. The $X_I,Y_I $ components of $\avec{v_P}$ can therefore be expressed as:

\begin{align} \label{eq:mod-pure-rolling-inertial-left}\tag{10} \left\{ \begin{array}{ll} \dot x_{P,r} &= \dot x_A + L\dot \theta \cos \theta \\ \dot y_{P,r} &= \dot y_A + L \dot \theta \sin \theta \end{array} \right. , \end{align}

and

\begin{align} \label{eq:mod-pure-rolling-inertial-right}\tag{11} \left\{ \begin{array}{ll} \dot x_{P,l} &= \dot x_A + L\dot \theta \cos \theta \\ \dot y_{P,l} &= \dot y_A + L \dot \theta \sin \theta \end{array} \right. . \end{align}

By recalling \eqref{eq:mod-orthogonality-cond} and \eqref{eq:mod-no-lat-slip-constraint-r}, the expression of left and right wheel velocities in the robot frame can be summarized in the *pure rolling constraint* equation:

\begin{align} \label{eq:mod-pure-rolling-constraint}\tag{12} \left\{ \begin{array}{ll} \dot x_{P,r} \cos \theta + \dot y_{P,r} \sin \theta &= R \dot \varphi_r \\ \dot x_{P,l} \cos \theta + \dot y_{P,l} \sin \theta &= R \dot \varphi_l \end{array} \right.. \end{align}

6.3. Differential drive robot kinematic model

In a differential drive robot, controlling the wheels at different speeds generates a rolling motion of rate $\omega = \dot \theta$. In a rotating field there always is a fixed point, the *center of instantaneous curvature* (ICC), and all points at distance d from it will have a velocity given by ωd, and direction orthogonal to that of the line connecting the ICC and the wheels (i.e., the *axle*). Therefore, by looking at Figure 6.1, we can write:

\begin{align} \label{eq:mod-kin-1}\tag{13} \left\{ \begin{array}{l} \dot \theta (d-L) &= v_l \\ \dot \theta (d+L) &= v_r \end{array} \right., \end{align}

from which:

\begin{align} \label{eq:mod-kin-2}\tag{14} \left\{ \begin{array}{l} d &= L \frac{v_r^r + v^r_l}{v_r^r - v_l^r} \\ \dot \theta &= \frac{v_r^r - v_l^r}{2L} \end{array} \right.. \end{align}

A few observations stem from \eqref{eq:mod-kin-2}:

- If $v_r^r = v_l^r$ the bot does not turn ($\dot \theta = 0$), hence the ICC is not defined;
- If $v_r^r = - v_l^r$, then the robot "turns on itself", i.e., $d=0$ and $ICC \equiv A$;
- If $v_r^r = 0$ (or $v_l^r = 0$), the rotation happens around the right (left) wheel and $d = 2L$ ($d = L$).

> **Note:** Moreover, a differential drive robot cannot move in the direction of the ICC, it is a singularity.

By recalling the *no lateral slipping motion* \eqref{eq:mod-no-lat-slip-constraint-r} hypothesis and the *pure rolling* constraint \eqref{eq:mod-pure-rolling}, and noticing that the translational velocity of A in the robot frame is $\dot x_A^r = \dot \theta d$ we can write:

\begin{align} \label{eq:mod-kin-3}\tag{15} \left\{ \begin{array}{l} \dot x_A^r &= R(\dot \varphi_R +\dot \varphi_L)/2 \\ \dot y_A^r &= 0 \\ \dot \theta &= \omega = R(\dot \varphi_R - \dot \varphi_L)/(2L) \end{array} \right., \end{align}

+ comment
using \frac in the above yields ugly visual results (the dots on the phi's are too close and barely noticeable)

which in more compact yields the *forward kinematics* in the robot frame:

\begin{align} \label{eq:mod-forward-kinematics-robot-frame}\tag{16} \left[\begin{array}{c} \dot x_A^r \\ \dot y_A^r \\ \dot \theta \end{array} \right] = \left[\begin{array}{cc} \frac{R}{2} & \frac{R}{2} \\ 0 & 0 \\ \frac{R}{2L} & -\frac{R}{2L} \\ \end{array} \right] \left[\begin{array}{c} \dot \varphi_R \\ \dot \varphi_L \end{array} \right]. \end{align}

Finally, by using \eqref{eq:mod-rot-mat}, we can recast \eqref{eq:mod-forward-kinematics-robot-frame} in the inertial frame.

> **Note:** The *forward kinematics* model of a differential drive robot is given by: \begin{align} \label{eq:mod-forward-kinematics-inertial-frame}\tag{17} \displaystyle \avec{\dot q}^I = \amat{R}(\theta) \left[\begin{array}{c} \dot x_A^r \\ \dot y_A^r \\ \dot \theta \end{array} \right] = \left[\begin{array}{cc} \frac{R}{2} \cos \theta & \frac{R}{2} \cos \theta \\ \frac{R}{2} \sin \theta & \frac{R}{2} \sin \theta \\ \frac{R}{2L} & -\frac{R}{2L} \\ \end{array} \right] \left[\begin{array}{c} \dot \varphi_R \\ \dot \varphi_L \end{array} \right]. \end{align}

6.4. Differential drive robot dynamic model

While kinematics studies the properties of motions of geometric (i.e., massless) points, dynamical modeling takes into account the actual material distribution of the system. Once mass comes into play, motion is the result of the equilibrium of forces and torques. While different approaches can be used to derive these equations, namely the Lagrangian or Newtonian approaches (former based on energy considerations, latter on equilibrium of generalized forces), we choose to follow the Newtonian one here for it grants, arguably, a more explicit physical intuition of the problem. Obviously both methods lead to the same results when the same hypothesis are made.

1) Notations

For starters, recalling that $C^r = (c, 0)$ is the center of mass of the robot, we define the relevant notations:

TABLE 6.1. NOTATIONS FOR DYNAMIC MODELING OF A DIFFERENTIAL DRIVE ROBOT

(v_u, v_w)	Longitudinal and lateral velocities of C, robot frame
(a_u, a_w)	Longitudinal and lateral accelerations of C, robot frame
(F_{u_R}, F_{u_L})	Longitudinal forces exerted on the vehicle by the right and left wheels
(F_{w_R}, F_{w_L})	Lateral forces exerted on the vehicle by the right and left wheels
(τ_R, τ_L)	Torques acting on right and left wheel
θ, $\omega = \dot \theta$	Vehicle orientation and angular velocity
M	Vehicle mass
J	Vehicle yaw moment of inertia with respect to the center of mass C

2) Free body diagram

The next step, and definitely the most critical, is writing the free body diagram of the problem (Figure 6.3). In this analysis the only forces acting on the robot are those applied to the wheels.

Before proceeding with the equilibrium of forces and moments, it is appropriate to recall the expressions of velocities and accelerations of a rigid body in a rotating frame expressed in polar coordinates.

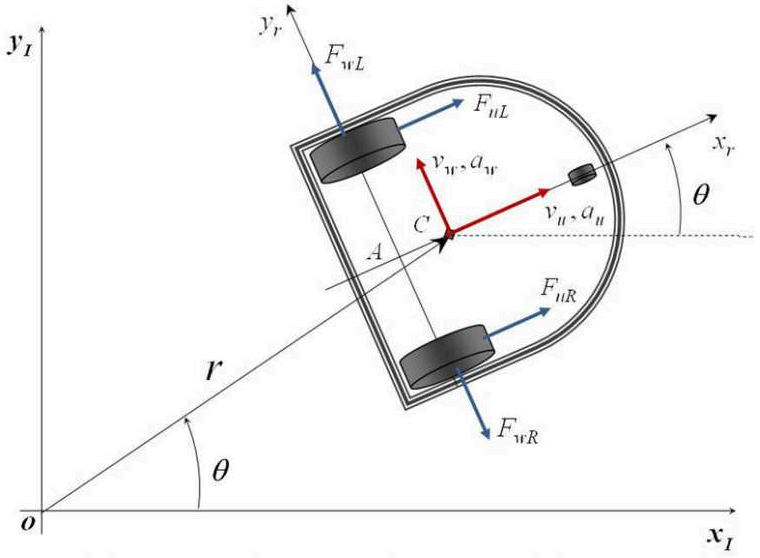


Figure 6.3. Free body diagram of a differential drive robot

3) Rotating frames in polar coordinates: a recap

Let the center of mass of the robot be identified by the vector $\avec{r}(t)$, as shown in Figure 6.3). The polar coordinated allow us to express $\avec{r}(t)$ using the complex notation:

$$ \label{eq:mod-dyn-r}\tag{4} \avec{r}(t) = r(t) e^{j\theta(t)}. $$

By differentiating in time \eqref{eq:mod-dyn-r}, and recalling the chain rule of derivatives, it is straightforward, although arguably boring to obtain the expression of radial velocity and acceleration, respectively:

\begin{align} \label{eq:mod-dyn-rd-rdd}\tag{18} \avec{\dot r}(t) &= \dot{r}(t) e^{j\theta(t)} + j r(t) \dot \theta(t) e^{j\theta(t)} \\ \avec{\ddot r}(t) &= \ddot{r}(t) e^{j\theta(t)} + 2 j \dot{r}(t) \dot \theta(t) e^{j\theta(t)} + j r(t) \ddot{\theta}(t) e^{j\theta(t)} - r(t) \dot \theta^2 e^{j\theta(t)}. \end{align}

By simplifying and writing the lateral components explicitly, \eqref{eq:mod-dyn-rd-rdd} becomes:

\begin{align} \label{eq:mod-dyn-rd-rdd2}\tag{19} \avec{\dot r}(t) &= v_u(t) e^{j\theta(t)} + v_w (t) e^{j\left(\theta (t)+\frac{\pi}{2}\right)} \\ \avec{\ddot r}(t) &= a_u(t) e^{j\theta(t)} + a_w (t) e^{j\left(\theta (t)+\frac{\pi}{2}\right)}, \end{align}

where:

\begin{align} \label{eq:mod-dyn-polar-v-dv}\tag{20} v_u(t) &= \dot{r}(t) \\ v_w(t) &= r(t) \dot \theta(t) \\ a_u(t) &= \ddot{r}(t) - r(t) \dot \theta^2 \\ a_w(t) &= 2 \dot{r}(t) \dot \theta(t) + r(t) \ddot \theta. \end{align}

Exercise: prove that $je^{j\theta}=e^{j(\theta+\pi/2)}$.

4) Equilibrium of forces and moments

We derive the dynamic model by imposing the simultaneous equilibrium of forces along the longitudinal and lateral directions in the robot frame with the respective

inertial forces, and of the moments around the vertical axis (coming out of the.. screen) passing through the center of mass of the robotic vehicle.

\begin{align} \label{eq:mod-dyn-equilibria}\tag{21} Ma_u(t) &= F_{u_L}+F_{u_R} \\ Ma_w(t) &= F_{w_L}-F_{w_R} \\ \ddot{\theta}(t) &= \frac{L}{J}(F_{u_R}-F_{u_L}) + \frac{c}{J}(F_{w_R}-F_{w_L}) \end{align}

By substituting the \eqref{eq:mod-dyn-polar-v-dv} in \eqref{eq:mod-dyn-equilibria}, the equilibrium of forces equations are expressed in terms of accelerations of the center of mass in the robot frame:

\begin{align} \dot{v}_u(t) &= v_w \dot{\theta}(t) + \frac{F_{u_L}+F_{u_R}}{M} \label{eq:mod-dyn-equilibria2a}\tag{22} \\ \dot{v}_w(t) &= -v_u \dot{\theta}(t)+\frac{F_{w_L}-F_{w_R}}{M} \label{eq:mod-dyn-equilibria2b}\tag{23} \\ \ddot{\theta}(t) &= \frac{L}{J}(F_{u_R}-F_{u_L}) + \frac{c}{J}(F_{w_R}-F_{w_L}). \label{eq:mod-dyn-equilibria2c}\tag{24} \end{align}

This general equation does not yet account for the the kinematic constraints discuss earlier.

5) Imposing the kinematic constraints

Equations \eqref{eq:mod-dyn-equilibria} of motion can be decoupled by imposing the kinematic constraints \eqref{eq:mod-no-lat-slip-constraint-r} and \eqref{eq:mod-pure-rolling}. In particular, to impose the no lateral slipping hypothesis \eqref{eq:mod-no-lat-slip-constraint-r}, we need to express the velocity of A in the local frame and set it to zero.

\begin{align} \label{eq:mod-dyn-vC-to-vA}\tag{25} x_C &= x_A + c \cos\theta \\ y_C &= y_A + c \sin\theta \end{align}

TODO: clarify this passage, see slides

then recall that through the rotation matrix $\amat{R}(t)$:

\begin{align} \label{eq:mod-dyn-constraints-rot-refresh}\tag{26} \left[\begin{array}{c} \dot x_C \\ \dot y_C \end{array} \right] = \left[\begin{array}{cc} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{array} \right] \left[\begin{array}{c} v_u \\ v_w \end{array} \right]. \end{align}

With these two conditions, it can be shown that $\dot y_A^r = v_w - c \dot \theta$. Hence, by imposing $\dot y_A^r = 0 \Rightarrow v_w = c \dot \theta$.

By using this condition in \eqref{eq:mod-dyn-equilibria2a} and \eqref{eq:mod-dyn-equilibria2b}, and combining with \eqref{eq:mod-dyn-equilibria2c}, we obtain the dynamical equations of a differential drive robot under the aforementioned non holonomic constraints:

\begin{align} \label{eq:mod-dyn-model-a}\tag{27} \dot{v}_u (t) &= c \dot \theta^2 + \frac{1}{M} (F_{u_L}+F_{u_R}) \\ \ddot \theta &= \frac{L}{Mc^2+J}(F_{u_R}-F_{u_L}) + \frac{Mcv_u}{Mc^2+J} \dot \theta. \end{align}

Although \eqref{eq:mod-dyn-model-a} describes the dynamics of the robot, the input forces are difficult to measure. As it will be clearer from the next section where a model of the DC motor will be provided, it is more practical to consider the torques $(\tau_R, \tau_L) = (RF_{u_R}, RF_{u_L})$ in \eqref{eq:mod-dyn-model-a} as inputs to this system. By implementing this consideration and rearranging in matrix form:

\begin{align} \label{eq:mod-dyn-matrix-form}\tag{28} \left[\begin{array}{cc} M & 0

\\ 0 & Mc^2+J \end{array} \right] \left[\begin{array}{c} \dot v_u \\ \ddot \theta \end{array} \right]+ \left[\begin{array}{cc} 0 & -Mc\dot\theta \\ Mc\dot\theta & 0 \end{array} \right] \left[\begin{array}{c} v_u \\ \dot \theta \end{array} \right] = \frac{1}{R}\left[\begin{array}{cc} 1 & 1 \\ L & -L \end{array} \right] \left[\begin{array}{c} \tau_R \\ \tau_L \end{array} \right]. \end{align}

This model now describes the input output relationship between torques and robot orientation and forward velocity in polar coordinates and can be further manipulated to obtain relations between the input torques and relevant variables such as the angular rates of the wheels.

But there is still a missing link, as the actual control input to the robot is not the torque, but the voltage applied to the DC motors. It is hence necessary to model the actuator dynamics.

6.5. DC motor dynamic model

The equations governing the behavior of a DC motor are driven by an input *armature voltage* $V(t)$:

\begin{align} V(t) &= Ri(t) + L \frac{di}{dt} + e(t) \tag{electrical equation} \\ e(t) &= K_b \dot\varphi(t) \tag{back electromotive force (emf)} \\ \tau_m(t) &= K_t i(t) \tag{toque equation} \\ \tau(t) &= N \tau_m(t) \tag{gear reduction equation}, \end{align}

where (K_b, K_t) are the back emf and torque constants respectively and N is the gear ratio ($N=1$ in the Duckiebot).

Figure 6.4 shows a diagram of a typical DC motor.

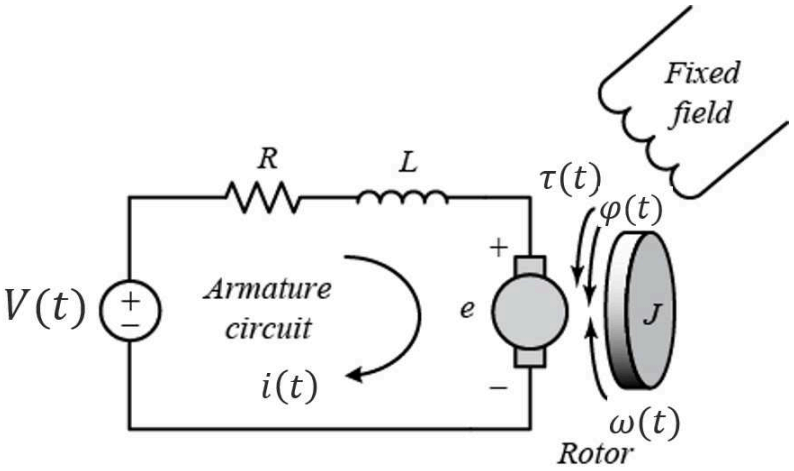


Figure 6.4. Diagram of a DC motor

Having a relation between the applied voltage and torque, in addition to the dynamic and kinematic models of a differential drive robot, allows us to determine all possible state variables of interest.

> Note: torque disturbances acting on the wheels, such as the effects of friction, can be modeled as additive terms (of sign opposite to τ) in the DC motor equations.

6.6. Conclusions

This chapter showed the methodology for which to achieve a model of a differential drive robot. Although simplifying assumption were made, e.g., rigid body motion, symmetry, pure rolling and no lateral slipping - still the model is nonlinear.

Regardless, we now have a chain of descriptive tools that receive as input the voltage signal sent by the controller, and produce as output any of the state variables, e.g., the position, velocity and orientation of the robot with respect to a fixed inertial frame.

Several outstanding questions remain. For example, we need to determine what is the best representation for our robotic platform - polar coordinates, Cartesian with respect to an arbitrary reference point? Or maybe there is a better choice?.

Finally, the above model assumes the knowledge of a number of constants that are characteristic of the robot's geometry, materials, and the DC motors. Without the knowledge of those constant the model could not work well. Determination of these parameters in a measurement driven way, i.e., the "system identification" of the robot's plant, is subject of the *odometry* class.

UNIT G-7

Computer vision basics [draft]

This section has been removed because it is in status 'draft'. If you are feeling adventurous, you can read it on master.

To disable this behavior, and completely hide the sections, set the parameter show_removed to false in fall2017.version.yaml.

UNIT G-8

Camera geometry [draft]

This section has been removed because it is in status 'draft'. If you are feeling adventurous, you can read it on master.

To disable this behavior, and completely hide the sections, set the parameter show_removed to false in fall2017.version.yaml.

UNIT G-9

Camera calibration [draft]

This section has been removed because it is in status 'draft'. If you are feeling adventurous, you can read it on master.

To disable this behavior, and completely hide the sections, set the parameter show_removed to false in fall2017.version.yaml.

UNIT G-10

Image filtering [draft]

This section has been removed because it is in status 'draft'. If you are feeling adventurous, you can read it on master.

To disable this behavior, and completely hide the sections, set the parameter show_removed to false in fall2017.version.yaml.

PART H
Reference Material

UNIT H-1
Configuration management [draft]

This section has been removed because it is in status 'draft'. If you are feeling adventurous, you can read it on master.

To disable this behavior, and completely hide the sections, set the parameter show_removed to false in fall2017.version.yaml.

PART I
Operation manual - Duckiebot

In this section you will find information to obtain the necessary equipment for Duckietowns and different Duckiebot configurations.

UNIT I-1
Duckiebot configurations

KNOWLEDGE AND ACTIVITY GRAPH

Requires: nothing

Results: Knowledge of Duckiebot configuration naming conventions, their components and functionalities.

Next: After reviewing the configurations, you can proceed to purchasing the components, reading a description of the components, or assembling your chosen configuration.

We define different Duckiebot configurations depending on their time of use and hardware components. This is a good starting point if you are wondering what parts you should obtain to get started.

1.1. Duckiebot Configurations: Fall 2017

The configurations are defined with a root: `DB17-`, indicating the "bare bones" Duckiebot used in the Fall 2017 synchronized course, and an appendix `y` which can be the

union (in any order) of any or all of the elements of the optional hardware set $\aset{O} = \{$`w`, `j`, `d`, `p`, `l`, `c`$\}$.

The elements of $\aset{O}$ are labels identifying optional hardware that aids in the development phase and enables the Duckiebot to talk to other Duckiebots. The labels stand for:

- `w`: 5 GHz **w**ireless adapter to facilitate streaming of images;
- `j`: wireless **j**oypad that facilitates manual remote control;
- `d`: USB **d**rive for additional storage space;
- `c`: a different **c**astor wheel to *replace* the preexisting omni-directional wheel;
- `l`: includes **L**EDs, LED hat, bumpers and the necessary mechanical bits to set the bumpers in place. Note that the installation of the bumpers induces the *replacement* of a few `DB17` components;

> Note: During the Fall 2017 course, three Duckietown Engineering Co. branches (Zurich, Montreal, Chicago) are using these configuration naming conventions. Moreover, all institutions release hardware to their Engineers in training in two phases. We summarize the configuration releases below.

1.2. Configuration functionality

1) DB17

This is the minimal configuration for a Duckiebot. It is the configuration of choice for tight budgets or when operation of a single Duckiebot is more of interest than fleet behaviors.

- **Functions**: A `DB17` Duckiebot can navigate autonomously in a Duckietown, but cannot communicate with other Duckiebots.
- **Components**: A "bare-bones" `DB17` configuration includes:

Table 1.1. Components of the DB17 configuration

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
DC Motor HAT	USD 22.50
Spliced USB-A power cable	USD 0
2 Stacking Headers	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
Total cost for `DB17` configuration	USD 173.6

- **Description of components**: Unit I-2 - Acquiring the parts for the Duckiebot `DB17-wjd`

- **Assembly instructions**: Unit I-4 - Assembling the Duckiebot DB17, DB17-jwd

2) DB17-w

This configuration is the same as DB17 with the *addition* of a 5 Ghz wireless adapter.

- **Functions**: This configuration has the same functionality of DB17. In addition, it equips the Duckiebot with a secondary, faster, Wi-Fi connection, ideal for image streaming.
- **Components**:

Table 1.2. Components of the DB17-w configuration

DB17	USD 173.6
Wireless Adapter (5 GHz)	USD 20
Total cost for DB17-w configuration	USD 193.6

- **Description of components**: Unit I-2 - Acquiring the parts for the Duckiebot DB17-wjd
- **Assembly instructions**: Unit I-4 - Assembling the Duckiebot DB17, DB17-jwd

3) DB17-j

This configuration is the same as DB17 with the *addition* of a 2.4 GHz wireless joypad.

- **Functions**: This configuration has the same functionality of DB17. In addition, it equips the Duckiebot with manual remote control capabilities. It is particularly useful for getting the Duckiebot our of tight spots or letting younger ones have a drive, in addition to providing handy shortcuts to different functions in development phase.
- **Components**:

Table 1.3. Components of the DB17-j configuration

DB17	USD 173.6
Joypad	USD 10.50
Total cost for DB17-j configuration	USD 184.1

- **Description of components**: Unit I-2 - Acquiring the parts for the Duckiebot DB17-wjd
- **Assembly instructions**: Unit I-4 - Assembling the Duckiebot DB17, DB17-jwd

4) DB17-d

This configuration is the same as DB17 with the *addition* of a USB flash hard drive.

- **Functions**: This configuration has the same functionality of DB17. In addition, it equips the Duckiebot with an external hard drive that 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 the day and bring it over to a computer for downloading and analyzing stored data.
- **Components**:

Table 1.4. Components of the DB17-d configuration

DB17	USD 173.6
Tiny 32GB USB Flash Drive	USD 12.50
Total cost for DB17-d configuration	USD 186.1

- **Description of components**: Unit I-2 - Acquiring the parts for the Duckiebot DB17-wjd
- **Assembly instructions**: Unit I-4 - Assembling the Duckiebot DB17, DB17-jwd

5) DB17-c

In this configuration, the DB17 omni-directional wheel is *replaced* with a caster wheel.

- **Functions**: The caster wheel upgrade provides a smoother ride.
- **Components**:

Table 1.5. Components of the DB17-c configuration

DB17	USD 173.6
Caster (DB17-c)	USD 6.55/4 pieces
4 Standoffs (M3 12mm F-F)	USD 0.63/piece
8 Screws (M3x8mm)	USD 4.58/100 pieces
8 Split washer lock	USD 1.59/100 pieces
Total cost for DB17-c configuration	USD 178.25

TODO: update links of mechanical bits from M3.5 to M3.

> Note: The omni-directional caster wheel is included in the chassis package, so replacing it does not reduce the DB17 cost.

- **Description of components**: Unit J-1 - Acquiring the parts for the Duckiebot DB17-1c
- **Assembly instructions**: Unit J-3 - Assembling the Duckiebot DB17-wjdlc [draft]

6) DB17-1

In this configuration the Duckiebot in equipped with the necessary hardware for controlling and placing 5 RGB LEDs on the Duckiebot. Differently from previous configurations that add or replace a single component, DB17-1 introduces several hardware components that are all necessary for a proper use of the LEDs.

It may be convenient at times to refer to hybrid configurations including any of the DB17-jwcd in conjunction with a *subset* of the DB17-1 components. In order to disambiguate, let the partial upgrades be defined as:

- DB17-11: *adds* a PWM hat to DB17, in addition to a short USB angled power cable and a M-M power wire;
- DB17-12: *adds* a bumpers set to DB17, in addition to the mechanical bits to assemble it;
- DB17-13: *adds* a LED hat and 5 RGB LEDs to DB17-1112, in addition to the F-F wires to connect the LEDs to the LED board.

> Note: introducing the PWM hat in DB17-11 induces a *replacement* of the spliced cable powering solution for the DC motor hat. Details can be found in Unit J-3 - Assembling the Duckiebot DB17-wjdlc [draft].

- **Functions**: DB17-1 is the necessary configuration to enable communication between Duckiebots, hence fleet behaviors (e.g., negotiating the crossing of an intersection). Subset configurations are sometimes used in a standalone way for: (DB17-11) avoid using a sliced power cable to power the DC motor hat in DB17, and (DB17-12) for purely aesthetic reasons.
- **Components**:

Table 1.6. Components of the DB17-l configuration

DB17	USD 173.6
PWM/Servo HAT (DB17-11)	USD 17.50
Power Cable (DB17-11)	USD 7.80
Male-Male Jumper Wire (150mm) (DB17-11)	USD 1.95
Bumper set (DB17-12)	USD 7 (custom made)
8 M3x10 pan head screws (DB17-12)	USD 7 (custom made)
8 M3 nuts (DB17-12)	USD 7 (custom made)
Bumpers (DB17-12)	USD 7 (custom made)
LEDs (DB17-13)	USD 10
LED HAT (DB17-13)	USD 28.20 for 3 pieces
20 Female-Female Jumper Wires (300mm) (DB17-13)	USD 8
4 4 pin female header (DB17-13)	USD 0.60/ piece
2 16 pin male [12 pin male header](http://www.digikey.com/product-detail/en/amphenol-fci/68000-412HLF/609-3266-ND/1878525) (DB17-13)USD 0.48/piece header (DB17-13)	USD 0.61/ piece
3 pin male header (DB17-13)	USD 0.10/ piece
2 pin female shunt jumper (DB17-13)	USD 2/piece
40 pin female header (DB17-13)	USD 1.50
5 200 Ohm resistors (DB17-13)	USD 0.10/ piece
10 130 Ohm resistors (DB17-13)	USD 0.10/ piece
Total for DB17-1 configuration	USD 305

- **Description of components**: Unit J-1 - Acquiring the parts for the Duckiebot DB17-1c
- **Assembly instructions**: Unit J-3 - Assembling the Duckiebot DB17-wjd1c [draft]

1.3. Branch configuration releases: Fall 2017

All branches release their hardware in two phases, namely `a` and `b`.

1) Zurich

- First release (`DB17-Zurich-a`): is a `DB17-wjd`.
- Second release (`DB17-Zurich-b`): is a `DB17-wjdcl`.

2) Montreal

- First release (`DB17-Montreal-a`): is a hybrid `DB17-wjd` + PWM hat (or `DB17-wjdl1`).
- Second release (`DB17-Montreal-b`): is a `DB17-wjdl`.

Note: The Montreal branch is not implementing the `DB17-c` configuration.

3) TTIC

- First release (`DB17-Chicago-a`): same as `DB17-Montreal-a`.
- Second release (`DB17-Chicago-b`): same as `DB17-Montreal-b`.

Note: The Chicago branch is not implementing the `DB17-c` configuration.

UNIT I-2

Acquiring the parts for the Duckiebot `DB17-wjd`

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.
- For some components, the links we provide contain more bits than actually needed.

KNOWLEDGE AND ACTIVITY GRAPH

Requires: Cost: USD 174 + Shipping Fees (minimal configuration `DB17`)

Requires: Time: 15 days (average shipping for cheapest choice of components)

Results: A kit of parts ready to be assembled in a `DB17` or `DB17-wjd` configuration.

Next: After receiving these components, you are ready to do some soldering (master) before assembling your `DB17` or `DB17-wjd` Duckiebot.

2.1. Bill of materials

Table 2.1. 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
DC Motor HAT	USD 22.50
2 Stacking Headers	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) (`DB17-w`)	USD 20
Joypad (`DB17-j`)	USD 10.50
Tiny 32GB USB Flash Drive (`DB17-d`)	USD 12.50
Total for `DB17` configuration	USD 173.6
Total for `DB17-w` configuration	USD 193.6
Total for `DB17-j` configuration	USD 184.1
Total for `DB17-d` configuration	USD 186.1
Total for `DB17-wjd` configuration	USD 216.6

2.2. Chassis

We selected the Magician Chassis as the basic chassis for the robot (Figure 2.1).

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 2 DC motors and wheels as well as the structural part, in addition to a screwdriver and several necessary mechanical bits (standoffs, screws and nuts).

Figure 2.1. The Magician Chassis

2.3. Raspberry Pi 3 - Model B

The Raspberry Pi is the central computer of the Duckiebot. Duckiebots use Model B

(Figure 2.2) (A1.2GHz 64-bit quad-core ARMv8 CPU, 1GB RAM), a small but powerful computer.

Figure 2.2. The Raspberry Pi 3 Model B

1) Power Supply

We want a hard-wired power source (5VDC, 2.4A, Micro USB) to supply the Raspberry Pi (Figure 2.3) while not driving. This charger can double down as battery charger as well.

Figure 2.3. The Power Supply

> **Note:** Students in the ETHZ-Fall 2017 course will receive a converter for US to CH plug.

2) Heat Sinks

The Raspberry Pi will heat up significantly during use. It is warmly recommended to add heat sinks, as in Figure 2.4. 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.

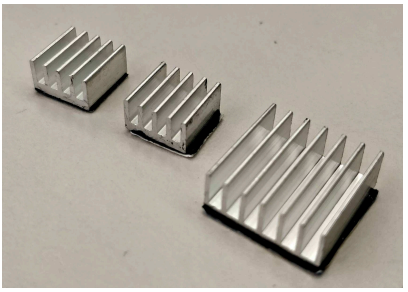


Figure 2.4. The Heat Sinks

3) Class 10 MicroSD Card

The MicroSD card (Figure 2.5) is the hard disk of the Raspberry Pi. 16 GB of capacity are sufficient for the system image.

Figure 2.5. The MicroSD card

4) Mirco SD card reader

A microSD card reader (Figure 2.6) 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 2.6. The Mirco SD card reader

2.4. 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 2.7).

Figure 2.7. The Camera with Fisheye Lens

1) Camera Mount

The camera mount (Figure 2.8) serves to keep the camera looking forward at the right angle to the road (looking slightly down). The front cover is not essential.

Figure 2.8. The Camera Mount

The assembled camera (without camera cable), is shown in (Figure 2.9).

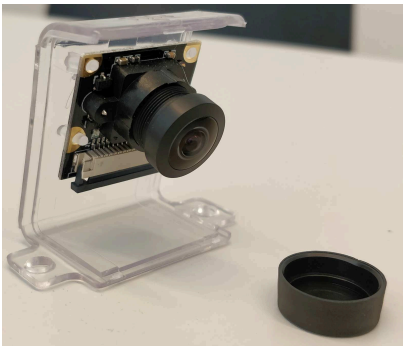


Figure 2.9. The Camera on its mount

2) 300mm Camera Cable

A longer (300 mm) camera cable Figure 2.10 makes assembling the Duckiebot easier, allowing for more freedom in the relative positioning of camera and computational stack.

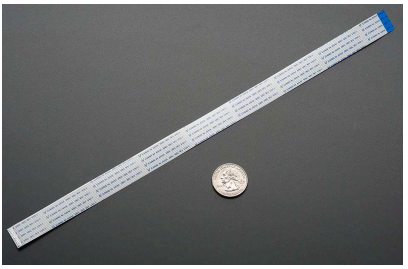


Figure 2.10. A 300 mm camera cable for the Raspberry Pi

2.5. DC Motor HAT

We use the DC Stepper motor HAT (Figure 2.11) to control the DC motors that drive the wheels. This item will require soldering to be functional. This HAT has dedicate PWM and H-bridge for driving the motors.

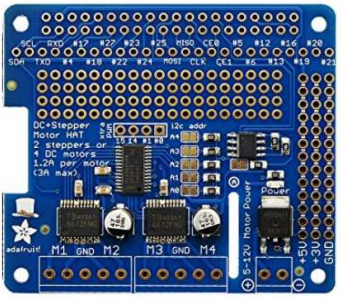


Figure 2.11. The Stepper Motor HAT

1) Stacking Headers

We use a long 20x2 GPIO stacking header (Figure 2.12) to connect the Raspberry Pi with the DC Motor HAT. This item will require soldering (master) to be functional.

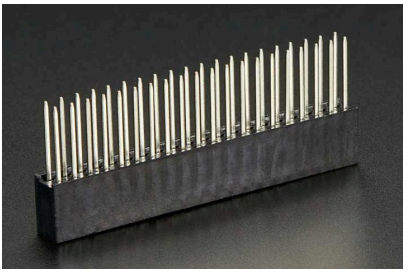


Figure 2.12. The Stacking Headers

2.6. Battery

The battery (Figure 2.13) 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. The battery linked in the table above comes with two USB to microUSB cables.

Figure 2.13. The Battery

2.7. 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.

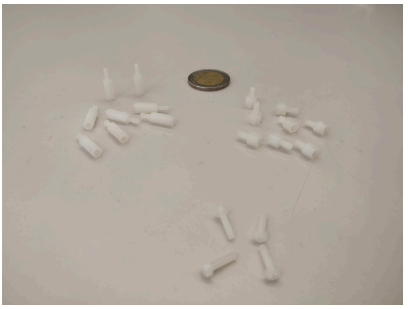


Figure 2.14. Standoffs, Nuts and Screws

2.8. Zip Tie

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

Figure 2.15. The zip ties

2.9. Configuration DB17-w

1) Wireless Adapter (5 GHz)

The Edimax AC1200 EW-7822ULC 5 GHz wireless adapter (Figure 2.16) boosts the connectivity of the Duckiebot, especially useful in busy Duckietowns (e.g., classroom). This additional network allows easy streaming of images.

Figure 2.16. The Edimax AC1200 EW-7822ULC wifi adapter

2.10. Configuration `DB17-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 2.17) does not include batteries.

Figure 2.17. A Wireless Joypad

Requires: 2 AA 1.5V batteries (Figure 2.18).

Figure 2.18. A Wireless Joypad

2.11. Configuration `DB17-d`

1) Tiny 32GB USB Flash Drive

In configuration `DB17-d`, the Duckiebot is equipped with an "external" hard drive (Figure 2.19). 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 2.19. The Tiny 32GB USB Flash Drive

UNIT I-3

Preparing the power cable for DB17

In configuration `DB17` we will need a cable to power the DC motor HAT from the battery. The keen observer might have noticed that such a cable was not included in the DB17 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 Unit I-1 - Duckiebot configurations.

It is important to note that these instructions are relevant only for assembling a `DB17-wjdc` configuration Duckiebot (or any subset of it). If you intend to build a `DB17-l` configuration Duckiebot, you can skip these instructions.

KNOWLEDGE AND ACTIVITY GRAPH

Requires: One male USB-A to anything cable.

Requires: A pair of scissors.

Requires: A multimeter (only if you are not purchasing the suggested components)

Requires: Time: 5 minutes

Results: One male USB-A to wires power cable

3.1. 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 Unit I-2 - Acquiring the parts for the Duckiebot `DB17-wjd`, you can use the longer USB cable contained inside the bat-

tery package (Figure 3.1), which will be used as an example in these instructions.

Figure 3.1. The two USB cables in the suggested battery pack.

Put the shorter cable back in the box, and open the longer cable (Figure 3.2)

Figure 3.2. Take the longer cable, and put the shorter on back in the box.

3.2. 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 3.3) at the desired length from the USB-A port.

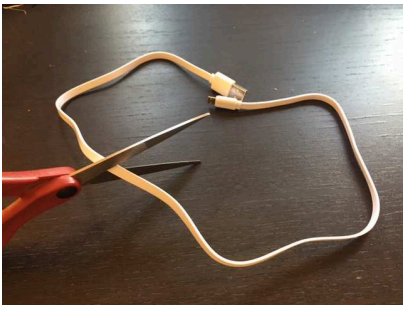


Figure 3.3. Cut the USB cable using the scissors.

The cut will look like in Figure 3.4.

Figure 3.4. A cut USB cable.

3.3. 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 3.5.

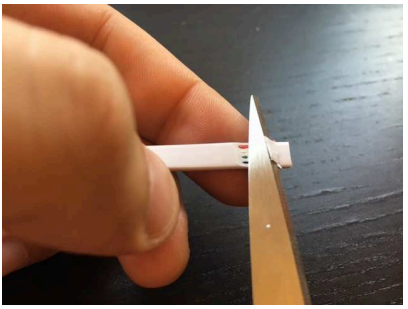


Figure 3.5. Stripping the external layer of the USB cable.

After removing the external plastic, you will see four wires: black, green, white and red (Figure 3.6).

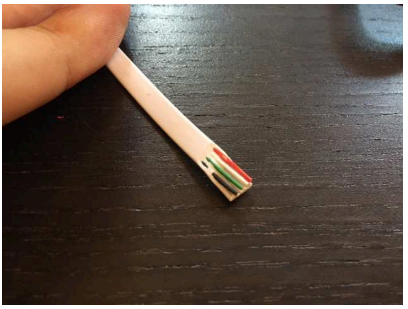


Figure 3.6. Under the hood of a USB-A cable.

Once the bottom part of the external cable is removed, you will have isolated the four wires (Figure 3.7).

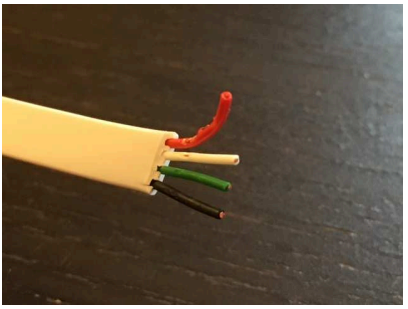


Figure 3.7. The four wires inside a USB-A cable.

3.4. 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 3.8).

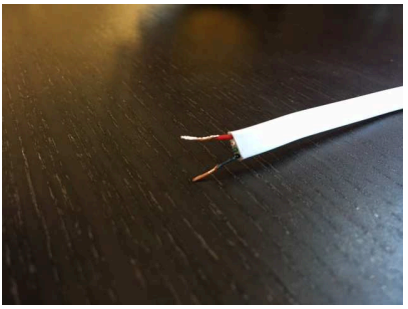


Figure 3.8. Strip the power wires and cut the data wires.

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

3.5. 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 that black and red actually are the power cables, take a multimeter and continue reading.

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 will 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 3.9).

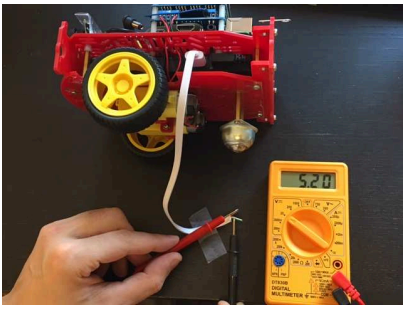


Figure 3.9. Finding which two wires are for power.

3.6. 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 Soldering boards for DB17 (master).

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 3.10).

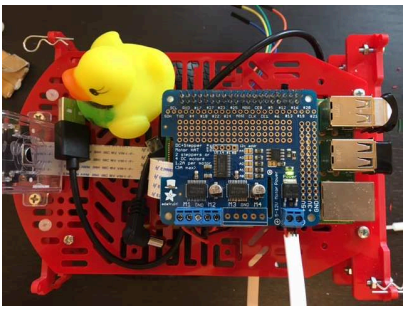


Figure 3.10. Connect the power wires to the DC motor HAT

UNIT I-4

Assembling the Duckiebot DB17, DB17-jwd

Point of contact: Shiying Li

Once you have received the parts and soldered the necessary components, it is time to assemble them in a Duckiebot. Here, we provide the assembly instructions for configurations `DB17-wjd`.

KNOWLEDGE AND ACTIVITY GRAPH

Requires: Duckiebot `DB17-wjd` parts. The acquisition process is explained in Unit I-2 - Acquiring the parts for the Duckiebot `DB17-wjd`.

Requires: Having soldered the `DB17-wjd` parts. The soldering process is explained in Soldering boards for DB17 (master).

Requires: Having prepared the power cable. The power cable preparation is explained in Unit I-3 - Preparing the power cable for DB17. Note: Not necessary if you intend to build a `DB17-1` configuration.

Requires: Having installed the image on the MicroSD card. The instructions on how to reproduce the Duckiebot system image are in Reproducing the image (master).

Requires: Time: about 40 minutes.

Results: An assembled Duckiebot in configuration `DB17-wjd`.

Note: The FAQ section at the bottom of this page may already answer some of you comments, questions or doubts.

Note: While assembling the Duckiebot, try to make as symmetric (along the longitudinal axis) as you can. It will help going forward.

4.1. Chassis

Open the Magician chassis package (Figure 4.1) and take out the following components:

- Chassis-bottom (1x), Chassis-up (1x);
- DC Motors (2x), motor holders (4x);
- Wheels (2x), steel omni-directional wheel (1x);
- All spacers and screws;
- Screwdriver.

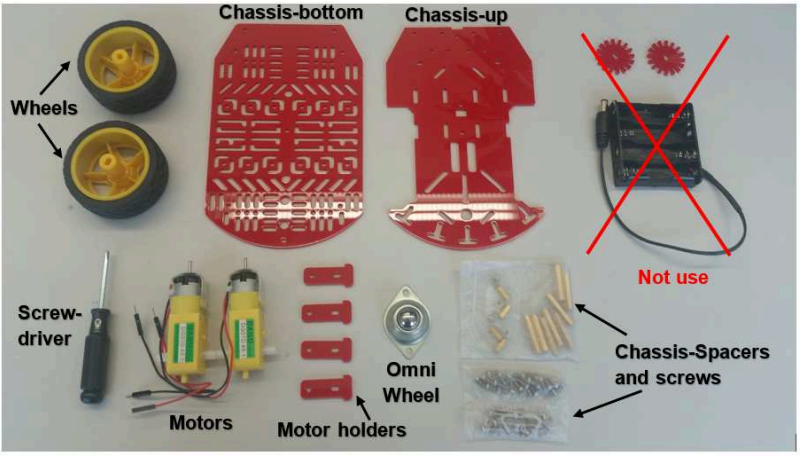


Figure 4.1. Components in Duckiebot package.

Note: You won't need the battery holder and speed board holder (on the right side in Figure 4.1).

1) Bottom

Insert the motor holders on the chassis-bottom and put the motors as shown in the figure below (with the longest screws (M3x30) and M3 nuts).

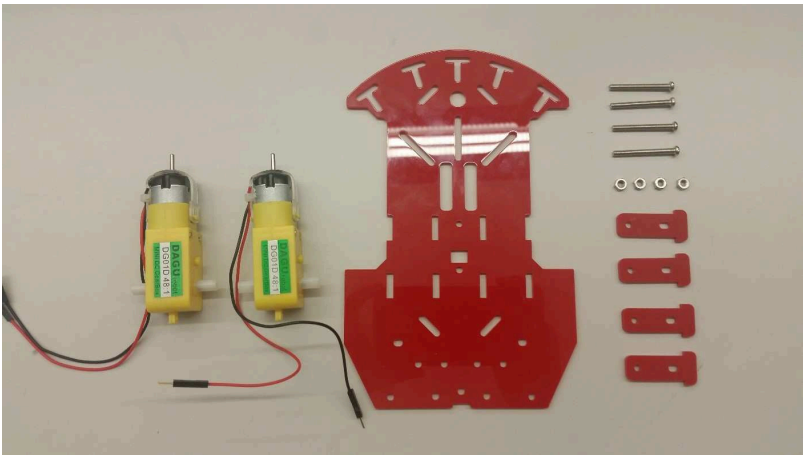


Figure 4.2. Components for mounting the motor

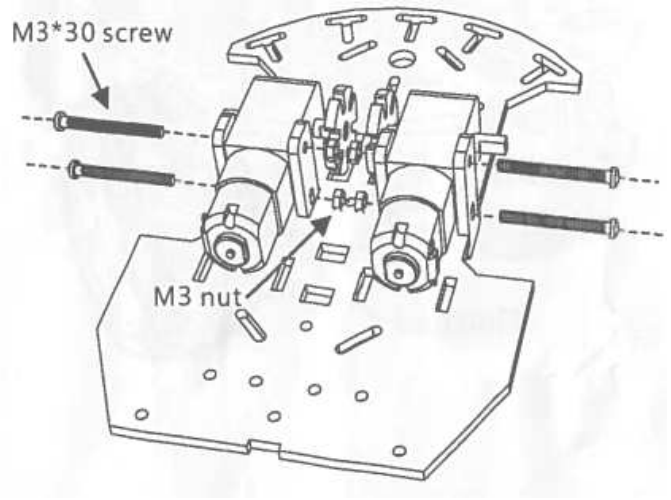


Figure 4.3. The scratch of assembling the motor

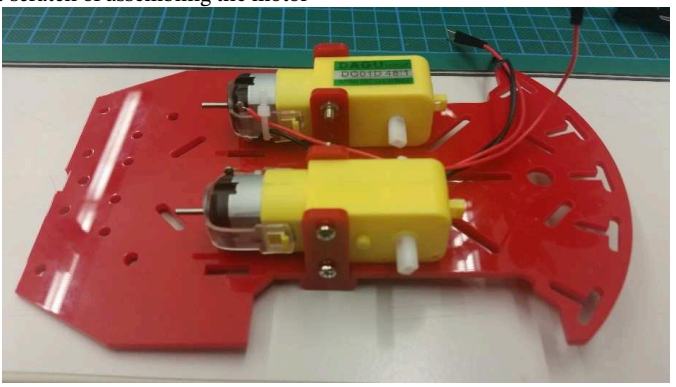


Figure 4.4. Assembled motor

> **Note:** Orient the motors so that their wires are inwards, i.e., towards the center of the chassis-bottom. The black wires should be closer to the chassis-bottom to make wiring easier down the line.

> Note: if your Magician Chassis package has unsoldered motor wires, you will have to solder them first. Check these instructions. In this case, your wires will not have the male pin headers on one end. Do not worry, you can still plug them in the stepper motor hat power terminals.

TODO: make instructions for soldering motor wires

2) Wheels

Plug in the wheels to the motor as follows (no screws needed):

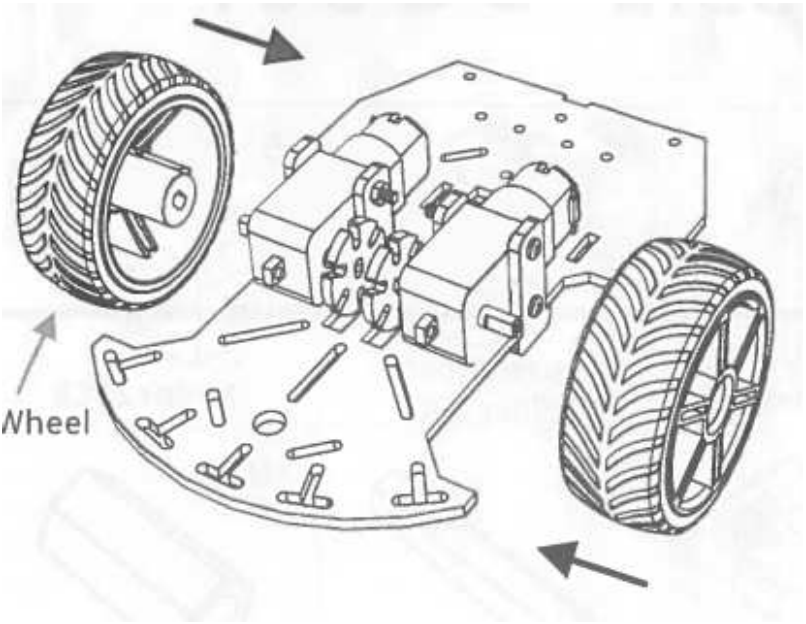


Figure 4.6. Wheel assembly schematics

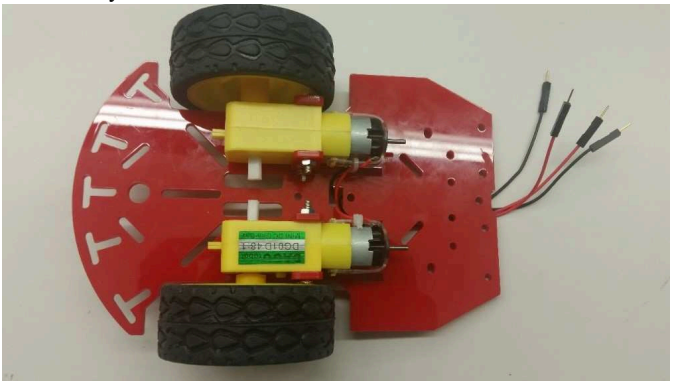


Figure 4.7. Assembled wheels

Figure 4.5. Wheel assembly instructions

3) Omni-directional wheel

The Duckiebot is driven by controlling the wheels attached to the DC motors. Still, it requires a “passive” omni-directional wheel (the *caster* wheel) on the back.

The Magician chassis package contains a steel omni-directional wheel, and the related standoffs and screws to secure it to the chassis-bottom part.

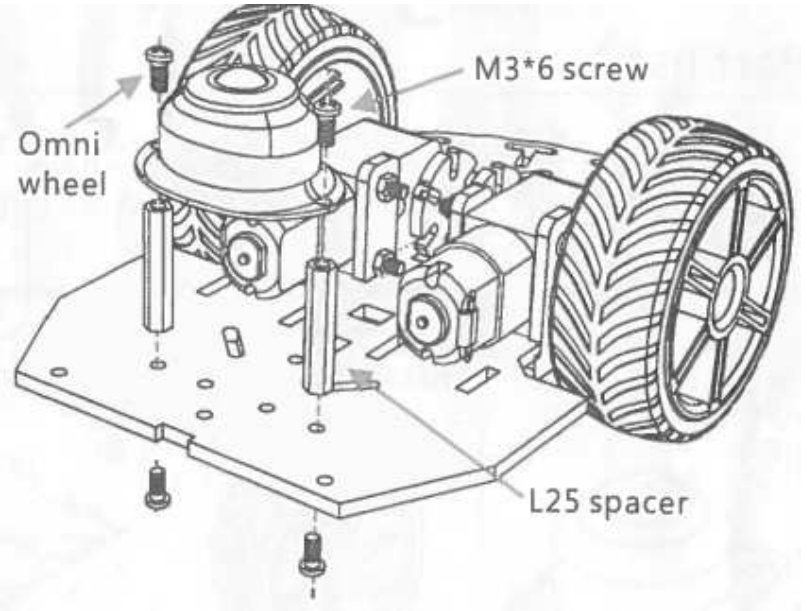


Figure 4.8. The omni-directional wheel schematics

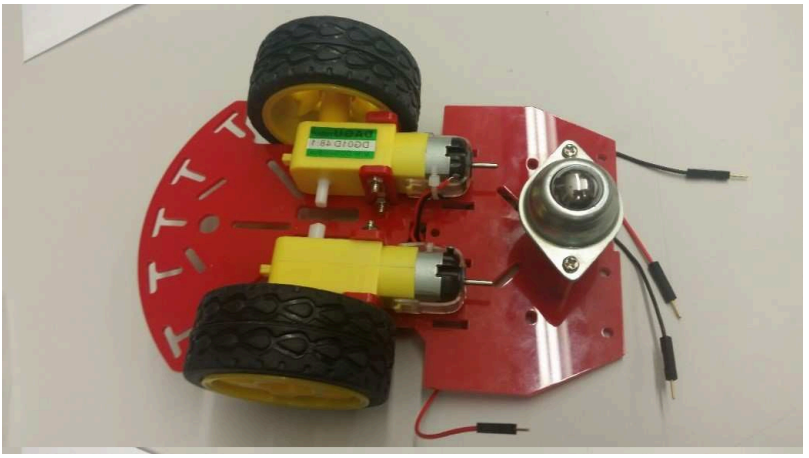


Figure 4.9. Assembled omni-directional wheel

4) Mounting the standoffs

Put the car upright (omni wheel pointing towards the table) and arrange wires so that they go through the center rectangle. Put 4 spacers with 4 of M3x6 screws on exact position of each corner as below Figure 4.11.

Figure 4.10. Metal spacers and M3x6mm screws

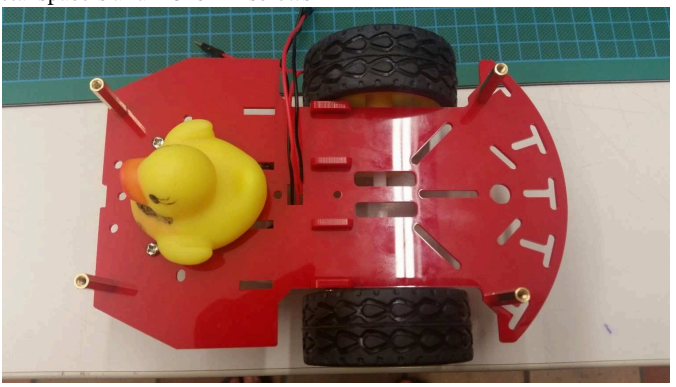


Figure 4.11. The spacers on each corner of the chassis-bottom

The bottom part of the Duckiebot's chassis is now ready. The next step is to assemble the Raspberry Pi on chassis-top part.

4.2. Assembling the Raspberry Pi, camera, and HATs

1) Raspberry Pi

Before attaching anything to the Raspberry Pi you should add the heat sinks to it. There are 2 small sinks and a big one. The big one best fits onto the processor (the big "Broadcom"-labeled chip in the center of the top of the Raspberry Pi). One of the small ones can be attached to the small chip that is right next to the Broadcom chip. The third heat sink is optional and can be attached to the chip on the underside of the Raspberry Pi. Note that the chip on the underside is bigger than the heat sink. Just mount the heat sink in the center and make sure all of them are attached tightly.

When this is done fasten the nylon standoffs on the Raspberry Pi, and secure them on the top of the chassis-up part by tightening the nuts on the opposite side of the chassis-up.

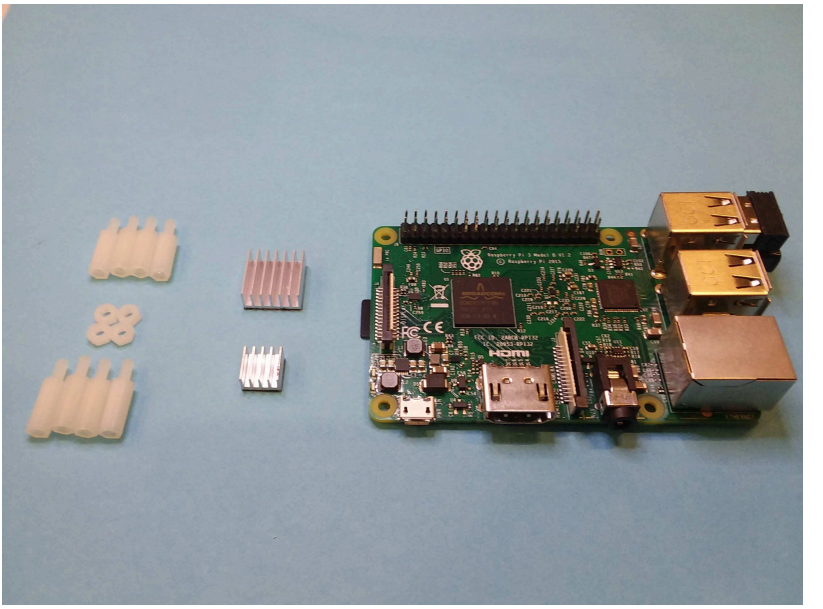


Figure 4.12. Components for Raspberry Pi3

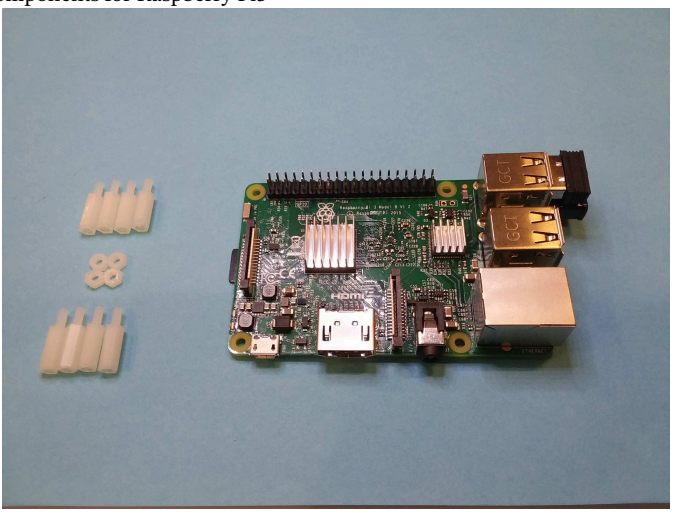


Figure 4.13. Heat sink on Raspberry Pi3

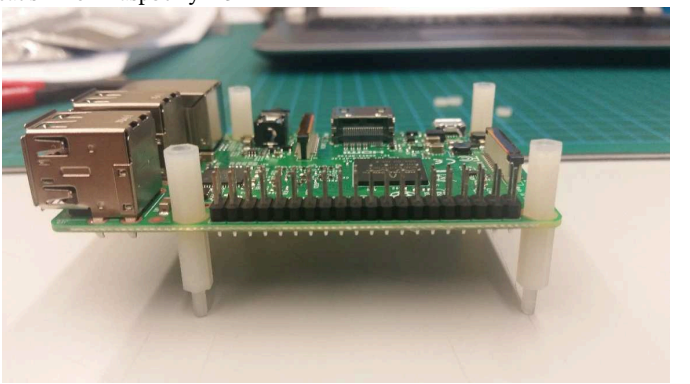


Figure 4.14. Nylon standoffs for Raspberry Pi3

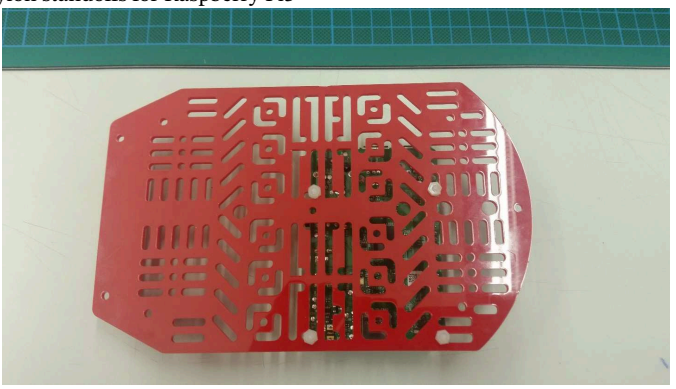


Figure 4.15. Attach the nylon huts for the standoffs (bottom view)

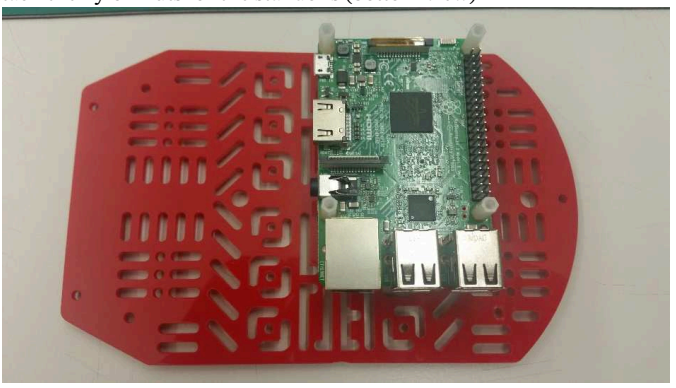


Figure 4.16. Assembled Raspberry Pi3 (top view)

2) Micro SD card

Requires: Having the Duckiebot image copied in the micro SD card.

Take the micro SD card from the Duckiebox and insert its slot on the Raspberry Pi. The SD card slot is just under the display port, on the short side of the PI, on the flip side of where the header pins are.

Figure 4.17. The micro SD card and mirco SD card readers

Figure 4.18. Inserted SD card

3) Camera

> **Note:** If you have camera cables of different lengths available, keep in mind that both are going to work. We suggest to use the longer one, and wrap the extra length under the Raspberry Pi stack.

The Raspberry Pi end:

First, identify the camera cable port on the Pi (between HDMI and power ports) and remove the orange plastic protection (it will be there if the Pi is new) from it. Then, grab the long camera cable (300 mm) and insert in the camera port. To do so, you will need to gently pull up on the black connector (it will slide up) to allow the cable to insert the port. Slide the connector back down to lock the cable in place, making sure it “clicks”.

TODO: insert image with long cable

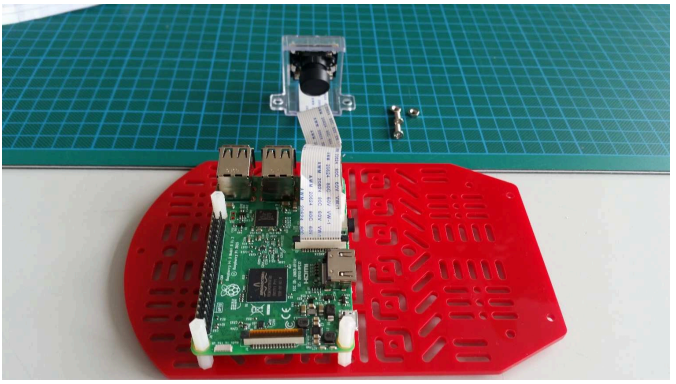


Figure 4.19. Camera port on the Raspberry Pi and camera cable

> **Note:** Make sure the camera cable is inserted in the right direction! The metal pins of the cable should be in contact with the metal terminals in the camera port of the PI.

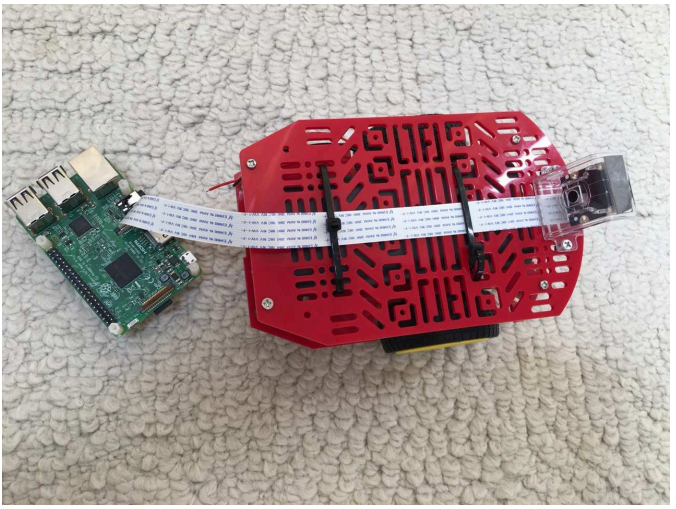


Figure 4.20. Camera with long cable

The camera end:

If you have the long camera cable, the first thing to do is removing the shorter cable that comes with the camera package. Make sure to slide up the black connectors of the camera-camera cable port in order to unblock the cable.

Take the rear part of the camera mount and use it hold the camera in place. Note that the camera is just press-fitted to the camera mount, no screws/nuts are needed.

In case you have not purchased the long camera cable, do not worry! It is still very possible to get a working configuration, but you will have little wiggling space and assembly will be a little harder.

Place the camera on the mount and fasten the camera mount on the chassis-up using M3x10 flathead screws and M3 nuts from the Duckiebox.

Protip: make sure that the camera mount is: (a) geometrically centered on the chassis-up; (b) fastened as forward as it can go; (c) it is tightly fastened. We aim at having a standardized position for the camera and to minimize the wiggling during move-

ment.

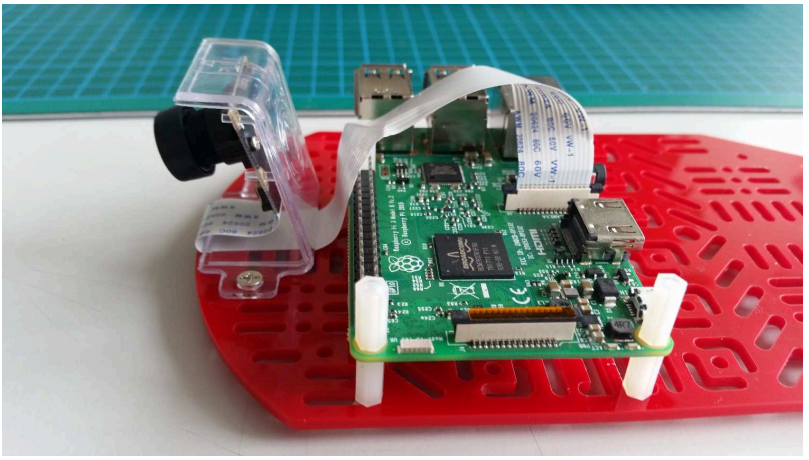


Figure 4.21. Raspberry Pi and camera with short cable

> **Note:** If you only have a short camera cable, make sure that the cable is oriented in this direction (text on cable towards the CPU). Otherwise you will have to disassemble the whole thing later. On the long cable the writing is on the other side.

4) Extending the intra-decks standoffs

In order to fit the battery, we will need to extend the Magician chassis standoffs with the provided nylon standoff spacers. Grab 4 of them, and secure them to one end of the long metal standoffs provided in the Magician chassis package.

Secure the extended standoff to the 4 corners of the chassis-bottom. The nylon standoffs should smoothly screw in the metal ones. If you feel resistance, don't force it or the nylon screw might break in the metal standoff. In that case, unscrew the nylon spacer and try again.

Figure 4.22. 4 nylon M3x5 extended standoffs and 4 M3x6 metal screws from Magician chassis package

5) Fasten the Battery with zip ties

Put the battery between the upper and lower decks of the chassis. It is strongly recommended to secure the battery from moving using zip ties.

Figure 4.23. Secure the battery to the chassis-top through the provided zipties. One can do the trick, two are better.

TODO: new image without M-M cable

6) Assemble chassis-bottom and chassis-up

Arrange the motor wires through the chassis-up, which will be connected to Stepper Motor HAT later.

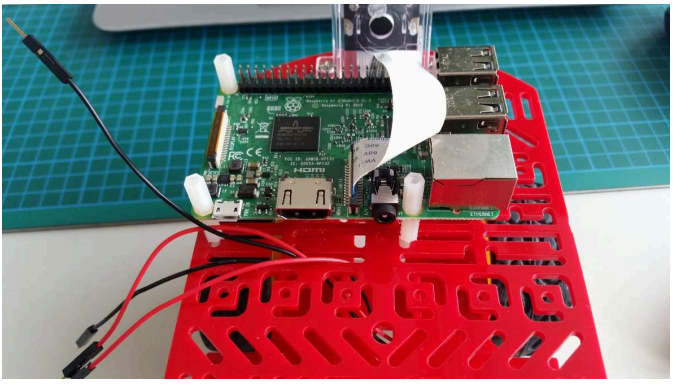


Figure 4.24. The motor wires go through the center of chassis-up

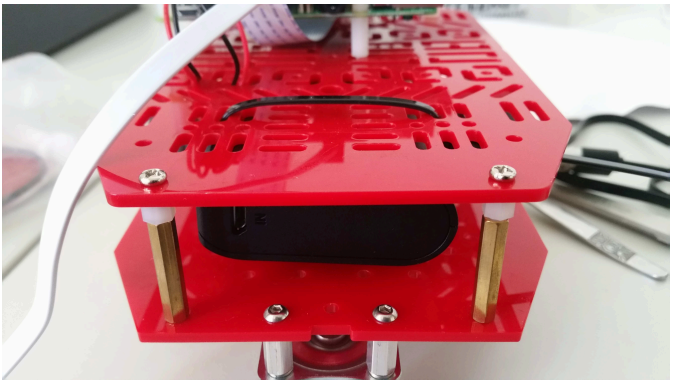


Figure 4.25. Side view of metal screws and the extended standoffs

> **Note:** Use the provided metal screws from chassis package for fastening the chassis-up above the nylon standoffs instead of the provided M3 nylon screws.

7) Place the DC Motor hat on top of the Raspberry Pi

Make sure the GPIO stacking header is carefully aligned with the underlying GPIO pins before applying pressure.

> **Note:** In case with short camera cable, ensure that you doesn't break the cable while mounting the HAT on the Raspberry Pi. In case with long camera cable,

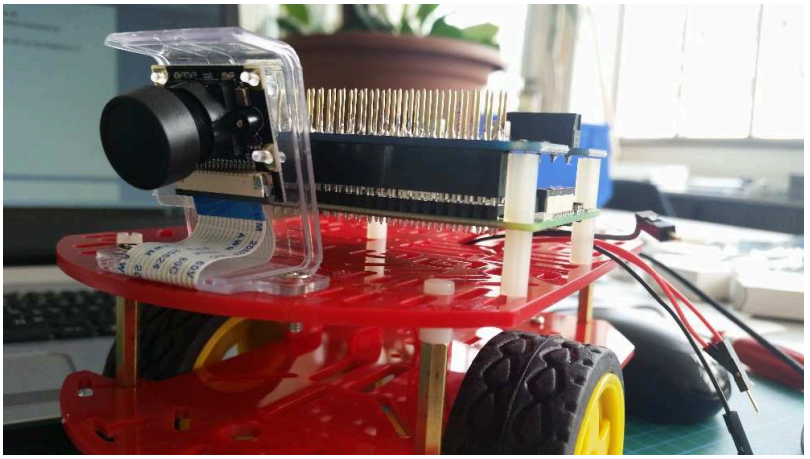


Figure 4.26. Assembled DC motor hat with short camera cable

TODO: insert pic with long camera cable

8) Connect the motor's wires to the terminal

We are using M1 and M2 terminals on the DC motor hat. The left (in robot frame) motor is connected to M1 and the right motor is connected to M2. If you have followed Part A correctly, the wiring order will look like as following pictures:

- Left Motor: Red
- Left Motor: Black
- Right Motor: Black
- Right Motor:Red

9) Connect the power cables

You are now ready to secure the prepared power wires in Unit I-3 - Preparing the power cable for DB17 to the DC motor HAT power pins.

Connect the **battery** (not the Raspberry Pi) 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 3.10).

Fix all the cables on the Duckiebot so that it can run on the way without barrier.

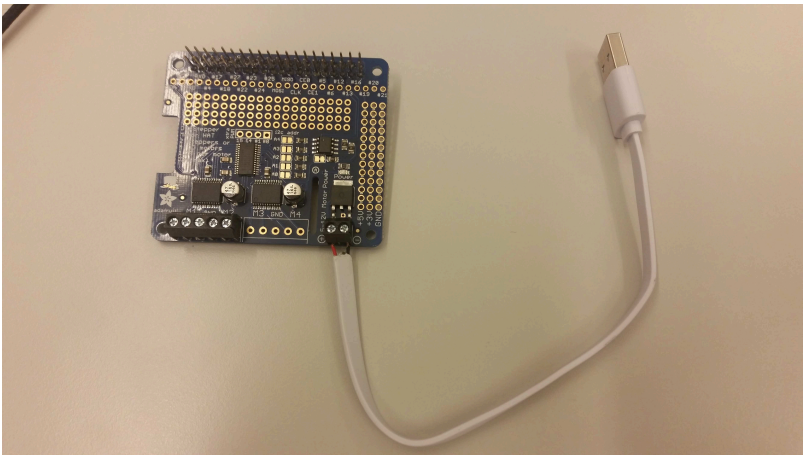


Figure 4.27. Insert the prepared power wire to DC motor HAT power pins.

> **Note:** If you have a `DB17-Montreal-a` or `DB17-Chicago-a` release, neglect this step and follow the pertinent instructions in Unit J-3 - Assembling the Duckiebot `DB17-wjdlc` [draft] regarding the assembly of the PWM hat, its powering through the short angled USB cable, and the power transfer step using a M-M wire.

10) Joypad

With each joypad (Figure 4.28) comes a joypad dongle (Figure 4.29). Don't lose it!

Figure 4.28. All components in the Joypad package

Insert the joypad dongle into one of the USB port of the Raspberry Pi.

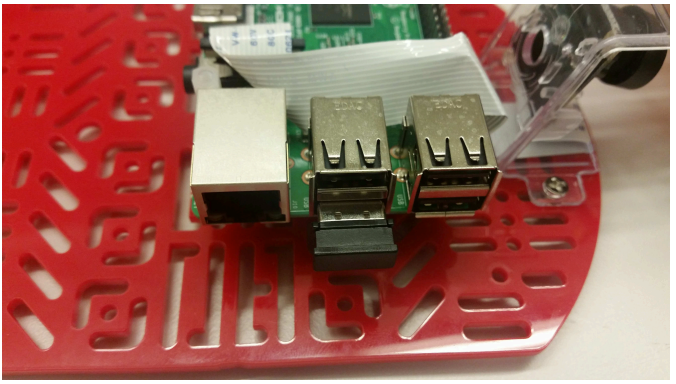


Figure 4.29. The dongle on the Raspberry Pi

Insert 2 AA batteries on the back side of the joypad Figure 4.30.

Figure 4.30. Joypad and 2 AA batteries

4.3. FAQ

Q: If we have the bumpers, at what point should we add them?

Answer: You shouldn't have the bumpers at this point. The function of bumpers is to keep the LEDs in place, i.e., they belong to `DB17-l` configuration. These instructions cover the `DB17-jwd` configurations. You will find the bumper assembly instructions in Unit J-3 - Assembling the Duckiebot `DB17-wjdlc` [draft].

Q: Yeah but I still have the bumpers and am reading this page. So?

Answer: The bumpers can be added after the Duckiebot assembly is complete.

Q: I found it hard to mount the camera (the holes weren't lining up).

Answer: Sometimes in life you have to push a little to make things happen. (But don't push too much or things will break!)

Q: The long camera cable is a bit annoying - I folded it and shoved it in between two hats.

Answer: The shorter cable is even more annoying. We suggest wrapping the long camera cable between the chassis and the Raspberry Pi. With some strategic planning, you can use the zipties that keep the battery in place to hold the camera cable in place as well (see figure below-to add)

TODO: add pretty cable handling pic

Q: I found that the screwdriver that comes with the chassis kit is too fat to screw in the wires on the hat.

Answer: It is possible you got one of the fatter screwdrivers. You will need to figure it out yourself (or ask a TA for help).

Q: I need something to cut the end of the zip tie with.

Answer: Scissors typically work out for these kind of jobs (and no, they're not provided in a Fall 2017 Duckiebox).

UNIT I-5

Assembling the Duckiebot `DB17-wjd` (TTIC)

Point of contact: Andrea F. Daniele

Once you have received the parts and soldered the necessary components, it is time to assemble them in a Duckiebot. Here, we provide the assembly instructions for the configuration `DB17-wjd` (TTIC only).

KNOWLEDGE AND ACTIVITY GRAPH

Requires: Duckiebot `DB17-wjd` parts. The acquisition process is explained in Unit I-2 - Acquiring the parts for the Duckiebot `DB17-wjd`.

Requires: Having soldered the `DB17-wjd` parts. The soldering process is explained in Soldering boards for `DB17` (master).

Requires: Having prepared the power cable. The power cable preparation is explained in Unit I-3 - Preparing the power cable for `DB17`. Note: Not necessary if you intend to build a `DB17-1` configuration.

Requires: Time: about 30 minutes.

Results: An assembled Duckiebot in configuration `DB17-wjd`.

Note: The FAQ section at the bottom of this page may already answer some of you comments, questions or doubts.

This section is comprised of 14 parts. Each part builds upon some of the previous parts, so make sure to follow them in the following order.

- Part I: Motors
- Part II: Wheels
- Part III: Omni-directional wheel
- Part IV: Chassis standoffs
- Part V: Camera kit
- Part VI: Heat sinks
- Part VII: Raspberry Pi 3
- Part VIII: Top plate
- Part IX: USB Power cable
- Part X: DC Stepper Motor HAT
- Part XI: Battery
- Part XII: Upgrade to `DB17-w`

- Part XIII: Upgrade to `DB17-j`
- Part XIV: Upgrade to `DB17-d`

5.1. Motors

Open the Magician Chassis package (Figure 4.1) and take out the following components:

- Chassis-bottom (1x)
- DC Motors (2x)
- Motor holders (4x)
- M3x30 screw (4x)
- M3 nuts (4x)

Figure 5.1 shows the components needed to complete this part of the tutorial.

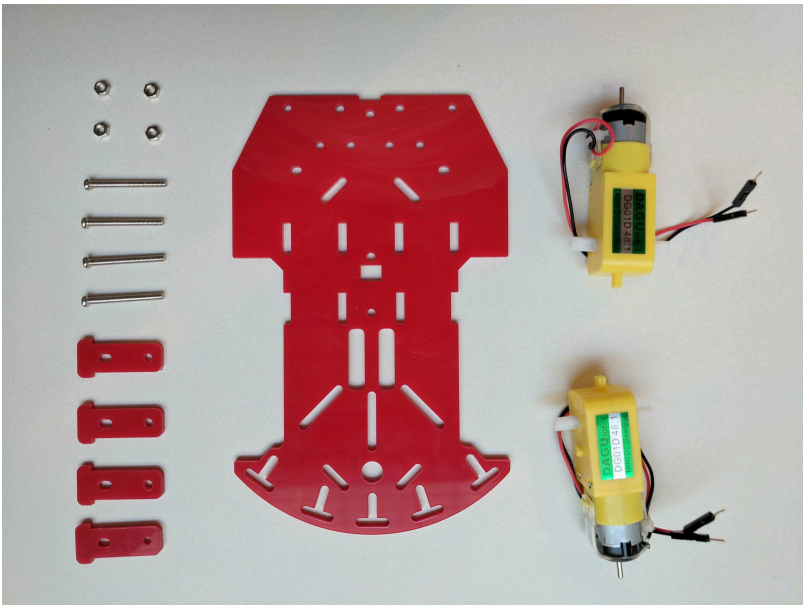


Figure 5.1. Components needed to mount the motors.

1) Video tutorial

The following video shows how to attach the motors to the bottom plate of the chassis.

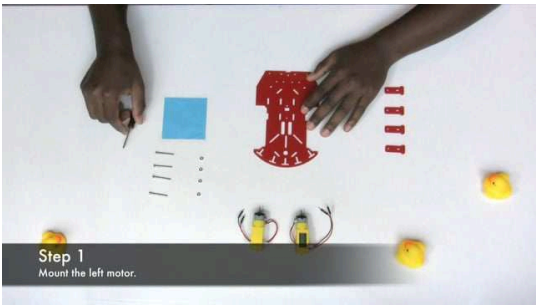


Figure 5.2

2) Step-by-step guide

Step 1:

Pass the motor holders through the openings in the bottom plate of the chassis as shown in Figure 5.3.

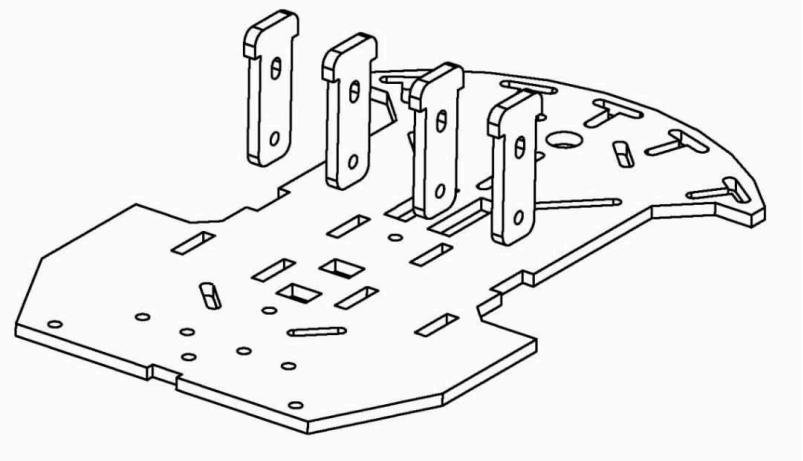


Figure 5.3. The sketch of how to mount the motor holders.

Step 2:

Put the motors between the holders as shown in Figure 5.4.

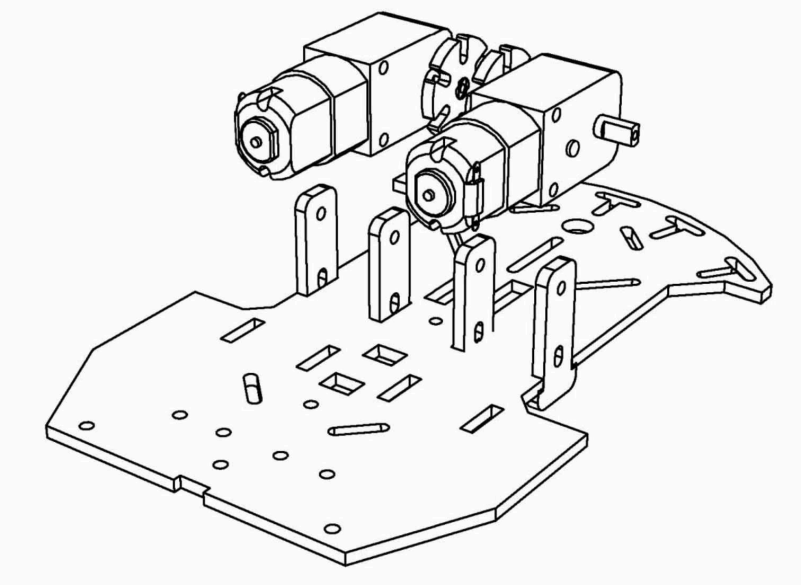


Figure 5.4. The sketch of how to mount the motors.

> Note: Orient the motors so that their wires are inwards (i.e., towards the center of the plate).

Step 3:

Use *4* M3x30 screws and *4* M3 nuts to secure the motors to the motor holders. Tighten the screws to secure the holders to the bottom plate of the chassis as shown in Figure 5.5.

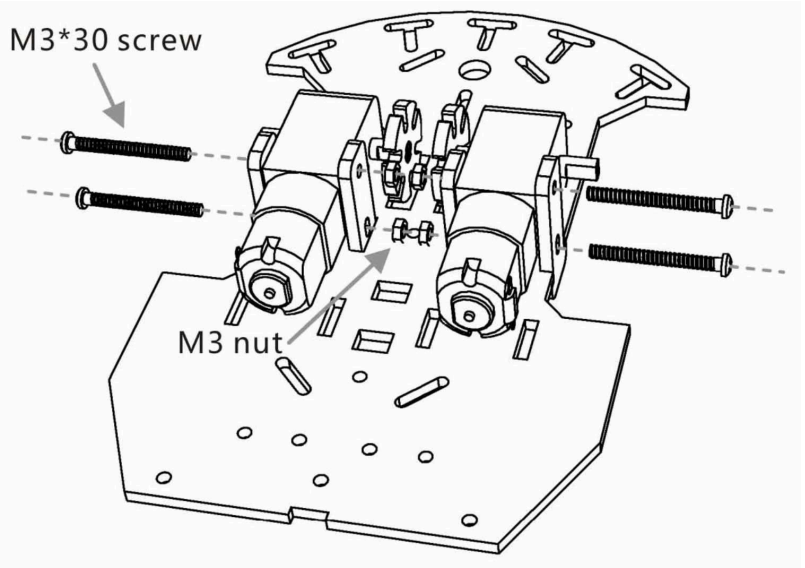


Figure 5.5. The sketch of how to secure the motors to the bottom plate.

3) Check the outcome

Figure 5.6 shows how the motors should be attached to the bottom plate of the chassis.

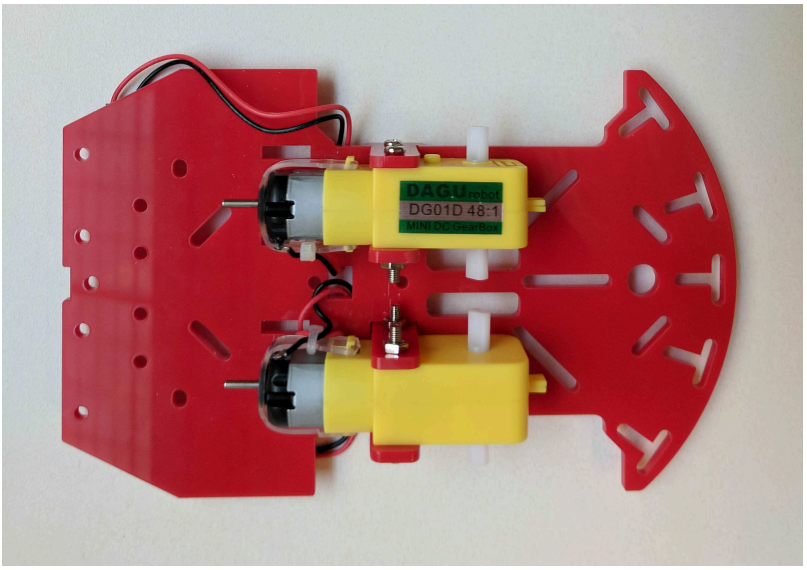


Figure 5.6. The motors are attached to the bottom plate of the chassis.

5.2. Wheels

From the Magician Chassis package take the following components:

- Wheels (2x)

Figure 5.7 shows the components needed to complete this part of the tutorial.

Figure 5.7. The wheels.

1) Video tutorial

The following video shows how to attach the wheels to the motors.

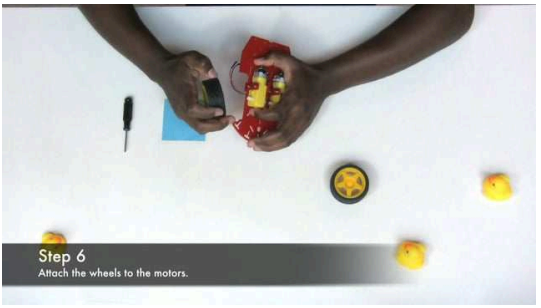


Figure 5.8

2) Check the outcome

Figure 5.9 shows how the wheels should be attached to the motors.

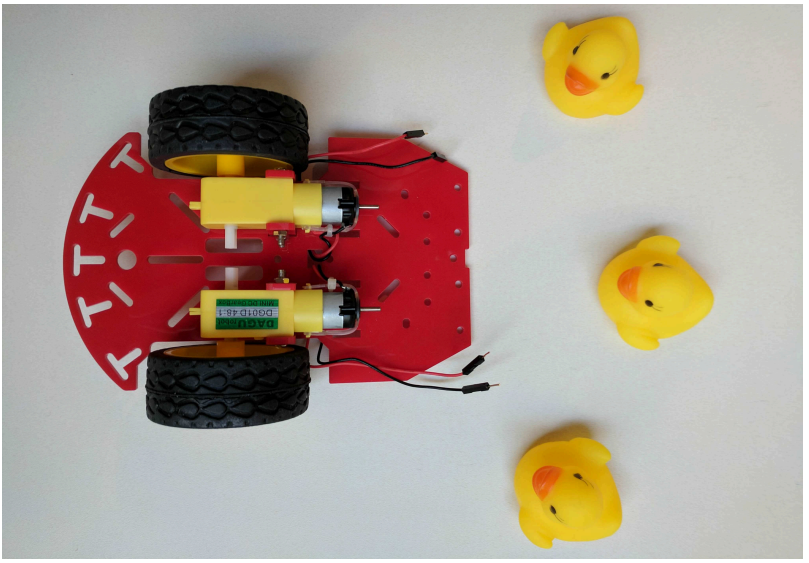


Figure 5.9. The wheels are attached to the motors.

5.3. Omni-directional wheel

The Duckiebot is driven by controlling the wheels attached to the DC motors. Still, it requires a *passive* support on the back. In this configuration an omni-directional wheel is attached to the bottom plate of the chassis to provide such support.

From the Magician Chassis package take the following components:

- Steel omni-directional wheel (1x)
- Long metal spacers (2x)
- M3x6 screws (4x)

Figure 5.10 shows the components needed to complete this part of the tutorial.

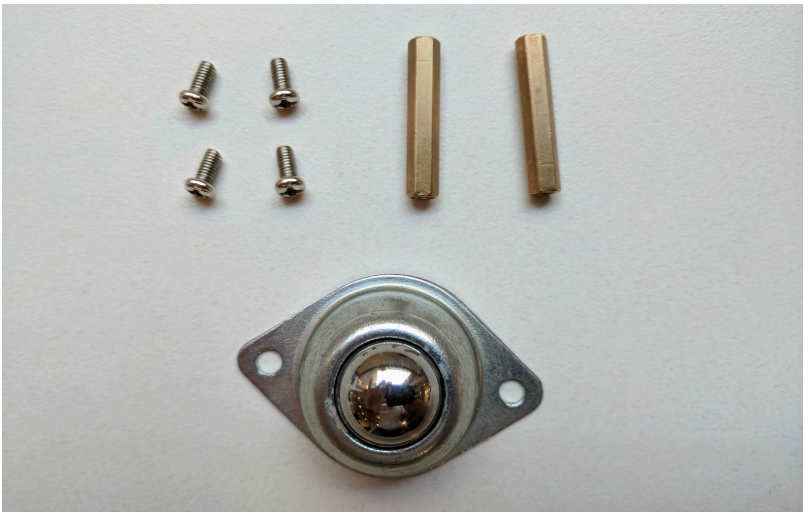


Figure 5.10. The omni-directional wheel with *2* long spacers and *4* M3x6 screws.

1) Video tutorial

The following video shows how to attach the omni-directional wheel to the bottom plate of the chassis.

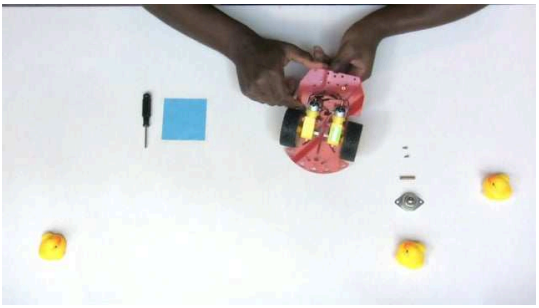


Figure 5.11

2) Step-by-step guide

Step 1:

Secure the long spacers to the plate using *2* M3x6 screws and the omni-directional wheel to the spacers using also *2* M3x6 screws as shown in Figure 5.12.

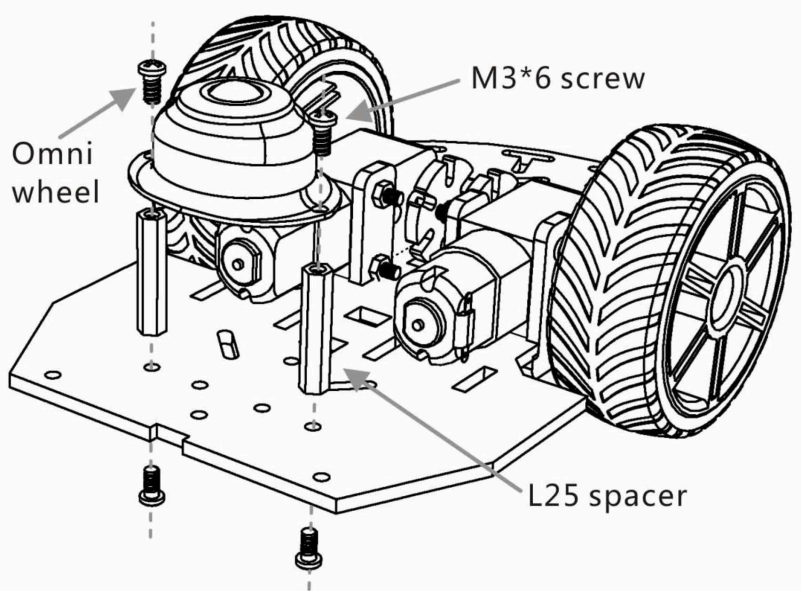


Figure 5.12. The sketch of how to mount the omni-directional wheel.

3) Check the outcome

Figure 5.13 shows how the omni-directional wheel should be attached to the plate.

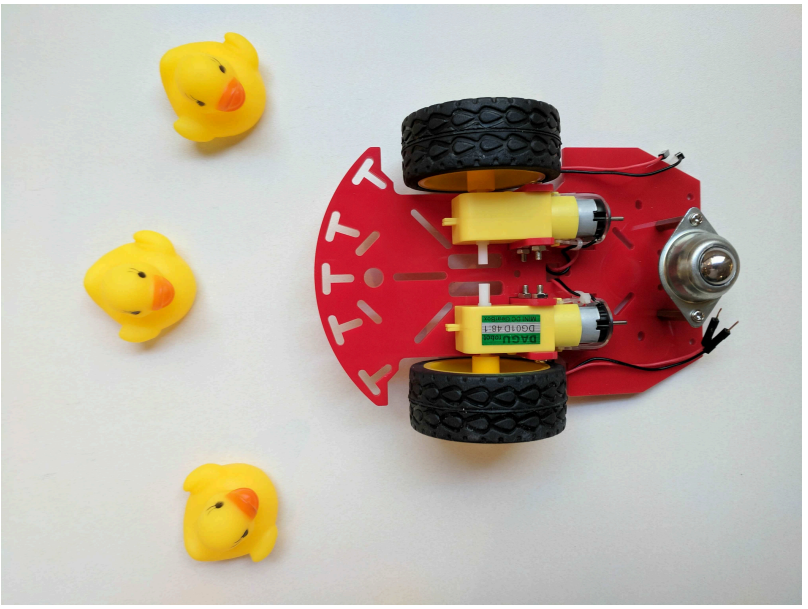


Figure 5.13. The omni-directional wheel is attached to the plate.

5.4. Chassis standoffs

From the Magician Chassis package take the following components:

- Long metal spacers/standoffs (4x)
- M3x6 screws (4x)

From the Duckiebot kit take the following components:

- M3x5 nylon spacers/standoffs (4x)

Figure 5.14 shows the components needed to complete this part of the tutorial.

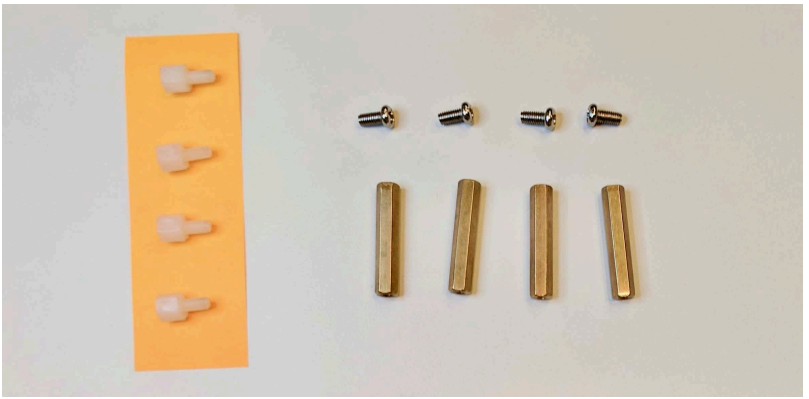


Figure 5.14. The standoffs to mount on the bottom plate.

1) Video tutorial

The following video shows how to attach the standoffs to the bottom plate of the chassis.

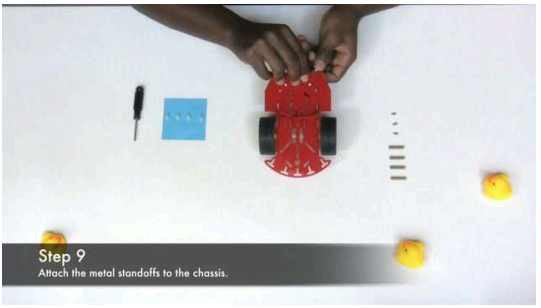


Figure 5.15

2) Step-by-step guide

Step 1:

Secure the long metal spacers to the bottom plate using *4* M3x6 screws as shown in Figure 5.16.

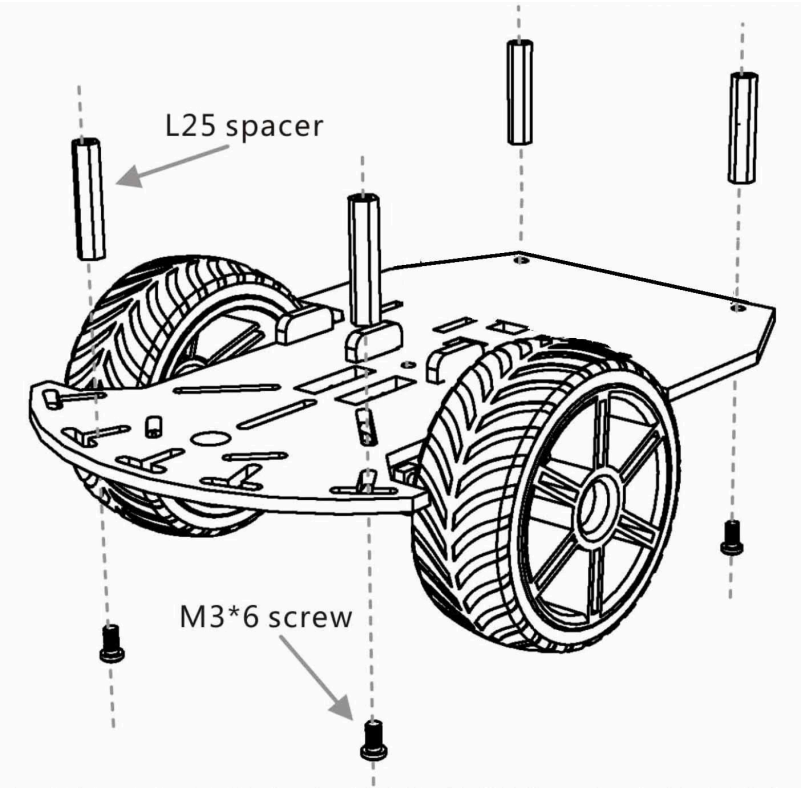


Figure 5.16. The sketch of how to mount the standoffs on the plate.

Step 2:

Attach the *4* nylon standoffs on top of the metal ones.

3) Check the outcome

Figure 5.17 shows how the standoffs should be attached to the plate.

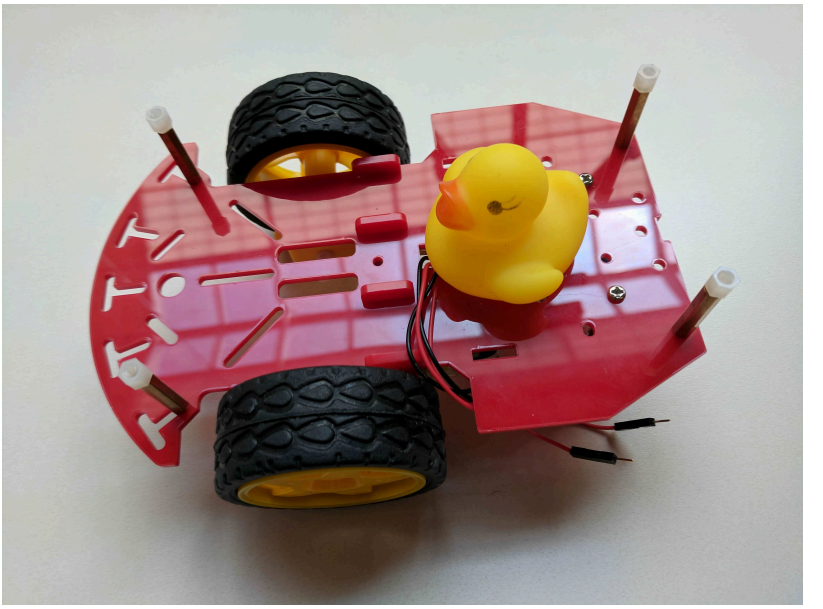


Figure 5.17. The standoffs attached to the plate.

5.5. Camera kit

From the Magician Chassis package take the following components:

- M3x10 flathead screws (2x)
- M3 nuts (2x)

From the Duckiebot kit take the following components:

- Camera Module (1x)
- (Optional) 300mm Camera cable (1x)
- Camera mount (1x)

> Note: If you have camera cables of different lengths available, keep in mind that both are going to work. We suggest to use the longer one, and wrap the extra length under the Raspberry Pi stack.

Figure 5.18 shows the components needed to complete this part of the tutorial.

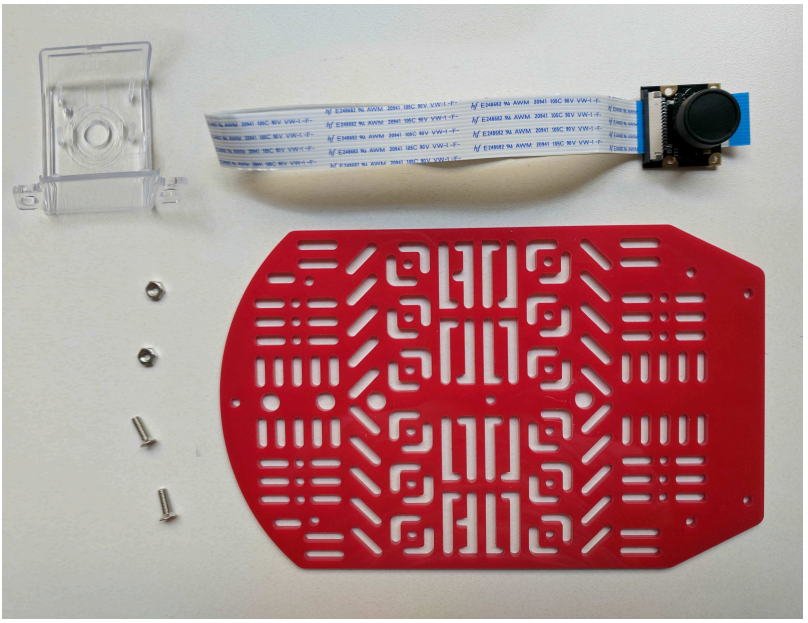


Figure 5.18. The parts needed to fix the camera on the top plate.

1) Video tutorial

The following video shows how to secure the camera to the top plate of the chassis.

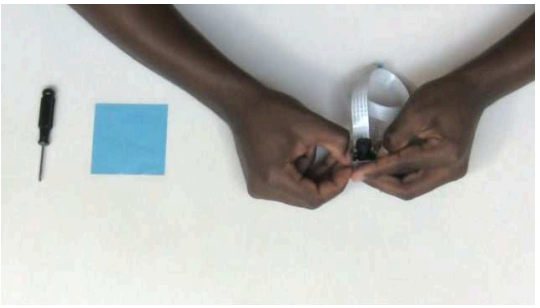


Figure 5.19

2) Step-by-step guide

Step 1 (Optional):

If you do not have the 300mm Camera cable you can jump to *Step 3*.

If you do have the long camera cable, the first thing to do is removing the shorter cable that comes attached to the camera module. Make sure to slide up the black connectors of the camera port on the camera module in order to unblock the cable.

Step 2:

Connect the camera cable to the camera module as shown in Figure 5.20.

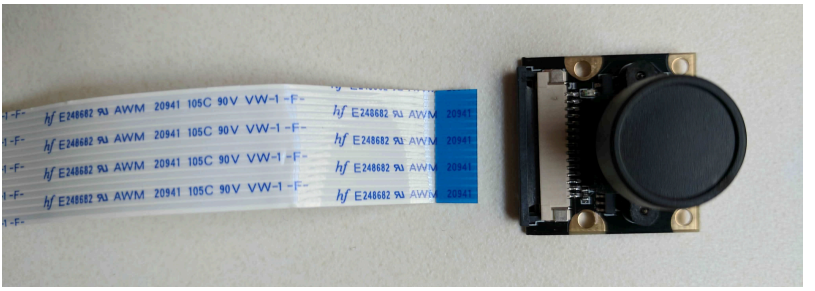


Figure 5.20. How to connect the camera cable to the camera module.

Step 3:

Attach the camera module to the camera mount as shown in Figure 5.21.

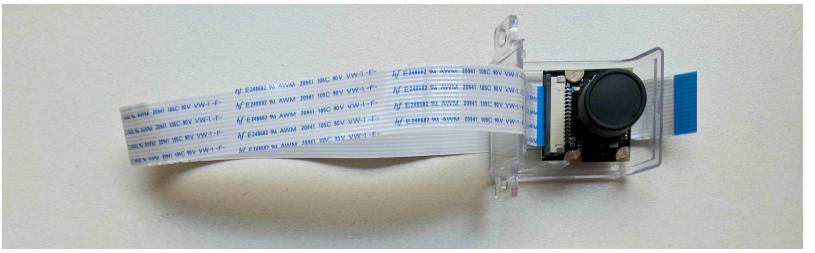


Figure 5.21. How to attach the camera to the camera mount.

> **Note:** The camera is just press-fitted to the camera mount, no screws/nuts are needed.

Step 4:

Secure the camera mount to the top plate by using the *2* M3x10 flathead screws and the nuts as shown in Figure 5.22.

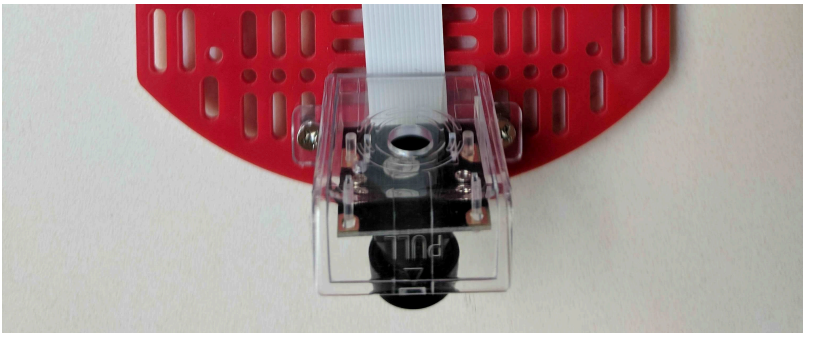


Figure 5.22. How to attach the camera mount to the top plate.

3) Check the outcome

Figure 5.23 shows how the camera should be attached to the plate.

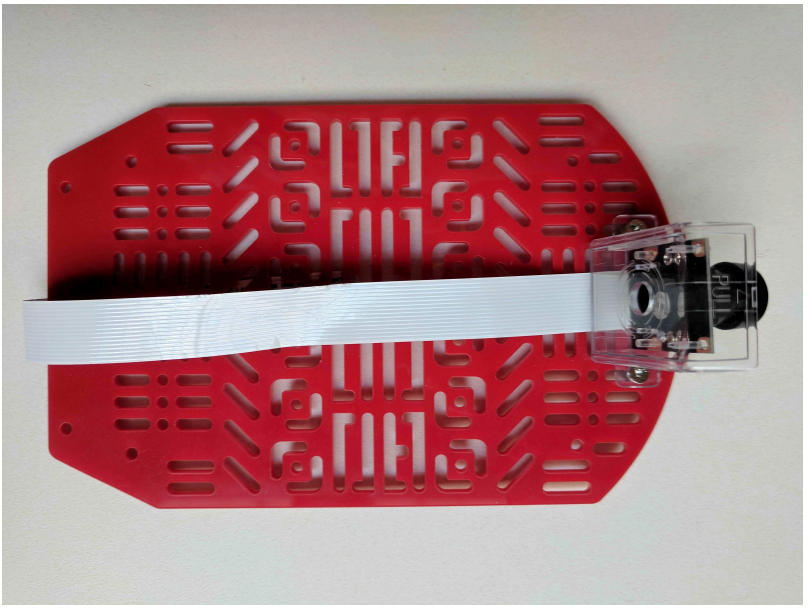


Figure 5.23. The camera attached to the plate.

5.6. Heat sinks

From the Duckiebot kit take the following components:

- Raspberry Pi 3 (1x)
- Heat sinks (2x)
- Camera mount (1x)

Figure 5.24 shows the components needed to complete this part of the tutorial.

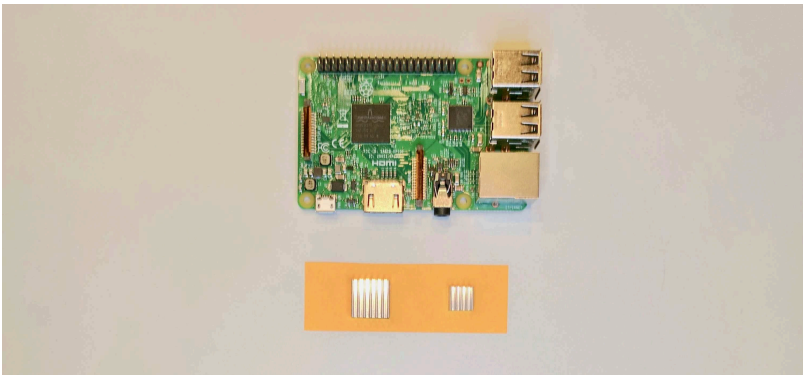


Figure 5.24. The heat sinks and the Raspberry Pi 3.

1) Video tutorial

The following video shows how to install the heat sinks on the Raspberry Pi 3.

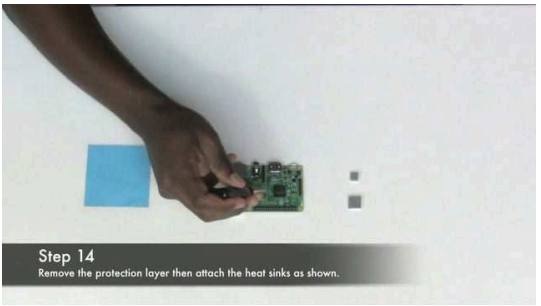


Figure 5.25

2) Step-by-step guide

Step 1:
Remove the protection layer from the heat sinks.

Step 2:
Install the big heat sink on the big “Broadcom”-labeled integrated circuit (IC).

Step 3:
Install the small heat sink on the small “SMSC”-labeled integrated circuit (IC).

3) Check the outcome

Figure 5.26 shows how the heat sinks should be installed on the Raspberry Pi 3.

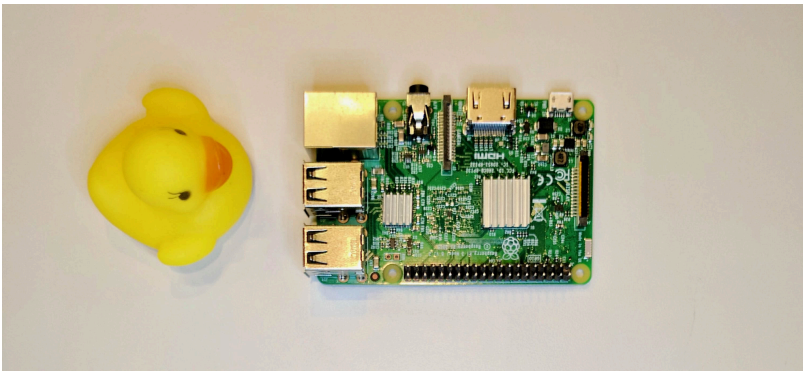


Figure 5.26. The heat sinks installed on the Raspberry Pi 3.

5.7. Raspberry Pi 3

From the Magician Chassis package take the following components:

- Top plate (with camera attached) (1x)

From the Duckiebot kit take the following components:

- Raspberry Pi 3 (with heat sinks) (1x)
- M2.5x12 nylon spacers/standoffs (8x)
- M2.5 nylon hex nuts (4x)

Figure 5.27 shows the components needed to complete this part of the tutorial.

Figure 5.27. The parts needed to mount the Raspberry Pi 3 on the top plate.

1) Video tutorial

The following video shows how to mount the Raspberry Pi 3 on the top plate of the chassis.

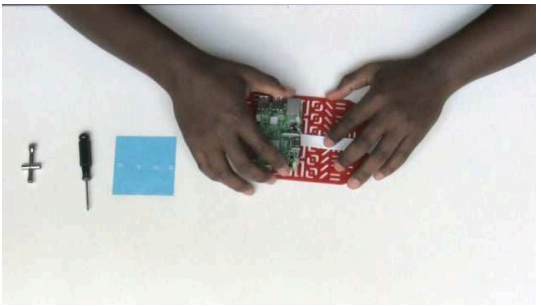


Figure 5.28

2) Step-by-step guide

Step 1:

Mount *8* M2.5x12 nylon standoffs on the Raspberry Pi 3 as shown in Figure 5.29.

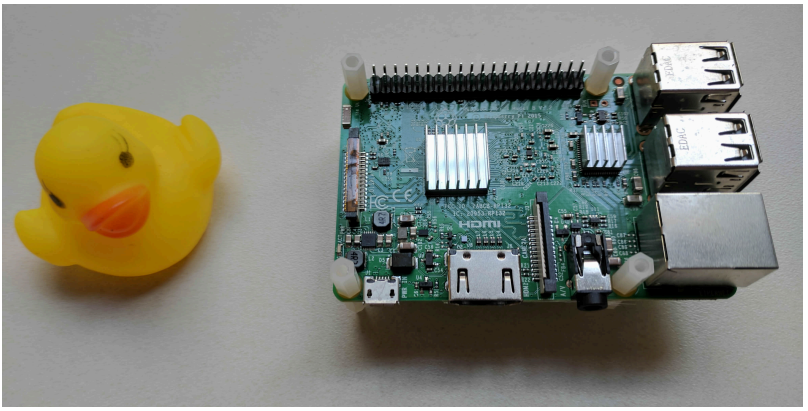


Figure 5.29. How to mount the nylon standoffs on the Raspberry Pi 3.

Step 2:

Use the M2.5 nylon hex nuts to secure the Raspberry Pi 3 to the top plate as shown in Figure 5.30.

Figure 5.30. How to mount the Raspberry Pi 3 on the top plate.

3) Check the outcome

Figure 5.31 shows how the Raspberry Pi 3 should be mounted on the top plate of the chassis.

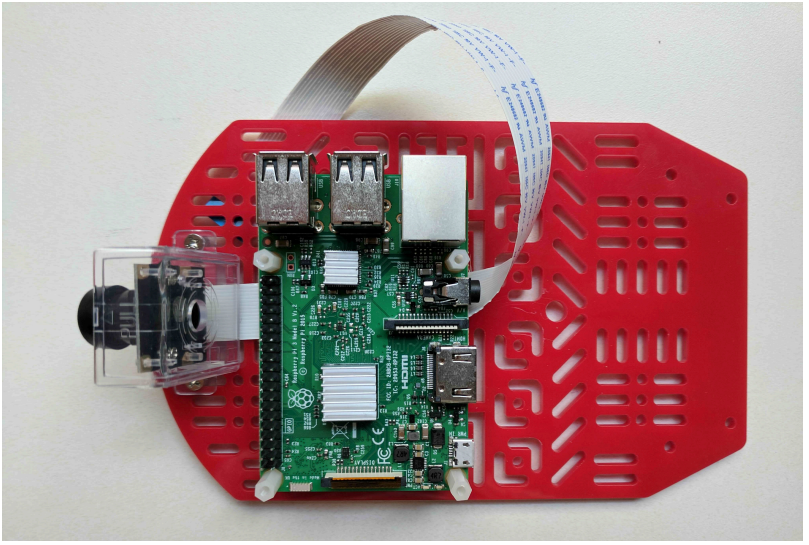


Figure 5.31. The Raspberry Pi 3 mounted on the top plate.

5.8. Top plate

From the Magician Chassis package take the following components:

- Top plate (with camera and Raspberry Pi 3 attached) (1x)
- M3x6 screws (4x)

Figure 5.32 shows the components needed to complete this part of the tutorial.

Figure 5.32. The parts needed to secure the top plate to the bottom plate.

1) Video tutorial

The following video shows how to secure the top plate on top of the bottom plate.

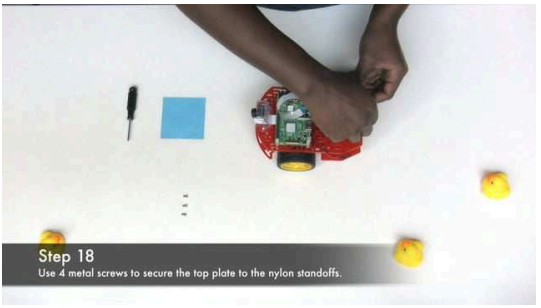


Figure 5.33

2) Step-by-step guide

Step 1:

Pass the motor wires through the openings in the top plate.

Step 2:

Use *4* M3x6 screws to secure the top plate to the nylon standoffs (mounted on the bottom plate in Section 5.4 - Chassis standoffs) as shown in Figure 5.34.

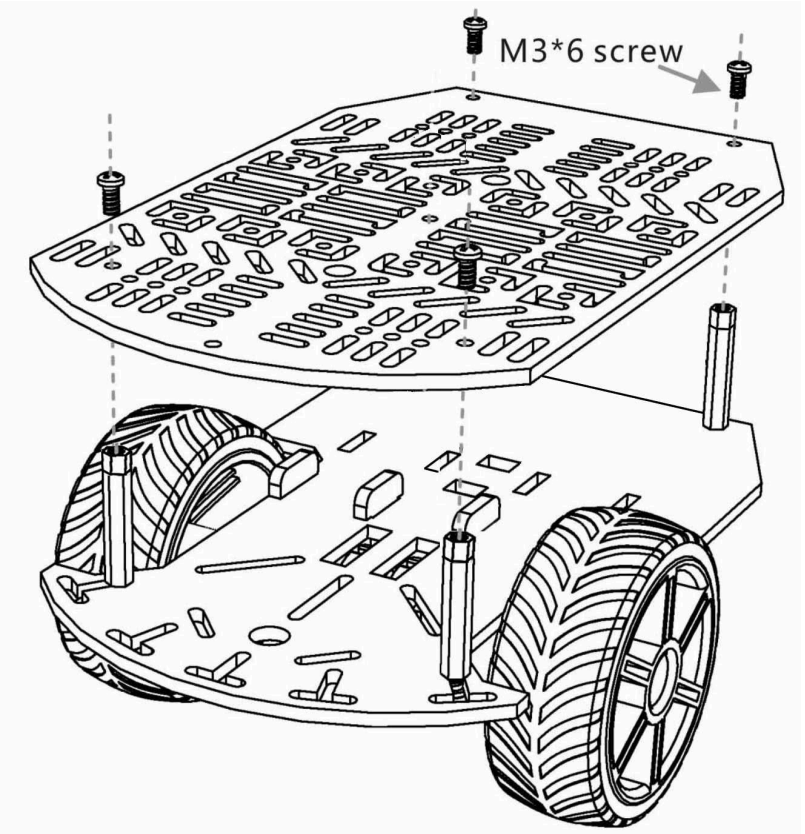


Figure 5.34. How to secure the top plate to the bottom plate.

3) Check the outcome

Figure 5.35 shows how the top plate should be mounted on the bottom plate.

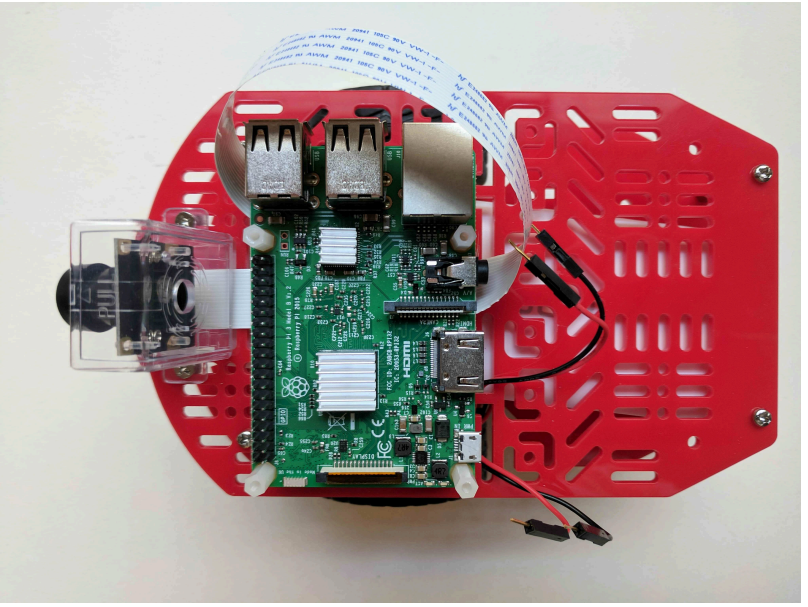


Figure 5.35. The chassis completed.

5.9. USB Power cable

The power cable preparation is explained in Unit I-3 - Preparing the power cable for DB17.

5.10. DC Stepper Motor HAT

From the Duckiebot kit take the following components:

- USB power cable (prepared in Unit I-3 - Preparing the power cable for DB17) (1x)
- DC Stepper Motor HAT (1x)
- M2.5x10 Nylon screws (or M2.5x12 nylon standoffs) (4x)

Figure 5.36 shows the components needed to complete this part of the tutorial.

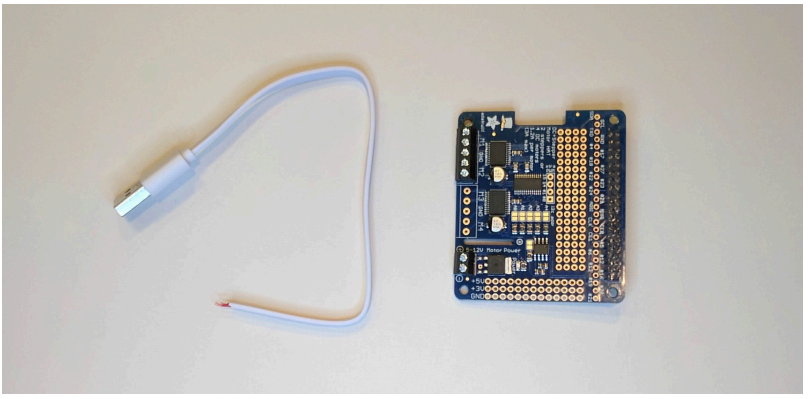


Figure 5.36. The parts needed to add the DC Stepper Motor HAT to the Duckiebot.

1) Video tutorial

The following video shows how to connect the DC Stepper Motor HAT to the Raspberry Pi 3.

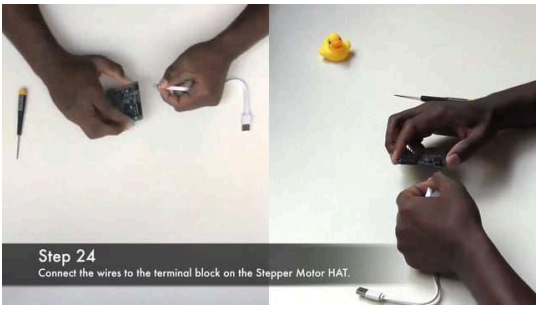


Figure 5.37

2) Step-by-step guide

Step 1:

Connect the wires of the USB power cable to the terminal block on the DC Stepper Motor HAT labeled as “5-12V Motor Power” as shown in Figure 5.38. The black wire goes to the negative terminal block (labeled with a minus: `-`) and the red wire goes to the positive terminal block (labeled with a plus: `+`).

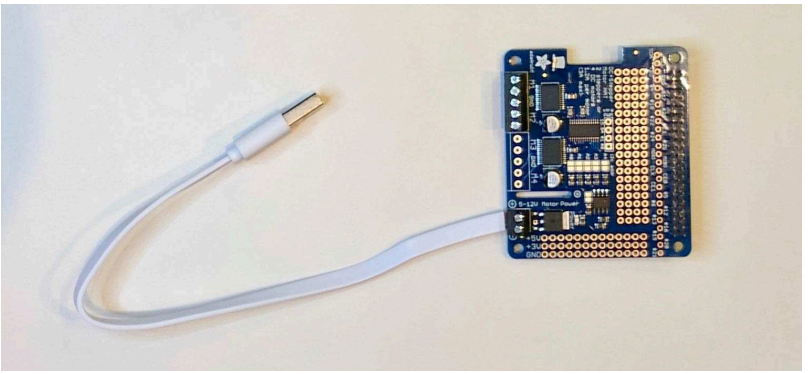


Figure 5.38. How to connect the USB power cable to the DC Stepper Motor HAT.

Step 2:

Pass the free end of the camera cable through the opening in the DC Stepper Motor HAT as shown in Figure 5.39.

Figure 5.39. How to pass the camera cable through the opening in the DC Stepper Motor HAT.

Step 3:

Connect the free end of the camera cable to the **CAMERA** port on the Raspberry Pi 3 as shown in Figure 5.40.

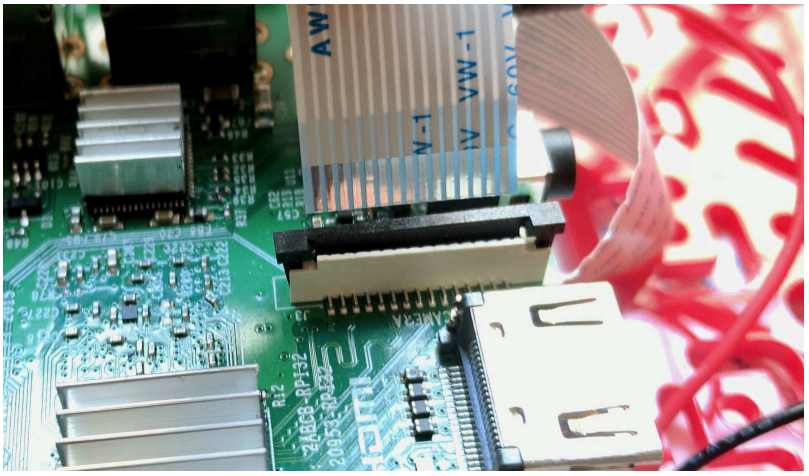


Figure 5.40. How to connect the camera cable to the CAMERA port on the Raspberry Pi 3.

To do so, you will need to gently pull up on the black connector (it will slide up) to allow the cable to insert the port. Slide the connector back down to lock the cable in place, making sure it “clicks”.

> **Note:** Make sure the camera cable is inserted in the right direction! The metal pins of the cable must be in contact with the metal terminals in the camera port of the PI.

Step 4:

Attach the DC Stepper Motor HAT to the GPIO header on the Raspberry Pi 3. Make sure that the GPIO stacking header of the Motor HAT is carefully aligned with the underlying GPIO pins before applying pressure.

> **Note:** In case you are using a short camera cable, ensure that the camera cable does not stand between the GPIO pins and the the GPIO header socket before applying pressure.

Step 5:

Secure the DC Stepper Motor HAT using *4* M2.5x10 nylon screws.

> **Note:** If you are planning on upgrading your Duckiebot to the configuration `DB17-1`, you can use *4* M2.5x12 nylon standoffs instead.

Step 6:

Connect the motor wires to the terminal block on the DC Stepper Motor HAT as shown in Figure 5.41.

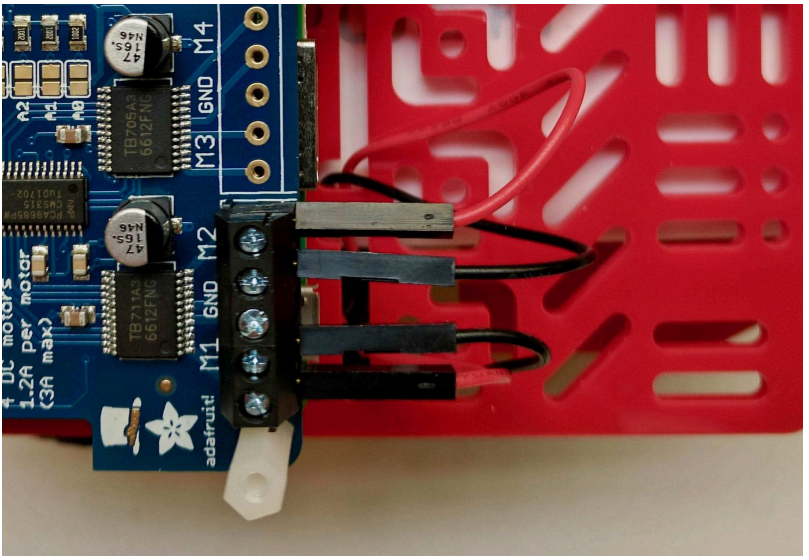


Figure 5.41. How to connect the motor wires to the terminal block on the DC Stepper Motor HAT.

While looking at the Duckiebot from the back, identify the wires for left and right motor. Connect the left motor wires to the terminals labeled as **M1** and the right motor wires to the terminals labeled as **M2**. This will ensure that the pre-existing software that we will later install on the Duckiebot will send the commands to the correct motors.

3) Check the outcome

Figure 5.42 shows how the DC Stepper Motor HAT should be connected to the Raspberry Pi 3.

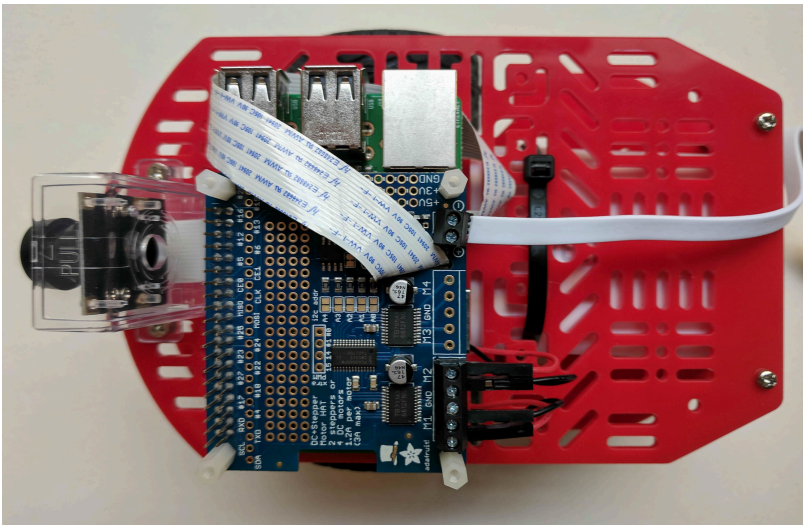


Figure 5.42. The DC Stepper Motor HAT connected to the Raspberry Pi 3.

5.11. Battery

From the Duckiebot kit take the following components:

- Battery (1x)
- Zip tie (1x)
- Short micro USB cable (1x)

Figure 5.43 shows the components needed to complete this part of the tutorial.

Figure 5.43. The parts needed to add the battery to the Duckiebot.

1) Video tutorial

The following video shows how to add the battery to the Duckiebot and turn it on.

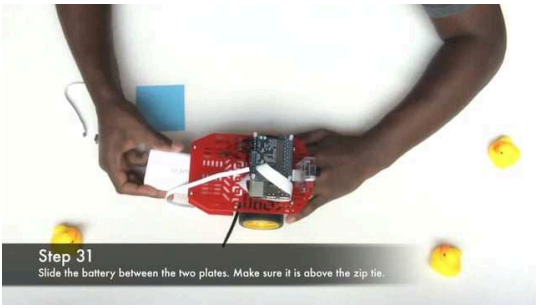


Figure 5.44

2) Step-by-step guide

Step 1:
Pass the zip tie through the opening in the top plate.

Step 2:
Slide the battery between the two plates. Make sure it is above the zip tie.

Step 3:
Push the free end of the zip tie through the opening in the top plate.

Step 4:
Tighten the zip tie to secure the battery.

Step 5:
Connect the short micro USB cable to the Raspberry Pi 3.

Step 6:
Connect the short micro USB cable to the battery.

Step 7:
Connect the USB power cable to the battery.

Step 8:
Make sure that the LEDs on the Raspberry Pi 3 and the DC Stepper Motor HAT are on.

3) Check the outcome

Figure 5.45 shows how the battery should be installed on the Duckiebot.

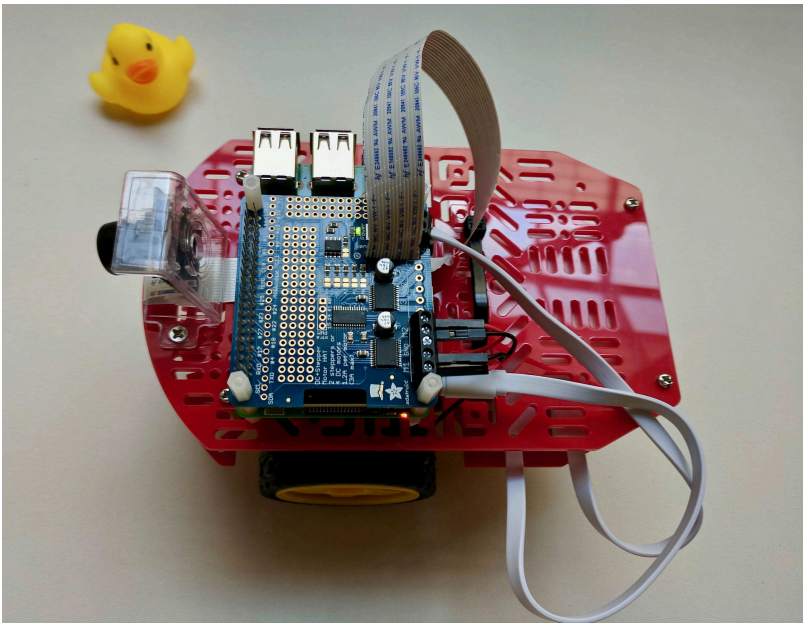


Figure 5.45. The configuration `DB17` completed.

5.12. Upgrade to `DB17-w`

This upgrade equips the Duckiebot with a secondary, faster, Wi-Fi connection, ideal for image streaming. The new configuration is called `DB17-w`.

Figure 5.46 shows the components needed to complete this upgrade.

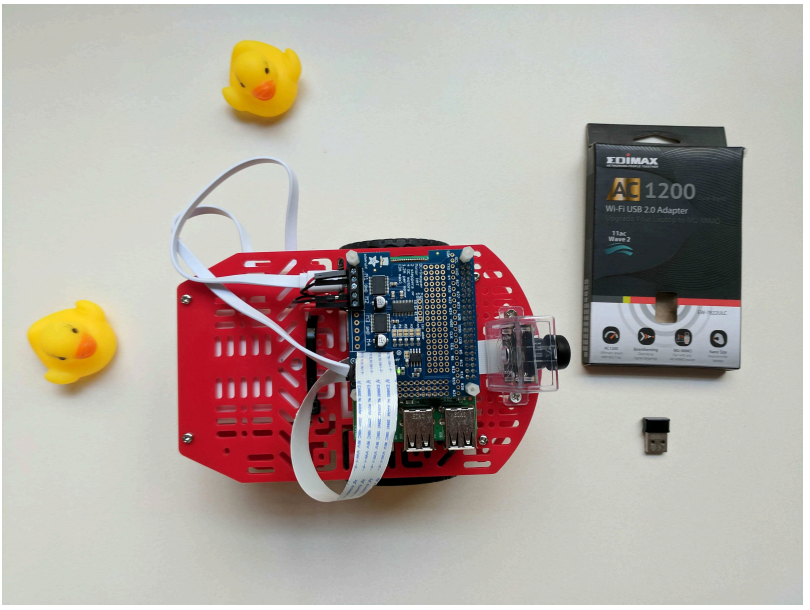


Figure 5.46. The parts needed to upgrade the Duckiebot to the configuration `DB17-w`.

1) Instructions

- Insert the USB WiFi dongle into one of the USB port of the Raspberry Pi.

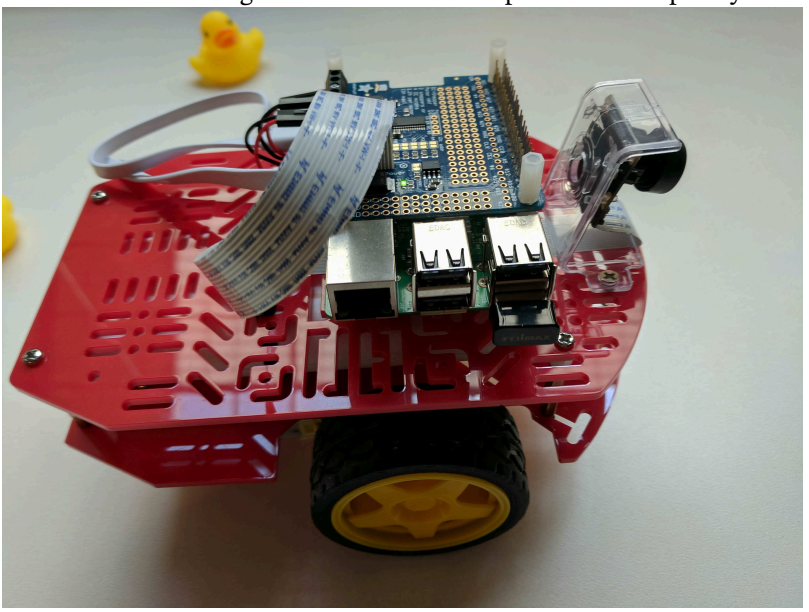


Figure 5.47. Upgrade to `DB17-w` completed.

5.13. Upgrade to `DB17-j`

This upgrade equips the Duckiebot with manual remote control capabilities. It is particularly useful for getting the Duckiebot out of tight spots or letting younger ones have a drive, in addition to providing handy shortcuts to different functions in development phase. The new configuration is called `DB17-j`.

Figure 5.48 shows the components needed to complete this upgrade.

Figure 5.48. The parts needed to upgrade the Duckiebot to the configuration `DB17-j`.

Note: The joypad comes with a USB receiver (as shown in Figure 5.48).

1) Instructions

- Insert the USB receiver into one of the USB port of the Raspberry Pi.
- Insert 2 AA batteries on the back side of the joypad.

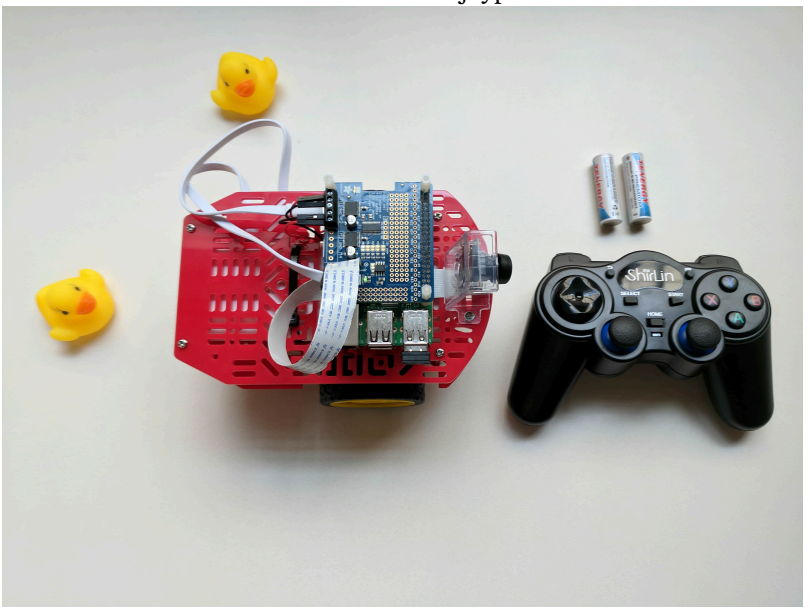


Figure 5.49. Upgrade to `DB17-j` completed.

5.14. Upgrade to `DB17-d`

This upgrade equips the Duckiebot with an external hard drive that 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 3. Moreover, it is easy to unplug it from the Duckiebot at the end of the day and bring it over to a computer for downloading and analyzing stored data. The new configuration is called `DB17-d`.

Figure 5.50 shows the components needed to complete this upgrade.

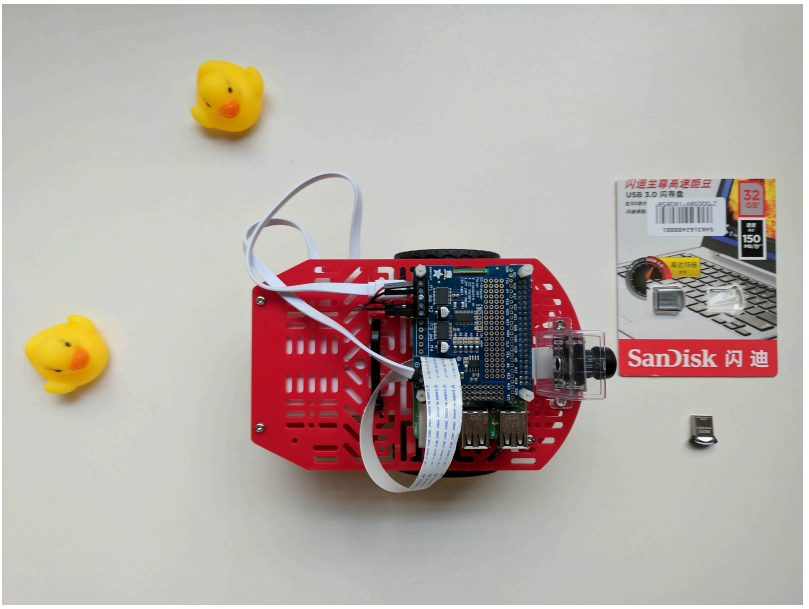


Figure 5.50. The parts needed to upgrade the Duckiebot to the configuration `DB17-d`.

1) Instructions

- Insert the USB drive into one of the USB port of the Raspberry Pi.

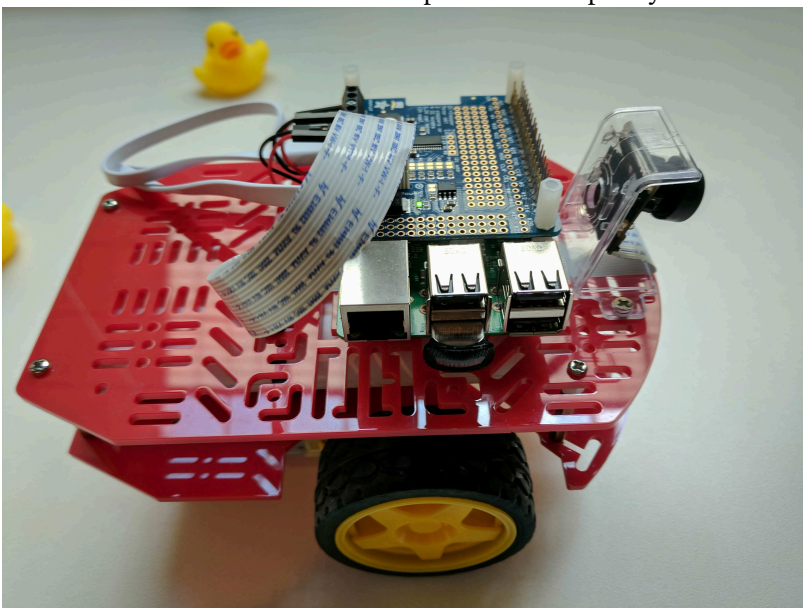


Figure 5.51. Upgrade to `DB17-d` completed.

5.15. FAQ

Q: If we have the bumpers, at what point should we add them?

Answer: You shouldn't have the bumpers at this point. The function of the bumpers is to keep the LEDs in place, i.e., they belong to `DB17-l` configuration. These instructions cover the `DB17-wjd` configurations. You will find the bumper assembly instructions in Unit J-3 - Assembling the Duckiebot `DB17-wjdlc` [draft].

Q: Yeah but I still have the bumpers and am reading this page. So?

Answer: The bumpers can be added after the Duckiebot assembly is complete.

Q: I found it hard to mount the camera (the holes weren't lining up).

Answer: Sometimes in life you have to push a little to make things happen. (But don't push too much or things will break!)

Q: The long camera cable is a bit annoying - I folded it and shoved it in between two hats.

Answer: The shorter cable is even more annoying. We suggest wrapping the long camera cable between the chassis and the Raspberry Pi. With some strategic planning, you can use the zipties that keep the battery in place to hold the camera cable in place as well (see figure below-to add)

TODO: add pretty cable handling pic

Q: I found that the screwdriver that comes with the chassis kit is too fat to screw in the wires on the hat.

Answer: It is possible you got one of the fatter screwdrivers. You will need to figure it out yourself (or ask a TA for help).

Q: I need something to cut the end of the zip tie with.

Answer: Scissors typically work out for these kind of jobs (and no, they're not provided in a Fall 2017 Duckiebox).

UNIT I-6

Installing Ubuntu on laptops

Assigned to: Andrea

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

KNOWLEDGE AND ACTIVITY GRAPH

Requires: A laptop with free disk space.

Requires: Internet connection to download the Ubuntu image.

Requires: About 30 minutes.

Results: A laptop ready to be used for Duckietown.

6.1. Install Ubuntu

Install Ubuntu 16.04.3.

➡ 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.

6.2. 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
```

6.3. Install ROS

Install ROS on your laptop.

➡ The procedure is given in Install ROS (master).

6.4. Other suggested software

1) Redshift

This is Flux for Linux. It is an accessibility/lab safety issue: bright screens damage eyes and perturb sleep [6].

Install redshift and run it.

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

Set to “autostart” from the icon.

6.5. 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 1.3 - Installing the documentation system.

6.6. Passwordless sudo

Set up passwordless `sudo`.

> ➡ This procedure is described in Passwordless sudo (master).

+ comment

Huh I don’t know - this is great for usability, but horrible for security. If you step away from your laptop for a second and don’t lock the screen, a nasty person could `sudo rm -rf /`. -FG

6.7. SSH and Git setup

1) Basic SSH config

Do the basic SSH config.

> ➡ The procedure is documented in Local configuration (master).

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 Creating an SSH keypair (master).

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

Add the public key to your Github account.

> ➡ The procedure is documented in Add a public key to Github (master).

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 Setting up global configurations for Git (master).

UNIT I-7

Duckiebot Initialization

Assigned to: Andrea

KNOWLEDGE AND ACTIVITY GRAPH

Requires: An SD card of dimensions at least 16 GB.

Requires: A computer with an internet connection, an SD card reader, and 16 GB of free space.

Requires: An assembled Duckiebot in configuration `DB17`. This is the result of Unit I-4 - Assembling the Duckiebot `DB17`, `DB17-jwd`.

Results: A Duckiebot that is configured correctly, that you can connect to with your laptop and hopefully also has internet access

7.1. Acquire and burn the image

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

https://www.dropbox.com/s/ckpqpp0cav3aucb/duckiebot-RPI3-AD-2017-09-12.img.xz?dl=1

The size is 1.7 GB.

You can use:

```
$ wget -O duckiebot-RPI3-AD-2017-09-12.img.xz URL above
```

+ comment

The original was:

```
$ curl -o duckiebot-RPI3-AD-2017-09-12.img.xz URL above
```

It looks like that `curl` cannot be used with Drobpox links because it does not follow redirects.

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

```
$ sha256sum duckiebot-RPI3-AD-2017-09-12.img.xz
7136f9049b230de68e8b2d6df29ece844a3f830cc96014aaa92c6d3f247b6130
duckiebot-RPI3-AD-2017-09-12.img.xz
```

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

Uncompress the file:

```
$ xz -d -k duckiebot-RPI3-AD-2017-09-12.img.xz
```

This will create a file of 11 GB in size.

Next, burn the image on disk.

> ➡ The procedure of how to burn an image is explained in How to burn an image to an SD card (master).

7.2. 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

+ comment

In general, for the battery: if it's off, a single click on the power button will turn the battery on. When it's on, a single click will show you the charge indicator (4 white lights = full), and holding the button for 3s will turn off the battery. Shutting down the Duckiebot is not recommended because it may cause corruption of the SD card.

7.3. 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.

7.4. 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
...
```

7.5. 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 Byobu (master) for an introduction to Byobu.

+ doubt

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

7.6. Setup network

➡ Unit I-8 - Networking aka the hardest part

7.7. 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
```

7.8. 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.

7.9. 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
```

7.10. 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        15G  6.3G  8.2G  44% /
devtmpfs        303M     0  303M   0% /dev
tmpfs           431M     0  431M   0% /dev/shm
tmpfs           431M   12M  420M   3% /run
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           431M     0  431M   0% /sys/fs/cgroup
/dev/mmcblk0p1   63M   21M   43M  34% /boot
tmpfs            87M     0   87M   0% /run/user/1000
```

You should see that the Size of your `/dev/root` Filesystem is “close” to the size of your SD card.

7.11. 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 groups `input`, `video`, and `i2c`

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

Set the shell `bash`:

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

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.

```
+ comment
What steps?? -LP
```

1) Basic SSH config

Do the basic SSH config.

➡ The procedure is documented in Local configuration (master).

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 Creating an SSH keypair (master).

3) Add SSH alias

Once you have your SSH key pair on both your laptop and your Duckiebot, as well as your new user- and hostname set up on your Duckiebot, then you should set up an SSH alias as described in SSH aliases (master). This allows your to log in for example with

```
$ ssh abc
```

instead of

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

where you can chose `abc` to be any alias / shortcut.

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

Add the public key to your Github account.

➡ The procedure is documented in Add a public key to Github (master).

If the step is done correctly, the following command should succeed and give you a welcome message:

```
$ ssh -T git@github.com
Hi username! You've successfully authenticated, but GitHub does not provide shell
access.
```

5) Local Git configuration

➡ This procedure is in Setting up global configurations for Git (master).

6) Set up the laptop-Duckiebot connection

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

➡ The procedure is explained in How to login without a password (master). 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
```

7) 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...

7.12. 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`.

7.13. 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 Download a file with SCP (master).

7.14. Final touches: duckie logo

In order to show that your Duckiebot is ready for the task of driving around happy little duckies, the robot has to fly the Duckietown flag. When you are still logged in to the Duckiebot you can download and install the banner like this:
Download the ANSI art file from Github:

```
$ wget --no-check-certificate -O duckie.art "https://raw.githubusercontent.com/
duckietown/Software/master/misc/duckie.art"
```

(optional) If you want, you can preview the logo by just outputting it onto the command line:

```
$ cat duckie.art
```

Next up create a new empty text file in your favorite editor and add the code for showing your duckie pride:
Let's say I use `nano`, I open a new file:

```
$ nano 20-duckie
```

And in there I add the following code (which by itself just prints the duckie logo):

```
#!/bin/sh
printf "\n$(cat /etc/update-motd.d/duckie.art)\n"
```

Then save and close the file. Finally you have to make this file executable...

```
$ chmod +x 20-duckie
```

...and copy both the duckie logo and the script into a specific directory `/etc/update-motd.d` to make it appear when you login via SSH. `motd` stands for "message of the day". This is a mechanism for system administrators to show users news and messages when they login. Every executable script in this directory which has a filename a la `NN-some name` will get exected when a user logs in, where `NN` is a two digit number that indicates the order.

```
sudo cp duckie.art /etc/update-motd.d
sudo cp 20-duckie /etc/update-motd.d
```

Finally log out of SSH via `exit` and log back in to see duckie goodness.

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.".

Symptom: random `wget`, `curl`, `git`, and `apt` calls fail with SSL errors.

Resolution: That's probably actually an issue with your system time. Type the command `timedatectl` into a terminal, hit enter and see if the time is off. If it is, you might want to follow the intructions from this article, or entirely uninstall your NTP service and manually grab the time on reboot. It's a bit dirty, but works surprisingly well.

Symptom: Cannot find `/etc` folder for configuring the Wi-Fi. I only see `Desktop`, `Downloads` when starting up the Duckiebot.

Resolution: If a directory name starts with `/`, it's not supposed to be in the home directory, but rather at the root of the filesystem. You are currently in `/home/ubuntu`. Type `ls /` to see the folders at the root, including `/etc.

Unit I-8

Networking aka the hardest part

Knowledge and activity graph

Requires: A Duckiebot in configuration `DB17-CO+W`

Requires: Either a router that you have control over that has internet access, or

your credentials for connecting to an existing wireless network

Requires: Patience (channel your inner Yoda)

Results: A Duckiebot that you can connect to and that is connected to the internet

Note: this page is primarily for folks operating with the "two-network" configuration, `C0+w`. For a one adapter setup you will can skip directly to Section 8.2 - Setting up wireless network configuration, but you will have to connect to a network that you can ssh through.

The basic idea is that we are going to use the "Edimax" thumbdrive adapter to create a dedicated wireless network that you can always connect to with your laptop. Then we are going to use the built-in Broadcom chip on the Pi to connect to the internet, and then the network will be bridged.

8.1. (For `C0+w`) Configure the robot-generated network

This part should work every time with very low uncertainty.

The Duckiebot in configuration `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
wlxAABBCCDDEEFFGG 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.

8.2. Setting up wireless network configuration

You are connected to the Duckiebot via WiFi, but the Duckiebot also needs to connect to the internet in order to get updates and install some software. This part is a little bit more of a "black art" since we cannot predict every possible network configurations. Below are some settings that have been verified to work in different situations:

1) Option 1: `duckietown` WiFi

Check with your phone or laptop if there is a WiFi in reach with the name of `duckietown`. If there is, you are all set. The defaut configuration for the Duckiebot is to have one WiFi adapter connect to this network and the other broadcast the access point which you are currently connected to.

2) Option 2.a): `eduroam` WiFi (Non-UdeM/McGill instructions)

If there should be no `duckietown` network in reach then you have to manually add a network configuration file for the network that you'd like to connect to. Most universities around the world should have to `eduroam` network available. You can use it for connecting your Duckiebot.

Save the following block as new file in `/etc/NetworkManager/system-connections/eduroam`:

```
[connection]
id=eduroam
uuid=38ea363b-2db3-4849-a9a4-c2aa3236ae29
type=wifi
permissions=user:oem:;
secondaries=

[wifi]
mac-address=the MAC address of your internal wifi adapter, wlan0
mac-address-blacklist=
mac-address-randomization=0
mode=infrastructure
seen-bssids=
ssid=eduroam

[wifi-security]
auth-alg=open
group=
key-mgmt=wpa-eap
pairwise=
proto=

[802-1x]
altsubject-matches=
eap=ttls;
identity=your eduroam username@your eduroam domain
password=your eduroam password
phase2-altsubject-matches=
phase2-auth=pap

[ipv4]
dns-search=
method=auto

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

Set the permissions on the new file to 0600.

```
sudo chmod 0600 /etc/NetworkManager/system-connections/eduroam
```

3) Option 2.b): `eduroam` WiFi (UdeM/McGill instructions)

Save the following block as new file in `/etc/NetworkManager/system-connections/eduroam-USER-NAME`: where USERNAME is the your logged-in username in the duckiebot.

```
[connection]
id=eduroam
uuid=38ea363b-2db3-4849-a9a4-c2aa3236ae29
type=wifi
permissions=user:USERNAME:;
secondaries=

[wifi]
mac-address=the MAC address of your internal wifi adapter, wlan0
mac-address-blacklist=
mac-address-randomization=0
mode=infrastructure
seen-bssids=
ssid=eduroam

[wifi-security]
auth-alg=open
group=
key-mgmt=wpa-eap
pairwise=
proto=

[802-1x]
altsubject-matches=
eap=peap;
identity=DGTIC UNIP
password=DGTIC PWD
phase2-altsubject-matches=
phase2-auth=mschapv2

[ipv4]
dns-search=
method=auto

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

Set the permissions on the new file to 0600.

```
sudo chmod 0600 /etc/NetworkManager/system-connections/eduroam-USERNAME
```

4) Option 3 (For Univeristé de Montréal students only): Use `UdeM avec cryptage`

TODO: someone replicate please - LP

> Note: your can use the `autoconnect-priority=XX` inside the `[connection]` block to establish a priority. If you want to connect to one network preferentially if two are available then give it a higher priority.

Save the following block as new file in `/etc/NetworkManager/system-connections/secure`:

```
[connection]
id=secure
uuid=e9cef1bd-f6fb-4c5b-93cf-cca837ec35f2
type=wifi
permissions=
secondaries=
timestamp=1502254646
autoconnect-priority=100

[wifi]
mac-address-blacklist=
mac-address-randomization=0
mode=infrastructure
ssid=UdeM avec cryptage
security=wifi-security

[wifi-security]
key-mgmt=wpa-eap

[802-1x]
eap=peap;
identity=DGTIC UNIP
phase2-auth=mschapv2
password=DGTIC PWD

[ipv4]
dns-search=
method=auto

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

Set the permissions on the new file to 0600.

```
sudo chmod 600 /etc/NetworkManager/system-connections/secure
```

5) Option 4: custom WiFi

First run the following to see what networks are available:

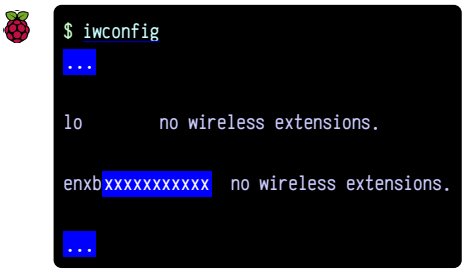
```
$ nmcli dev wifi list
```

You should see the network that you are trying to connect (SSID) to and you should know the password. To connect to it run:

6) Option 5: ETH Wifi

The following instructions will lead you to connect your PI to the “eth” wifi network.

First, run the following on duckiebot



Make note of the name `enxbxxxxxxxxxxx`. ![xxxxxxxxxxx] should be a string that has 11 characters that is formed by numbers and lower case letters.

Second, edit the file `/etc/network/interfaces` which requires `sudo` so that it looks like the following, and make sure the `enxbxxxxxxxxxxx` matches:

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

# The loopback network interface
auto lo
auto enxbxxxxxxxxxxx

# the wired network setting
iface enxbxxxxxxxxxxx inet dhcp

# the wireless network setting
auto wlan0
allow-hotplug wlan0
iface wlan0 inet dhcp
    pre-up wpa_supplicant -B -D wext -i wlan0 -c /etc/wpa_supplicant/wpa_supplicant.conf
    post-down killall -q wpa_supplicant
```

Third, edit the file `/etc/wpa_supplicant/wpa_supplicant.conf` which requires `sudo` so that it looks like the following, and make sure you substitute [identity] and [password] content with your eth account information:

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

network={
    ssid="eth"
    key_mgmt=WPA-EAP
    group=CCMP TKIP
    pairwise=CCMP TKIP
    eap=PEAP
    proto=RSN
    identity="your user name goes here"
    password="your password goes here"
    phase1="peaplabel=0"
    phase2="auth=MSCHAPV2"
    priority=1
}
```

Fourth, reboot your PI.

```
$ sudo reboot
```

Then everything shall be fine. The PI will connect to "eth" automatically everytime it starts.

UNIT I-9

Software setup and RC remote control

Assigned to: Andrea

KNOWLEDGE AND ACTIVITY GRAPH

Requires: Laptop configured, according to Unit I-6 - Installing Ubuntu on laptops.

Requires: You have configured the Duckiebot. The procedure is documented in Unit I-7 - Duckiebot Initialization.

Requires: You have created a Github account and configured public keys, both for the laptop and for the Duckiebot. The procedure is documented in Setup Github access (master).

Results: You can run the joystick demo.

9.1. 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 Setup Github access (master).

Symptom: Other weird errors.

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

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

Or see the hints in the troubleshooting section on the previous page.

9.2. Update the system

The software used for the Duckiebots changes every day, this means that also the dependencies change. In order to check whether your system meets all the requirements for running the software and install all the missing packages (if any), we can run the following script:

```
$ cd ~/duckietown
$ /bin/bash ./dependencies_since_image.sh
```

This command will install only the packages that are not already installed in your system.

9.3. Set up the 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 catkin_make (master).

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

+ comment
I got no error on first execution on the Raspberry Pi

9.4. Clone the duckiefleet repository

Clone the relevant `duckiefleet` repository into `~/duckiefleet`.

See see Duckiefleet directory `DUCKIEFLEET_ROOT` (master) to find the right `duckiefleet` repository.

In `~/.bashrc` set `DUCKIEFLEET_ROOT` to point to the directory:

```
export DUCKIEFLEET_ROOT=~/duckiefleet
```

9.5. Add your vehicle data to the robot database

Next, you need to add your robot to the vehicles database. This is not optional and required in order to launch any ROS scripts.

You have already a copy of the vehicles database in the folder `robots` of `DUCKIEFLEET_ROOT`.

Copy the file `emma.robot.yaml` to `robotname.robot.yaml`, where `robotname` is your robot's hostname. Then edit the copied file to represent your Duckiebot.

➡ For information about the format, see The "scuderia" (vehicle database) (master).

Generate the machines file.

➡ The procedure is listed here: The `machines` file (master).

Finally, push your robot configuration to the duckiefleet repo.

9.6. 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 fol-

lowing the instructions carefully!

➡ Consult again Section 7.11 - Create your user.

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.

9.7. 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".

XXX 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 9.6 - Test that the joystick is detected.

9.8. 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.

UNIT I-10

Reading from the camera

KNOWLEDGE AND ACTIVITY GRAPH

Requires: You have configured the Duckiebot. The procedure is documented in Unit I-7 - Duckiebot Initialization.

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

TODO: put reference

Results: You know that the camera works under ROS.

10.1. Check the camera hardware

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

➡ The procedure is documented in Section 7.13 - Hardware check: camera.

10.2. Create two windows

On the laptop, create two Byobu windows.

➡ A quick reference about Byobu commands is in Byobu (master).

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.

10.3. First window: launch the camera nodes

In the first window, we will launch the nodes that control the camera. All the following commands should be run in the `~/duckietown` directory:

```
$ cd ~/duckietown
```

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 roslaunch (master).

10.4. Second window: view published topics

Switch to the second window. All the following commands should be run in the `~/duckietown` directory:

```
$ cd ~/duckietown
```

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 rostopic (master).

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 Unit I-12 - RC+camera remotely, you will learn how to visualize the image stream on the laptop using `rviz`.

use Ctrl-C to stop `rostopic`.

TODO: Physically focus the camera.

UNIT I-11

RC control launched remotely

Assigned to: Andrea

KNOWLEDGE AND ACTIVITY GRAPH

Requires: You can run the joystick demo from the Raspberry Pi. The procedure is documented in Unit I-9 - Software setup and RC remote control.

Results: You can run the joystick demo from your laptop.

11.1. 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

11.2. 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 9.1 - Clone the Duckietown repository.

Then, you should build the repository.

> ➡ This procedure is documented in Section 9.3 - Set up the ROS environment on the Duckiebot.

11.3. 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 9.5 - Add your vehicle data to the robot database.

11.4. 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 roslaunch (master).

11.5. 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
```

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. If `rqt_console` does not show any message, check out the *Troubleshooting* section below.

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 rqt_console (master).

11.6. Troubleshooting

Symptom: `rqt_console` does not show any message.

Resolution: Open `rqt_console`. Go to the Setup window (top-right corner). Change the "Rosout Topic" field from `/rosout_agg` to `/rosout`. Confirm.

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 `HostKeyAl-`

`gorithms` option. Delete `~/.ssh/known_hosts` and fix your configuration.

➡ The procedure is documented in Local configuration (master).

UNIT I-12

RC+camera remotely

Assigned to: Andrea

KNOWLEDGE AND ACTIVITY GRAPH

Requires: You can run the joystick demo remotely. The procedure is documented in Unit I-11 - RC control launched remotely.

Requires: You can read the camera data from ROS. The procedure is documented in Unit I-10 - Reading from the camera.

Requires: You know how to get around in Byobu. You can find the Byobu tutorial in Byobu (master).

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

12.1. Assumptions

We are assuming that the joystick demo in Unit I-11 - RC control launched remotely worked.

We are assuming that the procedure in Unit I-10 - Reading from the camera succeeded.

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

12.2. Terminal setup

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

➡ A quick reference about Byobu commands is in Byobu (master).

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

12.3. First window: launch the joystick demo

In the first window, launch the joystick remotely using the same procedure in Section 11.4 - Start the demo.

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

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

12.4. 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.

12.5. 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
```

12.6. 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 rviz (master).

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.

12.7. 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.

UNIT I-13

Interlude: Ergonomics [draft]

This section has been removed because it is in status 'draft'. If you are feeling adventurous, you can read it on master.

To disable this behavior, and completely hide the sections, set the parameter show_removed to false in fall2017.version.yaml.

UNIT I-14

Camera calibration

Assigned to: Dzenan Lapandic

KNOWLEDGE AND ACTIVITY GRAPH

Requires: You can see the camera image on the laptop. The procedure is documented in Unit I-12 - RC+camera remotely.

Results: Calibrated camera of the robot

14.1. Intrinsic calibration

1) Setup

Make sure your Duckiebot is on, and both your laptop and Duckiebot are connected to the duckietown network. Next, download and print a PDF of the calibration checkerboard. Fix the checkerboard to a planar surface.

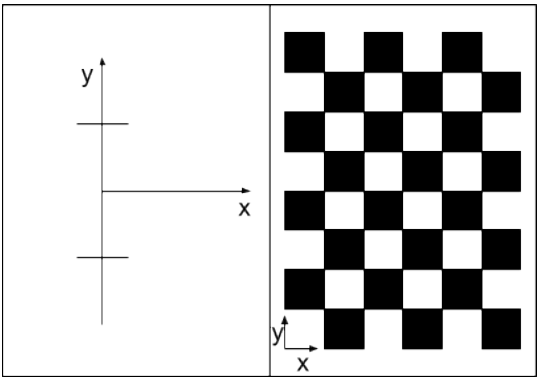


Figure 14.1

2) Calibration

Step 1:

Open three terminals on the laptop.

Step 2:

In the first terminal, remotely launch the joystick process:

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

Step 3:

In the second terminal run the camera calibration:

```
$ cd ~/duckietown
$ source environment.sh
$ source set_ros_master.sh robot name
$ roslaunch duckietown intrinsic_calibration.launch veh:=robot name raw:=true
```

You should see a display screen open on the laptop (Figure 14.2).

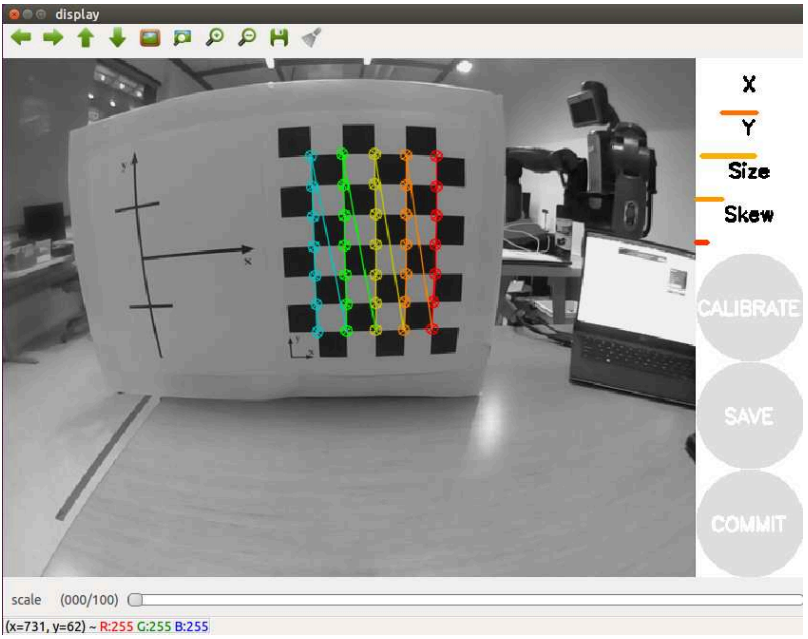


Figure 14.2

Position the checkerboard in front of the camera until you see colored lines overlaying the checkerboard. You will only see the colored lines if the entire checkerboard is within the field of view of the camera. You should also see colored bars in the sidebar of the display window. These bars indicate the current range of the checkerboard in the camera's field of view:

- X bar: the observed horizontal range (left - right)
- Y bar: the observed vertical range (top - bottom)
- Size bar: the observed range in the checkerboard size (forward - backward from the camera direction)
- Skew bar: the relative tilt between the checkerboard and the camera direction

Also, make sure to focus the image by rotating the mechanical focus ring on the lens of the camera.

Now move the checkerboard right/left, up/down, and tilt the checkerboard through various angles of relative to the image plane. After each movement, make sure to pause long enough for the checkerboard to become highlighted. Once you have collected enough data, all four indicator bars will turn green. Press the "CALIBRATE" button in the sidebar. Calibration may take a few moments. Note that the screen may dim. Don't worry, the calibration is working.

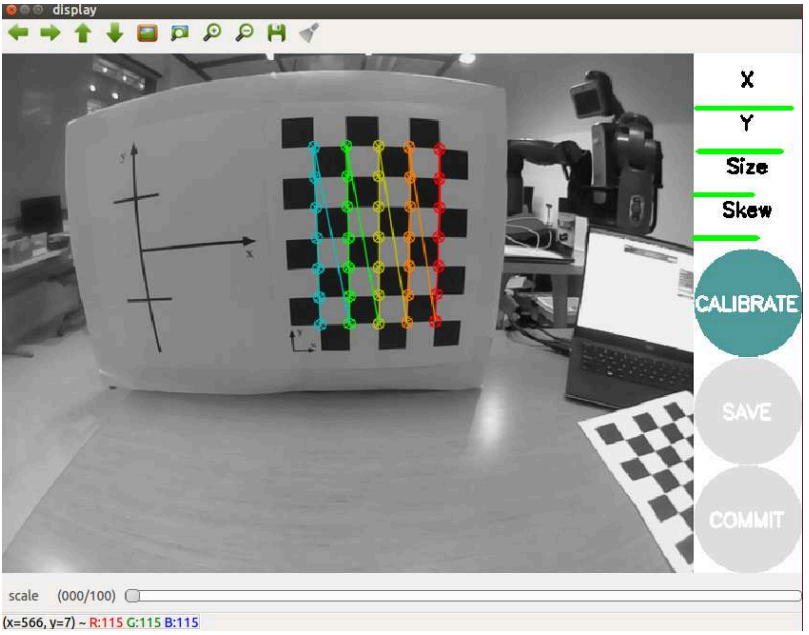


Figure 14.3

3) Save the calibration results

If you are satisfied with the calibration, you can save the results by pressing the "COMMIT" button in the side bar.

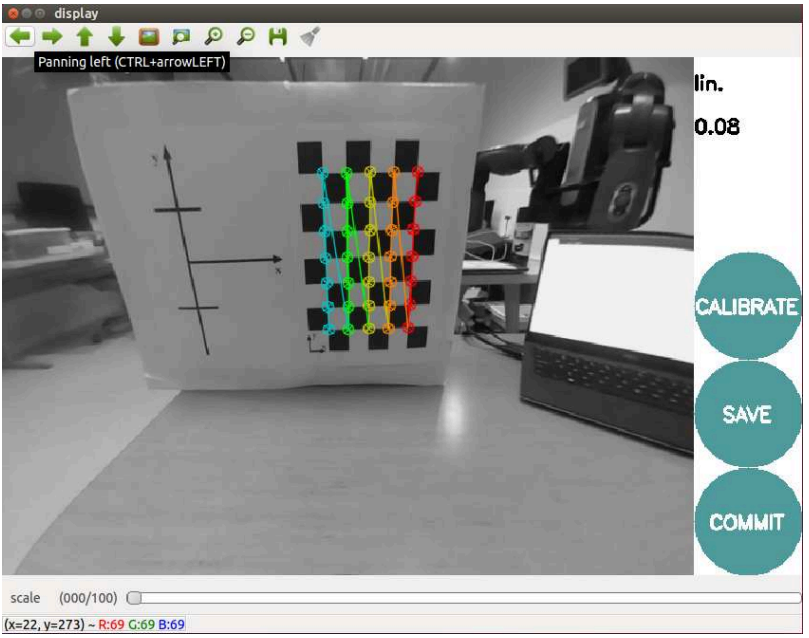


Figure 14.4

This will automatically save the calibration results on your Duckiebot:

```
~/duckietown/catkin_ws/src/00-infrastructure/duckietown/config/baseline/calibration/
camera_intrinsic/robot name.yaml
```

Step 7:

Now let's push the `robot name.yaml` file to the git repository. In the third terminal connect to your Duckiebot:

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

Update your local git repository:

```
$ cd ~/duckietown
$ git pull
```

Update your local git repository and push the changes to github:

```
$ git checkout -b git username-devel
$ git add ~/duckietown/catkin_ws/src/00-infrastructure/duckietown/config/baseline/
calibration/camera_intrinsic/robot name.yaml
$ git commit -m "add robot name intrinsic calibration file"
$ git push origin git username-devel
```

You can obtain the intrinsic calibration results on your laptop by updating your local git repository on your laptop:

```
$ cd ~/duckietown
$ git fetch
$ git checkout git username-devel
```

Before moving on to the extrinsic calibration, make sure to kill all running processes by pressing Ctrl-C in each of the terminal windows.

14.2. Extrinsic calibration

1) Setup

Arrange the Duckiebot and checkerboard according to Figure 14.5. Note that the axis of the wheels should be aligned with the y-axis.

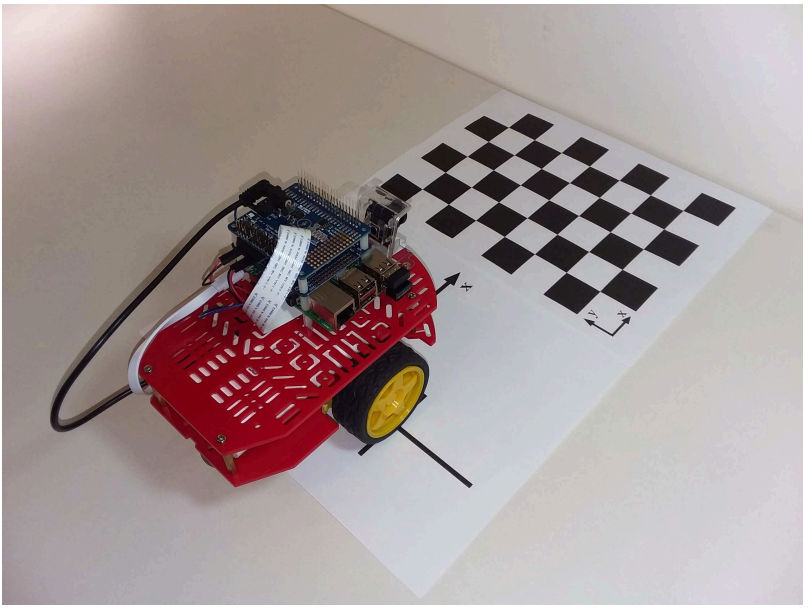


Figure 14.5

Figure 14.6 shows a view of the calibration checkerboard from the Duckiebot. To ensure proper calibration there should be no clutter in the background.

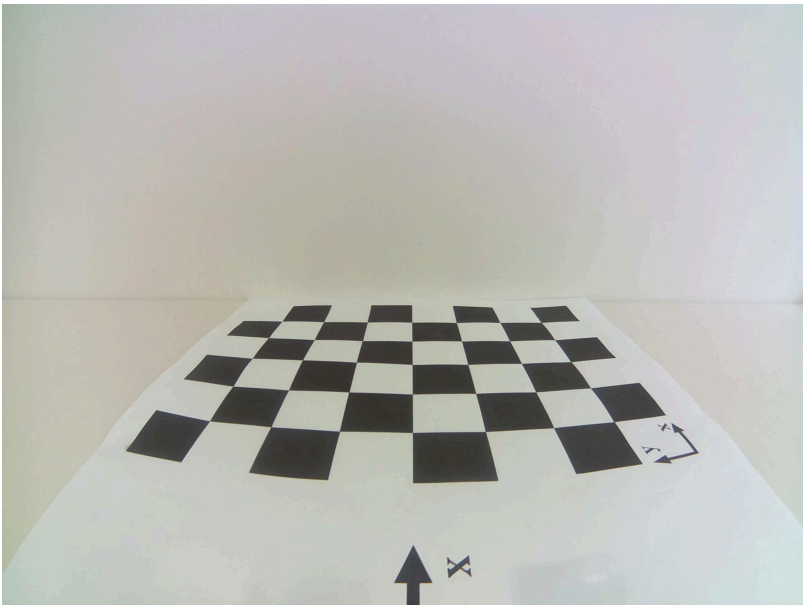


Figure 14.6

2) Calibration

Step 1:

Open four terminals terminals on the laptop.

Step 2:

In the first terminal, remotely launch the joystick process:

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

Step 3:

In the second terminal launch the camera:

```
$ cd ~/duckietown
$ source environment.sh
$ source set_ros_master.sh robot name
$ roslaunch duckietown camera.launch raw:=1 veh:=robot name
```

Step 4:

In the third terminal run the ground projection node:

```
$ cd ~/duckietown
$ source environment.sh
$ source set_ros_master.sh robot name
$ roslaunch ground_projection ground_projection.launch veh:=robot name local:=1
```

Step 5:

In the fourth terminal, check that everything is working properly.

```
$ cd ~/duckietown
$ source environment.sh
$ source set_ros_master.sh robot name
$ rostopic list
```

You should see new ros topics:

```
/robot name/camera_node/camera_info
/robot name/camera_node/framerate_high_switch
/robot name/camera_node/image/compressed
/robot name/camera_node/image/raw
/robot name/camera_node/raw_camera_info
```

The ground_projection node has two services. They are not used during operation. They just provide a command line interface to trigger the extrinsic calibration (and for debugging).

```
$ rosservice list
```

You should see something like this:

```
...
/robot name/ground_projection/estimate_homography
/robot name/ground_projection/get_ground_coordinate
...
```

If you want to check whether your camera output is similar to the one at the Figure 14.6 you can start `rqt_image_view`:

```
$ rosrun rqt_image_view rqt_image_view
```

In the `rqt_image_view` interface, click on the drop-down list and choose the image topic:

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

Now you can estimate the homography by executing the following command:

```
$ rosservice call /robot name/ground_projection/estimate_homography
```

This will do the extrinsic calibration and automatically save the file to your laptop:

```
~/duckietown/catkin_ws/src/00-infrastructure/duckietown/config/baseline/calibration/
camera_extrinsic/robot name.yaml
```

As before, add this file to your local Git repository on your laptop, push the changes, and update your the local git repository on your Duckiebot.

> **Note:** we are in the process of rewriting the configuration system, so in a while "commit to the repository" is not going to be the right thing to do. We will communicate when the transition will happen

UNIT I-15

Taking a log

KNOWLEDGE AND ACTIVITY GRAPH

> **Requires:** Unit I-10 - Reading from the camera
>
> **Requires:** Unit I-9 - Software setup and RC remote control
>
> **Results:** A log

> **Note:** it is recommended that you log to your USB and not to your SD card. To mount your USB see Mounting USB drives (master)

run on duckiebot:

```
$ make demo-joystick-camera
```

run on laptop:

```
$ cd DUCKIETOWN_ROOT
$ source set_ros_master.sh ROBOT_NAME
$ rqt_image_view
```

and verify that indeed your camera is streaming imagery.
on your duckiebot in a new tab (F2 in byobu)

```
$ rosbag record -a -o /media/logs/ROBOT_NAME
```

where here we are assuming that you are logging to the USB and have following Mounting USB drives (master).

UNIT I-16

Verify a log

On your robot run:

```
$ rosbag info FULL_PATH_TO_BAG --freq
```

- verify that the "duration" of the log seems "reasonable" - it's about as long as you ran the log command for
- verify that the "size" of the log seems "reasonable" - the log size should grow at about 220MB/min
- verify in the output that your camera was publishing very close to **30.0Hz** and verify that you joysick was publishing at very close to **2.0Hz**

TODO: More complex log verification methods

PART J

Operation manual - DB17-1c Duckiebot configurations

This section contains the acquisition, assembly and setup instructions for the `DB17-1c` configurations. These instructions are separate from the rest as they approximately match the `b` releases of the Fall 2017 Duckietown Engineering Co. branches.

To understand how configurations and releases are defined, refer to: Unit I-1 - Duckiebot configurations.

UNIT J-1

Acquiring the parts for the Duckiebot DB17-1c

Upgrading your `DB17` (`DB17-wjd`) configuration to `DB17-1c` (`DB17-wjd1c`) starts here, with purchasing the necessary components. We provide a link to all bits and pieces that are

needed to build a `DB17-1c` Duckiebot, along with their price tag. If you are wondering what is the difference between different Duckiebot configurations, read Unit I-1 - Duckiebot configurations.

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.

KNOWLEDGE AND ACTIVITY GRAPH

Requires: A Duckiebot in `DB17-wjd` configuration.

Requires: Cost: USD 77 + Bumpers manufacturing solution

Requires: Time: 21 Days (LED board manufacturing and shipping time)

Results: A kit of parts ready to be assembled in a `DB17-1c` configuration Duckiebot.

Next: After receiving these components, you are ready to do some soldering before assembling your `DB17-1c` Duckiebot.

1.1. Bill of materials

TABLE 1.1. BILL OF MATERIALS

LEDs (`DB17-1`)	USD 10
LED HAT (`DB17-1`)	USD 28.20 for 3 pieces
Power Cable (`DB17-1`)	USD 7.80
20 Female-Female Jumper Wires (300mm) (`DB17-1`)	USD 8
Male-Male Jumper Wire (150mm) (`DB17-1`)	USD 1.95
PWM/Servo HAT (`DB17-1`)	USD 17.50
Bumpers	TBD (custom made)
40 pin female header (`DB17-1`)	USD 1.50
5 4 pin female header (`DB17-1`)	USD 0.60/piece
2 16 pin male header (`DB17-1`)	USD 0.61/piece
12 pin male header (`DB17-1`)	USD 0.48/piece
3 pin male header (`DB17-1`)	USD 0.10/piece
2 pin female shunt jumper (`DB17-1`)	USD 2/piece
5 200 Ohm resistors (`DB17-1`)	USD 0.10/piece
10 130 Ohm resistors (`DB17-1`)	USD 0.10/piece
Caster (`DB17-c`)	USD 6.55/4 pieces
4 Standoffs (M3.5 12mm F-F) (`DB17-c`)	USD 0.63/piece
8 Screws (M3.5x8mm) (`DB17-c`)	USD 4.58/100 pieces
8 Split washer lock (`DB17-c`)	USD 1.59/100 pieces
Total for `DB17-wjd` configuration	USD 212
Total for `DB17-1c` components	USD 77 + Bumpers
Total for `DB17-wjd1c` configuration	USD 299+Bumpers

1.2. LEDs

The Duckiebot is equipped with 5 RGB LEDs (Figure 1.1). 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.

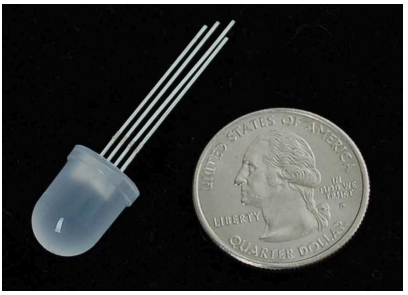


Figure 1.1. The RGB LEDs

1) LED HAT

The LED HAT (Figure 1.2) 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.

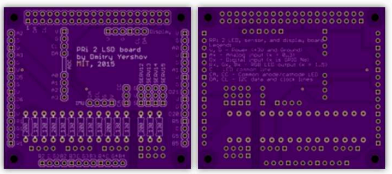


Figure 1.2. The LED HAT

2) PWM/Servo HAT

The PWM/Servo HAT (Figure 1.3) 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 1.3. 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 1.4).

Figure 1.4. 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 1.5) to power the DC Stepper Motor HAT from the PWM/Servo HAT.

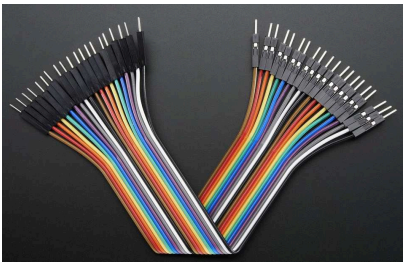


Figure 1.5. Premier Male-Male Jumper Wires

5) Female-Female Jumper Wires

20 Female-Female Jumper Wires (Figure 1.6) are necessary to connect 5 LEDs to the LED HAT.

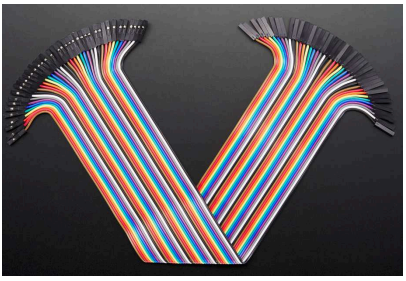


Figure 1.6. Premier Female-Female Jumper Wires

1.3. Bumpers

These bumpers are designed to keep the LEDs in place and are therefore used only in configuration `DB17-1`. They are custom designed parts, so they must be produced and cannot be bought. We used laser cutting facilities.

Figure 1.7. The Bumpers

1.4. Headers, resistors and jumper

Upgrading `DB17` to `DB17-1` 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 1.8. The Headers

Figure 1.9. The Resistors

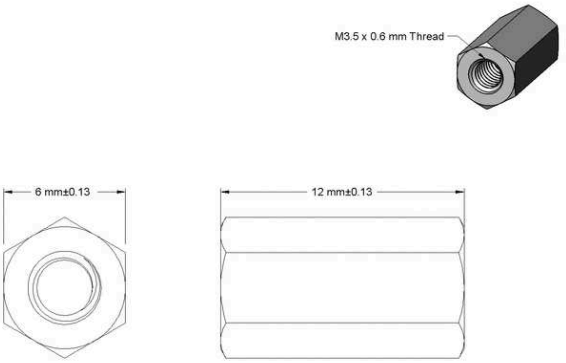
1.5. Caster (`DB17-c`)

The caster (Figure 1.10) is an `DB17-c` component that substitutes the steel omnidirectional wheel that comes in the Magician Chassis package. Although the caster is not essential, it provides smoother operations and overall enhanced Duckiebot performance.

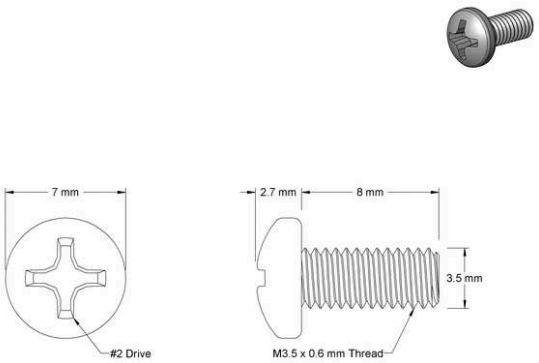
Figure 1.10. The caster wheel

To assemble the caster at the right height we will need to purchase:

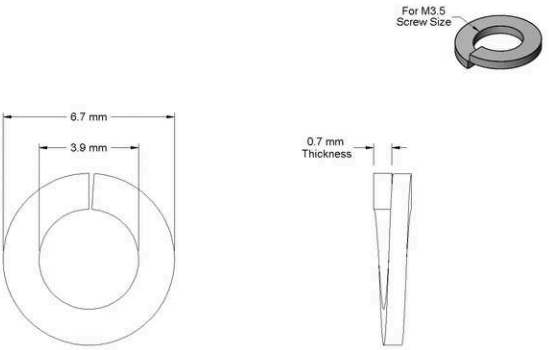
- 4 standoffs (M3 12mm F-F) (Figure 1.11a - Standoffs for caster wheel.),
- 8 screws (M3x8mm) (Figure 1.11b - Screws for caster wheel.), and
- 8 split lock washers (Figure 1.11c - Split lock washers for caster wheel.).



(a) Standoffs for caster wheel.



(b) Screws for caster wheel.



(c) Split lock washers for caster wheel.

Figure 1.11. Mechanical bits to assemble the caster wheel.

TODO: missing figures, update caster bits figures

UNIT J-2

Soldering boards for DB17-1 [draft]

This section has been removed because it is in status 'draft'. If you are feeling adventurous, you can read it on master.

To disable this behavior, and completely hide the sections, set the parameter show_removed to false in fall2017.version.yaml.

UNIT J-3

Assembling the Duckiebot DB17-wjdlc [draft]

This section has been removed because it is in status 'draft'. If you are feeling adventurous, you can read it on master.

To disable this behavior, and completely hide the sections, set the parameter show_removed to false in fall2017.version.yaml.

UNIT J-4

Bumper Assembly [draft]

This section has been removed because it is in status 'draft'. If you are feeling adventurous, you can read it on master.

To disable this behavior, and completely hide the sections, set the parameter show_removed to false in fall2017.version.yaml.

UNIT J-5

DB17-1 setup [draft]

This section has been removed because it is in status 'draft'. If you are feeling adventurous, you can read it on master.

To disable this behavior, and completely hide the sections, set the parameter show_removed to false in fall2017.version.yaml.

Part K
Appendix

Unit K-1
Bibliography

[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.

[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

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

[7] Albert Einstein. Zur Elektrodynamik bewegter Körper. (German) On the electrodynamics of moving bodies. *Annalen der Physik*, 322(10):891–921, 1905. DOI

[15] Ivan Savov. Linear algebra explained in four pages. https://minireference.com/static/tutorials/linear_algebra_in_4_pages.pdf, 2017. Online; accessed 23 August 2017.

[14] Kaare Brandt Petersen and Michael Syskind Pedersen. The matrix cookbook. .pdf

[17] Matrix (mathematics). Matrix (mathematics) — Wikipedia, the free encyclopedia, 2017. Online; accessed 2-September-2017. www:

[18] Peter D. Lax. *Functional Analysis*. Wiley Interscience, New York, NY, USA, 2002.

[19] Athanasios Papoulis and S. Unnikrishna Pillai. *Probability, Random Variables and Stochastic Processes*. McGraw Hill, fourth edition, 2002.

[20] David J. C. MacKay. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, New York, NY, USA, 2002.

[21] H. Choset, W. Burgard, S. Hutchinson, G. Kantor, L. E. Kavraki, K. Lynch, and S. Thrun. *Principles of robot motion: Theory, agorithms, and implementation*. MIT Press, June 2005.

[22] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, New York, NY, USA, 2006.

[23] M. Miskowicz. *Event-Based Control and Signal Processing*. Embedded Systems. CRC Press, 2015.

http

[24] Karl J. Aström. *Event Based Control*, pages 127–147. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. DOI http

[25] Edward A Lee and David G Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987.

[26] Rajit Manohar and K Mani Chandy. Delta-dataflow networks for event stream processing. pages 1–6, June 2010.

[27] Rached Dhaouadi and A. Abu Hatab. Dynamic modeling of differential-drive mobile robots using lagrange and newton-euler methodologies: A unified framework. *Advances in Robotics & Automation*, 2(2):1–7, 2013.

[28] Romano M DeSantis. Modeling and path-tracking control of a mobile wheeled robot with a differential drive. *Robotica*, 13(4):401–410, 1995.

[29] Richard Szeliski. *Computer Vision: Algorithms an Applications*. Springer, 2010. http

[30] David Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*. Prentce Hall, Upper Saddle River, NJ, 2011.

[31] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004.

UNIT K-2

Last modified

- Mon, Oct 02: Unit D-2 - Duckietown Appearance Specification (master) (liampaull)
- Mon, Oct 02: Unit A-11 - Montréal branch diary (Liam Paull)
- Mon, Oct 02: Unit I-5 - Assembling the Duckiebot `DB17-wjd` (TTIC) (Andrea F. Daniele)
- Mon, Oct 02: Unit I-4 - Assembling the Duckiebot `DB17`, `DB17-jwd` (Andrea F. Daniele)
- Sun, Oct 01: Unit C-16 - Wheel calibration (master) (Jacopo Tani)
- Sun, Oct 01: Unit I-3 - Preparing the power cable for `DB17` (Jacopo Tani)
- Sun, Oct 01: Unit F-1 - Probability basics (Selçuk Ercan)
- Sun, Oct 01: Unit A-7 - Organization (Andrea Censi)
- Sun, Oct 01: Unit I-9 - Software setup and RC remote control (Jacopo Tani)
- Sat, Sep 30: Unit H-7 - Exercise: Bag instagram (master) (Andrea F Daniele)
- Sat, Sep 30: Unit H-5 - Exercise: Bag thumbnails (master) (Andrea F Daniele)
- Sat, Sep 30: Unit H-3 - Exercise: Simple data analysis from a bag (master) (Andrea F Daniele)
- Sat, Sep 30: Unit I-2 - GNU/Linux general notions (master) (syangav)
- Sat, Sep 30: Part J - Operation manual - `DB17-1c` Duckiebot configurations (Jacopo Tani)
- Sat, Sep 30: Unit I-8 - Networking aka the hardest part (syangav)
- Sat, Sep 30: Unit I-1 - Duckiebot configurations (Jacopo Tani)
- Sat, Sep 30: Unit D-6 - Traffic lights Assembly (master) (Jacopo Tani)
- Sat, Sep 30: Unit D-7 - Semantics of LEDS (master) (Jacopo Tani)
- Sat, Sep 30: Unit J-3 - Assembling the Duckiebot `DB17-wjdlc` [draft] (Jacopo Tani)
- Sat, Sep 30: Unit J-5 - `DB17-l` setup [draft] (Jacopo Tani)
- Sat, Sep 30: Unit I-21 - Moving files between computers (master) (somml)

- Sat, Sep 30: Unit I-2 - Acquiring the parts for the Duckiebot DB17-wjd (Jacopo Tani)
- Sat, Sep 30: Unit J-1 - Acquiring the parts for the Duckiebot DB17-1c (Jacopo Tani)
- Sat, Sep 30: Unit F-3 - Sets (master) (Jacopo Tani)
- Sat, Sep 30: Unit F-5 - Complex numbers (master) (Jacopo Tani)
- Sat, Sep 30: Unit F-7 - Matrices basics (master) (Jacopo Tani)
- Sat, Sep 30: Unit F-8 - Matrix inversions (master) (Jacopo Tani)
- Sat, Sep 30: Unit F-9 - Matrices and vectors (master) (Jacopo Tani)
- Sat, Sep 30: Unit F-10 - Matrix operations (basic) (master) (Jacopo Tani)
- Sat, Sep 30: Unit F-11 - Matrix operations (complex) (master) (Jacopo Tani)
- Sat, Sep 30: Unit F-12 - Matrix diagonalization (master) (Jacopo Tani)
- Sat, Sep 30: Unit F-13 - Norms (master) (Jacopo Tani)
- Sat, Sep 30: Unit F-17 - Kinematics (master) (Jacopo Tani)
- Sat, Sep 30: Unit F-18 - Dynamics (master) (Jacopo Tani)
- Fri, Sep 29: Unit C-7 - Reproducing the image (master) (Andrea Censi)
- Fri, Sep 29: Unit H-1 - Exercise: Basic image operations (master) (Andrea F Daniele)
- Fri, Sep 29: Unit J-6 - Minimal ROS node - pkg_name (master) (Samuel Laferriere)
- Fri, Sep 29: Unit D-1 - Contributing to the documentation (Andrea Censi)
- Fri, Sep 29: Unit F-2 - Linearity and Vectors [draft] (Jacopo Tani)
- Fri, Sep 29: Unit F-14 - From ODEs to LTI systems (master) (Jacopo Tani)
- Fri, Sep 29: Unit F-15 - Linearization (master) (Jacopo Tani)
- Fri, Sep 29: Unit G-6 - Duckiebot modeling (Jacopo Tani)
- Fri, Sep 29: Unit C-3 - Soldering boards for DB17 (master) (Andrea Censi)
- Fri, Sep 29: Unit D-4 - Traffic lights Parts (master) (Andrea Censi)
- Thu, Sep 28: Unit I-14 - Camera calibration (Andrea Censi)
- Thu, Sep 28: Unit F-23 - Types (master) (Andrea Censi)
- Thu, Sep 28: Unit H-9 - Exercise: Augmented Reality (master) (Andrea Censi)
- Thu, Sep 28: Unit E-1 - Exercise: Basic image operations, adult version (Andrea F. Daniele)
- Thu, Sep 28: Unit A-10 - Zürich branch diary (Andrea Censi)
- Thu, Sep 28: Unit E-4 - Exercise: Live Instagram (Andrea Censi)
- Thu, Sep 28: Unit I-12 - RC+camera remotely (Andrea Censi)
- Wed, Sep 27: Unit I-15 - Taking a log (liampaull)
- Wed, Sep 27: Unit A-6 - Git usage guide for Fall 2017 (liampaull)
- Wed, Sep 27: Unit B-2 - Checkoff: Take a Log (Liam Paull)
- Wed, Sep 27: Unit B-3 - Homework: Data Processing (liampaull)
- Wed, Sep 27: Unit A-12 - Chicago branch Diary (Matthew Walter)
- Tue, Sep 26: Unit I-13 - Interlude: Ergonomics [draft] (Liam Paull)
- Tue, Sep 26: Unit I-19 - Accessing computers using SSH (master) (Liam Paull)
- Tue, Sep 26: Unit I-7 - Duckiebot Initialization (Liam Paull)
- Tue, Sep 26: Unit B-1 - Checkoff: Duckiebot Assembly and Configuration (liampaull)
- Mon, Sep 25: Unit A-2 - First Steps in Duckietown (Kirsten Bowser)
- Mon, Sep 25: Unit A-3 - Logistics for Zürich branch (Andrea Censi)
- Mon, Sep 25: Unit A-4 - Logistics for Montréal branch (Andrea Censi)
- Mon, Sep 25: Unit A-5 - Logistics for Chicago branch (Andrea Censi)
- Mon, Sep 25: Unit J-4 - Bumper Assembly [draft] (Jacopo Tani)
- Sun, Sep 24: Part I - Operation manual - Duckiebot (Jacopo Tani)
- Sun, Sep 24: Unit J-2 - Soldering boards for DB17-1 [draft] (Jacopo Tani)
- Sun, Sep 24: Unit I-6 - Installing Ubuntu on laptops (Jacopo Tani)
- Sun, Sep 24: Unit I-10 - Reading from the camera (Jacopo Tani)

- Sun, Sep 24: Unit I-11 - RC control launched remotely (Jacopo Tani)
- Sun, Sep 24: Unit A-9 - Slack Channels (Jacopo Tani)
- Sun, Sep 24: Unit E-3 - Exercise: Instagram filters (Andrea F. Daniele)
- Sun, Sep 24: Unit J-9 - Duckietown utility library (master) (Andrea F. Daniele)
- Sun, Sep 24: Unit E-2 - Exercise: Bag in, bag out (Andrea Censi)
- Sun, Sep 24: Unit J-1 - Python (master) (Andrea F. Daniele)
- Sat, Sep 23: Unit G-7 - Modern signal processing (master) (Andrea Censi)
- Sat, Sep 23: Part A - Fall 2017 (Andrea Censi)
- Sat, Sep 23: Unit A-1 - The Fall 2017 Duckietown experience (Andrea Censi)
- Sat, Sep 23: Unit A-13 - Guide for TAs (Andrea Censi)
- Sat, Sep 23: Unit A-14 - Guide for mentors [draft] (Andrea Censi)
- Sat, Sep 23: Unit I-30 - ROS installation and reference (master) (Andrea F. Daniele)
- Sat, Sep 23: Unit F-21 - Time series (master) (Andrea Censi)
- Sat, Sep 23: Unit F-24 - Computer science concepts (master) (Andrea Censi)
- Fri, Sep 22: Unit C-2 - Duckietown history and future (Andrea Censi)
- Thu, Sep 21: Part G - A course in autonomy (master) (Andrea Censi)
- Thu, Sep 21: Part E - Exercises (Andrea Censi)
- Thu, Sep 21: Unit H-10 - Exercise: Git and conventions (master) (Andrea Censi)
- Thu, Sep 21: Unit H-28 - Exercise: Who watches the watchmen? (optional) (master) (Andrea Censi)
- Thu, Sep 21: Unit F-22 - Transformations (master) (Falcon Dai)
- Thu, Sep 21: Unit F-1 - Chapter template (master) (Jacopo Tani)
- Thu, Sep 21: Unit G-12 - Odometry Calibration (master) (Jacopo Tani)
- Wed, Sep 20: Unit G-2 - Autonomy overview (Jon Michaux)
- Mon, Sep 18: Unit G-4 - System architectures basics (master) (Andrea Censi)
- Mon, Sep 18: Unit F-20 - Reference frames (master) (Falcon Dai)
- Sun, Sep 17: Unit D-4 - Using LaTeX constructs in documentation (Andrea Censi)
- Sun, Sep 17: Unit F-19 - Coordinate systems (master) (Andrea Censi)
- Sun, Sep 17: Unit G-4 - Autonomy architectures [draft] (Andrea Censi)
- Sun, Sep 17: Unit D-10 - Learning in Duckietown [draft] (Andrea Censi)
- Sun, Sep 17: Unit I-24 - Liclipse (master) (Andrea Censi)
- Sat, Sep 16: Unit D-8 - The Duckuments bot (Andrea Censi)

Page left blank