This book describes the features of our documentation system and the procedures to update it.

Part A - Contributing to the documentation ........................................3

Part B - Markduck format........................................................................20

Part C - Documentation style guide.......................................................41

Part D - Backend for docs.duckietown.org..........................................46

Part E - Slides.......................................................................................49
PART A

Contributing to the documentation

This part describes the workflow for contributing to our documentation.

Unit A-1 - Overview .................................................................4
Unit A-2 - The simple way to edit the documentation ......................5
Unit A-3 - Second method: local editing+docker ............................8
Unit A-4 - Troubleshooting for Duckumentation ...........................13
Unit A-5 - Markdown editors ......................................................15
Unit A-6 - (Experts) Compiling all books ....................................16
UNIT A-1

Overview

1.1. Where the documentation is
The documentation is contained in a series of repositories called docs- *short name*:

- docs-duckumentation (this book)
- docs-AIDO
- docs-software_devel

1.2. Documentation format
The documentation is written as a series of small files in Markdown format.
It is then processed by a series of scripts to create publication-quality PDF and an online HTML version.
You can find all these artifacts produced at the site docs.duckietown.org.
UNIT A-2

The simple way to edit the documentation

The simplest way to contribute to the documentation is to click any of the “✎” icons next to the headers, in the book itself.

Figure 2.1. Click on edit button.
They link to the “edit” page in Github. There, one can make and commit the edits in only a few seconds.

Figure 2.2. You will land on GitHub
Do your edits where appropriate.

Figure 2.3. Editing the docs.
You can check the outcome by clicking on Preview changes, note that not all functionalities are visible by the preview. For large changes refer to Unit A-3 - Second method:
local editing+docker.

Figure 2.4. A preview of the changes.
Then after your edits, in the bottom part of the webpage, describe your commit and click on Propose file change. When committing, please choose “create branch”.

Figure 2.5. Describe and commit.
Then, click on Create pull request, as you probably don’t have rights to push directly.

Figure 2.6. Create a pull request.
Again you need to confirm that you want to open the pull request.
The simple way to edit the documentation

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across tags.

Figure 2.7. Confirm a pull request.
UNIT A-3
Second method: local editing+docker

This section describes the workflow to edit the documentation for one single book.

In a nutshell:
- You *fork* the repos to your Github account.
- You compile locally using a Docker container (no installation necessary).
- You contribute by opening a pull request.

3.1. Workflow

1) Github setup

We assume that you have setup a Github account with working public keys.

- Basic SSH config

  ![Warning]

  I will ignore this because it is an external link.

  > I do not know what is indicated by the link '#software_reference/github-access'.

  Location not known more precisely.

  Created by function `check_if_any_href_is_invalid` in module `mcdp_docs.check_missing_links`.

- Key pair creation

  ![Warning]

  I will ignore this because it is an external link.

  > I do not know what is indicated by the link '#software_reference/howto-create-key-pair'.

  Location not known more precisely.

  Created by function `check_if_any_href_is_invalid` in module `mcdp_docs.check_missing_links`.

- Adding public key on Github

  ![Warning]

  I will ignore this because it is an external link.

  > I do not know what is indicated by the link '#software_reference/github-access'.

  Location not known more precisely.

  Created by function `check_if_any_href_is_invalid` in module `mcdp_docs.check_missing_links`.

I will ignore this because it is an external link.

> I do not know what is indicated by the link '#software_reference/github-access'.

Location not known more precisely.

Created by function check_if_any_href_is_invalid in module mcdp_docs.check_missing_links.

2) **Install Docker**

Before you start, make sure that you have installed Docker.

3) **Install the Duckietown Shell**

Install the Duckietown Shell using these instructions.

4) **Fork the docs-book repo on the Github site**

Fork one of the docs-book repos on the Github site (Figure 3.1).
This will create a new repo on your account that is linked to the original one.

![Forking a repository on Github](image)

Figure 3.1

5) **Checkout your fork locally**

Check out the forked repository as you would do normally.

6) **Initialize your folder**

Go into the folder:

```
$ cd docs-book
```

When compiling a book for the first time, you need to run:

```
$ git submodule init
```

And:
7) Do your edits

Do your edits on your local copy. The source files are in the director `book/ book`. The file and folder names start with numbers, these are used to determine the order that things show up in output.

8) Compile

Compile using the `docs` commands in the Duckietown Shell:

```bash
$ dts docs build
```

Re-compile from scratch using:

```bash
$ dts docs clean
$ dts docs build
```

If there are errors you should open `duckuments-dist/errors.html` and look at them and fix them.

9) Look at the local copy

Open the file `index.html` and navigate to whichever pages you have changed to make sure that they look the way want them to.

10) Commit and push

Commit and push as you would do normally.

11) Make a pull request

Create a pull request to the original repository.

*Option 1: Use the Github website:*

Github offers a nice interface to create a pull request.

*Option 2: Use the command-line program `hub`:*

You can create a pull request from the command-line using `hub` (unknown ref `software_reference/hub`)

(I will ignore this because it is an external link.

> I do not know what is indicated by the link '#software_reference/hub'.

Location not known more precisely.)
3.2. Using CircleCI

Circle CI makes it easier to check whether there are problems to be fixed.

1) Sign up on Circle

Sign up on the Circle CI service, at the link circleci.com.

2) Activate your build on Circle

Activate the building at the link:

https://circleci.com/setup-project/gh/ username /duckuments

where  username  is your Github username.

Click “start building”.

Make sure everything compiles on Circle:

Go to the URL:

https://circleci.com/gh/ username /duckuments

to see the status of your build.

You can also preview the results by clicking the “artifacts” tab and selecting index.html from the list.
SECOND METHOD: LOCAL EDITING+DOCKER

Test Summary

<table>
<thead>
<tr>
<th>Container 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>duckuments-dist/</td>
</tr>
<tr>
<td>➔ AE.driving_olympics/</td>
</tr>
<tr>
<td>➔ all_crossref.html</td>
</tr>
<tr>
<td>➔ class_fall2017/</td>
</tr>
<tr>
<td>➔ class_fall2017_projects/</td>
</tr>
<tr>
<td>➔ code_docs/</td>
</tr>
<tr>
<td>➔ deprecated/</td>
</tr>
<tr>
<td>➔ drafts/</td>
</tr>
<tr>
<td>➔ documentation/</td>
</tr>
<tr>
<td>errors_and_warnings.manifest.yml</td>
</tr>
<tr>
<td>errors_and_warnings.pickle</td>
</tr>
<tr>
<td>➔ exercises/</td>
</tr>
<tr>
<td>➔ guide_for_instructors/</td>
</tr>
<tr>
<td>index.html</td>
</tr>
<tr>
<td>➔ junit/</td>
</tr>
<tr>
<td>➔ learning_materials/</td>
</tr>
<tr>
<td>➔ openmanual_duckiebot/</td>
</tr>
<tr>
<td>➔ openmanual_duckietown/</td>
</tr>
<tr>
<td>➔ prerequisites/</td>
</tr>
<tr>
<td>➔ software_architecture/</td>
</tr>
<tr>
<td>➔ software_carpentry/</td>
</tr>
<tr>
<td>➔ software_devel/</td>
</tr>
<tr>
<td>summary.html</td>
</tr>
<tr>
<td>summary.manifest.yml</td>
</tr>
<tr>
<td>tasks.txt</td>
</tr>
<tr>
<td>➔ the_duckietown_project/</td>
</tr>
<tr>
<td>warnings.html</td>
</tr>
<tr>
<td>➔ wc/</td>
</tr>
<tr>
<td>package.tgz</td>
</tr>
</tbody>
</table>

Figure 3.2
UNIT A-4

Troubleshooting for Duckumentation

4.1. Markduck problems

First, see the section Unit B-6 - Troubleshooting for common problems and their resolution.

Please report problems with the duckuments using the duckuments issue tracker.

Special notes:

- 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 rev-parse HEAD
```

4.2. Problem: The building hangs

1) Cause: insufficient memory

This might be due to insufficient memory.

For example, on Mac, the default setting is 2GB of RAM. Try increasing it (Figure 4.1).
A workaround is to stop and restart the process (but without doing `dts docs clean`).

2) **Cause: downloading in the background**

Sometimes the problem is that the code is downloading something in the background, for example a Git repository.

Currently this is hard to debug.
We suggest the following:
- Find a visual Markdown editor that you like.
- Complement with a superb text-only editor.

5.1. Graphical Markdown editors

1) Typhora

Typhora is our favorite because it allows to preview LaTeX formulas.

5.2. Text editors good for Markdown

5.3. Atom

- (unknown ref software_reference/atom)

I will ignore this because it is an external link.

> I do not know what is indicated by the link '#software_reference/atom'.

Location not known more precisely.
Created by function check_if_any_href_is_invalid in module mcdp_docs.check_missing_links.

5.4. PyCharm

- Unit B-4 - PyCharm
UNIT A-6
(Experts) Compiling all books

This section describes how to compile all the books at the same time.

6.1. Setup

1) Fork the duckuments repo

In the following, we are going to assume that the documentation system is installed in ~/duckuments. However, it can be installed anywhere. You will need to add your public key to github following these instructions.

Fork the duckietown/duckuments repository in the ~/duckuments directory:

```bash
$ git clone --depth 10 git@github.com:duckietown/duckuments ~/duckuments
$ cd ~/duckuments
$ git submodule init --recursive
$ git submodule update --recursive
```

The command `--depth 10` tells it we do not care about the whole history.

2) Setup the virtual environment

```bash
$ cd ~/duckuments
$ virtualenv --system-site-packages deploy
$ git clone git@github.com:AndreaCensi/mcdp.git
```

6.2. Compiling the documentation

1) Compiling all the books

To compile everything from scratch, run:

```
$ make realclean all -j
```

To see the results, open the file

```
~/duckuments/duckuments-dist/index.html
```

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

```
$ make all -j
```
2) Compiling a single book

After you have compiled all the books, you can use one of the following commands to re-compile one single book:

```
$ make book-duckumentation
$ make book-the_duckietown_project
$ make book-opmanual_duckiebot
$ make book-opmanual_duckietown
$ make book-software_reference
$ make book-software_devel
$ make book-software_architecture
$ make book-class_fall2017
$ make book-class_fall2017_projects
$ make book-learning_materials
$ make book-exercises
$ make book-drafts
$ make book-guide_for_instructors
$ make book-deprecated
$ make book-preliminaries
```

The compilation is always incremental, but sometimes you might need to do a reset using:

```
$ make realclean
```

6.3. Editing the books

The books sources are in `docs/docs-book`.

Each of those are a submodule.

To edit, do the following.

Fork the `docs-book` repository that you wish to edit.

For example, for `docs-exercises`, it looks like this:

```
$ cd ~/duckuments/docs/docs-exercises
$ hub fork
Updating username
From ssh://github.com/duckietown/docs-exercises
* [new branch] master -> username/master
new remote: username
```

The command has forked the repository.

It also created a new Git remote called `username`:
Tell Git that you want to push to your fork:

```bash
$ git branch --set-upstream-to username/master
```

Now, when you push you will push to your fork:

```bash
$ git push
```

To create a pull request, use:

```bash
$ hub pull-request https://github.com/duckietown/docs-exercises/pull/id
```

Visit the URL to check your pull request.

### 6.4. Editing the book collection

Once you have changed your changes to the books, you need to do a pull-request for the *duckuments* repo.

1) **Forking**

As before, make sure you are on your fork:

```bash
$ cd ~/duckuments/
$ hub fork
Updating username
From ssh://github.com/duckietown/duckuments
* [new branch] master -> username/master
new remote: username
$ git branch --set-upstream-to username/master
```

2) **Updating**

For this you need to be able to know how to use submodules.

If you go to `~/duckuments`, the status will appear like this:
This means that you have changes in the submodules that you need to commit. Just like with files, you can use `git commit` to do so:

```
$ git commit -a -m "Update of repository"
$ git push
```

Now create the pull request using:

```
$ hub pull-request
```
PART B
Markduck format

This part describes the Markduck dialect that is used in the documentation.

Unit B-1 - Basic Markduck guide ................................................................. 21
Unit B-2 - Special paragraphs ................................................................. 27
Unit B-3 - LaTeX constructs ................................................................. 33
Unit B-4 - Embedding videos ................................................................. 38
Unit B-5 - move-here tag ................................................................. 39
Unit B-6 - Troubleshooting ................................................................. 40
UNIT B-1
Basic Markduck guide

The Duckiebooks are written in Markduck, a Markdown dialect. It supports many features that make it possible to create publication-worthy materials.

1.1. Markdown
The Duckiebook are written in a Markdown dialect.
   → A tutorial on Markdown.

1.2. 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.

1.3. 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 1.1.

<table>
<thead>
<tr>
<th>Table 1.1. Symbols to escape</th>
</tr>
</thead>
<tbody>
<tr>
<td>use &amp;#36;</td>
</tr>
<tr>
<td>use &amp;#96;</td>
</tr>
<tr>
<td>use &amp;#lt;</td>
</tr>
<tr>
<td>use &amp;#gt;</td>
</tr>
</tbody>
</table>
1.4. Keyboard keys
Use the \texttt{kbd} element for keystrokes.

\begin{example}
For example, to obtain:
- Press \texttt{a} then \texttt{Ctrl}-\texttt{C}.
use the following:

\begin{verbatim}
Press <kbd>a</kbd> then <kbd>Ctrl</kbd>-<kbd>C</kbd>.
\end{verbatim}
\end{example}

1.5. Figures
To create a figure, use the element \texttt{figure}:

\begin{verbatim}
<figure>
  <figcaption>Hello</figcaption>
  <img style='width:8em' src="duckietown-logo-transparent.png"/>
</figure>
\end{verbatim}

Figure 1.1. Hello

1) More attributes
Use \texttt{class="caption-left"} to have the caption show up on the left rather than on the bottom:

\begin{verbatim}
<figure class="caption-left">
  <figcaption>Hello</figcaption>
  <img style='width:8em' src="duckietown-logo-transparent.png"/>
</figure>
\end{verbatim}

Figure 1.2. Hello

1.6. Subfigures
You can create subfigures by nesting \texttt{figure} elements:

\begin{figure}
\begin{figcaption}
Main caption
\end{figcaption}
\begin{figure}
\begin{figcaption}
Hello
\end{figcaption}
\begin{img}
\texttt{width:8em}
\texttt{src=duckietown-logo-transparent.png}
\end{img}
\end{figure}
\begin{figure}
\begin{figcaption}
second
\end{figcaption}
\begin{img}
\texttt{width:8em}
\texttt{src=duckietown-logo-transparent.png}
\end{img}
\end{figure}
\end{figure}

(a) Hello

(b) second

Figure 1.3. Main caption

By default, they are displayed one per block. To make them flow horizontally, add \texttt{class=flow-subfigures} to the external figure:

\begin{figure} \texttt{class=flow-subfigures} \end{figure}

\begin{figure}
\begin{figcaption}
Main caption
\end{figcaption}
\begin{figure}
\begin{figcaption}
Hello
\end{figcaption}
\begin{img}
\texttt{width:8em}
\texttt{src=duckietown-logo-transparent.png}
\end{img}
\end{figure}
\begin{figure}
\begin{figcaption}
second
\end{figcaption}
\begin{img}
\texttt{width:8em}
\texttt{src=duckietown-logo-transparent.png}
\end{img}
\end{figure}
\end{figure}
For any element, adding an attribute called \texttt{figure-id} with value \texttt{fig:figure ID} or \texttt{tab:table ID} will create a figure that wraps the element.

1.7. Shortcut for tables

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

\begin{verbatim}
<col2 figure-id="tab:mytable" figure-caption="My table">
  <span>A</span>
  <span>B</span>
  <span>C</span>
  <span>D</span>
</col2>
\end{verbatim}

gives the following result:

\begin{table}
\centering
\caption{My table}
\begin{tabular}{ccc}
  A & B & C \\
  C & D & \\
\end{tabular}
\end{table}

1) \texttt{labels-row1} and \texttt{labels-row1}

Use the classes \texttt{labels-row1} and \texttt{labels-row1} to make pretty tables like the following. \texttt{labels-row1}: the first row is the headers. \texttt{labels-col1}: the first column is the headers.

\begin{table}
\centering
\caption{Using class="LABELS-COL1"}
\begin{tabular}{ccc}
  header A & B & C & 1 \\
  header D & E & F & 2 \\
  header G & H & I & 3 \\
\end{tabular}
\end{table}

\begin{table}
\centering
\caption{Using class="LABELS-ROW1"}
\begin{tabular}{ccc}
  header A & header B & header C \\
  D & E & F \\
  G & H & I \\
  1 & 2 & 3 \\
\end{tabular}
\end{table}
1.8. Linking to documentation

1) Establishing names of headers
You give IDs to headers using the format:

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

For example, for this subsection, we have used:

```yaml
### 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

```yaml
## 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:

```yaml
## 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

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

that’s certainly a no-no, because “localization” is too generic.

✔

```yaml
## 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 1.5).

**Table 1.5. Syntax for referring to sections.**

See `[](#establishing)`. See Subsection 1.8.1 - 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 1.8.1.

See `<a class="number_name" href="#establishing"></a>`. See Subsection 1.8.1 - Establishing names of headers.

### 1.9. Comments

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

```
# My section

<!-- this text is ignored -->

Let's start by...
```
2.1. Special paragraphs tags
The system supports parsing of some special paragraphs.

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  TOWRITE: towrite
```

To write: to write

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
6) Questions and answers

Q: question

A: answer

Answer: answer

7) Authors, maintainers, Point of Contact

Maintainer: maintainer

Assigned: AndreaCensi

Assigned to: AndreaCensi

8) References

See: see

- see

Reference: reference

- reference

Requires: requires
2.2. 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

```html
<div class='example-usage' markdown="1">
This is how you can use `rosbag`:

$ rosbag play log.bag
</div>
```

2) Check

```html
<div class='check' markdown="1">
Check that you didn't forget anything.
</div>
```
Check that you didn’t forget anything.

3) Requirements

List of requirements at the beginning of setup chapter.

2.3. 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:

A comment...

A second paragraph...

A comment...

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

A question...
Should it not be:

```
$ alternative command
```

A second paragraph...

A question...
Should it not be:

```
$ alternative command
```

A second paragraph...
UNIT B-3
Using LaTeX constructs in documentation

KNOWLEDGE AND ACTIVITY GRAPH

Requires: Working knowledge of LaTeX.

3.1. Embedded LaTeX
You can use \textit{\LaTeX} math, environment, and references. For example, take a look at

\[ x^2 = \int_0^t f(\tau) \, d\tau \]

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 3.1.

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

\[
  x^2 = \int_0^t f(\tau) \, d\tau
\]

or refer to \cite{prop:example}.

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

Listing 3.1. Use of \LaTeX\ code.

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

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

<table>
<thead>
<tr>
<th>Environment</th>
<th>Prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>definition</td>
<td>def: \textit{name}</td>
</tr>
</tbody>
</table>
Examples of all environments follow.

\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
3.2. LaTeX equations

We can refer to equations, such as \( (1) \):

\[
2a = a + a
\]

This uses align and contains \( (2) \) and \( (???) \).

\[
a = b \\
= c
\]

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 \eqref{eq: name}.

3.3. LaTeX symbols

You can place any LaTeX symbols definition in files called *.symbols.tex. These will be included as preamble.

For example, this repository contains a file a.symbols.tex containing:

\verbatim
\newcommand{\mysymbol}{\text{This is defined in a.symbols.tex}}
\end{verbatim}
So then when we create an equation with:

$$ \mysymbol $$

It gets rendered as:

This is defined in a.symbols.tex

### 3.4. Bibliography support

You need to have installed `bibtex2html`.
The system supports Bibtext 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 [1].

### 3.5. 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 in the `/docs` directory in a subfolder called `assets/svg2pdf`. You can then 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 1.5 - Figures) using `filename.pdf` as the filename in the `<img>` tag.
Figure 3.1. Embedding LaTeX in images

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

3.6. Using the HTML equivalent of Latex environments

You can create an exercise as follows:

```html
<div id="exercise:my-exercise" class="exercise">
    This is an exercise written labeled "exercise:my-exercise".
</div>

<div class="exercise">
    This is an exercise not labeled.
</div>

Referring to the exercise: [](#exercise:my-exercise) or [](#my-exercise).

**Exercise 1.** This is an exercise written labeled "exercise:my-exercise".

**Exercise 2.** This is an exercise not labeled.

Referring to the exercise: Exercise 1 or Exercise 1.
UNIT B-4
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:

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

**Example**

For example, this code:

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

produces this result:

![Duckietown](image)

Figure 4.1. 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;
- (future improvement) On the dead-tree version, a thumbnail is produced with a QR code linking to the video.
UNIT B-5

move-here tag

**Note:** This feature is deprecated, as it causes more problems than it solves.
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'/>
```
UNIT B-6
Troubleshooting

6.1. 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 'KEYMATHS000ENDKEY'` |
| 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. |

6.2. 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
```
PART C
Documentation style guide

This part describes the style guide for our documentation.

| Unit C-1 - Style guide .................................................................................................................................................. | 42 |
This chapter describes the conventions for writing the technical documentation.

1.1. Organization
The documentation is divided into books, parts (labeled ‘part:’) and units (with no CSS prefix).
To create a new part, put `{#part:name status=STATUS}` after the header, like so:

```markdown
## Safety {#part:safety status=ready}
```

1.2. 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.

  **Bad:** “The pwd is ubuntu.”

  **Better:** “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.

1.3. 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.

1.4. 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
```

Other rules:

- For a container use **container**.
- For a container on a Duckiebot use **duckiebot-container**.
• For a container on the laptop use `laptop-container`. This:

```bash
container $ command
```

will become:

```bash
$ command
```

This:

```bash
duckiebot-container $ command
```

will become:

```bash
$ command
```

This:

```bash
laptop-container $ command
```

will become:

```bash
$ command
```

### 1.5. Frequently misspelled words

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

### 1.6. Other conventions

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

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

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

is too much detail.

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

### 1.7. 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.
PART D

Backend for docs.duckietown.org

This part describes the backend that allows the website to be published at docs.duckietown.org.
UNIT D-1

*Compiling the PDF version*

Note: these are instructions that should not be needed anymore. 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.

1.1. Installing nodejs

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

Run:

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

If the version is 4 or less, remove nodejs:

```bash
$ sudo apt remove nodejs
```

Install nodejs using the instructions at this page.

Next, install the necessary Javascript libraries using npm:

```bash
$ 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.

1.2. Installing Prince

Install PrinceXML from this page.

1.3. Installing fonts

Copy the ~/duckuments/fonts directory in ~/.fonts:
```bash
$ mkdir -p ~/.fonts  # create if not exists
$ cp -R ~/duckuments/fonts ~/.fonts
```

and then rebuild the font cache using:

```bash
$ fc-cache -fv
```
Part E

Slides

Unit E-1 - Creating slides ................................................................. 50
? Making slides .................................................................................. 51
UNIT E-1
Creating slides

See this presentation: ? Making slides
Making slides

To create slides, use `type=slides` for an H1 header:

```markdown
# Making slides {#making-slides type=slides status=ready nonumber=1}
To create slides, use the attribute `type=slides` for an H1 header:
```

1.1. Next slides
Use second-level headers to make subsequent slides:

```markdown
# Making slides {#making-slides type=slides status=ready}

To create slides, use `type=slides` for an H1 header:

...

## Next slides
Use second-level headers to make subsequent slides:
```

1.2. Stepping
Use the symbol ▶ to make the corresponding fragment appear on click.

* Step 1 ▶
* Step 2 ▶
* Step 3 ▶

• Step 1 ▶
• Step 2 ▶
• Step 3 ▶

1.3. Maths
Latex still works here.
A simple test for math:

\[ a + b \geq \sqrt{c} \]

1.4. Stepping through equations
You can also step through equations:
Consider: ▶

$$a = b ▶$$

Then we get: ▶

$$c = d ▶$$

Stepping through partial parts of equations is not supported.

1.5. Sub-slides

You can have “sub slides” to make the presentation nonlinear. **Defining subslides:** Create a subslide by using header `h3`:

```markdown
### Subslide 1

This is lower.

### Subslide 2

This is even lower.
```

**Showing subslides:** Press down to show the subslides.

**Showing the slides map:** Press `ESC` again to look at the slides map.

1) Subslide 1

This is lower.

2) Subslide 2

This is even lower.

1.6. Presentation mode

Press the key `S` to enter presenters mode.
Press \( \text{ESC} \) to exit presenter mode.

1.7. Presenter notes
Use a blockquote at the end of a slide to encode the presenter notes.

```markdown
## Presenter notes

Use a blockquote at the end of a slide to encode the presenter notes:

> These are presenter notes that will appear in presenter mode.

These are presenter notes that will appear in presenter mode.
```

1.8. Under the hood
- All of this is built on top of \texttt{reveal.js}.
- Please see \texttt{reveal.js} for the complete list of features.

1.9. Figures
All other documents features work as expected.

Example of a figure:

![Figure 1.1](duckietown-logo-transparent.png)

1.10. Subfigures
Subfigures with animation:
1.11. Cross references
You can link to chapters and vice-versa: Unit E-1 - Creating slides

Link to the previous slide.

1.12. References