



The Duckietown Book

..



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

TABLE OF CONTENTS



Part 1 - The Duckietown project	12
Chapter 1 - What is Duckietown?.....	13
Section 1.1 - Goals and objectives.....	13
Section 1.2 - Learn about the Duckietown educational experience.....	13
Section 1.3 - Learn about the platform.....	13
Chapter 2 - Duckietown history and future.....	15
Section 2.1 - The beginnings of Duckietown	15
Section 2.2 - University-level classes in 2016.....	16
Section 2.3 - University-level classes in 2017.....	16
Section 2.4 - Chile.....	17
Section 2.5 - Duckietown High School.....	17
Chapter 3 - First steps.....	18
Section 3.1 - How to get started	18
Section 3.2 - Duckietown for instructors	18
Section 3.3 - Duckietown for self-guided learners	18
Section 3.4 - Introduction for companies.....	18
Section 3.5 - How to keep in touch	18
Section 3.6 - How to contribute	18
Section 3.7 - Frequently Asked Questions	18
Chapter 4 - Accounts	20
Section 4.1 - Complete list of accounts.....	20
Section 4.2 - For other contributors.....	20
Part 2 - Duckumentation documentation	21
Chapter 5 - Contributing to the documentation	22
Section 5.1 - Where the documentation is	22
Section 5.2 - Editing links.....	22
Section 5.3 - Comments	22
Section 5.4 - Installing the documentation system	22
Section 5.5 - Compiling the documentation	23
Section 5.6 - Troubleshooting compilation	24
Section 5.7 - The workflow to edit documentation	25
Section 5.8 - *Deploying the documentation	25
Section 5.9 - *Compiling the PDF version	25
Chapter 6 - Features of the documentation writing system.....	27
Section 6.1 - Markdown	27
Section 6.2 - Embedded LaTeX	27
Section 6.3 - LaTeX symbols.....	27
Section 6.4 - Variables in command lines and command output	28
Section 6.5 - Character escapes	28
Section 6.6 - Keyboard keys	28
Section 6.7 - Figures	28
Section 6.8 - Subfigures.....	29
Section 6.9 - Shortcut for tables	30
Section 6.10 - Linking to documentation from inside and outside the documentation.....	31
Section 6.11 - Embedding videos	32
Section 6.12 - Bibliography	33
Chapter 7 - Documentation style guide	34
Section 7.1 - General guidelines for technical writing.....	34
Section 7.2 - Style guide for the Duckietown documentation.....	34
Section 7.3 - Writing command lines	34
Section 7.4 - Frequently misspelled words	35
Section 7.5 - Other conventions	35

Section 7.6 - Troubleshooting sections	35
Chapter 8 - Knowledge graph	37
Section 8.1 - Formalization.....	37
Section 8.2 - Atoms properties	38
Section 8.3 - Markdown format for text-like atoms	38
Section 8.4 - How to describe the semantic graphs of atoms	39
Section 8.5 - How to describe modules	39
Part 3 - Operation manual - Duckiebot	41
Chapter 9 - Duckiebot configurations	42
Section 9.1 - Configuration list.....	42
Section 9.2 - Configuration functionality.....	42
Chapter 10 - Acquiring the parts for the Duckiebot CØ	44
Section 10.1 - Bill of materials.....	44
Section 10.2 - Chassis	45
Section 10.3 - Raspberry Pi 3 - Model B	45
Section 10.4 - Camera	47
Section 10.5 - DC Stepper Motor HAT	48
Section 10.6 - Battery	49
Section 10.7 - Standoffs, Nuts and Screws	49
Section 10.8 - Zip Tie.....	49
Section 10.9 - Configuration CØ-w.....	50
Section 10.10 - Configuration CØ-j.....	50
Section 10.11 - Configuration CØ-d.....	51
Chapter 11 - Soldering boards for CØ	52
Chapter 12 - Assembling the Duckiebot CØ	53
Chapter 13 - Reproducing the image	54
Section 13.1 - Download and uncompress the Ubuntu Mate image	54
Section 13.2 - Burn the image to an SD card	54
Section 13.3 - Raspberry Pi Config	55
Section 13.4 - Install packages	55
Section 13.5 - Install Edimax driver	56
Section 13.6 - Install ROS	56
Section 13.7 - Wireless configuration (old version)	56
Section 13.8 - Wireless configuration	57
Section 13.9 - SSH server config.....	59
Section 13.10 - Create swap Space	59
Section 13.11 - Passwordless sudo	60
Section 13.12 - Clean up	60
Section 13.13 - Ubuntu user configuration	61
Section 13.14 - Check that all required packages were installed	62
Section 13.15 - Creating the image	62
Section 13.16 - Some additions since last image to add in the next image	62
Chapter 14 - Installing Ubuntu on laptops	64
Section 14.1 - Install Ubuntu	64
Section 14.2 - Install useful software	64
Section 14.3 - Install ROS	65
Section 14.4 - Other suggested software.....	65
Section 14.5 - Installation of the duckuments system	65
Section 14.6 - Passwordless sudo	65
Section 14.7 - SSH and Git setup.....	65
Chapter 15 - Duckiebot Initialization	67
Section 15.1 - Acquire and burn the image.....	67
Section 15.2 - Turn on the Duckiebot.....	67
Section 15.3 - Connect the Duckiebot to a network	68
Section 15.4 - Ping the Duckiebot	68
Section 15.5 - SSH to the Duckiebot	68
Section 15.6 - (For D17-C1) Configure the robot-generated network	68
Section 15.7 - Setting up wireless network configuration	69
Section 15.8 - Update the system	70

Section 15.9 - Give a name to the Duckiebot.....	70
Section 15.10 - Change the hostname	70
Section 15.11 - Expand your filesystem.....	71
Section 15.12 - Create your user	72
Section 15.13 - Other customizations.....	73
Section 15.14 - Hardware check: camera	73
Chapter 16 - Software setup and RC remote control	75
Section 16.1 - Clone the Duckietown repository	75
Section 16.2 - Set up ROS environment on the Duckiebot.....	75
Section 16.3 - Add your vehicle to the scuderia file	76
Section 16.4 - Test that the joystick is detected	76
Section 16.5 - Run the joystick demo	76
Section 16.6 - The proper shutdown procedure for the Raspberry Pi	77
Chapter 17 - Reading from the camera	79
Section 17.1 - Check the camera hardware.....	79
Section 17.2 - Create two windows.....	79
Section 17.3 - First window: launch the camera nodes	79
Section 17.4 - Second window: view published topics.....	80
Chapter 18 - RC control launched remotely	82
Section 18.1 - Two ways to launch a program	82
Section 18.2 - Download and setup Software repository on the laptop	82
Section 18.3 - Edit the machines files on your laptop	82
Section 18.4 - Start the demo.....	82
Section 18.5 - Watch the program output using rqt_console.....	83
Section 18.6 - Troubleshooting.....	83
Chapter 19 - RC+camera remotely	84
Section 19.1 - Assumptions	84
Section 19.2 - Terminal setup	84
Section 19.3 - First window: launch the joystick demo	84
Section 19.4 - Second window: launch the camera nodes.....	85
Section 19.5 - Third window: view data flow	85
Section 19.6 - Fourth window: visualize the image using rviz	85
Section 19.7 - Proper shutdown procedure	85
Chapter 20 - Interlude: Ergonomics	87
Section 20.1 - set_ros_master.sh	87
Section 20.2 - SSH aliases	87
Chapter 21 - Wheel calibration	89
Chapter 22 - Camera calibration	90
Chapter 23 - Taking a log	91
Part 4 - Operation manual - Duckietowns	92
Chapter 24 - Duckietown parts	93
Section 24.1 - Bill of materials.....	93
Section 24.2 - Duckies.....	93
Section 24.3 - Floor Mats	94
Section 24.4 - Duck Tape	94
Section 24.5 - Traffic Signs.....	95
Chapter 25 - Traffic lights Parts	96
Section 25.1 - Bill of materials.....	96
Section 25.2 - Raspberry Pi	96
Chapter 26 - Duckietown Assembly	97
Chapter 27 - Traffic lights Assembly	98
Chapter 28 - The Duckietown specification	99
Section 28.1 - Topology	99
Section 28.2 - Signs placement	99
Part 5 - Operation manual - Duckiebot with LEDs.....	100
Chapter 29 - Acquiring the parts for the Duckiebot C1.....	101
Section 29.1 - Bill of materials.....	101

Section 29.2 - LEDs	102
Section 29.3 - Bumpers	103
Section 29.4 - Headers, resistors and jumper	104
Chapter 30 - Soldering boards for C1	105
Chapter 31 - Assembling the Duckiebot C1.....	106
Chapter 32 - C1 (LEDs) setup.....	107
Part 6 - Theory chapters.....	108
Chapter 33 - Chapter template.....	109
Chapter 34 - Symbols and conventions.....	110
Section 34.1 - Conventions	110
Section 34.2 - Table of symbols	110
Chapter 35 - Linear algebra.....	111
Chapter 36 - Probability basics.....	112
Chapter 37 - Dynamics	113
Chapter 38 - Autonomy overview.....	114
Section 38.1 - Perception, planning, control.....	114
Chapter 39 - Autonomy architectures.....	115
Section 39.1 - Contracts	115
Chapter 40 - Representations.....	116
Section 40.1 - Preliminaries.....	116
Section 40.2 - Robot Representations	116
Section 40.3 - Environment Representations	116
Chapter 41 - Software architectures and middlewares.....	117
Chapter 42 - Modern signal processing.....	118
Chapter 43 - Basic Kinematics	119
Chapter 44 - Basic Dynamics	120
Chapter 45 - Odometry Calibration.....	121
Chapter 46 - Computer vision basics.....	122
Chapter 47 - Illumination invariance.....	123
Chapter 48 - Line Detection	124
Chapter 49 - Feature extraction	125
Chapter 50 - Place recognition	126
Chapter 51 - Filtering 1.....	127
Chapter 52 - Filtering 2.....	128
Chapter 53 - Mission planning.....	129
Chapter 54 - Planning in discrete domains	130
Chapter 55 - Motion planning.....	131
Chapter 56 - RRT	132
Chapter 57 - Feedback control	133
Chapter 58 - PID Control.....	134
Chapter 59 - MPC Control.....	135
Chapter 60 - Object detection.....	136
Chapter 61 - Object classification	137
Chapter 62 - Object tracking	138
Chapter 63 - Reacting to obstacles.....	139
Chapter 64 - Semantic segmentation	140
Chapter 65 - Text recognition.....	141
Chapter 66 - SLAM - Problem formulation	142
Chapter 67 - SLAM - Broad categories	143
Chapter 68 - VINS	144
Chapter 69 - Advanced place recognition	145
Chapter 70 - Fleet level planning (placeholder).....	146
Chapter 71 - Fleet level planning (placeholder).....	147
Chapter 72 - Bibliography.....	148
Part 7 - Exercises	149
Chapter 73 - ROS Exercises	150
Section 73.1 - Parameters.....	150

TABLE OF CONTENTS

Section 73.2 - Running from a log	150
Section 73.3 - Unit tests	150
Section 73.4 - Analytics.....	150
Section 73.5 - Visualization	150
Chapter 74 - Line detection	151
Chapter 75 - Data processing	152
Chapter 76 - Git and conventions	153
 Part 8 - Software reference	154
Chapter 77 - Ubuntu packaging with APT	155
Section 77.1 - apt install	155
Section 77.2 - apt update	155
Section 77.3 - apt-key	155
Section 77.4 - apt-mark.....	155
Section 77.5 - add-apt-repository	155
Section 77.6 - wajig	155
Section 77.7 - dpigs	155
Chapter 78 - GNU/Linux general notions	156
Section 78.1 - Background reading	156
Chapter 79 - Every day Linux.....	157
Section 79.1 - cd	157
Section 79.2 - sudo.....	157
Section 79.3 - ls	157
Section 79.4 - cp	157
Section 79.5 - mkdir	157
Section 79.6 - touch	157
Section 79.7 - reboot	157
Section 79.8 - shutdown.....	157
Section 79.9 - rm	157
Chapter 80 - Users.....	158
Section 80.1 - passwd	158
Chapter 81 - UNIX tools	159
Section 81.1 - cat.....	159
Section 81.2 - tee.....	159
Section 81.3 - truncate.....	159
Chapter 82 - Linux disks and files	160
Section 82.1 - fdisk	160
Section 82.2 - mount	160
Section 82.3 - umount	160
Section 82.4 - losetup	160
Section 82.5 - gparted	160
Section 82.6 - dd	160
Section 82.7 - sync	160
Section 82.8 - df	160
Chapter 83 - Other administration commands	161
Section 83.1 - visudo	161
Section 83.2 - update-alternatives	161
Section 83.3 - udevadm	161
Section 83.4 - systemctl	161
Chapter 84 - Make	162
Section 84.1 - make	162
Chapter 85 - Python-related tools	163
Section 85.1 - virtualenv	163
Section 85.2 - pip.....	163
Chapter 86 - Raspberry-PI commands	164
Section 86.1 - raspi-config	164
Section 86.2 - vcgencmd.....	164
Section 86.3 - raspistill	164
Section 86.4 - jstest	164
Section 86.5 - swapon	164

Section 86.6 - <code>mkswap</code>	164
Chapter 87 - Users and permissions.	165
Section 87.1 - <code>chmod</code>	165
Section 87.2 - <code>groups</code>	165
Section 87.3 - <code>adduser</code>	165
Section 87.4 - <code>useradd</code>	165
Chapter 88 - Downloading	166
Section 88.1 - <code>curl</code>	166
Section 88.2 - <code>wget</code>	166
Section 88.3 - <code>sha256sum</code>	166
Section 88.4 - <code>xz</code>	166
Chapter 89 - Shells and environments	167
Section 89.1 - <code>source</code>	167
Section 89.2 - <code>which</code>	167
Section 89.3 - <code>export</code>	167
Chapter 90 - Other misc commands.	168
Section 90.1 - <code>pgrep</code>	168
Section 90.2 - <code>npm</code>	168
Section 90.3 - <code>node.js</code>	168
Section 90.4 - <code>ntpdate</code>	168
Section 90.5 - <code>chsh</code>	168
Section 90.6 - <code>echo</code>	168
Section 90.7 - <code>sh</code>	168
Section 90.8 - <code>fc-cache</code>	168
Chapter 91 - Linux resources usage	169
Section 91.1 - Measuring CPU usage using <code>htop</code>	169
Section 91.2 - Measuring I/O usage using <code>iostop</code>	169
Section 91.3 - How fast is the SD card?	169
Chapter 92 - SD Cards tools	170
Section 92.1 - Testing SD Card and disk speed	170
Section 92.2 - How to burn an image to an SD card	170
Section 92.3 - How to shrink an image	171
Chapter 93 - Networking tools	174
Section 93.1 - <code>hostname</code>	174
Section 93.2 - Visualizing information about the network	174
Chapter 94 - Accessing computers using SSH	175
Section 94.1 - Background reading	175
Section 94.2 - Installation of SSH	175
Section 94.3 - Local configuration	175
Section 94.4 - How to login with SSH and a password	175
Section 94.5 - Creating an SSH keypair	176
Section 94.6 - How to login without a password	177
Section 94.7 - Fixing SSH Permissions	178
Section 94.8 - <code>ssh-keygen</code>	178
Chapter 95 - Wireless networking in Linux	179
Section 95.1 - <code>iwconfig</code>	179
Section 95.2 - <code>iwlist</code>	179
Chapter 96 - Moving files between computers	181
Section 96.1 - <code>SCP</code>	181
Section 96.2 - <code>RSync</code>	181
Chapter 97 - VIM	182
Section 97.1 - External documentation	182
Section 97.2 - Installation	182
Section 97.3 - <code>vi</code>	182
Section 97.4 - Suggested configuration	182
Section 97.5 - Visual mode	182
Section 97.6 - Indenting using VIM	182
Chapter 98 - Atom	184
Chapter 99 - Eclipse	185
Section 99.1 - Installing LiClipse	185

Chapter 100 - Byobu.....	186
Section 100.1 - Advantages of using Byobu	186
Section 100.2 - Installation	186
Section 100.3 - Documentation.....	186
Section 100.4 - Quick command reference	186
Section 100.5 - Commands on OS X.....	187
Chapter 101 - Source code control with Git.....	188
Section 101.1 - Background reading	188
Section 101.2 - Installation	188
Section 101.3 - Setting up global configurations for Git	188
Section 101.4 - Git tips	188
Section 101.5 - Git troubleshooting	188
Section 101.6 - git.....	189
Chapter 102 - Git LFS	190
Section 102.1 - Generic installation instructions	190
Section 102.2 - Ubuntu 16 installation (laptop)	190
Section 102.3 - Ubuntu 16 Mate installation (Raspberry Pi 3).....	190
Chapter 103 - Setup Github access	191
Section 103.1 - Create a Github account	191
Section 103.2 - Become a member of the Duckietown organization	191
Section 103.3 - Add a public key to Github.....	191
Chapter 104 - ROS installation and reference	193
Section 104.1 - Install ROS	193
Section 104.2 - rqt_console	193
Section 104.3 - roslaunch.....	193
Section 104.4 - rviz.....	194
Section 104.5 - rostopic.....	194
Section 104.6 - catkin_make	194
Section 104.7 - rosrun	194
Section 104.8 - rostest	194
Section 104.9 - rospack	194
Section 104.10 - rosparam	194
Section 104.11 - rosdep	194
Section 104.12 - roswtf	195
Section 104.13 - rosbag	195
Section 104.14 - Troubleshooting ROS.....	195
Section 104.15 - Other materials about ROS.	195
 Part 9 - Software development guide.....	196
Chapter 105 - Python	197
Section 105.1 - Background reading	197
Section 105.2 - Python virtual environments	197
Section 105.3 - Useful libraries.....	197
Chapter 106 - Duckietown code conventions.....	198
Section 106.1 - Python	198
Chapter 107 - Configuration	199
Section 107.1 - Environment variables.....	199
Section 107.2 - The scuderia file	199
Section 107.3 - The machines file.....	200
Section 107.4 - People database.....	200
Chapter 108 - Node configuration mechanisms.....	201
Chapter 109 - Minimal ROS node - pkg_name	202
Section 109.1 - The files in the package	202
Section 109.2 - Writing a node: talker.py	203
Section 109.3 - The Talker class.....	205
Section 109.4 - Launch File	206
Section 109.5 - Testing the node	207
Section 109.6 - Documentation.....	209
Section 109.7 - Guidelines	209
Chapter 110 - ROS package verification	210

Section 110.1 - Naming	210
Section 110.2 - package.xml	210
Section 110.3 - Messages.....	210
Section 110.4 - Readme file	210
Section 110.5 - Launch files.....	210
Section 110.6 - Test files.....	210
Chapter 111 - Creating unit tests with ROS.....	211
Part 10 - Duckietown Software architecture.....	212
Chapter 112 - HW Drivers	213
Section 112.1 - Camera	213
Section 112.2 - Actuators	213
Section 112.3 - IMU	213
Chapter 113 - Low-level processing.....	214
Section 113.1 - Image acquisition	214
Section 113.2 - Anti-instagram.....	214
Section 113.3 - Line detection	214
Section 113.4 - Ground projection	214
Chapter 114 - Lane filter.....	215
Chapter 115 - Lane control.....	216
Chapter 116 - Finite state machine	217
Part 11 - Fall 2017.....	218
Chapter 117 - General remarks.....	219
Section 117.1 - The rules of Duckietown	219
Section 117.2 - Synchronization between classes.....	219
Section 117.3 - Accounts for students	219
Section 117.4 - Accounts for all instructors and TAs	220
Chapter 118 - Additional information for ETH Zürich students.....	221
Chapter 119 - Additional information for UdeM students.....	222
Chapter 120 - Additional information for TTIC students	223
Chapter 121 - Additional information for NCTU students	224
Chapter 122 - Milestone: ROS node working	225
Chapter 123 - Homework: Take and process a log.....	226
Chapter 124 - Milestone: Calibrated robot.....	227
Chapter 125 - Homework: Camera geometry.....	228
Chapter 126 - Milestone: Illumination invariance.....	229
Chapter 127 - Homework: Place recognition	230
Chapter 128 - Milestone: Lane following.....	231
Chapter 129 - Homework: localization	232
Chapter 130 - Milestone: Navigation	233
Chapter 131 - Homework: group forming	234
Chapter 132 - Milestone: Ducks in a row.....	235
Chapter 133 - Homework: Comparison of PID	236
Chapter 134 - Homework: RRT	237
Chapter 135 - Caffe tutorial	238
Chapter 136 - Milestone: Object Detection	239
Chapter 137 - Homework: Object Detection	240
Chapter 138 - Milestone: Semantic perception	241
Chapter 139 - Homework: Semantic perception	242
Chapter 140 - Milestone: Reacting to obstacles	243
Chapter 141 - Homework: Reacting to obstacles	244
Chapter 142 - Milestone: SLAM demo	245
Chapter 143 - Homework: SLAM	246
Chapter 144 - Milestone: fleet demo	247
Chapter 145 - Homework: fleet.....	248
Chapter 146 - Project proposals	249
Chapter 147 - Template of a project	250
Section 147.1 - Checklist for students	250

Section 147.2 - Checklist for TAs	250
Part 12 - Packages - Infrastructure.....	251
Chapter 148 - Package duckietown	252
Chapter 149 - Package duckietown_msgs	253
Chapter 150 - Package easy_node.....	254
Section 150.1 - Transition plan	254
Section 150.2 - YAML file format.....	254
Section 150.3 - Automatic docs generation.....	256
Chapter 151 - Package what_the_duck	257
Section 151.1 - What the duck	257
Section 151.2 - Adding more tests to what-the-duck	257
Section 151.3 - Tests already added	257
Section 151.4 - List of tests to add	258
Part 13 - Packages - Lane control.....	260
Chapter 152 - Package adafruit_drivers	261
Chapter 153 - Package anti_instagram	262
Section 153.1 - Unit tests integrated with rostest	262
Section 153.2 - Unit tests needed external files	262
Section 153.3 - Node anti_instagram_node	262
Chapter 154 - Package car_supervisor	264
Chapter 155 - Package dagu_car	265
Chapter 156 - Package ground_projection	266
Chapter 157 - Package joy_mapper	267
Section 157.1 - Testing	267
Section 157.2 - Dependencies.....	267
Section 157.3 - Node: joy_mapper.py	267
Chapter 158 - Package lane_control	269
Section 158.1 - lane_controller_node.....	269
Chapter 159 - Package lane_filter	270
Section 159.1 - lane_filter_node.....	270
Chapter 160 - Package line_detector2	273
Section 160.1 - Testing the line detector using visual inspection	273
Section 160.2 - Quantitative tests.....	275
Section 160.3 - line_detector_node2	275
Chapter 161 - Package line_detector	277
Chapter 162 - Package pi_camera	278
Part 14 - Packages - Indefinite navigation.....	279
Chapter 163 - Package fsm	280
Chapter 164 - Package indefinite_navigation.....	281
Chapter 165 - Package intersection_control	282
Chapter 166 - Package navigation	283
Chapter 167 - Package stop_line_filter	284
Part 15 - Packages - Localization and planning	285
Part 16 - Packages - Coordination.....	286
Part 17 - Packages - Additional functionality	287
Part 18 - Packages - Templates	288
Chapter 168 - Package pkg_name.....	289
Section 168.1 - Status	289
Chapter 169 - Package rostest_example	290

Part 19 - Packages - Convenience	291
Chapter 170 - Package duckietown_demos	292
Chapter 171 - Package duckietown_unit_test	293
Part 20 - Packages - To sort.....	294
Chapter 172 - Package adafruit_imu	295
Section 172.1 - Testing	295
Section 172.2 - Dependencies.....	295
Section 172.3 - Node adafruit_imu.....	295
Chapter 173 - Package apriltags_ros.....	296
Chapter 174 - Package duckie_rr_bridge	297
Chapter 175 - Package duckiebot_visualizer	298
Chapter 176 - Package duckietown_description.....	299
Chapter 177 - Package duckietown_logs	300
Chapter 178 - Package bag_stamper	301
Chapter 179 - Package kinematics	302
Chapter 180 - Package visual_odometry	303
Chapter 181 - LED emitter	304
Section 181.1 - LED detector	304
Section 181.2 - Unit tests	305
Chapter 182 - Package led_detection.....	306
Section 182.1 - Unit tests	306
Chapter 183 - Package led_emitter	307
Chapter 184 - Package led_interpreter	308
Chapter 185 - Package led_joy_mapper	309
Chapter 186 - Package rgb_led	310
Section 186.1 - Demos.....	310
Chapter 187 - Package traffic_light.....	311
Chapter 188 - Package localization	312
Chapter 189 - Package mdoap	313
Chapter 190 - Package parallel_autonomy.....	314
Chapter 191 - Package scene_segmentation.....	315
Chapter 192 - Package veh_coordinator	316
Chapter 193 - Package vehicle_detection.....	317
Chapter 194 - Package visual_odometry_line	318
Part 21 - Packages - Failed projects.....	319
Chapter 195 - Package mouse_encoder	320
Section 195.1 - Publish Topic	320
Section 195.2 - Parameters.....	320
Section 195.3 - Getting access to /dev/input/mice.....	320
Chapter 196 - Package simcity - Map Editor Version 0.1	321
Section 196.1 - How to run the map editor	321
Section 196.2 - How to edit the map.....	321
Section 196.3 - What am I looking at, anyway?.....	321
Section 196.4 - What else is there to do?	321
Chapter 197 - Package slam.....	322
Chapter 198 - Package street_name_detector	323

PART 1

The Duckietown project



CHAPTER 1

What is Duckietown?

1.1. Goals and objectives

Duckietown is a robotics educations and outreach effort.

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

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

We have two targets:

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

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

1.2. Learn about the Duckietown educational experience

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

TODO: Add Duckumentary

Figure 1. The Duckumentary

See also this documentary by Red Hat:



Figure 2. The road to autonomy

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

1.3. Learn about the platform

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

show off the capabilities of the platform.

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

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

Can you do it by night?



Figure 3. Cool Duckietown by night

CHAPTER 2

Duckietown history and future

2.1. The beginnings of Duckietown

The original Duckietown class was at MIT in 2016.



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



Figure 5. The need for autonomy



Figure 6. Advertisement



Figure 7. The elves of Duckietown

2.2. University-level classes in 2016

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

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



Figure 8. Duckietown at NCTU in 2016

2.3. University-level classes in 2017

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

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

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

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

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

2.4. Chile

TODO: to write



2.5. Duckietown High School

TODO: to write



CHAPTER 3

First steps

3.1. How to get started

If you are an instructor, please jump to [Section 3.2](#).

If you are a self-guided learner, please jump to [Section 3.3](#).

If you are a company, and interested in working with Duckietown, please jump to [Section 3.4](#).

3.2. Duckietown for instructors

TODO: to write

3.3. Duckietown for self-guided learners

TODO: to write

3.4. Introduction for companies

TODO: to write

3.5. How to keep in touch

TODO: add link to Facebook

TODO: add link to Mailing list

TODO: add link to Slack?

3.6. How to contribute

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

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

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

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

3.7. Frequently Asked Questions

1) General questions

Q: What is Duckietown?

Duckietown is a low-cost educational and research platform.

Q: Is Duckietown free to use?

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

Q: Is everything ready?

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

Q: How can I start?

See the section [First Steps](#).

Q: How can I help?

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

2) FAQ by students / independent learners

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

TODO: to write

3) FAQ by instructors

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

TODO: to write

Q: What is the budget for the robot?

TODO: to write

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

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

Q: Why the duckies?

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

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

TODO: copy usual discussion from somewhere else.

TODO: add picture of kids with Duckiebots.

CHAPTER 4

Accounts

4.1. Complete list of accounts

Currently, Duckietown has the following accounts:

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

4.2. For other contributors

If you are an international contributor:

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

PART 2

Duckumentation documentation

..

CHAPTER 5

Contributing to the documentation

5.1. Where the documentation is

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

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

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

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

5.2. Editing links

The simplest way to contribute to the documentation is to click any of the “” icons next to the headers.

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

5.3. Comments

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

5.4. Installing the documentation system

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

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

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
$ sudo apt install bibtex2html pdftk
```

2) Download the duckuments repo

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

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

3) Setup the virtual environment

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

```
$ cd ~/duckuments
```

Create the virtual environment using `virtualenv`:

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

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

Activate the virtual environment:

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

4) Setup the mcdp external repository

Make sure you are in the directory:

```
$ cd ~/duckuments
```

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

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

Install it and its dependencies:

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

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

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

```
$ pip install numpy matplotlib
```

5.5. Compiling the documentation

Check before you continue

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

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

Then:

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

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

Check before you continue

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

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

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

```
$ make clean compile
```

To see the result, open the file

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

1) Incremental compilation

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

```
$ make compile
```

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

5.6. Troubleshooting compilation



Symptom: “Invalid XML”

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

Symptom: “Tabs are evil”

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

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

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

5.7. The workflow to edit documentation.

This is the workflow:

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

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

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

5.8. *Deploying the documentation

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

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

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

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

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

5.9. *Compiling the PDF version

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

1) Installing nodejs

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

Run:

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

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

```
$ sudo apt remove nodejs
```

Install `nodejs` using the instructions at this page.

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

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

2) Troubleshooting node.js installation problems

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

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

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

3) Installing Prince

Install PrinceXML from [this page](#).

4) Installing fonts

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

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

and then rebuild the font cache using:

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

5) Compiling the PDF

To compile the PDF, use:

```
$ make compile-pdf
```

This creates the file:

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

CHAPTER 6

Features of the documentation writing system

The Duckiebook is written in a Markdown dialect. A subset of LaTeX is supported. There are also some additional features that make it possible to create publication-worthy materials.

6.1. Markdown

The Duckiebook is written in a Markdown dialect.

→ [A tutorial on Markdown.](#)

6.2. Embedded LaTeX

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

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

or refer to [Proposition 1](#).

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

The above was written as in [Listing 1](#).

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

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

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

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

Listing 1. Use of LaTeX code.

TODO: other LaTeX features supported

6.3. LaTeX symbols

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

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

6.4. Variables in command lines and command output

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

Example .

For example, to obtain:

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

Use the following:

For example, to obtain:

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

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

6.5. Character escapes

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

Other symbols to escape are shown in [Table 1](#).

TABLE 1. SYMBOLS TO ESCAPE

use <code>&#36;</code>	instead of \$
use <code>&#96;</code>	instead of `
use <code>&#lt;</code>	instead of <
use <code>&gt;</code>	instead of >

6.6. Keyboard keys

Use the `kbd` element for keystrokes.

Example .

For example, to obtain:

Press `a` then `Ctrl-C`.

use the following:

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

6.7. Figures

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

For example:

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

It will create HMTL of the form:

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

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

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

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

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

To refer to the figure, use an empty link:

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

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

6.8. Subfigures

You can also create subfigures, using the following syntax.



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

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

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

Content of first subfig

(a) Caption 1

Content of second subfig

(b) Caption 2

Figure 9. Caption of big figure

6.9. Shortcut for tables

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

The following code:

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

gives the following result:

TABLE 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 3. USING CLASS="LABELS-COL1"

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

TABLE 4. USING CLASS="LABELS-ROW1"

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

6.10. Linking to documentation from inside and outside the documentation

1) Establishing names of headers

You give IDs to headers using the format:

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

For example, for this subsection, we have used:

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

With this, we have given this header the ID “establishing”.

2) Linking from the documentation to the documentation

You can use the syntax:

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

to refer to the header.

You can also use some slightly more complex syntax that also allows to link to only the name, only the number or both (Table 5).

TABLE 5. SYNTAX FOR REFERRING TO SECTIONS.

See [](#establishing).

See Subsection 6.10.1

See .

See Establishing names of headers.

See .

See 6.10.1.

See .

See Subsection 6.10.1 - Establishing names of headers.

3) Linking to the documentation from outside the documentation

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

To do so, use links of the form:

http://purl.org/dth/topic_ID

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

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

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

When the user clicks on the link, they will be redirected to [Section 107.2](#).

6.11. Embedding videos



It is possible to embed Vimeo videos in the documentation.

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

This is the syntax:

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

For example, this code:

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

produces this result:



Figure 10. Cool Duckietown by night

Depending on the output media, the result will change:

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



6.12. Bibliography

You need to have installed `bibtex2html`.

The system supports Bibtex files.

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

Then you can refer to them using the syntax:

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

For example:

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

Will result in:

Please see [3].

CHAPTER 7

Documentation style guide

This chapter describes the conventions for writing the technical documentation.

7.1. General guidelines for technical writing

The following holds for all technical writing.

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

7.2. Style guide for the Duckietown documentation

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

7.3. Writing command lines

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

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

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

It will become:



```
$ cd ~/duckietown
```

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

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

It will become:



```
$ cd ~/duckietown
```

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

```
$ cd ~/duckietown
```

7.4. Frequently misspelled words

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

7.5. Other conventions

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

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

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

is too much detail.

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

7.6. Troubleshooting sections

Write the documentation as if every step succeeds.

Then, at the end, make a “Troubleshooting” section.

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

The following is an example of a troubleshooting section.

1) Troubleshooting

| **Symptom:** This strange thing happens.

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

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

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

CHAPTER 8

Knowledge graph

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

8.1. Formalization

1) Atoms

Definition 1. (Atom) An *atom* is a concrete resource (text, video) that is the smallest unit that is individually addressable. It is indivisible.

Each atom as a type, as follows:

```
text
  text/theory
  text/setup
  text/demo
  text/exercise
  text/reference
  text/instructor-guide
  text/quiz

video
  video/lecture
  video/instructable
  video/screencast
  video/demo
```

2) Semantic graph of atoms

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

Each node is an atom.

The graph has four different types of edges:

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

3) Modules

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

Definition 2. (Module) A *module* is a directed graph, where the nodes are either atoms or other modules, and the edges can be of the four types described in Subsection 8.1.2.

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

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

8.2. Atoms properties



Each atom has the following properties:

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

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

A version is a tuple of attributes.

The attributes are:

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

Each atom version has:

- A status value: one of draft, beta, ready, to-update (Table 6).
- A human-readable title.
- A human-readable summary (1 short paragraph).

TABLE 6. STATUS CODES

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

8.3. Markdown format for text-like atoms



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

The name of the file does not matter.

All files are encoded in UTF-8.

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

The header has the following attributes:

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

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

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

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

Listing 2. `calibration.en.md`

And this is how the Italian translation would look like:

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

Questo paragrafo sarà usato come un sommario del testo.

Listing 3. `calibration.it.md`

8.4. How to describe the semantic graphs of atoms



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

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

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

This is the first setup step.

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

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

This is the second setup step.

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

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

TABLE 7. SEMANTIC LINKS

Requires

Requires: You need to have done [](#setup-step).

Recommended

Recommended: It is better if you have setup Wifi as in [](#setup-wifi).

Reference

Reference: For more information about `rostopic`, see [](#rostopic).

See also

See also: If you are interested in feature detection, you might want to learn about [SIFT](#SIFT).

8.5. How to describe modules



TODO: Define a micro-format for this.

PART 3

Operation manual - Duckiebot



CHAPTER 9

Duckiebot configurations

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

9.1. Configuration list

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

9.2. Configuration functionality

1) C0

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

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

2) C0+w

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

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

3) C0+j

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

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

4) C0+d

In this configuration, the minimal **C0** version is augmented with a USB flash hard drive. This drive is convenient for storing videos (logs) as it provides both extra capacity and faster data transfer rates than the microSD card in the Raspberry Pi. Moreover, it is easy

to unplug it from the Duckiebot at the end of teh day and bring it over to a computer for downloading and analyzing stored data.

TODO: Insert pic of assembled Duckiebot in `c0+d` configuration.

5) C0+wjd

The upgrades of the minimal `c0` version are not mutually exclusive. We will refer to `C0+wjd` when any or all of the add-ons to the minimal version are considered.

TODO: Insert pic of assembled Duckiebot in `c0+wjd` configuration.

6) C1

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

TODO: Insert pic of assembled Duckiebot in `c1` configuration.

CHAPTER 10

Acquiring the parts for the Duckiebot C0



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

In general, keep in mind that:

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

Requires: - Cost: USD 169 + Shipping Fees (minimal configuration C0) - Time: 15 days (average shipping for cheapest choice of components)

Results: - A kit of parts ready to be assembled in a C0 or C0+wd configuration.

Next Steps: - After receiving these components, you are ready to do some [soldering](#) before [assembling](#) your C0 or C0+wd Duckiebot.

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

10.1. Bill of materials



TABLE 8. 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	USD 7.50
16 GB Class 10 MicroSD Card	USD 10
Mirco SD card reader	USD 6
Stepper Motor HAT	USD 22.50
Stacking Header	USD 2.50/piece
Battery	USD 20
16 Nylon Standoffs (M2.5 12mm F 6mm M	USD 0.05/piece
4 Nylon Hex Nuts (M2.5)	USD 0.02/piece
4 Nylon Screws (M2.5x10)	USD 0.05/piece
2 Zip Ties (300x5mm)	USD 9
Wireless Adapter (5 GHz) (C0+w)	USD 20
Joypad (C0+j)	USD 10.50
Tiny 32GB USB Flash Drive (C0+d)	USD 12.50
Total for C0 configuration	USD 159
Total for C0+w configuration	USD 179
Total for C0+j configuration	USD 169.50
Total for C0+d configuration	USD 171.50
Total for C0+wd configuration	USD 212

10.2. Chassis

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

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

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

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



Figure 11. The Magician Chassis

10.3. Raspberry Pi 3 - Model B

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



Figure 12. The Raspberry Pi 3 Model B

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

1) Power Supply

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



Figure 13. The Power Supply

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

2) Heat Sinks

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



Figure 14. The Heat Sinks

3) Class 10 MicroSD Card

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



Figure 15. The MicroSD card

4) Mirco SD card reader

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



Figure 16. The Mirco SD card reader

10.4. Camera

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



Figure 17. The Camera with Fisheye Lens

1) Camera Mount

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

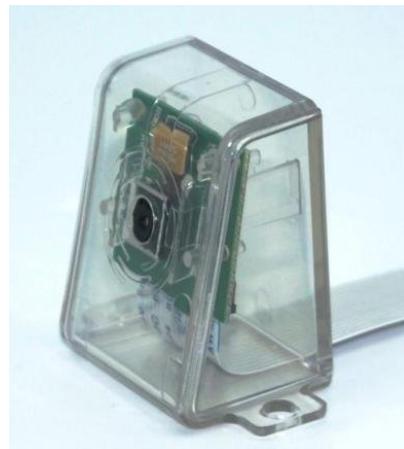


Figure 18. The Camera Mount

2) 300mm Camera Cable

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

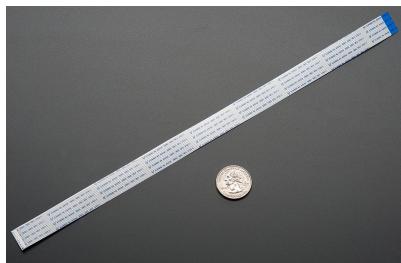


Figure 19. A 300 mm camera cable for the Raspberry Pi

10.5. DC Stepper Motor HAT

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

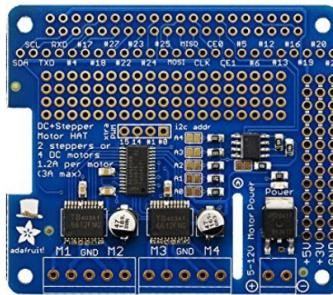


Figure 20. The Stepper Motor HAT

1) Stacking Headers

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



Figure 21. The Stacking Headers

10.6. Battery

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

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



Figure 22. The Battery

10.7. Standoffs, Nuts and Screws

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

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

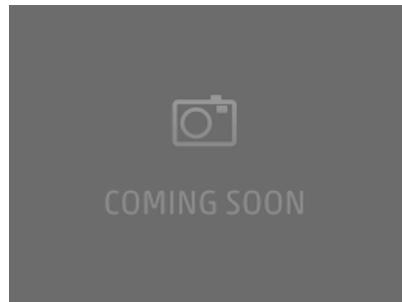


Figure 23. Standoffs, Nuts and Screws

10.8. Zip Tie

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



Figure 24. The zip ties

10.9. Configuration C0-w

1) Wireless Adapter (5 GHz)

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



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

10.10. Configuration C0-j

1) Joypad

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

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



Figure 26. A Wireless Joypad

TODO: Add figure with 2 AA batteries

10.11. Configuration C0-d

1) Tiny 32GB USB Flash Drive

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



Figure 27. The Tiny 32GB USB Flash Drive

CHAPTER 11

Soldering boards for C0



Assigned to: Shiyeng

Resources necessary:

Requires: - Duckiebot `C0+wjd` parts. The acquisition process is explained in [Chapter 10](#). The configurations are described in [Chapter 9](#).

Requires: - Time: ??? minutes

Results:

- ...

TODO: finish above

CHAPTER 12

Assembling the Duckiebot C0



Assigned to: Shiying

Requires: - Duckiebot C0+wd parts. The acquisition process is explained in Chapter 10.

- Having soldered the C0+wd parts. The soldering process is explained in Chapter 11.
- Time: about ??? minutes.

TODO: estimate time.

Results:

- An assembled Duckiebot in configuration C0+wd.

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

CHAPTER 13

Reproducing the image



Assigned to: Andrea

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

Please note that the image is already available, so you don't need to do this manually. However, this documentation might be useful if you would like to port the software to a different distribution.

Resources necessary:

Requires: Internet connection to download the packages. Requires: A PC running any Linux with an SD card reader. Requires: Time: about 20 minutes.

Results:

- A baseline Ubuntu Mate 16.04.2 image with updated software.

13.1. Download and uncompress the Ubuntu Mate image



Download the image from the page

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

The file we are looking for is:

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

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

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

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

Then decompress using the command `xz`:

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

13.2. Burn the image to an SD card



Next, burn the image on to the SD card.

- This procedure is explained in [Section 92.2](#).

1) Verify that the SD card was created correctly



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

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

2) Installation

Boot the disk in the Raspberry Pi.

Choose the following options:

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

Choose the option to log in automatically.

Reboot.

3) Update installed software

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

Afterwards I upgraded all the software preinstalled with these commands:



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

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

13.3. Raspberry Pi Config

The Raspberry Pi is not accessible by SSH by default.

Run `raspi-config`:



```
$ sudo raspi-config
```

choose “3. Interfacing Options”, and enable SSH,

We need to enable the camera and the I2C bus.

choose “3. Interfacing Options”, and enable camera, and I2C.

Also disable the graphical boot

13.4. Install packages

Install these packages.

Etckeeper:



```
$ sudo apt install etckeeper
```

Editors / shells:



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

Git:

 \$ sudo apt install -y git git-extras

Other:

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

Development:

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

Python:

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

I2C:

 \$ sudo apt install -y i2c-tools

13.5. Install Edimax driver

First, mark the kernel packages as not upgradeable:

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

Then, download and install the Edimax driver from [this repository](#).

```
$ git clone git@github.com:duckietown/rtl18822bu.git  
$ cd rtl18822bu  
$ make  
$ sudo make install
```

13.6. Install ROS

Install ROS.

→ The procedure is given in [Section 104.1](#).

13.7. Wireless configuration (old version)

This is the old version.

There are two files that are important to edit.

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

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

auto wlan0

# The loopback network interface
auto lo
iface lo inet loopback

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

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

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

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

13.8. Wireless configuration



The files that describe the network configuration are in the directory

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

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

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

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

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

[ipv4]
dns-search=
method=auto

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

This is the file

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

Contents:

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

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

[ipv4]
dns-search=
method=shared

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

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

13.9. SSH server config

This enables the SSH server:

```
$ sudo systemctl enable ssh
```

13.10. Create swap Space

Do the following:

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



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

This is for a 512 MB swap space.

Format the file for use as swap:



```
$ sudo mkswap /swap0
```

Add the swap file to the system configuration:



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

Add this line to the bottom:

```
/swap0 swap swap
```

Activate the swap space:



```
$ sudo swapon -a
```

13.11. Passwordless sudo

First, make `vi` the default editor, using

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

and then choose `vim.basic`.

Then run:

```
$ sudo visudo
```

And then change this line:

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

into this line:

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

13.12. Clean up

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

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

Either:

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

Stuff to remove:

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

At the end, remove extra dependencies:

```
$ sudo apt autoremove
```

And remove the `apt` cache using:

```
$ sudo apt clean
```

The total size should be around 6.6GB.

13.13. Ubuntu user configuration

1) Groups

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



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

: forgot to add to aug20 image:



```
$ sudo adduser ubuntu video
```

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



```
$ sudo udevadm trigger
```

2) Basic SSH config

Do the basic SSH config.

- The procedure is documented in [Section 94.3](#).

Note: this is not in the aug10 image.

3) Passwordless SSH config

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

The key is at the URL

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

Download to `.ssh/authorized_keys`:



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

4) Shell prompt

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

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

export EDITOR=vim
```

13.14. Check that all required packages were installed

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

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

13.15. Creating the image

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



`$ sudo shutdown -h now`

and put the SD card back in your laptop.

- The procedure of how to burn an image is explained in [Section 92.2](#); except you will invert the `if` and `of` destinations.

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

- The procedure of how to shrink an image is explained in [Section 92.3](#).

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

Note here the additions since the last image was created.

Create a file

`/etc/duckietown-image.yaml`

Containing these lines

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

So that we know which image is currently in used.

Install `ntpdate`:

```
$ sudo apt install ntpdate
```

Note: We should install Git LFS on the Raspberry Pi, but so far AC did not have any luck. See [Section 102.1](#).

CHAPTER 14

Installing Ubuntu on laptops



Assigned to: Andrea

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

Requires: A laptop with free disk space.

Requires: Internet connection to download the Ubuntu image.

Requires: About ??? minutes .

TODO: estimate time

Results:

- A laptop ready to be used for Duckietown.

14.1. Install Ubuntu



Install Ubuntu 16.04.2.

→ For instructions, see for example [this online tutorial](#).

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

14.2. 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

14.3. Install ROS

Install ROS on your laptop.

- The procedure is given in [Section 104.1](#).

14.4. Other suggested software

1) Redshift

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

Install redshift and run it.

 \$ sudo apt install redshift-gtk

Set to “autostart” from the icon.

14.5. Installation of the duckuments system

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

- The procedure is documented in [Section 5.4](#).

14.6. Passwordless sudo

Set up passwordless `sudo`.

- This procedure is described in [Section 13.11](#).

14.7. SSH and Git setup

1) Basic SSH config

Do the basic SSH config.

- The procedure is documented in [Section 94.3](#).

2) Create key pair for `username`

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

- The procedure is documented in [Section 94.5](#).

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

Add the public key to your Github account.

- The procedure is documented in [Section 103.3](#).

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



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

4) Local Git setup

Set up Git locally.

- The procedure is described in [Section 101.3](#).

CHAPTER 15

Duckiebot Initialization

Assigned to: Andrea

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

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

Requires: An assembled Duckiebot in configuration D17-C0. This is the result of Chapter 12.

Result:

- A Duckiebot that is ready to use.

What does it mean “ready to use”?

15.1. Acquire and burn the image

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

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

The size is 2.5 GB.

You can use:

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

Uncompress the file:

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

This will create a file of 32 GB in size.

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

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

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

Next, burn the image on disk.

- The procedure of how to burn an image is explained in Section 92.2.

15.2. 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

15.3. 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.

15.4. 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
...
```

15.5. SSH to the Duckiebot

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

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

The password is `ubuntu`.

By default, the robot boots into Byobu.

Please see [Chapter 100](#) for an introduction to Byobu.

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

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

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

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

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



```
$ iwconfig
wlan0 AABBCCDDEEFFGG unassociated Nickname:"rt18822bu"
...
lo      no wireless extensions.

enx827eb1f81a4  no wireless extensions.

wlan1    IEEE 802.11bgn ESSID:"duckietown"
...
...
```

Make note of the name `wlan0 AABBCCDDEEFFGG`.

Look up the MAC address using the command:



```
$ ifconfig wlan0 AABBCCDDEEFFGG
wlan0: 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 “`wlan0 AABBCCDDEEFFGG`”.
- 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.

15.7. Setting up wireless network configuration

This part should not be necessary anymore

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

You can add a new network by editing the file:

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

You will see a block like the following:

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

Add a new one with your SSID and password.

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

15.8. Update the system

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

Use these commands



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

15.9. Give a name to the Duckiebot

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

These are the criteria:

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

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

15.10. Change the hostname

We will put the robot name in configuration files.

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



```
$ sudo vi filename
```

Edit the file

```
/etc/hostname
```

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

Also edit the file

```
/etc/hosts
```

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

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

```
127.0.0.1 localhost
127.0.1.1 robot name
```

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

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

Then reboot the Raspberry Pi using the command

```
$ sudo reboot
```

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

```
$ hostname
robot name
```

15.11. Expand your filesystem

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



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

and then reboot



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

once rebooted you can test whether this was successful by doing



```
$ df -lh
```

the output should give you something like:

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

You should see that the Size of your `/dev/sda1` partition is “close” to the size of your SD card.

15.12. Create your user

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

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

To create a new user:

 \$ sudo useradd -m `username`

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

 \$ sudo adduser `username` sudo

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

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

Set the shell `bash`:

 \$ sudo chsh -s /bin/bash andrea

To set a password, use:

 \$ sudo passwd `username`

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

 \$ ssh `username@robot_name`

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

1) Basic SSH config

Do the basic SSH config.

- The procedure is documented in [Section 94.3](#).

2) Create key pair for `username`

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

- The procedure is documented in [Section 94.5](#).

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

Add the public key to your Github account.

- The procedure is documented in [Section 103.3](#).

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



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

4) Local Git configuration

- This procedure is in [Section 101.3](#).

5) Set up the laptop-Duckiebot connection

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

- The procedure is explained in [Section 94.6](#). In this case, we have: `local` = laptop, `local-user` = your local user on the laptop, `remote` = `robot name`, `remote-user` = `username`.

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



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

6) Some advice on the importance of passwordless access

In general, if you find yourself:

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

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

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

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

15.13. Other customizations

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

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

15.14. Hardware check: camera

Check that the camera is connected using this command:



```
$ v4lencmd get_camera  
supported=1 detected=1
```

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

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

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

 \$ raspistill -t 1 -o out.jpg

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

- For instructions on how to use `scp`, see [Subsection 96.1.1](#).

1) Troubleshooting



Symptom: `detected=0`

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

If you see an error that starts like this:

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

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

CHAPTER 16

Software setup and RC remote control

Assigned to: Andrea

Requires: Laptop configured, according to [Chapter 14](#).

Requires: You have configured the Duckiebot. The procedure is documented in [Chapter 15](#).

Requires: You have created a Github account and configured public keys, both for the laptop and for the Duckiebot. The procedure is documented in [Chapter 103](#).

Results:

- You can run the joystick demo.

16.1. Clone the Duckietown repository

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



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

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

It should not ask for a password.

1) Troubleshooting

Symptom: It asks for a password.

Resolution: You missed some of the steps described in [Chapter 103](#).

Symptom: Other weird errors.

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

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

16.2. Set up ROS environment on the Duckiebot

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



```
$ cd ~/duckietown
```

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



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

Then, build the workspace using:



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

* For more information about `catkin_make`, see [Section 104.6](#).

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

16.3. Add your vehicle to the scuderia file

Add your vehicle to the scuderia file.

→ See [Section 107.2](#).

16.4. Test that the joystick is detected

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

To make sure that the joystick is detected, run:

 \$ ls /dev/input/

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

Check before you continue

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

 \$ groups
username sudo input i2c

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

→ Consult again [Section 15.12](#).

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

 \$ jstest /dev/input/js0

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

16.5. Run the joystick demo

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

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

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

Now make sure the motor shield is connected.

Run the command:



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

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

This is the expected result of the commands:

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

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

Is all of the above valid with the new joystick?

Close the program using **Ctrl-C**.

1) Troubleshooting



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

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

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

Symptom: The left joystick does not work.

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

Symptom: The robot does not move at all.

Resolution: The cables are disconnected.

Resolution: The program assumes that the joystick is at `/dev/input/js0`. In doubt, see [Section 16.4](#).



16.6. The proper shutdown procedure for the Raspberry Pi

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

To completely shutdown the robot, issue the following command:



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

Then wait 30 seconds.

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

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

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

CHAPTER 17

Reading from the camera

Requires: You have configured the Duckiebot. The procedure is documented in [Chapter 15](#).

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

TODO: put reference

Results:

- You know that the camera works under ROS.

17.1. Check the camera hardware

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

- The procedure is documented in [Section 15.14](#).

17.2. Create two windows

On the laptop, create two Byobu windows.

- A quick reference about Byobu commands is in [Chapter 100](#).

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.

17.3. First window: launch the camera nodes

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

Activate ROS:

 \$ source environment.sh

Run the launch file called `camera.launch`:

 \$ rosrun duckietown camera.launch veh:=`robot name`

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

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

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

* For more information about `roslaunch` and “launch files”, see [Section 104.3](#).

17.4. Second window: view published topics

Switch to the second window.

Activate the ROS environment:

 \$ source environment.sh

1) List topics

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

 \$ rostopic list

* For more information about `rostopic`, see [Section 104.5](#).

You should see the following topics:

```
/robot_name/camera_node/camera_info
/robot_name/camera_node/image/compressed
/robot_name/camera_node/image/raw
/rosout
/rosout_agg
```

2) Show topics frequency

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

 \$ rostopic hz /robot_name/camera_node/image/compressed

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

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

3) Show topics data

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

 \$ rostopic echo /robot_name/camera_node/image/compressed

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

If you are Neo, then this already makes sense. If you are not Neo, in [Chapter 19](#), you will learn how to visualize the image stream on the laptop using `rviz`.

use `Ctrl-C` to stop `rostopic`.

CHAPTER 18

RC control launched remotely



Assigned to: Andrea

Requires: You can run the joystick demo from the Raspberry Pi. The procedure is documented in [Chapter 16](#).

Results:

- You can run the joystick demo from your laptop.

18.1. 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

18.2. 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 16.1](#).

Then, you should build the repository.

- This procedure is documented in [Section 16.2](#).

18.3. 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 16.3](#).

18.4. Start the demo



Now you are ready to launch the joystick demo remotely.

Check before you continue

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

```
💻 $ ssh username@robot_name.local
```

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

Run this *on the laptop*:

```
💻 $ source environment.sh  
$ roslaunch duckietown joystick.launch veh:=robot_name
```

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

* For more information about `roslaunch`, see [Section 104.3](#).

18.5. Watch the program output using `rqt_console`

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

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

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

```
💻 $ export ROS_MASTER_URI=http://robot_name.local:11311/  
$ rqt_console
```

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

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

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

* For more information about `rqt_console`, see [Section 104.2](#).

18.6. Troubleshooting

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

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

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

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

→ The procedure is documented in [Section 94.3](#).

CHAPTER 19

RC+camera remotely



Assigned to: Andrea

Requires: You can run the joystick demo remotely. The procedure is documented in [Chapter 18](#).

Requires: You can read the camera data from ROS. The procedure is documented in [Chapter 17](#).

Requires: You know how to get around in Byobu. You can find the Byobu tutorial in [Chapter 100](#).

Results:

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

19.1. Assumptions



We are assuming that the joystick demo in [Chapter 18](#) worked.

We are assuming that the procedure in [Chapter 17](#) succeeded.

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

19.2. Terminal setup



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

- A quick reference about Byobu commands is in [Chapter 100](#).

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

19.3. First window: launch the joystick demo



In the first window, launch the joystick remotely using the same procedure in [Section 18.4](#).



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

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

19.4. 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  
$ rosrun duckietown camera.launch veh:=robot name
```

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

19.5. 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
```

19.6. Fourth window: visualize the image using rviz

Launch `rviz` by using these commands:

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

* For more information about `rviz`, see [Section 104.4](#).

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

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

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

19.7. Proper shutdown procedure

To stop the nodes: You can stop the node by pressing `Ctrl-C` on the terminal where

`roslaunch` was executed. In this case, you can use `Ctrl-C` in the terminal where you launched the `camera.launch`.

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

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

CHAPTER 20

Interlude: Ergonomics



Assigned to: Andrea

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

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

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

Results:

- You will know about some useful shortcuts.

20.1. set_ros_master.sh



Instead of using:

```
$ export ROS_MASTER_URI=http://robot_name.local:11311/
```

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

```
$ source set_ros_master.sh robot_name
```

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

20.2. SSH aliases



Instead of using

```
$ ssh username@robot_name.local
```

You can set up SSH so that you can use:

```
$ ssh my-robot
```

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

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

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

Note that you **cannot** do

```
$ ping my-robot
```

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

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

scp.

CHAPTER 21

Wheel calibration

..

| Assigned to: Andrea

CHAPTER 22

Camera calibration



CHAPTER 23

Taking a log

..

| Assigned to: Andrea

PART 4
Operation manual - Duckietowns

•

CHAPTER 24

Duckietown parts



Duckietowns are the cities where Duckiebots drive. Here, we provide a link to all bits and pieces that are needed to build a Duckietown, along with their price tag. Note that while the topography of the map is highly customizable, we recommend using the components listed below. Before purchasing components for a Duckietown, read [Chapter 28](#) to understand how Duckietowns are built.

In general, keep in mind that:

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

Requires: Cost (per m^2): USD ?? + Shipping Fees Requires: Time: ?? days (average shipping time)

Results: A kit of parts ready to be assembled in a Duckietown.

Next Steps: [Assemblying](#) a Duckietown.

TODO: Add “Tools” section?

24.1. Bill of materials



TABLE 9. BILL OF MATERIALS FOR DUCKIETOWN

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

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



24.2. Duckies

Duckies ([Figure 28](#)) are essential yet non functional.

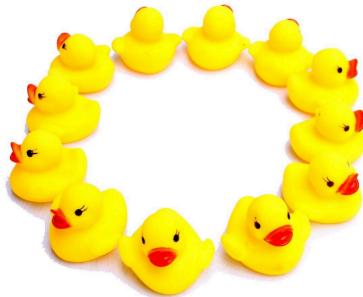


Figure 28. The Duckies

24.3. Floor Mats

The floor mats ([Figure 29](#)) are the ground on which the Duckiebots drive.

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



Figure 29. The Floor Mats

Each mat can be a segment of road: straight, a curve, or an intersection (3, or 4 way). To design your Duckietown, see [Chapter 28](#).

24.4. Duck Tape

We use duck (duct) tape of different colors ([Figure 30](#)) for defining the roads and their signals. White indicates the road boundaries, yellow determines lane boundaries and red are stop signs.

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



Figure 30. The Duck Tapes

To verify how much tape you need for each road segment type, see [Chapter 28](#).

24.5. Traffic Signs

Traffic signs ([Figure 31](#)) inform Duckiebots on the map of Duckietown, allowing them to make driving decisions.



Figure 31. The Signs

Depending on the chose road topography, the number of necessary road signal will vary. To design your Duckietown, see [Chapter 28](#).

CHAPTER 25

Traffic lights Parts



Traffic lights regulate intersections in Duckietown. Here, we provide a link to all bits and pieces that are needed to build a traffic light, along with their price tag. You will need one traffic per either three, or four way intersections. The components listed below meet the appearance specifications described in [Chapter 28](#).

In general, keep in mind that:

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

| Requires: - Cost: USD ?? + Shipping Fees - Time: ?? days (average shipping time)

Results: - A kit of parts ready to be assembled in a traffic light.

Next Steps: - [Assembling](#) a traffic light.

TODO: Estimate time and costs



25.1. Bill of materials

TABLE 10. BILL OF MATERIALS FOR TRAFFIC LIGHT

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

TODO: Complete table



25.2. Raspberry Pi

([Figure 32](#)) are essential yet non functional.

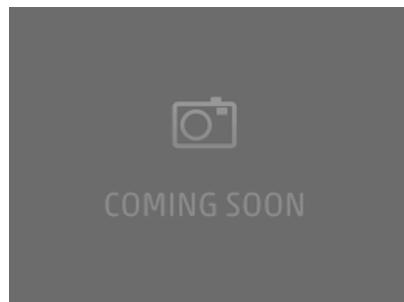


Figure 32. The placeholder

CHAPTER 26

Duckietown Assembly

..

| Assigned to: Shiying

CHAPTER 27

Traffic lights Assembly



| Assigned to: Shiying

CHAPTER 28

The Duckietown specification

| Assigned to: Liam?

28.1. Topology

1) Topology constraints

28.2. Signs placement

PART 5

Operation manual - Duckiebot with LEDs



CHAPTER 29

Acquiring the parts for the Duckiebot C1



Upgrading your `c0+wjd` configuration to `C1` starts here, with purchasing the necessary components. We provide a link to all bits and pieces that are needed to build a `C1` Duckiebot, along with their price tag. If you are wondering what is the difference between different Duckiebot configurations, read [Chapter 9](#).

In general, keep in mind that:

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

Requires: - A Duckiebot in `c0+wjd` configuration. - Cost: USD 69 + Bumpers manufacturing solution - Time: 21 Days (LED board manufacturing and shipping time)

Results: - A kit of parts ready to be assembled in a `C1` configuration Duckiebot.

Next Steps: - After receiving these components, you are ready to do some [soldering](#) before [assembling](#) your `C1` Duckiebot.



29.1. Bill of materials

TABLE 11. BILL OF MATERIALS

20 Female-Female Jumper Wires (300mm)	USD 8
Male-Male Jumper Wire (150mm)	USD 1.95
LEDs	USD 10
LED HAT	USD 28.20 for 3 pieces
PWM/Servo HAT	USD 17.50
Bumpers	TBD (custom made)
40 pin female header	USD 1.50
5 4 pin female header	USD 0.60/piece
2 16 pin male header	USD 0.61/piece
12 pin male header	USD 0.48/piece
3 pin male header	USD 0.10/piece
2 pin female shunt jumper	USD 2/piece
5 200 Ohm resistors	USD 0.10/piece
10 130 Ohm resistors	USD 0.10/piece
Total for <code>c0+wjd</code> configuration	USD 212
Total for <code>C1</code> components	USD 69 + Bumpers
Total for <code>C1</code> configuration	USD 281+Bumpers

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

29.2. LEDs

The Duckiebot is equipped with 5 RGB LEDs (Figure 33). 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.

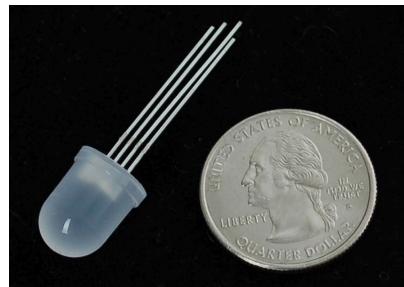


Figure 33. The RGB LEDs

1) LED HAT

The LED HAT (Figure 34) 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.

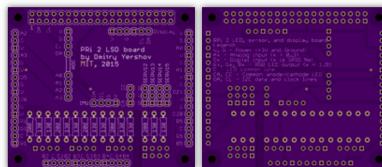


Figure 34. The LED HAT

2) PWM/Servo HAT

The PWM/Servo HAT (Figure 35) mates to the LED HAT and provides the signals to control the LEDs, without taking computational resources away from the Raspberry Pi itself. This item will require [soldering](#).



Figure 35. The PWM-Servo HAT

3) Male-Male Jumper Wires

The Duckiebot needs one male-male jumper wire ([Figure 36](#)) to power the DC Stepper Motor HAT from the PWM/Servo HAT.

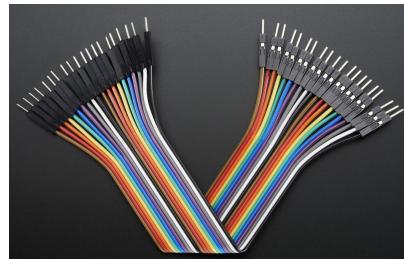


Figure 36. Premier Male-Male Jumper Wires

4) Female-Female Jumper Wires

20 Female-Female Jumper Wires ([Figure 37](#)) are necessary to connect 5 LEDs to the LED HAT.

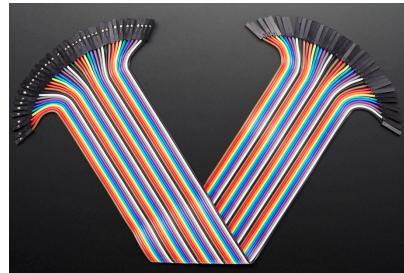


Figure 37. Premier Female-Female Jumper Wires

29.3. Bumpers

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

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

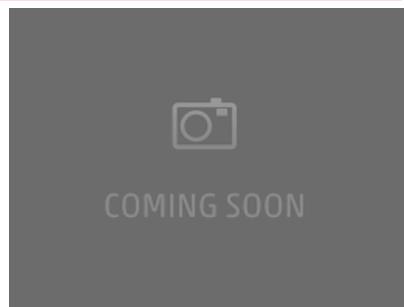


Figure 38. The Bumpers

29.4. Headers, resistors and jumper



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

These items require [soldering](#).

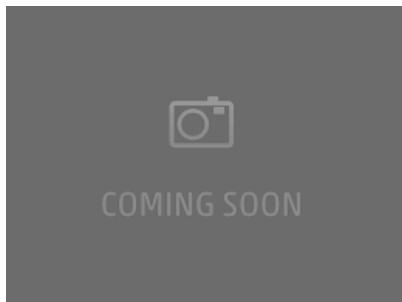


Figure 39. The Headers

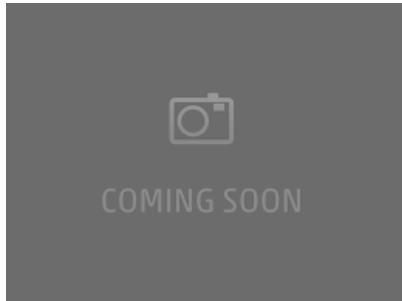


Figure 40. The Resistors

TODO: Missing figures.

CHAPTER 30

Soldering boards for C1



Assigned to: Shiying

Resources necessary:

Requires: - Duckiebot C1 parts. The acquisition process is explained in [Chapter 29](#).
The configurations are described in [Chapter 9](#).

Requires: - Time: ??? minutes

Results:

- ...

TODO: finish above

CHAPTER 31

Assembling the Duckiebot C1



Assigned to: Shiyiing

Requires: Duckiebot c1 parts. The acquisition process is explained in [Chapter 29](#).

Requires: Soldering c1 parts. The soldering process is explained in [Chapter 30](#).

Requires: Time: about ??? minutes.

TODO: estimate time.

Results:

- An assembled Duckiebot in configuration c1.

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

CHAPTER 32

C1 (LEDs) setup

..

| Assigned to: Andrea

PART 6

Theory chapters

A small cluster of three blue decorative dots located in the top right corner of the slide.

These are the theory chapters.

CHAPTER 33

Chapter template

..

| Assigned to: Jacopo

CHAPTER 34

Symbols and conventions

Assigned to: Andrea

34.1. Conventions

If \mathbf{x} is a function of time, use \mathbf{x}_t rather than $\mathbf{x}(t)$.

- * Consider the function $\mathbf{x}(t)$.
- ✓ Consider the function \mathbf{x}_t .

34.2. Table of symbols

Here are some useful symbols.

TABLE 12. SPACES

command	result	
<code>\SOthree</code>	SO(3)	Rotation matrices
<code>\SEthree</code>	SE(3)	Euclidean group
<code>\SEtwo</code>	SE(2)	Euclidean group
<code>\setwo</code>	se(2)	Euclidean group algebra

States and poses:

TABLE 13. POSES AND STATES

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

CHAPTER 35

Linear algebra

..

| Assigned to: Jacopo

CHAPTER 36

Probability basics



| Assigned to: Liam?

CHAPTER 37

Dynamics



| Assigned to: Jacopo

CHAPTER 38

Autonomy overview



| Assigned to: Liam

38.1. Perception, planning, control



CHAPTER 39

Autonomy architectures



| Assigned to: Andrea



39.1. Contracts

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

CHAPTER 40

Representations



Assigned to: Matt

Discuss:

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

40.1. Preliminaries



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

Discuss basics associated with

- Coordinate systems
- Reference frames
- Transformations

40.2. Robot Representations



Define the notion of:

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

Discuss specific robot state representation for Duckietown.

40.3. Environment Representations



Discuss:

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

1) Duckietown Environment Representation



Discuss specific environment representation for Duckietown.

CHAPTER 41

Software architectures and middlewares



| Assigned to: Andrea

CHAPTER 42

Modern signal processing



| Assigned to: Andrea

CHAPTER 43

Basic Kinematics



| Assigned to: Jacopo

CHAPTER 44

Basic Dynamics



| Assigned to: Jacopo

CHAPTER 45

Odometry Calibration

..

| Assigned to: Jacopo

CHAPTER 46

Computer vision basics



| Assigned to: Matt

CHAPTER 47

Illumination invariance

..

| Assigned to: Matt

CHAPTER 48

Line Detection



| Assigned to: Matt

CHAPTER 49

Feature extraction

..

| Assigned to: Matt

CHAPTER 50

Place recognition



| Assigned to: Matt

CHAPTER 51

Filtering 1

..

| Assigned to: Liam

CHAPTER 52

Filtering 2

..

| Assigned to: Liam

CHAPTER 53

Mission planning

| Assigned to: ETH

CHAPTER 54

Planning in discrete domains



| Assigned to: ETH

CHAPTER 55

Motion planning

..

| Assigned to: ETH

CHAPTER 56

RRT

• •

| Assigned to: ETH

CHAPTER 57

Feedback control

..

| Assigned to: Jacopo

CHAPTER 58

PID Control



| Assigned to: Jacopo

CHAPTER 59

MPC Control



| Assigned to: Jacopo

CHAPTER 60

Object detection



| Assigned to: Nick and David

CHAPTER 61

Object classification

..

| Assigned to: Nick and David

CHAPTER 62

Object tracking



| Assigned to: Nick and David

CHAPTER 63

Reacting to obstacles



| Assigned to: Jacopo

CHAPTER 64

Semantic segmentation



| Assigned to: Nick and David

CHAPTER 65

Text recognition



| Assigned to: Nick

CHAPTER 66

SLAM - Problem formulation



| Assigned to: Liam

CHAPTER 67

SLAM - Broad categories

..

| Assigned to: Liam

CHAPTER 68
VINS

..

| Assigned to: Liam

CHAPTER 69

Advanced place recognition

| Assigned to: Liam

CHAPTER 70

Fleet level planning (placeholder)



| Assigned to: ETH

CHAPTER 71

Fleet level planning (placeholder)

..

| Assigned to: ETH

CHAPTER 72

Bibliography



- [1] Jacopo Tani, Liam Paull, Maria Zuber, Daniela Rus, Jonathan How, John Leonard, and Andrea Censi. **Duckietown: an innovative way to teach autonomy.** In *EduRobotics 2016*. Athens, Greece, December 2016. pdf
- [2] Liam Paull, Jacopo Tani, Heejin Ahn, Javier Alonso-Mora, Luca Carlone, Michal Cap, Yu Fan Chen, Changhyun Choi, Jeff Dusek, Daniel Hoechener, Shih-Yuan Liu, Michael Novitzky, Igor Franzoni Okuyama, Jason Pazis, Guy Rosman, Valerio Varricchio, Hsueh-Cheng Wang, Dmitry Yershov, Hang Zhao, Michael Benjamin, Christopher Carr, Maria Zuber, Sertac Karaman, Emilio Frazzoli, Domitilla Del Vecchio, Daniela Rus, Jonathan How, John Leonard, and Andrea Censi. **Duckietown: an open, inexpensive and flexible platform for autonomy education and research.** In *IEEE International Conference on Robotics and Automation (ICRA)*. Singapore, May 2017. pdf
- [3] Bruno Siciliano and Oussama Khatib. *Springer Handbook of Robotics*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.
- [4] Tosini, G., Ferguson, I., Tsubota, K. *Effects of blue light on the circadian system and eye physiology*. Molecular Vision, 22, 61–72, 2016 (online).

PART 7

Exercises

..

These are the exercises.

CHAPTER 73

ROS Exercises



Progression of ROS skills:

73.1. Parameters



- Reading parameters
- Dynamic modification of parameters

73.2. Running from a log



- Running from a log

73.3. Unit tests



- Unit tests
- Integration with ROS tests

73.4. Analytics



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

73.5. Visualization



- Making a plot displayed using topic
- Using ROS parameters

CHAPTER 74

Line detection

..

CHAPTER 75

Data processing



CHAPTER 76

Git and conventions

..

PART 8

Software reference



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

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

CHAPTER 77

Ubuntu packaging with APT

77.1. apt install

TODO: to write

77.2. apt update

TODO: to write

1) apt dist-upgrade

TODO: hold back packages

77.3. apt-key

TODO: to write

77.4. apt-mark

TODO: to write

77.5. add-apt-repository

TODO: to write

77.6. wajig

TODO: to write

77.7. dpigs

TODO: to write

CHAPTER 78

GNU/Linux general notions

Assigned to: Andrea

78.1. Background reading

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

CHAPTER 79

Every day Linux

79.1. cd

TODO: to write

79.2. sudo

TODO: to write

79.3. ls

TODO: to write

79.4. cp

TODO: to write

79.5. mkdir

TODO: to write

79.6. touch

TODO: to write

79.7. reboot

TODO: to write

79.8. shutdown

TODO: to write

79.9. rm

TODO: to write

CHAPTER 80

Users

80.1. passwd

TODO: to write

CHAPTER 81

UNIX tools

..

81.1. cat

..

TODO: to write

..

81.2. tee

..

TODO: to write

..

81.3. truncate

..

TODO: to write

..

CHAPTER 82

Linux disks and files

82.1. `fdisk`

TODO: to write



82.2. `mount`

TODO: to write



82.3. `umount`

TODO: to write



82.4. `losetup`

TODO: to write

82.6. `dd`

TODO: to write



82.7. `sync`

TODO: to write



82.8. `df`

TODO: to write

CHAPTER 83

Other administration commands

83.1. visudo

TODO: to write

83.2. update-alternatives

TODO: to write

83.3. udevadm

TODO: to write

83.4. systemctl

TODO: to write

CHAPTER 84

Make

84.1. make

TODO: to write

CHAPTER 85

Python-related tools

85.1. `virtualenv`

TODO: to write

85.2. `pip`

TODO: to write

CHAPTER 86

Raspberry-PI commands

86.1. raspi-config

TODO: to write

86.2. vcgencmd

TODO: to write

86.3. raspistill

TODO: to write

86.4. jstest

TODO: to write

86.5. swapon

TODO: to write

86.6. mkswap

TODO: to write

CHAPTER 87

Users and permissions

87.1. chmod

TODO: to write



87.2. groups

TODO: to write



87.3. adduser

TODO: to write



87.4. useradd

TODO: to write

CHAPTER 88

Downloading

88.1. curl

TODO: to write

88.2. wget

TODO: to write

88.3. sha256sum

TODO: to write

88.4. xz

TODO: to write

CHAPTER 89

Shells and environments

89.1. source

TODO: to write



89.2. which

TODO: to write



89.3. export

TODO: to write



CHAPTER 90

Other misc commands

90.1. pgrep

TODO: to write

90.2. npm

TODO: to write

90.3. nodejs

TODO: to write

90.4. ntpdate

TODO: to write

90.5. chsh

TODO: to write

90.6. echo

TODO: to write

90.7. sh

TODO: to write

90.8. fc-cache

TODO: to write

CHAPTER 91

Linux resources usage

91.1. Measuring CPU usage using htop

You can use `htop` to monitor CPU usage.

```
$ sudo apt install htop
```

TODO: to write

91.2. Measuring I/O usage using iotop

Install using:

```
$ sudo apt install iotop
```

TODO: to write

91.3. How fast is the SD card?

→ [Section 92.1.](#)

CHAPTER 92

SD Cards tools

92.1. Testing SD Card and disk speed

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

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

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

Example results:

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

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

92.2. How to burn an image to an SD card

Requires:

- A blank SD card.
- An image file to burn.
- An Ubuntu computer with an SD reader.

Results:

- A burned image.

1) Finding your device name for the SD card

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

Insert the SD Card in the slot.

Run the command:

```
$ sudo fdisk -l
```

Find your device name, by looking at the sizes.

For example, the output might contain:

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

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

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

2) Unmount partitions

Before proceeding, unmount all partitions.

Run `df -h`. If there are partitions like `/dev/mmcblk0p1`, then unmount each of them. For example:

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

3) Burn the image

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

Let the image file be `image file`.

Burn the image using the command `dd`:

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

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

92.3. How to shrink an image

Requires:

- An image file to burn.
- An Ubuntu computer.

Results:

- A shrunk image.

Note: Majority of content taken from [here](#)

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

```
 $ sudo apt install gparted
```

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

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

It should give you something like:

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

Take note of the start of the Linux partition (in our case 131072), let's call it `start`. Now

we are going to mount the Linux partition from the image:

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

and then run `gparted`:

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

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

Note: This didn't work well for me - I had to add much more than 20MB for it to work. Click the “Apply” check mark. *Before* closing the final screen click through the arrows in the dialogue box to find a line such a “`resize2fs -p /dev/loop0 1410048K`”. Take note of the new size of your partition. Let's call it `new size`.

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

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

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

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

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

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

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

Disregard the final error.

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

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

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

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

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

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

```
💻 $ xz image file
```

CHAPTER 93

Networking tools



Assigned to: Andrea

Preliminary reading:

- Basics of networking, including
 - what are IP addresses
 - what are subnets
 - how DNS works
 - how .local names work
 - ...

→ (ref to find).

TODO: to write

Make sure that you know:

93.1. hostname



TODO: to write

93.2. Visualizing information about the network



1) ping: are you there?



TODO: to write

2) ifconfig



TODO: to write

```
$ ifconfig
```

CHAPTER 94

Accessing computers using SSH

Assigned to: Andrea

94.1. Background reading

TODO: to write

- Encryption
- Public key authentication

94.2. Installation of SSH

This installs the client:

```
$ sudo apt install ssh
```

This installs the server:

TODO: to write

This enables the server:

TODO: to write

94.3. Local configuration

The SSH configuration as a client is in the file

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

Create the directory with the right permissions:

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

Then add the following lines:

```
HostKeyAlgorithms ssh-rsa
```

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

94.4. How to login with SSH and a password

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

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

1) Troubleshooting

Symptom: “Offending key error”.

If you get something like this:

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

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

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

94.5. Creating an SSH keypair

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

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

Use the convention

```
/home/username/.ssh/usernamehost name  
/home/username/.ssh/usernamehost name.pub
```

where:

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

An SSH key can be generated with the command:

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

The session output will look something like this:

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

At this point, tell it to choose this file:

```
/home/username/.ssh/usernamehost name
```

Then:

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

Press enter; you want an empty passphrase.

```
Enter same passphrase again:
```

Press enter.

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

Note that the program created two files.

The file that contains the private key is

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

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

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

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

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

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

Add a line containing

```
IdentityFile PRIVATE_KEY_FILE
```

(using the filename for the private key).

Check that the config file is correct:

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

94.6. How to login without a password

Assumptions:

- You have two computers, called “`local`” and “`remote`”, with users “`local-user`” and “`remote-user`”.
- The two computers are on the same network.

- You have created a keypair for `local-user` on `local`.
 - This procedure is described in Section 94.5.

Results:

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

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

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

Do not continue if you cannot do this successfully.

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

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

This file is in the form:

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

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

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

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

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

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

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

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

94.7. Fixing SSH Permissions

Sometimes, SSH does not work because you have the wrong permissions on some files. In doubt, these lines fix the permissions for your `.ssh` directory.

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

94.8. ssh-keygen

TODO: to write

CHAPTER 95

Wireless networking in Linux

95.1. iwconfig

TODO: to write

95.2. iwlist

1) Getting a list of WiFi networks

What wireless networks do I have around?

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

2) Do I have 5 GHz?

Does the interface support 5 GHz channels?

```
$ sudo iwlist interface freq
```

Example output:

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

Note that in this example only *some* 5Ghz channels are supported (36, 40, 44, 48, 149, 153, 157, 161, 165); for example, channel 38, 42, 50 are not supported. This means that

you need to set up the router not to use those channels.

CHAPTER 96

Moving files between computers

96.1. SCP

TODO: to write

- 1) Download a file with SCP

TODO: to write

96.2. RSync

TODO: to write

CHAPTER 97

VIM



Assigned to: Andrea

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

97.1. External documentation



→ [A VIM tutorial.](#)

97.2. Installation



Install like this:

```
$ sudo apt install vim
```

97.3. vi



TODO: to write

97.4. Suggested configuration



Suggested `~/.vimrc`:

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

97.5. Visual mode



TODO: to write

97.6. Indenting using VIM



Use the `>` command to indent.

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

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

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

Use `<` to dedent.

CHAPTER 98

Atom



TODO: to write

CHAPTER 99

Eclipse

..

TODO: to write

99.1. Installing LiClipse

..

TODO: to write

CHAPTER 100

Byobu



Assigned to: Andrea

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



100.1. Advantages of using Byobu

TODO: To write



100.2. Installation

On Ubuntu, install using:

```
$ sudo apt install byobu
```



100.3. Documentation



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



100.4. Quick command reference

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

Commands to use windows:

TABLE 14. WINDOWS

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

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

TABLE 15. COMMANDS FOR PANES

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

Other commands:

TABLE 16. OTHER

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

100.5. Commands on OS X

Scroll up and down using [**fn**][**option**][**↑**] and [**fn**][**option**][**↓**].

Highlight using [**alt**].



CHAPTER 101

Source code control with Git



Assigned to: Andrea

101.1. Background reading



TODO: to write

- Git
- GitFlow

101.2. Installation



The basic Git program is installed using

```
$ sudo apt install git
```

Additional utilities for `git` are installed using:

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

This include the `git-ignore` utility.

101.3. Setting up global configurations for Git



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

These options tell Git who you are:

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

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

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

101.4. Git tips



1) Shallow clone



You can clone without history with the command:

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

101.5. Git troubleshooting



1) Problem 1: https instead of ssh:

The symptom is:

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

Diagnosis: the `remote` is not correct.

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

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

Expectation:

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

Solution:

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

2) Problem 1: `git push` complains about upstream

The symptom is:

```
fatal: The current branch branch name has no upstream branch.
```

Solution:

```
$ git push --set-upstream origin branch name
```

101.6. `git`

TODO: to write

CHAPTER 102

Git LFS



This describes Git LFS.

102.1. Generic installation instructions



See instructions at:

<https://git-lfs.github.com/>

102.2. Ubuntu 16 installation (laptop)



Following [these instructions](#), run the following:

```
$ sudo add-apt-repository ppa:git-core/ppa
$ curl -s https://packagecloud.io/install/repositories/github/git-lfs/script.deb.sh | sudo bash
$ sudo apt update
$ sudo apt install git-lfs
```

102.3. Ubuntu 16 Mate installation (Raspberry Pi 3)



Note: unresolved issues.

The instructions above do not work.

Following [this](#), the error that appears is that golang on the Pi is 1.6 instead it should be 1.7.

1) Troubleshooting



Symptom: The binary files are not downloaded. In their place, there are short “pointer” files.

If you have installed LFS after pulling the repository and you see only the pointer files, do:

```
$ git lfs pull --all
```

CHAPTER 103

Setup Github access

Assigned to: Andrea

This chapter describes how to create a Github account and setup SSH on the robot and on the laptop.

103.1. Create a Github account

Our example account is the following:

```
Github name: greta-p  
E-mail: greta-p@duckietown.com
```

Create a Github account ([Figure 41](#)).



Figure 41

Go to your inbox and verify the email.

103.2. Become a member of the Duckietown organization

Give the administrators your account name. They will invite you.

Accept the invitation to join the organization that you will find in your email.

103.3. Add a public key to Github

You will do this procedure twice: once for the public key created on the laptop, and later with the public key created on the robot.

Requires:

- A public/private keypair already created and configured.
 - This procedure is explained in [Section 94.5](#).

Result:

- You can access Github using the key provided.

Go to settings ([Figure 42](#)).

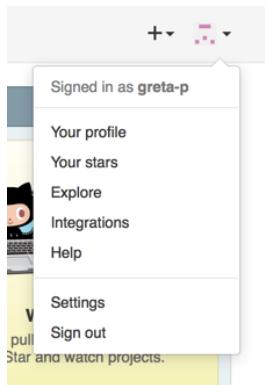


Figure 42

Add the public key that you created:



Figure 43

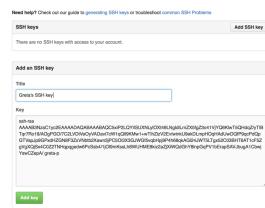


Figure 44



Figure 45

To check that all of this works, use the command

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

The command tries to connect to Github using the private keys that you specified. This is the expected output:

```
Warning: Permanently added the RSA host key for IP address 'ip address' to the list of known hosts.
```

```
Hi username! You've successfully authenticated, but GitHub does not provide shell access.
```

If you don't see the greeting, stop.

Repeat what you just did for the Duckiebot on the laptop as well, making sure to change the name of the file containing the private key.

CHAPTER 104

ROS installation and reference

Assigned to: Liam

104.1. Install ROS

This part installs ROS. You will run this twice, once on the laptop, once on the robot. The first commands are copied from [this page](#).

Tell Ubuntu where to find ROS:

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

Tell Ubuntu that you trust the ROS people (they are nice folks):

```
$ sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key  
421C365BD9FF1F717815A3895523BAEEB01FA116
```

Fetch the ROS repo:

```
$ sudo apt update
```

Now install the mega-package `ros-kinetic-desktop-full`.

```
$ sudo apt install ros-kinetic-desktop-full
```

There's more to install:

```
$ sudo apt install  
ros-kinetic-{tf-conversions,cv-bridge,image-transport,camera-info-manager,theora-image-transport,joy,image-
```

Note: Do not install packages by the name of `ros-X`, only those by the name of `ros-kinetic-X`. The packages `ros-X` are from another version of ROS.

: not done in aug20 image:

Initialize ROS:

```
$ sudo rosdep init  
$ rosdep update
```

104.2. rqt_console

TODO: to write

104.3. roslaunch

TODO: to write

104.4. `rviz`

TODO: to write

104.5. `rostopic`

TODO: to write

1) `rostopic hz`

TODO: to write

2) `rostopic echo`

TODO: to write

104.6. `catkin_make`

TODO: to write

104.7. `rosrun`

TODO: to write

104.8. `rostest`

TODO: to write

104.9. `rospack`

TODO: to write

104.10. `rosparam`

TODO: to write

104.11. `rosdep`

TODO: to write

104.12. roswhf

TODO: to write



104.13. rosbag

```
$ rosbag reindex bag file
```



104.14. Troubleshooting ROS

| **Symptom:** `computer` is not in your SSH `known_hosts` file

See [this thread](#). Remove the `known_hosts` file and make sure you have followed the instructions in [Section 94.3](#).



104.15. Other materials about ROS.



- * *A gentle introduction to ROS*

PART 9

Software development guide



This part is about how to develop software for the Duckiebot.

CHAPTER 105

Python



105.1. Background reading



- Python
- Python tutorial



105.2. Python virtual environments

Install using:

```
$ sudo apt install virtualenv
```



105.3. Useful libraries

```
matplotlib  
seaborn  
numpy  
panda  
scipy  
opencv  
...
```

CHAPTER 106

Duckietown code conventions

106.1. Python

Never use tabs in the file.

✓ checked by what-the-duck.

Indentation is 4 spaces.

Lines should be below 90 characters.

All files have an encoding declared.

Executable files start with:

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

Comments refer to the next line.

Comments, bad:

```
from std_msgs.msg import String # This is my long comment
```

Comments, better:

```
# This is my long comment
from std_msgs.msg import String
```

CHAPTER 107

Configuration

This chapter explains what are the assumptions about the configuration.

While the “Setup” parts are “imperative” (do this, do that); this is the “declarative” part, which explains what are the properties of a correct configuration (but it does not explain how to get there).

* The tool `what-the-duck` checks these conditions (Section 151.1). If you make a change from the existing conditions, make sure that it gets implemented in `what-the-duck` by filing an issue.

107.1. Environment variables

You need to have set up the variables in Table 17.

TABLE 17. ENVIRONMENT VARIABLES USED BY THE SOFTWARE

variable	reasonable value	contains
DUCKIETOWN_ROOT	<code>~/duckietown</code>	Software repository
DUCKIEFLEET_ROOT	<code>~/duckiefleet</code>	A repository that contains <code>scuderia.yaml</code> and other team-specific configuration.
DUCKIETOWN_DATA	<code>~/duckietown-data</code>	Contains data for unit tests (Dropbox folder)

1) Duckietown root directory DUCKIETOWN_ROOT

TODO: to write

2) Duckiefleet directory DUCKIEFLEET_ROOT

For Fall 2017, this is the the repository `duckiefleet-fall2017`.

For self-guided learners, this is an arbitrary repository to create.

107.2. The scuderia file

In the `${DUCKIEFLEET_ROOT}` directory, there needs to exist a file called:

`${DUCKIEFLEET_ROOT}/scuderia.yaml`

The file must contain YAML entries of the type:

```
robot-name:  
  username: username  
  owner_duckietown_id: owner duckietown ID
```

A minimal example is in Listing 4.

```
emma:  
  username: andrea  
  owner_duckietown_id: censi
```

Listing 4. Minimal scuderia file

Explanations of the fields:

- **robot_name**: the name of the robot, also equal to the host name.
- **username**: the name of the Linux user on the robot, from which to run programs.
- **owner_duckietown_id**: the owner's globally-unique Duckietown ID.

107.3. The machines file

The `machines` file is created using:

```
$ rosrun duckietown create-machines-file
```

107.4. People database

Assigned to: Andrea

TODO: Describe the people database; this is the evolution of the yaml files

1) The globally-unique Duckietown ID

This is a globally-unique ID for people in the Duckietown project.

It is equal to the Slack username.

CHAPTER 108

Node configuration mechanisms

..

TODO: Where the config files are, how they are used.

CHAPTER 109

Minimal ROS node - `pkg_name`

Assigned to: Andrea

This document outline the process of writing a ROS package and nodes in Python. To follow along, it is recommend that you duplicate the `pkg_name` folder and edit the content of the files to make your own package.

109.1. The files in the package

1) `CMakeLists.txt`

We start with `CMakeLists.txt`.

Every ROS package needs a file `CMakeLists.txt`, even if you are just using Python code in your package.

** documentation about `CMakeLists.txt`*

For a Python package, you only have to pay attention to the following parts.

The line:

```
project(pkg_name)
```

defines the name of the project.

The `find_package` lines:

```
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  duckietown_msgs # Every duckietown packages must use this.
  std_msgs
)
```

You will have to specify the packages on which your package is dependent.

In Duckietown, most packages depend on `duckietown_msgs` to make use of the customized messages.

The line:

```
catkin_python_setup()
```

tells `catkin` to setup Python-related stuff for this package.

** ROS documentation about `setup.py`*

2) `package.xml`

The file `package.xml` defines the meta data of the package.

Catkin makes use of it to flush out the dependency tree and figures out the order of compiling.

Pay attention to the following parts.

`<name>` defines the name of the package. It has to match the project name in `CMakeLists.txt`.

`<description>` describes the package concisely.

`<maintainer>` provides information of the maintainer.

`<build_depend>` and `<run_depend>`. The catkin packages this package depends on. This usually match the `find_package` in `CMakeLists.txt`.

3) `setup.py`

The file `setup.py` configures the Python modules in this package.

The part to pay attention to is

```
setup_args = generate_distutils_setup(  
    packages=['pkg_name'],  
    package_dir={'': 'include'},  
)
```

The `packages` parameter is set to a list of strings of the name of the folders inside the `include` folder.

The convention is to set the folder name the same as the package name. Here it's the `include/pkg_name` folder.

You should put ROS-independent and/or reusable module (for other packages) in the `include/pkg_name` folder.

Python files in this folder (for example, the `util.py`) will be available to scripts in the `catkin` workspace (this package and other packages too).

To use these modules from other packages, use:

```
from pkg_name.util import *
```

109.2. Writing a node: `talker.py`

Let's look at `src/talker.py` as an example.

ROS nodes are put under the `src` folder and they have to be made executable to function properly.

→ You use `chmod` for this; see [Section 87.1](#).

1) Header

Header:

```
#!/usr/bin/env python
import rospy
# Imports module. Not limited to modules in this package.
from pkg_name.util import HelloGoodbye
# Imports msg
from std_msgs.msg import String
```

The first line, `#!/usr/bin/env python`, specifies that the script is written in Python.

Every ROS node in Python must start with this line.

The line `import rospy` imports the `rospy` module necessary for all ROS nodes in Python.

The line `from pkg_name.util import HelloGoodbye` imports the class `HelloGoodbye` defined in the file `pkg_name/util.py`.

Note that you can also include modules provided by other packages, if you specify the dependency in `CMakeLists.txt` and `package.xml`.

The line `from std_msgs.msg import String` imports the `String` message defined in the `std_msgs` package.

Note that you can use `rosmsg show std_msgs/String` in a terminal to lookup the definition of `String.msg`.

2) Main

This is the main file:

```
if __name__ == '__main__':
    # Initialize the node with rospy
    rospy.init_node('talker', anonymous=False)

    # Create the NodeName object
    node = Talker()

    # Setup proper shutdown behavior
    rospy.on_shutdown(node.on_shutdown)

    # Keep it spinning to keep the node alive
    rospy.spin()
```

The line `rospy.init_node('talker', anonymous=False)` initializes a node named `talker`.

Note that this name can be overwritten by a launch file. The launch file can also push this node down namespaces. If the `anonymous` argument is set to `True` then a random string of numbers will be append to the name of the node. Usually we don't use anonymous nodes.

The line `node = Talker()` creates an instance of the `Talker` object. More details in the next section.

The line `rospy.on_shutdown(node.on_shutdown)` ensures that the `node.on_shutdown` will be called when the node is shutdown.

The line `rospy.spin()` blocks to keep the script alive. This makes sure the node stays alive and all the publication/subscriptions work correctly.

109.3. The Talker class

We now discuss the `Talker` class in `talker.py`.

1) Constructor

In the constructor, we have:

```
self.node_name = rospy.get_name()
```

saves the name of the node.

This allows to include the name of the node in printouts to make them more informative. For example:

```
rospy.loginfo("[%s] Initializing." % (self.node_name))
```

The line:

```
self.pub_topic_a = rospy.Publisher("~topic_a", String, queue_size=1)
```

defines a publisher which publishes a `String` message to the topic `~topic_a`. Note that the `~` in the name of topic under the namespace of the node. More specifically, this will actually publish to `talker/topic_a` instead of just `topic_a`. The `queue_size` is usually set to 1 on all publishers.

- For more details see [rospy overview: publisher and subscribers](#).

The line:

```
self.sub_topic_b = rospy.Subscriber("~topic_b", String, self.cbTopic)
```

defines a subscriber which expects a `String` message and subscribes to `~topic_b`. The message will be handled by the `self.cbTopic` callback function. Note that similar to the publisher, the `~` in the topic name puts the topic under the namespace of the node. In this case the subscriber actually subscribes to the topic `talker/topic_b`.

It is strongly encouraged that a node always publishes and subscribes to topics under their `node_name` namespace. In other words, always put a `~` in front of the topic names when you define a publisher or a subscriber. They can be easily remapped in a launch file. This makes the node more modular and minimizes the possibility of confusion and naming conflicts. See [the launch file section][howto-launch-file] for how remapping works.

The line

```
self.pub_timestep = self.setupParameter("~pub_timestep", 1.0)
```

Sets the value of `self.pub_timestep` to the value of the parameter `~pub_timestep`. If the parameter doesn't exist (not set in the launch file), then set it to the default value `1.0`. The `setupParameter` function also writes the final value to the parameter server. This means that you can `rosparam list` in a terminal to check the actual values of parameters being set.

The line:

```
self.timer = rospy.Timer(rospy.Duration.from_sec(self.pub_timestep), self.cbTimer)
```

defines a timer that calls the `self.cbTimer` function every `self.pub_timestep` seconds.

2) Timer callback

Contents:

```
def cbTimer(self,event):
    singer = HelloGoodbye()
    # Simulate hearing something
    msg = String()
    msg.data = singer.sing("duckietown")
    self.pub_topic_name.publish(msg)
```

Everyt ime the timer ticks, a message is generated and published.

3) Subscriber callback

Contents:

```
def cbTopic(self,msg):
    rospy.loginfo("[%(node_name)s] %(msg.data)s" %{'node_name':self.node_name,'msg.data':msg.data})
```

Every time a message is published to `~topic_b`, the `cbTopic` function is called. It simply prints the messae using `rospy.loginfo`.

109.4. Launch File

You should always write a launch file to launch a node. It also serves as a documentation on the I/O of the node.

Let's take a look at `launch/test.launch`.

```
<launch>
  <node name="talker" pkg="PKG_NAME" type="talker.py" output="screen">

    <param name="~pub_timestep" value="0.5"/>

    <remap from="~topic_b" to="~topic_a"/>
  </node>
</launch>
```

For the `<node>`, the `name` specify the name of the node, which overwrites `rospy.init_node()` in the `__main__` of `talker.py`. The `pkg` and `type` specify the package and the script of the node, in this case it's `talker.py`.

Don't forget the `.py` in the end (and remember to make the file executable through `chmod`).

The `output="screen"` direct all the `rospy.loginfo` to the screen, without this you won't see any printouts (useful when you want to suppress a node that's too talkative.)

The `<param>` can be used to set the parameters. Here we set the `~pub_timestep` to `0.5`. Note

that in this case this sets the value of `talker/pub_timestep` to `0.5`.

The `<remap>` is used to remap the topic names. In this case we are replacing `~topic_b` with `~topic_a` so that the subscriber of the node actually listens to its own publisher. Replace the line with

```
<remap from="~topic_b" to="talker/topic_a"/>
```

will have the same effect. This is redundant in this case but very useful when you want to subscribe to a topic published by another node.

109.5. Testing the node

First of all, you have to `catkin_make` the package even if it only uses Python. `catkin` makes sure that the modules in the include folder and the messages are available to the whole workspace. You can do so by

```
$ cd ${DUCKIETOWN_ROOT}/catkin_ws  
$ catkin_make
```

Ask ROS to re-index the packages so that you can auto-complete most things.

```
$ rospack profile
```

Now you can launch the node by the launch file.

```
$ rosrun pkg_name test.launch
```

You should see something like this in the terminal:

```
... logging to /home/username/.ros/log/d4db7c80-b272-11e5-8800-5c514fb7f0ed/roslaunch-robot
name-15961.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is 1GB.

started roslaunch server http://robot name.local:33925/

SUMMARY
=====

PARAMETERS
* /rosdistro: $ROS_DISTRO
* /rosversion: 1.11.16
* /talker/pub_timestep: 0.5

NODES
/
    talker (pkg_name/talker.py)

auto-starting new master
process[master]: started with pid [15973]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to d4db7c80-b272-11e5-8800-5c514fb7f0ed
process[rosout-1]: started with pid [15986]
started core service [/rosout]
process[talker-2]: started with pid [15993]
[INFO] [WallTime: 1451864197.775356] [/talker] Initialzing.
[INFO] [WallTime: 1451864197.780158] [/talker] ~pub_timestep = 0.5
[INFO] [WallTime: 1451864197.780616] [/talker] Initialzed.
[INFO] [WallTime: 1451864198.281477] [/talker] Goodbye, duckietown.
[INFO] [WallTime: 1451864198.781445] [/talker] Hello, duckietown.
[INFO] [WallTime: 1451864199.281871] [/talker] Goodbye, duckietown.
[INFO] [WallTime: 1451864199.781486] [/talker] Hello, duckietown.
[INFO] [WallTime: 1451864200.281545] [/talker] Goodbye, duckietown.
[INFO] [WallTime: 1451864200.781453] [/talker] Goodbye, duckietown.
```

Open another terminal and run:

```
$ rostopic list
```

You should see

```
/rosout
/rosout_agg
/talker/topic_a
```

In the same terminal, run:

```
$ rosparam list
```

You should see the list of parameters, including `/talker/pub_timestep`.

You can see the parameters and the values of the `talker` node with

```
$ rosparam get /talker
```

109.6. Documentation

You should document the parameters and the publish/subscribe topic names of each node in your package. The user should not have to look at the source code to figure out how to use the nodes.

109.7. Guidelines

- Make sure to put all topics (publish or subscribe) and parameters under the name-space of the node with `~`. This makes sure that the IO of the node is crystal clear.
- Always include the name of the node in the printouts.
- Always provide a launch file that includes all the parameters (using `<param>`) and topics (using `<remap>`) with each node.

CHAPTER 110

ROS package verification

This chapter describes formally what makes a conforming ROS package in the Duckietown software architecture.

110.1. Naming

- For exercises packages, the name of the package must be `package_handle`.

110.2. `package.xml`

- There is a `package.xml` file.
- Checked by `what-the-duck`.

110.3. Messages

- The messages are called

110.4. Readme file

- There is a `README.md` file
- Checked by `what-the-duck`.

110.5. Launch files

- there is the first launch file

110.6. Test files

TODO: to write

CHAPTER 111

Creating unit tests with ROS



PART 10

Duckietown Software architecture



This part describes the Duckietown algorithms/architecture, but without the software details, which have been already talked about at length in [Part 9](#).

CHAPTER 112

HW Drivers

112.1. Camera

TODO: to write

112.2. Actuators

TODO: to write

112.3. IMU

TODO: to write

CHAPTER 113

Low-level processing

113.1. Image acquisition

TODO: to write

113.2. Anti-instagram

TODO: to write

113.3. Line detection

TODO: to write

113.4. Ground projection

TODO: to write

CHAPTER 114

Lane filter

..

TODO: to write

CHAPTER 115

Lane control



TODO: to write

CHAPTER 116

Finite state machine

..

PART 11

Fall 2017



This is the first time that a class is taught jointly across 3 continents!

There are 4 universities involved in the joint teaching for the term:

- ETH Zürich (ETHZ), with instructors Emilio Frazzoli, Andrea Censi, Jacopo Tani.
- University of Montreal (UdeM), with instructor Liam Paull.
- TTI-Chicago (TTIC), with instructor Matthew Walter.
- National C T University (NCTU), with instructor Nick Wang.

This part of the Duckiebook describes all the information that is needed by the students of the four institutions.

CHAPTER 117

General remarks

Assigned to: Andrea

117.1. 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.

117.2. Synchronization between classes

At ETHZ, UdeM, TTIC, the class will be more-or-less synchronized. The materials are the same; there is some slight variation in the ordering.

Moreover, there will be some common groups for the projects.

The NCTU class is undergraduate level. Students will learn slightly simplified materials. They will not collaborate directly with the classes.

117.3. Accounts for students

To participate in Duckietown, students must use two accounts: Slack and Github.

1) Slack

You need a Slack account, for team discussion and organization.

TODO: Sign up link here:

TODO: Account naming convention

2) Github

TODO: Account naming convention

- A Github account;
- Membership in the Duckietown organization.

117.4. Accounts for all instructors and TAs



As an instructor/TA for the Fall 2017 class, in addition to the accounts above, these are two more accounts that you need.

1) Twist



Twist is used for class organization (such as TAs, logistics);

TODO:

2) Google docs



Google Docs is used to maintain TODOs and other coordination materials.

TODO: how to be authorized?

In particular:

- This is the schedule: XXX
- This is the calendar in which to annotate everything: XXX

CHAPTER 118

Additional information for ETH Zürich students

Assigned to: Andrea

This section describes information specific for ETH Zürich students.

TODO: to write

1) Website

All really important information, such as deadlines, is in the authoritative website:

2) Duckiebox distribution

TODO: to write

3) Lab access

TODO: To write

4) The local TAs

TODO: to write

CHAPTER 119

Additional information for UdeM students



| Assigned to: Liam

TODO: to write

CHAPTER 120

Additional information for TTIC students

..

| Assigned to: Matt

TODO: to write

CHAPTER 121

Additional information for NCTU students



Assigned to: Nick

TODO: to write

CHAPTER 122

Milestone: ROS node working

..

CHAPTER 123**Homework: Take and process a log**

CHAPTER 124

Milestone: Calibrated robot



CHAPTER 125

Homework: Camera geometry



CHAPTER 126

Milestone: Illumination invariance

..

CHAPTER 127**Homework: Place recognition**

CHAPTER 128

Milestone: Lane following

..

CHAPTER 129**Homework: localization**

• •

CHAPTER 130

Milestone: Navigation

..

CHAPTER 131

Homework: group forming

•

•

CHAPTER 132

Milestone: Ducks in a row



CHAPTER 133**Homework: Comparison of PID**

10

CHAPTER 134
Homework: RRT

..

CHAPTER 135
Caffe tutorial

..

CHAPTER 136

Milestone: Object Detection



CHAPTER 137
Homework: Object Detection

• •

CHAPTER 138

Milestone: Semantic perception

..

CHAPTER 139

Homework: Semantic perception



CHAPTER 140

Milestone: Reacting to obstacles

..

CHAPTER 141**Homework: Reacting to obstacles**

10

CHAPTER 142
Milestone: SLAM demo



CHAPTER 143
Homework: SLAM

• •

CHAPTER 144

Milestone: fleet demo

..

CHAPTER 145
Homework: fleet

• •

CHAPTER 146

Project proposals

..

CHAPTER 147

Template of a project

147.1. Checklist for students

- Have a Github account. See [Chapter 103](#). See name conventions (TODO).
- Be part of the Duckietown Github organization. You are sure only when you commit and push one change to one of our repositories.
- Be part of the Duckietown Slack. See name conventions (TODO).

147.2. Checklist for TAs

- Be signed up on

PART 12

Packages - Infrastructure

..

TODO: to write

CHAPTER 148

Package duckietown



TODO: to write

CHAPTER 149

Package duckietown_msgs

TODO: to write

CHAPTER 150

Package `easy_node`



`easy_node` is a framework to make it easier to create and document ROS nodes. It allows a *declarative approach* to declaring subscriptions, publishers, and parameters.

The user can directly describe what are the subscription, the publishers, the parameters in a YAML file. The framework then takes care of calling the necessary boilerplate ROS commands for subscribing, publishing, etc.

In addition, `easy_node` can also create the Markdown documentation from the YAML file.

Using `easy_node` allows to cut 40%-50% of the code required for programming a node. For an example, see the package `line_detector2`, which contains a re-implementation of `line_detector` using the new framework.

150.1. Transition plan



The plan is to first use `easy_node` just for documentation of the nodes; then, later, convert all the nodes to use it.

150.2. YAML file format



If you have a node with the name `my_node`, implemented in the file `my_node.py` you must create a file by the name `my_node.easy_node.yaml` somewhere in the package.

The YAML file must contain 4 sections, each of which is a dictionary.

This is the smallest example of an empty configuration:

```
parameters:
subscriptions:
publishers:
contracts:
```

1) Configuring parameters



This is the syntax:

```
parameters:
  name parameter:
    type: type
    desc: description
    default: default value
```

where:

- `![type]` is one of `float`, `int`, `bool`, `str`.
- `![description]` is a description that will appear in the documentation.
- The optional field `default` gives a default value for the parameter.

For example:

```
parameters:
  k_d:
    type: float
    desc: The derivative gain for $\theta$.
    default: 1.02
```

2) Describing publishers and subscriptions

The syntax for describing subscribers is:

```
subscriptions:
  name subscription:
    topic: topic name
    type: message type
    desc: description

    queue_size: queue size
    latch: latch
```

where:

- **topic name** is the name of the topic to subscribe.
- **message type** is a ROS message type name, such as `sensor_msgs/Joy`.
- **description** is a Markdown description string.
- **queue size**, **latch** are optional parameters for ROS publishing/subscribing functions.

The syntax for describing publishers is similar.

Example:

```
subscriptions:
  segment_list:
    topic: ~segment_list
    type: duckietown_msgs/SegmentList
    desc: Line detections
    queue_size: 1

publishers:
  lane_pose:
    topic: ~lane_pose
    type: duckietown_msgs/LanePose
    desc: Estimated pose
    queue_size: 1
```

3) Describing contracts

This is not implemented yet. The idea is to have a place where we can describe constraints such as:

- “This topic must publish at least at 30 Hz.”
- “Panic if you didn’t receive a message for 2 seconds.”
- “The maximum latency for this is 0.2 s”

Then, we can implement all these checks once and for all in a proper way, instead of relying on multiple broken implementations

150.3. Automatic docs generation



Generate the docs for each node using this command:

```
$ rosrun easy_node generate_docs.py
```

CHAPTER 151

Package what_the_duck

`what-the-duck` is a program that tests *dozens* of configuration inconsistencies that can happen on a Duckiebot.

151.1. What the duck

To use it, first compile the repository, and then run:

```
$ ./what-the-duck
```

151.2. Adding more tests to what-the-duck

The idea is to add to `what-the-duck` all the tests that can be automated.
The documentation about to do that is not ready yet.

151.3. Tests already added

Here is the list of tests already added:

```
✓ Camera is detected
✓ Scipy is installed
✓ sklearn is installed
✓ Date is set correctly
✓ Not running as root
✓ Not running as ubuntu
✓ Member of group sudo
✓ Member of group input
✓ Member of group video
✓ Member of group i2c
✓ ~/.ssh exists
✓ ~/.ssh permissions
✓ ~/.ssh/config exists
✓ SSH option HostKeyAlgorithms is set
✓ At least one key is configured.
✓ ~/.ssh/authorized_keys exists
✓ Git configured
✓ Git email set
✓ Git name set
✓ Git push policy set
✓ Edimax detected
✓ The hostname is configured
✓ /etc/hosts is sane
✓ Correct kernel version
✓ Messages are compiled
✓ Shell is bash
✓ Working internet connection
✓ Github configured
✓ Joystick detected
✓ Environment variable DUCKIETOWN_ROOT
✓ ${DUCKIETOWN_ROOT} exists
✓ Environment variable DUCKIETOWN_FLEET
✓ ${DUCKIETOWN_FLEET} exists
✓ ${DUCKIETOWN_FLEET}/scuderia.yaml exists
✓ ${DUCKIETOWN_FLEET}/scuderia.yaml is valid
✓ machines file is valid
✓ Wifi network configured
✓ Python: No CamelCase
✓ Python: No tab chars
✓ Python: No half merges
```

151.4. List of tests to add



Please add below any configuration test that can be automated:

- Editor is set to vim.
- Syntax on in ~/.vimrc
- They put the right MAC address in the network configuration
- Ubuntu user is in group video, input, i2c (even if run from other user.)
- There is at least X.YGB of free disk space.

- If the SD is larger than 8GB, the disk has been resized.

PART 13

Packages - Lane control



TODO: to write

CHAPTER 152

Package adafruit_drivers

TODO: to write

CHAPTER 153

Package `anti_instagram`



TODO: to write

153.1. Unit tests integrated with `rostest`



Unit tests are integrated with [`rostest`][#rostest].

To run manually, use:

```
$ rostest anti_instagram antiinstagram_correctness_test.test
$ rostest anti_instagram antiinstagram_stub_test.test
$ rostest anti_instagram antiinstagram_performance_test.test
```

153.2. Unit tests needed external files



These are other unittest that require the logs in DUCKIETOWN_DATA:

```
$ rosrun anti_instagram annotations_test.py
```

153.3. Node `anti_instagram_node`



1) Parameters

Parameter `publish_corrected_image: bool`; default value: `False`



Whether to compute and publish the corrected image.

2) Subscriptions

Subscription `image: topic ~uncorrected_image (CompressedImage)`



This is the compressed image to read.

Subscription `click: topic ~click (BoolStamped)`



Activate the calibration phase with this switch.

3) Published topics



Publisher `image: topic ~corrected_image (Image)`

The corrected image.

Publisher `health: topic ~colorSegment (AntiInstagramHealth)`

The health of the process.

Publisher `transform: topic ~transform (AntiInstagramTransform)`

The computed transform.

CHAPTER 154

Package car_supervisor



TODO: to write

CHAPTER 155

Package dagu_car

TODO: to write

CHAPTER 156

Package ground_projection



TODO: to write

CHAPTER 157

Package joy_mapper

TODO: to write

157.1. Testing

To test run:

1) connect joystick 2) `roslaunch launch/joy_mapper_test.launch` 3) the robot should move when you push buttons

157.2. Dependencies

- `rospy`
- `sensor_msgs`: for the `Joy.msg`
- `duckietown_msgs`: for the `CarControl.msg`

157.3. Node: joy_mapper.py

This node takes a `sensor_msgs/Joy.msg` and converts it to a `duckietown_msgs/CarControl.msg`.

It publishes at a fixed interval with a zero-order hold.

1) Parameters

Parameter `v_gain`: `float`; default value: `0.41`

TODO: Missing description for entry “`v_gain`”.

Parameter `omega_gain`: `float`; default value: `8.3`

TODO: Missing description for entry “`omega_gain`”.

Parameter `bicycle_kinematics`: `int`; default value: `0`

TODO: Missing description for entry “`bicycle_kinematics`”.

Parameter `simulated_vehicle_length`: `float`; default value: `0.18`

TODO: Missing description for entry “`simulated_vehicle_length`”.

Parameter `steer_angle_gain`: `int`; default value: `1`

TODO: Missing description for entry “`steer_angle_gain`”.

2) Subscriptions

Subscription `joy`: topic `joy` (`Joy`)

The `Joy.msg` from `joy_node` of the `joy` package. The vertical axis of the left stick maps to speed. The horizontal axis of the right stick maps to steering.

3) Published topics

Publisher avoidance: topic ~start_avoidance (BoolStamped)

TODO: Missing description for entry “ avoidance ”.

Publisher car_cmd: topic ~car_cmd (Twist2DStamped)

TODO: Missing description for entry “ car_cmd ”.

Publisher joy_override: topic ~joystick_override (BoolStamped)

TODO: Missing description for entry “ joy_override ”.

Publisher parallel_autonomy: topic ~parallel_autonomy (BoolStamped)

TODO: Missing description for entry “ parallel_autonomy ”.

Publisher e_stop: topic wheels_driver_node/emergency_stop (BoolStamped)

TODO: Missing description for entry “ e_stop ”.

Publisher anti_instagram: topic anti_instagram_node/click (BoolStamped)

TODO: Missing description for entry “ anti_instagram ”.

CHAPTER 158

Package lane_control

TODO: to write

158.1. lane_controller_node

Note: there is some very funny business inside. It appears that `k_d` and `k_theta` are switched around.

1) Parameters

Parameter `k_theta`: float

Proportional gain for θ .

Parameter `theta_thres`: float

Maximum desired θ .

Parameter `d_offset`: float

A configurable offset from the lane position.

Parameter `d_thres`: float

Cap for error in d .

Parameter `v_bar`: float

Nominal linear velocity (m/s).

Parameter `k_d`: float

Proportional gain for d .

2) Subscriptions

Subscription `lane_reading`: topic `~lane_pose` (`LanePose`)

TODO: Missing description for entry “`lane_reading`”.

3) Published topics

Publisher `car_cmd`: topic `~car_cmd` (`Twist2DStamped`)

TODO: Missing description for entry “`car_cmd`”.

CHAPTER 159

Package lane_filter



Assigned to: Liam

TODO: to write

159.1. lane_filter_node



1) Parameters

Parameter `peak_val`: float; default value: `10.0`

TODO: Missing description for entry “`peak_val`”.

Parameter `l_max`: float; default value: `2.0`

TODO: Missing description for entry “`l_max`”.

Parameter `lanewidth`: float; default value: `0.4`

TODO: Missing description for entry “`lanewidth`”.

Parameter `mean_d_θ`: float; default value: `0.0`

TODO: Missing description for entry “`mean_d_θ`”.

Parameter `min_max`: float; default value: `0.3`

Expressed in nats.

Parameter `d_max`: float; default value: `0.5`

TODO: Missing description for entry “`d_max`”.

Parameter `use_distance_weighting`: bool; default value: `False`

For use of distance weighting (dw) function.

Parameter `linewidth_white`: float; default value: `0.04`

TODO: Missing description for entry “`linewidth_white`”.

Parameter `cov_omega`: float; default value: `0.01`

Angular velocity “input”.

which units?

Parameter `use_max_segment_dist`: bool; default value: `False`

For use of maximum segment distance.

Parameter `delta_d`: float; default value: `0.02`

(meters)

Parameter `min_segs`: int; default value: `10`

For use of minimum segment count.

Parameter `phi_min`: float ; default value: -1.5707

TODO: Missing description for entry “`phi_min`”.

Parameter `sigma_d_θ`: float ; default value: 0.0

TODO: Missing description for entry “`sigma_d_θ`”.

Parameter `phi_max`: float ; default value: 1.5707

TODO: Missing description for entry “`phi_max`”.

Parameter `zero_val`: float ; default value: 1.0

TODO: Missing description for entry “`zero_val`”.

Parameter `delta_phi`: float ; default value: 0.0

(radians)

Parameter `l_peak`: float ; default value: 1.0

TODO: Missing description for entry “`l_peak`”.

Parameter `cov_v`: float ; default value: 0.5

Linear velocity “input”.

which units?

Parameter `sigma_phi_mask`: float ; default value: 0.05

TODO: Missing description for entry “`sigma_phi_mask`”.

Parameter `sigma_d_mask`: float ; default value: 0.05

TODO: Missing description for entry “`sigma_d_mask`”.

Parameter `mean_phi_θ`: float ; default value: 0.0

TODO: Missing description for entry “`mean_phi_θ`”.

Parameter `sigma_phi_θ`: float ; default value: 0.0

TODO: Missing description for entry “`sigma_phi_θ`”.

Parameter `d_min`: float ; default value: -0.7

TODO: Missing description for entry “`d_min`”.

Parameter `linewidth_yellow`: float ; default value: 0.02

TODO: Missing description for entry “`linewidth_yellow`”.

Parameter `use_min_segs`: bool ; default value: False

For use of minimum segment count.

Parameter `use_propagation`: bool ; default value: False

For propagation.

Parameter `max_segment_dist`: float ; default value: 1.0

For use of maximum segment distance.

2) Subscriptions



Subscription **velocity**: topic `~velocity` (`Twist2DStamped`)

TODO: Missing description for entry “`velocity`”.

Subscription **segment_list**: topic `~segment_list` (`SegmentList`)

TODO: Missing description for entry “`segment_list`”.

3) Published topics

Publisher **belief_img**: topic `~belief_img` (`Image`)

TODO: Missing description for entry “`belief_img`”.

Publisher **switch**: topic `~switch` (`BoolStamped`)

TODO: Missing description for entry “`switch`”.

Publisher **lane_pose**: topic `~lane_pose` (`LanePose`)

TODO: Missing description for entry “`lane_pose`”.

Publisher **entropy**: topic `~entropy` (`Float32`)

TODO: Missing description for entry “`entropy`”.

Publisher **in_lane**: topic `~in_lane` (`BoolStamped`)

TODO: Missing description for entry “`in_lane`”.

CHAPTER 160

Package line_detector2



This is a re-implementation of the package `line_detector` using the new facilities provided by `easy_node`.



160.1. Testing the line detector using visual inspection

The following are instructions to test the line detector from bag files.

You can run from a bag with the following:



```
$ roslaunch line_detector line_detector2_bag veh:=vehicle bagin:=bag in bagout:=bag out  
verbose:=true
```

Where:

- `bag in` is the **absolute path** of the input bag.
- `vehicle` is the name of the vehicle that took the log.
- `bag out` is the **absolute path** if the output bag.

Note: you always need to use absolute paths for bag files.

You can let this run for a few seconds, then stop using `Ctrl-C`.

You can then inspect the result using:

```
$ roscore &  
$ rosbag play -l bag out  
$ rviz &
```

In `rviz` click “add”, click “by topic” tab, expand “`line_detector`” and click “`image_with_lines`”.

Observe on the result that:

1. There are *lots* of detections.
2. Predominantly white detections (indicated in black) are on white lines, yellow detections (shown in blue) are on blue lines, and red detections (shown in green) are on red lines.

These are some sample logs on which to try:

```
$ wget -O 160122-manual1_ferrari.bag https://www.dropbox.com/s/8bpi656j7qox5kv?dl=1  
  
https://www.dropbox.com/s/vwznjke4xvnhi9o/160122_manual2-ferrari.bag?dl=1  
  
https://www.dropbox.com/s/y7ulj198punj0mp/160122_manual3_corner-ferrari.bag?dl=1  
  
https://www.dropbox.com/s/d4n9otmlans4i62/160122-calibration-good_lighting-tesla.bag?dl=1
```

Sample output:

```
From cmd:roslaunch duckietown camera.launch veh:=${VEHICLE_NAME}
[INFO] [WallTime: 1453839555.948481] [LineDetectorNode] number of white lines = 14
[INFO] [WallTime: 1453839555.949102] [LineDetectorNode] number of yellow lines = 33
[INFO] [WallTime: 1453839555.986520] [LineDetectorNode] number of white lines = 18
[INFO] [WallTime: 1453839555.987039] [LineDetectorNode] number of yellow lines = 34
[INFO] [WallTime: 1453839556.013252] [LineDetectorNode] number of white lines = 14
[INFO] [WallTime: 1453839556.013857] [LineDetectorNode] number of yellow lines = 29
[INFO] [WallTime: 1453839556.014539] [LineDetectorNode] number of red lines = 2
[INFO] [WallTime: 1453839556.047944] [LineDetectorNode] number of white lines = 18
[INFO] [WallTime: 1453839556.048672] [LineDetectorNode] number of yellow lines = 28
[INFO] [WallTime: 1453839556.049534] [LineDetectorNode] number of red lines = 2
[INFO] [WallTime: 1453839556.081400] [LineDetectorNode] number of white lines = 13
[INFO] [WallTime: 1453839556.081944] [LineDetectorNode] number of yellow lines = 34
[INFO] [WallTime: 1453839556.082479] [LineDetectorNode] number of red lines = 1
```

The output from `rviz` looks like [Figure 46](#).

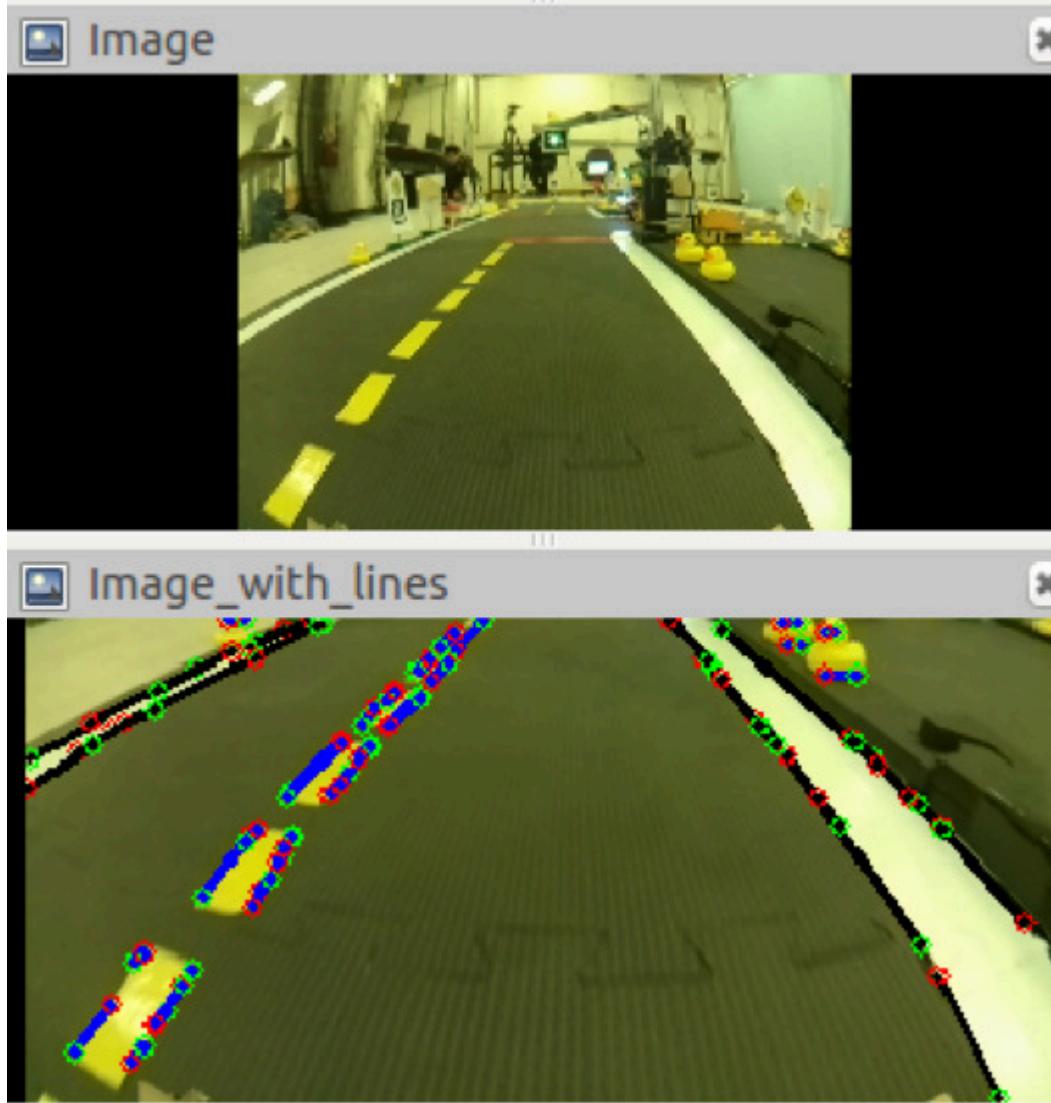


Figure 46

160.2. Quantitative tests

TODO: Something more quantitative (to be filled in by Liam or Hang)

160.3. line_detector_node2

This is a rewriting of `line_detector_node` using the EasyNode framework.

1) Parameters

Parameter `top_cutoff`: `int`; default value: `None`

This parameter decides how much of the image we should cut off. This is a performance improvement.

Parameter `detector`: not known

This parameter defines the line detector instantiation. It consists of an array of two elements, where the first is a classname or function, and the second is a dictionary of parameters to pass to the function.

Parameter `img_size`: not known

TODO: Missing description for entry “`img_size`”.

Parameter `verbose`: `bool`; default value: `True`

Whether the node is verbose or not. If set to `True`, the node will write timing statistics to the log.

2) Subscriptions

Subscription `switch`: topic `~switch` (`BoolStamped`)

This is a switch that allows to control the activity of this node. If the message is true, the node becomes active. If false, it switches off. The node starts as active.

Subscription `image`: topic `~image` (`CompressedImage`)

This is the compressed image to read. Note that it takes a long time to simply decode the image JPG.

Note: The data is processed *asynchronously* in a different thread.

Subscription `transform`: topic `~transform` (`AntiInstagramTransform`)

The anti-instagram transform to apply. See [Chapter 153](#).

3) Published topics

Publisher `color_segment`: topic `~colorSegment` (`Image`)

TODO: Missing description for entry “`color_segment`”.

Publisher `edge`: topic `~edge` (`Image`)

TODO: Missing description for entry “`edge`”.

Publisher `segment_list`: topic `~segment_list` (`SegmentList`)

TODO: Missing description for entry “`segment_list`”.

Publisher `image_with_lines`: topic `~image_with_lines` (`Image`)

TODO: Missing description for entry “`image_with_lines`”.

CHAPTER 161

Package `line_detector`

..

This package is being replaced by the cleaned-up version, [line_detector2](#). Do not write documentation here.

However, at the moment, all the launch files still call this one.

CHAPTER 162

Package `pi_camera`



TODO: to write

PART 14

Packages - Indefinite navigation

..

TODO: to write

CHAPTER 163

Package fsm

• •

TODO: to write

CHAPTER 164

Package `indefinite_navigation`

..

TODO: to write

CHAPTER 165

Package intersection_control



TODO: to write

CHAPTER 166

Package navigation

..

TODO: to write

CHAPTER 167

Package `stop_line_filter`

TODO: to write

PART 15

Packages - Localization and planning

..

TODO: to write

PART 16

Packages - Coordination



TODO: to write

PART 17

Packages - Additional functionality

..

TODO: to write

PART 18

Packages - Templates



These are templates.

CHAPTER 168

Package `pkg_name`



The package `pkg_name` is a template for ROS packages.

For the tutorial, see [Chapter 109](#).



168.1. Status

Given an honest assessment of the status of this package.

CHAPTER 169

Package `rostest_example`

TODO: to write

PART 19

Packages - Convenience

TODO: to write



CHAPTER 170

Package duckietown_demos



TODO: to write

CHAPTER 171

Package duckietown_unit_test

..

TODO: to write

PART 20

Packages - To sort



We need to decide where these packages go.

CHAPTER 172

Package `adafruit_imu`

172.1. Testing

To test run:

TODO

172.2. Dependencies

- `rospy`
- `sensor_msgs`: for the `Joy.msg`
- `duckietown_msgs`: for the `CarControl.msg`

172.3. Node `adafruit_imu`

This node reads sensor data from adafruit IMU and publishes it to `sensor_msgs.Imu` and `sensor_msgs.MagneticField`.

1) Parameters

- `~pub_timestep`: Time steps (in seconds) between publishings of `CarControl` msgs. Default to 0.02 (50 Hz).

2) Publish Topics

- `~adafruit_imu: sensor_msgs.Imu`
`Imu.angular_velocity`: Vector3 of angular velocity vector.
`Imu.linear_acceleration`: Vector3 of linear acceleration.
- `~adafruit_mag: sensor_msgs.MagneticField`
`MagneticField.magnetic_field`: Vector3 of magnetic field.

3) Services

None

CHAPTER 173

Package `apriltags_ros`

AprilTags for ROS.

build passing

CHAPTER 174

Package duckie_rr_bridge

TODO: to write

CHAPTER 175

Package `duckiebot_visualizer`



TODO: to write

CHAPTER 176

Package duckietown_description

TODO: to write

CHAPTER 177

Package `duckietown_logs`



TODO: to write.

Until we fix the dependencies:

```
sudo pip install SystemCmd==1.2 ros_node_utils==1.0 ConfTools==1.8 QuickApp==1.2.2  
sudo apt-get install -y mplayer mencoder  
  
sudo add-apt-repository ppa:mc3man/trusty-media  
sudo apt-get update  
sudo apt-get install -y ffmpeg gstreamer0.10-ffmpeg
```

CHAPTER 178

Package bag_stamper

TODO: to write

CHAPTER 179

Package kinematics



TODO: to write

CHAPTER 180

Package `visual_odometry`

..

TODO: to write

CHAPTER 181

LED emitter



The coordination team will use 3 signals: CAR_SIGNAL_A, CAR_SIGNAL_B, CAR_SIGNAL_C. To test the LED emitter with your joystick, run the following command:

```
$ roslaunch led_joy_mapper led_joy_with_led_emitter_test.launch veh:=robot name
```

This launches the joy controller, the mapper controller, and the led emitter nodes. You should not need to run anything external for this to work. Use the joystick buttons A, B and C to change your duckiebot's LED's blinking frequency.

Button A broadcasts signal CAR_SIGNAL_A (2.8hz), button B broadcasts signal CAR_SIGNAL_B (4.1hz), and button CAR_SIGNAL_C (Y on the controller) broadcasts signal C(5hz).The LB button will make the LEDs all white, the RB button will make some LEDs blue and some LEDs green, and the logitek button (middle button) will make the LEDs all red

Repeat this for each vehicle at the intersection that you wish to be blinking. Use previous command replacing **robot name** the names of the vehicles and try command different blinking patterns on different duckiebots.

(optional tests) For a grasp of the low level LED emitter, run:

```
$ roslaunch led_emitter led_emitter_node.launch veh:=robot name
```

You can then publish to the topic manually by running the following command in another screen on the duckiebot:

```
$ rostopic pub /robot name/led_emitter_node/change_to_state std_msgs/Float32 float-value
```

Where **float-value** is the desired blinking frequency, e.g. 1.0, .5, 3.0, etc. If you wish to run the LED emitter test, run the following:

```
$ roslaunch led_emitter led_emitter_node_test.launch veh:=robot name
```

This will cycle through frequencies of 3.0hz, 3.5hz, and 4hz every 5 seconds. Once done, kill everything and make sure you have joystick control as described above.

181.1. LED detector



Pick your favourite duckiebot as the observer-bot. Refer to it as **robot name** for this step. If you are in good company, this can be tried on all the available duckiebots. First, activate the camera on the observer-bot:

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

In a separate terminal, fire up the LED detector and the custom GUI by running:

```
$ rosrun led_detector LED_detector_with_gui.launch veh:=robot name
```

| **Note:** to operate without a GUI:



```
$ rosrun led_detector LED_detector.launch veh:=robot name
```

The LED_detector_node will be launched on the robot, while LED_visualizer (a simple GUI) will be started on your laptop. Make sure the camera image from the observer-bot is visualized and updated in the visualizer (tip: check that your camera cap is off).

Hit on Detect and wait to trigger a detection. This will not have any effect if LED_detector_node is not running on the duckiebot (it is included in the above launch file). After the capture and processing phases, the outcome will look like:

The red numbers represent the frequencies directly inferred from the camera stream, while the selected detections with the associated signaling frequencies will be displayed in green. You can click on the squares to visualize the brightness signals and the Fourier amplitude spectra of the corresponding cells in the video stream. You can also click on the camera image to visualize the variance map.

181.2. Unit tests



To run the unit tests for the LED detector, you need to have the F23 rosbags on you hard disk. These bag files should be synced from [this dropbox link] (https://www.dropbox.com/sh/5kx8qwggttu69fhr/AAASLpOVjV5r1xpzeW7xWZh_a?dl=0).

For the test to locate the bag files, you should have the DUCKIETOWN_DATA environment variable set, pointing to the location of you duckietown-data folder. This can be achieved by:

```
$ export DUCKIETOWN_DATA=local-path-to-duckietown-data-folder
```

All the available tests are specified in file `all_tests.yaml` in the scripts/ folder of the package led_detection in the duckietown ROS workspace. To run these, use the command:

```
$ rosrun led_detection unittests.py algorithm name-of-test
```

Currently, `algorithm` can be either 'baseline' or 'LEDDetector_plots' to also display the plot in the process. To run all test with all algorithms, execute:

```
$ rosrun led_detection unittests.py '*' '*'
```

CHAPTER 182

Package led_detection



TODO: to write

182.1. Unit tests



Quick command:

```
$ rosrun led_detection unitests.py '*' '*'
```

More in general:

```
$ rosrun led_detection unitests.py tests algorithms
```

where:

- **tests** is a comma separated list of algorithms. May use “*”.
- **algorithms** is a comma separated list of algorithms. May use “*”.

For example, this runs all tests on all algorithms:

```
$ rosrun led_detection unitests.py '*' '*'
```

The default algorithm is called “`baseline`”, and its tests are invoked using:

```
$ rosrun led_detection <script> '*' 'baseline'
```

CHAPTER 183

Package led_emitter

TODO: to write

CHAPTER 184

Package led_interpreter



TODO: to write

CHAPTER 185

Package led_joy_mapper

TODO: to write

CHAPTER 186

Package `rgb_led`

TODO: to write

186.1. Demos

To test the traffic light:

```
$ rosrun rgb_led blink trafficlight4way
```

Fancy test:

```
$ rosrun rgb_led blink trafficlight4way
```

To do other tests:

```
$ rosrun rgb_led blink
```

CHAPTER 187

Package traffic_light

TODO: to write

CHAPTER 188

Package localization



TODO: to write

CHAPTER 189

Package mdoap

..

TODO: to write

CHAPTER 190

Package parallel_autonomy



TODO: to write

CHAPTER 191

Package scene_segmentation

TODO: to write

CHAPTER 192

Package `veh_coordinator`

TODO: to write

CHAPTER 193

Package vehicle_detection

..

TODO: to write

CHAPTER 194

Package `visual_odometry_line`

TODO: to write

PART 21

Packages - Failed projects

..

These packages are abandoned failed projects.

CHAPTER 195

Package `mouse_encoder`



Use a mouse as encoder. Requires read permission to `/dev/input/mice`.

195.1. Publish Topic



- `mouse_encoder/tick: geometry_msg/Point` message with number of ticks in the x and y direction.

195.2. Parameters



- `~dev_path`: Default to `/dev/input/mice`. Point to the device path of the mouse.

195.3. Getting access to `/dev/input/mice`.



- Create a group named `input`

```
$ sudo groupadd input
```
* Add yourself to the `input` group
```
$ sudo adduser your_user_name input
```

- Log out and log back in for the change to take effect
- Put all devices under `/dev/input/` into the `input` group to grant the group read/write permission. Can be done by adding a file name `99-pure-data.rules` under `/etc/udev/rules.d` with the following line:

```
SUBSYSTEM=="input", GROUP="input", MODE="660"
```

- Reboot for the rule to take effect.

CHAPTER 196

Package simcity - Map Editor Version 0.1

All ./ references point to duckietown(Software)/catkin_ws/src/simcity A good reference for duckietown packages in general is catkin_ws/src/pkg_name/howto.md

196.1. How to run the map editor

(V0.1)

Inside ./launch is basic_map_tiler.launch. Ensure that the map file's path is correct for your machine. We start simcity, given a specific map file. We also start rviz, ROS' common gui.

196.2. How to edit the map

(V0.1)

The map is contained in ./maps as a YAML file. map.yaml is a small example of some circular streets, and censi_map.yaml is the map of the duckietown currently up and running.

196.3. What am I looking at, anyway?

(V0.1)

The magenta arrows point in the direction of traffic. These arrows are lines indicating where traffic flows.

196.4. What else is there to do?

(V0.1)

Lots of things. This part is mostly for rmata (1/11/16)

- (1) Beautify it. Arrows are straight and ugly. Roads in duckietown can be curved, have not-so-subtle lane markings, stop signs, grass, and potentially cones and duckies.....
- (2) Make it interactive. MarkerArrays consist of Markers. What does an InteractiveMarker consist of?
- (3) Establish consistency and validation when adding a tile to a map. This would involve:
 - checking node positions at adjacent tiles
 - multiplying the sparse lane matrices and checking that number of lanes is consistent
 - having a perhaps separate node receive messages from the interactive server, or the basic_map_tiler node, and doing these computations for each change

CHAPTER 197

Package slam



TODO: to write

CHAPTER 198

Package `street_name_detector`

..

TODO: to write

Page left blank