

The Duckiebook

..



The last version of this book and other documents are available at the URL

<http://book.duckietown.org/>

[Google search the documentation](#) (with some delay, for Google to index the pages)

TABLE OF CONTENTS



Part 1 - The Duckietown project	14
Chapter 1 - What is Duckietown?.....	15
Section 1.1 - Goals and objectives.....	15
Section 1.2 - Learn about the Duckietown educational experience.....	15
Section 1.3 - Learn about the platform.....	15
Chapter 2 - Duckietown history and future.....	17
Section 2.1 - The beginnings of Duckietown	17
Section 2.2 - University-level classes in 2016.....	18
Section 2.3 - University-level classes in 2017.....	18
Section 2.4 - Chile.....	19
Section 2.5 - Duckietown High School.....	19
Chapter 3 - Duckietown classes	20
Section 3.1 - 2016.....	20
Section 3.2 - 2017.....	21
Chapter 4 - First steps	23
Section 4.1 - How to get started	23
Section 4.2 - Duckietown for instructors	23
Section 4.3 - Duckietown for self-guided learners	23
Section 4.4 - Introduction for companies	23
Section 4.5 - How to keep in touch	23
Section 4.6 - How to contribute	23
Section 4.7 - Frequently Asked Questions	23
Part 2 - Duckumentation documentation	25
Chapter 5 - Accounts	26
Section 5.1 - Complete list of accounts.....	26
Section 5.2 - For other contributors.....	26
Chapter 6 - Contributing to the documentation	27
Section 6.1 - Where the documentation is	27
Section 6.2 - Editing links.....	27
Section 6.3 - Comments	27
Section 6.4 - Installing the documentation system	27
Section 6.5 - Compiling the documentation	29
Section 6.6 - Troubleshooting compilation	29
Section 6.7 - The workflow to edit documentation	30
Section 6.8 - *Deploying the documentation	30
Section 6.9 - *Compiling the PDF version	30
Chapter 7 - Features of the documentation writing system	33
Section 7.1 - Markdown	33
Section 7.2 - Embedded LaTeX	33
Section 7.3 - LaTeX equations	34
Section 7.4 - LaTeX symbols.....	35
Section 7.5 - Variables in command lines and command output	35
Section 7.6 - Character escapes	36
Section 7.7 - Keyboard keys.....	36
Section 7.8 - Figures	36
Section 7.9 - Subfigures.....	37
Section 7.10 - Shortcut for tables	38
Section 7.11 - Linking to documentation	39
Section 7.12 - Embedding videos	40
Section 7.13 - Bibliography	41
Section 7.14 - move-here tag	41
Section 7.15 - Comments	42

Section 7.16 - Special paragraphs tags	42
Section 7.17 - Div environments	45
Section 7.18 - Referring to Github files	46
Section 7.19 - Putting code from the repository in line	47
Chapter 8 - Documentation style guide	49
Section 8.1 - General guidelines for technical writing	49
Section 8.2 - Style guide for the Duckietown documentation	49
Section 8.3 - Writing command lines	49
Section 8.4 - Frequently misspelled words	50
Section 8.5 - Other conventions	50
Section 8.6 - Troubleshooting sections	50
Chapter 9 - Knowledge graph	52
Section 9.1 - Formalization	52
Section 9.2 - Atoms properties	53
Section 9.3 - Markdown format for text-like atoms	53
Section 9.4 - How to describe the semantic graphs of atoms	54
Section 9.5 - How to describe modules	54
Chapter 10 - Translations	56
Section 10.1 - File organization	56
Section 10.2 - Guidelines for English writers	56
Section 10.3 - File format	56
 Part 3 - Operation manual - Duckiebot	 58
Chapter 11 - Duckiebot configurations	59
Section 11.1 - Configuration list	59
Section 11.2 - Configuration functionality	59
Chapter 12 - Acquiring the parts for the Duckiebot C0	61
Section 12.1 - Bill of materials	61
Section 12.2 - Chassis	62
Section 12.3 - Raspberry Pi 3 - Model B	62
Section 12.4 - Camera	64
Section 12.5 - DC Stepper Motor HAT	65
Section 12.6 - Battery	66
Section 12.7 - Standoffs, Nuts and Screws	66
Section 12.8 - Zip Tie	67
Section 12.9 - Configuration C0-w	67
Section 12.10 - Configuration C0-j	68
Section 12.11 - Configuration C0-d	68
Chapter 13 - Soldering boards for C0	69
Chapter 14 - Preparing the power cable for C0	70
Section 14.1 - Step 1: Find a cable	70
Section 14.2 - Step 2: Cut the cable	71
Section 14.3 - Step 3: Strip the cable	71
Section 14.4 - Step 3: Strip the wires	72
Section 14.5 - Step 4: Find the power wires	73
Section 14.6 - Step 5: Test correct operation	73
Chapter 15 - Assembling the Duckiebot C0	75
Chapter 16 - Reproducing the image	76
Section 16.1 - Download and uncompress the Ubuntu Mate image	76
Section 16.2 - Burn the image to an SD card	76
Section 16.3 - Raspberry Pi Config	77
Section 16.4 - Install packages	77
Section 16.5 - Install Edimax driver	78
Section 16.6 - Install ROS	79
Section 16.7 - Wireless configuration (old version)	79
Section 16.8 - Wireless configuration	80
Section 16.9 - SSH server config	82
Section 16.10 - Create swap Space	82
Section 16.11 - Passwordless sudo	83
Section 16.12 - Clean up	83

Section 16.13 - Ubuntu user configuration	84
Section 16.14 - Check that all required packages were installed.....	85
Section 16.15 - Creating the image	85
Section 16.16 - Some additions since last image to add in the next image	86
Chapter 17 - Installing Ubuntu on laptops	87
Section 17.1 - Install Ubuntu	87
Section 17.2 - Install useful software	87
Section 17.3 - Install ROS	88
Section 17.4 - Other suggested software.....	88
Section 17.5 - Installation of the duckuments system	88
Section 17.6 - Passwordless sudo	88
Section 17.7 - SSH and Git setup.....	88
Chapter 18 - Duckiebot Initialization	90
Section 18.1 - Acquire and burn the image.....	90
Section 18.2 - Turn on the Duckiebot.....	90
Section 18.3 - Connect the Duckiebot to a network.....	91
Section 18.4 - Ping the Duckiebot.....	91
Section 18.5 - SSH to the Duckiebot.....	91
Section 18.6 - (For D17-C1) Configure the robot-generated network	92
Section 18.7 - Setting up wireless network configuration	92
Section 18.8 - Update the system	93
Section 18.9 - Give a name to the Duckiebot.....	93
Section 18.10 - Change the hostname	93
Section 18.11 - Expand your filesystem.....	94
Section 18.12 - Create your user	95
Section 18.13 - Other customizations.....	97
Section 18.14 - Hardware check: camera	97
Chapter 19 - Software setup and RC remote control	98
Section 19.1 - Clone the Duckietown repository.....	98
Section 19.2 - Set up ROS environment on the Duckiebot.....	98
Section 19.3 - Add your vehicle to the scuderia file	99
Section 19.4 - Test that the joystick is detected	99
Section 19.5 - Run the joystick demo	99
Section 19.6 - The proper shutdown procedure for the Raspberry Pi	100
Chapter 20 - Reading from the camera	102
Section 20.1 - Check the camera hardware.....	102
Section 20.2 - Create two windows.....	102
Section 20.3 - First window: launch the camera nodes	102
Section 20.4 - Second window: view published topics.....	103
Chapter 21 - RC control launched remotely	105
Section 21.1 - Two ways to launch a program	105
Section 21.2 - Download and setup Software repository on the laptop	105
Section 21.3 - Edit the machines files on your laptop	105
Section 21.4 - Start the demo.....	105
Section 21.5 - Watch the program output using rqt_console.....	106
Section 21.6 - Troubleshooting.....	106
Chapter 22 - RC+camera remotely	108
Section 22.1 - Assumptions	108
Section 22.2 - Terminal setup	108
Section 22.3 - First window: launch the joystick demo	108
Section 22.4 - Second window: launch the camera nodes.....	109
Section 22.5 - Third window: view data flow	109
Section 22.6 - Fourth window: visualize the image using rviz.....	109
Section 22.7 - Proper shutdown procedure	110
Chapter 23 - Interlude: Ergonomics	111
Section 23.1 - set_ros_master.sh	111
Section 23.2 - SSH aliases	111
Chapter 24 - Wheel calibration	113
Chapter 25 - Camera calibration	114
Chapter 26 - Taking a log	115

Part 4 - Operation manual - Duckietowns	116
Chapter 27 - Duckietown parts	117
Section 27.1 - Bill of materials.....	117
Section 27.2 - Duckies.....	117
Section 27.3 - Floor Mats	118
Section 27.4 - Duck Tape	118
Section 27.5 - Traffic Signs.....	119
Chapter 28 - Traffic lights Parts	120
Section 28.1 - Bill of materials.....	120
Section 28.2 - Raspberry Pi	120
Chapter 29 - Duckietown Assembly.....	121
Chapter 30 - Traffic lights Assembly	122
Chapter 31 - The Duckietown specification	123
Section 31.1 - Topology	123
Section 31.2 - Signs placement.....	123
Part 5 - Operation manual - Duckiebot with LEDs.....	124
Chapter 32 - Acquiring the parts for the Duckiebot C1.....	125
Section 32.1 - Bill of materials.....	125
Section 32.2 - LEDs	126
Section 32.3 - Bumpers	127
Section 32.4 - Headers, resistors and jumper	128
Chapter 33 - Soldering boards for C1	129
Chapter 34 - Assembling the Duckiebot C1.....	130
Chapter 35 - C1 (LEDs) setup	131
Part 6 - Theory chapters.....	132
Chapter 36 - Chapter template.....	133
Section 36.1 - Example Title: PID control	133
Section 36.2 - Problem Definition.....	133
Section 36.3 - Introduced Notions	134
Section 36.4 - Examples	135
Section 36.5 - Pointers to Exercises.....	135
Section 36.6 - Conclusions.....	136
Section 36.7 - Next Steps.....	136
Section 36.8 - References	136
Chapter 37 - Symbols and conventions	137
Section 37.1 - Conventions	137
Section 37.2 - Table of symbols	137
Chapter 38 - Linear algebra.....	138
Section 38.1 - Problem Definition.....	138
Section 38.2 - Vectors	139
Section 38.3 - Matrices	140
Section 38.4 - Matrix as representation of linear (vector) spaces	141
Section 38.5 - Examples	141
Section 38.6 - Pointers to Exercises.....	141
Section 38.7 - Conclusions.....	141
Section 38.8 - Next Steps	142
Chapter 39 - Probability basics.....	143
Chapter 40 - Dynamics	144
Chapter 41 - Coordinate systems	145
Chapter 42 - Reference frames	146
Chapter 43 - Transformations	147
Chapter 44 - Autonomy overview	148
Section 44.1 - Basic Building Blocks of Autonomy	148
Section 44.2 - Advanced Building Blocks of Autonomy	150
Chapter 45 - Autonomy architectures	151
Section 45.1 - Contracts	151

Chapter 46 - Representations	152
Section 46.1 - Preliminaries.....	152
Section 46.2 - Robot Representations	153
Section 46.3 - Environment Representations	153
Chapter 47 - Software architectures and middlewares.....	154
Chapter 48 - Modern signal processing.....	155
Chapter 49 - Basic Kinematics	156
Chapter 50 - Basic Dynamics	157
Chapter 51 - Odometry Calibration.....	158
Chapter 52 - Computer vision basics.....	159
Chapter 53 - Illumination invariance.....	160
Chapter 54 - Line Detection	161
Chapter 55 - Feature extraction	162
Chapter 56 - Place recognition.....	163
Chapter 57 - Filtering 1.....	164
Chapter 58 - Filtering 2.....	165
Chapter 59 - Mission planning.....	166
Chapter 60 - Planning in discrete domains	167
Chapter 61 - Motion planning.....	168
Chapter 62 - RRT	169
Chapter 63 - Feedback control	170
Chapter 64 - PID Control.....	171
Chapter 65 - MPC Control.....	172
Chapter 66 - Object detection.....	173
Chapter 67 - Object classification	174
Chapter 68 - Object tracking	175
Chapter 69 - Reacting to obstacles.....	176
Chapter 70 - Semantic segmentation	177
Chapter 71 - Text recognition.....	178
Chapter 72 - SLAM - Problem formulation	179
Chapter 73 - SLAM - Broad categories	180
Chapter 74 - VINS	181
Chapter 75 - Advanced place recognition.....	182
Chapter 76 - Fleet level planning (placeholder).....	183
Chapter 77 - Fleet level planning (placeholder).....	184
Chapter 78 - Bibliography.....	185
Part 7 - Exercises	186
Chapter 79 - ROS Exercises	187
Section 79.1 - Parameters.....	187
Section 79.2 - Running from a log	187
Section 79.3 - Unit tests	187
Section 79.4 - Analytics.....	187
Section 79.5 - Visualization	187
Chapter 80 - Line detection	188
Chapter 81 - Data processing	189
Chapter 82 - Git and conventions	190
Part 8 - Software reference	191
Chapter 83 - Ubuntu packaging with APT	192
Section 83.1 - apt install	192
Section 83.2 - apt update	192
Section 83.3 - apt-key	192
Section 83.4 - apt-mark	192
Section 83.5 - add-apt-repository	192
Section 83.6 - wajig	192
Section 83.7 - dpigs	192
Chapter 84 - GNU/Linux general notions	193
Section 84.1 - Background reading	193

Chapter 85 - Every day Linux.....	194
Section 85.1 - cd	194
Section 85.2 - sudo.....	194
Section 85.3 - ls	194
Section 85.4 - cp	194
Section 85.5 - mkdir	194
Section 85.6 - touch.....	194
Section 85.7 - reboot	194
Section 85.8 - shutdown.....	194
Section 85.9 - rm	194
Chapter 86 - Users.....	195
Section 86.1 - passwd	195
Chapter 87 - UNIX tools	196
Section 87.1 - cat.....	196
Section 87.2 - tee.....	196
Section 87.3 - truncate.....	196
Chapter 88 - Linux disks and files	197
Section 88.1 - fdisk.....	197
Section 88.2 - mount	197
Section 88.3 - umount	197
Section 88.4 - losetup	197
Section 88.5 - gparted	197
Section 88.6 - dd	197
Section 88.7 - sync	197
Section 88.8 - df	197
Chapter 89 - Other administration commands	198
Section 89.1 - visudo	198
Section 89.2 - update-alternatives	198
Section 89.3 - udevadm	198
Section 89.4 - systemctl	198
Chapter 90 - Make	199
Section 90.1 - make.....	199
Chapter 91 - Python-related tools	200
Section 91.1 - virtualenv	200
Section 91.2 - pip.....	200
Chapter 92 - Raspberry-PI commands	201
Section 92.1 - raspi-config	201
Section 92.2 - vcgencmd.....	201
Section 92.3 - raspistill	201
Section 92.4 - jstest	201
Section 92.5 - swapon	201
Section 92.6 - mkswap	201
Chapter 93 - Users and permissions.....	202
Section 93.1 - chmod	202
Section 93.2 - groups	202
Section 93.3 - adduser	202
Section 93.4 - useradd	202
Chapter 94 - Downloading	203
Section 94.1 - curl.....	203
Section 94.2 - wget	203
Section 94.3 - sha256sum	203
Section 94.4 - xz	203
Chapter 95 - Shells and environments	204
Section 95.1 - source	204
Section 95.2 - which	204
Section 95.3 - export	204
Chapter 96 - Other misc commands.....	205
Section 96.1 - pgrep	205
Section 96.2 - npm.....	205
Section 96.3 - nodejs	205

Section 96.4 - ntpdate	205
Section 96.5 - chsh.....	205
Section 96.6 - echo.....	205
Section 96.7 - sh	205
Section 96.8 - fc-cache.....	205
Chapter 97 - Linux resources usage.....	206
Section 97.1 - Measuring CPU usage using htop	206
Section 97.2 - Measuring I/O usage using iotop.....	206
Section 97.3 - How fast is the SD card?	206
Chapter 98 - SD Cards tools	207
Section 98.1 - Testing SD Card and disk speed.....	207
Section 98.2 - How to burn an image to an SD card	207
Section 98.3 - How to shrink an image	208
Chapter 99 - Networking tools.....	211
Section 99.1 - hostname.....	211
Section 99.2 - Visualizing information about the network	211
Chapter 100 - Accessing computers using SSH.....	212
Section 100.1 - Background reading	212
Section 100.2 - Installation of SSH.....	212
Section 100.3 - Local configuration	212
Section 100.4 - How to login with SSH and a password	212
Section 100.5 - Creating an SSH keypair	213
Section 100.6 - How to login without a password	215
Section 100.7 - Fixing SSH Permissions	216
Section 100.8 - ssh-keygen	216
Chapter 101 - Wireless networking in Linux.....	217
Section 101.1 - iwconfig	217
Section 101.2 - iwlist	217
Chapter 102 - Moving files between computers.....	219
Section 102.1 - SCP	219
Section 102.2 - RSync	219
Chapter 103 - VIM.....	220
Section 103.1 - External documentation	220
Section 103.2 - Installation	220
Section 103.3 - vi	220
Section 103.4 - Suggested configuration	220
Section 103.5 - Visual mode	220
Section 103.6 - Indenting using VIM.....	220
Chapter 104 - Atom.....	222
Chapter 105 - Eclipse	223
Section 105.1 - Installing LiClipse	223
Chapter 106 - Byobu.....	224
Section 106.1 - Advantages of using Byobu	224
Section 106.2 - Installation	224
Section 106.3 - Documentation.....	224
Section 106.4 - Quick command reference	224
Section 106.5 - Commands on OS X.....	225
Chapter 107 - Source code control with Git.....	226
Section 107.1 - Background reading	226
Section 107.2 - Installation	226
Section 107.3 - Setting up global configurations for Git	226
Section 107.4 - Git tips	226
Section 107.5 - Git troubleshooting	227
Section 107.6 - git.....	227
Chapter 108 - Git LFS	228
Section 108.1 - Generic installation instructions	228
Section 108.2 - Ubuntu 16 installation (laptop)	228
Section 108.3 - Ubuntu 16 Mate installation (Raspberry Pi 3).....	228
Chapter 109 - Setup Github access	229
Section 109.1 - Create a Github account	229

Section 109.2 - Become a member of the Duckietown organization	229
Section 109.3 - Add a public key to Github.....	229
Chapter 110 - ROS installation and reference	231
Section 110.1 - Install ROS	231
Section 110.2 - <code>rqt_console</code>	231
Section 110.3 - <code>roslaunch</code>	232
Section 110.4 - <code>rviz</code>	232
Section 110.5 - <code>rostopic</code>	232
Section 110.6 - <code>catkin_make</code>	232
Section 110.7 - <code>rosrun</code>	232
Section 110.8 - <code>rostest</code>	232
Section 110.9 - <code>rospack</code>	232
Section 110.10 - <code>rosparam</code>	232
Section 110.11 - <code>rosdep</code>	233
Section 110.12 - <code>roswtf</code>	233
Section 110.13 - <code>rosbag</code>	233
Section 110.14 - <code>roscore</code>	233
Section 110.15 - Troubleshooting ROS	233
Section 110.16 - Other materials about ROS.....	233
Part 9 - Software development guide.....	234
Chapter 111 - Python	235
Section 111.1 - Background reading	235
Section 111.2 - Python virtual environments	235
Section 111.3 - Useful libraries.....	235
Section 111.4 - Context managers.....	235
Chapter 112 - Duckietown code conventions.....	236
Section 112.1 - Python	236
Section 112.2 - Logging.....	237
Section 112.3 - Exceptions	237
Section 112.4 - Scripts	237
Chapter 113 - Configuration	238
Section 113.1 - Environment variables.....	238
Section 113.2 - The scuderia file	238
Section 113.3 - The machines file.....	239
Section 113.4 - People database.....	239
Section 113.5 - Modes of operation.....	239
Chapter 114 - Node configuration mechanisms.....	240
Chapter 115 - Minimal ROS node - <code>pkg_name</code>	241
Section 115.1 - The files in the package	241
Section 115.2 - Writing a node: <code>talker.py</code>	242
Section 115.3 - The <code>Talker</code> class.....	244
Section 115.4 - Launch File	245
Section 115.5 - Testing the node	246
Section 115.6 - Documentation	248
Section 115.7 - Guidelines	248
Chapter 116 - ROS package verification	249
Section 116.1 - Naming	249
Section 116.2 - <code>package.xml</code>	249
Section 116.3 - Messages.....	249
Section 116.4 - <code>Readme</code> file	249
Section 116.5 - Launch files.....	249
Section 116.6 - Test files.....	249
Chapter 117 - Creating unit tests with ROS.....	250
Part 10 - Duckietown system.....	251
Chapter 118 - Teleoperation	252
Section 118.1 - Implementation	252
Section 118.2 - Camera	252

Section 118.3 - Actuators	252
Section 118.4 - IMU	252
Chapter 119 - Parallel autonomy	253
Chapter 120 - Lane control	254
Section 120.1 - Implementation	254
Chapter 121 - Indefinite navigation	255
Section 121.1 - Implementation	255
Chapter 122 - Planning	256
Section 122.1 - Implementation	256
Chapter 123 - Coordination	257
Section 123.1 - Implementation	257
Chapter 124 - Duckietown ROS Guidelines	258
Section 124.1 - Node and Topics	258
Section 124.2 - Parameters	258
Section 124.3 - Launch file	258
Part 11 - Fall 2017	260
Chapter 125 - General remarks	261
Section 125.1 - The rules of Duckietown	261
Section 125.2 - Synchronization between classes	261
Section 125.3 - Accounts for students	261
Section 125.4 - Accounts for all instructors and TAs	262
Chapter 126 - Additional information for ETH Zürich students	263
Chapter 127 - Additional information for UdeM students	264
Chapter 128 - Additional information for TTIC students	265
Chapter 129 - Additional information for NCTU students	266
Chapter 130 - Milestone: ROS node working	267
Chapter 131 - Homework: Take and process a log	268
Chapter 132 - Milestone: Calibrated robot	269
Chapter 133 - Homework: Camera geometry	270
Chapter 134 - Milestone: Illumination invariance	271
Chapter 135 - Homework: Place recognition	272
Chapter 136 - Milestone: Lane following	273
Chapter 137 - Homework: localization	274
Chapter 138 - Milestone: Navigation	275
Chapter 139 - Homework: group forming	276
Chapter 140 - Milestone: Ducks in a row	277
Chapter 141 - Homework: Comparison of PID	278
Chapter 142 - Homework: RRT	279
Chapter 143 - Caffe tutorial	280
Chapter 144 - Milestone: Object Detection	281
Chapter 145 - Homework: Object Detection	282
Chapter 146 - Milestone: Semantic perception	283
Chapter 147 - Homework: Semantic perception	284
Chapter 148 - Milestone: Reacting to obstacles	285
Chapter 149 - Homework: Reacting to obstacles	286
Chapter 150 - Milestone: SLAM demo	287
Chapter 151 - Homework: SLAM	288
Chapter 152 - Milestone: fleet demo	289
Chapter 153 - Homework: fleet	290
Chapter 154 - Project proposals	291
Chapter 155 - Template of a project	292
Section 155.1 - Checklist for students	292
Section 155.2 - Checklist for TAs	292
Part 12 - Packages - Infrastructure	293
Chapter 156 - Package duckietown	294
Section 156.1 - Package information	294
Chapter 157 - Package duckietown_msgs	295

Section 157.1 - Package information	295
Section 157.2 - Package information	295
Chapter 158 - Package easy_algo.....	296
Section 158.1 - Package information	296
Chapter 159 - Package easy_logs.....	297
Chapter 160 - Package easy_node.....	298
Section 160.1 - Package information	298
Section 160.2 - YAML file format.....	298
Section 160.3 - Using the easy_node API.....	300
Section 160.4 - Configuration using easy_node: the user's point of view.....	304
Section 160.5 - Visualizing the configuration database.....	306
Section 160.6 - Benchmarking	308
Section 160.7 - Automatic documentation generation	309
Section 160.8 - Parameters and services defined for all packages	310
Section 160.9 Other ideas	310
Chapter 161 - Package what_the_duck.....	311
Section 161.1 - Package information	311
Section 161.2 - What the duck	311
Section 161.3 - Adding more tests to what-the-duck	311
Section 161.4 - Tests already added	311
Section 161.5 - List of tests to add	312
Part 13 - Packages - Lane control.....	314
Chapter 162 - Package adafruit_drivers	315
Section 162.1 - Package information	315
Chapter 163 - Package anti_instagram.....	316
Section 163.1 - Package information	316
Section 163.2 - Unit tests integrated with rostest	316
Section 163.3 - Unit tests needed external files	316
Section 163.4 - Node anti_instagram_node	316
Chapter 164 - Package dagu_car	318
Section 164.1 - Package information	318
Section 164.2 - Node forward_kinematics_node	318
Section 164.3 - Node inverse_kinematics_node	318
Section 164.4 - Node velocity_to_pose_node	319
Section 164.5 - Node car_cmd_switch_node	319
Section 164.6 - Node wheels_driver_node	319
Section 164.7 - Node wheels_trimmer_node	320
Chapter 165 - Package ground_projection.....	321
Section 165.1 - Package information	321
Section 165.2 - Node ground_projection_node	321
Chapter 166 - Package joy_mapper	322
Section 166.1 - Package information	322
Section 166.2 - Testing	322
Section 166.3 - Dependencies.....	322
Section 166.4 - Node joy_mapper_node2.....	322
Chapter 167 - Package lane_control	324
Section 167.1 - Package information	324
Section 167.2 - lane_controller_node.....	324
Chapter 168 - Package lane_filter	326
Section 168.1 - Package information	326
Section 168.2 - lane_filter_node.....	326
Chapter 169 - Package line_detector2	329
Section 169.1 - Package information	329
Section 169.2 - Testing the line detector using visual inspection	329
Section 169.3 - Quantitative tests.....	331
Section 169.4 - line_detector_node2	331
Chapter 170 - Package line_detector	333
Section 170.1 - Package information	333
Chapter 171 - Package pi_camera.....	334

Section 171.1 - Package information	334
Section 171.2 - Node camera_node_sequence.....	334
Section 171.3 - Node camera_node_continuous	334
Section 171.4 - Node decoder_node.....	335
Section 171.5 - Node img_process_node.....	335
Section 171.6 - Node cam_info_reader_node.....	336
Part 14 - Packages - Indefinite navigation.....	337
Chapter 172 - AprilTags library.....	338
Section 172.1 - Package information	338
Chapter 173 - Package apriltags_ros.....	340
Section 173.1 - Package information	340
Chapter 174 - Package fsm	341
Section 174.1 - Package information	341
Section 174.2 - Node fsm_node.....	341
Chapter 175 - Package indefinite_navigation.....	343
Section 175.1 - Package information	343
Chapter 176 - Package intersection_control	344
Section 176.1 - Package information	344
Chapter 177 - Package navigation	345
Section 177.1 - Package information	345
Chapter 178 - Package stop_line_filter	346
Section 178.1 - Package information	346
Part 15 - Packages - Localization and planning	347
Chapter 179 - Package duckietown_description.....	348
Section 179.1 - Package information	348
Chapter 180 - Package localization	349
Section 180.1 - Package information	349
Part 16 - Packages - Coordination.....	350
Chapter 181 - Package led_detection.....	351
Section 181.1 - Package information	351
Section 181.2 - LED detector	351
Section 181.3 - Unit tests	352
Chapter 182 - Package led_emitter	353
Section 182.1 - Package information	353
Section 182.2 - In depth	353
Chapter 183 - Package led_interpreter	355
Section 183.1 - Package information	355
Chapter 184 - Package led_joy_mapper	356
Section 184.1 - Package information	356
Chapter 185 - Package rgb_led	357
Section 185.1 - Package information	357
Section 185.2 - Demos.....	357
Chapter 186 - Package traffic_light	358
Section 186.1 - Package information	358
Part 17 - Packages - Additional functionality	359
Chapter 187 - Package mdap	360
Section 187.1 - Package information	360
Chapter 188 - Package parallel_autonomy	361
Section 188.1 - Package information	361
Chapter 189 - Package vehicle_detection	362
Section 189.1 - Package information	362
Part 18 - Packages - Templates	363
Chapter 190 - Package pkg_name.....	364

Section 190.1 - Package information	364
Chapter 191 - Package rostest_example	365
Section 191.1 - Package information	365
Part 19 - Packages - Convenience	366
Chapter 192 - Package ducky_rr_bridge	367
Section 192.1 - Package information	367
Chapter 193 - Package duckiebot_visualizer	368
Section 193.1 - Package information	368
Section 193.2 - Node duckiebot_visualizer.....	368
Chapter 194 - Package duckietown_demos	369
Section 194.1 - Package information	369
Chapter 195 - Package duckietown_logs	370
Section 195.1 - Package information	370
Section 195.2 - Misc.....	370
Chapter 196 - Package duckietown_unit_test	371
Section 196.1 - Package information	371
Chapter 197 - Package veh_coordinator	372
Section 197.1 - Package information	372
Part 20 - Packages - To sort.....	373
Part 21 - Packages - Failed projects.....	374
Chapter 198 - Package bag_stamper	375
Section 198.1 - Package information	375
Section 198.2 - Node bag_stamper_node.....	375
Chapter 199 - Package kinematics	377
Section 199.1 - Package information	377
Chapter 200 - Package visual_odometry	378
Section 200.1 - Package information	378
Chapter 201 - Package mouse_encoder	379
Section 201.1 - Publish Topic	379
Section 201.2 - Parameters.....	379
Section 201.3 - Getting access to /dev/input/mice.....	379
Chapter 202 - Package simcity - Map Editor Version 0.1	380
Section 202.1 - How to run the map editor	380
Section 202.2 - How to edit the map.....	380
Section 202.3 - What am I looking at, anyway?.....	380
Section 202.4 - What else is there to do?	380
Chapter 203 - Package slam.....	381
Chapter 204 - Package street_name_detector	382
Chapter 205 - Package adafruit_imu	383
Section 205.1 - Package information	383
Section 205.2 - Testing	383
Section 205.3 - Dependencies.....	383
Section 205.4 - Node adafruit_imu.....	383
Chapter 206 - Package car_supervisor	385
Section 206.1 - Package information	385
Chapter 207 - Package scene_segmentation	386
Section 207.1 - Package information	386
Chapter 208 - Package visual_odometry_line	387
Section 208.1 - Package information	387

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:

- [National Chiao Tung University 2016](#), Taiwan - Prof. Nick Wang;
- [Tsinghua University](#), People's Republic of China - Prof. (Samuel) Qing-Shan Jia's *Computer Networks with Applications* course;
- [Rensselaer Polytechnic Institute 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:

- Rensselaer Polytechnic Institute (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

Duckietown classes



Duckietown is an international effort. Many educational institutions have adopted the platform and used to teach robotics and related subjects. If you have used Duckietown in any of your course and cannot find a mention to it here, contact us.

TODO: add contact email

Here, we provide a chronologically ordered compendium of the Duckietown related learning experiences worldwide.

3.1. 2016



Duckietown was created in 2016.

1) Massachusetts Institute of Technology



Location: United States of America, Cambridge, Massachusetts

Course title: 2.166 Vehicle Autonomy

Instructors: *plenty*

Educational Level: Graduate

Time period:

Link:

Highlights: Role-playing experience (Duckietown Engineering Co.), public demo

Summary:

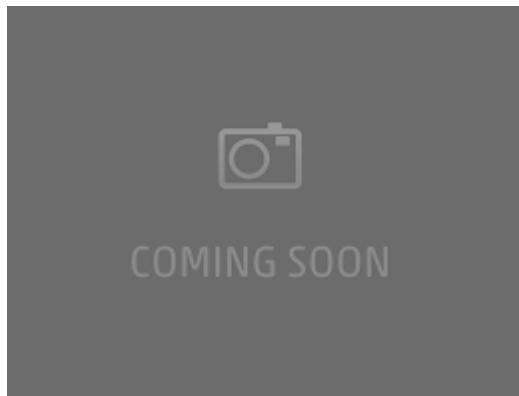


Figure 9. Duckietown at MIT in 2016

2) National Chiao Tung University



Location:

Course title:

Instructors: Prof. Nick Wang

Educational Level:

Time period:

Link:

Summary:

Highlights:

3) Tsinghua University



Location:

Course title:

Instructors:

Educational Level:

Time period:

Link:

Summary:

Highlights:

4) Rensselaer Polytechnic Institute



Location:

Course title:

Instructors:

Educational Level:

Time period:

Link:

Summary:

Highlights:

3.2. 2017



1) ETH Zürich



Assigned to: Censi/Tani

2) Université de Montréal



Location: Montreal

Course title: IFT 6080 Sujets en exploitation des ordinateurs

Instructors: Liam Paull

Educational Level: Graduate

Time period: Sept. - Dec. 2017

Link: [Official Course Website](#)

Summary: 12 students admitted spanning across Université de Montréal, École Polytechnic, McGill University and Concordia University

Highlights:

3) Toyota Technological Institute at Chicago / University of Chicago

Assigned to: Matt Walter

CHAPTER 4

First steps

4.1. How to get started

If you are an instructor, please jump to [Section 4.2](#).

If you are a self-guided learner, please jump to [Section 4.3](#).

If you are a company, and interested in working with Duckietown, please jump to [Section 4.4](#).

4.2. Duckietown for instructors

TODO: to write

4.3. Duckietown for self-guided learners

TODO: to write

4.4. Introduction for companies

TODO: to write

4.5. How to keep in touch

TODO: add link to Facebook

TODO: add link to Mailing list

TODO: add link to Slack?

4.6. 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...

4.7. 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.

PART 2

Duckumentation documentation

..

CHAPTER 5

Accounts

5.1. 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.

5.2. 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.

CHAPTER 6

Contributing to the documentation

6.1. 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](#).

6.2. 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.

6.3. 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.)

6.4. Installing the documentation system

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

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

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

1) Dependencies (Ubuntu 16.04)

On Ubuntu 16.04, these are the dependencies to install:

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

2) Download the duckuments repo

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

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

3) Setup the virtual environment

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

Change into that directory:

```
$ cd ~/duckuments
```

Create the virtual environment using `virtualenv`:

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

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

Activate the virtual environment:

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

4) Setup the `mcdp` external repository

Make sure you are in the directory:

```
$ cd ~/duckuments
```

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

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

Install it and its dependencies:

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

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

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

```
$ pip install numpy matplotlib
```

You also should run:

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

6.5. 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`.



6.6. 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.

6.7. 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.

6.8. *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.

6.9. *Compiling the PDF version

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

1) Installing nodejs

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

Run:

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

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

```
$ sudo apt remove nodejs
```

Install `nodejs` using [the instructions at this page](#).

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

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

2) Troubleshooting `nodejs` installation problems

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

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

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

3) Installing Prince

Install PrinceXML from [this page](#).

4) Installing fonts

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

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

and then rebuild the font cache using:

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

5) Compiling the PDF

To compile the PDF, use:

```
$ make compile-pdf
```

This creates the file:

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


CHAPTER 7

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.

7.1. Markdown

The Duckiebook is written in a Markdown dialect.

→ [A tutorial on Markdown.](#)

7.2. 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.

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

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

TABLE 1. LATEX ENVIRONMENTS AND LABEL PREFIXES

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

Examples of all environments follow.

example

```
<div id='def:lorem' class="definition latex_env" markdown="1">Lorem</div>
```

Definition 1. Lorem

```
<div id='prop:lorem' class="proposition latex_env" markdown="1">Lorem</div>
```

Proposition 2. Lorem

```
<div id='rem:lorem' class="remark latex_env" markdown="1">Lorem</div>
```

Remark 1. Lorem

```
<div id='prob:lorem' class="problem latex_env" markdown="1">Lorem</div>
```

Problem 1. Lorem

```
<div id='exa:lorem' class="example latex_env" markdown="1">Lorem</div>
```

Example 1. Lorem

```
<div id='thm:lorem' class="theorem latex_env" markdown="1">Lorem</div>
```

Theorem 1. Lorem

```
<div id='lem:lorem' class="lemma latex_env" markdown="1">Lorem</div>
```

Lemma 1. Lorem

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

I can also refer to all of them: [Definition 1](#), [Proposition 2](#), [Remark 1](#), [Problem 1](#), [Example 1](#), [Theorem 1](#), [Lemma 1](#).

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

$$2a = a + a \quad (1)$$

This uses `align` and contains (2) and (3).

$$a = b \quad (2)$$

$$= c \quad (3)$$

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

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

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

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

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

TODO: other LaTeX features supported

7.4. 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.

7.5. 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.

7.6. Character escapes

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

Other symbols to escape are shown in [Table 2](#).

TABLE 2. SYMBOLS TO ESCAPE

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

7.7. Keyboard keys

Use the `kbd` element for keystrokes.

example

For example, to obtain:

Press `a` then `Ctrl`-`C`.

use the following:

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

7.8. 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: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”.

7.9. 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 10. Caption of big figure

7.10. Shortcut for tables

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

The following code:

```

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

```

gives the following result:

TABLE 3. MY TABLE

A	B
C	D

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

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

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

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

TABLE 4. USING CLASS="LABELS-COL1"

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

TABLE 5. USING CLASS="LABELS-ROW1"

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

7.11. Linking to documentation

1) Establishing names of headers

You give IDs to headers using the format:

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

For example, for this subsection, we have used:

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

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

2) Linking from the documentation to the documentation

You can use the syntax:

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

to refer to the header.

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

TABLE 6. SYNTAX FOR REFERRING TO SECTIONS.

See `[](#establishing)`.

See [Subsection 7.11.1](#)

See ``.

See [Establishing names of headers](#).

See ``.

See [7.11.1](#).

See ``.

See [Subsection 7.11.1 - 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 mypackagemyscript
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 113.2](#).

7.12. Embedding videos

It is possible to embed Vimeo videos in the documentation.

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

This is the syntax:

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

example For example, this code:

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

produces this result:



Figure 11. 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).

7.13. 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 [\[3\]](#).

7.14. move-here tag

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

the place of the tag.

This is used for autogenerated documentation.

Syntax:

```
# Node `node`  
  
<move-here src="#package-node-autogenerated">
```

7.15. Comments

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

```
# My section  
  
<!-- this text is ignored -->  
  
Let's start by...
```

7.16. Special paragraphs tags

The system supports parsing of some special paragraphs.

Note that some of these might be redundant.

For now, documenting what is implemented.

1) Todos, task markers

TODO: todo

TODO: todo

TOWRITE: towrite

To write: towrite

Task: task

Task: task

Assigned: assigned

Assigned to: assigned

2) Notes and remarks

Remark: remark

|| **Remark:** remark

Note: note

|| **Note:** note

Symptom: symptom

|| **Symptom:** symptom

Warning: warning

|| **Warning:** warning

3) Troubleshooting

Symptom: symptom

|| **Symptom:** symptom

Resolution: resolution

4) Guidelines

Bad: bad

* bad

Better: better

✓ better

5) Questions and answers

Q: question

Q: question

A: answer

Answer: answer

6) Authors, maintainers, point of contact

Maintainer: maintainer

Maintainer: maintainer

Author: author

Author: author

Point of contact: point of contact

Point of contact: point of contact

Slack channel: slack channel

Slack channel: slack channel

7) References

See: see

→ see

Reference: reference

→ reference

Requires: requires

Requires: requires

Recommended: recommended

Recommended: recommended

See also: see also

* *see also*

7.17. Div environments

For these, note the rules:

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

1) Example usage

```
<div class='example-usage' markdown='1'>  
  
This is how you can use `rosbag`:  
  
  $ rosbag play log.bag  
  
</div>
```

example This is how you can use `rosbag`:

`$ rosbag play log.bag`

2) Check

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

Check before you continue

Check that you didn't forget anything.

3) Requirements

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

List of requirements at the beginning of setup chapter.

```
</div>
```

List of requirements at the beginning of setup chapter.

7.18. Referring to Github files



You can refer to files in the repository by using:

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

The available keys are:

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

For example, you can refer to [the file `pkg_name/src/subscriber_node.py`](#) by using the following syntax:

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

You can also refer to a particular line:

This is done using the following parameters:

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

For example, you can refer to [the line containing “Initialize”](#) of [pkg_name/src/subscriber_node.py](#) by using the following syntax:

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

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

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

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

You cannot give `from_text` and `from_line` at the same time. You cannot give a `to...` without the `from....`

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

This is the source of the previous paragraph:

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

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

7.19. Putting code from the repository in line

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

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

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

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

and the result is the following automatically generated listing:

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

# Imports message type
from std_msgs.msg import String

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

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

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

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

Listing 2. `subscriber_node.py`

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

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

creates the following automatically generated listing:

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

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

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

Listing 3. [subscriber_node.py](#)

CHAPTER 8

Documentation style guide

This chapter describes the conventions for writing the technical documentation.

8.1. 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.

8.2. Style guide for the Duckietown documentation

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

8.3. 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
```

8.4. Frequently misspelled words

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

8.5. 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.)

8.6. 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 9

Knowledge graph



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

9.1. Formalization



1) Atoms



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

Each atom as a type, as follows:

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

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

2) Semantic graph of atoms



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

Each node is an atom.

The graph has four different types of edges:

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

3) Modules



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

Definition 3. (Module) A *module* is a directed graph, where the nodes are either atoms or other modules, and the edges can be of the four types described in [Subsection 9.1.2](#).

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.

9.2. Atoms properties

Each atom has the following properties:

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

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

A version is a tuple of attributes.

The attributes are:

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

Each atom version has:

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

TABLE 7. STATUS CODES

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

9.3. 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 4. `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 5. `calibration.it.md`

9.4. 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 8. SEMANTIC LINKS

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

9.5. How to describe modules



TODO: Define a micro-format for this.

CHAPTER 10

Translations



Note: This part is not implemented yet.

10.1. File organization



Translations are organized file-by-file.

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

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

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

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

10.2. Guidelines for English writers



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

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

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

10.3. File format



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

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

For example, if this is the original version:

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

Hopefully it will never happen.
```

Then the translated version should be:

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

Speriamo che non succeda.

PART 3

Operation manual - Duckiebot



CHAPTER 11

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.

11.1. 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.

11.2. 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 the 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 crossing an intersection).

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

CHAPTER 12

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 159 + Shipping Fees (minimal configuration C0)

Requires: 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 (?)



12.1. Bill of materials

TABLE 9. BILL OF MATERIALS

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

12.2. Chassis

We selected the Magician Chassis as the basic chassis for the robot ([Figure 12](#)).

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.



Figure 12. The Magician Chassis

12.3. Raspberry Pi 3 - Model B

The Raspberry Pi is the central computer of the Duckiebot. Duckiebots use Model B ([Figure 13](#)) (A1.2GHz 64-bit quad-core ARMv8 CPU, 1GB RAM), a small but powerful computer.



Figure 13. 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 14](#)).



Figure 14. 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 15](#). 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 15. The Heat Sinks

3) Class 10 MicroSD Card

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



Figure 16. The MicroSD card

4) Mirco SD card reader

A microSD card reader ([Figure 17](#)) 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 17. The Mirco SD card reader

12.4. 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 18](#)).



Figure 18. The Camera with Fisheye Lens

1) Camera Mount

The camera mount ([Figure 19](#)) serves to keep the camera looking forward at the right angle to the road (looking slightly down). The front cover is not essential.

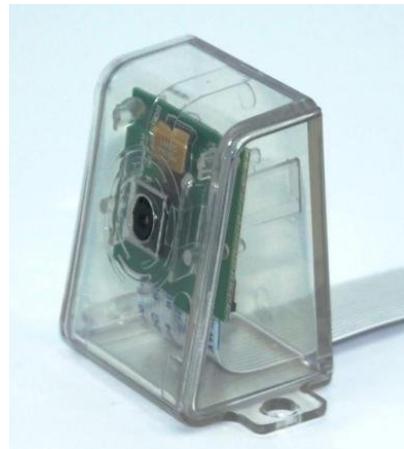


Figure 19. The Camera Mount

2) 300mm Camera Cable

A longer (300 mm) camera cable [Figure 20](#) make assembling the Duckiebot easier, allowing for more freedom in the relative positioning of camera and computational stack.

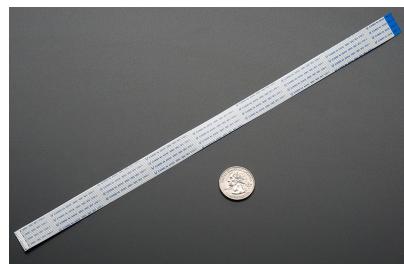


Figure 20. A 300 mm camera cable for the Raspberry Pi

12.5. DC Stepper Motor HAT

We use the DC Stepper motor HAT ([Figure 27](#)) to control the DC motors that drive the wheels. This item will require [soldering](#) to be functional.

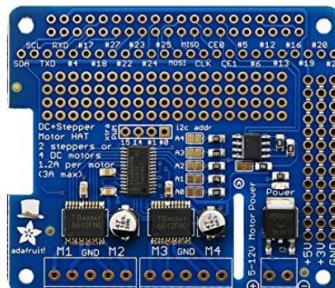


Figure 21. The Stepper Motor HAT

1) Stacking Headers

We use a long 20x2 stacking header ([Figure 22](#)) to connect the Raspberry Pi with the DC Stepper Motor HAT. This item will require [soldering](#) to be functional.



Figure 22. The Stacking Headers

12.6. Battery

The battery ([Figure 23](#)) 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.



Figure 23. The Battery

12.7. 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.

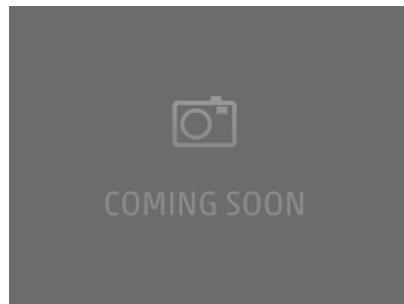


Figure 24. Standoffs, Nuts and Screws

12.8. Zip Tie

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



Figure 25. The zip ties

12.9. Configuration C0-w

1) Wireless Adapter (5 GHz)

The Edimax AC1200 EW-7822ULC 5 GHz wireless adapter ([Figure 26](#)) boosts the connectivity of the Duckiebot, especially useful in busy Duckietowns (e.g., classroom).



Figure 26. The Edimax AC1200 EW-7822ULC wifi adapter

12.10. 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 27](#)) does not include batteries.

Requires: 2 AA 1.5V batteries.



Figure 27. A Wireless Joypad

TODO: Add figure with 2 AA batteries

12.11. Configuration C0-d

1) Tiny 32GB USB Flash Drive

In configuration C0+d, the Duckiebot is equipped with a “external” hard drive ([Figure 28](#)). 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 28. The Tiny 32GB USB Flash Drive

CHAPTER 13

Soldering boards for C0

..

Assigned to: Shiying

Resources necessary:

Requires: Duckiebot ~~C0+wjd~~ parts. The acquisition process is explained in [Chapter 12](#). The configurations are described in [Chapter 11](#).

Requires: Time: ??? minutes

Results:

- ...

TODO: finish above

CHAPTER 14

Preparing the power cable for C0



In configuration C0 we will need a cable to power the DC motor HAT from the battery. The keen observer might have noticed that such a cable was not included in the [C0 Duckiebot parts](#) chapter. Here, we create this cable by splitting open any USB-A cable, identifying and stripping the power wires, and using them to power the DC motor HAT. If you are unsure about the definitions of the different Duckiebot configurations, read [Chapter 11](#).

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

Resources necessary:

- ─ **Requires:** One male USB-A to anything cable.
- ─ **Requires:** A pair of scissors.
- ─ **Requires:** A multimeter (only if you are not purchasing the [suggested components](#))
- ─ **Requires:** Time: 5 minutes

Results: One male USB-A to wires power cable

14.1. Step 1: Find a cable



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

If you have purchased the suggested components listed in [Chapter 12](#), you can use the longer USB cable contained inside the battery package ([Figure 29](#)), which will be used as an example in these instructions.



Figure 29. The two USB cables in the suggested battery pack.

Put the shorter cable back in the box, and open the longer cable ([Figure 30](#))



Figure 30. Take the longer cable, and put the shorter on back in the box.

14.2. Step 2: Cut the cable



Check before you continue

Make sure the USB cable is *unplugged* from any power source before proceeding.

Take the scissors and cut it ([Figure 31](#)) at the desired length from the USB-A port.

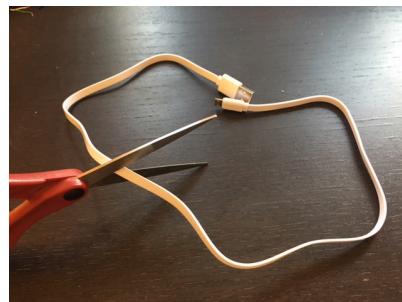


Figure 31. Cut the USB cable using the scissors.

The cut will look like in [Figure 32](#).



Figure 32. A cut USB cable.

14.3. Step 3: Strip the cable



Paying attention not to get hurt, strip the external white plastic. A way to do so without damaging the wires is shown in [Figure 33](#).



Figure 33. Stripping the external layer of the USB cable.

After removing the external plastic, you will see four wires: black, green, white and red ([Figure 34](#)).

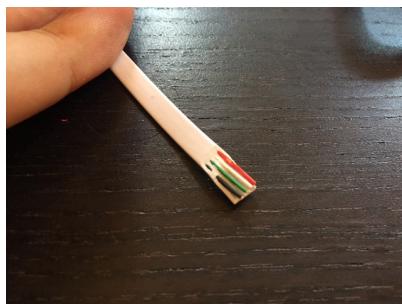


Figure 34. Under the hood of a USB-A cable.

Once the bottom part of the external cable is removed, you will have isolated the four wires ([Figure 35](#)).

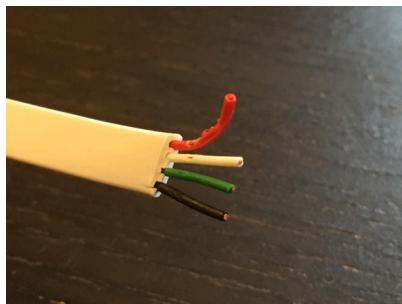


Figure 35. The four wires inside a USB-A cable.

14.4. Step 3: Strip the wires



Check before you continue

Make sure the USB cable is *unplugged* from any power source before proceeding.

Once you have isolated the wires, strip them, and use the scissors to cut off the data wires (green and white, central positions) ([Figure 36](#)).



Figure 36. Strip the power wires and cut the data wires.

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

14.5. Step 4: Find the power wires

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

If you are using a different USB cable, or are curious to verify that black and red actually are the power cables, take a multimeter and continue reading.

Plug the USB port inside a power source, e.g., the Duckiebot's battery. You can use some scotch tape to keep the cable from moving while probing the different pairs of wires with a multimeter. The voltage across the pair of power cables will be roughly twice the voltage between a power and data cable. The pair of data cables will have no voltage differential across them. If you are using the suggested Duckiebot battery as power source, you will measure around 5V across the power cables ([Figure 37](#)).

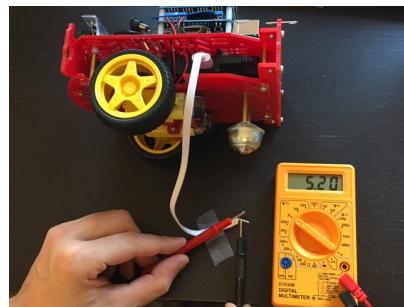


Figure 37. Finding which two wires are for power.

14.6. Step 5: Test correct operation

You are now ready to secure the power wires to the DC motor HAT power pins. To do so though, you need to have soldered the boards first. If you have not done so yet, read [Chapter 13](#).

If you have soldered the boards already, you may test correct functionality of the newly crafted cable. Connect the battery with the DC motor HAT by making sure you plug the black wire in the pin labeled with a minus: and the red wire to the plus: ([Figure 38](#)).

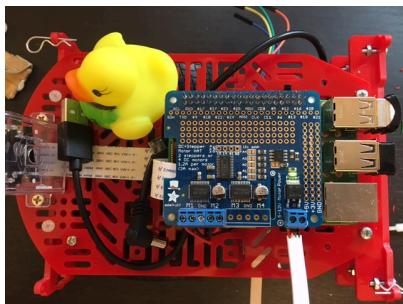


Figure 38. Connect the power wires to the DC motor HAT

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

CHAPTER 15

Assembling the Duckiebot

..

Assigned to: Shiying

Requires: Duckiebot `c0+wjd` parts. The acquisition process is explained in [Chapter 12](#).

Requires: Having soldered the `c0+wjd` parts. The soldering process is explained in [Chapter 13](#).

Requires: Having prepared the power cable. The power cable preparation is explained in [Chapter 14](#).

Requires: Time: about ??? minutes.

TODO: estimate time.

Results: An assembled Duckiebot in configuration `c0+wjd`.

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

CHAPTER 16

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.

16.1. 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
```

16.2. Burn the image to an SD card



Next, burn the image on to the SD card.

- This procedure is explained in [Section 98.2](#).

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).

16.3. 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

16.4. 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
```

16.5. Install Edimax driver



First, mark the kernel packages as not upgradeable:

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

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

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

16.6. Install ROS

Install ROS.

- The procedure is given in [Section 110.1](#).

16.7. 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
}
```

16.8. 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
uid=e9cef1bd-f6fb-4c5b-93cf-cca837ec35f2
type=wifi
permissions=
secondaries=
timestamp=1502254646

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

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

[ipv4]
dns-search=
method=auto

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

This is the file

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

Contents:

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

[wifi]
band=a
# Put the Edimax MAC address here
mac-address=74:DA:38:C9: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.

16.9. SSH server config

This enables the SSH server:

```
$ sudo systemctl enable ssh
```

16.10. 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`

16.11. 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`

16.12. Clean up

You can use the command `df -h` 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.

16.13. 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 100.3](#).

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
```

16.14. 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.)

16.15. 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 98.2](#); 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 98.3](#).

16.16. 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 108.1](#).

CHAPTER 17

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.

Results: A laptop ready to be used for Duckietown.

17.1. 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.

17.2. 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

17.3. Install ROS

Install ROS on your laptop.

- The procedure is given in [Section 110.1](#).

17.4. 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.

17.5. 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 6.4](#).

17.6. Passwordless sudo

Set up passwordless `sudo`.

- This procedure is described in [Section 16.11](#).

17.7. SSH and Git setup

1) Basic SSH config

Do the basic SSH config.

- The procedure is documented in [Section 100.3](#).

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 100.5](#).

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

Add the public key to your Github account.

- The procedure is documented in [Section 109.3](#).

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 107.3](#).

CHAPTER 18

Duckiebot Initialization



Assigned to: Andrea

Requires: An SD card of dimensions at least ??? 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 15](#).

Results: A Duckiebot that is ready to use.

What does it mean “ready to use”?

18.1. 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 98.2](#).

18.2. 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

18.3. 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.

18.4. 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
...
```

18.5. 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 106](#) for an introduction to Byobu.

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

18.6. (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.

enxb827eb1f81a4  no wireless extensions.

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

Make note of the name `wlan0 ABBCCDDEEFFGG`.

Look up the MAC address using the command:



```
$ ifconfig wlan0 ABBCCDDEEFFGG
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 ABBCCDDEEFFGG`”.
- 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.

18.7. 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.

18.8. 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
```

18.9. 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.

18.10. 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
```

18.11. Expand your filesystem



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



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

and then reboot



`$ sudo shutdown -r now`

once rebooted you can test whether this was successful by doing



`$ df -lh`

the output should give you something like:

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

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

18.12. 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 username
```

To set a password, use:



```
$ sudo passwd username
```

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

 \$ ssh `username@robot name`

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

1) Basic SSH config

Do the basic SSH config.

- The procedure is documented in [Section 100.3](#).

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 100.5](#).

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

Add the public key to your Github account.

- The procedure is documented in [Section 109.3](#).

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

 \$ ssh -T git@github.com

4) Local Git configuration

- This procedure is in [Section 107.3](#).

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 100.6](#). 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...

18.13. 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`.

18.14. Hardware check: camera

Check that the camera is connected using this command:



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

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

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

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



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

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

→ For instructions on how to use `scp`, see [Subsection 102.1.1](#).

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 19

Software setup and RC remote control



Assigned to: Andrea

Requires: Laptop configured, according to [Chapter 17](#).

Requires: You have configured the Duckiebot. The procedure is documented in [Chapter 18](#).

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 109](#).

Results: You can run the joystick demo.



19.1. 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 109](#).

Symptom: Other weird errors.

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

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



19.2. 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 110.6](#).

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

19.3. Add your vehicle to the scuderia file

Add your vehicle to the scuderia file.

→ See [Section 113.2](#).

19.4. 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 18.12](#).

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.

19.5. 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 19.4](#).

19.6. 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 20

Reading from the camera



Requires: You have configured the Duckiebot. The procedure is documented in [Chapter 18](#).

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

TODO: put reference

Results: You know that the camera works under ROS.

20.1. Check the camera hardware



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

→ The procedure is documented in [Section 18.14](#).

20.2. Create two windows



On the laptop, create two Byobu windows.

→ A quick reference about Byobu commands is in [Chapter 106](#).

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.

20.3. First window: launch the camera nodes



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

Activate ROS:

 \$ `source environment.sh`

Run the launch file called `camera.launch`:

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

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

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

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

* For more information about `roslaunch` and “launch files”, see [Section 110.3](#).

20.4. 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 110.5](#).

You should see the following topics:

```
/robot_name/camera_node/camera_info
/robot_name/camera_node/image/compressed
/robot_name/camera_node/image/raw
/rosvout
/rosvout_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 22](#), you will learn how to visualize the image stream on the laptop using `rviz`.

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

CHAPTER 21

RC control launched remotely

Assigned to: Andrea

Requires: You can run the joystick demo from the Raspberry Pi. The procedure is documented in [Chapter 19](#).

Results: You can run the joystick demo from your laptop.

21.1. 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

21.2. 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 19.1](#).

Then, you should build the repository.

- This procedure is documented in [Section 19.2](#).

21.3. 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 19.3](#).

21.4. 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 110.3](#).

21.5. 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 110.2](#).

21.6. 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 100.3](#).

CHAPTER 22

RC+camera remotely



Assigned to: Andrea

Requires: You can run the joystick demo remotely. The procedure is documented in [Chapter 21](#).

Requires: You can read the camera data from ROS. The procedure is documented in [Chapter 20](#).

Requires: You know how to get around in Byobu. You can find the Byobu tutorial in [Chapter 106](#).

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

22.1. Assumptions



We are assuming that the joystick demo in [Chapter 21](#) worked.

We are assuming that the procedure in [Chapter 20](#) succeeded.

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

22.2. Terminal setup



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

→ A quick reference about Byobu commands is in [Chapter 106](#).

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

22.3. First window: launch the joystick demo



In the first window, launch the joystick remotely using the same procedure in [Section 21.4](#).



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

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

22.4. Second window: launch the camera nodes

In the second window, we will launch the nodes that control the camera. The launch file is called `camera.launch`:



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

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

22.5. 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
```

22.6. 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 110.4](#).

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.

22.7. Proper shutdown procedure



To stop the nodes: You can stop the node by pressing `Ctrl`-`C` on the terminal where `ros.launch` 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 23

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.

Requires: Time: ??? minutes.

Results: You will know about some useful shortcuts.



23.1. 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.



23.2. 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 24

Wheel calibration



Assigned to: Andrea

CHAPTER 25

Camera calibration



CHAPTER 26

Taking a log

| Assigned to: Andrea

PART 4
Operation manual - Duckietowns

•

CHAPTER 27

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 31](#) 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: [Assembling](#) a Duckietown.

TODO: Figure out costs

27.1. Bill of materials

TABLE 10. BILL OF MATERIALS FOR DUCKIETOWN

<u>Duckies</u>	USD 17/100 pieces
<u>Floor Mats</u>	USD 37.5/6 pieces (24 sqft)
<u>Duct tape - Red</u>	USD 8.50/roll
<u>Duct tape - White</u>	USD 8.50/roll
<u>Duct tape - Yellow</u>	USD 8/roll
<u>Traffic signs</u>	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

27.2. Duckies

Duckies ([Figure 39](#)) are essential yet non functional.

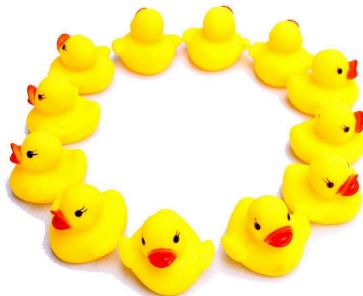


Figure 39. The Duckies

27.3. Floor Mats

The floor mats ([Figure 40](#)) 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 40. 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 31](#).

27.4. Duck Tape

We use duck (duct) tape of different colors ([Figure 41](#)) 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 41. The Duck Tapes

To verify how much tape you need for each road segment type, see [Chapter 31](#).

27.5. Traffic Signs

Traffic signs ([Figure 42](#)) inform Duckiebots on the map of Duckietown, allowing them to make driving decisions.



Figure 42. The Signs

Depending on the chose road topography, the number of necessary road signal will vary. To design your Duckietown, see [Chapter 31](#).

CHAPTER 28

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 31](#).

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

| Requires: Time: ?? days (average shipping time)

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

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

TODO: Estimate time and costs

28.1. Bill of materials

TABLE 11. BILL OF MATERIALS FOR TRAFFIC LIGHT

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

TODO: Complete table

28.2. Raspberry Pi

([Figure 43](#)) are essential yet non functional.

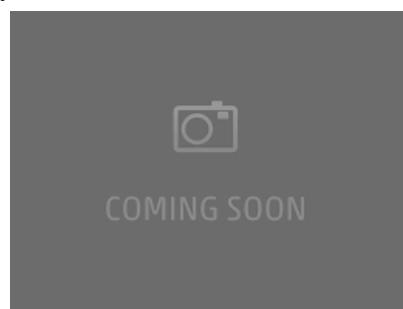


Figure 43. The placeholder

CHAPTER 29

Duckietown Assembly

..

Assigned to: Shiying

CHAPTER 30

Traffic lights Assembly



Assigned to: Shiying

CHAPTER 31

The Duckietown specification

Assigned to: Liam?

31.1. Topology

1) Topology constraints

31.2. Signs placement

PART 5

Operation manual - Duckiebot with LEDs



CHAPTER 32

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 11](#).

In general, keep in mind that:

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

Requires: - A Duckiebot in `c0+wjd` configuration. - Cost: USD 77 + Bumpers manufacturing solution - Time: 21 Days (LED board manufacturing and shipping time)

Results: - A kit of parts ready to be assembled in a `c1` configuration Duckiebot.

Next Steps: - After receiving these components, you are ready to do some [soldering](#) before [assembling](#) your `c1` Duckiebot.



32.1. Bill of materials

TABLE 12. BILL OF MATERIALS

<u>LEDs</u>	USD 10
<u>LED HAT</u>	USD 28.20 for 3 pieces
<u>Power Cable</u>	USD 7.80
<u>20 Female-Female Jumper Wires (300mm)</u>	USD 8
<u>Male-Male Jumper Wire (150mm)</u>	USD 1.95
<u>PWM/Servo HAT</u>	USD 17.50
<u>Bumpers</u>	TBD (custom made)
<u>40 pin female header</u>	USD 1.50
<u>5 4 pin female header</u>	USD 0.60/piece
<u>2 16 pin male header</u>	USD 0.61/piece
<u>12 pin male header</u>	USD 0.48/piece
<u>3 pin male header</u>	USD 0.10/piece
<u>2 pin female shunt jumper</u>	USD 2/piece
<u>5 200 Ohm resistors</u>	USD 0.10/piece
<u>10 130 Ohm resistors</u>	USD 0.10/piece
Total for <code>c0+wjd</code> configuration	USD 212
Total for <code>c1</code> components	USD 77 + Bumpers
Total for <code>c1</code> configuration	USD 299+Bumpers

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

32.2. LEDs

The Duckiebot is equipped with 5 RGB LEDs ([Figure 44](#)). 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.

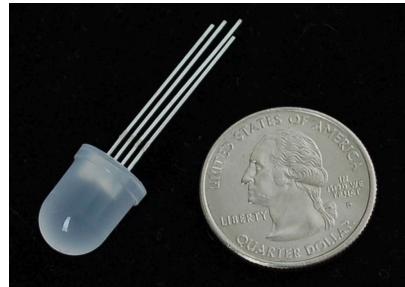


Figure 44. The RGB LEDs

1) LED HAT

The LED HAT ([Figure 45](#)) 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.

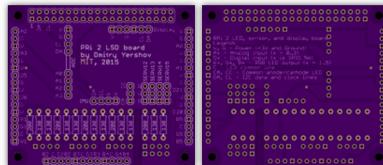


Figure 45. The LED HAT

2) PWM/Servo HAT

The PWM/Servo HAT ([Figure 46](#)) 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 46. The PWM-Servo HAT

3) Power Cable

To power the PWM/Servo HAT from the battery, we use a short (30cm) angled male USB-A to 5.5/2.1mm DC power jack cable ([Figure 47](#)).



Figure 47. The 30cm angled USB to 5.5/2.1mm power jack cable.

4) Male-Male Jumper Wires

The Duckiebot needs one male-male jumper wire ([Figure 48](#)) to power the DC Stepper Motor HAT from the PWM/Servo HAT.

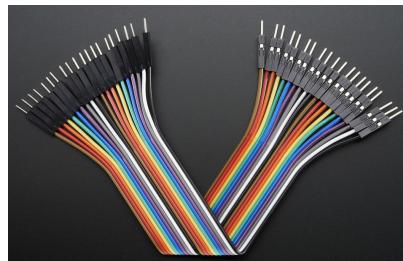


Figure 48. Premier Male-Male Jumper Wires

5) Female-Female Jumper Wires

20 Female-Female Jumper Wires ([Figure 49](#)) are necessary to connect 5 LEDs to the LED HAT.

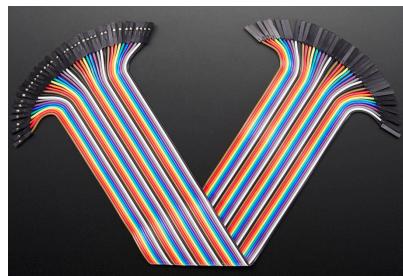


Figure 49. Premier Female-Female Jumper Wires

32.3. 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

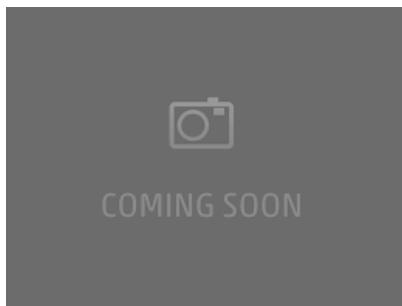


Figure 50. The Bumpers

32.4. 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](#).

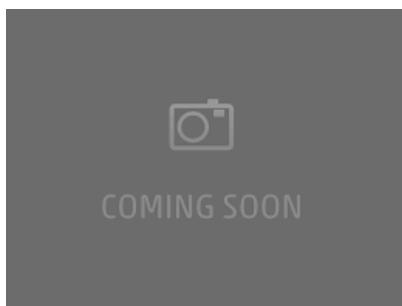


Figure 51. The Headers

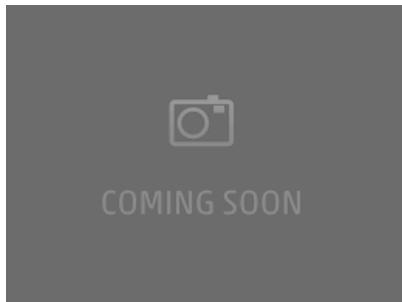


Figure 52. The Resistors

TODO: Missing figures.

CHAPTER 33

Soldering boards for C1

..

Assigned to: Shiying

Resources necessary:

Requires: - Duckiebot C1 parts. The acquisition process is explained in [Chapter 32](#).
The configurations are described in [Chapter 11](#).

Requires: - Time: ??? minutes

Results:

- ...

TODO: finish above

CHAPTER 34

Assembling the Duckiebot C1



Assigned to: Shiyiing

Requires: Duckiebot C1 parts. The acquisition process is explained in [Chapter 32](#).

Requires: Soldering C1 parts. The soldering process is explained in [Chapter 33](#).

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 35

C1 (LEDs) setup

..

Assigned to: Andrea

PART 6

Theory chapters



These are the theory chapters.

CHAPTER 36

Chapter template

Theory chapters benefit from a standardized exposition. Here, we define the template for these chapters. Rememeber to check [Chapter 7](#) for a comprehensive and up-to-date list of Duckiebook supported features.

36.1. Example Title: PID control

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

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

Knowledge necessary:

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

Requires: Terminology: [autonomy overview](#)

Requires: System Modeling: [basic kinematics](#), [basic dynamics](#), [linear algebra](#), [State space representations](#), [Linear Time Invariant Systems](#)

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

Recommended: Definitions of Stability, Performances and Robustness: [\[5\]](#), ...

Recommended: observability/detectability and controllability/reachability: [\[5\]](#)

Recommended: Discrete time PID: [\[5\]](#)

Recommended: Bode diagrams: [\[5\]](#)

Recommended: Nyquist plots: [\[5\]](#)

Recommended: [...]

36.2. Problem Definition

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

Let:

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

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

([Duckiebot Modeling](#)) that:

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{bmatrix}$$

$$B = [b_1 \ \dots \ b_m]^T$$

$$C = [c_1 \ \dots \ c_p]$$

$$D = 0.$$

[...]

Remember you can use the `problem` environment of *LATEX* to formally state a problem:

Problem 2. (PID) Given a system [\(1\)](#) and measurements of the output

`Undefined control sequence \textasciitilde`, find a set of PID co-

efficients that meet the specified requirements for: - stability, - performance, - robustness.

as shown in ([Figure 53](#)).

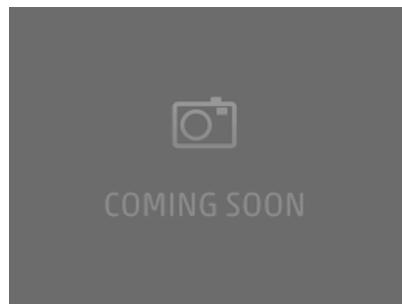


Figure 53. A classical block diagram for PID control. We like to use a lot of clear figures in the Duckiebook.

36.3. Introduced Notions

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

Definition 4. (Reference signals) A reference signal $\tilde{y}_t \in \mathcal{L}_2(\mathcal{T})$ is ...

[Definition 4](#) is very important.

Check before you continue

Insert ‘random’ checks to keep the reader’s attention up:

if you can’t be woken up in the middle of the night and rememeber the definition of $\mathcal{L}_2(\cdot)$, read: [\[5\]](#)

Definition 5. (Another definition) Lorem

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

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

Corem

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

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

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

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

Overshoot, Module at resonance, Settling Time, Rising Time

[...]

example

This is a 'think about it' interrupt, used as attention grabber:

When a Duckiebot 'overshoots', it means that [...] and the following will happen [...].

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

And finally, this is how you save the world, in theory.

36.4. Examples

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

1) Theoretical Examples

More academic examples

T-Example 1:

Imagine a spring-mass-damper system...

T-Example M:

[...]

2) Implementation Examples

More Duckiebot related examples

I-Example 1:

I-Example M:

[...]

36.5. Pointers to Exercises

Here we just add references to the suggested exercises, defined in the appropriate [exer-](#)

[cise chapters.](#)

36.6. Conclusions

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

36.7. Next Steps

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

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

- * *learn all there is to know about PID: [\[5\]](#)*
- * *become a linear algebra master: [Matrix cookbook](#)*

36.8. References

Do not include a reference chapter. References are automatically compiled to [the Bibliography Section](#).

Author: Jacopo Maintainer: Jacopo Point of Contact: Jacopo

CHAPTER 37

Symbols and conventions

Assigned to: Andrea

37.1. 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 .

37.2. Table of symbols

Here are some useful symbols.

TABLE 13. SPACES

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

States and poses:

TABLE 14. POSES AND STATES

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

CHAPTER 38

Linear algebra



TODO: This Section is work in progress.

Assigned to: Jacopo

Linear algebra provides the set of mathematical tools to (a) study linear relationships and (b) describe linear spaces. It is a field of mathematics with important ramifications.

Linearity is an important concept because it is powerful in describing the input-output behaviour of many natural phenomena (or *systems*). As a matter of fact, all those systems that cannot be modeled as linear, still can be approximated as linear to gain an intuition, and sometimes much more, of what is going on.

So, in a way or the other, linear algebra is a starting point for investigating the world around us, and Duckietown is no exception.

Knowledge necessary:

Requires: Real numbers are complex for you?: Number theory [addref](#)

Requires: \forall is a typo for A and \in are Euros? Mathematical symbolic language: [ad-dref](#)

TODO: find appropriate references and fill in above

38.1. Problem Definition



In this section we discuss vectors, matrices and linear spaces, along with their properties. Before introducing the these arguments, we need to formally define what we mean by linearity. The word *linear* comes from the latin *linearis*, which means *pertaining to or resembling a line*. You should recall that a line is represented by an equation like $y = mx + q$, but here we intend linearity as a property of maps, so there is a little more to linearity than lines (although lines *are* linear maps indeed). To avoid confusions, let us translate the concept of linearity in mathematical language.

First, let us define a *function*, as a mapping between sets.

Definition 6. (Set) A set $\mathbb{X} = \{x_1, x_2, \dots\}$ is a well-defined collection of distinct *elements*, or *members* of the set, $x_i, i = 1, 2, \dots$. For the time being, we assume elements to (complex) numbers.

Definition 7. (Function) A function $f: \mathbb{X} \rightarrow \mathbb{Y}$ is a mapping between the sets \mathbb{X} and \mathbb{Y} . For every input element $x \in \mathbb{X}$, the mapping will produce an output $y = f(x) \in \mathbb{Y}$.

Recommended: Functions can be classified by the nature of the relationship between inputs and outputs in: *injective*, *surjective* or *bijective* [add-ref](#).

TODO: add references

Definition 8. (Linearity) A function $f: \mathbb{X} \rightarrow \mathbb{Y}$ is linear when, $\forall x_i \in \mathbb{X}, i = \{1, 2\}$, and $\forall a \in \mathbb{R}$:

$$f(ax_1) = af(x_1), \quad \text{and:} \quad (1)$$

$$f(x_1 + x_2) = f(x_1) + f(x_2) \quad (2)$$

Condition (1) is referred to as the property of *homogeneity* (of order 1), while condition (2) is referred to as *additivity*.

Remark 2. (Superposition Principle) Conditions (1) and (2) can be merged to express the same meaning through:

$$f(ax_1 + bx_2) = af(x_1) + bf(x_2), \forall x_i \in \mathbb{X}, i = \{1, 2\}, \forall a, b \in \mathbb{R}. \quad (3)$$

This equivalent condition (3) is instead referred to as *superposition principle*, which unveils the bottom line of the concept of linearity: adding up (equivalently, scaling up) inputs results in an added up (equivalently, scaled up) output.

38.2. Vectors

Let n belong to the set of natural numbers \mathbb{N} , i.e., $n \in \mathbb{N}$, and let $a_i \in \mathbb{R}, i = \{1, \dots, n\}$ be real coefficients. While \mathbb{R} is the set of real numbers, \mathbb{R}^n is the set of all n -tuples of real numbers.

Definition 9. (Vector and components) An n -dimensional *vector* is an n -tuple:

$$\mathbf{v} = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \in \mathbb{R}^{n \times 1} \equiv \mathbb{R}^n, \quad (4)$$

of components $v_1, \dots, v_n \in \mathbb{R}$.

You can imagine a vector as a “directional *number*”, or an arrow that starts a certain point and goes in a certain direction (in \mathbb{R}^n). In this representation, the *number* is the length of the arrow, or the *modulus* of the vector, and it can be derived through the vector’s components.

Definition 10. (Length of a vector) We define the length, or *modulus*, of a vector $\mathbf{v} \in \mathbb{R}^n$ as:

$$\|\mathbf{v}\| = \sqrt{v_1^2 + \dots + v_n^2} \in \mathbb{R}. \quad (5)$$

Remark 3. (2-norm) Generally speaking, it is not always possible to define the length of a vector (addref). But when it is possible (e.g., [Hilbert spaces](#)), and in Duckietown it always is, there are many ways to define it. The most common and intuitive definition is the *Euclidian*- or *2-norm*, which is defined above in (5).

- Definition of:
- norms
- p-norm, ∞ -norm
- unit vector

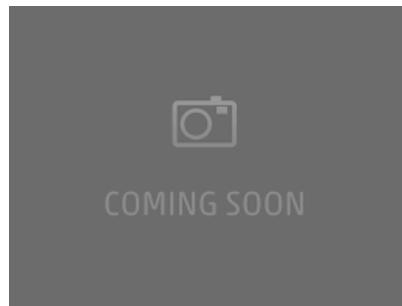


Figure 54. A vector and its components.

1) Vector algebra

- sum of vectors
- dot product / inner product
- cross product
-

Orthogonality between vectors:

38.3. Matrices

Definitions:

- matrix dimensions
- flat and tall matrix
- adjoint matrix
- inverse matrix
- rank of a matrix
- condition number of a matrix (?)
- identity matrix
- null matrix
- diagonal matrix
- symmetric matrix
- unit matrix
- trace of a matrix

1) Matrix algebra

- sum of matrices
- product of matrices
- matrix transpose
- matrix scalar product
- matrix Hadamard product
- matrix concatenation
- matrix-vector product
- matrix power
- matrix exponential

Determinant:

- 2x2
- 3x3

- $n \times n$

Inverse:

- general expression

Left and Right Inverse (topic for advanced-linear-algebra?):

- what if the matrix is not square? (topic for advanced-linear-algebra?)
- Moore-Penrose pseudo-inverse

Eigenvalues and Eigenvectors:

- for square matrices
- for rectangular matrices (topic for advanced-linear-algebra?)
- singular value decomposition SVD (topic for advanced-linear-algebra?)

38.4. Matrix as representation of linear (vector) spaces

- linear system to matrix representation
- linearly dependent and independent spaces

1) Fundamental spaces

- Null space
- Range/image

2) Preferred spaces (matrix diagonalization)

- show how to diagonalize matrices and why it is relevant (it will come in handy for state space representation chapter chapter)

38.5. Examples

1) Theoretical Examples

Calculate a (square) Matrix Inverse:

Find eigenvalues and eigenvectors:

Find range and null spaces of a matrix:

2) Implementation Examples

Inverting a well conditioned matrix:

Inverting a ill conditioned matrix:

38.6. Pointers to Exercises

Here we just add references to the suggested exercises, defined in the appropriate [exercise chapters](#).

38.7. Conclusions

In this section we have defined the fundamental concept of linearity and introduced the mathematical tools pertaining to it, in particular vectors and matrices. Moreover,

we have introduced their properties and interpretations as linear spaces.

We have found that matrices are a very convenient way to represent linear spaces, and that the properties of the matrices such as eigenvalues and eigenvectors have important implications in characterizing these spaces.

These tools are useful because they are at the foundation of *modeling* of natural phenomena. Modeling will be invaluable in understanding the behaviour of systems, and a powerful tool to *predict* future behaviours of the system, and *control* them when needed.

We have learned that ...

38.8. Next Steps



- linearization of non linear equations
- state space representations
- basic kinematics
- basic dynamics

* [\[13\]](#)

* [Matrix cookbook](#)

Author: Jacopo Maintainer: Jacopo Point of Contact: Jacopo

CHAPTER 39

Probability basics



Assigned to: Liam?

CHAPTER 40

Dynamics



Assigned to: Jacopo

TODO: this is a repetition of [Chapter 50](#).

CHAPTER 41

Coordinate systems



Assigned to: Falcon

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

CHAPTER 42

Reference frames



Assigned to: TBD

Check before you continue

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

TODO: Provide a basic description of reference frames. This section is dependent upon the [coordinate systems](#) chapter and could be moved there.

CHAPTER 43

Transformations



Assigned to: TBD

Check before you continue

Required Reading: The following assumes a working familiarity with 2D and 3D Cartesian reference frames. If you are not familiar with Cartesian reference frames, please read the [chapter on reference frames](#).

TODO: Provide a basic description of transformations (rotations and translations) in 2D and 3D Cartesian reference frames.

CHAPTER 44

Autonomy overview

Assigned to: Liam

In this chapter we will introduce some basic concepts ubiquitous in autonomous vehicle navigation.

44.1. Basic Building Blocks of Autonomy

The minimal basic backbone processing pipeline for autonomous vehicle navigation is shown in [Figure 55](#).

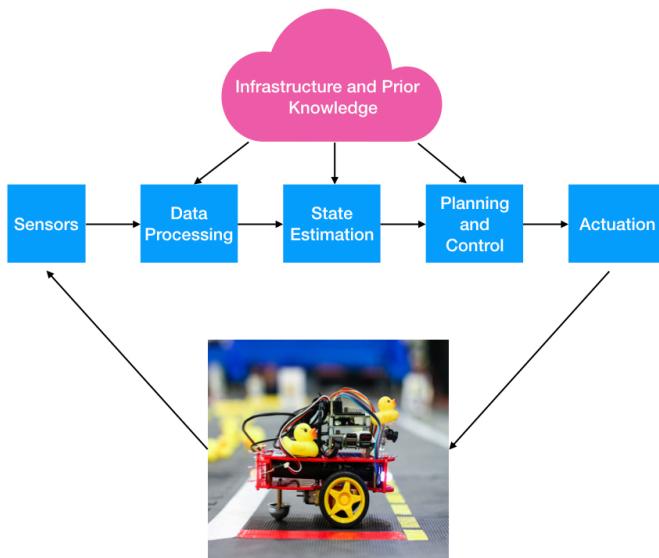


Figure 55. The basic building blocks of any autonomous vehicle

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

1) Sensors

Definition 11. (Sensor) A *sensor* is a device that or mechanism that is capable of generating a measurement of some external physical quantity

In general, sensors have two major types. *Passive* sensors generate measurements without affecting the environment that they are measuring. Examples include inertial sensors, odometers, GPS receivers, and cameras. *Active* sensors emit some form of energy into the environment in order to make a measurement. Examples of this type of

sensor include Light Detection And Ranging (LiDAR), Radio Detection And Ranging (RaDAR), and Sound Navigation and Ranging (SoNAR). All of these sensors emit energy (from different spectra) into the environment and then detect some property of the energy that is reflected from the environment (e.g., the time of flight or the phase shift of the signal)

2) Raw Data Processing

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

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

We will distinguish between two fundamentally types of calibrations.

Definition . (Intrinsic Calibration) {def:intrinsic-calibration} An *intrinsic calibration* is required to determine sensor-specific parameters that are internal to a specific sensor.

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

Check before you continue

For more information about reference frames check out [Chapter 42](#)

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

Once we have properly calibrated data in some meaningful units, we often do some preprocessing to reduce the overall size of the data. This is true particularly for sensors that generate a lot of data, like cameras. Rather than deal with every pixel value generated by the camera, we will process an image to generate feature-points of interest. In “classical” computer vision many different feature descriptors have been proposed (Harris, BRIEF, BRISK, SURF, SIFT, etc), and more recently Convolutional Neural Networks (CNNs) are being used to learn these features.

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

TODO: add a picture with basic feature detections

3) State Estimation

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

4) Planning and Control

State->control

5) Actuation

Control applied to vehicle

6) Infrastructure and Prior Information

44.2. Advanced Building Blocks of Autonomy

CHAPTER 45

Autonomy architectures



Assigned to: Andrea



45.1. Contracts

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

CHAPTER 46

Representations



Assigned to: Matt

Check before you continue

Required Reading: The following assumes working knowledge of 2D and 3D Cartesian coordinate systems, reference frames, and coordinate transformations. If you are not familiar with these topics, please see the chapters on [coordinate systems](#), [reference frames](#), and [coordinate transformations](#).

Discuss:

- Introduction to the notion of *state* as a sufficient statistic that represents the agent (robot) and environment.
- Describe qualities: sufficient statistic; compact (i.e., not conveying unnecessary information); and readily interpretable.
- Define notion of *static* and *dynamic* states.
- Provide examples of robot and environment states.

The planning and control capabilities necessary for autonomy rely upon shared representations of the agent (i.e., the robot) and the environment in which it operates. Referred to as the *state*, this representation consists of a compilation of all knowledge about the robot and its environment that is *sufficient* both to perform a particular task as well as to predict the future. Ignoring prior information, this knowledge is often extracted from the robot's multimodal sensor streams (e.g., wheel encoders and cameras). Consequently, a natural choice is to formulate the state as the collection of all of the measurements that the robot acquires over time. Indeed, the use of low-level sensor measurements as the representation of the state has a long history in robotics and artificial intelligence (cite) and has received renewed attention of-late (cite).

However, while measurement history is a sufficient representation of the robot and its operating environment, several limitations limit its utility for planning and control. First, measurement history is redundant within individual and across successive observations. Second, the observations contain a large amount of unnecessary information (e.g., pixel intensities associated with clouds are not useful for self-driving vehicles). Third, measurement history is very inefficient: its size grows linearly with time (i.e., as the robot acquires new measurements), and it may be computationally intractable to access and process such a large amount of data. This motivates the desire for a *minimal* representation that expresses knowledge that is both necessary and sufficient for the robot to perform a given task. More concretely, we will consider parameterized (symbolic) formulations of the state and will prefer representations that involve as small a number of parameters as possible, subject to the constraints imposed by the task.

46.1. Preliminaries



I've created (currently empty) chapters for each of the following

- Coordinate systems)
- Reference frames
- Coordinate transformations

46.2. Robot Representations

Define the notion of:

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

Discuss specific robot state representation for Duckietown.

46.3. 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 47

Software architectures and middlewares



Assigned to: Andrea

CHAPTER 48

Modern signal processing

Assigned to: Andrea

CHAPTER 49

Basic Kinematics



Assigned to: Jacopo

CHAPTER 50

Basic Dynamics

..

Assigned to: Jacopo

CHAPTER 51

Odometry Calibration



Assigned to: Jacopo

CHAPTER 52

Computer vision basics



Assigned to: Matt

CHAPTER 53

Illumination invariance



Assigned to: Matt

CHAPTER 54

Line Detection

..

Assigned to: Matt

CHAPTER 55

Feature extraction



Assigned to: Matt

CHAPTER 56

Place recognition

..

Assigned to: Matt

CHAPTER 57

Filtering 1



Assigned to: Liam

CHAPTER 58

Filtering 2

..

Assigned to: Liam

CHAPTER 59

Mission planning



| Assigned to: ETH

CHAPTER 60

Planning in discrete domains

..

Assigned to: ETH

CHAPTER 61

Motion planning



Assigned to: ETH

CHAPTER 62

RRT

Assigned to: ETH

..

CHAPTER 63

Feedback control



Assigned to: Jacopo

CHAPTER 64

PID Control

..

Assigned to: Jacopo

CHAPTER 65

MPC Control



Assigned to: Jacopo

CHAPTER 66

Object detection

..

Assigned to: Nick and David

CHAPTER 67

Object classification



Assigned to: Nick and David

CHAPTER 68

Object tracking

..

Assigned to: Nick and David

CHAPTER 69

Reacting to obstacles



Assigned to: Jacopo

CHAPTER 70

Semantic segmentation



Assigned to: Nick and David

CHAPTER 71

Text recognition



Assigned to: Nick

CHAPTER 72

SLAM - Problem formulation



Assigned to: Liam

CHAPTER 73

SLAM - Broad categories

Assigned to: Liam

CHAPTER 74

VINS

Assigned to: Liam

CHAPTER 75

Advanced place recognition



Assigned to: Liam

CHAPTER 76

Fleet level planning (placeholder)

..

Assigned to: ETH

CHAPTER 77

Fleet level planning (placeholder)



| Assigned to: ETH

CHAPTER 78

Bibliography



- [1] [Jacopo Tani](#), [Liam Paull](#), [Maria Zuber](#), [Daniela Rus](#), [Jonathan How](#), [John Leonard](#), and Andrea Censi. Duckietown: an innovative way to teach autonomy. In *EduRobotics 2016*. Athens, Greece, December 2016. [pdf](#)
- [2] [Liam Paull](#), [Jacopo Tani](#), Heejin Ahn, Javier Alonso-Mora, Luca Carlone, Michal Cap, Yu Fan Chen, Changhyun Choi, Jeff Dusek, Daniel Hoehener, Shih-Yuan Liu, Michael Novitzky, Igor Franzoni Okuyama, Jason Pazis, Guy Rosman, Valerio Varricchio, Hsueh-Cheng Wang, Dmitry Yershov, Hang Zhao, Michael Benjamin, [Christopher Carr](#), [Maria Zuber](#), [Sertac Karaman](#), [Emilio Frazzoli](#), [Domitilla Del Vecchio](#), [Daniela Rus](#), [Jonathan How](#), [John Leonard](#), and Andrea Censi. Duckietown: an open, inexpensive and flexible platform for autonomy education and research. In *IEEE International Conference on Robotics and Automation (ICRA)*. Singapore, May 2017. [pdf](#)
- [3] Bruno Siciliano and Oussama Khatib. *Springer Handbook of Robotics*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.
- [4] Tosini, G., Ferguson, I., Tsubota, K. *Effects of blue light on the circadian system and eye physiology*. Molecular Vision, 22, 61–72, 2016 ([online](#)).
- [5] Albert Einstein. Zur Elektrodynamik bewegter Körper. (German) On the electrodynamics of moving bodies. *Annalen der Physik*, 322(10):891–921, 1905. [DOI](#)
- [13] Ivan Savov. Linear algebra explained in four pages. https://minireference.com/static/tutorials/linear_algebra_in_4_pages.pdf, 2017. Online; accessed 23 August 2017.
- [12] Kaare Brandt Petersen and Michael Syskind Pedersen. The matrix cookbook. [.pdf](#)

PART 7
Exercises

• •

These are the exercises.

CHAPTER 79

ROS Exercises

Progression of ROS skills:

79.1. Parameters

- Reading parameters
- Dynamic modification of parameters

79.2. Running from a log

- Running from a log

79.3. Unit tests

- Unit tests
- Integration with ROS tests

79.4. Analytics

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

79.5. Visualization

- Making a plot displayed using topic
- Using ROS parameters

CHAPTER 80

Line detection

..

CHAPTER 81

Data processing

..

CHAPTER 82

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 83

Ubuntu packaging with APT

83.1. apt install

TODO: to write

83.2. apt update

TODO: to write

1) apt dist-upgrade

TODO: hold back packages

83.3. apt-key

TODO: to write

83.4. apt-mark

TODO: to write

83.5. add-apt-repository

TODO: to write

83.6. wajig

TODO: to write

83.7. dpigs

TODO: to write

CHAPTER 84

GNU/Linux general notions



Assigned to: Andrea



84.1. Background reading

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

CHAPTER 85

Every day Linux

85.1. cd

TODO: to write

85.2. sudo

TODO: to write

85.3. ls

TODO: to write

85.4. cp

TODO: to write

85.5. mkdir

TODO: to write

85.6. touch

TODO: to write

85.7. reboot

TODO: to write

85.8. shutdown

TODO: to write

85.9. rm

TODO: to write

CHAPTER 86

Users

86.1. passwd

TODO: to write

CHAPTER 87

UNIX tools

87.1. `cat`

TODO: to write

87.2. `tee`

TODO: to write

87.3. `truncate`

TODO: to write

CHAPTER 88

Linux disks and files

88.1. `fdisk`

TODO: to write

88.2. `mount`

TODO: to write

88.3. `umount`

TODO: to write

88.4. `losetup`

TODO: to write

88.5. `gparted`

TODO: to write

88.6. `dd`

TODO: to write

88.7. `sync`

TODO: to write

88.8. `df`

TODO: to write

CHAPTER 89

Other administration commands

89.1. visudo

TODO: to write

89.2. update-alternatives

TODO: to write

89.3. udevadm

TODO: to write

89.4. systemctl

TODO: to write

CHAPTER 90

Make

90.1. make

TODO: to write

CHAPTER 91

Python-related tools

91.1. virtualenv

TODO: to write

91.2. pip

TODO: to write

CHAPTER 92

Raspberry-PI commands

92.1. raspi-config

TODO: to write

92.2. vcgencmd

TODO: to write

92.3. raspistill

TODO: to write

92.4. jstest

TODO: to write

92.5. swapon

TODO: to write

92.6. mkswap

TODO: to write

CHAPTER 93

Users and permissions

93.1. chmod

TODO: to write

93.2. groups

TODO: to write

93.3. adduser

TODO: to write

93.4. useradd

TODO: to write

CHAPTER 94

Downloading

94.1. curl

TODO: to write

94.2. wget

TODO: to write

94.3. sha256sum

TODO: to write

94.4. xz

TODO: to write

CHAPTER 95

Shells and environments

95.1. source

TODO: to write

95.2. which

TODO: to write

95.3. export

TODO: to write

CHAPTER 96

Other misc commands

96.1. pgrep

TODO: to write

96.2. npm

TODO: to write

96.3. nodejs

TODO: to write

96.4. ntpdate

TODO: to write

96.5. chsh

TODO: to write

96.6. echo

TODO: to write

96.7. sh

TODO: to write

96.8. fc-cache

TODO: to write

CHAPTER 97

Linux resources usage

97.1. Measuring CPU usage using htop

You can use `htop` to monitor CPU usage.

```
$ sudo apt install htop
```

TODO: to write

97.2. Measuring I/O usage using iotop

Install using:

```
$ sudo apt install iotop
```

TODO: to write

97.3. How fast is the SD card?

→ [Section 98.1.](#)

CHAPTER 98

SD Cards tools

98.1. 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.

98.2. 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`.

98.3. How to shrink an image

Requires:

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

Results:

- A shrunk image.

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

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

```
💻 $ sudo apt install gparted
```

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

```
💻 $ sudo fdisk -l image file
```

It should give you something like:

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

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

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

and then run `gparted`:

💻 `$ sudo gparted /dev/loop0`

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

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

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

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

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

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

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

```
Command (m for help): enter w
```

```
The partition table has been altered.
```

```
Calling ioctl() to re-read partition table.
```

```
Re-reading the partition table failed.: Invalid argument
```

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

Disregard the final error.

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

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

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

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

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

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

```
 $ xz image file
```

CHAPTER 99

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:

99.1. hostname



TODO: to write

99.2. Visualizing information about the network



1) ping: are you there?



TODO: to write

2) ifconfig



TODO: to write

`$ ifconfig`

CHAPTER 100

Accessing computers using SSH

Assigned to: Andrea

100.1. Background reading

TODO: to write

- Encryption
- Public key authentication

100.2. 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

100.3. 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](#).

100.4. 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`.

100.5. Creating an SSH keypair

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

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

Use the convention

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

where:

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

An SSH key can be generated with the command:

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

The session output will look something like this:

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

At this point, tell it to choose this file:

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

Then:

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

Press enter; you want an empty passphrase.

Enter same passphrase again:

Press enter.

```
Your identification has been saved in /home/username/.ssh/username@host_name
Your public key has been saved in /home/username/.ssh/username@host_name.pub
The key fingerprint is:
XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX
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
...
```

100.6. 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 100.5](#).

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.

100.7. 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/*
```

100.8. ssh-keygen



TODO: to write

CHAPTER 101

Wireless networking in Linux

101.1. iwconfig

TODO: to write

101.2. 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 102

Moving files between computers

102.1. SCP

TODO: to write

- 1) Download a file with SCP

TODO: to write

102.2. RSync

TODO: to write

CHAPTER 103

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”).

103.1. External documentation



→ [A VIM tutorial.](#)

103.2. Installation



Install like this:

```
$ sudo apt install vim
```

103.3. vi



TODO: to write

103.4. Suggested configuration



Suggested `~/.vimrc`:

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

103.5. Visual mode



TODO: to write

103.6. 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 104

Atom



TODO: to write

CHAPTER 105

Eclipse

..

TODO: to write

105.1. Installing LiClipse

..

TODO: to write

CHAPTER 106

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.)

106.1. Advantages of using Byobu

TODO: To write

106.2. Installation

On Ubuntu, install using:

```
$ sudo apt install byobu
```

106.3. Documentation

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

106.4. Quick command reference

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

Commands to use windows:

TABLE 15. WINDOWS

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

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

TABLE 16. COMMANDS FOR PANES

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

Other commands:

TABLE 17. OTHER

Using function keys	Using escape sequences
Help	<code>Ctrl-A</code> then <code>? </code>
Detach	<code>Ctrl-A</code> then <code>D </code>

106.5. Commands on OS X

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

Highlight using `alt`



CHAPTER 107

Source code control with Git



Assigned to: Andrea

107.1. Background reading



TODO: to write

- Git
- GitFlow

107.2. 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.

107.3. 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
```

107.4. Git tips



1) Shallow clone



You can clone without history with the command:

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

107.5. 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
```

107.6. git

TODO: to write

CHAPTER 108

Git LFS



This describes Git LFS.

108.1. Generic installation instructions



See instructions at:

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

108.2. 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
```

108.3. 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 109

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.

109.1. 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 56](#)).

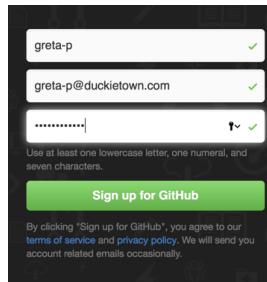


Figure 56

Go to your inbox and verify the email.

109.2. 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.

109.3. 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 100.5](#).

Result:

- You can access Github using the key provided.

Go to settings (Figure 57).

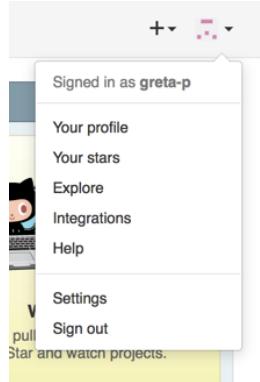


Figure 57

Add the public key that you created:



Figure 58



Figure 59

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 110

ROS installation and reference

Assigned to: Liam

110.1. 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-keyserver.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
```

110.2. rqt_console

TODO: to write

110.3. `roslaunch`

TODO: to write

110.4. `rviz`

TODO: to write

110.5. `rostopic`

TODO: to write

1) `rostopic hz`

TODO: to write

2) `rostopic echo`

TODO: to write

110.6. `catkin_make`

TODO: to write

110.7. `rosrun`

TODO: to write

110.8. `rostest`

TODO: to write

110.9. `rospack`

TODO: to write

110.10. `rosparam`

TODO: to write

110.11. `rosdep`

TODO: to write



110.12. `rosdep`

TODO: to write



110.13. `rosbag`

```
$ rosbag reindex bag file
```



110.14. `roscore`

TODO: to write



110.15. Troubleshooting ROS

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

See [this thread](#). Remove the `known_hosts` file and make sure you have followed the instructions in [Section 100.3](#).



110.16. Other materials about ROS.

* [*A gentle introduction to ROS*](#)



PART 9

Software development guide



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

CHAPTER 111

Python

111.1. Background reading

- Python
- Python tutorial

111.2. Python virtual environments

Install using:

```
$ sudo apt install virtualenv
```

111.3. Useful libraries

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

111.4. Context managers

TODO: to write

CHAPTER 112

Duckietown code conventions

112.1. Python

1) Tabs

Never use tabs in Python file.

The tab characters are evil in Python code. Please be *very* careful in changing them.

Do *not* use a tool to do it (e.g. “Convert tabs to spaces”); it will get it wrong.

✓ checked by `what-the-duck`.

2) Spaces

Indentation is 4 spaces.

3) Line lengths

Lines should be below 85 characters.

✓ `what-the-duck` report those above 100 characters.

This is just a symptom of a bigger problem.

The problem here is that you do not do how to program well, therefore you create programs with longer lines.

Do not go and try to shorten the lines; the line length is just the symptom. Rather, ask somebody to take a look at the code and tell you how to make it better.

4) Spaces

Indentation is 4 spaces.

5) The encoding line

All files must have an encoding declared, and this encoding must be `utf-8`:

```
# -*- coding: utf-8 -*-
```

6) Sha-bang lines

Executable files start with:

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

7) Comments

Comments refer to the next line.

Comments, bad:

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

Comments, better:

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

112.2. Logging

For logging, import this logger:

```
from duckietown_utils import logger
```

112.3. Exceptions

```
DTConfigException

raise_wrapped
compact = True
```

112.4. Scripts

```
def summary():
    fs = get_all_configuration_files()

    if __name__ == '__main__':
        wrap_script_entry_point(summary)
```

1) Imports

Do not do a star import, like the following:

```
from rostest_example.Quacker import *
```

CHAPTER 113

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` ([Section 161.2](#)) checks some of these conditions. If you make a change from the existing conditions, make sure that it gets implemented in `what-the-duck` by filing an issue.

113.1. Environment variables

You need to have set up the variables in [Table 18](#).

TABLE 18. ENVIRONMENT VARIABLES USED BY THE SOFTWARE

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

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.

113.2. The scuderia file

In the `${DUCKIEFLEET_ROOT} /scuderia.yaml` 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 6](#).

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

Listing 6. 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.

TODO: I am still not satisfied. We should have a separate file for every robot, called `ro-
bot_name.vehicle.yaml`. In this way, people don't have to work on the same file. Then `scud-
eria.yaml` can be autogenerated from those files.

113.3. The machines file

The `machines` file is created using:

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

113.4. 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.

113.5. Modes of operation

There are 3 modes of operation:

1. **MODE-normal**: Everything runs on the robot.
2. **MODE-offload**: Drivers run on the robot, but heavy computation runs on the laptop.
3. **MODE-bag**: The data is provided from a bag file, and computation runs on the laptop.

TABLE 19. OPERATION MODES

mode name	who is the ROS master	where data comes from	where heavy computation happen
MODE-normal	duckiebot	Drivers on Duckiebot	duckiebot
MODE-offload	duckiebot	Drivers on Duckiebot	laptop
MODE-bag	laptop	Bag file	laptop

CHAPTER 114

Node configuration mechanisms



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

CHAPTER 115

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.

115.1. 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 *
```

115.2. 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 93.1](#).

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 `rosmg 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.

115.3. 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("[%(name)s] Initializing." % (self.node_name))
```

The line:

```
self.pub_topic_a = rospy.Publisher("~topic_a", String, queue_size=1)
```

defines a publisher which publishes a `String` message to the topic `~topic_a`. Note that the `~` in the name of topic under the namespace of the node. More specifically, this will actually publishes to `talker/topic_a` instead of just `topic_a`. The `queue_size` is usually set to 1 on all publishers.

→ For more details see [rospy overview: publisher and subscribers](#).

The line:

```
self.sub_topic_b = rospy.Subscriber("~topic_b", String, self.cbTopic)
```

defines a subscriber which expects a `String` message and subscribes to `~topic_b`. The message will be handled by the `self.cbTopic` callback function. Note that similar to the publisher, the `~` in the topic name puts the topic under the namespace of the node. In this case the subscriber actually subscribes to the topic `talker/topic_b`.

It is strongly encouraged that a node always publishers and subscribes to topics under their `node_name` namespace. In other words, always put a `~` in front of the topic names when you defines a publisher or a subscriber. They can be easily remapped in a launch file. This makes the node more modular and minimizes the possibility of confusion and naming conflicts. See [the launch file section](#) for how remapping works.

The line

```
self.pub_timestep = self.setupParameter("~pub_timestep", 1.0)
```

Sets the value of `self.pub_timestep` to the value of the parameter `~pub_timestep`. If the parameter doesn't exist (not set in the launch file), then set it to the default value `1.0`. The `setupParameter` function also writes the final value to the parameter server. This means that you can `rosparam list` in a terminal to check the actual values of parameters being set.

The line:

```
self.timer = rospy.Timer(rospy.Duration.from_sec(self.pub_timestep), self.cbTimer)
```

defines a timer that calls the `self.cbTimer` function every `self.pub_timestep` seconds.

2) Timer callback

Contents:

```
def cbTimer(self,event):
    singer = HelloGoodbye()
    # Simulate hearing something
    msg = String()
    msg.data = singer.sing("duckietown")
    self.pub_topic_name.publish(msg)
```

Everyt ime the timer ticks, a message is generated and published.

3) Subscriber callback

Contents:

```
def cbTopic(self,msg):
    rospy.loginfo("[%s] %s" %(self.node_name,msg.data))
```

Every time a message is published to `~topic_b`, the `cbTopic` function is called. It simply prints the messae using `rospy.loginfo`.

115.4. 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.

115.5. Testing the node



First of all, you have to `catkin_make` the package even if it only uses Python. `catkin` makes sure that the modules in the include folder and the messages are available to the whole workspace. You can do so by

```
$ cd ${DUCKIETOWN_ROOT}/catkin_ws
$ catkin_make
```

Ask ROS to re-index the packages so that you can auto-complete most things.

```
$ rospack profile
```

Now you can launch the node by the launch file.

```
$ roslaunch pkg_name test.launch
```

You should see something like this in the terminal:

```
... logging to /home/username/.ros/log/d4db7c80-b272-11e5-8800-5c514fb7f0ed/roslaunc-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 roslaunc 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
```

115.6. 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.

115.7. 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 116

ROS package verification

This chapter describes formally what makes a conforming ROS package in the Duckietown software architecture.

116.1. Naming

- For exercises packages, the name of the package must be `package_handle`.

116.2. `package.xml`

- There is a `package.xml` file.
- Checked by `what-the-duck`.

116.3. Messages

- The messages are called

116.4. Readme file

- There is a `README.md` file
- Checked by `what-the-duck`.

116.5. Launch files

- there is the first launch file

116.6. Test files

`TODO: to write`

CHAPTER 117

Creating unit tests with ROS



PART 10

Duckietown system



This part describes the Duckietown algorithms and system architecture. We do not go in the software details. The implementation details have been already talked about at length in [Part 9](#). We do give links to the ROS packages implementing the functionality.

CHAPTER 118

Teleoperation



TODO: add video here



118.1. Implementation



Drivers:

- [Chapter 162 - Package adafruit_drivers](#)
- [Chapter 171 - Package pi_camera](#)

Operator interface:

- [Chapter 166 - Package joy_mapper](#)



118.2. Camera



TODO: to write



118.3. Actuators



TODO: to write



118.4. IMU



TODO: to write

CHAPTER 119

Parallel autonomy

..

TODO: to write

CHAPTER 120

Lane control



TODO: video here



120.1. Implementation



Perception:

- [Chapter 163 - Package anti_instagram](#)
- [Chapter 165 - Package ground_projection](#)
- [Chapter 170 - Package line_detector](#), [Chapter 169 - Package line_detector2](#)
- [Chapter 168 - Package lane_filter](#)

Control:

- [Chapter 167 - Package lane_control](#)
- [Chapter 206 - Package car_supervisor](#)
- [Chapter 164 - Package dagu_car](#)

CHAPTER 121

Indefinite navigation

TODO: add video here

121.1. Implementation

The packages involved in this functionality are:

- [Chapter 173 - Package `apriltags_ros`](#)
- [Chapter 174 - Package `fsm`](#)
- [Chapter 175 - Package `indefinite_navigation`](#)
- [Chapter 176 - Package `intersection_control`](#)
- [Chapter 177 - Package `navigation`](#)

| **Note:** we don't discuss the details of the packages here; we just give pointers to them.

CHAPTER 122

Planning



TODO: add video here



122.1. Implementation



The packages involved in this functionality are:

- [Chapter 180 - Package `localization`](#)
- [Chapter 179 - Package `duckietown_description`](#)

Note: we don't discuss the details of the packages here; we just give pointers to them.



CHAPTER 123

Coordination



TODO: add video here



123.1. Implementation

- [Chapter 181 - Package led_detection](#)
- [Chapter 182 - Package led_emitter](#)
- [Chapter 183 - Package led_interpreter](#)
- [Chapter 184 - Package led_joy_mapper](#)
- [Chapter 186 - Package traffic_light](#)
- [Chapter 185 - Package rgb_led](#)

CHAPTER 124

Duckietown ROS Guidelines

124.1. Node and Topics

In the source code, a node must only publish/subscribe to private topics.

In `rospy`, this means that the topic argument of `rospy.Publisher` and `rospy.Subscriber` should always have a leading `~`. ex: `~wheels_cmd`, `~mode`.

In `roscpp`, this means that the node handle should always be initialized as a private node handle by supplying with a `"/~"` argument at initialization. Note that the leading `"~"` must then be omitted in the topic names of. ex:

```
ros::NodeHandle nh_("~");
sub_lineseglist_ = nh_.subscribe("lineseglist_in", 1, &GroundProjection::lineseglist_cb, this);
pub_lineseglist_ = nh_.advertise<duckietown_msgs::SegmentList> ("lineseglist_out", 1);
```

124.2. Parameters

All the parameters of a node must be private parameters to that node.

All the nodes must write the value of the parameters being used to the parameter server at initialization. This ensures transparency of the parameters. Note that the `get_param(name,default_value)` does not write the default value to the parameter server automatically.

The default parameter of `pkg_name/node_name` should be put in `~/duckietown/catkin_ws/src/duckietown/config/baseline/pkg_name/node_name/default.yaml`. The elemental launch file of this node should load the parameter using `<rosparam>`.

124.3. Launch file

Each node must have a launch file with the same name in the `launch` folder of the package. ex: `joy_mapper.py` must have a `joy_mapper.launch`. These are referred to as the elemental launch files.

Each elemental launch file must only launch one node.

The elemental launch file should put the node under the correct namespace through the `veh` arg, load the correct configuration and parameter file through `config` and `param_file_name` args respectively. `veh` must not have a default value. This is to ensure the user to always provide the `veh` arg. `config` must be default to `baseline` and `param_file_name` must be default to `default`.

When a node can be run on the vehicle or on a laptop, the elemental launch file should provide a `local` arg. When set to true, the node must be launch on the launching machine, when set to false, the node must be launch on a vehicle through the `machine` attribute.

A node should always be launched by calling its corresponding launch file instead of using `rosrun`. This ensures that the node is put under the correct namespace and all the necessary parameters are provided.

Do not use `<remapp>` in the elemental launch files.

Do not use `<param>` in the elemental launch files.

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 125

General remarks

Assigned to: Andrea

125.1. 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.

125.2. 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.

125.3. 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.

125.4. 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 126

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 127

Additional information for UdeM students



Assigned to: Liam

TODO: to write

CHAPTER 128

Additional information for TTIC students

..

Assigned to: Matt

TODO: to write

CHAPTER 129

Additional information for NCTU students



Assigned to: Nick

TODO: to write

CHAPTER 130

Milestone: ROS node working

..

CHAPTER 131

Homework: Take and process a log



CHAPTER 132

Milestone: Calibrated robot



CHAPTER 133

Homework: Camera geometry



CHAPTER 134

Milestone: Illumination invariance



CHAPTER 135

Homework: Place recognition

»

CHAPTER 136

Milestone: Lane following

..

CHAPTER 137

Homework: localization

• •

CHAPTER 138

Milestone: Navigation

..

CHAPTER 139

Homework: group forming

• •

CHAPTER 140

Milestone: Ducks in a row



CHAPTER 141

Homework: Comparison of PID

10

CHAPTER 142
Homework: RRT

..

CHAPTER 143
Caffe tutorial

..

CHAPTER 144

Milestone: Object Detection



CHAPTER 145

Homework: Object Detection



CHAPTER 146

Milestone: Semantic perception

..

CHAPTER 147

Homework: Semantic perception



CHAPTER 148

Milestone: Reacting to obstacles

..

CHAPTER 149

Homework: Reacting to obstacles

•

•

CHAPTER 150
Milestone: SLAM demo



CHAPTER 151

Homework: SLAM



CHAPTER 152

Milestone: fleet demo

..

CHAPTER 153

Homework: fleet

• •

CHAPTER 154
Project proposals

..

CHAPTER 155

Template of a project



155.1. Checklist for students



- Have a Github account. See [Chapter 109](#). 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).

155.2. Checklist for TAs



- Be signed up on

PART 12

Packages - Infrastructure

..

TODO: to write

CHAPTER 156

Package **duckietown**

156.1. Package information

[Link to package on Github](#)

Essentials

TODO: add authors.

Maintainer: [Mack](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

The duckietown meta package

CHAPTER 157

Package duckietown_msgs

157.1. Package information

[Link to package on Github](#)

Essentials

TODO: add authors.

Maintainer: [Mack](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: Add a description of package `duckietown_msgs` in `package.xml`.

157.2. Package information

[Link to package on Github](#)

Essentials

Author: [Andrea Censi](#)

Maintainer: [Andrea Censi](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

The `easy_algo` package

CHAPTER 158

Package easy_algo



line_detector.easy_algo.yaml

```
interface: line_detector.LineDetectorInterface
description: |
  These are the line detectors.
tests:
- how to test it
```

my_line_detector.line_detector.yaml

```
description: |
  These are
code:
- ClassName
- param1: value1
  param2: value2
```

158.1. Package information



[Link to package on Github](#)

Essentials



TODO: add authors.

Maintainer: [Andrea Censi](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions



The `easy_logs` package

CHAPTER 159

Package easy_logs



line_detector.easy_algo.yaml

```
interface: line_detector.LineDetectorInterface
description: |
  These are the line detectors.
tests:
- how to test it
```

my_line_detector.line_detector.yaml

```
description: |
  These are
code:
- ClassName
- param1: value1
  param2: value2
```

CHAPTER 160

Package `easy_node`

160.1. Package information

[Link to package on Github](#)

Essentials

TODO: add authors.**Maintainer:** [Andrea Censi](#)**TODO:** add code to generate list of dependency.**TODO:** add code to generate a link to a Github issue table for this package.**TODO:** add code to show unit tests build indicator.

Descriptions

`easy_node` is a framework to make it easier to create and document ROS nodes.The main idea is to provide a *declarative approach* to describe:

- The node parameters;
- The node's subscriptions;
- The node's publishers;
- The node's assumptions (contracts).

The user describes subscriptions, publishers, and parameters in a YAML file.

The framework then automatically takes care of:

- Calling the necessary boilerplate ROS commands for subscribing and publishing topics.
- Loading and monitoring configuration.
- Create the Markdown documentation that describes the nodes.
- Provide a set of common functionality, such as benchmarking and monitoring latencies.

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.**Transition plan:** The plan is to first use `easy_node` just for documenting the nodes. Then, later, convert all the nodes to use it.

160.2. YAML file format

For a node with the name `my_node`, implemented in the file `src/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) parameters section: 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.
    default: 1.02
```

2) publishers and subscriptions section

The syntax for describing subscribers is:

```
subscriptions:
  name subscription:
    topic: topic name
    type: message type
    desc: description

    queue_size: queue size
    latch: latch
    process: process
```

The parameters are as follows.

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 down description string.

queue size, **latch** are optional parameters for ROS publishing/subscribing functions.

See the [ROS documentation](#).

The optional parameter `process`, one of `synchronous` (default) or `asynchronous` describes whether to process the message in a synchronous or asynchronous way (in a separated thread).

The optional parameter `timeout` describes a timeout value. If no message is received for more than this value, the function `on_timeout_subscription()` is called.

TODO: implement this timeout functionality.

The syntax for describing publishers is similar; it does not have the the `process` and `timeout` value.

Example:

```
subscriptions:
  segment_list:
    topic: ~segment_list
    type: duckietown_msgs/SegmentList
    desc: Line detections
    queue_size: 1
    timeout: 3

publishers:
  lane_pose:
    topic: ~lane_pose
    type: duckietown_msgs/LanePose
    desc: Estimated pose
    queue_size: 1
```

3) Describing contracts

Note: 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

160.3. Using the `easy_node` API

1) Initialization

Here is a minimal example of a node that conforms with the API:

```
from easy_node import EasyNode

class MyNode(EasyNode):

    def __init__(self):
        EasyNode.__init__(self, 'my_package', 'my_node')
        self.info('Initialized.')

    if __name__ == '__main__':
        MyNode().spin()
```

The node class must derive from `EasyNode`. You need to tell EasyNode what is the package name and the node name.

To initialize, call the function `spin()`.

The `EasyNode` class provides the following functions:

```
info()
debug()
error()
```

These are mapped to `rospy.loginfo()` etc.; they include the name of the node.

2) Using configuration parameters

This next example shows how to use configuration parameters.

First, create a file `my_node.easy_node.yaml` containing:

```
parameters:
  num_cells:
    desc: Number of cells.
    type: int
    default: 42
subscriptions:
  contracts:
  publishers:
```

Then, implement the method `on_parameters_changed()`. It takes two parameters:

- `first_time` is a boolean that tells whether this is the first time that the function is called (initialization time).
- `updated` is a set of strings that describe the set of parameters that changed. The first time, it contains the set of all parameters.

To access the parameter value, access `self.config.parameter`.

Example:

```

class MyNode():

    def __init__(self):
        EasyNode.__init__(self, 'my_package', 'my_node')

    def on_parameters_changed(self, first_time, updated):
        if first_time:
            self.info('Initializing array for the first time.')
            self.cells = [0] * self.config.num_cells

        else:
            if 'num_cells' in updated:
                self.info('Number of cells changed.')
                self.cells = [0] * self.config.num_cells

    if __name__ == '__main__':
        Node().spin()

```

EasyNode will monitor the ROS parameter server, and will call the function `on_parameters_changed` if the user changes any parameters.

3) Using subscriptions

To automatically subscribe to topics, add an entry in the `subscriptions` section of the YAML file.

For example:

```

subscriptions:
  joy:
    desc: |
      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.
    type: sensor_msgs/Joy
    topic: ~joy
    timeout: 3.0

```

Then, implement the function `on_received_name`.

This function will be passed 2 arguments:

- a context object; this can be used for benchmarking ([Section 160.6](#)).
- the message object.

Example:

```
class MyNode():
    # ...

    def on_received_joy(self, context, msg):
        # This is called any time a message arrives
        self.info('Message received: %s' % msg)
```

4) Time-out

TODO: to implement

The function `on_timeout_subscription()` is called when there hasn't been a message for the specified timeout interval.

```
class MyNode():
    # ...

    def on_timeout_joy(self, context, time_since):
        # This is called when we have not seen a message for a while
        self.error('No joystick received since %s.' % time_since)
```

5) Publishers

The publisher object can be accessed at `self.publishers.name`. EasyNode has taken care of all the initialization for you.

For example, suppose we specify a publisher `command` using:

```
publishers:
    command:
        desc: The control command.
        type: duckietown_msgs/Twist2DStamped
        topic: ~car_cmd
```

Then we can use it as follows.

```
class MyNode():
    # ...

    def on_received_joy(self, context, msg):
        out = Twist2DStamped()
        out.header.stamp = 0
        out.v = 0
        out.omega = 0

        self.publishers.command.publish(out)
```

6) `on_init()` and `on_shutdown()`

Define the two methods `on_init()` and `on_shutdown()` to c

```

class MyNode(EasyNode):
    # ...
    def on_init(self):
        self.info('Step 1 - Initialized')

    def on_parameters_changed(self, first_time, changed):
        self.info('Step 2 - Parameters received')

    def on_shutdown(self):
        self.info('Step 3 - Preparing for shutdown.')

```

Note that `on_init()` is called before `on_parameters_changed()`.

160.4. Configuration using easy_node: the user's point of view

So far, we have seen how to use parameters from the node, but we did not talk about how to specify the parameters from the user's point of view.

EasyNode introduces lots of flexibility compared to the legacy system.

1) Configuration file location

The user configuration is specified using files by the pattern

```
package_name-node_name.config_name.config.yaml
```

where `config_name` is a short string (e.g., `baseline`).

The files can be anywhere in:

- The directory `${DUCKIETOWN_ROOT}/catkin_ws/src` ;
- The directory `${DUCKIEFLEET_ROOT}` .

Several config files can exist at the same time. For example, we could have somewhere:

```

line_detector-line_detector.baseline.config.yaml
line_detector-line_detector.fall2017.config.yaml
line_detector-line_detector.andrea.config.yaml

```

where the `baseline` versions are the baseline parameters, `fall2017` are the parameters we are using for Fall 2017, and `andrea` are temporary parameters that the user is using.

However, there cannot be two configurations with the same filename e.g. two copies of `line_detector-line_detector.baseline.config.yaml`. In this case, EasyNode will raise an error.

TODO: implement this functionality.

2) Configuration file format

The format of the `*.config.yaml` file is as follows:

```

description: |
  description of what this configuration accomplishes
extends: [config name, config name]
values:
  parameter name: value
  parameter name: value

```

The `extends` field (optional) is a list of string. It allows to use the specified configurations as the defaults for the current one.

For example, the file `line_detector-line_detector.baseline.config.yaml` could contain:

```

description: |
  These are the standard values for the line detector.
extends: []
values:
  img_size: [120,160]
  top_cutoff: 40

```

3) Configuration sequence

Which parameters are used depend on the **configuration sequence**.

The configuration sequence is a list of configuration names.

It can be specified by the environment variable `DUCKIETOWN_CONFIG_SEQUENCE`, using a colon-separated list of strings. For example:

```
$ export DUCKIETOWN_CONFIG_SEQUENCE=baseline:fall2017:andrea
```

The line above specifies that the configuration sequence is `baseline`, `fall2017`, `andrea`.

The system loads the configuration in order. First, it loads the `baseline` version. Then it loads the `fall2017` version. If a value was already specified in the `baseline` version, it is overwritten. If a version does not exist, it is simply skipped.

If a parameter is not specified in any configuration, an error is raised.

Using this functionality, it is easy to have team-based customization and user-based customization.

There are two special configuration names:

1. The configuration name “`defaults`” loads the defaults specified by the node. Note that the defaults are ignored otherwise.
2. The configuration name “`vehicle`” expands to the name of the vehicle being used.

TODO: the `vehicle` part is not implemented yet.

4) Time-variant configuration

EasyNode allows to describe configuration that can change in time.

The use case for this is the configuration of calibration parameters:

- Calibration parameters change with time.
- We still want to access old calibration parameters, when processing logs.

The solution is to allow a date tag in the configuration name. The format for this is

```
package_name-node_name.config_name.date.config.yaml
```

For example, we could have the files:

```
kinematics-kinematics.ferrari.20160404.config.yaml  
kinematics-kinematics.ferrari.20170101.config.yaml
```

Given this, EasyNode will select the configuration to use intelligently. When reading from a bag file from 2016, the first configuration is going to be used; for logs in 2017, the second is going to be used.

TODO: not implemented yet.

160.5. Visualizing the configuration database

There are a few tools used to visualize the configuration database.

1) easy_node desc: Describing a node

The command

```
$ rosrun easy_node desc package_name node_name
```

shows a description of the node, as specified in `package_name.node_name.easy_node.yaml`.

For example:

```
$ rosrun easy_node desc line_detector2 line_detector2
```

shows the following:

```
Configuration for node "line_detector_node2" in package "line_detector2"
=====
```

Parameters

name	type	default
en_update_params_interval	float	2.0
top_cutoff	int	(none)
detector	(n/a)	(none)
img_size	(n/a)	(none)
verbose	bool	True

Subscriptions

name	type	topic	options	process
switch	BoolStamped	~switch	queue_size = 1	synchronous
image	CompressedImage	~image	queue_size = 1	threaded
transform	AntiInstagramTransform	~transform	queue_size = 1	synchronous

Publishers

name	type	topic	options
color_segment	Image	~colorSegment	queue_size = 1
edge	Image	~edge	queue_size = 1
segment_list	SegmentList	~segment_list	queue_size = 1
image_with_lines	Image	~image_with_lines	queue_size = 1

2) easy_node eval: Evaluating a configuration sequence

The program `eval` script allows to evaluate a certain configuration sequence.

The syntax is:

```
$ rosrun easy_node eval package_name node_name config_sequence
```

The tool shows also which file is responsible for the value of each parameter.

For example, the command

```
$ rosrun easy_node eval line_detector2 line_detector_node2 defaults:andrea
```

evaluates the configuration for the `line_detector_node2` node with the configuration sequence `defaults:andrea`.

The result is:

```
Configuration result for node `line_detector_node2` (package `line_detector2`)
The configuration sequence was ['defaults', 'baseline', 'andrea'].
The following is the list of parameters set and their origin:
  parameter          value          origin
  -----  -----
  en_update_params_interval  2.0          defaults
  top_cutoff                40           baseline
  detector
    - line_detector.LineDetectorHSV  baseline
    - configuration:
      canny_thresholds: [80, 200]
      dilation_kernel_size: 3
      hough_max_line_gap: 1
      hough_min_line_length: 3
      hough_threshold: 2
      hsv_red1: [0, 140, 100]
      hsv_red2: [15, 255, 255]
      hsv_red3: [165, 140, 100]
      hsv_red4: [180, 255, 255]
      hsv_white1: [0, 0, 150]
      hsv_white2: [180, 60, 255]
      hsv_yellow1: [25, 140, 100]
      hsv_yellow2: [45, 255, 255]
  img_size                [120, 160]    baseline
  verbose                  true          defaults
```

Note how we can tell which configuration file is responsible for setting each parameter.

3) easy_node summary: Describing and validating all configuration files

The program `summary` reads all the configuration files described in the repository and validates them against the node description.

If a configuration file refers to parameters that do not exist, the configuration file is established to be invalid.

The syntax is:

```
$ rosrun easy_node summary
```

For example, the output can be:

```
package name      node name      config_name      effective      extends      valid
error      description
line_detector2    line_detector_node2    baseline      2017-01-01      ()      yes
                           These are the standard values for t [...]
```

160.6. Benchmarking

EasyNode implements some simple timing statistics. These are accessed using the `context` object passed to the message received callbacks.

Here's an example use, from `line_detector2`:

```
def on_received_image(self, context, image_msg):
    with context.phase('decoding'):
        ...
    with context.phase('resizing'):
        # Resize and crop image
        ...
    stats = context.get_stats()
    self.info(stats)
```

The idea is to enclose the different phases of the computation using the [context manager](#) `phase(name)`.

A summary of the statistics can be accessed by using `context.get_stats()`.

For example, this will print:

```
Last 24.4 s: received 734 (30.0 fps) processed 301 (12.3 fps) skipped 433 (17.7 fps) (59 %)
    decoding | total latency 25.5 ms | delta wall 20.7 ms | delta clock 20.7 ms
    resizing | total latency 26.6 ms | delta wall 0.8 ms | delta clock 0.7 ms
    correcting | total latency 29.1 ms | delta wall 2.2 ms | delta clock 2.2 ms
    detection | total latency 47.7 ms | delta wall 18.2 ms | delta clock 21.3 ms
    preparing-images | total latency 55.0 ms | delta wall 7.0 ms | delta clock 7.0 ms
    publishing | total latency 55.5 ms | delta wall 0.1 ms | delta clock 0.1 ms
    draw-lines | total latency 59.7 ms | delta wall 4.0 ms | delta clock 3.9 ms
    published-images | total latency 61.2 ms | delta wall 0.9 ms | delta clock 0.8 ms
  pub_edge/pub_segment | total latency 86.3 ms | delta wall 24.7 ms | delta clock 24.0 ms
```

160.7. Automatic documentation generation

EasyNode generated the documentation automatically from the `*.easy_node.yaml` files.

Note that this can be used independently of the fact that the node implements the `EasyNode` API. So, we can use this EasyNode functionality also to document the legacy nodes.

To generate the docs, use this command:

```
$ rosrun easy_node generate_docs
```

For each node documented using a `*.easy_node.yaml`, this generates a Markdown file called “`node_name-easy_node-autogenerated.md`” in the package directory.

The contents are enclosed in a `div` with ID `#package_name-node_name-autogenerated`.

The intended use is that this can be used to move the contents to the place in the documentation using [the move-here tag](#).

For example, in the `README.md` of the `joy_mapper` package, we have:

```
## Node `joy_mapper_node`  
<move-here src="#joy_mapper-joy_mapper_node-autogenerated"/>
```

The result can be seen at [Chapter 166](#).

160.8. Parameters and services defined for all packages

(Generated from [configuration easy_node.yaml](#).)

These are properties common to all nodes.

Parameters

No parameters defined.

Subscriptions

No subscriptions defined.

Publishers

No publishers defined.

160.9. Other ideas

(Add here other ideas that we can implement.)

CHAPTER 161

Package `what_the_duck`

161.1. Package information

[Link to package on Github](#)

Essentials

TODO: add authors.

Maintainer: [Andrea Censi](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

What the duck?

`what-the-duck` is a program that tests *dozens* of configuration inconsistencies that can happen on a Duckiebot.

161.2. What the duck

To use it, first compile the repository, and then run:

```
$ ./what-the-duck
```

161.3. 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.

161.4. 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
```

161.5. 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 162

Package adafruit_drivers

162.1. Package information

[Link to package on Github](#)

Essentials

Author: [Dmitry Yershov](#)

Maintainer: [Mack](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: Add a description of package `adafruit_drivers` in `package.xml`.

These are the Adafruit drivers.

TODO: What is the original location of this package?

CHAPTER 163

Package `anti_instagram`

163.1. Package information

[Link to package on Github](#)

Essentials

Author: [mfe](#)

Maintainer: [Mack](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: Add a description of package `anti_instagram` in `package.xml`.

163.2. Unit tests integrated with `rostest`

Unit tests are integrated with [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
```

163.3. Unit tests needed external files

These are other unittest that require the logs in DUCKIETOWN_DATA:

```
$ rosrun anti_instagram annotations_test.py
```

163.4. Node `anti_instagram_node`

(Generated from [configuration `anti_instagram_node.easy_node.yaml`](#).)

TODO: Missing node description in `anti_instagram_node.easy_node.yaml`.

Parameters

Parameter `publish_corrected_image: bool`; default value: `False`

Whether to compute and publish the corrected image.

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.

Publishers

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 164

Package `dagu_car`

164.1. Package information

[Link to package on Github](#)

Essentials

Author: [Dmitry Yershov](#)

Author: [Shih-Yuan Liu](#)

Maintainer: [Mack](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: Add a description of package `dagu_car` in `package.xml`.

164.2. Node `forward_kinematics_node`

(Generated from [configuration `forward_kinematics_node.easy_node.yaml`](#).)

TODO: Missing node description in `forward_kinematics_node.easy_node.yaml`.

Parameters

No parameters defined.

Subscriptions

No subscriptions defined.

Publishers

No publishers defined.

164.3. Node `inverse_kinematics_node`

(Generated from [configuration `inverse_kinematics_node.easy_node.yaml`](#).)

TODO: Missing node description in `inverse_kinematics_node.easy_node.yaml`.

Parameters

No parameters defined.

Subscriptions

No subscriptions defined.

Publishers

No publishers defined.

164.4. Node `velocity_to_pose_node`

(Generated from [configuration `velocity_to_pose_node.easy_node.yaml`](#).)

TODO: Missing node description in `velocity_to_pose_node.easy_node.yaml`.

Parameters

No parameters defined.

Subscriptions

No subscriptions defined.

Publishers

No publishers defined.

164.5. Node `car_cmd_switch_node`

(Generated from [configuration `car_cmd_switch_node.easy_node.yaml`](#).)

TODO: Missing node description in `car_cmd_switch_node.easy_node.yaml`.

Parameters

No parameters defined.

Subscriptions

No subscriptions defined.

Publishers

No publishers defined.

164.6. Node `wheels_driver_node`

(Generated from [configuration `wheels_driver_node.easy_node.yaml`](#).)

TODO: Missing node description in `wheels_driver_node.easy_node.yaml`.

Parameters

No parameters defined.

Subscriptions

No subscriptions defined.

Publishers

No publishers defined.

164.7. Node `wheels_trimmer_node`

(Generated from [configuration `wheels_trimmer_node.easy_node.yaml`.](#))

TODO: Missing node description in `wheels_trimmer_node.easy_node.yaml`.

Parameters

No parameters defined.

Subscriptions

No subscriptions defined.

Publishers

No publishers defined.

CHAPTER 165

Package ground_projection

165.1. Package information

[Link to package on Github](#)

Essentials

Author: [Changhyun Choi](#)

Maintainer: [Mack](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: Add a description of package `ground_projection` in `package.xml`.

165.2. Node ground_projection_node

(Generated from [configuration `ground_projection.easy_node.yaml`](#).)

TODO: Missing node description in `ground_projection.easy_node.yaml`.

Parameters

No parameters defined.

Subscriptions

No subscriptions defined.

Publishers

No publishers defined.

CHAPTER 166

Package joy_mapper

166.1. Package information

[Link to package on Github](#)

Essentials

TODO: add authors.

Maintainer: [Mack](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

The joy_mapper package for duckietown. Takes sensor_msgs.Joy and convert it to duckietown_msgs.CarControl.

166.2. Testing

Connect joystick.

Run:

```
$ roslaunch launch/joy_mapper_test.launch
```

The robot should move when you push buttons.

166.3. Dependencies

- `rospy`
- `sensor_msgs`: for the `Joy.msg`
- `duckietown_msgs`: for the `CarControl.msg`

166.4. Node joy_mapper_node2

(Generated from [configuration_joy_mapper_node2_easy_node.yaml](#).)

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.

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`”.

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.

Publishers

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 167

Package `lane_control`

167.1. Package information

[Link to package on Github](#)

Essentials

Author: [steven](#)Author: [Shih-Yuan Liu](#)Maintainer: [Liam Paull](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: Add a description of package `lane_control` in `package.xml`.167.2. `lane_controller_node`(Generated from [configuration `lane_controller_node.eeasy_node.yaml`](#).)**Note:** there is some very funny business inside. It appears that `k_d` and `k_theta` are switched around.

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 .

Subscriptions

Subscription `lane_reading`: topic `~lane_pose` (`LanePose`)

TODO: Missing description for entry “lane_reading”.

Publishers

Publisher `car_cmd`: topic `~car_cmd` (`Twist2DStamped`)

TODO: Missing description for entry “car_cmd”.

CHAPTER 168

Package `lane_filter`

Assigned to: Liam

168.1. Package information

[Link to package on Github](#)

Essentials

TODO: add authors.

Maintainer: [Liam Paull](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: Add a description of package `lane_filter` in `package.xml`.

168.2. `lane_filter_node`

(Generated from [configuration `lane_filter_node.easy_node.yaml`](#).)

TODO: Missing node description in `lane_filter_node.easy_node.yaml`.

Parameters

Parameter `peak_val`: `float`; default value: `10.0`

TODO: Missing description for entry “`peak_val`”.

Parameter `l_max`: `float`; default value: `2.0`

TODO: Missing description for entry “`l_max`”.

Parameter `lanewidth`: `float`; default value: `0.4`

TODO: Missing description for entry “`lanewidth`”.

Parameter `mean_d_0`: `float`; default value: `0.0`

TODO: Missing description for entry “`mean_d_0`”.

Parameter `min_max`: `float`; default value: `0.3`

Expressed in nats.

Parameter `d_max`: `float`; default value: `0.5`

TODO: Missing description for entry “`d_max`”.

Parameter `use_distance_weighting`: `bool`; default value: `False`

For use of distance weighting (dw) function.

Parameter `linewidth_white`: `float`; default value: `0.04`

TODO: Missing description for entry “`linewidth_white`”.

Parameter `cov_omega`: `float`; default value: `0.01`

Angular velocity “input”.

which units?

Parameter `use_max_segment_dist`: `bool`; default value: `False`

For use of maximum segment distance.

Parameter `delta_d`: `float`; default value: `0.02`

(meters)

Parameter `min_segs`: `int`; default value: `10`

For use of minimum segment count.

Parameter `phi_min`: `float`; default value: `-1.5707`

TODO: Missing description for entry “`phi_min`”.

Parameter `sigma_d_0`: `float`; default value: `0.0`

TODO: Missing description for entry “`sigma_d_0`”.

Parameter `phi_max`: `float`; default value: `1.5707`

TODO: Missing description for entry “`phi_max`”.

Parameter `zero_val`: `float`; default value: `1.0`

TODO: Missing description for entry “`zero_val`”.

Parameter `delta_phi`: `float`; default value: `0.0`

(radians)

Parameter `l_peak`: `float`; default value: `1.0`

TODO: Missing description for entry “`l_peak`”.

Parameter `cov_v`: `float`; default value: `0.5`

Linear velocity “input”.

which units?

Parameter `sigma_phi_mask`: `float`; default value: `0.05`

TODO: Missing description for entry “`sigma_phi_mask`”.

Parameter `sigma_d_mask`: `float`; default value: `0.05`

TODO: Missing description for entry “`sigma_d_mask`”.

Parameter `mean_phi_0`: `float`; default value: `0.0`

TODO: Missing description for entry “`mean_phi_0`”.

Parameter `sigma_phi_0`: `float`; default value: `0.0`

TODO: Missing description for entry “`sigma_phi_0`”.

Parameter `d_min`: `float`; default value: `-0.7`

TODO: Missing description for entry “`d_min`”.

Parameter `linewidth_yellow`: `float`; default value: `0.02`

TODO: Missing description for entry “`linewidth_yellow`”.

Parameter `use_min_segs`: `bool`; default value: `False`

For use of minimum segment count.

Parameter `use_propagation`: `bool`; default value: `False`

For propagation.

Parameter `max_segment_dist`: `float`; default value: `1.0`

For use of maximum segment distance.

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`”.

Publishers

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 169

Package line_detector2

169.1. Package information

[Link to package on Github](#)

Essentials

TODO: add authors.

Maintainer: [Andrea Censi](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: Add a description of package `line_detector2` in `package.xml`.

This is a re-implementation of the package [line_detector](#) using the new facilities provided by [easy_node](#).

169.2. 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 of 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/vwznjke4xvnh19o/160122_manual2-ferrari.bag?dl=1  
https://www.dropbox.com/s/y7u1j198punj0mp/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 61](#).

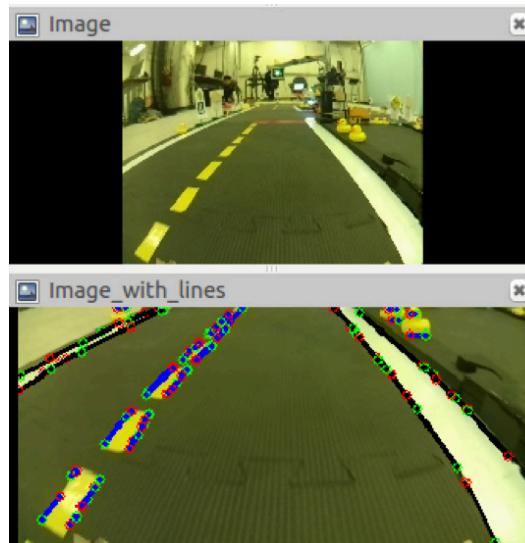


Figure 61. RViz output

169.3. Quantitative tests

TODO: Something more quantitative (to be filled in by Liam or Hang)

169.4. line_detector_node2

(Generated from [configuration line_detector_node2.easy_node.yaml](#).)

This is a rewriting of `line_detector_node` using the [EasyNode](#) framework.

Parameters

Parameter `top_cutoff: int`

This parameter decides how much of the image we should cut off. This is a performance improvement.

Parameter `line_detector: str`

This is the instance of `line_detector` to use.

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.

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 163](#).

Publishers

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 170

Package line_detector

170.1. Package information

[Link to package on Github](#)

Essentials

Author: [Hang Zhao](#)

Maintainer: [Mack](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: Add a description of package `line_detector` in `package.xml`.

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 171

Package `pi_camera`

171.1. Package information

[Link to package on Github](#)

Essentials

TODO: add authors.

Maintainer: [Andrea Censi](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: Add a description of package `pi_camera` in `package.xml`.

171.2. Node `camera_node_sequence`

(Generated from [configuration `camera_node_sequence.easy_node.yaml`](#).)

Camera driver, second approach.

Parameters

No parameters defined.

Subscriptions

No subscriptions defined.

Publishers

No publishers defined.

171.3. Node `camera_node_continuous`

(Generated from [configuration `camera_node_continuous.easy_node.yaml`](#).)

Camera driver.

Parameters

Parameter `res_w`: `int`; default value: `320`

Resolution (width)

Parameter `framerate`: `float`; default value: `60.0`

Frame rate

Parameter `res_h`: `int`; default value: `200`

Resolution (height)

Subscriptions

No subscriptions defined.

Publishers

Publisher `image_compressed`: topic `~image/compressed` (`CompressedImage`)

TODO: Missing description for entry “`image_compressed`”.

171.4. Node `decoder_node`

(Generated from [configuration_decoder_node.easy_node.yaml](#).)

A node that decodes a compressed image into a regular image. This is useful so that multiple nodes that need to use the image do not do redundant computation.

Parameters

Parameter `publish_freq`: `float`; default value: `1.0`

Frequency at which to publish (Hz).

Subscriptions

Subscription `compressed_image`: topic `~compressed_image` (`CompressedImage`)

The image to decode.

Subscription `switch`: topic `~switch` (`BoolStamped`)

Switch to turn on or off. The node starts as active.

Publishers

Publisher `raw`: topic `~image/raw` (`Image`)

The decoded image.

171.5. Node `img_process_node`

(Generated from [configuration_img_process_node.easy_node.yaml](#).)

Apparently, a template, or a node never finished.

Parameters

No parameters defined.

Subscriptions

No subscriptions defined.

Publishers

No publishers defined.

171.6. Node cam_info_reader_node

(Generated from [configuration cam_info_reader_node.easy_node.yaml](#).)

Publishes a CameraInfo message every time it receives an image.

Parameters

Parameter `image_type`: str ; default value: 'compressed'

TODO: Missing description for entry "image_type".

Parameter `config`: str ; default value: 'baseline'

TODO: Missing description for entry "config".

Parameter `cali_file_name`: str ; default value: 'default'

TODO: Missing description for entry "cali_file_name".

Subscriptions

Subscription `compressed_image`: topic `~compressed_image` (CompressedImage)

If image_type is "compressed" then it's CompressedImage, otherwise Image.

Publishers

Publisher `camera_info`: topic `~camera_info` (CameraInfo)

TODO: Missing description for entry "camera_info".

PART 14

Packages - Indefinite navigation

TODO: to write

CHAPTER 172

AprilTags library

172.1. Package information

[Link to package on Github](#)

Essentials

Author: Michael Kaess

Author: Hordur Johannson

Maintainer: [Mitchell Wills](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

A catkin version of the C++ apriltags library

Detect April tags (2D bar codes) in images; reports unique ID of each detection, and optionally its position and orientation relative to a calibrated camera.

See examples/apriltags_demo.cpp for a simple example that detects April tags (see tags/pdf/tag36h11.pdf) in laptop or webcam images and marks any tags in the live image.

Ubuntu dependencies: sudo apt-get install subversion cmake libopencv-dev libeigen3-dev libv4l-dev

Mac dependencies: sudo port install pkgconfig opencv eigen3

Uses the pods build system in connection with cmake, see: <http://sourceforge.net/p/pods/>

Michael Kaess October 2012

AprilTags were developed by Professor Edwin Olson of the University of Michigan. His Java implementation is available on this web site: <http://april.eecs.umich.edu>.

Olson's Java code was ported to C++ and integrated into the Tekkotsu framework by Jeffrey Boyland and David Touretzky.

See this Tekkotsu wiki article for additional links and references: <http://wiki.tekkotsu.org/index.php/AprilTags>

This C++ code was further modified by Michael Kaess (kaess@mit.edu) and Hordur Johannson (hordurj@mit.edu) and the code has been released under the LGPL 2.1 license.

- converted to standalone library
- added stable homography recovery using OpenCV
- robust tag code table that does not require a terminating 0 (omission results in false positives by illegal codes being accepted)

- changed example tags to agree with Ed Olson's Java version and added all his other tag families
- added principal point as parameter as in original code - essential for homography
- added some debugging code (visualization using OpenCV to show intermediate detection steps)
- added fast approximation of arctan2 from Ed's original Java code
- using interpolation instead of homography in Quad: requires less homography computations and provides a small improvement in correct detections

todo: - significant speedup could be achieved by performing image operations using OpenCV (Gaussian filter, but also operations in TagDetector.cc) - replacing arctan2 by precomputed lookup table - converting matrix operations to Eigen (mostly for simplifying code, maybe some speedup)

CHAPTER 173

Package `apriltags_ros`

AprilTags for ROS.

build passing



173.1. Package information



[Link to package on Github](#)



Essentials



Author: Mitchell Wills

Maintainer: [Mitchell Wills](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.



Descriptions



A package that provides a ROS wrapper for the apriltags C++ package

CHAPTER 174

Package fsm

174.1. Package information

[Link to package on Github](#)

Essentials

Author: [Michael Misha Novitzky](#)

Maintainer: [Mack](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

The finite state machine coordinates the modes of the car.

174.2. Node fsm_node

(Generated from [configuration fsm_node_easy_node.yaml](#).)

Note that this node publishes on many topics according to the configuration:

```
for node_name, topic_name in nodes.items():
    self.pub_dict[node_name] = rospy.Publisher(topic_name, BoolStamped, ...)
```

Parameters

Parameter states: `dict`; default value: ''

TODO: Missing description for entry “states”.

Parameter nodes: not known

TODO: Missing description for entry “nodes”.

Parameter initial_state: `str`; default value: ''

TODO: Missing description for entry “initial_state”.

Parameter global_transitions: `dict`; default value: ''

TODO: Missing description for entry “global_transitions”.

Parameter events: `dict`; default value: ''

TODO: Missing description for entry “events”.

Subscriptions

No subscriptions defined.

Publishers

Publisher `mode`: topic `~mode` (`FSMState`)

TODO: Missing description for entry “`mode`”.

CHAPTER 175

Package `indefinite_navigation`

175.1. Package information

[Link to package on Github](#)

Essentials

TODO: add authors.

Maintainer: [Liam Paull](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: Add a description of package `indefinite_navigation` in `package.xml`.

CHAPTER 176

Package intersection_control

176.1. Package information

[Link to package on Github](#)

Essentials

Author: [Michael Misha Novitzky](#)

Maintainer: [Mack](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

The intersection_control package implements a dead reckoning controller until we do something smarter.

CHAPTER 177

Package navigation

177.1. Package information

[Link to package on Github](#)

Essentials

Author: [igor](#)

Maintainer: [Mack](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: Add a description of package `navigation` in `package.xml`.

CHAPTER 178

Package `stop_line_filter`

178.1. Package information

[Link to package on Github](#)

Essentials

TODO: add authors.

Maintainer: [Liam Paull](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: Add a description of package `stop_line_filter` in `package.xml`.

PART 15

Packages - Localization and planning

TODO: to write

CHAPTER 179

Package `duckietown_description`

179.1. Package information

[Link to package on Github](#)

Essentials

Author: [Teddy Ort](#)

Maintainer: [Mack](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: Add a description of package `duckietown_description` in `package.xml`.

CHAPTER 180

Package localization

180.1. Package information

[Link to package on Github](#)

Essentials

Author: [teddy](#)

Maintainer: [Mack](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: Add a description of package localization in `package.xml`.

PART 16

Packages - Coordination



TODO: to write

CHAPTER 181

Package led_detection

181.1. Package information

[Link to package on Github](#)

Essentials

TODO: add authors.

Maintainer: [Andrea Censi](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: Add a description of package `led_detection` in `package.xml`.

181.2. 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:

```
$ rosrun 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.

181.3. Unit tests

To run the unit tests for the LED detector, you need to have the F23 rosbags on your hard disk. These bag files should be synced from [this dropbox link] (https://www.dropbox.com/sh/5kx8qwgttu69fhr/AAASLpOVjV5r1xpzeW7xWZh_a?dl=0).

For the test to locate the bag files, you should have the `DUCKIETOWN_DATA` environment variable set, pointing to the location of your 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 unitests.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 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 “*”.

The default algorithm is called “`baseline`”, and its tests are invoked using:

```
$ rosrun led_detection <script> '*' 'baseline'
```

CHAPTER 182

Package led_emitter

182.1. Package information

[Link to package on Github](#)

Essentials

TODO: add authors.

Maintainer: [Andrea Censi](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: description for `led_emitter` package.

182.2. In depth

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:

```
$ rosrun 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:

```
$ rosrun 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:

```
$ rosrun led_emitter led_emitter_node_test.launch veh:=robot_name
```

This will cycle through frequencies of 3.0hz, 3.5hz, and 4hz every 5 seconds. Once done, kill everything and make sure you have joystick control as described above.

CHAPTER 183

Package led_interpreter

183.1. Package information

[Link to package on Github](#)

Essentials

Author: [qlai](#)

Maintainer: [Mack](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: Add a description of package `led_interpreter` in `package.xml`.

CHAPTER 184

Package led_joy_mapper

184.1. Package information

[Link to package on Github](#)

Essentials

Author: [lapentab](#)Maintainer: [Mack](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: Add a description of package `led_joy_mapper` in `package.xml`.

CHAPTER 185

Package `rgb_led`

185.1. Package information

[Link to package on Github](#)

Essentials

Author: [Dmitry Yershov](#)

Maintainer: [Mack](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: Add a description of package `rgb_led` in `package.xml`.

185.2. 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 186

Package `traffic_light`

186.1. Package information

[Link to package on Github](#)

Essentials

TODO: add authors.

Maintainer: [Mack](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: Add a description of package `traffic_light` in `package.xml`.

PART 17

Packages - Additional functionality

..

TODO: to write

CHAPTER 187

Package mdoap

187.1. Package information

[Link to package on Github](#)

Essentials

Author: [Catherine Liu](#)

Author: [Sam](#)

Author: [Ari](#)

Maintainer: [Mack](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

The mdoap package which handles object recognition and uses ground projection to estimate distances of objects for duckie safety. Also contains controllers for avoiding said obstacles

CHAPTER 188

Package parallel_autonomy

188.1. Package information

[Link to package on Github](#)

Essentials

TODO: add authors.

Maintainer: [Liam Paull](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: description for the parallel autonomy package

CHAPTER 189

Package `vehicle_detection`

189.1. Package information

[Link to package on Github](#)

Essentials

Author: [Andrea Tacchetti](#)

Maintainer: [Mack](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: Add a description of package `vehicle_detection` in `package.xml`.

PART 18

Packages - Templates

..

These are templates.

CHAPTER 190

Package `pkg_name`

190.1. Package information

[Link to package on Github](#)

Essentials

Author: [Shih-Yuan Liu](#)

Maintainer: [Andrea Censi](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

The package `pkg_name` is a template for ROS packages.

For the tutorial, see [Chapter 115](#).

CHAPTER 191

Package rostest_example

191.1. Package information

[Link to package on Github](#)

Essentials

Author: [Teddy Ort](#)

Maintainer: [Mack](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: Add a description of package rostest_example in `package.xml`.

PART 19**Packages - Convenience**

These packages are convenience packages that group together launch files and tests.

CHAPTER 192

Package `duckie_rr_bridge`

192.1. Package information

[Link to package on Github](#)

Essentials

Author: [greg](#)

Maintainer: [Mack](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

The `duckie_rr_bridge` package. Creates a Robot Raconteur Service to drive the Duckiebot.

CHAPTER 193

Package `duckiebot_visualizer`

193.1. Package information

[Link to package on Github](#)

Essentials

Author: [Shih-Yuan Liu](#)

Maintainer: [Mack](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: Add a description of package `duckiebot_visualizer` in `package.xml`.

193.2. Node `duckiebot_visualizer`

(Generated from [configuration `duckiebot_visualizer.easy_node.yaml`](#).)

TODO: Missing node description in `duckiebot_visualizer.easy_node.yaml`.

Parameters

Parameter `veh_name`: str ; default value: `'megaman'`

TODO: Missing description for entry “`veh_name`”.

Subscriptions

Subscription `seg_list`: topic `~segment_list` (`SegmentList`)

TODO: Missing description for entry “`seg_list`”.

Publishers

Publisher `pub_seg_list`: topic `~segment_list_markers` (`MarkerArray`)

TODO: Missing description for entry “`pub_seg_list`”.

CHAPTER 194

Package `duckietown_demos`

194.1. Package information

[Link to package on Github](#)

Essentials

TODO: add authors.

Maintainer: [Liam Paull](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: description of `duckietown_demos`

CHAPTER 195

Package `duckietown_logs`

195.1. Package information

[Link to package on Github](#)

Essentials

TODO: add authors.

Maintainer: [Andrea Censi](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: Add a description of package `duckietown_logs` in `package.xml`.

195.2. Misc

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 196

Package duckietown_unit_test

196.1. Package information

[Link to package on Github](#)

Essentials

TODO: add authors.

Maintainer: [Mack](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

The `duckietown_unit_test` meta package contains all the unit test launch files for Duckietown.

CHAPTER 197

Package `veh_coordinator`

197.1. Package information

[Link to package on Github](#)

Essentials

Author: [Michal Cap](#)

Maintainer: [Mack](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

The simple vehicle coordination package

I think is used to fake the vehicle coordination for the FSM.

PART 20

Packages - To sort



We need to decide where these packages go.

PART 21

Packages - Failed projects



These packages are abandoned failed projects.

CHAPTER 198

Package bag_stamper

198.1. Package information

[Link to package on Github](#)

Essentials

Author: [Teddy Ort](#)

Maintainer: [Mack](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: Add a description of package `bag_stamper` in `package.xml`.

198.2. Node bag_stamper_node

(Generated from [configuration `bag_stamper_node.easy_node.yaml`](#).)

TODO: Missing node description in `bag_stamper_node.easy_node.yaml`.

Parameters

Parameter `bagout`: `str` ; default value: `None`

TODO: Missing description for entry “`bagout`”.

Parameter `topicin`: `str` ; default value: `'wheels_driver/wheels_cmd'`

TODO: Missing description for entry “`topicin`”.

Parameter `bagin`: `str`

TODO: Missing description for entry “`bagin`”.

Parameter `topicout`: `bool` ; default value: `True`

TODO: Missing description for entry “`topicout`”.

Subscriptions

No subscriptions defined.

Publishers

No publishers defined.

376

PACKAGE BAG_STAMPER

CHAPTER 199

Package kinematics

199.1. Package information

[Link to package on Github](#)

Essentials

TODO: add authors.

Maintainer: [Mack](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: Add a description of package `kinematics` in `package.xml`.

TODO: So many nodes here! Which ones are useful?

CHAPTER 200

Package **visual_odometry**

200.1. Package information

[Link to package on Github](#)

Essentials

TODO: add authors.

Maintainer: [Mack](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: Add a description of package `visual_odometry` in `package.xml`.

CHAPTER 201

Package mouse_encoder

Use a mouse as encoder. Requires read permission to `/dev/input/mice`.

201.1. Publish Topic

- `mouse_encoder/tick: geometry_msg/Point` message with number of ticks in the x and y direction.

201.2. Parameters

- `~dev_path`: Default to `/dev/input/mice`. Point to the device path of the mouse.

201.3. 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 202

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

202.1. 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.

202.2. 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.

202.3. 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.

202.4. 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 203
Package slam

..

CHAPTER 204

Package **street_name_detector**



CHAPTER 205

Package `adafruit_imu`

205.1. Package information

[Link to package on Github](#)

Essentials

Author: [Dmitry Yershov](#)

Maintainer: [Mack](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

The Adafruit IMU package for duckietown. Reads sensor data from adafruit IMU and publishes it to `sensor_msgs.Imu` and `sensor_msgs.MagneticField`.

205.2. Testing

To test run:

TODO

205.3. Dependencies

- `rospy`
- `sensor_msgs`: for the `Joy.msg`
- `duckietown_msgs`: for the `CarControl.msg`

205.4. 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 206

Package car_supervisor

206.1. Package information

[Link to package on Github](#)

Essentials

TODO: add authors.

Maintainer: [Mack](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

The car_supervisor package for duckietown.

CHAPTER 207

Package `scene_segmentation`

207.1. Package information

[Link to package on Github](#)

Essentials

Author: [Changyun Choi](#)

Maintainer: [Mack](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: Add a description of package `scene_segmentation` in `package.xml`.

CHAPTER 208

Package `visual_odometry_line`

208.1. Package information

[Link to package on Github](#)

Essentials

Author: [Wyatt Ubellacker](#)

Maintainer: [Mack](#)

TODO: add code to generate list of dependency.

TODO: add code to generate a link to a Github issue table for this package.

TODO: add code to show unit tests build indicator.

Descriptions

TODO: Add a description of package `lane_filter` in `package.xml`.

Page left blank