The Duckiebook

From kits of parts to an autonomous fleet
in 857 easy steps
without hiding anything

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

For searching, see the one-page version.

Table of contents

Part A - The Duckietown project ... 20
Unit A-1 - What is Duckietown? ... 21
Section 1.1 - Goals and objectives ... 21
Section 1.2 - Learn about the Duckietown educational experience ... 21
Section 1.3 - Learn about the platform ... 22
Unit A-2 - Duckietown history and future ... 23
Section 2.1 - The beginnings of Duckietown ... 23
Section 2.2 - University-level classes in 2016 ... 24
Section 2.3 - University-level classes in 2017 ... 25
Section 2.4 - Chile ... 25
Section 2.5 - Duckietown High School ... 25
Unit A-3 - Duckietown classes ... 28
Section 3.1 - 2016 ... 28
Section 3.2 - 2017 ... 29
Unit A-4 - First steps ... 31
Section 4.1 - How to get started ... 31
Section 4.2 - Duckietown for instructors ... 31
Section 4.3 - Duckietown for self-guided learners ... 31
Section 4.4 - Introduction for companies ... 31
Section 4.5 - How to keep in touch ... 31
Section 4.6 - How to contribute ... 31
Section 4.7 - Frequently Asked Questions ... 31

Part B - Duckumentation documentation ... 33
Unit B-1 - Contributing to the documentation ... 34
Section 1.1 - Where the documentation is ... 34
Section 1.2 - Editing links ... 34
Section 1.3 - Installing the documentation system ... 34
Section 1.4 - Compiling the documentation (updated Sep 12) ... 36
Section 1.5 - The workflow to edit documentation (updated Sep 12) ... 37
Section 1.6 - Reporting problems ... 37
Unit B-2 - Basic Markduck guide ... 38
Section 2.1 - Markdown ... 38
Section 2.2 - Variables in command lines and command output ... 38
Section 2.3 - Character escapes ... 38
Section 2.4 - Keyboard keys ... 38
Section 2.5 - Figures ... 39
Section 2.6 - Subfigures ... 40
Section 2.7 - Shortcut for tables ... 41
Section 2.8 - Linking to documentation ... 41
Unit B-3 - Special paragraphs and environments ... 44
Section 3.1 - Special paragraphs tags ... 44
Section 3.2 - Other `div` environments ... 47
Section 3.3 - Notes and questions ... 48
Unit B-4 - Using LaTeX constructs in documentation ... 50
Section 4.1 - Embedded LaTeX ... 50
Section 4.2 - LaTeX equations ... 52
Section 4.3 - LaTeX symbols ... 52
Section 4.4 - Bibliography support ... 52
Section 4.5 - Embedding Latex in Figures through SVG ... 53
Unit B-5 - Advanced Markduck guide ... 55
Section 5.1 - Embedding videos ... 55
Section 5.2 - `move-here` tag ... 55

Section 5.3 - Comments ... 56
Section 5.4 - Referring to Github files ... 56
Section 5.5 - Putting code from the repository in line ... 57
Unit B-6 - *Compiling the PDF version ... 59
Section 6.1 - Installing `nodejs` ... 59
Section 6.2 - Installing Prince ... 59
Section 6.3 - Installing fonts ... 59
Section 6.4 - Compiling the PDF ... 60
Unit B-7 - Markduck troubleshooting ... 61
Section 7.1 - Changes don't appear on the website ... 61
Section 7.2 - Troubleshooting errors in the compilation process ... 61
Section 7.3 - Common mistakes with Markdown ... 61
Unit B-8 - The Duckuments bot ... 63
Section 8.1 - Documentation deployment ... 63
Section 8.2 - Understand what's going on ... 63
Section 8.3 - Logging ... 63
Section 8.4 - Debugging Github Pages problems ... 64
Unit B-9 - Documentation style guide ... 65
Section 9.1 - General guidelines for technical writing ... 65
Section 9.2 - Style guide for the Duckietown documentation ... 65
Section 9.3 - Writing command lines ... 66
Section 9.4 - Frequently misspelled words ... 66
Section 9.5 - Other conventions ... 66
Section 9.6 - Troubleshooting sections ... 67
Unit B-10 - Learning in Duckietown ... 68
Section 10.1 - The learning feedback loop ... 68
Section 10.2 - Knowledge, Skills and Competences ... 69
Unit B-11 - Knowledge graph ... 72
Section 11.1 - Formalization ... 72
Section 11.2 - Atoms properties ... 73
Section 11.3 - Markdown format for text-like atoms ... 73
Section 11.4 - How to describe the semantic graphs of atoms ... 74
Section 11.5 - How to describe modules ... 75
Unit B-12 - Translations ... 76
Section 12.1 - File organization ... 76
Section 12.2 - Guidelines for English writers ... 76
Section 12.3 - File format ... 76

Part C - Operation manual - Duckiebot ... 78
Unit C-1 - Duckiebot configurations ... 79
Section 1.1 - Configuration list ... 79
Section 1.2 - Configuration functionality ... 79
Unit C-2 - Acquiring the parts for the Duckiebot `DB17-wjd` ... 81
Section 2.1 - Bill of materials ... 81
Section 2.2 - Chassis ... 82
Section 2.3 - Raspberry Pi 3 - Model B ... 82
Section 2.4 - Camera ... 84
Section 2.5 - DC Motor HAT ... 85
Section 2.6 - Battery ... 86
Section 2.7 - Standoffs, Nuts and Screws ... 86
Section 2.8 - Zip Tie ... 87
Section 2.9 - Configuration `DB17-w` ... 87
Section 2.10 - Configuration `DB17-j` ... 88
Section 2.11 - Configuration `DB17-d` ... 88
Unit C-3 - Soldering boards for `DB17` ... 90
Section 3.1 - General tips ... 90
Unit C-4 - Preparing the power cable for `DB17` ... 93
Section 4.1 - Step 1: Find a cable ... 93
Section 4.2 - Step 2: Cut the cable ... 94
Section 4.3 - Step 3: Strip the cable ... 94

Section 4.4 - Step 3: Strip the wires 95
Section 4.5 - Step 4: Find the power wires 96
Section 4.6 - Step 5: Test correct operation 96
Unit C-5 - Assembling the Duckiebot DB17 98
Section 5.1 - Chassis 98
Section 5.2 - Assembling the Raspberry Pi, camera, and HATs 103
Unit C-6 - Reproducing the image 114
Section 6.1 - Download and uncompress the Ubuntu Mate image 114
Section 6.2 - Burn the image to an SD card 114
Section 6.3 - Raspberry Pi Config 115
Section 6.4 - Install packages 115
Section 6.5 - Install Edimax driver 116
Section 6.6 - Install ROS 117
Section 6.7 - Wireless configuration (old version) 117
Section 6.8 - Wireless configuration 118
Section 6.9 - SSH server config 120
Section 6.10 - Create swap Space 120
Section 6.11 - Passwordless sudo 121
Section 6.12 - Clean up 121
Section 6.13 - Ubuntu user configuration 122
Section 6.14 - Check that all required packages were installed 123
Section 6.15 - Creating the image 123
Section 6.16 - TODO: Git LFS 124
Unit C-7 - Installing Ubuntu on laptops 125
Section 7.1 - Install Ubuntu 125
Section 7.2 - Install useful software 125
Section 7.3 - Install ROS 126
Section 7.4 - Other suggested software 126
Section 7.5 - Installation of the duckuments system 126
Section 7.6 - Passwordless `sudo` 126
Section 7.7 - SSH and Git setup 127
Unit C-8 - Duckiebot Initialization 128
Section 8.1 - Acquire and burn the image 128
Section 8.2 - Turn on the Duckiebot 129
Section 8.3 - Connect the Duckiebot to a network 129
Section 8.4 - Ping the Duckiebot 129
Section 8.5 - SSH to the Duckiebot 130
Section 8.6 - Setup network 130
Section 8.7 - Update the system 130
Section 8.8 - Give a name to the Duckiebot 130
Section 8.9 - Change the hostname 130
Section 8.10 - Expand your filesystem 131
Section 8.11 - Create your user 132
Section 8.12 - Other customizations 134
Section 8.13 - Hardware check: camera 134
Section 8.14 - Final touches: duckie logo 135
Unit C-9 - Networking aka the hardest part 137
Section 9.1 - (For `C0+w`) Configure the robot-generated network 137
Section 9.2 - Setting up wireless network configuration 138
Unit C-10 - Software setup and RC remote control 143
Section 10.1 - Clone the Duckietown repository 143
Section 10.2 - Update the system 143
Section 10.3 - Set up the ROS environment on the Duckiebot 144
Section 10.4 - Clone the duckiefleet repository 144
Section 10.5 - Add your vehicle data to the robot database 144
Section 10.6 - Test that the joystick is detected 145
Section 10.7 - Run the joystick demo 145
Section 10.8 - The proper shutdown procedure for the Raspberry Pi 146
Unit C-11 - Reading from the camera 148
Section 11.1 - Check the camera hardware 148

Section 11.2 - Create two windows ... 148
Section 11.3 - First window: launch the camera nodes ... 148
Section 11.4 - Second window: view published topics ... 149
Unit C-12 - RC control launched remotely ... 151
Section 12.1 - Two ways to launch a program ... 151
Section 12.2 - Download and setup `Software` repository on the laptop ... 151
Section 12.3 - Edit the `machines` files on your laptop ... 151
Section 12.4 - Start the demo ... 151
Section 12.5 - Watch the program output using `rqt_console` ... 152
Section 12.6 - Troubleshooting ... 152
Unit C-13 - RC+camera remotely ... 154
Section 13.1 - Assumptions ... 154
Section 13.2 - Terminal setup ... 154
Section 13.3 - First window: launch the joystick demo ... 154
Section 13.4 - Second window: launch the camera nodes ... 155
Section 13.5 - Third window: view data flow ... 155
Section 13.6 - Fourth window: visualize the image using `rviz` ... 155
Section 13.7 - Proper shutdown procedure ... 156
Unit C-14 - Interlude: Ergonomics ... 157
Section 14.1 - `set_ros_master.sh` ... 157
Section 14.2 - SSH aliases ... 157
Unit C-15 - Wheel calibration ... 159
Section 15.1 - Introduction ... 159
Section 15.2 - What is the Calibration step? ... 159
Section 15.3 - Perform the Calibration ... 160
Unit C-16 - Camera calibration ... 163
Section 16.1 - Intrinsic calibration ... 163
Section 16.2 - Extrinsic calibration ... 166
Unit C-17 - Taking a log ... 170
Section 17.1 - Verify a log ... 170

Part D - Operation manual - Duckietowns ... 171
Unit D-1 - Duckietown parts ... 172
Section 1.1 - Bill of materials ... 172
Section 1.2 - Duckies ... 172
Section 1.3 - Floor Mats ... 173
Section 1.4 - Duck Tape ... 173
Section 1.5 - Traffic Signs ... 174
Unit D-2 - Duckietown Appearance Specification ... 175
Section 2.1 - Version history ... 175
Section 2.2 - Overview ... 175
Section 2.3 - Layer 1 - The Tile Layer ... 175
Section 2.4 - Layer 2 - Signage and Lights ... 179
Section 2.5 - Traffic Signs ... 179
Section 2.6 - Street Name Signs ... 181
Section 2.7 - Traffic Lights ... 182
Unit D-3 - Signage ... 184
Unit D-4 - Traffic lights Parts ... 185
Section 4.1 - Bill of materials ... 185
Section 4.2 - Raspberry Pi ... 185
Unit D-5 - Duckietown Assembly ... 187
Unit D-6 - Traffic lights Assembly ... 188
Unit D-7 - Semantics of LEDS ... 189

Part E - Operation manual - Duckiebot with LEDs ... 190
Unit E-1 - Acquiring the parts for the Duckiebot `DB17-lc` ... 191
Section 1.1 - Bill of materials ... 191
Section 1.2 - LEDs ... 192
Section 1.3 - Bumpers ... 194

Section 1.4 - Headers, resistors and jumper 194
Section 1.5 - Caster (`DB17-c`) 195
Unit E-2 - Soldering boards for `DB17-l` 197
Section 2.1 - General rules 197
Section 2.2 - 16-channel PWM/Servo HAT 197
Section 2.3 - LSD board 198
Section 2.4 - LED connection 199
Section 2.5 - Putting everything together! 199
Unit E-3 - Assembling the Duckiebot `DB17-wjdlc` 201
Section 3.1 - Put PWM HAT with 4 standoffs on the top of Stepper Motor HAT 201
Unit E-4 - Bumper Assembly 203
Unit E-5 - `C1` (LEDs) setup 212
Section 5.1 - Connecting Wires to LEDs 212
Section 5.2 - Connecting LEDs to LSD Hat 212
Section 5.3 - Running the Wires Through the Chassis 213
Section 5.4 - Final LED tweaks, Confirm LED Function and Placement 214

Part F - Preliminaries 215
Unit F-1 - Chapter template 216
Section 1.1 - Example Title: PID control 216
Section 1.2 - Problem Definition 216
Section 1.3 - Introduced Notions 217
Section 1.4 - Examples 218
Section 1.5 - Pointers to Exercises 219
Section 1.6 - Conclusions 219
Section 1.7 - Next Steps 219
Section 1.8 - References 219
Unit F-2 - Symbols and conventions 220
Section 2.1 - Conventions 220
Section 2.2 - Spaces 221
Unit F-3 - Sets 222
Section 3.1 - Definition 222
Section 3.2 - Maps 222
Section 3.3 - Definition 222
Section 3.4 - Properties of maps 222
Unit F-4 - Numbers 223
Section 4.1 - Natural numbers 223
Section 4.2 - Integers 223
Section 4.3 - Rationals 223
Section 4.4 - Irrationals 224
Section 4.5 - Reals 224
Section 4.6 - Complex 224
Unit F-5 - Complex numbers 225
Section 5.1 - Powers of i 225
Section 5.2 - Complex conjugate 225
Unit F-6 - Linearity and Vectors 226
Section 6.1 - Linearity 226
Section 6.2 - Vectors 227
Section 6.3 - Linear dependance 232
Section 6.4 - Pointers to Exercises 232
Section 6.5 - Conclusions 232
Unit F-7 - Matrices basics 234
Section 7.1 - Matrix dimensions 234
Section 7.2 - Matrix diagonals 235
Section 7.3 - Diagonal matrix 235
Section 7.4 - Identity matrix 235
Section 7.5 - Null matrix 235
Section 7.6 - Determinant 235
Section 7.7 - Rank of a matrix 235
Section 7.8 - Trace of a matrix 236

Unit F-8 - Matrix inversions ... 237
Section 8.1 - Adjugate matrix ... 237
Section 8.2 - Matrix Inverse ... 237
Section 8.3 - Nonsingularity of a matrix ... 237
Unit F-9 - Matrices and vectors ... 238
Section 9.1 - Matrix as representation of linear (vector) spaces ... 238
Unit F-10 - Matrix operations (basic) ... 239
Unit F-11 - Matrix operations (complex) ... 240
Unit F-12 - Matrix diagonalization ... 241
Section 12.1 - Left and Right Inverse (topic for advanced-linear-algebra?) ... 241
Unit F-13 - Linearization ... 242
Section 13.1 - Linearization of a nonlinear system ... 242
Unit F-14 - Norms ... 243
Section 14.1 - Definition ... 243
Section 14.2 - Properties ... 244
Unit F-15 - From ODEs to LTI systems ... 245
Section 15.1 - LTI Systems ... 245
Unit F-16 - Probability basics ... 246
Section 16.1 - Random Variables ... 246
Unit F-17 - Kinematics ... 253
Unit F-18 - Dynamics ... 254
Unit F-19 - Coordinate systems ... 255
Section 19.1 - Motivation ... 255
Section 19.2 - Definitions ... 255
Section 19.3 - Examples ... 255
Section 19.4 - Further reading ... 259
Unit F-20 - Reference frames ... 260
Section 20.1 - Motivation ... 260
Section 20.2 - Definition ... 260
Section 20.3 - Example ... 260
Section 20.4 - Translating motions in one frame to another ... 260
Unit F-21 - Time series ... 262
Section 21.1 - Definition of time series ... 262
Section 21.2 - Operations on time series ... 262
Section 21.3 - Further reading ... 262
Unit F-22 - Transformations ... 263
Section 22.1 - Introduction ... 263
Section 22.2 - Definitions and important examples ... 263
Section 22.3 - Applications ... 263
Unit F-23 - Types ... 264
Section 23.1 - Types vs sets ... 264
Section 23.2 - Types of functions ... 264
Section 23.3 - Sum types / coproduct ... 264
Section 23.4 - In Duckietown ... 264
Section 23.5 - Historical notes ... 264
Section 23.6 - Further reading ... 264
Unit F-24 - Computer science concepts ... 265
Section 24.1 - Real-time Operating system ... 265

Part G - A course in autonomy ... 266
Unit G-1 - Autonomous Vehicles ... 267
Section 1.1 - Autonomous Vehicles in the News ... 267
Section 1.2 - Levels of Autonomy ... 267
Unit G-2 - Autonomy overview ... 269
Section 2.1 - Basic Building Blocks of Autonomy ... 269
Section 2.2 - Advanced Building Blocks of Autonomy ... 274
Unit G-3 - Modern Robotic Systems ... 276
Section 3.1 - No robot is an island ... 276
Section 3.2 - Development of modern robotic systems ... 276
Section 3.3 - Example of a typical data processing pipeline ... 277

Section 3.4 - Vignettes from an optimistic future ... 278
Section 3.5 - Take-away points ... 278
Section 3.6 - Further reading ... 278
Unit G-4 - System architectures basics ... 280
Section 4.1 - Logical and physical architecture ... 280
Section 4.2 - Logical architecture ... 280
Section 4.3 - Actor model ... 280
Section 4.4 - Physical architecture ... 280
Section 4.5 - Mapping logical architecture onto physical ... 281
Section 4.6 - Example in Duckietown ... 281
Section 4.7 - Take-away points ... 281
Section 4.8 - Further reading ... 281
Unit G-5 - Autonomy architectures ... 283
Section 5.1 - The state of the art in autonomy architectures ... 283
Section 5.2 - Further reading ... 283
Section 5.3 - Figures ... 283
Unit G-6 - Representations ... 286
Section 6.1 - Robot Representations ... 287
Section 6.2 - Environment Representations ... 288
Unit G-7 - Modern signal processing ... 290
Section 7.1 - Event-based processing ... 290
Section 7.2 - Periodic vs event-based processing ... 290
Section 7.3 - Why working with events ... 290
Section 7.4 - Definition of message statistics ... 291
Section 7.5 - On the independence of latency and frequency ... 291
Section 7.6 - Design considerations and trade-offs ... 291
Section 7.7 - Further reading ... 292
Unit G-8 - Middleware ... 293
Section 8.1 - Why using middleware ... 293
Section 8.2 - What middleware provides ... 293
Section 8.3 - A few alternatives of robotics middleware ... 293
Section 8.4 - Trade-offs and design considerations ... 293
Unit G-9 - Modularization and Contracts ... 295
Section 9.1 - Monolithic vs modular design ... 295
Section 9.2 - Contracts ... 295
Section 9.3 - Level 0: API (Types and signatures) ... 295
Section 9.4 - Level 1: Timing: latency, frequency, availability ... 295
Section 9.5 - Level 2: Resources: Computation and memory ... 295
Section 9.6 - Level 3: Semantics ... 295
Section 9.7 - Level 4: Quality ... 295
Unit G-10 - Configuration management ... 297
Section 10.1 - Motivation ... 297
Section 10.2 - Stages of configuration management: from clueless to pro ... 297
Section 10.3 - The primordial stage (clueless developer) ... 297
Section 10.4 - Recognition that parameters are important ... 297
Section 10.5 - Parameters in interfaces ... 298
Section 10.6 - Encapsulation of functionality objects ... 298
Section 10.7 - Convenient interfaces ... 299
Section 10.8 - Abstracting the configuration ... 299
Section 10.9 - Duckietown solution ... 300
Section 10.10 - Beyong: Customization, History tracking ... 300
Unit G-11 - Duckiebot modeling ... 301
Section 11.1 - Preliminaries ... 301
Section 11.2 - Kinematics ... 303
Section 11.3 - Differential drive robot kinematic model ... 304
Section 11.4 - Differential drive robot dynamic model ... 305
Section 11.5 - DC motor dynamic model ... 309
Section 11.6 - Conclusions ... 310
Unit G-12 - Odometry Calibration ... 311
Unit G-13 - Computer vision basics ... 312

Unit G-14 - Camera geometry ... 313
Unit G-15 - Camera calibration ... 314
Unit G-16 - Image filtering ... 315
Unit G-17 - Illumination invariance ... 316
Unit G-18 - Line Detection ... 317
Unit G-19 - Feature extraction ... 318
Unit G-20 - Place recognition ... 319
Unit G-21 - Filtering 1 ... 320
Unit G-22 - Filtering 2 ... 321
Unit G-23 - Mission planning ... 322
Unit G-24 - Planning in discrete domains ... 323
Unit G-25 - Motion planning ... 324
Unit G-26 - RRT ... 325
Unit G-27 - Feedback control ... 326
Unit G-28 - PID Control ... 327
Unit G-29 - MPC Control ... 328
Unit G-30 - Object detection ... 329
Unit G-31 - Object classification ... 330
Unit G-32 - Object tracking ... 331
Unit G-33 - Reacting to obstacles ... 332
Unit G-34 - Semantic segmentation ... 333
Unit G-35 - Text recognition ... 334
Unit G-36 - SLAM - Problem formulation ... 335
Unit G-37 - SLAM - Broad categories ... 336
Unit G-38 - VINS ... 337
Unit G-39 - Advanced place recognition ... 338
Unit G-40 - Fleet level planning (placeholder) ... 339
Unit G-41 - Fleet level planning (placeholder) ... 340
Unit G-42 - Bibliography ... 341

Part H - Exercises ... 343
Unit H-1 - Exercise: Basic image operations ... 344
Section 1.1 - Skills learned ... 344
Section 1.2 - Instructions ... 344
Section 1.3 - Specification of `dt-image-flip0` ... 344
Section 1.4 - Useful APIs ... 344
Unit H-2 - Exercise: Basic image operations, adult version ... 346
Section 2.1 - Skills learned ... 346
Section 2.2 - Instructions ... 346
Section 2.3 - `dt-image-flip` specification ... 346
Section 2.4 - Useful APIs ... 346
Section 2.5 - Testing it works with `image-ops-tester` ... 347
Section 2.6 - Bottom line ... 347
Unit H-3 - Exercise: Simple data analysis from a bag ... 348
Section 3.1 - Skills learned ... 348
Section 3.2 - Instructions ... 348
Section 3.3 - Specification for `dt-bag-analyze` ... 348
Section 3.4 - Useful APIs ... 348
Section 3.5 - Test that it works ... 349
Unit H-4 - Exercise: Bag in, bag out ... 350
Section 4.1 - Skills learned ... 350
Section 4.2 - Instructions ... 350
Section 4.3 - Specification of `dt-bag-decimate` ... 350
Section 4.4 - Useful new APIs ... 350
Section 4.5 - Check that it works ... 350
Unit H-5 - Exercise: Bag thumbnails ... 352
Section 5.1 - Skills learned ... 352
Section 5.2 - Instructions ... 352
Section 5.3 - Specification for `dt-bag-thumbnails` ... 352
Section 5.4 - Test data ... 352

Section 5.5 - Useful APIs ... 352
Unit H-6 - Exercise: Instagram filters ... 354
Section 6.1 - Skills learned ... 354
Section 6.2 - Instructions ... 354
Section 6.3 - Specification for `dt-instagram` ... 354
Section 6.4 - Useful new APIs ... 355
Unit H-7 - Exercise: Bag instagram ... 356
Section 7.1 - Instructions ... 356
Section 7.2 - Specification for `dt-bag-instagram` ... 356
Section 7.3 - Test data ... 356
Section 7.4 - Useful new APIs ... 356
Section 7.5 - Check that it works ... 356
Unit H-8 - Exercise: Live Instagram ... 358
Section 8.1 - Skills learned ... 358
Section 8.2 - Instructions ... 358
Section 8.3 - Specification for the program `dt-live-instagram` ... 358
Section 8.4 - Check that it works ... 358
Unit H-9 - Exercise: Git and conventions ... 359
Unit H-10 - Exercise: Use our API for arguments ... 360
Section 10.1 - Skills learned ... 360
Section 10.2 - Instructions ... 360
Section 10.3 - Useful new API learned ... 360
Unit H-11 - Exercise: Bouncing ball ... 361
Section 11.1 - Skills learned ... 361
Section 11.2 - Instructions ... 361
Section 11.3 - Useful new API learned ... 361
Unit H-12 - Exercise: Visualizing data on image ... 362
Section 12.1 - Skills learned ... 362
Section 12.2 - Instruction ... 362
Section 12.3 - Specification for `bag-mark-spots` ... 362
Unit H-13 - Exercise: Make that into a node ... 363
Section 13.1 - Learned skills ... 363
Section 13.2 - Instructions ... 363
Unit H-14 - Exercise: Instagram with EasyAlgo interface ... 364
Section 14.1 - Learned skills ... 364
Section 14.2 - Instructions ... 364
Section 14.3 - See documentation ... 364
Section 14.4 - Use in your code ... 364
Unit H-15 - Milestone: Illumination invariance (anti-instagram) ... 365
Unit H-16 - Exercise: Launch files basics ... 366
Section 16.1 - Learned skills ... 366
Unit H-17 - Exercise: Unit tests ... 367
Section 17.1 - Learned skills ... 367
Section 17.2 - Unit tests with nosetests ... 367
Section 17.3 - Unite tests with ROS tests ... 367
Section 17.4 - Unite tests with comptests ... 367
Unit H-18 - Exercise: Parameters (standard ROS api) ... 368
Section 18.1 - Learned skills ... 368
Unit H-19 - Exercise: Parameters (our API) ... 369
Section 19.1 - Learned skills ... 369
Unit H-20 - Exercise: Place recognition abstraction ... 370
Section 20.1 - Learned skills ... 370
Section 20.2 - Instructions ... 370
Section 20.3 - Try a basic feature: ... 370
Section 20.4 - Bottom line ... 370
Unit H-21 - Exercise: Parallel processing ... 371
Section 21.1 - Learned skills ... 371
Section 21.2 - Instructions ... 371
Unit H-22 - Exercise: Adding new test to integration tests ... 372
Section 22.1 - Learned skills ... 372

Section 22.2 - Instructions ... 372
Section 22.3 - Bottom line ... 372
Section 22.4 - Milestone: Lane following ... 372
Unit H-23 - Exercise: localization ... 373
Unit H-24 - Exercise: Ducks in a row ... 374
Unit H-25 - Exercise: Comparison of PID ... 375
Unit H-26 - Exercise: RRT ... 376
Unit H-27 - Exercise: Who watches the watchmen? (optional) ... 377
Section 27.1 - Skills learned ... 377
Section 27.2 - Instructions ... 377
Section 27.3 - `image-ops-tester-tester` specification ... 377
Section 27.4 - Testing it works with `image-ops-tester-tester-tester` ... 377
Section 27.5 - Bottom line ... 377

Part I - Software reference ... 378
Unit I-1 - Ubuntu packaging with APT ... 379
Section 1.1 - `apt install` ... 379
Section 1.2 - `apt update` ... 379
Section 1.3 - `apt-key` ... 379
Section 1.4 - `apt-mark` ... 379
Section 1.5 - `add-apt-repository` ... 379
Section 1.6 - `wajig` ... 379
Section 1.7 - `dpigs` ... 379
Unit I-2 - GNU/Linux general notions ... 380
Section 2.1 - Background reading ... 380
Unit I-3 - Every day Linux ... 381
Section 3.1 - `cd` ... 381
Section 3.2 - `sudo` ... 381
Section 3.3 - `ls` ... 381
Section 3.4 - `cp` ... 381
Section 3.5 - `mkdir` ... 381
Section 3.6 - `touch` ... 381
Section 3.7 - `reboot` ... 381
Section 3.8 - `shutdown` ... 381
Section 3.9 - `rm` ... 381
Unit I-4 - Users ... 382
Section 4.1 - `passwd` ... 382
Unit I-5 - UNIX tools ... 383
Section 5.1 - `cat` ... 383
Section 5.2 - `tee` ... 383
Section 5.3 - `truncate` ... 383
Unit I-6 - Linux disks and files ... 384
Section 6.1 - `fdisk` ... 384
Section 6.2 - `mount` ... 384
Section 6.3 - `umount` ... 384
Section 6.4 - `losetup` ... 384
Section 6.5 - `gparted` ... 384
Section 6.6 - `dd` ... 384
Section 6.7 - `sync` ... 384
Section 6.8 - `df` ... 384
Section 6.9 - How to make a partition ... 384
Unit I-7 - Other administration commands ... 385
Section 7.1 - `visudo` ... 385
Section 7.2 - `update-alternatives` ... 385
Section 7.3 - `udevadm` ... 385
Section 7.4 - `systemctl` ... 385
Unit I-8 - Make ... 386
Section 8.1 - `make` ... 386
Unit I-9 - Python-related tools ... 387
Section 9.1 - `virtualenv` ... 387

Section 9.2 - `pip` ... 387
Unit I-10 - Raspberry-PI commands ... 388
Section 10.1 - `raspi-config` ... 388
Section 10.2 - `vcgencmd` ... 388
Section 10.3 - `raspistill` ... 388
Section 10.4 - `jstest` ... 388
Section 10.5 - `swapon` ... 388
Section 10.6 - `mkswap` ... 388
Unit I-11 - Users and permissions ... 389
Section 11.1 - `chmod` ... 389
Section 11.2 - `groups` ... 389
Section 11.3 - `adduser` ... 389
Section 11.4 - `useradd` ... 389
Unit I-12 - Downloading ... 390
Section 12.1 - `curl` ... 390
Section 12.2 - `wget` ... 390
Section 12.3 - `sha256sum` ... 390
Section 12.4 - `xz` ... 390
Unit I-13 - Shells and environments ... 391
Section 13.1 - `source` ... 391
Section 13.2 - `which` ... 391
Section 13.3 - `export` ... 391
Unit I-14 - Other misc commands ... 392
Section 14.1 - `pgrep` ... 392
Section 14.2 - `npm` ... 392
Section 14.3 - `nodejs` ... 392
Section 14.4 - `ntpdate` ... 392
Section 14.5 - `chsh` ... 392
Section 14.6 - `echo` ... 392
Section 14.7 - `sh` ... 392
Section 14.8 - `fc-cache` ... 392
Unit I-15 - Linux resources usage ... 393
Section 15.1 - Measuring CPU usage using `htop` ... 393
Section 15.2 - Measuring I/O usage using `iotop` ... 393
Section 15.3 - How fast is the SD card? ... 393
Unit I-16 - SD Cards tools ... 394
Section 16.1 - Testing SD Card and disk speed ... 394
Section 16.2 - How to burn an image to an SD card ... 394
Section 16.3 - How to shrink an image ... 395
Unit I-17 - Networking tools ... 398
Section 17.1 - `hostname` ... 398
Section 17.2 - Visualizing information about the network ... 398
Unit I-18 - Accessing computers using SSH ... 399
Section 18.1 - Background reading ... 399
Section 18.2 - Installation of SSH ... 399
Section 18.3 - Local configuration ... 399
Section 18.4 - How to login with SSH and a password ... 400
Section 18.5 - Creating an SSH keypair ... 400
Section 18.6 - How to login without a password ... 402
Section 18.7 - Fixing SSH Permissions ... 403
Section 18.8 - `ssh-keygen` ... 403
Unit I-19 - Wireless networking in Linux ... 404
Section 19.1 - `iwconfig` ... 404
Section 19.2 - `iwlist` ... 404
Unit I-20 - Moving files between computers ... 406
Section 20.1 - SCP ... 406
Section 20.2 - RSync ... 406
Unit I-21 - VIM ... 407
Section 21.1 - External documentation ... 407
Section 21.2 - Installation ... 407

Section 21.3 - `vi` ... 407
Section 21.4 - Suggested configuration ... 407
Section 21.5 - Visual mode ... 407
Section 21.6 - Indenting using VIM ... 407
Unit I-22 - Atom ... 409
Section 22.1 - Install Atom ... 409
Unit I-23 - Liclipse ... 410
Section 23.1 - Why using IDEs ... 410
Section 23.2 - Other alternatives ... 410
Section 23.3 - Installing LiClipse ... 410
Section 23.4 - Set shortcuts for LiClipse ... 411
Section 23.5 - Shortcuts for LiClipse ... 411
Section 23.6 - Other configuration for Liclipse ... 411
Unit I-24 - Slack ... 412
Section 24.1 - Installing the Slack app on Linux ... 412
Section 24.2 - Disabling Slack email notifications ... 412
Section 24.3 - Disabling Slack pop-up notification on the desktop ... 412
Unit I-25 - Byobu ... 413
Section 25.1 - Advantages of using Byobu ... 413
Section 25.2 - Installation ... 413
Section 25.3 - Documentation ... 413
Section 25.4 - Quick command reference ... 413
Section 25.5 - Commands on OS X ... 414
Unit I-26 - Source code control with Git ... 415
Section 26.1 - Background reading ... 415
Section 26.2 - Installation ... 415
Section 26.3 - Setting up global configurations for Git ... 415
Section 26.4 - Git tips ... 415
Section 26.5 - Git troubleshooting ... 416
Section 26.6 - `git` ... 417
Section 26.7 - `hub` ... 417
Unit I-27 - Git LFS ... 418
Section 27.1 - Generic installation instructions ... 418
Section 27.2 - Ubuntu 16 installation (laptop) ... 418
Section 27.3 - Raspberry Pi 3 ... 418
Section 27.4 - Troubleshooting ... 418
Unit I-28 - Setup Github access ... 419
Section 28.1 - Create a Github account ... 419
Section 28.2 - Become a member of the Duckietown organization ... 419
Section 28.3 - Add a public key to Github ... 419
Unit I-29 - ROS installation and reference ... 421
Section 29.1 - Install ROS ... 421
Section 29.2 - `rqt_console` ... 421
Section 29.3 - `roslaunch` ... 422
Section 29.4 - `rviz` ... 422
Section 29.5 - `rostopic` ... 422
Section 29.6 - `catkin_make` ... 422
Section 29.7 - `rosrun` ... 422
Section 29.8 - `rostest` ... 422
Section 29.9 - `rospack` ... 422
Section 29.10 - `rosparam` ... 422
Section 29.11 - `rosdep` ... 423
Section 29.12 - `roswtf` ... 423
Section 29.13 - `rosbag` ... 423
Section 29.14 - `roscore` ... 424
Section 29.15 - Troubleshooting ROS ... 424
Section 29.16 - Other materials about ROS. ... 424

Part J - Software development guide ... 425
Unit J-1 - Python ... 426

Section 1.1 - Background reading 426
Section 1.2 - Python virtual environments 426
Section 1.3 - Useful libraries 426
Section 1.4 - Context managers 426
Section 1.5 - Exception hierarchies 426
Section 1.6 - Object orientation - Abstract classes, class hierarchies 426
Section 1.7 - Downloading resources 426
Section 1.8 - IPython 427
Section 1.9 - Idioms 427
Unit J-2 - Working with YAML files 428
Section 2.1 - Pointers to documentation 428
Section 2.2 - Editing YAML files 428
Section 2.3 - Reading and writing YAML files in Python 428
Section 2.4 - Duckietown wrapping API 428
Unit J-3 - Duckietown code conventions 429
Section 3.1 - Python 429
Section 3.2 - Logging 430
Section 3.3 - Exceptions 430
Section 3.4 - Scripts 430
Unit J-4 - Configuration 431
Section 4.1 - Environment variables (updated Sept 12) 431
Section 4.2 - The “scuderia” (vehicle database) 431
Section 4.3 - The `machines` file 432
Section 4.4 - People database 432
Section 4.5 - Modes of operation 433
Unit J-5 - Node configuration mechanisms 434
Unit J-6 - Minimal ROS node - `pkg_name` 435
Section 6.1 - The files in the package 435
Section 6.2 - Writing a node: `talker.py` 436
Section 6.3 - The `Talker` class 438
Section 6.4 - Launch File 439
Section 6.5 - Testing the node 440
Section 6.6 - Documentation 442
Section 6.7 - Guidelines 442
Unit J-7 - Makefile system 443
Section 7.1 - User guide 443
Section 7.2 - Makefile organization 443
Section 7.3 - Makefile help 444
Unit J-8 - ROS package verification 446
Section 8.1 - Naming 446
Section 8.2 - `package.xml` 446
Section 8.3 - Messages 446
Section 8.4 - Readme file 446
Section 8.5 - Launch files 446
Section 8.6 - Test files 446
Unit J-9 - Duckietown utility library 447
Section 9.1 - Images 447
Unit J-10 - Creating unit tests with ROS 449
Unit J-11 - Continuous integration 450
Section 11.1 - Never break the build 450
Section 11.2 - How to stay in the green 450
Section 11.3 - How to make changes to `master`: pull requests 451

Part K - Duckietown system 454
Unit K-1 - Teleoperation 455
Section 1.1 - Implementation 455
Section 1.2 - Camera 455
Section 1.3 - Actuators 455
Section 1.4 - IMU 455
Unit K-2 - Parallel autonomy 456

Unit K-3 - Lane control ... 457
Section 3.1 - Implementation ... 457
Unit K-4 - Indefinite navigation ... 458
Section 4.1 - Implementation ... 458
Unit K-5 - Planning ... 459
Section 5.1 - Implementation ... 459
Unit K-6 - Coordination ... 460
Section 6.1 - Implementation ... 460
Unit K-7 - Duckietown ROS Guidelines ... 461
Section 7.1 - Node and Topics ... 461
Section 7.2 - Parameters ... 461
Section 7.3 - Launch file ... 461

Part L - Fall 2017 ... 463
Unit L-1 - The Fall 2017 Duckietown experience ... 464
Section 1.1 - The rules of Duckietown ... 464
Unit L-2 - First Steps in Duckietown ... 465
Section 2.1 - Onboarding Procedure ... 465
Section 2.2 - (optional) Add Duckietown Engineering Linkedin profile ... 466
Section 2.3 - Laptops ... 466
Section 2.4 - Next steps for people in Zurich ... 467
Section 2.5 - Next steps for people in Chicago ... 467
Unit L-3 - Logistics for Zürich branch ... 468
Section 3.1 - HR ... 468
Section 3.2 - Website / class journal ... 468
Section 3.3 - Duckiebox ... 468
Section 3.4 - Duckietown room access ... 468
Section 3.5 - Extra spaces ... 468
Unit L-4 - Logistics for Montréal branch ... 470
Section 4.1 - Website ... 470
Section 4.2 - Class Schedule ... 470
Section 4.3 - Lab Access ... 470
Section 4.4 - The Local Staff ... 470
Section 4.5 - Storing Your Robot ... 470
Unit L-5 - Logistics for Chicago branch ... 471
Unit L-6 - Git usage guide for Fall 2017 ... 472
Section 6.1 - Repositories ... 472
Section 6.2 - Git policy for homeworks ... 473
Section 6.3 - Git policy for project development ... 474
Unit L-7 - Organization ... 475
Section 7.1 - The Activity Tracker ... 475
Section 7.2 - The Areas sheet ... 477
Unit L-8 - Getting and giving help ... 478
Section 8.1 - *Who* to ask for help ... 478
Section 8.2 - *How* to ask for help ... 478
Unit L-9 - Slack Channels ... 480
Section 9.1 - Channels ... 480
Unit L-10 - Zurich branch diary ... 481
Section 10.1 - Wed Sep 20: Welcome to Duckietown! ... 481
Section 10.2 - Monday Sep 25: Introduction to autonomy ... 481
Section 10.3 - Monday Sep 25, late at night: Onboarding instructions ... 482
Section 10.4 - Wednesday Sep 27: Duckiebox distribution, and getting to know each other ... 482
Section 10.5 - (Template for every lecture) Date: Topic ... 483
Unit L-11 - Montréal branch diary ... 484
Section 11.1 - Wed Sept 6 ... 484
Section 11.2 - Friday Sept 8 ... 484
Section 11.3 - Sun Sept 10 ... 484
Section 11.4 - Mon Sept 11 ... 484
Section 11.5 - Wed Sept 13 ... 484
Section 11.6 - Mon Sept 18 ... 485

Section 11.7 - Wed Sept 20 ... 485
Section 11.8 - Mon Sept 25 ... 485
Section 11.9 - Wed Sept 27 ... 486
Unit L-12 - Chicago branch Diary ... 487
Section 12.1 - Monday September 25: Introduction to Duckietown ... 487
Section 12.2 - Tuesday September 26 ... 487
Section 12.3 - Wednesday, September 27 ... 487
Unit L-13 - Guide for TAs ... 488
Section 13.1 - Dramatis personae ... 488
Section 13.2 - First steps ... 488
Unit L-14 - Checkoff: Duckiebot Assembly and Configuration ... 490
Section 14.1 - Pick up your Duckiebox ... 490
Section 14.2 - Soldering your boards ... 490
Section 14.3 - Assemble your Robot ... 490
Section 14.4 - Optional: Reproduce the SD Card Image ... 490
Section 14.5 - Setup your laptop ... 490
Section 14.6 - Make your robot move ... 491
Unit L-15 - Checkoff: Take a Log ... 492
Section 15.1 - Take a Log ... 492
Section 15.2 - Upload the log ... 492
Unit L-16 - Homework: Data Processing ... 493
Unit L-17 - Guide for mentors ... 494
Unit L-18 - Project proposals ... 495
Unit L-19 - Template of a project ... 496

Part M - Packages - Infrastructure ... 497
Unit M-1 - Package `duckieteam` ... 498
Section 1.1 - Package information ... 498
Section 1.2 - `create-machines` ... 498
Section 1.3 - `create-roster` ... 498
Unit M-2 - Package `duckietown` ... 499
Section 2.1 - Package information ... 499
Unit M-3 - Package `duckietown_msgs` ... 500
Section 3.1 - Package information ... 500
Unit M-4 - Package `easy_algo` ... 501
Section 4.1 - Package information ... 501
Section 4.2 - Motivation ... 501
Section 4.3 - Usage ... 501
Section 4.4 - User interface ... 502
Section 4.5 - The developer's point of view ... 502
Unit M-5 - Package `easy_logs` ... 504
Section 5.1 - Package information ... 504
Section 5.2 - Command-line utilities ... 504
Section 5.3 - Selector language ... 505
Section 5.4 - Automatic log download using `require` ... 506
Section 5.5 - Browsing the cloud ... 507
Section 5.6 - Advanged log indexing and generation ... 507
Section 5.7 - How to use a file from the cloud ... 509
Unit M-6 - Package `easy_node` ... 510
Section 6.1 - Package information ... 510
Section 6.2 - YAML file format ... 510
Section 6.3 - Using the `easy_node` API ... 512
Section 6.4 - Configuration using `easy_node`: the user's point of view ... 516
Section 6.5 - Visualizing the configuration database ... 518
Section 6.6 - Benchmarking ... 520
Section 6.7 - Automatic documentation generation ... 521
Section 6.8 - Parameters and services defined for all packages ... 522
Section 6.9 - Other ideas ... 522
Unit M-7 - Package `easy_regression` ... 523
Section 7.1 - Package information ... 523

Section 7.2 - Design goals ... 523
Section 7.3 - Formalization ... 523
Section 7.4 - Example of configuration file ... 525
Section 7.5 - Language for the conditions to check ... 525
Unit M-8 - Package `what_the_duck` ... 527
Section 8.1 - Package information ... 527
Section 8.2 - Adding more tests to `what-the-duck` ... 527
Section 8.3 - Tests already added ... 527
Section 8.4 - List of tests to add ... 528

Part N - Packages - Teleoperation ... 530
Unit N-1 - Package `adafruit_drivers` ... 531
Section 1.1 - Package information ... 531
Unit N-2 - Package `dagu_car` ... 532
Section 2.1 - Package information ... 532
Section 2.2 - Node `forward_kinematics_node` ... 532
Section 2.3 - Node `inverse_kinematics_node` ... 532
Section 2.4 - Node `velocity_to_pose_node` ... 533
Section 2.5 - Node `car_cmd_switch_node` ... 533
Section 2.6 - Node `wheels_driver_node` ... 533
Section 2.7 - Node `wheels_trimmer_node` ... 534
Unit N-3 - Package `joy_mapper` ... 535
Section 3.1 - Package information ... 535
Section 3.2 - Testing ... 535
Section 3.3 - Dependencies ... 535
Section 3.4 - Node `joy_mapper_node2` ... 535
Unit N-4 - Package `pi_camera` ... 537
Section 4.1 - Package information ... 537
Section 4.2 - Node `camera_node_sequence` ... 537
Section 4.3 - Node `camera_node_continuous` ... 537
Section 4.4 - Node `decoder_node` ... 538
Section 4.5 - Node `img_process_node` ... 538
Section 4.6 - Node `cam_info_reader_node` ... 538

Part O - Packages - Lane control ... 540
Unit O-1 - Package `anti_instagram` ... 541
Section 1.1 - Package information ... 541
Section 1.2 - Unit tests integrated with `rostest` ... 541
Section 1.3 - Unit tests needed external files ... 541
Section 1.4 - Node `anti_instagram_node` ... 541
Unit O-2 - Package `complete_image_pipeline` ... 543
Section 2.1 - Package information ... 543
Section 2.2 - Program `single_image_pipeline` ... 543
Unit O-3 - Package `ground_projection` ... 545
Section 3.1 - Package information ... 545
Section 3.2 - Node `ground_projection_node` ... 545
Unit O-4 - Package `lane_control` ... 546
Section 4.1 - Package information ... 546
Section 4.2 - `lane_controller_node` ... 546
Unit O-5 - Package `lane_filter` ... 548
Section 5.1 - Package information ... 548
Section 5.2 - `lane_filter_node` ... 548
Unit O-6 - Package `line_detector2` ... 551
Section 6.1 - Package information ... 551
Section 6.2 - Testing the line detector using visual inspection ... 551
Section 6.3 - Quantitative tests ... 552
Section 6.4 - `line_detector_node2` ... 553
Unit O-7 - Package `line_detector` ... 555
Section 7.1 - Package information ... 555

Part P - Packages - Indefinite navigation 556
Unit P-1 - AprilTags library 557
Section 1.1 - Package information 557
Unit P-2 - Package `apriltags_ros` 559
Section 2.1 - Package information 559
Unit P-3 - Package `fsm` 560
Section 3.1 - Package information 560
Section 3.2 - Node `fsm_node` 560
Unit P-4 - Package `indefinite_navigation` 562
Section 4.1 - Package information 562
Unit P-5 - Package `intersection_control` 563
Section 5.1 - Package information 563
Unit P-6 - Package `navigation` 564
Section 6.1 - Package information 564
Unit P-7 - Package `stop_line_filter` 565
Section 7.1 - Package information 565

Part Q - Packages - Localization and planning 566
Unit Q-1 - Package `duckietown_description` 567
Section 1.1 - Package information 567
Unit Q-2 - Package `localization` 568
Section 2.1 - Package information 568

Part R - Packages - Coordination 569
Unit R-1 - Package `led_detection` 570
Section 1.1 - Package information 570
Section 1.2 - LED detector 570
Section 1.3 - Unit tests 571
Unit R-2 - Package `led_emitter` 572
Section 2.1 - Package information 572
Section 2.2 - In depth 572
Unit R-3 - Package `led_interpreter` 574
Section 3.1 - Package information 574
Unit R-4 - Package `led_joy_mapper` 575
Section 4.1 - Package information 575
Unit R-5 - Package `rgb_led` 576
Section 5.1 - Package information 576
Section 5.2 - Demos 576
Unit R-6 - Package `traffic_light` 577
Section 6.1 - Package information 577

Part S - Packages - Additional functionality 578
Unit S-1 - Package `mdoap` 579
Section 1.1 - Package information 579
Unit S-2 - Package `parallel_autonomy` 580
Section 2.1 - Package information 580
Unit S-3 - Package `vehicle_detection` 581
Section 3.1 - Package information 581

Part T - Packages - Templates 582
Unit T-1 - Package `pkg_name` 583
Section 1.1 - Package information 583
Section 1.2 - How to use this template 583
Unit T-2 - Package `rostest_example` 584
Section 2.1 - Package information 584

Part U - Packages - Convenience 585

Unit U-1 - Package `duckie_rr_bridge` 586
Section 1.1 - Package information 586
Unit U-2 - Package `duckiebot_visualizer` 587
Section 2.1 - Package information 587
Section 2.2 - Node `duckiebot_visualizer` 587
Unit U-3 - Package `duckietown_demos` 588
Section 3.1 - Package information 588
Unit U-4 - Package `duckietown_logs` 589
Section 4.1 - Package information 589
Section 4.2 - Misc 589
Unit U-5 - Package `duckietown_unit_test` 590
Section 5.1 - Package information 590
Unit U-6 - Package `veh_coordinator` 591
Section 6.1 - Package information 591
Unit U-7 - Last modified 592

Part A
The Duckietown project

Unit A-1
What is Duckietown?

1.1. Goals and objectives

Duckietown is a robotics education and outreach effort.

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

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

We have two targets:

1. For **instructors**, we want to create a "class-in-a-box" that allows people to offer a modern and engaging learning experience. Currently, this is feasible at the advanced undergraduate and graduate level, though in the future we would like to provide a platform that can be adapted to a range of different grade and experience levels.
2. For **self-guided learners**, we want to create a "self-learning experience" that allows students to go from having zero knowledge of robotics to a graduate-level understanding.

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

1.2. Learn about the Duckietown educational experience

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

Figure 1.1. The Duckumentary, created by Chris Welch.

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

Figure 1.2. The road to autonomy

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

1.3. Learn about the platform

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

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

Figure 1.3. Duckietown functionality

Unit A-2
Duckietown history and future

2.1. The beginnings of Duckietown

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

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

Duckietown was built by elves (Figure 2.2).

+ Resource error

```
Cannot open URL
| Traceback (most recent call last):
|   File "/home/duckietown/scm/duckuments/mcdp/src/mcdp_docs/videos.py", line 110, in
get_vimeo_info
|     response = urllib2.urlopen(url)
|   File "/usr/lib/python2.7/urllib2.py", line 154, in urlopen
|     return opener.open(url, data, timeout)
|   File "/usr/lib/python2.7/urllib2.py", line 435, in open
|     response = meth(req, response)
|   File "/usr/lib/python2.7/urllib2.py", line 548, in http_response
|     'http', request, response, code, msg, hdrs)
|   File "/usr/lib/python2.7/urllib2.py", line 473, in error
|     return self._call_chain(*args)
|   File "/usr/lib/python2.7/urllib2.py", line 407, in _call_chain
|     result = func(*args)
|   File "/usr/lib/python2.7/urllib2.py", line 556, in http_error_default
|     raise HTTPError(req.get_full_url(), code, msg, hdrs, fp)
| HTTPError: HTTP Error 503: Service Unavailable
```

Figure 2.2. The elves of Duckietown

These are some advertisement videos we used.

Figure 2.3. The need for autonomy

Figure 2.4. Advertisement

Figure 2.5. Cool Duckietown by night

2.2. University-level classes in 2016

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

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

Figure 2.6. Duckietown at NCTU in 2016

2.3. University-level classes in 2017

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

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

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

- Rennselaer Polytechnic Institute (Jeff Trinkle)
- National Chiao Tung University, Taiwan - Prof. Yon-Ping Chen's *Dynamic system simulation and implementation* course.
- Chosun University, Korea - Prof. Woosuk Sung's course;
- Petra Christian University, Indonesia - Prof. Resmana Lim's *Mobile Robot Design Course*
- National Tainan Normal University, Taiwan - Prof. Jen-Jee Chen's *Vehicle to Everything* (V2X) course; and
- Yuan Zhu University, Taiwan - Prof. Kan-Lin Hsiung's Control course.

2.4. Chile

TODO: to write

2.5. Duckietown High School

1) Introduction

DuckietownHS is inspired by the Duckietown project and targeted for high schools. The goal is to build and program duckiebots capable of moving autonomously on the streets of Duckietown. The technical objectives of DuckietownHS are simplified compared to those of the Duckietown project intended for universities so it is perfectly suited to the technical knowledge of the classes

involved. The purpose is to create self-driving DuckiebotHS vehicles which can make choices and move autonomously on the streets of Duckietown, using sensors installed on the vehicles and special road signs positioned within Duckietown.

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

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

2) Purpose

The project has two main purposes:

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

3) Perlatecnica's role

Perlatecnica assumes the role of the customer and commissions the Duckietown Engineering company to design and construct the Duckietown and DuckiebotHS. It will provide all necessary product requirements and will assume the responsibility to validate the compliancy of all deliverables to the required specifications.

4) The details of the project

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

5) Where to start

The purchase of the necessary materials is the first step to take. For both DTHS and DBHS a list of these materials is provided with links to possible sellers. Even though Amazon is typically indicated as a seller this is nothing more than an indication to facilitate the purchase for those less experienced. It is left to the individual DTE to choose where to buy the required parts. It is allowed to buy and

use parts that are not on the list but this is not recommended as they will make the Duckiebot unfit to enter in official competitions. When necessary an assembly tutorial will be provided together with the list of materials. Once the DTHS city and the DBHS robots have been assembled, the next step will be the development of the software for the running of both the city and the DuckiebotHS. The city and the Duckiebot run on a board based on a microcontroller STM32 from STMicroelectronics the Nucleo F401RE that will be programmed via the online development environment mbed. Perlatecnica will not release any of the official codes necessary for the navigation of the DuckiebotHS as these are owned by the DTE who developed them. The full standard document is available on the project official web site.

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

6) The first mission of the Duckiebot

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

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

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

The main functions are those of perception and movement.

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

Unit A-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 3.1. 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) Rennselaer 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

Location: Chicago

Course title: TTIC 31240 Self-driving Vehicles: Models and Algorithms for Autonomy

Instructors: Matthew R. Walter

Educational Level: Undergraduate/Graduate

Time period: Sept. - Dec. 2017

Link: Official Course Website

Summary: 12 students admitted from TTI-Chicago and the University of Chicago

Highlights:

UNIT A-4

First steps

4.1. How to get started

If you are an instructor, please jump to Section 4.2 - Duckietown for instructors.

If you are a self-guided learner, please jump to Section 4.3 - Duckietown for self-guided learners.

If you are a company, and interested in working with Duckietown, please jump to Section 4.4 - Introduction for companies.

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 B

Duckumentation documentation

UNIT B-1
Contributing to the documentation

1.1. Where the documentation is

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

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

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

- a publication-quality PDF;
- an online HTML version, split in multiple pages;
- a one-page version.

1.2. Editing links

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

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

1.3. Installing the documentation system

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

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

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

1) Dependencies (Ubuntu 16.04)

On Ubuntu 16.04, these are the dependencies to install:

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

2) Download the duckuments repo

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

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

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

If it fails, it means that you do not have Git LFS installed. See Unit I-27 - Git LFS.

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

3) Setup the virtual environment

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

Change into that directory:

```
$ cd ~/duckuments
```

Create the virtual environment using `virtualenv`:

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

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

1.4. Compiling the documentation (updated Sep 12)

Check before you continue

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

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

To compile the master versions of the docs, run:

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

To see the result, open the file

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

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

```
$ make master
```

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

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

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

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

To see the result, open the file

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

For incremental compilation, use:

```
$ make fall2017
```

2) Single-file compilation

There is also the option to compile one single file.
To do this, use:

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

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

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

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

This is the basic workflow:

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

➡ Create a pull request from the command-line using `hub`.

1.6. Reporting problems

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

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

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

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

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

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

UNIT B-2
Basic Markduck guide

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

2.1. Markdown

The Duckiebook is written in a Markdown dialect.

➡ A tutorial on Markdown.

2.2. Variables in command lines and command output

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

example

For example, to obtain:

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

Use the following:

```
For example, to obtain:

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

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

2.3. Character escapes

Use the string "`$`" to write the dollar symbol "$", otherwise it gets confused with LaTeX math materials. Also notice that you should probably use "USD" to refer to U.S. dollars.
Other symbols to escape are shown in Table 2.1.

TABLE 2.1. SYMBOLS TO ESCAPE

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

2.4. Keyboard keys

Use the `kbd` element for keystrokes.

example

For example, to obtain:

Press a then Ctrl-C.

use the following:

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

2.5. Figures

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

For example:

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

It will create HMTL of the form:

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

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

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

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

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

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

To refer to the figure, use an empty link:

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

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

2.6. Subfigures

You can also create subfigures, using the following syntax.

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

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

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

This is the result:

first subfig

(a) Caption 1

second subfig

(b) Caption 2

Figure 2.1. Caption of big figure

By default, the subfigures are displayed one per line.

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

first subfig

(a) Caption 1

second subfig

(b) Caption 2

Figure 2.2. Caption of big figure

2.7. Shortcut for tables

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

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

gives the following result:

TABLE 2.2. MY TABLE

A	B
C	D

1) labels-row1 and labels-row1

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

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

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

TABLE 2.3. USING CLASS="LABELS-COL1"

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

TABLE 2.4. USING CLASS="LABELS-ROW1"

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

2.8. Linking to documentation

1) Establishing names of headers

You give IDs to headers using the format:

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

For example, for this subsection, we have used:

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

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

2) How to name IDs - and why it's not automated

Some time ago, if there was a section called

```
## My section
```

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

This behavior has been removed, for several reasons.

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

```
## My better section
```

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

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

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

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

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

E.g. if you make a section called

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

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

✔

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

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

3) Linking from the documentation to the documentation

You can use the syntax:

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

to refer to the header.

You can also use some slightly more complex syntax that also allows to link to only the name, only the number or both (Table 2.5).

TABLE 2.5. SYNTAX FOR REFERRING TO SECTIONS.

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

See Subsection 2.8.1 - Establishing names of headers

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

See Establishing names of headers.

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

See 2.8.1.

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

See Subsection 2.8.1 - Establishing names of headers.

4) Linking to the documentation from outside the documentation

You are encouraged to put links to the documentation from the code or scripts.
To do so, use links of the form:

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

Here "`dth`" stands for "Duckietown Help". This link will get redirected to the corresponding document on the website.
For example, you might have a script whose output is:

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

When the user clicks on the link, they will be redirected to Section 4.2 - The "scuderia" (vehicle database).

Unit B-3
Special paragraphs and environments

3.1. Special paragraphs tags

The system supports parsing of some special paragraphs.

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

1) Special paragraphs must be separated by a line

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

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

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

For example, this is invalid:

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

This is correct:

```
See: this book

See: this other book
```

Similarly, this is invalid:

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

and this is correct:

```
Author: author

Maintainer: maintainer
```

2) Todos, task markers

```
TODO: todo
```

TODO: todo

```
TOWRITE: towrite
```

To write: towrite

```
Task: task
```

Task: task

```
Assigned: assigned
```

> **Assigned to:** assigned

3) Notes and remarks

```
Remark: remark
```

> **Remark:** remark

```
Note: note
```

> **Note:** note

```
Warning: warning
```

Warning: warning

4) Troubleshooting

```
Symptom: symptom
```

> **Symptom:** symptom

```
Resolution: resolution
```

Resolution: resolution

5) Guidelines

```
Bad: bad
```

✖ bad

```
Better: better
```

✔ better

6) Questions and answers

```
Q: question
```

Q: question

```
A: answer
```

Answer: answer

7) Authors, maintainers, Point of Contact

```
Maintainer: maintainer
```

Maintainer: maintainer

```
Author: author
```

Author: author

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

Point of contact: Point of Contact name

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

Slack channel: slack channel name

8) References

```
See: see
```

➡ see

```
Reference: reference
```

➡ reference

```
Requires: requires
```

Requires: requires

```
Results: results
```

Results: results

```
Next steps: next steps
```

Next: next steps

```
Recommended: recommended
```

Recommended: recommended

```
See also: see also
```

☀ see also

3.2. Other `div` environments

For these, note the rules:

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

1) Example usage

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

This is how you can use `rosbag`:

    $ rosbag play log.bag

</div>
```

example

This is how you can use `rosbag`:

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

2) Check

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

Check that you didn't forget anything.

</div>
```

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

3) Requirements

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

List of requirements at the beginning of setup chapter.

</div>
```

KNOWLEDGE AND ACTIVITY GRAPH

List of requirements at the beginning of setup chapter.

3.3. Notes and questions

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

These are the one-paragraph forms:

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

+ comment
this is a comment on one paragraph.

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

+ question
this is a question on one paragraph.

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

+ doubt
I have my doubts on one paragraph.

These are the multiple-paragraphs forms:

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

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

+ comment

A comment...

A second paragraph...

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

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

+ question

A question...

A second paragraph...

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

Should it not be:

    $ alternative command

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

+ doubt

A question...

Should it not be:

```
$ alternative command
```

A second paragraph...

Unit B-4
Using LaTeX constructs in documentation

Knowledge and activity graph

Requires: Working knowledge of LaTeX.

4.1. Embedded LaTeX

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

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

or refer to Proposition 1 - Proposition example.

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

The above was written as in Listing 4.1.

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

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

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

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

Listing 4.1. Use of LaTeX code.

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

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

Table 4.1. LaTeX environments and label prefixes

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

Examples of all environments follow.

example

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

Definition 1. Lorem

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

Proposition 2. Lorem

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

Remark 1. Lorem

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

Problem 1. Lorem

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

Example 1. Lorem

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

Theorem 1. Lorem

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

Lemma 1. Lorem

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

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

4.2. LaTeX equations

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

$$2a = a + a \tag{1}$$

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

$$\begin{aligned} a &= b && (2) \\ &= c && (3) \end{aligned}$$

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

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

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

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

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

4.3. LaTeX symbols

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

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

4.4. Bibliography support

You need to have installed `bibtex2html`.

The system supports Bibtex files.

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

Then you can refer to them using the syntax:

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

For example:

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

Will result in:

> Please see [3].

4.5. Embedding Latex in Figures through SVG

KNOWLEDGE AND ACTIVITY GRAPH

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

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

You can run:

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

And the SVG file will be compiled into a PDF figure with the LaTeX commands properly interpreted.

You can then include the PDF file in a normal way (Section 2.5 - Figures) using `filename.pdf` as the filename in the `<img>` tag.

(a) Image saved as svg

(b) Image as PDF after processing

Figure 4.1. Embedding LaTeX in images

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

UNIT B-5

Advanced Markduck guide

5.1. Embedding videos

It is possible to embed Vimeo videos in the documentation.

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

This is the syntax:

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

example

For example, this code:

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

produces this result:

Figure 5.1. Cool Duckietown by night

Depending on the output media, the result will change:

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

5.2. `move-here` tag

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

This is used for autogenerated documentation.

Syntax:

```
# Node `node`

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

5.3. Comments

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

```
# My section

<!-- this text is ignored -->

Let's start by...
```

5.4. Referring to Github files

You can refer to files in the repository by using:

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

The available keys are:

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

For example, you can refer to [the file `pkg_name/src/subscriber_node.py`]() 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
```

5.5. Putting code from the repository in line

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

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

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

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

and the result is the following automatically generated listing:

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

# Imports message type
from std_msgs.msg import String

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

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

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

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

Listing 5.2. subscriber_node.py

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

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

creates the following automatically generated listing:

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

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

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

Listing 5.3. subscriber_node.py

Unit B-6
*Compiling the PDF version

This part describes how to compile the PDF version.

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

6.1. Installing `nodejs`

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

Run:

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

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

```
$ sudo apt remove nodejs
```

Install `nodejs` using the instructions at this page.

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

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

1) Troubleshooting `nodejs` installation problems

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

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

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

6.2. Installing Prince

Install PrinceXML from this page.

6.3. Installing fonts

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

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

and then rebuild the font cache using:

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

6.4. Compiling the PDF

To compile the PDF, use:

```
$ make compile-pdf
```

This creates the file:

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

UNIT B-7

Markduck troubleshooting

7.1. Changes don't appear on the website

For these issues, see [Unit B-8 - The Duckuments bot](#).

7.2. Troubleshooting errors in the compilation process

> **Symptom:** "Invalid XML"

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

> **Symptom:** "Tabs are evil"

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

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

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

7.3. Common mistakes with Markdown

Here are some common mistakes encountered.

1) Not properly starting a list

There must be an empty line before the list starts.

This is correct:

```
I want to learn:

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

This is incorrect:

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

and it will be rendered as follows:

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

UNIT B-8

The Duckuments bot

Note: This is an advanced section mainly for Liam.

8.1. Documentation deployment

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

8.2. Understand what's going on

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

Every minute, it tries to do the following:

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

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

8.3. Logging

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

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

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

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

This shows that the compilation took 5 minutes.

Every two hours you will see something like this:

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

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

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

8.4. Debugging Github Pages problems

Sometimes, it's Github Pages that lags behind.

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

This can identify if the problem is Github.

UNIT B-9

Documentation style guide

This chapter describes the conventions for writing the technical documentation.

9.1. General guidelines for technical writing

The following holds for all technical writing.

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

9.2. Style guide for the Duckietown documentation

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

✖ behaviour

✔ behavior

✖ localisation

✔ localization

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

  - ✖ "Ctrl-C from the command line".
  - ✔ "Use Ctrl-C from the command line".
- Subtle humor and puns about duckies are encouraged.

9.3. Writing command lines

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

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

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

It will become:

```
$ cd ~/duckietown
```

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

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

It will become:

```
$ cd ~/duckietown
```

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

```
$ cd ~/duckietown
```

9.4. Frequently misspelled words

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

9.5. Other conventions

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

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

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

is too much detail.

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

9.6. Troubleshooting sections

Write the documentation as if every step succeeds.

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

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

The following is an example of a troubleshooting section.

1) Troubleshooting

Symptom: This strange thing happens.

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

Symptom: This other strange thing happens.

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

Unit B-10
Learning in Duckietown

Assigned to: Jacopo

Note: This chapter is a draft.

- ✖ We do not refer to teachers and students
- ✔ We refer to learners. Instructors are learners ahead in the learning curve in comparison to other learners
- ✖ We don't examinate
- ✔ We assess competences

10.1. The learning feedback loop

A relevant contribution of modern educational theory is recognizing the importance of feedback in the learning process ().

We love feedback, as it stands at the foundation of control systems theory.

Figure 10.1. The learning loop.

Here, we provide definitions of the key elements in a learning feedback loop, and analogies to their control systems counterparts.

1) Intended Learning Outcomes

Definition 2. (Intended Learning Outcome)
An intended learning outcome is a desired, *measurable*, output of the learning process.

Intended learning outcomes are:

- the starting point in the construction of a learning activity ([1]),
- more effective when formulated with active verbs,
- the equivalent of reference trajectories, or setpoints, in control systems. They represent the ideal output of the controlled system.

example

- ✖ Students will understand robotics
- ✔ Students each build a Duckiebot, implement software and produce demos

2) Learning Activities

Definition 3. (Learning Activities)
Methods chosen by the instructor to ease the learners achievement of the intended learning outcomes.

Active learning practices ([1]) have been shown to improve learning.

example

- ✖ Instructor explains at the blackboard
- ✔ Learners work in groups to achieve objectives

Learning activities are analogous to the the output of the controller (instructor), or input to the system (learners). They have to meet the requirements imposed by external constraints, not shown in

3) Assessment

Definition 4. (Assessment)
An assessment is a procedure to quantify the level of fulfillment of learning outcomes.

An assessment is analogous to a sensor in a control system loop. It measures the system's output.

example

Examples of assessments include: quizzes, colloquia, produced documentation, homework, etc.

10.2. Knowledge, Skills and Competences

Here, we provide definitions for knowledge, skills and competences, in addition to describing their relationships (Figure 10.2).

Figure 10.2. The relationship between Knowledge, Skills and Competences.

1) Knowledge

Definition 5. (Knowledge)
Theoretical facts and information aimed at enabling understanding, and generating or improving *skills*.

example Bayesian inference is handy piece of knowledge when doing robotics.

2) Practice

Definition 6. (Practice)
Practical procedures aimed at generating or improving *skills*, either directly or indirectly by improving knowledge.

example Exercises and proofs can be used to practice different skills.

3) Skills

Definition 7. (Skills)
A proficiency, facility or dexterity that enables carrying out a function. Skills stem from *knowledge*, *practice* and/or aptitude. Skills can be clustered in cognitive, technical and interpersonal, respectively relating to ideas, things and people.

example

Analyzing tradeoffs between performances and constraints is a critical cognitive skill for robotics.

Python language is a useful technical skill in robotics.

Public speaking is a valuable interpersonal skill useful beyond robotics.

In Duckietown we formalize didactic indivisible units, or *atoms*, aimed at improving skills through knowledge and practice. Knowledge atoms are listed in XXX. We define as practice atoms:

TODO: add general reference to all learning atmos, folder atoms_30_learning_material

Definition 8. (Exercise)
An exercise is a practice atom aimed at improving technical skills. Exercises are listed in XXX.

Exercises are targeted to different "things" to which technical skills are related. They may be mathematical exercises aimed at practicing a method, or they may be coding exercises aimed at practicing resolutions of hardware implementation challenges.

Definition 9. (Proof)
A proof is a practice atom aimed at improving cognitive skills.

example Deriving the Kalman filter equations helps practice the *idea* that there is no better approach to state estimation for linear time invariant systems, with "well behaved" measurement and process noises.

4) Competences

Definition 10. (Competences)
Set of skills and/or knowledge that leads to superior performance in carrying out a function. Competences must be *measurable*.

Competences are desirable intended learning outcomes, and typically address the *how* of the learning process.

example

Programming is a competence. It requires a skill, e.g., Python, and knowledge, e.g., Bayesian inference, to know what to code. Practice can help improve knowledge or hone skills.

Unit B-11
Knowledge graph

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

11.1. Formalization

1) Atoms

Definition 11. (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 12. (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 Subsec-

tion 11.1.2 - Semantic graph of atoms.

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.

11.2. 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 human-readable **title**.
- A human-readable **summary** (1 short paragraph).

1) Status values (updated Sep 12)

Each document has a **status** value.

The allowed values are described in Table 11.1.

TABLE 11.1. STATUS CODES

`draft`	We just started working on it, and it is not ready for public consumption.
`beta`	Early reviewers should look at it now.
`ready`	The document is ready for everybody.
`recently-updated`	The document has been recently updated (less than 1 week)
`to-update`	A new pass is needed on this document, because it is not up to date anymore.
`deprecated`	The document is ready for everybody.

11.3. 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 status is given by an attribute `status`, which should be value of the values in Table 11.1.
3. (Optional) The language is given by an attribute `lang` (`{lang=en-US}`).
4. (Optional) The type is given by an attribute `type` (`{type=demo}`).

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 11.4. `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 11.5. `calibration.it.md`

11.4. How to describe the semantic graphs of atoms

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

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

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

This is the first setup step.
```

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

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

This is the second setup step.

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

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

TABLE 11.2. SEMANTIC LINKS

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

11.5. How to describe modules

TODO: Define a micro-format for this.

UNIT B-12
Translations

Note: This part is not implemented yet.

12.1. 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.

12.2. 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.

12.3. 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 C
Operation manual - Duckiebot

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

UNIT C-1

Duckiebot configurations

We define different Duckiebot configurations depending on their time of use and hardware components. This is a good starting point if you are wondering what parts you should obtain to get started. Once you have decided which configuration best suits your needs, you can proceed to the detailed descriptions for DB17-wjd or DB17-wjdlc Duckiebot.

1.1. Configuration list

The configurations are defined with a root: DB17-, indicating the "bare bones" Duckiebot used in the Fall 2017 synchronized course, and an appendix y which can be the union (in any order) of any or all of the elements of the optional hardware set $\mathcal{O} = \{$w, j, d, l, c$\}$. A DB17 Duckiebot can navigate autonomously in a Duckietown, but cannot communicate with other Duckiebots.

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

- w: 5 GHz wireless adapter to facilitate streaming of images;
- j: wireless joypad that facilitates manual remote control;
- d: USB drive for additional storage space;
- l: includes LEDs, bumpers and the necessary bits to set the LEDs in place. This hardware enables Duckiebot communications and fleet level behaviors. This is a major hardware upgrade;
- c: a different castor wheel to replace the preexisting one, providing an overall smoother drive.

1.2. Configuration functionality

1) DB17

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 behaviors.

2) DB17-w

In this configuration, the minimal DB17 version is augmented with a 5 GHz wireless adapter, which drastically improves connectivity enabling, e.g., streaming of images. This feature is particularly useful in connection saturated environments, e.g., classrooms.

3) DB17-j

In this configuration, the minimal DB17 version is augmented with a 2.4 GHz wireless joypad, used for manual remote control of the Duckiebot. It is particularly useful for getting the Duckiebot our of tight spots or letting younger ones have a drive, in addition

to providing handy shortcuts to different functions in development phase.

4) DB17-d

In this configuration, the minimal `DB17` 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.

5) DB17-wjd

This is the Duckiebot configuration that is first handed out at ETHZ during the Fall 2017 course edition. An upgrade will be provided during the course.

6) DB17-l

In this configuration the Duckiebot in equipped with 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 behaviors (e.g., negotiating crossing an intersection).

7) DB17-c

In this configuration, the standard dotation castor wheel (the omnidirectional wheel on the back of the Duckiebot) is replace with a different omnidirectional wheel. The castor wheel upgrade provides a smoother ride. This configuration includes the mechanical bits necessary to mount the new castor wheel at the right height.

UNIT C-2

Acquiring the parts for the Duckiebot DB17-wjd

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

In general, keep in mind that:

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

KNOWLEDGE AND ACTIVITY GRAPH

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

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

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

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

2.1. Bill of materials

TABLE 2.1. BILL OF MATERIALS

Chassis	USD 20
Camera with 160-FOV Fisheye Lens	USD 22
Camera Mount	USD 8.50
300mm Camera Cable	USD 2
Raspberry Pi 3 - Model B	USD 35
Heat Sinks	USD 5
Power supply for Raspberry Pi	USD 7.50
16 GB Class 10 MicroSD Card	USD 10
Mirco SD card reader	USD 6
DC Motor HAT	USD 22.50
2 Stacking Headers	USD 2.50/piece
Battery	USD 20
16 Nylon Standoffs (M2.5 12mm F 6mm M)	USD 0.05/piece
4 Nylon Hex Nuts (M2.5)	USD 0.02/piece
4 Nylon Screws (M2.5x10)	USD 0.05/piece
2 Zip Ties (300x5mm)	USD 9
Wireless Adapter (5 GHz) (`DB17-w`)	USD 20
Joypad (`DB17-j`)	USD 10.50
Tiny 32GB USB Flash Drive (`DB17-d`)	USD 12.50
Total for `DB17` configuration	USD 173.6
Total for `DB17-w` configuration	USD 193.6
Total for `DB17-j` configuration	USD 184.1
Total for `DB17-d` configuration	USD 186.1
Total for `DB17-wjd` configuration	USD 216.6

2.2. Chassis

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

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

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

Figure 2.1. The Magician Chassis

2.3. Raspberry Pi 3 - Model B

The Raspberry Pi is the central computer of the Duckiebot. Duckiebots use Model B (Figure 2.2) (A1.2GHz 64-bit quad-core ARMv8 CPU, 1GB RAM), a small but powerful computer.

Figure 2.2. The Raspberry Pi 3 Model B

1) Power Supply

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

Figure 2.3. The Power Supply

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

2) Heat Sinks

The Raspberry Pi will heat up significantly during use. It is warmly recommended to add heat sinks, as in Figure 2.4. Since we will be stacking HATs on top of the Raspberry Pi with 15 mm standoffs, the maximum height of the heat sinks should be well below 15 mm. The chip dimensions are 15x15mm and 10x10mm.

Figure 2.4. The Heat Sinks

3) Class 10 MicroSD Card

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

Figure 2.5. The MicroSD card

4) Mirco SD card reader

A microSD card reader (Figure 2.6) is useful to copy the system image to a Duckiebot

from a computer to the Raspberry Pi microSD card, when the computer does not have a native SD card slot.

Figure 2.6. The Mirco SD card reader

2.4. Camera

The Camera is the main sensor of the Duckiebot. All versions equip a 5 Mega Pixels 1080p camera with wide field of view (**160°**) fisheye lens (Figure 2.7).

Figure 2.7. The Camera with Fisheye Lens

1) Camera Mount

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

Figure 2.8. The Camera Mount

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

Figure 2.9. The Camera on its mount

2) 300mm Camera Cable

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

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

2.5. DC Motor HAT

We use the DC Stepper motor HAT (Figure 2.11) to control the DC motors that drive

the wheels. This item will require soldering to be functional. This HAT has dedicate PWM and H-bridge for driving the motors.

Figure 2.11. The Stepper Motor HAT

1) Stacking Headers

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

Figure 2.12. The Stacking Headers

2.6. Battery

The battery (Figure 2.13) provides power to the Duckiebot.

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

Figure 2.13. The Battery

2.7. Standoffs, Nuts and Screws

We use non electrically conductive standoffs (M2.5 12mm F 6mm M), nuts (M2.5), and

screws (M2.5x10mm) to hold the Raspberry Pi to the chassis and the HATs stacked on top of the Raspberry Pi.

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

Figure 2.14. Standoffs, Nuts and Screws

2.8. Zip Tie

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

Figure 2.15. The zip ties

2.9. Configuration DB17-w

1) Wireless Adapter (5 GHz)

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

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

2.10. Configuration DB17-j

1) Joypad

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

The model linked in the table (Figure 2.17) does not include batteries.

Figure 2.17. A Wireless Joypad

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

Figure 2.18. A Wireless Joypad

2.11. Configuration DB17-d

1) Tiny 32GB USB Flash Drive

In configuration `DB17-d`, the Duckiebot is equipped with an "external" hard drive (Figure 2.19). This add-on is very convenient to store logs during experiments and later port them to a workstation for analysis. It provides storage capacity and faster data transfer than the MicroSD card.

Figure 2.19. The Tiny 32GB USB Flash Drive

UNIT C-3
Soldering boards for DB17

Assigned to: Shiying

KNOWLEDGE AND ACTIVITY GRAPH

Requires: Parts: Duckiebot `DB17` parts. The acquisition process is explained in Unit C-2 - Acquiring the parts for the Duckiebot `DB17-wjd`. The configurations are described in Unit C-1 - Duckiebot configurations. In particular:

- GPIO Stacking Header
- DC and Stepper Motor HAT for Raspberry Pi

Requires: Tools: Solderer

Requires: Experience: some novice-level experience with soldering.

Requires: Time: 30 minutes

Results: Soldered DC Motor HAT

Note: It is better to be safe than sorry. Soldering is a potentially hazardous activity. There is a fire hazard as well as the risk of inhaling toxic fumes. Stop a second and make sure you are addressing the safety standards for soldering when following these instructions. If you have never soldered before, seek advice.

3.1. General tips

Note: There is a general rule in soldering: solder the components according to their height, from lowest to highest.

In this instruction set we will assume you have soldered something before and are acquainted with the soldering fundamentals. If not, before proceeding, read this great tutorial on soldering:

➡ Alternative instructions: how to solder on Headers and Terminal Block

1) Preparing the components

Take the GPIO stacking header Figure 3.1 out of Duckiebox and sort the following components from DC motor HAT package:

- Adafruit DC/Stepper Motor HAT for Raspberry Pi
- 2-pin terminal block (2x), 3-pin terminal block (1x)

Figure 3.1. GPIO_Stacking_Header

Figure 3.2. DC/Stepper Motor HAT and solder components

2) Soldering instructions

- [1.] Make a 5 pin terminal block by sliding the included 2 pin and 3 pin terminal blocks into each other [](#fig:terminal_block).

Figure 3.3. 5 pin terminal_block

- [2.] Slide this 5 pin block through the holes just under "M1 GND M2" on the board. Solder it on (we only use two motors and do not need connect anything at the "M3 GND M4" location) ([](#figure:upview_Stepper_Motor));
- [3.] Slide a 2 pin terminal block into the corner for power. Solder it on. ([](#figure:sideview_terminal));
- [4.] Slide in the GPIO Stacking Header onto the 2x20 grid of holes on the edge opposite the terminal blocks and with vice versa direction ([](#figure:GPIO_HAT_orientation)). Solder it on. Note: stick the GPIO Stacking Header from bottom to top, different orientation than terminal blocks (from top to bottom).

Figure 3.4.

Figure 3.5. Side view of finished soldering DC/Stepper Motor HAT

Figure 3.6. upside view of finished soldering DC/Stepper Motor HAT

UNIT C-4

Preparing the power cable for DB17

In configuration `DB17` we will need a cable to power the DC motor HAT from the battery. The keen observer might have noticed that such a cable was not included in the `DB17` Duckiebot parts chapter. Here, we create this cable by splitting open any USB-A cable, identifying and stripping the power wires, and using them to power the DC motor HAT. If you are unsure about the definitions of the different Duckiebot configurations, read Unit C-1 - Duckiebot configurations.

It is important to note that these instructions are relevant only for assembling a `DB17-wjdc` configuration Duckiebot (or any subset of it). If you intend to build a `DB17-1` configuration Duckiebot, you can skip these instructions.

KNOWLEDGE AND ACTIVITY GRAPH

Requires: One male USB-A to anything cable.

Requires: A pair of scissors.

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

Requires: Time: 5 minutes

Results: One male USB-A to wires power cable

4.1. Step 1: Find a cable

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

If you have purchased the suggested components listed in Unit C-2 - Acquiring the parts for the Duckiebot `DB17-wjd`, you can use the longer USB cable contained inside the battery package (Figure 4.1), which will be used as an example in these instructions.

Figure 4.1. The two USB cables in the suggested battery pack.

Put the shorter cable back in the box, and open the longer cable (Figure 4.2)

Figure 4.2. Take the longer cable, and put the shorter on back in the box.

4.2. 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 4.3) at the desired length from the USB-A port.

Figure 4.3. Cut the USB cable using the scissors.

The cut will look like in Figure 4.4.

Figure 4.4. A cut USB cable.

4.3. 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 4.5.

Figure 4.5. Stripping the external layer of the USB cable.

After removing the external plastic, you will see four wires: black, green, white and red (Figure 4.6).

Figure 4.6. Under the hood of a USB-A cable.

Once the bottom part of the external cable is removed, you will have isolated the four wires (Figure 4.7).

Figure 4.7. The four wires inside a USB-A cable.

4.4. 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 4.8).

Figure 4.8. Strip the power wires and cut the data wires.

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

4.5. 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 4.9).

Figure 4.9. Finding which two wires are for power.

4.6. 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 Unit C-3 - Soldering boards for DB17.

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 4.10).

Figure 4.10. Connnect the power wires to the DC motor HAT

+ comment

We need to test this on a working Duckiebot. -JT If the green LED turns on, the DC motor HAT is receving power.

Unit C-5
Assembling the Duckiebot DB17

Point of contact: Shiying Li

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

Knowledge and activity graph

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

Requires: Having soldered the `DB17-wjd` parts. The soldering process is explained in Unit C-3 - Soldering boards for `DB17`.

Requires: Having prepared the power cable. The power cable preparation is explained in Unit C-4 - Preparing the power cable for `DB17`. Note: Not necessary if you are proceeding directly to configuration C1.

Requires: Having installed the image on the MicroSD card. The instructions on how to reproduce the Duckiebot system image are in Unit C-6 - Reproducing the image.

Requires: Time: about 40 minutes.

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

+ comment

Notes - if we have the bumpers, at what point should we add them? I think that the battery could actually be attached before the levels of the chassis are joined. I found it hard to mount the camera (the holes weren't lining up). the long camera cable is a bit annoying - I folded it and shoved it in between two hats. We should decide if PWM hat is part of this configuration, why not leave it for now and forget about the spliced cable for the class. I found that the screwdriver that comes with the chassis kit is too fat to screw in the wires on the hat. The picture of where to put the zip tie for the battery is not very clear. need something to cut the end of the zip tie with.

+ comment

In general I would recommend having diagonal pliers as well as a few mini screwdrivers at hand. Both can be obtained from a local dollar store for about 6$ total. The pliers / cutters are required either for making your own power cord or for cutting the zip ties after they've been attached to the chassis (because they are too long). The screwdrivers are required for tightening the screws on the hats after the cables have been plugged in because the chassis screwdriver is too wide for that.

5.1. Chassis

Open the Magician chassis package Figure 5.1 and take out the following components:

- Chassis-bottom (1x), Chassis-up (1x)
- DC Motors (2x), motor holders (4x)

- Wheels (2x), steel omnidirectional wheel (1x)
- All spacers and screws
- Screwdriver

Figure 5.1. Components in Duckiebot package.

> **Note:** You won't need the battery holder and speed board holder (on the right side in Figure 5.1).

1) Bottom

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

Figure 5.2. Components for mounting the motor

Figure 5.3.

Figure 5.4.

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

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

2) Wheels

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

Figure 5.5. The scratch of wheels

Figure 5.6. Assembled wheels

3) Omnidirectional wheel

The Duckiebot is driven by controlling the wheels attached to the DC motors. Still, it requires a “passive” omnidirectional wheel on the back.

If you have purchased the optional caster wheel, read on to the next section.

The Magician chassis package contains a steel omnidirectional wheel, and the related standoffs and screws to secure it to the chassis-bottom part.

Figure 5.7. The scratch of omni wheel

Figure 5.8. Assembled omni wheel

Caster wheel:

As alternative to omnidirection wheel, caster wheel has less friction.

If you have purchased caster wheel, read this section.

To assemble the caster wheel, the following materials is needed:

- caster wheel
- 4 standoffs (M3x12mm F-F, 6mm diameter)
- 8 metal screws (M3x8mm)
- 8 split lock washers

TODO: add instructions for Caster wheel assembly.

4) Mounting the spacers

Put the car upright (omni wheel pointing towards the table) and arrange wires so that they go through the center rectangle. Put 4 spacers with 4 of M3x6 screws on each corner as below.

Figure 5.9.

Figure 5.10.

The bottom part of the Duckiebot's chassis is now ready. The next step is to assemble the Raspberry Pi on chassis-top part.

5.2. Assembling the Raspberry Pi, camera, and HATs

1) Raspberry Pi

Before attaching anything to the Raspberry Pi you should add the heat sinks to it. There are 2 small sinks and a big one. The big one best fits onto the processor (the big "Broadcom"-labeled chip in the center of the top of the Raspberry Pi). One of the small ones can be attached to the small chip that is right next to the Broadcom chip. The third heat sink is optional and can be attached to the chip on the underside of the Raspberry Pi. Note that the chip on the underside is bigger than the heat sink. Just mount the heat sink in the center and make sure all of them are attached tightly.

When this is done fasten the nylon standoffs on the Raspberry Pi, and secure them on the top of the chassis-up part by tightening the nuts on the opposite side of the chassis-up.

Figure 5.11. Components for Raspberry Pi3

Figure 5.12. Heat sink on Raspberry Pi3

Figure 5.13. Standoffs for Raspberry Pi3

Figure 5.14. Attach the nylon huts for the standoffs (Bottom view)

Figure 5.15. Top view of assembled Raspberry Pi3

Micro SD card:

Requires: Having the Duckiebot image copied in the micro SD card.

Take the micro SD card from the duckiebox and insert its slot on the Raspberry Pi. The SD card slot is just under the display port, on the short side of the PI, on the flip-side of where the header pins are.

Figure 5.16. The Micro SD card and SD card readers.

Figure 5.17. Display port and inserted SD card

2) Camera

The Raspberry Pi end:

First, identify the camera cable port on the Pi (between HDMI and power ports) and remove the orange plastic protection (it will be there if the Pi is new) from it. Then, grab the long camera cable (300 mm) and insert in the camera port. To do so, you will need to gently pull up on the black connector (it will slide up) to allow the cable to insert the port. Slide the connector back down to lock the cable in place, making sure it “clicks”.

Protip: Make sure the camera cable is inserted in the right direction! The metal pins of the cable should be in contact with the metal terminals in the camera port of the PI.

Figure 5.18.

Note: Insert the cable in the right direction to connect the camera to the Raspberry Pi.

Figure 5.19. Camera with long cable

The camera end:

If you have purchased the long camera cable, the first thing to do is removing the shorter cable that comes with the camera package. Make sure to slide up the black connectors of the camera-camera cable port in order to unblock the cable.

Take the rear part of the camera mount and use it hold the camera in place. Note that the camera is just press-fitted to the camera mount, no screws/nuts are needed.

In case you have not purchased the long camera cable, do not worry! It is still very possible to get a working configuration, but you will have little wiggling space and assembly will be a little harder.

Place the camera on the mount and fasten the camera mount on the chassis-up using

M3x10 flathead screws and M3 nuts from the Duckiebox.

Protip: make sure that the camera mount is: (a) geometrically centered on the chassis-up; (b) fastened as forward as it can go; (c) it is tightly fastened. We aim at having a standardized position for the camera and to minimize the wiggling during movement.

Figure 5.20.

> **Note:** make sure that the cable is oriented in this direction (writing towards the CPU). Otherwise you will have to disassemble the whole thing later.

3) Assemble chassis-bottom and chassis-up

Arrange the motor wires through the chassis-up, which will be connected to Stepper Motor HAT later.

Figure 5.21.

4) Extending the intra-decks standoffs

In order to fit the battery, we will need to extend the Magician Chassis standoffs with the provided nylon standoff spacers. Grab 4 of them, and secure them to one end of the long metal standoffs provided in the Magician Chassis package.

Secure the extended standoff to the 4 corners of the chassis-bottom. The nylon standoffs should smoothly screw in the metal ones. If you feel resistance, don't force it or the nylon screw might break in the metal standoff. In that case, unscrew the nylon spacer and try again.

Figure 5.22. 4 nylon M3x5 Standoff Spacer and 4 M3x10 screws

Figure 5.23.

5) Put a Stepper Motor HAT with 4 standoffs on the top of Raspberry Pi

The GPIO Stacking Header need to be correctly stacked into the pins before fastening the standoffs. It connects the Pi with the HAT. Place the camera cable properly when you mount the HAT on the Raspberry Pi.

Figure 5.24.

Figure 5.25.

6) Connect the motor's wires to the terminal

We are using M1 and M2. The left (in robot frame) motor is connected to M1 and the right motor is connected to M2. If you have followed Part A correctly, the wiring order will look like as following pictures:

- Left Motor: Red
- Left Motor: Black
- Right Motor: Black
- Right Motor:Red

Figure 5.26.

7) Battery

Put the battery between the upper and lower decks of the chassis. It is strongly recommended to secure the battery from moving using zip ties.

Figure 5.27.

8) Joypad

With each joypad (Figure 5.28) comes a joypad dongle (Figure 5.29). Don't lose it!

Figure 5.28. All components in the Joypad package

Insert the joypad dongle into one of the USB port of the Raspberry Pi.

Figure 5.29

Insert 2 AA batteries on the back side of the joypad Figure 5.30.

Figure 5.30. Joypad and 2 AA batteries

9) Connect the power cables

You are now ready to secure the prepared power wires in Unit C-4 - Preparing the power cable for DB17. to the DC motor HAT power pins.

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 4.10).

Fix all the cables on the Duckiebot so that it can run on the way without barrier.

Figure 5.31. Insert the prepared power wire to DC motor HAT power pins.

Unit C-6
Reproducing the image

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.

Knowledge and activity graph

Requires: Internet connection to download the packages.

Requires: A PC running any Linux with an SD card reader.

Requires: Time: about 4 hours (most of it spent waiting for things to download/compile).

Results: A baseline Ubuntu Mate 16.04.2 image with updated software.

6.1. 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
```

6.2. Burn the image to an SD card

Next, burn the image on to the SD card.

➡ This procedure is explained in Section 16.2 - How to burn an image to an SD card.

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).

6.3. 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
choose “2. Boot Options”, configure option for startup. –>B1. Console Text console

6.4. 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 raspberrypi-kernel-headers
```

Python:

```
$ sudo apt install -y python-dev ipython python-sklearn python-smbus
$ sudo apt install -y python-termcolor
$ sudo apt install python-frozendict
$ sudo apt install python-tables
$ pip install comptests
$ pip install procgraph
$ sudo pip install scipy --upgrade
$ sudo pip install ruamel.yaml --upgrade
```

Note: scipy –upgrade(0.19.1) took about an hour with ethernet connection.

I2C:

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

6.5. 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
```

6.6. Install ROS

Install ROS.

➡ The procedure is given in Section 29.1 - Install ROS.

6.7. Wireless configuration (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
}
```

6.8. Wireless configuration

The files that describe the network configuration are in the directory

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

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

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

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

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

[ipv4]
dns-search=
method=auto

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

This is the file

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

Contents:

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

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

[ipv4]
dns-search=
method=shared

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

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

6.9. SSH server config

This enables the SSH server:

```
$ sudo systemctl enable ssh
```

6.10. 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
```

6.11. 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
```

6.12. Clean up

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

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

Either:

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

Stuff to remove:

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

At the end, remove extra dependencies:

```
$ sudo apt autoremove
```

And remove the `apt` cache using:

```
$ sudo apt clean
```

The total size should be around 6.6GB.

6.13. 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
$ sudo adduser ubuntu video
```

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

```
$ sudo udevadm trigger
```

2) Basic SSH config

Do the basic SSH config.

➡ The procedure is documented in Section 18.3 - Local configuration.

> **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
```

6.14. 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.)

6.15. 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 16.2 - How to burn an image to an SD card; 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 16.3 -

How to shrink an image.

6.16. TODO: Git LFS

Note: We should install Git LFS on the Raspberry Pi, but so far AC did not have any luck. See Section 27.1 - Generic installation instructions.

UNIT C-7

Installing Ubuntu on laptops

Assigned to: Andrea

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

KNOWLEDGE AND ACTIVITY GRAPH

Requires: A laptop with free disk space.

Requires: Internet connection to download the Ubuntu image.

Requires: About 30 minutes.

Results: A laptop ready to be used for Duckietown.

7.1. Install Ubuntu

Install Ubuntu 16.04.3.

➡ For instructions, see for example this online tutorial.

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

7.2. 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
```

7.3. Install ROS

Install ROS on your laptop.

➡ The procedure is given in Section 29.1 - Install ROS.

7.4. Other suggested software

1) Redshift

This is Flux for Linux. It is an accessibility/lab safety issue: bright screens damage eyes and perturb sleep [6].

Install redshift and run it.

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

Set to "autostart" from the icon.

7.5. Installation of the duckuments system

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

➡ The procedure is documented in Section 1.3 - Installing the documentation system.

7.6. Passwordless sudo

Set up passwordless `sudo`.

➡ This procedure is described in Section 6.11 - Passwordless sudo.

+ comment

Huh I don't know - this is great for usability, but horrible for security. If you step away from your laptop for a second and don't lock the screen, a nasty person could `sudo rm -rf /`. -FG

7.7. SSH and Git setup

1) Basic SSH config

Do the basic SSH config.

➡ The procedure is documented in Section 18.3 - Local configuration.

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 18.5 - Creating an SSH key-pair.

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

Add the public key to your Github account.

➡ The procedure is documented in Section 28.3 - Add a public key to Github.

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 26.3 - Setting up global configurations for Git.

Unit C-8
Duckiebot Initialization

Assigned to: Andrea

Knowledge and activity graph

Requires: An SD card of dimensions at least 16 GB.

Requires: A computer with an internet connection, an SD card reader, and 16 GB of free space.

Requires: An assembled Duckiebot in configuration `DB17`. This is the result of Unit C-5 - Assembling the Duckiebot `DB17`.

Results: A Duckiebot that is configured correctly, that you can connect to with your laptop and hopefully also has internet access

8.1. Acquire and burn the image

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

> https://www.dropbox.com/s/ckpqpp0cav3aucb/duckiebot-RPI3-AD-2017-09-12.img.xz?dl=1

The size is 1.7 GB.

You can use:

```
$ wget -O duckiebot-RPI3-AD-2017-09-12.img.xz URL above
```

+ comment

The original was:

```
$ curl -o duckiebot-RPI3-AD-2017-09-12.img.xz URL above
```

It looks like that `curl` cannot be used with Drobpox links because it does not follow redirects.

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

```
$ sha256sum duckiebot-RPI3-AD-2017-09-12.img.xz
7136f9049b230de68e8b2d6df29ece844a3f830cc96014aaa92c6d3f247b6130
duckiebot-RPI3-AD-2017-09-12.img.xz
```

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

Uncompress the file:

```
$ xz -d -k duckiebot-RPI3-AD-2017-09-12.img.xz
```

This will create a file of 11 GB in size.

Next, burn the image on disk.

➡ The procedure of how to burn an image is explained in Section 16.2 - How to burn an image to an SD card.

8.2. Turn on the Duckiebot

Put the SD Card in the Duckiebot.

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

TODO: Add figure

+ comment

In general, for the battery: if it's off, a single click on the power button will turn the battery on. When it's on, a single click will show you the charge indicator (4 white lights = full), and holding the button for 3s will turn off the battery. Shutting down the Duckiebot is not recommended because it may cause corruption of the SD card.

8.3. 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.

8.4. 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
...
```

8.5. 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 Unit I-25 - Byobu for an introduction to Byobu.

+ doubt

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

8.6. Setup network

➡ Unit C-9 - Networking aka the hardest part

8.7. Update the system

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

Use these commands

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

8.8. Give a name to the Duckiebot

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

These are the criteria:

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

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

8.9. Change the hostname

We will put the robot name in configuration files.

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

```
$ sudo vi filename
```

Edit the file

```
/etc/hostname
```

and put "`robot name`" instead of `duckiebot-not-configured`.
Also edit the file

```
/etc/hosts
```

and put "`robot name`" where `duckiebot-not-configured` appears.
The first two lines of `/etc/hosts` should be:

```
127.0.0.1   localhost
127.0.1.1   robot name
```

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

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

Then reboot the Raspberry Pi using the command

```
$ sudo reboot
```

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

```
$ hostname
robot name
```

8.10. Expand your filesystem

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

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

and then reboot

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

once rebooted you can test whether this was successful by doing

```
$ df -lh
```

the output should give you something like:

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/root        15G  6.3G  8.2G  44% /
devtmpfs        303M     0  303M   0% /dev
tmpfs           431M     0  431M   0% /dev/shm
tmpfs           431M   12M  420M   3% /run
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           431M     0  431M   0% /sys/fs/cgroup
/dev/mmcblk0p1   63M   21M   43M  34% /boot
tmpfs            87M     0   87M   0% /run/user/1000
```

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

8.11. Create your user

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

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

To create a new user:

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

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

```
$ sudo adduser username sudo
```

Make the user a member of the groups `input`, `video`, and `i2c`

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

Set the shell `bash`:

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

To set a password, use:

```
$ sudo passwd username
```

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

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

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

+ comment
What steps?? -LP

1) Basic SSH config

Do the basic SSH config.

➡ The procedure is documented in Section 18.3 - Local configuration.

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 18.5 - Creating an SSH key-pair.

3) Add SSH alias

Once you have your SSH key pair on both your laptop and your Duckiebot, as well as your new user- and hostname set up on your Duckiebot, then you should set up an SSH alias as described in Section 14.2 - SSH aliases. This allows your to log in for example with

```
$ ssh abc
```

instead of

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

where you can chose `abc` to be any alias / shortcut.

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

Add the public key to your Github account.

➡ The procedure is documented in Section 28.3 - Add a public key to Github.

If the step is done correctly, the following command should succeed and give you a welcome message:

```
$ ssh -T git@github.com
Hi username! You've successfully authenticated, but GitHub does not provide shell
access.
```

5) Local Git configuration

➡ This procedure is in Section 26.3 - Setting up global configurations for Git.

6) Set up the laptop-Duckiebot connection

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

➡ The procedure is explained in Section 18.6 - How to login without a password. In this case, we have: `local` = laptop, `local-user` = your local user on the laptop, `remote` = `robot name`, `remote-user` = `username`.

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

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

7) Some advice on the importance of passwordless access

In general, if you find yourself:

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

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

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

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

8.12. Other customizations

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

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

8.13. Hardware check: camera

Check that the camera is connected using this command:

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

If you see `detected=0`, it means that the hardware connection is not working.
You can test the camera right away using a command-line utility called `raspistill`.
Use the `raspistill` command to capture the file `out.jpg`:

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

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

➡ For instructions on how to use `scp`, see Subsection 20.1.1 - Download a file with SCP.

8.14. Final touches: duckie logo

In order to show that your Duckiebot is ready for the task of driving around happy little duckies, the robot has to fly the Duckietown flag. When you are still logged in to the Duckiebot you can download and install the banner like this:
Download the ANSI art file from Github:

```
$ wget --no-check-certificate -O duckie.art "https://raw.githubusercontent.com/
duckietown/Software/master/misc/duckie.art"
```

(optional) If you want, you can preview the logo by just outputting it onto the command line:

```
$ cat duckie.art
```

Next up create a new empty text file in your favorite editor and add the code for showing your duckie pride:
Let's say I use `nano`, I open a new file:

```
$ nano 20-duckie
```

And in there I add the following code (which by itself just prints the duckie logo):

```
#!/bin/sh
printf "\n$(cat /etc/update-motd.d/duckie.art)\n"
```

Then save and close the file. Finally you have to make this file executable...

```
$ chmod +x 20-duckie
```

...and copy both the duckie logo and the script into a specific directory `/etc/update-motd.d` to make it appear when you login via SSH. `motd` stands for "message of the day". This is a mechanism for system administrators to show users news and messages when they login. Every executable script in this directory which has a filename a la `NN-some name` will get exected when a user logs in, where `NN` is a two digit number that indicates the order.

```
sudo cp duckie.art /etc/update-motd.d
sudo cp 20-duckie /etc/update-motd.d
```

Finally log out of SSH via `exit` and log back in to see duckie goodness.

1) Troubleshooting

Symptom: `detected=0`

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

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

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

Symptom: random `wget`, `curl`, `git`, and `apt` calls fail with SSL errors.

Resolution: That's probably actually an issue with your system time. Type the command `timedatectl` into a terminal, hit enter and see if the time is off. If it is, you might want to follow the intructions from this article, or entirely uninstall your NTP service and manually grab the time on reboot. It's a bit dirty, but works surprisingly well.

Symptom: Cannot find `/etc` folder for configuring the Wi-Fi. I only see `Desktop`, `Downloads` when starting up the Duckiebot.

Resolution: If a directory name starts with `/`, it's not supposed to be in the home directory, but rather at the root of the filesystem. You are currently in `/home/ubuntu`. Type `ls /` to see the folders at the root, including `/etc.

Unit C-9

Networking aka the hardest part

Knowledge and activity graph

Requires: A Duckiebot in configuration `DB17-C0+w`

Requires: Either a router that you have control over that has internet access, or your credentials for connecting to an existing wireless network

Requires: Patience (channel your inner Yoda)

Results: A Duckiebot that you can connect to and that is connected to the internet

Note: this page is primarily for folks operating with the "two-network" configuration, `C0+w`. For a one adapter setup you will can skip directly to Section 9.2 - Setting up wireless network configuration, but you will have to connect to a network that you can ssh through.

The basic idea is that we are going to use the "Edimax" thumbdrive adapter to create a dedicated wireless network that you can always connect to with your laptop. Then we are going to use the built-in Broadcom chip on the Pi to connect to the internet, and then the network will be bridged.

9.1. (For `C0+w`) Configure the robot-generated network

This part should work every time with very low uncertainty.

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

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

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

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

...

lo        no wireless extensions.

enxb827eb1f81a4  no wireless extensions.

wlan1     IEEE 802.11bgn  ESSID:"duckietown"

...
```

Make note of the name `wlxAABBCCDDEEFFGG`.

Look up the MAC address using the command:

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

Then, edit the connection file

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

Make the following changes:

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

Reboot.

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

You can connect with the laptop to that network.

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

9.2. Setting up wireless network configuration

You are connected to the Duckiebot via WiFi, but the Duckiebot also needs to connect to the internet in order to get updates and install some software. This part is a little bit more of a "black art" since we cannot predict every possible network configurations. Below are some settings that have been verified to work in different situations:

1) Option 1: `duckietown` WiFi

Check with your phone or laptop if there is a WiFi in reach with the name of `duckietown`. If there is, you are all set. The defaut configuration for the Duckiebot is to have one WiFi adapter connect to this network and the other broadcast the access point which you are currently connected to.

2) Option 2.a): `eduroam` WiFi (Non-UdeM/McGill instructions)

If there should be no `duckietown` network in reach then you have to manually add a network configuration file for the network that you'd like to connect to. Most universities around the world should have to `eduroam` network available. You can use it for connecting your Duckiebot.

Save the following block as new file in `/etc/NetworkManager/system-connections/eduroam`:

```
[connection]
id=eduroam
uuid=38ea363b-2db3-4849-a9a4-c2aa3236ae29
type=wifi
permissions=user:oem:;
secondaries=

[wifi]
mac-address=the MAC address of your internal wifi adapter, wlan0
mac-address-blacklist=
mac-address-randomization=0
mode=infrastructure
seen-bssids=
ssid=eduroam

[wifi-security]
auth-alg=open
group=
key-mgmt=wpa-eap
pairwise=
proto=

[802-1x]
altsubject-matches=
eap=ttls;
identity=your eduroam username@your eduroam domain
password=your eduroam password
phase2-altsubject-matches=
phase2-auth=pap

[ipv4]
dns-search=
method=auto

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

Set the permissions on the new file to 0600.

```
sudo chmod 0600 /etc/NetworkManager/system-connections/eduroam
```

3) Option 2.b): eduroam WiFi (UdeM/McGill instructions)

Save the following block as new file in `/etc/NetworkManager/system-connections/eduroam-USERNAME`: where USERNAME is the your logged-in username in the duckiebot.

```
[connection]
id=eduroam
uuid=38ea363b-2db3-4849-a9a4-c2aa3236ae29
type=wifi
permissions=user:USERNAME:;
secondaries=

[wifi]
mac-address=the MAC address of your internal wifi adapter, wlan0
mac-address-blacklist=
mac-address-randomization=0
mode=infrastructure
seen-bssids=
ssid=eduroam

[wifi-security]
auth-alg=open
group=
key-mgmt=wpa-eap
pairwise=
proto=

[802-1x]
altsubject-matches=
eap=peap;
identity=DGTIC UNIP
password=DGTIC PWD
phase2-altsubject-matches=
phase2-auth=mschapv2

[ipv4]
dns-search=
method=auto

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

Set the permissions on the new file to 0600.

```
sudo chmod 0600 /etc/NetworkManager/system-connections/eduroam-USERNAME
```

4) Option 3 (For Univeristé de Montréal students only): Use `UdeM avec cryptage`

TODO: someone replicate please - LP

> **Note:** your can use the `autoconnect-priority=XX` inside the `[connection]` block to establish a priority. If you want to connect to one network preferentially if two are available then give it a higher priority.

Save the following block as new file in `/etc/NetworkManager/system-connections/secure`:

```
[connection]
id=secure
uuid=e9cef1bd-f6fb-4c5b-93cf-cca837ec35f2
type=wifi
permissions=
secondaries=
timestamp=1502254646
autoconnect-priority=100

[wifi]
mac-address-blacklist=
mac-address-randomization=0
mode=infrastructure
ssid=UdeM avec cryptage
security=wifi-security

[wifi-security]
key-mgmt=wpa-eap

[802-1x]
eap=peap;
identity=DGTIC UNIP
phase2-auth=mschapv2
password=DGTIC PWD

[ipv4]
dns-search=
method=auto

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

Set the permissions on the new file to 0600.

```
sudo chmod 600 /etc/NetworkManager/system-connections/secure
```

5) Option 4: custom WiFi

If neither `duckietown` nor `eduroam` are available, you can add your own configuration file. Here is an example for a standard WPA2-private home network. Save the following block as `TODO: filename custom wifi setting`:

```
TODO code block for custom connection
```

Note: Whichever option you pick, we don't recommend to create or modify the `/etc/wpa_supplicant.conf` file, because this verion of Ubuntu uses the `NetworkManager` ser-

vice to deal with WiFi connections (and no longer the `wpasupplicant` daemon).

Find the SSID of your Wi-Fi network. It is usually the name of the wifi network. Example: BELL343. Let's call it `WIFINAME` To find the seen-bssed, run

```
$ sudo iw wlan0 scan | egrep '^BSS|SSID:'
```

The AA:BB:CC:DD:EE:FF address above `SSID: BELL343` is your seen-bssed. Let's call it `WIFIBSS`

Save the following block as new file in `/etc/NetworkManager/system-connections/BELL343`:

```
[connection]
id=WIFINAME
uuid=d8aa333b-2db3-4849-a9a4-c2aa3236ae29
type=wifi
permissions=
secondaries=
timestamp=1502254646

[wifi]
mac-address=
mac-address-blacklist=
mac-address-randomization=0
mode=infrastructure
seen-bssed=WIFIBSS
ssid=WIFINAME

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

[ipv4]
dns-search=
method=auto

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

Set the permissions on the new file to 0600.

```
sudo chmod 0600 /etc/NetworkManager/system-connections/BELL343
```

UNIT C-10

Software setup and RC remote control

Assigned to: Andrea

KNOWLEDGE AND ACTIVITY GRAPH

Requires: Laptop configured, according to Unit C-7 - Installing Ubuntu on laptops.

Requires: You have configured the Duckiebot. The procedure is documented in Unit C-8 - Duckiebot Initialization.

Requires: You have created a Github account and configured public keys, both for the laptop and for the Duckiebot. The procedure is documented in Unit I-28 - Setup Github access.

Results: You can run the joystick demo.

10.1. 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 Unit I-28 - Setup Github access.

Symptom: Other weird errors.

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

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

Or see the hints in the troubleshooting section on the previous page.

10.2. Update the system

The software used for the Duckiebots changes every day, this means that also the dependencies change. In order to check whether your system meets all the requirements for running the software and install all the missing packages (if any), we can run the following script:

```
$ cd ~/duckietown
$ /bin/bash ./dependencies_since_image.sh
```

This command will install only the packages that are not already installed in your system.

10.3. Set up the ROS environment on the Duckiebot

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

```
$ cd ~/duckietown
```

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

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

+ comment

If you are using ZSH then instead of this, you should call `source /opt/ros/kinetic/setup.bash`.

Then, build the workspace using:

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

☀ For more information about `catkin_make`, see Section 29.6 - `catkin_make`.

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

10.4. Clone the duckiefleet repository

Clone the relevant `duckiefleet` repository into `~/duckiefleet`.

See see Subsection 4.1.2 - Duckiefleet directory `DUCKIEFLEET_ROOT` to find the right `duckiefleet` repository.

In `~/.bashrc` set `DUCKIEFLEET_ROOT` to point to the directory:

```
export DUCKIEFLEET_ROOT=~/duckiefleet
```

10.5. Add your vehicle data to the robot database

Next, you need to add your robot to the vehicles database. This is not optional and required in order to launch any ROS scripts.

You have already a copy of the vehicles database in the folder `robots` of `DUCKIEFLEET_ROOT`.

Copy the file `emma.robot.yaml` to `robotname.robot.yaml`, where `robotname` is your robot's hostname. Then edit the copied file to represent your Duckiebot.

➡ For information about the format, see Section 4.2 - The "scuderia" (vehicle database).

Generate the machines file.

➡ The procedure is listed here: Section 4.3 - The `machines` file.

Finally, push your robot configuration to the duckiefleet repo.

10.6. Test that the joystick is detected

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

To make sure that the joystick is detected, run:

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

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

Check before you continue

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

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

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

➡ Consult again Section 8.11 - Create your user.

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

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

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

10.7. Run the joystick demo

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

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

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

Now make sure the motor shield is connected.

Run the command:

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

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

This is the expected result of the commands:

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

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

XXX Is all of the above valid with the new joystick?

Close the program using Ctrl-C.

1) Troubleshooting

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

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

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

Symptom: The left joystick does not work.

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

Symptom: The robot does not move at all.

Resolution: The cables are disconnected.

Resolution: The program assumes that the joystick is at `/dev/input/js0`. In doubt, see Section 10.6 - Test that the joystick is detected.

10.8. The proper shutdown procedure for the Raspberry Pi

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

To completely shutdown the robot, issue the following command:

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

Then wait 30 seconds.

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

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

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

Unit C-11
Reading from the camera

Knowledge and activity graph

Requires: You have configured the Duckiebot. The procedure is documented in Unit C-8 - Duckiebot Initialization.

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

TODO: put reference

Results: You know that the camera works under ROS.

11.1. Check the camera hardware

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

➡ The procedure is documented in Section 8.13 - Hardware check: camera.

11.2. Create two windows

On the laptop, create two Byobu windows.

➡ A quick reference about Byobu commands is in Unit I-25 - Byobu.

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.

11.3. First window: launch the camera nodes

In the first window, we will launch the nodes that control the camera. All the following commands should be run in the `~/duckietown` directory:

```
$ cd ~/duckietown
```

Activate ROS:

```
$ source environment.sh
```

Run the launch file called `camera.launch`:

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

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

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

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

☀ For more information about `roslaunch` and "launch files", see Section 29.3 - roslaunch.

11.4. Second window: view published topics

Switch to the second window. All the following commands should be run in the `~/duckietown` directory:

```
$ cd ~/duckietown
```

Activate the ROS environment:

```
$ source environment.sh
```

1) List topics

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

```
$ rostopic list
```

☀ For more information about `rostopic`, see Section 29.5 - rostopic.

You should see the following topics:

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

2) Show topics frequency

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

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

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

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

3) Show topics data

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

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

You should see a large sequence of numbers being printed to your terminal.

That's the "image" — as seen by a machine.

If you are Neo, then this already makes sense. If you are not Neo, in Unit C-13 - RC+camera remotely, you will learn how to visualize the image stream on the laptop using `rviz`.

use Ctrl-C to stop `rostopic`.

TODO: Physically focus the camera.

UNIT C-12

RC control launched remotely

Assigned to: Andrea

KNOWLEDGE AND ACTIVITY GRAPH

Requires: You can run the joystick demo from the Raspberry Pi. The procedure is documented in Unit C-10 - Software setup and RC remote control.

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

12.1. 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

12.2. 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 10.1 - Clone the Duckietown repository.

Then, you should build the repository.

➡ This procedure is documented in Section 10.3 - Set up the ROS environment on the Duckiebot.

12.3. 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 10.5 - Add your vehicle data to the robot database.

12.4. 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 29.3 - `roslaunch`.

12.5. Watch the program output using `rqt_console`

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

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

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

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

You should see a nice interface listing all the printouts in real time, completed with filters that can help you find that message you are looking for in a sea of messages. If `rqt_console` does not show any message, check out the *Troubleshooting* section below.

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

☀ For more information about `rqt_console`, see Section 29.2 - `rqt_console`.

12.6. Troubleshooting

Symptom: `rqt_console` does not show any message.

Resolution: Open `rqt_console`. Go to the Setup window (top-right corner). Change the "Rosout Topic" field from `/rosout_agg` to `/rosout`. Confirm.

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

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

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

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

➡ The procedure is documented in Section 18.3 - Local configuration.

Unit C-13
RC+camera remotely

Assigned to: Andrea

Knowledge and activity graph

Requires: You can run the joystick demo remotely. The procedure is documented in Unit C-12 - RC control launched remotely.

Requires: You can read the camera data from ROS. The procedure is documented in Unit C-11 - Reading from the camera.

Requires: You know how to get around in Byobu. You can find the Byobu tutorial in Unit I-25 - Byobu.

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

13.1. Assumptions

We are assuming that the joystick demo in Unit C-12 - RC control launched remotely worked.

We are assuming that the procedure in Unit C-11 - Reading from the camera succeeded.

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

13.2. Terminal setup

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

➡ A quick reference about Byobu commands is in Unit I-25 - Byobu.

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

13.3. First window: launch the joystick demo

In the first window, launch the joystick remotely using the same procedure in Section 12.4 - Start the demo.

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

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

13.4. 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.

13.5. 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
```

13.6. 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 29.4 - `rviz`.

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.

13.7. Proper shutdown procedure

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

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

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

UNIT C-14

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.

KNOWLEDGE AND ACTIVITY GRAPH

Requires: Time: 5 minutes.

Results: You will know about some useful shortcuts.

14.1. `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.

14.2. 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`.

UNIT C-15

Wheel calibration

Assigned to: Andrea Daniele

KNOWLEDGE AND ACTIVITY GRAPH

Requires: You can run the joystick demo remotely. The procedure is documented in Unit C-12 - RC control launched remotely.

Results: Calibrate the wheels of the Duckiebot such that it goes in a straight line when you command it to. Set the maximum speed of the Duckiebot.

15.1. Introduction

The motors used on the Duckiebots are called "Voltage-controlled motors". This means that the velocity of each motor is directly proportional to the voltage it is subject to. Even though we use the same model of motor for left and right wheel, they are not exactly the same. In particular, every motor responds to a given voltage signal in a slightly different way. Similarly, the wheels that we are using look "identical", but they might be slightly different.

If you drive the Duckiebot around using the joystick, you might notice that it doesn't really go in a straight line when you command it to. This is due to those small differences between the motors and the wheels explained above. Different motors can cause the left wheel and right wheel to travel at different speed even though the motors received the same command signal. Similarly, different wheels travel different distances even though the motors made the same rotation.

+ comment

It might be helpful to talk about the ROS Parameter Server here, or at least reference another page. -AD

15.2. What is the Calibration step?

We can counter this behavior by *calibrating* the wheels. A calibrated Duckiebot sends two different signals to left and right motor such that the robot moves in a straight line when you command it to.

The relationship between the velocities and the voltages of left and right motors are:

$$V_{\text{right}} = (g + r) * (v + \frac{1}{2}\omega l)$$
$$V_{\text{left}} = (g - r) * (v - \frac{1}{2}\omega l)$$

where V_{right} and V_{left} are the voltages for the two motors, g is called *gain*, r is called *trim*, v and ω are the desired linear and the angular velocity of the robot, and l is the distance between the two wheels. The gain parameter g controls the maximum speed of the robot. With $g > 1.0$, the vehicle goes faster given the same velocity command,

and for $g < 1.0$ it goes slower. The trim parameter r controls the balance between the two motors. With $r > 0$, the right wheel will turn slightly more than the left wheel given the same velocity command; with $r < 0$, the left wheel will turn slightly more the right wheel.

15.3. Perform the Calibration

1) Calibrating the `trim` parameter

The trim parameter is set to **0.00** by default, under the assumption that both motors and wheels are perfectly identical. You can change the value of the trim parameter by running the command:

```
$ rosservice call /robot name/inverse_kinematics_node/set_trim -- trim value
```

Calibrate the trim parameter using the following steps.

Step 1:

Make sure that your Duckiebot is ON and connected to the network.

Step 2:

On your Duckiebot, launch the joystick process:

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

Step 3:

Use some tape to create a straight line on the floor (Figure 15.1).

Figure 15.1. Straight line useful for wheel calibration

Step 4:

Place your Duckiebot on one end of the tape. Make sure that the Duckiebot is perfectly centered with respect to the line.

Step 5:

Command your Duckiebot to go straight for about 2 meters. Observe the Duckiebot from the point where it started moving and annotate on which side of the tape the Duckiebot drifted (Figure 15.2).

Figure 15.2. Left/Right drift

Step 6:

Measure the distance between the center of the tape and the center of the axle of the Duckiebot after it traveled for about 2 meters (Figure 15.3).

Make sure that the ruler is orthogonal to the tape.

Figure 15.3. Measure the amount of drift after 2 meters run

If the Duckiebot drifted by less than **10** centimeters you can stop calibrating the trim parameter. A drift of **10** centimeters in a **2** meters run is good enough for Duckietown. If the Duckiebot drifted by more than **10** centimeters, continue with the next step.

Step 7:

If the Duckiebot drifted to the left side of the tape, decrease the value of r, by running, for example:

```
$ rosservice call /robot name/inverse_kinematics_node/set_trim -- -0.1
```

Step 8:

If the Duckiebot drifted to the right side of the tape, increase the value of r, by running, for example:

```
$ rosservice call /robot name/inverse_kinematics_node/set_trim -- 0.1
```

Step 9:

Repeat the steps 4-8.

2) Calibrating the `gain` parameter

The gain parameter is set to **1.00** by default. You can change its value by running the command:

```
$ rosservice call /robot name/inverse_kinematics_node/set_gain -- gain value
```

Test the Duckiebot for different values of the gain parameter.

+ doubt

@liampaull What is the correct value to use for the gain? -AC

3) Store the calibration

When you are all done, save the parameters by running:

```
$ rosservice call /robot name/inverse_kinematics_node/save_calibration
```

The first time you save the parameters, this command will create the file

```
src/00-infrastructure/duckietown/config/baseline/calibration/kinematics/robot name.yaml
```

You can add and commit it to the repository.

> Note: we are in the process of rewriting the configuration system, so in a while "commit to the repository" is not going to be the right thing to do.

UNIT C-16

Camera calibration

Assigned to: Dzenan Lapandic

KNOWLEDGE AND ACTIVITY GRAPH

Requires: You can see the camera image on the laptop. The procedure is documented in Unit C-13 - RC+camera remotely.

Results: Calibrated camera of the robot

16.1. Intrinsic calibration

1) Setup

Make sure your Duckiebot is on, and both your laptop and Duckiebot are connected to the duckietown network. Next, download and print a PDF of the calibration checkerboard. Fix the checkerboard to a planar surface.

Figure 16.1

2) Calibration

Step 1:

Open three terminals on the laptop.

Step 2:

In the first terminal, remotely launch the joystick process:

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

Step 3:

In the second terminal run the camera calibration:

```
$ cd ~/duckietown
$ source environment.sh
$ source set_ros_master.sh robot name
$ roslaunch duckietown intrinsic_calibration.launch veh:=robot name raw:=true
```

You should see a display screen open on the laptop (Figure 16.2).

Figure 16.2

Position the checkerboard in front of the camera until you see colored lines overlaying the checkerboard. You will only see the colored lines if the entire checkerboard is within the field of view of the camera. You should also see colored bars in the sidebar of the display window. These bars indicate the current range of the checkerboard in the camera's field of view:

- X bar: the observed horizontal range (left - right)
- Y bar: the observed vertical range (top - bottom)
- Size bar: the observed range in the checkerboard size (forward - backward from the camera direction)
- Skew bar: the relative tilt between the checkerboard and the camera direction

Also, make sure to focus the image by rotating the mechanical focus ring on the lens of the camera.

Now move the checkerboard right/left, up/down, and tilt the checkerboard through various angles of relative to the image plane. After each movement, make sure to pause long enough for the checkerboard to become highlighted. Once you have collected enough data, all four indicator bars will turn green. Press the "CALIBRATE" button in the sidebar. Calibration may take a few moments. Note that the screen may dim. Don't worry, the calibration is working.

Figure 16.3

3) Save the calibration results

If you are satisfied with the calibration, you can save the results by pressing the "COMMIT" button in the side bar.

Figure 16.4

This will automatically save the calibration results on your Duckiebot:

```
~/duckietown/catkin_ws/src/00-infrastructure/duckietown/config/baseline/calibration/
camera_intrinsic/robot name.yaml
```

Step 7:

Now let's push the `robot name.yaml` file to the git repository. In the third terminal connect to your Duckiebot:

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

Update your local git repository:

```
$ cd ~/duckietown
$ git pull
```

Update your local git repository and push the changes to github:

```
$ git checkout -b git username-devel
$ git add ~/duckietown/catkin_ws/src/00-infrastructure/duckietown/config/baseline/
calibration/camera_intrinsic/robot name.yaml
$ git commit -m "add robot name intrinsic calibration file"
$ git push origin git username-devel
```

You can obtain the intrinsic calibration results on your laptop by updating your local git repository on your laptop:

```
$ cd ~/duckietown
$ git fetch
$ git checkout git username-devel
```

Before moving on to the extrinsic calibration, make sure to kill all running processes by pressing Ctrl-C in each of the terminal windows.

16.2. Extrinsic calibration

1) Setup

Arrange the Duckiebot and checkerboard according to Figure 16.5. Note that the axis of the wheels should be aligned with the y-axis.

Figure 16.5

Figure 16.6 shows a view of the calibration checkerboard from the Duckiebot. To ensure proper calibration there should be no clutter in the background.

Figure 16.6

2) Calibration

Step 1:

Open four terminals terminals on the laptop.

Step 2:

In the first terminal, remotely launch the joystick process:

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

Step 3:

In the second terminal launch the camera:

```
$ cd ~/duckietown
$ source environment.sh
$ source set_ros_master.sh robot name
$ roslaunch duckietown camera.launch raw:=1 veh:=robot name
```

Step 4:

In the third terminal run the ground projection node:

```
$ cd ~/duckietown
$ source environment.sh
$ source set_ros_master.sh robot name
$ roslaunch ground_projection ground_projection.launch veh:=robot name local:=1
```

Step 5:

In the fourth terminal, check that everything is working properly.

```
$ cd ~/duckietown
$ source environment.sh
$ source set_ros_master.sh robot name
$ rostopic list
```

You should see new ros topics:

```
/robot name/camera_node/camera_info
/robot name/camera_node/framerate_high_switch
/robot name/camera_node/image/compressed
/robot name/camera_node/image/raw
/robot name/camera_node/raw_camera_info
```

The ground_projection node has two services. They are not used during operation. They just provide a command line interface to trigger the extrinsic calibration (and for debugging).

```
$ rosservice list
```

You should see something like this:

```
...
/robot name/ground_projection/estimate_homography
/robot name/ground_projection/get_ground_coordinate
...
```

If you want to check whether your camera output is similar to the one at the Figure 16.6 you can start `rqt_image_view`:

```
$ rosrun rqt_image_view rqt_image_view
```

In the `rqt_image_view` interface, click on the drop-down list and choose the image topic:

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

Now you can estimate the homography by executing the following command:

```
$ rosservice call /robot name/ground_projection/estimate_homography
```

This will do the extrinsic calibration and automatically save the file to your laptop:

```
~/duckietown/catkin_ws/src/00-infrastructure/duckietown/config/baseline/calibration/
camera_extrinsic/robot name.yaml
```

As before, add this file to your local Git repository on your laptop, push the changes, and update your the local git repository on your Duckiebot.

> **Note:** we are in the process of rewriting the configuration system, so in a while "commit to the repository" is not going to be the right thing to do. We will communicate when the transition will happen

Unit C-17
Taking a log

Knowledge and activity graph

Requires: Unit C-11 - Reading from the camera
Results: A log

TODO: what is the launch file to run

TODO: where should the log go

TODO: Mounting the USB drive?

17.1. Verify a log

TODO: How to verify that it's good?

Part D

Operation manual - Duckietowns

Unit D-1
Duckietown parts

Duckietowns are the cities where Duckiebots drive. Here, we provide a link to all bits and pieces that are needed to build a Duckietown, along with their price tag. Note that while the topography of the map is highly customable, we recommend using the components listed below. Before purchasing components for a Duckietown, read Unit D-2 - Duckietown Appearance Specification 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.

Knowledge and activity graph

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: Assemblying a Duckietown.

TODO: Figure out costs

1.1. Bill of materials

Table 1.1. Bill of materials for Duckietown

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

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

1.2. Duckies

Duckies (Figure 1.1) are essential yet non functional.

Figure 1.1. The Duckies

1.3. Floor Mats

The floor mats (Figure 1.2) 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 1.2. 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 Unit D-2 - Duckietown Appearance Specification.

1.4. Duck Tape

We use duck (duct) tape of different colors (Figure 1.3) 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 1.3. The Duck Tapes

To verify how much tape you need for each road segment type, see Unit D-2 - Duckietown Appearance Specification.

1.5. Traffic Signs

Traffic signs (Figure 1.4) inform Duckiebots on the map of Duckietown, allowing them to make driving decisions.

Figure 1.4. The Signs

Depending on the chose road topograhy, the number of necessary road signal will vary. To design your Duckietown, see Unit D-2 - Duckietown Appearance Specification.

UNIT D-2

Duckietown Appearance Specification

Assigned to: Liam

This document describes the Duckietown specification. Any Duckietown not adhering to the rules described here cannot call itself a “Duckietown”, since it is not one. Additionally, any Duckietown not adhering to these rules may cause the Duckiebots to fail in unexpected ways. These are a set of rules for which a functional system has been verified.

2.1. Version history

Note here the changes to the specification, so that we are able to keep in sync the different Duckietowns.

- Version 2.0 - current version

2.2. Overview

Duckietown is built with two layers:

1. The first is the *floor layer*. The floor is built of interconnected exercise mats with tape on them.
2. The second layer is the *signals layer* and contains all the signs and other objects that sit on top of the mats.

> **Note:** the visual appearance of the area where the Duckietown is created is variable. If you discover that this appearance is causing negative performance, a “wall” of blank tiles constructed vertically can be used to reduce visual clutter.

2.3. Layer 1 - The Tile Layer

Each tile is a 2 ft x 2 ft square and is able to interlock with the others.

There are five primary types of tiles, as shown in Figure 2.1

(a) Straight tile (b) Turn tile (c) 3-way intersection tile

(d) 4-way intersection tile (e) Empty tile

Figure 2.1. The principal tile types in Duckietown

1) Tapes

There are 3 colors of tapes: white, yellow, and red.

White tape:

Proposition 3. A Duckiebot never collides with Duckietown if it never crosses or touches a white tape strip.

Here are some facts about the white tapes:

- White tapes must be solid (not dashed)
- The width of the white tape is 1 inch.
- The white tape is always placed on the right hand side of a lane. We assume that the Duckiebots drive on the right hand side of the road.

+ comment

this should be part of the “traffic rules” sections.

- For curved roads, the white lane marker is formed by five pieces of white tape, while the inner corner is formed by three pieces, placed according to the specifications in the image below, where the edge pieces are matched to adjacent straight or curved tiles (Figure 2.2).

Figure 2.2. The specification for a curved road tile

Yellow tape:

On a two-way road, the yellow tape should be dashed. Each piece should have a length of approximately **2 in** with a **1 in** gap separating each piece.

Yellow tapes on curves: see curved road image in white tape section, pieces at tile edges should be in center of lane, piece at the middle of the curve should be approximately 20.5 cm from middle of inner center white piece of tape, with approximated circular arc in between.

Red tape:

Red tapes MAY **only** appear on **intersection** tiles.

The red tape must be the full width of the duck tape roll and should cross the entire lane perpendicular to the lane.

+ comment
what is the width of the roll? 1 in? - AC

+ comment
Red and White rolls have a 2 inch thickness. Yellow is 1 inch wide. -JT

The placement of red tape should always be **under** yellow and white tape.

A Duckiebot navigates Duckietown by a sequence of:

- Navigating one or more straight ties until a red tape appears,
- Wait for the coordination signal,
- Execute an intersection traversal,

- Relocalize in a StraightTile.

The guarantee is:

Proposition . If the Duckiebot stops before or ON the red strip, no collisions are possible.

2) Topological Constraints During Map Construction

Here are some topological rule constraints that must be met:

1. An intersection can NOT be adjacent to a curved road tile or another intersection tile.
2. Any two adjacent non-empty tiles must have a feasible path from one to the other **of length two**: if they are adjacent, they must be connected.

Some examples of **non-conforming** topologies are shown in Figure 2.3.

(a) Topology violates rule 2 since the bottom two curved tiles are adjacent but not connected

(b) Topology violates rule 1 since curved tiles are adjacent to intersection tiles

(c) Topology violates rule 2 since left-most tiles are adjacent but not connected

Figure 2.3. Some non-conforming Duckietown map topologies

3) Parking Lots

Note: An experimental new development.

A parking lot is a place for Duckiebots to go when they are tired and need a rest.

A parking lot introduces three additional tile types:

1. **Parking lot entry tile**: This is similar to a straight tile except with a red stop in the middle. The parking lot sign (Figure 2.4o - parking) will be visible from this stop line.
2. **Parking spot tiles**:
3. **Parking spot access tiles**:

TODO: the tape on the spot and spot access tiles is currently not yet specified.

The following are the rules for a conforming parking lot:

1. One "parking spot" has size one tile.
2. From each parking spot, there is a path to go to the parking lot entry tile that does not intersect any other parking spot. (i.e. when a Duckiebot is parked, nobody will disturb it).
3. From any position in any parking spot, a Duckiebot can see at least two orthogonal lines or an sign with an April tag.

TODO: this point needs further specification

4) Launch Tiles

> **Note:** Experimental

A "launch tile" is used to introduce a new Duckiebot into Duckietown in a controllable way. The launch file should be placed adjacent to a turn tile so that a Duckiebot may "merge" into Duckietow,n once the initialization procedure is complete.

TODO: Specification for tape on the launch tile

A "yield" sign should be visible from the launch tile.

2.4. Layer 2 - Signage and Lights

IMPORTANT: All signage should sit with base on the floor and stem coming through the connection between the tiles. Generally, it is advisable to adhere the sign to the floor with double-sided tape. **Under no circumstances should the white (any other tape) be obscured.**

2.5. Traffic Signs

> **Requires:** To print and assemble the signs refer to Unit D-3 - Signage.

1) Specs

Center of signs are 13 cm height with apriltags of 6.5 cm sq. and a white border pasted below them.

2) Type

The allowable signs are as in Figure 2.4.

(a) stop (b) yield (c) no-right-turn (d) no-left-turn

(e) do-not-enter (f) oneway-right (g) oneway-left (h) 4-way-intersect

(i) right-T-intersect (j) left-T-intersect (k) T-intersection (l) pedestrian

(m) t-light-ahead (n) duck-crossing (o) parking

Figure 2.4. Duckietown Traffic Signs

Each sign is printed from the signs and tags doc

3) Placement

Signs may appear on the opposite side and at the corner of the adjacent tile from which they are viewed. In the absence of any signs, it is assumed that all network flows are allowed so a sign MUST be placed and visible whenever this is not the case.

Signs must only be placed on empty tiles, or next to one of the other tile types if on the border of a map. The sign placements for four different cases are shown in Figure 2.5. At intersections, from each stop line 2 signs should be clearly visible: 1) the intersection type (traffic light or stop sign) and 2) the intersection topology.

At present, 4-way intersections must be equipped with traffic lights for safe navigation.

(a) 4-way intersection

(b) 3-way intersection

(c) straight road

(d) curved road

Figure 2.5. Placement of Traffic Signs

On straight and curved roads, additional signs can be added as desired. Their placement is indicated in Figure 2.5c - straight road and Figure 2.5d - curved road. The signs should be placed at the border between two tiles and should face towards oncoming traffic as indicated.

In these figures the arrow is the direction of the sign.

2.6. Street Name Signs

1) Specs

- Font: arial.
- Color: Perhaps we could start with real-world settings: white as foreground and green as background.
- Border: currently no additional borders
- The rounded corners are modified into 90 degrees.

- Height: sign board height is 1.5 in. (**2.1 in**),
- Width: Currently 4.5 in for id 500-511. (**6.1 in +1.1 in "ST" or 5.5 in + 1.7 in "AVE"**)
- Alphabet = English upper case. Different writing systems may need different algorithms.
- Text direction: Horizontal for alphabetical languages

2) Placement

- **Similar to traffic light**: The street name should sit on a pole that is based at the corner of the tile outside of the allowable driving region. The bottom of the street name should be at a height of 7in, and allow a duckiebot to pass through. The street names should be visible from both sides of the road.

Every segment of road must have at least one road name sign.

Every turn tile should have a road name sign.

The placement of the road name signs is as indicated in Figure 2.6.

(a) Turn (b) Straight

Figure 2.6. Placement of Road Name Signs

Street name signs should never be perpendicular to the road - they are too big and obtrusive.

2.7. Traffic Lights

Requires: The assembly procedure for building the a traffic light is found in Unit D-6 - Traffic lights Assembly

1) Specs

To write: towrite

2) Placement

The lights must be at a height of exactly 20 cm above the center of the intersection tile.

The Raspberry Pi should sit on a pole that is based at the corner of the tile outside of the allowable driving region.

Unit D-3
Signage

> Assigned to: Liam

TODO: Describe the process of printing and assembling the signs

1) Assembly

TODO: Fig : Placement of an apriltag on a sign.

- each street sign must have an april tag according to the April Tags DB

UNIT D-4

Traffic lights Parts

Traffic lights regulate intersections in Duckietown. Here, we provide a link to all bits and pieces that are needed to build a traffic light, along with their price tag. You will need one traffic per either three, or four way intersections. The components listed below meet the appearence specifications described in Unit D-2 - Duckietown Appearance Specification.

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.

KNOWLEDGE AND ACTIVITY GRAPH

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: Assemblying a traffic light.

TODO: Estimate time and costs

4.1. Bill of materials

TABLE 4.1. BILL OF MATERIALS FOR TRAFFIC LIGHT

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

TODO: Complete table

4.2. Raspberry Pi

(Figure 4.1) are essential yet non functional.

Figure 4.1. The placeholder

UNIT D-5

Duckietown Assembly

Assigned to: Shiying

Unit D-6

Traffic lights Assembly

Assigned to: Shiying

UNIT D-7

Semantics of LEDS

Assigned to: ???

headlights: white, constant

Assumption:

- **20 fps** to do LED detection
- 1s to decide
- 3 frequencies to detect

tail lights: red, **6 hz square wave**

traffic light “GO” = green, 1 hz **square wave**

traffic light “STOP” = red, 1.5 Hz **square wave**

duckie light on top, state 0 = off

duckie light on top, state 1 = blue, **3 Hz, square wave**

duckie light on top, state 2 = ?, **2.5 Hz square wave**

duckie light on top, state 3 = ?, **2 Hz square wave**

Part E

Operation manual - Duckiebot with LEDs

UNIT E-1

Acquiring the parts for the Duckiebot `DB17-lc`

Upgrading your `DB17` (`DB17-wjd`) configuration to `DB17-lc` (`DB17-wjdlc`) starts here, with purchasing the necessary components. We provide a link to all bits and pieces that are needed to build a `DB17-lc` Duckiebot, along with their price tag. If you are wondering what is the difference between different Duckiebot configurations, read Unit C-1 - Duckiebot configurations.

In general, keep in mind that:

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

KNOWLEDGE AND ACTIVITY GRAPH

Requires: A Duckiebot in `DB17-wjd` configuration.

Requires: Cost: USD 77 + Bumpers manufacturing solution

Requires: Time: 21 Days (LED board manufacturing and shipping time)

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

Next: After receiving these components, you are ready to do some soldering before assembling your `DB17-lc` Duckiebot.

1.1. Bill of materials

TABLE 1.1. BILL OF MATERIALS

LEDs (DB17-1)	USD 10
LED HAT (DB17-1)	USD 28.20 for 3 pieces
Power Cable (DB17-1)	USD 7.80
20 Female-Female Jumper Wires (300mm) (DB17-1)	USD 8
Male-Male Jumper Wire (150mm) (DB17-1)	USD 1.95
PWM/Servo HAT (DB17-1)	USD 17.50
Bumpers	TBD (custom made)
40 pin female header (DB17-1)	USD 1.50
5 4 pin female header (DB17-1)	USD 0.60/piece
2 16 pin male header (DB17-1)	USD 0.61/piece
12 pin male header (DB17-1)	USD 0.48/piece
3 pin male header (DB17-1)	USD 0.10/piece
2 pin female shunt jumper (DB17-1)	USD 2/piece
5 200 Ohm resistors (DB17-1)	USD 0.10/piece
10 130 Ohm resistors (DB17-1)	USD 0.10/piece
Caster (DB17-c)	USD 6.55/4 pieces
4 Standoffs (M3.5 12mm F-F) (DB17-c)	USD 0.63/piece
8 Screws (M3.5x8mm) (DB17-c)	USD 4.58/100 pieces
8 Split washer lock (DB17-c)	USD 1.59/100 pieces
Total for DB17-wjd configuration	USD 212
Total for DB17-1c components	USD 77 + Bumpers
Total for DB17-wjd1c configuration	USD 299+Bumpers

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

1.2. LEDs

The Duckiebot is equipped with 5 RGB LEDs (Figure 1.1). LEDs can be used to signal to other Duckiebots, or just make *fancy* patterns.

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

Figure 1.1. The RGB LEDs

1) LED HAT

The LED HAT (Figure 1.2) provides an interface for our RGB LEDs and the computational stack. This board is a daughterboard for the Adafruit 16-Channel PWM/Servo HAT, and enables connection with additional gadgets such as ADS1015 12 Bit 4 Channel ADC, Monochrome 128x32 I2C OLED graphic display, and Adafruit 9-DOF IMU

Breakout - L3GD20H+LSM303. This item will require soldering.

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

Figure 1.2. The LED HAT

2) PWM/Servo HAT

The PWM/Servo HAT (Figure 1.3) mates to the LED HAT and provides the signals to control the LEDs, without taking computational resources away from the Rasperry Pi itself. This item will require soldering.

Figure 1.3. The PWM-Servo HAT

3) Power Cable

To power the PWM/Servo HAT from the battery, we use a short (30cm) angled male USB-A to 5.5/2.1mm DC power jack cable (Figure 1.4).

Figure 1.4. The 30cm angled USB to 5.5/2.1mm power jack cable.

4) Male-Male Jumper Wires

The Duckiebot needs one male-male jumper wire (Figure 1.5) to power the DC Stepper Motor HAT from the PWM/Servo HAT.

Figure 1.5. Premier Male-Male Jumper Wires

5) Female-Female Jumper Wires

20 Female-Female Jumper Wires (Figure 1.6) are necessary to connect 5 LEDs to the LED HAT.

Figure 1.6. Premier Female-Female Jumper Wires

1.3. Bumpers

These bumpers are designed to keep the LEDs in place and are therefore used only in configuration `DB17-1`. They are custom designed parts, so they must be produced and cannot be bought. We used laser cutting facilities.

Figure 1.7. The Bumpers

1.4. Headers, resistors and jumper

Upgrading `DB17` to `DB17-1` requires several electrical bits: 5 of 4 pin female header, 2 of 16 pin male headers, 1 of 12 pin male header, 1 of 3 pin male header, 1 of 2 pin female shunt jumper, 5 of 200 Ohm resistors and finally 10 of 130 Ohm resistors.

These items require soldering.

Figure 1.8. The Headers

Figure 1.9. The Resistors

1.5. Caster (DB17-c)

The caster (Figure 1.10) is an `DB17-c` component that substitutes the steel omnidirectional wheel that comes in the Magician Chassis package. Although the caster is not essential, it provides smoother operations and overall enhanced Duckiebot performance.

Figure 1.10. The caster wheel

To assemble the caster at the right height we will need to purchase:

- 4 standoffs (M3 12mm F-F) (Figure 1.11a - Standoffs for caster wheel.),
- 8 screws (M3x8mm) (Figure 1.11b - Screws for caster wheel.), and
- 8 split lock washers (Figure 1.11c - Split lock washers for caster wheel.).

(a) Standoffs for caster wheel.

(b) Screws for caster wheel.

(c) Split lock washers for caster wheel.

Figure 1.11. Mechanical bits to assemble the caster wheel.

TODO: missing figures, update caster bits figures

Unit E-2
Soldering boards for DB17-1

Note: General rule in soldering

- soldering the components according to the height of components - from lowest to highest

Assigned to: Shiying

Knowledge and activity graph

Requires: Duckiebot `DB17-1` parts. The acquisition process is explained in Unit E-1 - Acquiring the parts for the Duckiebot `DB17-1c`. The configurations are described in Unit C-1 - Duckiebot configurations.

Requires: Time: 30 minutes

Results: A `DB17-1` Duckiebot

2.1. General rules

General rule in soldering:

- soldering the components according to the height of components - from lowest to highest

2.2. 16-channel PWM/Servo HAT

(alternative instructions: how to solder on the PWM/Servo HAT)

1) Prepare the components

Put the following components on the table according the Figure

- GPIO Stacking Header for A+/B+/Pi 2
- Adafruit Mini Kit of 16-Channel PWM / Servo HAT for Raspberry Pi
  - 3x4 headers (4x)
  - 2-pin terminal block
  - 16-Channel PWM / Servo HAT for Raspberry Pi (1x)

Figure 2.1.

2) Soldering instructions

1. Solder the 2 pin terminal block next to the power cable jack

2. Solder the four 3x4 headers onto the edge of the HAT, below the words “Servo/PWM Pi HAT!”
3. Solder the GPIO Stacking Header at the top of the board, where the 2x20 grid of holes is located.

2.3. LSD board

TODO: add LSD board image, top and bottom.

1) Prepare the components

Put the following components according the figure on the table:

- 1 x 40 pin female header
- 5 x 4 pin female header
- 2 x 16 pin male header
- 1 x 12 pin male header
- 1 x 3 pin male header
- 1 x 2 pin female shunt jumper
- 5 x 200 Ohm resistors
- 10 x 130 Ohm resistors
- 3 x 4 pin male header for servos

Figure 2.2. LSD HAT and all of needed components

2) Soldering instructions

1. Put the resistors on the top of the board according to silkscreen markings, solder it on from the bottom side.

Tips:

1. Solder all female headers to the bottom of the board. Alignment becomes easy if the female headers are plugged into the PWM heat, and the LSD board rests on top.
2. Solder all male headers to the top of the board. Male header positions are outlined on the silkscreen.

2.4. LED connection

Parts list:

- 4 x 6” female-female jumper cable

Instructions:

1. Connect LED accordingly to silkscreen indication on PRi 2 LSD board
2. silkscreen legend: Rx, Gx, Bx are red, green, and blue channels, accordingly, where x is the LED number; C is a common line (either common anode or common cathode)
3. For adafruit LEDs are common anode type. The longest pin is common anode. Single pin on the side of common is red channel. The two other pins are Green and Blue channels, with the blue furthest from the common pin.
4. Both types of LEDs are supported. Use shunt jumper to select either common anode (CA) or common cathode (CC) on 3-pin male header. Note, however, that all LEDs on the board must be of the same type.

2.5. Putting everything together!

1. Stack the boards
   a. Screw the first eight standoffs into the Pi - provide hints on the location of standoffs and the suggested orientation of the boards w/r to the chassis
   b. connect the camera to the Pi [image showing the connector ?]
   c. Stack the DC/Stepper Motor HAT onto the Pi, aligning both sets of GPIO pins over each other and screw the standoffs to secure it. Try to not bend the camera connector too much during this step
   d. Stack the 16-channel PWM/Servo HAT onto the Pi, both sets of GPIO pins over each other and screw the standoffs to secure it
2. Slide the battery between the two chassis plates
3. Power the PWM/Servo HAT and Pi connecting them to the battery with the cables included in the duckiebox

4. Power the DC/Stepper motor from the PWM/Servo HAT using the male-to-male cable in the duckiebox, connect the positive
5. connect the Pi to the board
6. Finished!

UNIT E-3

Assembling the Duckiebot DB17-wjdlc

Assigned to: Shiying

KNOWLEDGE AND ACTIVITY GRAPH

Requires: Duckiebot DB17 parts. The acquisition process is explained in Unit E-1 - Acquiring the parts for the Duckiebot DB17-1c.

Requires: Soldering DB17 parts. The soldering process is explained in Unit E-2 - Soldering boards for DB17-1.

Requires: Having assembled the Duckiebot in configuration DB17. The acquisition process is explained in Unit C-5 - Assembling the Duckiebot DB17.

Requires: Time: about 30 minutes.

Results: An assembled Duckiebot in configuration DB17-wjdlc.

Before you continue in this chapter, make sure that you have assembled the duckiebot according the instruction of DB17-wjd configuration.

3.1. Put PWM HAT with 4 standoffs on the top of Stepper Motor HAT

Put a soldered Servo/PWM HAT board (in your Duckiebox) with 4 standoffs on the top of Stepper Motor HAT.

Figure 3.1.

1) Power Supply for PWM HAT

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

Figure 3.2. Male USB-A to 5.5/2.1mm DC power jack cable

TODO: finish above, estimate assembly time

Unit E-4

Bumper Assembly

Knowledge and activity graph

Requires: A Duckiebot in `DB17` configuration

Requires: Time: about ??? minutes.

TODO: estimate time.

Results: A Duckiebot with bumpers attached

1) Locate all required parts

The following should be included in your parts envelope (See image below for these components):

- 1x front bumper
- 1x rear bumperco
- 2x rear bumper brace
- 8x M3x10 pan head screws (McMaster #: 92005A120)
- 8x M3 nuts (McMaster #: 90591A250)
- 8x Reusable Cotter Pins (McMaster #: 98335A034)

The following is not included in your parts envelope but will be needed for assembly:

- Small Phillips screwdriver
- Small pliers (needle-nose is nice)
- Transparent tape

Figure 4.1

2) Reminder: Use care when assembling!

When assembling, be sure to be gentle! Tighten screws just enough so that the parts will remain stationary. When inserting LEDs, gently work them into their holders. While the acrylic is relatively tough, it can be fractured with modest force. We don't have many replacements (at this moment) so we may not be able to replace a broken part.

3) Remove protective paper

Peel protective layer off of all parts on all sides.

Figure 4.2

4) Rear Spacers Reassembly

Disassemble the rear spacers. KEEP ALL HARDWARE (you will be using the 2 short screws that were connected to the bottom of the spacer later). Reassemble in the configuration shown in the images below (note that the rear bumper braces now act as spacers to preserve the height offset). Reinstall the longer M3 screws that were originally connected at the top, and replace the bottom screws with 2 of your M3x10 screws included in the envelope. You may need the pliers to grasp the hex spacers when tightening. Note Reminder: Use care when assembling!

M3x10 screws attaching bottom rear brace:

Figure 4.3

Back View, fully assembled:

Figure 4.4

Old M3x~12 screws attaching top rear brace:

Figure 4.5

5) Mount Rear Bumper

Carefully guide rear bumper on to rear bumper brace tabs. Ensure that the hole for charging aligns with the charging port on your battery.

Figure 4.6

Locate 4 M3 nuts and 4 M3x10 screws. Place a nut in the wide part of the t-slot and

thread a screw into the nut as shown in the following pictures. Note Use care when assembling! If you are having trouble with the nuts falling out, take a small piece of transparent tape and place it over both sides of the t-slot with the nut inside. It won't look as nice but it will be much easier to assemble.

Figure 4.7

Figure 4.8

Figure 4.9

The rear bumper should now look like this:

Figure 4.10

Install the left and right rear LEDs carefully by pushing them into their flexure holders. Note Use care when assembling!

Figure 4.11

Install 6 removable cotter pins in the positions below. The top 2 will hold the shell on, the bottom 4 will hold a vehicle detector target (if you have one).

Figure 4.12

Congrats! your rear bumper assembly is complete!

6) Mount Front Bumper

Take 2 M3x10 screws and 2 M3 nuts and install them as shown in the following pictures. The first picture shows the correct holes to mount these screws (The correct position is the widest pair of 3mm holes beside the camera). The nuts should tightened on by a few threads (these are the two nuts that are not yet tightened at the top of the second picture):

Figure 4.13

Figure 4.14

Take the front bumper and carefully press the LEDs into the flexure holders. Take care that the wires are routed as suggested. Also note that the front center LED wire should not be crushed between the bumper and the right spacer (you will likely fracture the bumper if you try to force it). The center LED should be bent at a right angle in the direction that the wire is fed through the body.

Figure 4.15

Position the bumper so that the nuts align with the t-slots. You may need to loosen or tigthen the screws to align the nuts. You may also need to angle the front bumper when inserting to get it past the camera screws.

Figure 4.16

Gently tighten the nuts. The front bumper should now stay in position (but not done securing yet!)

Figure 4.17

Remember those short M3 screws that we said to keep track of? Now it's time to use

them! Mount those 2 screws and 2 M3 nuts at the position shown below (to keep the bumper from being tilted up and breaking).

Figure 4.18

Figure 4.19

Slide the 2 remaining removable cotter pins into the holes at the top and congrats! You are finished assembling your front bumper!

7) Mount Vehicle Detector Target (optional)

If you were given a vehicle detector target, now it is time to use those removable cotter pins in the back! Attach as shown in the following image. To recharge, simply remove the target and plug in the battery.

Figure 4.20

Unit E-5
C1 (LEDs) setup

Assigned to: Shiying?

5.1. Connecting Wires to LEDs

The LEDs are common anode type. The longest pin is called the common. The single pin on the side of common is red channel. The two other pins are Green and Blue channels, with the blue furthest from the common pin.

Use the long wires with two female ends. Attach one to each of the pins on the LED.

To figure out the order to connect them to the LSD hat, use the legend on the silkscreen and the information above. i.e. RX - means the red pin, CX - means the common, GX means the green, and BX means the blue. The “X” varies in number from 1-5 depending on which LED is being connected as discussed in the next section.

5.2. Connecting LEDs to LSD Hat

Define the following names for the lights:

“top” = top light - the “top” light is now at the bottom fl = front left fr = front right br = back right bl = back left

The LEDs are wired according to Figure 5.1.

Figure 5.1

Mappings from the numbers on the LED hats to the positions shown (TOP is now the one in the middle at the front) FR - 5 BR - 4 TOP - 3 BL - 2 FL - 1

5.3. Running the Wires Through the Chassis

It is advised that the LED cables are routed through the positions noted in the images below before installing the bumpers:

Front Left, Front Middle, and Front Right LED Wiring suggestion:

Figure 5.2

Rear Left LED Wiring Suggestion:

Figure 5.3

Rear Right LED Wiring Suggestion:

Figure 5.4

5.4. Final LED tweaks, Confirm LED Function and Placement

Adjust the LED terminals (particularly in the front) so that they do not interfere with the wheels. This can be accomplished by bending them up, away from the treads.

Part F
Preliminaries

TODO: to write

Unit F-1
Chapter template

Theory chapters benefit from a standardized exposition. Here, we define the template for these chapters. Rememeber to check Unit B-2 - Basic Markduck guide for a comprehensive and up-to-date list of Duckiebook supported features.

1.1. Example Title: PID control

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

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

Knowledge and activity graph

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

Knowledge and activity graph

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: [7], ...

Recommended: observability/detectability and controllability/reachability: [7]

Recommended: Discrete time PID: [7]

Recommended: Bode diagrams: [7]

Recommended: Nyquist plots: [7]

Recommended: [...]

1.2. 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:

$$\begin{aligned} \dot{x}_t &= Ax_t + Bu_t \\ y &= Cx_t + Du_t \end{aligned} \tag{1}$$

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

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

[...]

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 $\tilde{y}_t = y_t + n_t, n_t \sim \mathcal{N}(0, \sigma)$, find a set of PID coefficients that meet the specified requirements for: - stability, - performance, - robustness.

as shown in (Figure 1.1).

Figure 1.1. A classical block diagram for PID control. We like to use a lot of clear figures in the Duckiebook.

1.3. Introduced Notions

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

Definition 13. (Reference signals) A reference signal $\tilde{y}_t \in \mathcal{L}_2(\mathcal{T})$ is ...

Definition 13 - Reference signals 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 remember the definition

of $\mathcal{L}_2(\cdot)$, read: [7]

Definition 14. (Another definition) Lorem

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

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

$$\mathcal{L}orem$$

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.

1.4. Examples

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

1) Theoretical Examples

More academic examples

T-Example 1:

Immagine a spring-mass-damper system...

T-Example M:

[...]

2) Implementation Examples

More Duckiebot related examples

I-Example 1:

I-Example M:

[...]

1.5. Pointers to Exercises

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

1.6. Conclusions

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

1.7. Next Steps

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

KNOWLEDGE AND ACTIVITY GRAPH

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

- ☀ learn all there is to know about PID: [7]
- ☀ become a linear algebra master: Matrix cookbook

1.8. References

Do not include a reference chapter. References are automatically compiled to the Bibliography Section.

Author: Jacopo

Maintainer: Jacopo

Point of contact: Jacopo

Unit F-2
Symbols and conventions

Assigned to: Andrea

2.1. Conventions

You should not have to use presentation macros like `\mathcal`, `\boldsymbol`, etc.; rather, for each class of things that we have (set, matrices, random variables, etc.) we are going to define a LaTeX macro.

1) Sets

Use the macro `\aset{X}` to refer to the set $\mathcal{X}$.

2) Matrices

Use the macro `\amat{M}` to refer to the matrix $\mathbf{M}$.

3) Tuples

To indicate tuples, use the macro `\tup`, which produces $\langle a, b, c \rangle$.

4) Time series

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

✖ Consider the function $x(t)$.

✔ Consider the function x_t.

To refer to the time variable, use `\Time`: $t \in \mathbb{T}$.

5) Random variables

To refer to a random variable, use the macro `\rv`. This is rendered using a **bold** symbol.

Example . $p(\boldsymbol{x} = x_0)$ is the probability that the random variable $\boldsymbol{x}$ has the value $x_0 \in \mathcal{X}$.

6) Well-known sets

Use `\reals` for the real numbers.

Use `\nats` for the natural numbers.

Use `\ints` for the integers numbers.

TABLE 2.1. BASIC SYMBOLS

command	result	
`\aset{X}, \aset{Y}`	$\mathcal{X}, \mathcal{Y}$	Symbols for sets
`\amat{M}, \amat{P}`	$\mathbf{M}, \mathbf{P}$	Symbols for matrices
`\avec{u}, \avec{v}`	$\mathbf{u}, \mathbf{v}$	Symbols for vectors
`\nats`	$\mathbb{N}$	Natural numbers
`\ints`	$\mathbb{Z}$	Integers
`\reals`	$\mathbb{R}$	Real numbers
`\definedas`	$\triangleq$	Defined as
`\tup{a,b,c}`	$\langle a, b, c \rangle$	Tuples
`\Time`	$\mathbb{T}$	Time axis

2.2. Spaces

Here are some useful symbols to refer to geometric spaces.

TABLE 2.2. SPACES

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

States and poses:

TABLE 2.3. POSES AND STATES

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

UNIT F-3
Sets

KNOWLEDGE AND ACTIVITY GRAPH

Results: k:sets

3.1. Definition

Definition 15. (Set) A set $\mathcal{X} = \{x_1, x_2, \ldots\}$ is a well-defined collection of distinct *elements*, or *members* of the set, x_i, $i = 1, 2, \ldots$.

3.2. Maps

KNOWLEDGE AND ACTIVITY GRAPH

Requires: k:sets
Results: k:maps

3.3. Definition

We define a *function* (or *map*) as a mapping between *sets*.

Definition 16. (Function) A function $f : \mathcal{X} \to \mathcal{Y}$ is a mapping between the sets $\mathcal{X}$ and $\mathcal{Y}$. For every input element $x \in \mathcal{X}$, the mapping will associate an output $y = f(x) \in \mathcal{Y}$.

3.4. Properties of maps

Maps can be classified by the nature of the relationship between inputs and outputs in: *injective*, *surjective* or *bijective* add-ref.

1) Injective maps

TODO: to write

2) Surjective maps

TODO: to write

3) Bijective maps

TODO: to write

Unit F-4
Numbers

Knowledge and activity graph

Requires: k:sets
Results: k:naturals, k:integers, k:reals

4.1. Natural numbers

$\mathbb{N} = \{0, 1, 2, \cdots\}$

The natural numbers are the set positive numbers, including zero.

Given two natural their addition is always a natural number:

$$a + b = c \in \mathbb{N}, \forall a, b \in \mathbb{N}. \tag{1}$$

The same does not hold of the subtraction operation:

$$a - b = c \in \mathbb{N} \iff a \geq b.$$

For this reason set of integer numbers is defined.

4.2. Integers

$\mathbb{Z} = \{\cdots, -2, -1, 0, 1, 2, \cdots\}$

The integers are the set of positive and negative natural numbers, including the zero. By definition, the set of integers includes the naturals: $\mathbb{Z} \subset \mathbb{N}$.

The sum (i.e., addition and subtraction) of two integers is always an integer:

$$a + b = c \in \mathbb{Z}, \forall a, b \in \mathbb{N}$$
$$a - b = c \in \mathbb{Z}, \forall a, b \in \mathbb{N}.$$

The multiplication of two integers is always an integer, but the same does not apply for the division operation:

$$\frac{a}{b} = c \in \mathbb{Z} \iff a = kb, k \in \mathbb{Z}, b \neq 0.$$

For this reason the rational numbers are introduced.

4.3. Rationals

The set of rational numbers includes all fractions of integers: $\mathbb{Q} = \{c | \frac{a}{b} = c, a, b \in \mathbb{Z}, b \neq 0\}$.

The set of rational number is complete under sum and product (i.e., multiplication and division), but not under other operations such as the root. E.g., $\sqrt{2}$ cannot be expressed as a fraction of two integers. These numbers are not rational, and therefore

are defined as irrationals.

4.4. Irrationals

Irrational numbers are all those numbers that cannot be expressed as a fraction. Notable examples of irrational numbers include the aforementioned $\sqrt{2}$, but even pi (π) and the Euler number (e).

Irrational numbers are not typically referred to as as a set by themselves, rather, the union of the rational and irrational numbers defines the set of *reals*.

4.5. Reals

The real numbers ($\mathbb{R}$) are arguably the most used set of numbers, and are often considered the default set if no specification is provided.

The real numbers are defined as the union of rational and irrational numbers, and therefore by definition include the integers and the naturals.

The reals are still not complete under all "canonical" operations. In fact, there is no solution to the root (of even index) of a negative number.

For this reason, the complex numbers are introduced.

4.6. Complex

Complex numbers are defined as the sum of a real and an imaginary part:

$$z = a + ib, a, b \in \mathbb{R}, i = \sqrt{-1}$$

and can be represented on the plane of Gauss, a Cartesian plane featuring the real part of z, $Re(z) = a$, on the x-axis and the imaginary part, $Im(z) = b$, on the y-axis (Figure 4.1).

Figure 4.1. The Gaussian plane is used to represent complex numbers

Complex numbers introduce the concept of *phase* of a number, which is related to its "orientation", and are invaluable for describing many natural phenomena such as electricity and applications such as signal decoders.

For more information on the algebra and properties of natural numbers:

➡ Unit F-5 - Complex numbers.

UNIT F-5

Complex numbers

5.1. Powers of i

The powers of i

$$
\begin{aligned}
i &= sqrt{-1} \\
i^2 &= -1 \\
i^3 &= i^2 \cdot i = -i \\
i &= i^2 \cdot i^2 = 1 \\
i^5 &= i^4 / cdot i = i \\
&\vdots
\end{aligned}
$$

5.2. Complex conjugate

UNIT F-6
Linearity and Vectors

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 behavior 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.

Note: This chapter is not intended to be a comprehensive compendium of linear algebra.

➡ this reference

✳ this other reference

TODO: add references throughout all chapter

KNOWLEDGE AND ACTIVITY GRAPH

Requires: Real numbers are complex for you?: Number theory addref

Requires: ∀ is a typo for A and ∈ are Euros? Mathematical symbolic language.

6.1. Linearity

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 can be 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.

Definition 17. (Linearity) A function $f : \mathcal{X} \to \mathcal{Y}$ is linear when, $\forall x_i \in \mathcal{X}$, $i = \{1, 2\}$, and $\forall a \in \mathbb{R}$:

$$f(ax_1) = af(x_1), \quad \text{and:} \tag{2}$$
$$f(x_1 + x_2) = f(x_1) + f(x_2) \tag{3}$$

Condition (2) is referred to as the property of *homogeneity* (of order 1), while condi-

tion (3) is referred to as *additivity*.

Remark 2. (Superposition Principle) Conditions (2) and (3) can be merged to express the same meaning through:

$$f(ax_1 + bx_2) = af(x_1) + bf(x_2), \forall x_i \in \mathcal{X}, i = \{1, 2\}, \forall a, b \in \mathbb{R}. \tag{4}$$

This equivalent condition (4) 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.

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

$$\mathbf{v} = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \in \mathbb{R}^{n \times 1} \equiv \mathbb{R}^n, \tag{5}$$

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

Remark 3. (Vector notation) A more general notation for tuples can be used when denoting vectors:

$$\mathbf{v} = \langle v_1, \cdots, v_n \rangle. \tag{6}$$

In these preliminaries, we will adopt the (5) "engineering" notation as it arguably simplifies remembering vector-matrix operations (Unit F-9 - Matrices and vectors).

You can imagine a vector Figure 6.1 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 *magnitude* of the vector (sometimes referred to even as *modulus*), and it can be derived through the vector's components.

Definition 19. (Length of a vector) We define the length, or *modulus*, of a vector $\mathbf{v} \in \mathbb{R}^n$ as:

$$\|\mathbf{v}\| = \sqrt{v_1^2 + \cdots + v_n^2} \in \mathbb{R}. \tag{7}$$

Remark 4. (2-norm) Generally speaking, it is not always possible to define the length of a vector (addref). But when it is possible (e.g., in 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 (7).

We will discuss norms more in detail in Unit F-14 - Norms.

Definition 20. (Unit vector) A unit vector, or *versor*, is a vector $\mathbf{e}$ of of unit length:

$$\|\mathbf{e}\| = 1. \tag{8}$$

Unit vectors are used to define the directions of the components of a vector, allowing for an algebraic rather than vectorial representation. As we will see in Subsection 6.2.1 - Vector algebra, this will make the algebra of vectors more intuitive.

Figure 6.1. A vector, its components expressed as multiples of unit vectors.

example

Let $\mathbf{v} \in \mathbb{R}^3$ be a vector defined in the Cartesian space. Let, moreover, $(\mathbf{i}, \mathbf{j}, \mathbf{k})^T$ be the versor of the Cartesian axis, i.e.:

$$\begin{aligned} \mathbf{i} &= [1, 0, 0]^T; \\ \mathbf{j} &= [0, 1, 0]^T; \\ \mathbf{k} &= [0, 0, 1]^T. \end{aligned} \tag{9}$$

Then, a vector can be written equivalently in vector or algebraic form: $\mathbf{v} = [v_1, v_2, v_3]^T = v_1\mathbf{i} + v_2\mathbf{j} + v_3\mathbf{k}$. Unit vectors are sometimes explicitly denoted with a hat (^), e.g., $\hat{\mathbf{i}}, \hat{\mathbf{j}}, \hat{\mathbf{k}}$.

Remark 5. (Normalizing vectors) Every vector can be made into a unit vector, or *normalized*, by dividing each of its components by the vector's magnitude:

$$\hat{\mathbf{v}} = \frac{\mathbf{v}}{\|\mathbf{v}\|} = \left[\frac{v_1}{\|\mathbf{v}\|}, \frac{v_2}{\|\mathbf{v}\|}, \frac{v_3}{\|\mathbf{v}\|}\right]^T. \tag{10}$$

1) Vector algebra

We here define operations amongst two given vectors defined in the same space: $\mathbf{u} = [u_1, u_2, u_3]^T, \mathbf{v} = [v_1, v_2, v_3]^T \in \mathbb{R}^3$.

Vectorial Sum:

The sum of two vectors is a vector, and its components are the sum of the two vectors components.

Definition 21. (Vectorial sum)

$$\mathbf{u} + \mathbf{v} = [u_1 + v_1, u_2 + v_2, u_3 + v_3]^T. \tag{11}$$

Remark 6. (Sum) Mathematical operations come in pairs, which represent the same concept. A *sum* operation, sometimes more extensively referred to as the *algebric sum*, is the concept of summing, i.e., it includes both addition and subtraction. (A subtraction is nothing but an addition between positive and negative numbers.)

The parallelogram law helps visualize the results of the vectorial sum operation Fig-

ure 6.2.

Figure 6.2. The sum of two vectors can be visualized with the parallelogram law.

Dot, or scalar, product:

The dot, or scalar, product of two vectors ($\mathbf{u}$,$\mathbf{v} \in \mathbb{R}^3$) is a scalar ($a \in \mathbb{R}$) equal to the sum of the products of the components of the vectors. Equivalently, it can be expressed as the product of the magnitudes of the two vectors times the cosine of the angle between them, $\phi \in [0, 2\pi)$.

Definition 22. (Scalar product)

$$\mathbf{u} \cdot \mathbf{v} = u_1 v_1 + u_2 v_2 + u_3 v_3 = \|u\| \|v\| \cos(\phi) \in \mathbb{R} \tag{12}$$

The dot product is a measure of the *projection* of vectors on one another (Figure 6.3).

Note: When the two vectors are perpendicular, or orthogonal, the dot product is zero ($\cos(\pi/2) = 0$). This fact is often used as a test for orthogonality. Orthogonality is an important concept for linear spaces, as the most "efficient" basis are orthogonal.

Figure 6.3. The scalar product between two vectors measures the projection of one on each other.

Cross, or vector, product:

While the dot product depends on the metric chosen in the space (the Euclidian

norm, in our case), the cross product even requires the definition of an orientation, or handedness.

Proposition 4. (Standard Basis) In the Euclidian space $\mathbb{R}^3$, $\hat{\mathbf{i}}, \hat{\mathbf{j}}, \hat{\mathbf{k}}$ are the unit vectors for the standard basis, which is right handed.

In a right handed reference system such as the standard basis, the right hand rule (Figure 6.4) is the handy-est way to identify the direction of the vector resulting from a cross product.

ProTip: There is a valid reason for which it is called the *right hand* rule. Don't use your left hand because you are holding a pen with the right one.

Figure 6.4. The right hand rule points in the direction of the resulting vector from a cross product.

The cross, or vector, product between two vectors ($\mathbf{u}$, $\mathbf{v} \in \mathbb{R}^3$) is a vector that is orthogonal to each of the two vectors, hence is normal, or perpendicular, to the plane containing them. Its magnitude is given by the product of their magnitude times the sine of the angle between them, and its direction is indicated by the normal unit vector ($\hat{\mathbf{n}} \in \mathbb{R}^3$), identified by the right hand rule.

Definition 23. (Vector product)

$$\mathbf{u} \times \mathbf{v} = [u_2 v_3 - u_3 v_2, u_3 v_1 - u_1 v_3, u_1 v_2 - u_2 v_1]^T = \|u\| \|v\| \sin(\phi) \hat{\mathbf{n}}. \tag{13}$$

Remark 7. (Geometric interpretation) A cross product encodes two pieces on information: a direction, which is *orthogonal* to the plane spanned by the two vectors, and a magnitude, which is equal to the area of the parallelogram having $\mathbf{u}$, and $\mathbf{v}$ as sides.

> Note: Keeping **(13)** and Remark 7 - Geometric interpretation in mind, it should be intuitive to understand that:

$$\begin{aligned} \mathbf{v} \times \mathbf{v} &= \mathbf{0}, \forall \mathbf{v} \in \mathbb{R}^n, \\ \mathbf{v} \times \mathbf{0} &= \mathbf{0}, \forall \mathbf{v} \in \mathbb{R}^n. \end{aligned} \tag{14}$$

Note: The zero vector ($\mathbf{0}$) is a vector with zero magnitude, not the same as the number zero (0).

Note: Each component of $\mathbf{w}$ is the difference of the products of the two *other* components of $\mathbf{u}$, and $\mathbf{v}$, in the order given by the chosen handedness of the basis. This combination resembles a *cross* (Figure 6.5), from which the name of *cross product*.

Figure 6.5. Each component of the resulting vector is the product of the alternated other components, forming a cross.

Note: The components of a cross product can be computed through the Sarrus rule (see Section 7.6 - Determinant).

As consequence of the vectorial product's definition and right handedness of the basis, the following hold true in the Cartesian space:

$$\begin{aligned} \hat{\mathbf{i}} \times \hat{\mathbf{j}} &= \hat{\mathbf{k}} \\ \hat{\mathbf{j}} \times \hat{\mathbf{k}} &= \hat{\mathbf{i}} \\ \hat{\mathbf{k}} \times \hat{\mathbf{i}} &= \hat{\mathbf{j}}. \end{aligned} \tag{15}$$

2) Properties of vectors

In this section we highlight the properties of vector operations, that derive from their definitions.

Sum:

The vector sum obejs the following:

- $\mathbf{u} + \mathbf{v} = \mathbf{v} + \mathbf{u}$,
- $(\mathbf{u} + \mathbf{v}) + \mathbf{w} = \mathbf{u} + (\mathbf{v} + \mathbf{w})$,
- $a(\mathbf{u} + \mathbf{v}) = a\mathbf{u} + a\mathbf{v}$,
- $(a + b)\mathbf{u} = a\mathbf{u} + b\mathbf{u}$,
- $\mathbf{u} + \mathbf{0} = \mathbf{u}$, therefore $\mathbf{u} + (-\mathbf{u}) = \mathbf{0}$.

Dot product:

Letting $\phi \in [0, 2\pi)$ be the angle between two vectors $\mathbf{u}, \mathbf{v}$, the dot product obejs the following:

- $\mathbf{u} \cdot \mathbf{v} = \|\mathbf{u}\|\|\mathbf{v}\| \cos(\phi)$,
- $\mathbf{u} \cdot \mathbf{u} = \|\mathbf{u}\|^2$,
- $\mathbf{u} \cdot \mathbf{v} = \mathbf{v} \cdot \mathbf{u}$,
- $\mathbf{u} \cdot (\mathbf{v} + \mathbf{w}) = \mathbf{u} \cdot \mathbf{v} + \mathbf{u} \cdot \mathbf{w}$,
- $a(\mathbf{u} \cdot \mathbf{v}) = (a\mathbf{u}) \cdot \mathbf{v}$,
- $\mathbf{0} \cdot \mathbf{u} = 0$
- $\mathbf{u} \cdot \mathbf{v} = 0 \iff \mathbf{u} = \mathbf{0}, \mathbf{v} = \mathbf{0}$, or $\mathbf{u} \perp \mathbf{v}$.

Cross product:

Letting $\phi \in [0, 2\pi)$ be the angle between two vectors $\mathbf{u}, \mathbf{v}$, the cross product obejs the following:

- $\mathbf{u} \times \mathbf{v} = \|\mathbf{u}\|\|\mathbf{v}\| \sin(\phi)\hat{\mathbf{n}}$,
- $\mathbf{u} \times \mathbf{v} = -\mathbf{v} \times \mathbf{u}$,
- $(a\mathbf{u}) \times \mathbf{v} = \mathbf{u} \times (a\mathbf{v}) = a(\mathbf{u} \times \mathbf{v})$,
- $\mathbf{u} \times (\mathbf{v} + \mathbf{w}) = \mathbf{u} \times \mathbf{v} + \mathbf{u} \times \mathbf{w}$,
- $\mathbf{u} \cdot (\mathbf{v} \times \mathbf{w}) = (\mathbf{u} \times \mathbf{v}) \cdot \mathbf{w}$,
- $\mathbf{u} \times (\mathbf{v} + \mathbf{w}) = (\mathbf{w} \cdot \mathbf{u})\mathbf{v} - (\mathbf{v} \cdot \mathbf{u})\mathbf{w} \neq (\mathbf{u} \times \mathbf{v}) + \mathbf{w}$,
- $\mathbf{u} \times \mathbf{v} = 0 \iff \mathbf{u} = \mathbf{0}, \mathbf{v} = \mathbf{0}$, or $\mathbf{u} \parallel \mathbf{v}$.

6.3. Linear dependance

Definition 24. (Linear dependance) Two or more vectors $\{\mathbf{v_1}, \cdots, \mathbf{v_n}\}$ are *linearly dependant* if there exists a set of scalars $\{a_1, \cdots, a_k\}, k \leq n$, that are *not all zero*, such that:

$$a_1\mathbf{v_1} + \cdots + a_k\mathbf{v_k} = \mathbf{0}. \tag{1}$$

Note: When (1) is true, it is possible to write at least one vector as a linear combination of the others.

Definition 25. (Linear independance) Two or more vectors $\mathbf{v_1}, \cdots, \mathbf{v_n}$ are *linearly independant* if (1) can be satisfied only by $k = n$ and $a_i = 0, \forall i = 1, \cdots, n$.

6.4. Pointers to Exercises

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

TODO: add exercises

6.5. Conclusions

In this section we have defined the fundamental concept of linearity and linear dependance. Moreover, we have introduced vectors, with their operations and algebraic properties.

Vectors and linearity are the base for understanding linear spaces, which are useful because they introduce some fundamental concepts related to the foundation of

modeling of natural phenomena. Modeling will be invaluable in understanding the behavior of systems, and a powerful tool to *predict* future behaviors of the system, and *control* them when needed.

KNOWLEDGE AND ACTIVITY GRAPH

- ☀ [15]
- ☀ Matrix cookbook

Author: Jacopo

Maintainer: Jacopo

Point of contact: Jacopo

Unit F-7
Matrices basics

Assigned to: Jacopo

Knowledge and activity graph

Requires: k:basic_math
Requires: k:linear_algebra
Results: k:matrices

A matrix:

$$\mathbf{A} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \in \mathbb{R}^{m \times n} \tag{1}$$

is a table ordered by (m) horizontal rows and (n) vertical columns. Its elements are typically denoted with lower case latin letters, with subscripts indicating their row and column respectively. For example, a_{ij} is the element of A at the i-th row and j-th column.

Figure 7.1. A matrix. This image is taken from [17]

Note: A vector is a matrix with one column.

7.1. Matrix dimensions

The number of rows and columns of a matrix are referred to as the matrix *dimensions*. $\mathbf{A} \in \mathbb{R}^{m \times n}$ has dimensions m and n.

Definition 26. (Fat matrix) When $n > m$, i.e., the matrix has more columns than rows, $\mathbf{A}$ is called *fat* matrix.

Definition 27. (Tall matrix) When $n < m$, i.e., the matrix has more rows than columns, $\mathbf{A}$ is called *tall* matrix.

Definition 28. (Fat matrix) When $n = m$, $\mathbf{A}$ is called *square* matrix.

Note: Square matrices are particularly important.

7.2. Matrix diagonals

- Main diagonal
- Secondary diagonal

7.3. Diagonal matrix

Definition 29. (Diagonal matrix) A diagonal matrix has non zero elements only on its main diagonal.

$$\mathbf{A} = \begin{bmatrix} a_1 1 & 0 & \dots & 0 \\ 0 & a_2 2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & a_n n \end{bmatrix}$$

7.4. Identity matrix

Definition 30. (Identity matrix) An identity matrix is a diagonal square matrix with all elements equal to one.

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & 1 \end{bmatrix}$$

7.5. Null matrix

Definition 31. (Null matrix) The null, or Zero, matrix is a matrix whos elements are all zeros.

$$\mathbf{0} = \begin{bmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix}$$

7.6. Determinant

- 2x2
- 3x3
- nxn

7.7. Rank of a matrix

7.8. Trace of a matrix

UNIT F-8

Matrix inversions

Assigned to: Jacopo

8.1. Adjugate matrix

$\mathbf{A}\mathrm{adj}(\mathbf{A}) = \det(A)\mathbf{I}$

8.2. Matrix Inverse

Square matrix:

$\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$

8.3. Nonsingularity of a matrix

Exercise: Calculate a (square) Matrix Inverse

Exercise: Inverting a well-conditioned matrix (practice)

Exercise: Inverting an ill-conditioned matrix (practice)

UNIT F-9
Matrices and vectors

Assigned to: Jacopo

- matrix-vector product

9.1. 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) Eigenvalues and Eigenvectors

- for square matrices
- for rectangular matrices (topic for advanced-linear-algebra?)
- condition number of a matrix (?)

UNIT F-10

Matrix operations (basic)

Assigned to: Jacopo

- sum of matrices
- product of matrices
- matrix transpose – Symmetric matrix {#mat-sym}
- matrix concatenation

1) Properties

UNIT F-11

Matrix operations (complex)

Assigned to: Jacopo

- matrix scalar product
- matrix Hadamart product
- matrix power
- matrix exponential

1) Properties

UNIT F-12

Matrix diagonalization

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

1) 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)

12.1. 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

Example: Find eigenvalues and eigenvectors:

Example: Find range and null spaces of a matrix:

UNIT F-13

Linearization

13.1. Linearization of a nonlinear system

UNIT F-14
Norms

Assigned to: Jacopo

TODO: finish writing

Other metrics can be defined to measure the "length" of a vector. Here, we report some commonly used norms. For a more in depth discussion of what consitutes a norm, and their properties:

➡ Unit F-14 - Norms,

☀ Unit F-14 - Norms

1) p-norm

Let $p \geq 1 \in \mathbb{R}$. The p-norm is defined as:

Definition 32. (p-norm)

$$\|\mathbf{v}\|_p = \left(\sum_{i=1}^{n} |v_i|^p\right)^{\frac{1}{p}}. \tag{1}$$

The p-norm is a generalization of the 2-norm ($p = 2$ in (???)) introduced above (Definition 19 - Length of a vector). The following 1-norm and ∞-norm can as well be obtained from (1) with $p = 1$ and $p \to \infty$ respectively.

2) One norm

The 1-norm is the sum of the absolute values of a vector's components. It is sometimes referred to as the *Taxicab norm*, or *Manhattan distance* as it well describes the distance a cab has to travel to get from a zero starting point to a final destination v_i on a grid.

Definition 33. (1-norm) Given a vector $\mathbf{v} \in \mathbb{R}^n$, the 1-norm is defined as:

$$\|\mathbf{v}\| = \sum_{i=1}^{n} |v_i|. \tag{2}$$

3) ∞-norm

The infinity norm measures the maximum component, in absolute value, of a vector.

Definition 34. (∞-norm)

$$\|\mathbf{v}\| = \max(|v_1|, \cdots, |v_n|). \tag{3}$$

14.1. Definition

14.2. Properties

UNIT F-15

From ODEs to LTI systems

Assigned to: Jacopo

15.1. LTI Systems

1) Properties of LTI systems

UNIT F-16
Probability basics

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

16.1. Random Variables

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

Definition 35. (Random Variable) A (either discrete or continuous) variable that can take on any value that corresponds to the feasible output of a random trial.

For example, we could model the event of flipping a fair coin with the random variable X. We write the probability that X takes HEADS as $p(X = \text{HEADS})$. The set of all possible values for the variable X is its *domain*, $\mathcal{X}$. In this case,

$$\mathcal{X} = \{\text{HEADS}, \text{TAILS}\}.$$

Since X can only take one of two values, it is a *binary* random variable. In the case of a die roll,

$$\mathcal{X} = \{1, 2, 3, 4, 5, 6\},$$

and we refer to this as a *discrete* random variable. If the output is real value or a subset of the real numbers, e.g., $\mathcal{X} = \mathbb{R}$, then we refer to X as a *continuous* random variable.

Consider once again the coin tossing event. If the coin is fair, the have $p(X = HEADS) = p(X = TAILS) = 0.5$. Here, the function $p(x)$ is called the *probability mass function* or pmf. The pmf is shown in Figure 16.1.

Figure 16.1. The pmf for a fair coin toss

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

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

In the case of continuous RVs, technically the $p(X = x)$ for any value x is zero since $\mathcal{X}$ is infinite. To deal with this, we also define another important function, the *cumulative density function*, which is given by $F(x) \triangleq p(X \leq x)$, and now we can define $f(x) \triangleq \frac{d}{dx}F(x)$. A pdf and corresponding cdf are shown in Figure 16.2 (This happens to be a Gaussian distribution, defined more precisely in Subsection 16.1.8 - The Gaussian Distribution.).

Figure 16.2. The continuous pdf and cdf

1) Joint Probabilities

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

2) Conditional Probabilities

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

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

The joint and conditional distributions are related by the following (which could be

considered a definition of the joint distribution):

$$p(x, y) = p(x|y)p(y) \tag{1}$$

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

$$p(x|y) = \frac{p(x, y)}{p(y)} \text{ if } p(y) > 0 \tag{2}$$

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

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

3) Bayes' Rule

Upon closer inspection of (1), we can see that the choice of which variable to condition upon is completely arbitrary. We can write:

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

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

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} \tag{3}$$

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

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

Check before you continue

From Wikipedia: Suppose a drug test has a 99% true positive rate and a 99% true negative rate, and that we know that exactly 0.5% of people are using the drug. Given that a person's test gives a positive result, what is the probability that this person is actually a user of the drug.

Answer: ≈ 33.2%. This answer should surprise you. It highlights the power of the *prior*.

4) Marginal Distribution

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

$$p(x) = \sum_{y} p(x, y) f(x) \qquad = \int p(x, y) dy$$

This can be thought of as projecting a higher dimensional distribution onto a lower dimensional subspace. For example, consider Figure 16.3, which shows some data plotted on a 2D scatter plot, and then the marginal histogram plots along each dimension of the data.

Figure 16.3. A 2D joint data and 2 marginal 1D histogram plots

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

5) Conditional Independence

Two RVs, X and Y may be correlated, we may be able to encapsulate the dependence through a third random variable Z. Therefore, if we know Z

Figure 16.4. A graphical representation of the conditional independence of X and Y given Z

+ comment

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

6) Moments

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

$$E[f(X)] = \sum_{\mathcal{X}} x\, f(x)$$

in the discrete case and

$$E[f(X)] = \int x\, f(x) dx$$

in the continuous case.

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

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

7) Entropy

Definition 36. The *entropy* of an RV is a scalar measure of the uncertainty about the value the RV.

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

$$H(X) = -E[\log_2 p(x)] \tag{4}$$

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

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

$$H(X) = -p \log_2 p - (1 - p) \log_2 (1 - p) \tag{5}$$

Figure 16.5. The Shannon entropy of a binary RV X

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

8) The Gaussian Distribution

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

+ comment

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

+ comment

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

The 1-D Gaussian distribution pdf is given by:

$$\mathcal{N}(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2} \tag{6}$$

where μ is called the *mean* of the distribution, and σ is called the *standard deviation*. A plot of the 1D Gaussian was previously shown in Figure 16.2.

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

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2 * \pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}} \exp[-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu})] \tag{7}$$

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

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

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

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

Exercise: derive the formula for Gaussian entropy

UNIT F-17

Kinematics

Assigned to: Jacopo

Kinematics is the study of *position*, *velocity* and *acceleration* of geometric points.

Note: A point has no dimensions. For example, the center of mass of a body is a point.

Unit F-18
Dynamics

Assigned to: Jacopo

UNIT F-19

Coordinate systems

19.1. Motivation

In order to uniquely specify a position in some space, we use some numbers, *coordinates*, in a *coordinate system*. For example, in daily life, we would use the intersection of two streets, each with a unique name in the city, to specify the location of some cafe. If you find yourself in the ocean, you might communicate your location to someone by telling them the latitude and longitude readout on your GPS device. Generally speaking, a coordinate system provides us a way to denote *any* position in an *unambiguous* way. So you can communicate any position to another person, and by using the given coordinates, that person can arrive at the same exact position in space. Note, however, different coordinates can correspond to the same point.

19.2. Definitions

Definition 37. (Coordinate system) A coordinate system is a surjective function mapping from a tuple of reals to some space S, respecting the local topology. (The local topology, which is beyond the scope of this chapter, roughly means that "nearby" points have coordinates "close" together.)

19.3. Examples

Because the ability of *naming* or specifying a point in space is so fundamentally important, we often take that coordinates given by some coordinate system, often a Cartesian coordinate system, as the name of the point.

1) 1D

Consider the real number line $\mathbb{R}$ as the space S. We can name an arbitrary point $x \in \mathbb{R}$, which happens to be a real number, by itself. So to check with our definition, any two distinct points on the real line would have two distinct coordinates. Furthermore any point has a coordinate. Let's call this coordinate system A.

One should note that the coordinate system given above is not the only possible way to name the points on a real line. We can assign coordinate $-x$ to the point $x \in \mathbb{R}$ and obtain an equally valid coordinate system B. To be more specific, given a point p with the coordinate a in the first coordinate system A, we know that in the second coordinate system B, p would have the coordinate $-a$. Therefore, there is a way to translate between coordinates in the two systems.

2) 2D plane

Consider the real plane $\mathbb{R}^2$ as the space S. This is an important space within robotics and beyond. It can be used to represent images, with each pixel having its own position, or the location of your Duckiebot in Duckietown.

2D Cartesian coordinate systems:

We can draw two perpendicular lines on the plane S. We call these two lines the x

-axis and the y-axis. We assign $(0,0)$ to the point of intersection, which we call *the origin*. Then we decide on the positive directions and units for each axis and the unit. We will use coordinates (x,y) to specify a point located x-many units in the positive x -direction and y-many units in the positive y-direction away from the origin, respectively. If we draw the axis-parallel lines with *integral* x coordinate and lines with integral y coordinate, we obtain a visualization similar to that in Figure 19.1.

When representing the location of your Duckiebot in Duckietown, you might decide to choose a corner of the map as the origin and take east as the positive x-direction and north as the positive y-direction, and 1 meter as the unit length. In this case, a Duckiebot with location $(1,-2)$ would sit at 1 meter east and 2 meters south of the designated corner of the map.

Figure 19.1. A Cartesian coordinate system in the 2D plane

Remark 8. (Image space) It is customary to put the origin of an image at the top-left corner with the x-axis being horizontal and increasing to the right and the y-axis vertical and increasing downwards. In this way, the x and y coordinates index the column and row, respectively, of a particular pixel in the image. Such a convention is observed in `OpenCV` and other software libraries.

Polar coordinate systems:

An alternative coordinate system for the plane is the polar coordinate system, where we specify a point by its direction and distance from a fixed reference point. To set up a polar coordinate system, you first decide on the *pole*, the reference point, then the *polar axis*, the reference direction. We will call the distance from the pole, the *radial coordinate* or *radius*, commonly denoted by r or ρ, and the angle from the polar axis, the *angular coordinate* or *polar angle*, commonly denoted by ϕ, φ or θ. See an example in Figure 19.2.

Figure 19.2. A polar coordinate system with pole O and polar axis L.

Note that in a polar coordinate system, a point has many equally valid names.

> **Check before you continue**
>
> Exercise to readers: provide two such points and a few of their coordinates each.

Now, consider a Cartesian coordinate system C whose origin is at the pole and its positive x-direction coincides with the polar axis. It is not hard to convert polar coordinates to Cartesian coordinates in C.

> **Check before you continue**
>
> Exercise to readers: consult Figure 19.3 and write out the conversion formulae.)

Figure 19.3. Converting from polar coordinates to Cartesian coordinates.

Given the many options, you might wonder which coordinate system to use in any given situation. The answer is a practical one. Choose the one that helps simplify the problem at hand. As a trivial example, consider the equation for a unit circle. In a Cartesian coordinate system, it would be $x^2 + y^2 = 1$ whereas in a polar coordinate system, it would be much simpler: $r = 1$.

Check before you continue

Exercise to readers: how about the equation for a straight line in polar coordinates?

3) 3D space

This is an important space since we live in a three-dimensional world. Since many of our robots operate in this same world, many robots similarly represent coordinates in three dimensions, including unmanned aerial vehicles (UAVs) and autonomous underwater vehicles (AUVs).

3D Cartesian coordinate systems:

Suppose the plane is the page or screen you are reading from, which is just a slice through the 3D space around it, we can extend a 2D Cartesian coordinate system on the plane to 3D by adding a z-axis that is perpendicular to the page, i.e., the z-, x-, and y-axis are mutually perpendicular. As done before, we need to choose a positive z-direction and there are two choices: coming out of the page or going into the page. They form the right-handed coordinate system or the left-handed coordinate system, respectively. We shall use right-handed coordinate systems unless otherwise noted. For more on handedness, see Wikipedia. Now the resulting coordinates become (x, y, z), see Figure 19.4.

Figure 19.4. A 3D Cartesian coordinate system.

Spherical coordinate systems:

Similarly, we can extend the polar coordinate systems on the page or screen to 3D by defining a *zenith direction* (upwards) perpendicular to the plane, the *polar angle* to be the angle away from zenith, and the *azimuth angle* to be the orthogonal projection of a point's angle away from the polar axis on the plane. Together, a point has coordinates (r, θ, ϕ) where θ denotes the polar angle and ϕ, the azimuth angle. See Figure 19.5.

Check before you continue

Exercise to readers: how to convert spherical coordinates to 3D Cartesian coordinates? and back?

Figure 19.5. A spherical coordinate system.

19.4. Further reading

If you find this topic interesting, there are many more coordinate systems than the ones covered here, such as the:

- cylindrical coordinate system in 3D space and
- parabolic coordinate system in 2D space.

Author: Falcon Dai

Maintainer: Falcon Dai

UNIT F-20
Reference frames

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.

20.1. Motivation

Does Earth move around the sun or does the sun move around Earth? It turns out that this is an ill-posed question until we specify a *frame of reference* for measurement. For us, observers on Earth, it appears that the sun is moving. But for an observer on the moon, both Earth and the sun are moving. In general, motions are relative and we need a reference when measuring motion.

20.2. Definition

A *reference frame*, or just *frame*, is an abstract coordinate system and the set of physical reference points that uniquely specify the coordinate system (location of the origin and orientation). As a way of specifying the reference frame, we often say object $\boldsymbol{A}$ is moving *relative to* object $\boldsymbol{B}$, instead of specifying explicitly a reference frame $\boldsymbol{F_B}$ that is attached to object $\boldsymbol{B}$.

20.3. Example

Suppose there are two cars A and B both moving at 60 miles per hour eastward. When saying this, we implicitly assume the reference frame of the ground. In the ground's reference frame, the ground itself is at rest, car A and car B are both moving at 60 miles per hour eastward. In car A's reference frame, however, car B is at rest and the ground is *moving* 60 miles per hour westward!

20.4. Translating motions in one frame to another

To simplify the following discussion, we additionally assume that we choose reference frames such that their axes are parallel. In order to translate motions between reference frames, we assume two rules.

- Relative motions are equal in magnitude and opposite in direction. If frame $\boldsymbol{R}$ is moving relative to frame $\boldsymbol{S}$ at $\mathbf{u}$, then frame $\boldsymbol{S}$ is moving at $-\mathbf{u}$ relative to $\boldsymbol{R}$.
- Motions are additive. If a frame $\boldsymbol{T}$ is moving at $\mathbf{u}$ relative to frame $\boldsymbol{S}$, and frame $\boldsymbol{S}$ is moving at $\mathbf{v}$ relative to frame $\boldsymbol{R}$, then frame $\boldsymbol{T}$ is moving at $\mathbf{u}+\mathbf{v}$ relative to frame $\boldsymbol{R}$.

Check before you continue

Exercise to readers: derive these rules from kinematics.

As a corollary to the first rule, we immediately derive that frame $\boldsymbol{R}$ is at rest relative to itself, since $\mathbf{0}$ is the only vector is also its own opposite.

Check before you continue

Exercise to readers: translate car B's motion relative to the ground to relative to car A.

Author: Falcon Dai

Maintainer: Falcon Dai

UNIT F-21

Time series

KNOWLEDGE AND ACTIVITY GRAPH

Requires: Knowledge of k:tuple

Results: Knowledge of time series.

Results: Knowledge of time series operations, such as upsampling and down-sampling.

21.1. Definition of time series

Definition 38. A *time series* with domain $\mathcal{X}$ and time domain $\mathbb{T}$ is a sequence of tuples $\langle t_k, x_k \rangle \in \mathbb{T} \times \mathcal{X}$.

21.2. Operations on time series

1) Up-sampling

TODO: to write

2) Down-sampling

TODO: to write

21.3. Further reading

Super-dense time

UNIT F-22

Transformations

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. Some familiarity with linear alegra is also helpful.

22.1. Introduction

Transformations are functions that map points to other points in space, i.e. $f : X \to X$. These maps are useful for describing motions over time. A particularly important class of transformations are *linear transformations*. These transformations can be represented by square matrices as they are *linear* and has the same domain and image.

22.2. Definitions and important examples

Please refer to the *Robotics Handbook* section 1.2, and in the context of this course, reader's goal is to attain a conceptual understanding, not necessarily knowing the exact formulae.

TODO: we need to provide links to the handbook. -AC

22.3. Applications

TODO: adapt the concepts in the context of Duckietown.

Author: Falcon Dai

Maintainer: Falcon Dai

Unit F-23
Types

Knowledge and activity graph

Results: k:types
Requires: k:sets

23.1. Types vs sets

One could define types independently of sets...

23.2. Types of functions

TODO: To write

23.3. Sum types / coproduct

TODO: to write

23.4. In Duckietown

Types of messages

23.5. Historical notes

TODO: to write

23.6. Further reading

Type theory

UNIT F-24

Computer science concepts

This unit will contain brief explanations / pointers to basic concepts of computer science.

24.1. Real-time Operating system

TODO: to write

Part G
A course in autonomy

These are the theory units.

Unit G-1
Autonomous Vehicles

Assigned to: Liam

1.1. Autonomous Vehicles in the News

These days it is hard to separate the fact from the fiction when it comes to autonomous vehicles, particularly self-driving cars. Virtually every major car manufacturer has pledged to deploy some form of self-driving technology in the next five years. In addition, there are many startups and software companies which are also known to be developing self-driving car technology.

Here's a non-exhaustive list of some of companies that are actively developing autonomous cars:

- Waymo (Alphabet/Google group)
- Tesla Autopilot project
- Uber Advanced Technologies Group
- [Cruise Automation] (the recent developments)
- nuTonomy
- Toyota Research Institute (Broader than just autonomous cars)
- Aurora Innovation
- Zoox
- Audi
- Nissan's autonomous car
- Baidu
- Apple "Project Titan" (no official details released)

1.2. Levels of Autonomy

Before even discussing any detailed notion of autonomy, we have to specify exactly what we are talking about. In the United States, the governing body is the NHTSA, and they have recently (Oct 2016) redefined the so-called "levels of autonomy" for self-driving vehicles.

In broad terms, they are as follows

- **Level 0**: the human driver does everything;
- **Level 1**: an automated system on the vehicle can sometimes assist the human driver conduct some parts of the driving task;
- **Level 2**: an automated system on the vehicle can actually conduct some parts of the driving task, while the human continues to monitor the driving environment and performs the rest of the driving task;
- **Level 3**: an automated system can both actually conduct some parts of the driving task and monitor the driving environment in some instances, but the human driver must be ready to take back control when the automated system requests;
- **Level 4**: an automated system can conduct the driving task and monitor the driving environment, and the human need not take back control, but the automated system can operate only in certain environments and under certain conditions; and

- **Level 5**: the automated system can perform all driving tasks, under all conditions that a human driver could perform them.

Unit G-2
Autonomy overview

Assigned to: Liam

This unit introduces some basic concepts ubiquitous in autonomous vehicle navigation.

2.1. Basic Building Blocks of Autonomy

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

Figure 2.1. The basic building blocks of any autonomous vehicle

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

1) Sensors

(a) Velodyne 3D Laser Scanner (b) Camera (c) Automotive Radar

(d) GPS Receiver (e) Inertial Measurement Unit

Figure 2.2. Some common sensors used for autonomous navigation

Definition 39. (Sensor) A *sensor* is a device that or mechanism that is capable of generating a measurement of some external phyiscal quantity

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

2) Raw Data Processing

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

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

We will distiguish between two fundamentally types of calibrations.

Definition 40. (Intrinsic Calibration) An *intrinsic calibration* is required to determine sensor-specific paramaters that are internal to a specific sensor.

Definition 41. (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 Unit F-20 - Reference frames

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

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

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

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

3) State Estimation

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

Definition 42. (State) The state $\boldsymbol{x}_t \in \mathcal{X}$ is a *sufficient statistic* of the environment, i.e. it contains all sufficient information required for the robot to carry out its task in that environment. This can (and usually does) include the *configuration* of the robot itself.

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

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

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

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

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

4) Navigation and Planning

Figure 2.5. An example of nested control loops

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

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

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

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

Figure 2.6. Navigation in Duckietown

5) Control

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

The most basic feedback control law (See Unit G-27 - Feedback control is called PID (for proportional, integral, derivative) which will be discussed in Unit G-28 - PID Control. For an excellent introduction to this control policy see Figure 2.7.

Figure 2.7. Controlling Self Driving Cars

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

6) Actuation

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

7) Infrastructure and Prior Information

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

Infrastructure example: Maps or GPS satellites

Structure example: Known color and location of lane markings

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

2.2. Advanced Building Blocks of Autonomy

The basic building blocks enable static navigation in Duckietown. However, many other components are necessary for more realistic scenarios.

1) Object Detection

Figure 2.8. Advanced Autonomy: Object Detection

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

2) SLAM

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

Figure 2.9. Localization in Duckietown

3) Other Advanced Topics

Other topics that will be covered include:

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

Unit G-3
Modern Robotic Systems

Assigned to: Andrea Censi

Knowledge and activity graph

Results: The many parts of a robotic system
Results: Modern robotics development
Results: Example of cloud learning, annotators in the loop scenario.

3.1. No robot is an island

A robot is only a small part of a modern robotic system.

We briefly discuss what we could consider the possible components of a robotic system.

Of course, **the robot**, which includes:

- The hardware: actuation, sensing, computation;
- The software;

The other robots with which the robot interacts. Perhaps they just need to avoid each other; perhaps they are collaborating.

The **other machines**. For example, a battery dock with that the robot can use to recharge

The **infrastructure**, including the network and off-board storage and computation resources.

The **people**, including:

- The supervisors;
- The safety operator;
- The customers;
- ...

3.2. Development of modern robotic systems

While robotic systems have existed for decades, modern autonomous robotic systems are developed in a different way than traditional robotics/automation projects.

The classical model of development consists of:

- design;
- product development;
- integration (by “system integrators”);
- installation;
- support.

These are instead the characteristics of modern robotics development model:

- continuous feedback from the users

- learning from data, acquired by the robot
- continuous integration
- incremental updates
- agile development

3.3. Example of a typical data processing pipeline

Variations on the following idea are what is typically implemented by autonomous vehicles developers for object detection.

The problem is to implement a machine-learning system that learns to perform an object detection task based on supervised learning.

Training and validation happen on large datasets that are continuously updated with new data coming from the cars, and new annotations coming from annotators.

1) The cloud as a component

We can consider "the cloud" as a component of a modern robotic system. The cloud can be modeled as a component that provides:

- Essentially "infinite" computation;
- Essentially "infinite" storage;
- Very large latency.

Because of the latency, it is not possible for real-time robotics applications to run completely on the cloud.

Therefore, the cloud is much more useful for tasks like the following:

- running simulations;
- training machine learning models;
- running regression tests.

As of 2017, the largest cloud-computing services are the ones offered by Amazon (AWS), Microsoft (Azure), Google (Google Cloud).

2) The annotators as components

For supervised learning tasks, one needs to have large annotations databases.

The idea of using human annotators as a "software service" was first deployed on a large scale by Amazon with the "Mechanical Turk" project.

Nowadays, there exist companies that are specialized in providing annotations for AI tasks.

Examples of annotation services are:

- MightyAI: "Training Data as a Service"
- Samasource

3) Putting it all together in feedback

Now that we have introduced the components, we will see how one can put everything together in a system (Figure 3.1).

Figure 3.1. Example of modern data-based pipeline

This system works as follows:

1. The robots collect data during normal operation.
2. The data becomes part of a large cloud-base storage.
3. The data is divided in training and validation data.
4. Continuously, a new model is trained based on the latest data.
5. The new candidate model is evaluated using regression tests; the goal is to outperform the previous model.
6. The new models are broadcast to the robots.
7. The regression tests also look for which new data is most useful to annotate, and this information is passed to the annotators, who create more annotations.
8. The regression tests also look for which new data would be interesting to collect, and this is done by a dedicated car.

3.4. Vignettes from an optimistic future

In the near future, it might be that the design of robotic systems might become even more complicated. For example, it might be that blockchain technologies will allow machines to trade between them.

example

A self-driving car realizes it is too dirty; by itself, it finds a robotic car-wash, and together they agree on the time and the price, and the car pays by itself using Bitcoin.

3.5. Take-away points

- The robot is but a small part of a robotic system.
- Development methods have changed recently: data is very important, as well as delocalized computation.

3.6. Further reading

- One of the cloud robotics papers XXX
- Mechanical Turk XXX
- The Robot Design Game
- A Blockchain tutorial XXX

Unit G-4
System architectures basics

Assigned to: Andrea Censi

Knowledge and activity graph

Results: Physical and logical architectures.
Results: Deployment as mapping a physical architecture onto the logical.
Requires: Basic graph concepts.
Requires: Basic operating systems concepts.

4.1. Logical and physical architecture

When we design a robotic system, or any cyber-physical system, we distinguish between what we shall call "logical architecture" and the "physical architecture".

The logical architecture describes how the functionality is divided in abstract modules, and what these modules communicate.

The physical architecture describes how the modules are instantiated. For example, this includes the information of how many processors are used, and which processor runs which routine.

4.2. Logical architecture

The logical architecture is independent of the implementation (the hardware, the language.)

The logical architecture describes:

- The system decomposition in components
- The data flow: who tells whom what
- How knows what: other sources of information, such as priors.
- How information is represented

A logical architecture would also describe what are the representations used. This is explored more fully in the unit about representations.

4.3. Actor model

TODO: Brief discussion of actor model

4.4. Physical architecture

TODO: Processors

TODO: Buses / networks

TODO: Middleware

TODO: Orchestrators

TODO: In ROS that is the `roscore` program.

4.5. Mapping logical architecture onto physical

For deployment, one must choose how the logical architecture is mapped on the physical architecture.

This can be seen as a graph mapping problem.

One can define a computation graph as a graph where nodes are algorithms and edges are events.

A resource graph is a graph where nodes are processors and edges are communication channels.

Given a computation graph and a resource graph one must choose where to put each node in the computation graph in the resource graph.

> **Remark:** More formally, the assignment is called a graph homomorphism.

4.6. Example in Duckietown

You will see a concrete example of different ways to map software components on logical architectures in one of the first exercises.

Duckietown uses ROS. In ROS, the components are called *nodes*. In ROS, the granularity is at the level of hosts rather than processors. In regular vanilla Linux, the kernel decides which physical processor executes which process at any time.

In ROS, the assignment of nodes to processors happens using a *launch file*.

By modifying the launch file we can choose the layout of the computation.

Typically you will encounter three ways to deploy a graph:

1. **Running everything on the robot.** This is the regular "autonomous" mode.
2. **Running everything on the robot, but orchestrating from the laptop**. In this case, the `roscore` program runs on a laptop, and the other components on the robot.
3. **Running heavy computation on the laptop**. In this mode, the heavy computation processes run on the laptop, while the actuation and sensing drivers run on the robot.

4.7. Take-away points

- The logical architecture describes the system decomposition, independent of the implementation.
- The physical architecture describes how is the computation physically realized.
- There are multiple ways to map a *computation graph* onto a *resource graph*. This is something that is immediately useful to understand for rapid development.

4.8. Further reading

- E. A. Lee and S. A. Seshia, Embedded Systems – A Cyber-Physical Systems Ap-

proach, MIT Press, Second Edition, 2017.

Unit G-5
Autonomy architectures

Assigned to: Andrea Censi

5.1. The state of the art in autonomy architectures

Fundamentally, we do not know how to build good robotic architectures.?

Not a smooth evolution from previous practices

industrial robot -> self-driving car: a big gap?

5.2. Further reading

- Braitenberg vehicles - Online version in German, on Springer
- 4D RCS architecture
- How the body shapes the way we think.

5.3. Figures

Figure 5.1. Add caption here

Figure 5.2. Add caption here

Figure 5.3. Add caption here

Figure 5.4. Add caption here

Figure 5.5. Add caption here

Figure 5.6. Add caption here

Figure 5.7. Add caption here

Figure 5.8. Add caption here

Unit G-6
Representations

Assigned to: Matt

Knowledge and activity graph

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 following preliminary chapters.

- Coordinate systems
- Reference frames
- Coordinate transformations.

Robots are embodied autonomous agents that interact with a physical environment.

As you read through this book, you will find that shared representations of the agent (i.e., the robot) and the environment in which it operates are fundamental to a robot's ability to sense, plan, and act—the capabilities that are key to making a robot a robot.

The *state* $\boldsymbol{x}_t \in \mathcal{X}$ is a representation that 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. Of course, "predict the future" is vague, and we can not expect the state to include knowledge to predict *everything* about the future, but rather what is relevant in the context of the task.

Definition 43. The state $\boldsymbol{x}_t \in \mathcal{X}$ is a representation of the robot and the environment that is sufficient to predict the future in the context of the task being performed.

+ comment

I understand that explicitly referencing the task as relevant to the notion of state is odd, but I want to convey that the representation that we choose for the state does depend upon the task.

Now, let's be a bit more formal with regards to what we mean by a state being "sufficient" to predict the future. We are specifically referring to a state that exhibits the *Markov property*, i.e., that it provides a complete summary of the past (again, in the context of the task). Mathematically, we say that a state is *Markov* if the future state is independent of the past given the present state (technically, this corresponds to first-order Markov):

$$p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{a}_t, \boldsymbol{x}_{t-1}, \boldsymbol{a}_{t-1}, \ldots, \boldsymbol{x}_0, \boldsymbol{a}_0) = p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{a}_t)$$

A state exhibits the Markov property if and only if the above holds.

+ comment

I usually say that "state" is all that is needed to predict the future. The agent

only needs to keep track of a smaller thing than the state to act optimally. There isn't a good name to use; but it should be distinct from "state".

+ comment

I think that this oversimplifies things. What is it about the future that is being predicted? Certainly, the states used by autonomous systems aren't sufficient to predict everything about the future (e.g., a self-driving car can't predict whether a street light is going to come on, but that probably doesn't matter).

Knowledge about the robot and its environment is often extracted from the robot's multimodal sensor streams, such as wheel encoders and cameras. Consequently, one might choose 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 and has received renewed attention of-late, notably in the context of deep learning approaches to visuomotor policy learning.

However, while measurement history is a sufficient representation of the robot and its operating environment, it serves as a challenging definition of state.

First, raw measurements are redundant, both within a single observation and across successive measurements. For example, one doesn't need to reason over all pixels in an image, let alone all pixels within successive images to understand the location of a street sign.

Second, raw measurements contain a large amount of information that is not necessary for a given task. For example, the pixel intensities that capture the complex diffusion light due to clouds convey information that is not useful for self-driving cars. Requiring algorithms that deal with state to reason over these pixels would be unnecessarily burdensome.

Third, measurement history is very inefficient: its size grows linearly with time as the robot makes new observations, 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 few parameters as possible, subject to the state being Markov and the constraints imposed by the task.

Metric spaces (namely, Euclidean spaces) constitute the most commonly adopted state space in robotics, be it through the use of feature-based parameterizations of the state or gridmap representation. However, it is not uncommon to define the state of the robot and its environment in terms of a topological space or a hybrid metric-topological space.

Importantly, the exteroceptive and proprioceptive sensor measurements from which the robot's perception algorithms infer the state are noisy, the models that describe the motion of the robot and environment are error-prone, and many aspects of the state are not directly observable (e.g., your Duckiebot may not be able to "see" some of the other Duckiebots on the road due to occlusions or the cameras limited field-of-view). As a result, robots must reason over probabilistic models of the state, commonly referred to as the *belief*, in order to effectively mitigate this uncertainty.

6.1. Robot Representations

TODO: Discuss conventions

The state of the robot typically includes its *pose* $\boldsymbol{q_t}$, which specifies the position and orientation of a coordinate frame affixed to the robot relative to a fixed global coordinate frame commonly referred to as the "world frame". The pose then defines the rigid-body rigid-body transformation between the two reference frames.

For rigid-body robots that operate in a plane ($\mathbb{R}^2$), the pose $\boldsymbol{q} \in \mathbf{SE(2)}$ consists of the Cartesian coordinates $(\boldsymbol{x}, \boldsymbol{y})$ that specify the robot's position and the angle $\boldsymbol{\theta}$ that defines the robot's orientation (yaw). For robots in $\mathbb{R}^3$, including some ground platforms, aerial vehicles, and underwater vehicles, the pose $\boldsymbol{q} \in \mathbf{SE(3)}$ consists of three Cartesian coordinates $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ that encode the robot's position, and three Euler angles $\boldsymbol{\phi}, \boldsymbol{\theta}, \boldsymbol{\psi}$ that specify the robot's orientation.

Figure 6.1. The pose $\pose_t$ of a robot operating in a two-dimensional world consists of the robots (x,y) position and orientation θ relative to a fixed reference frame.

In addition to the robot's pose, it is often useful to include its linear and angular velocities as elements of the state, resulting in an additional set of three or six parameters for robots that operate in two dimensions and three dimensions, respectively.

TODO: Discuss specific robot state representation for Duckietown.

6.2. Environment Representations

The two most commonly used metric representations of the robot's environment are occupancy grid models and feature-based representations.

Occupancy grid representations discretize the world into a set of grid cells. Associated with each cell are its Cartesian coordinates in a global reference frame, $(\boldsymbol{x}, \boldsymbol{y})$ in $\mathbb{R}^2$ and $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ in $\mathbb{R}^3$, and a binary label that indicates whether the cell is occupied, where a value of one denotes that the cell is occupied.

+ comment
pictures for these two? -AC

Feature-based models constitute the second commonly used environment representation. Feature-based representations model the environment as a collection of landmarks, and parameterize each in terms of the landmark's position ($(\boldsymbol{x}, \boldsymbol{y})$ or $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ in $\mathbb{R}^2$ and $\mathbb{R}^3$, respectively) and possibly its orientation.

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

TODO: Discuss specific environment representation for Duckietown.

Author: Matthew Walter

Maintainer: Matthew Walter

Unit G-7
Modern signal processing

Assigned to: Andrea Censi

Knowledge and activity graph

Requires: Understanding of time series. See: Unit F-21 - Time series.

Results: Understanding of the basic concepts of event-based processing and why it is different from periodic processing.

Results: Understanding of latency, frequency.

7.1. Event-based processing

The response to these challenges claims the reformulation of answers to fundamental questions referred to discrete-time systems: "when to sample," "when to update control action," or "when to transmit information." XXX

One of the generic answers to these questions is covered by adoption of the event-based paradigm that represents a model of calls for resources only if it is really necessary in system operation. In event-based systems, any activity is triggered as a response to events defined usually as a significant change of the state of relevance. Most "man-made" products, such as computer networks, complex software architectures, and production processes are natural, discrete event systems as all activity in these systems is driven by asynchronous occurrences of events. XXX

7.2. Periodic vs event-based processing

In standard signal processing, the data is assumed to arrive periodically with a certain fixed interval; in robotics, it is common to work with streams of data that arrive irregularly.

As per Definition 38, a time series is a sequence $\langle t_k, x_k \rangle \in \mathbb{T} \times \mathcal{X}$, where $\mathbb{T}$ is time and $\mathcal{X}$ is the domain in which the signals take values.

If the time series is periodic, it means that

$$t_k = t_0 + \Delta t_k.$$

Therefore, periodic processing is a special case of event-based processing.

7.3. Why working with events

How

1) Sensor-driven processing

The sensor tells the controller when there is interesting data to process.

2) Task-driven processing

The controller tells the sensor when to send data.

7.4. Definition of message statistics

1) Latency

TODO: to write

2) Frequency

TODO: to write

3) Jitter

TODO: to write

4) Signal reliability

TODO: to write

Some events might be lost, because the network loses the packet.

7.5. On the independence of latency and frequency

Latency and frequency are to be considered two completely independent quantities. Here's

7.6. Design considerations and trade-offs

Should you structure your application with periodic processing, or event-based processing? Here's a few things to keep in mind.

1) Need for real-time system

To have truly periodic processing, you need to have an operating system that is "real-time".

➡ Section 24.1 - Real-time Operating system

A real-time operating system will be able to schedule processing of data at given intervals.

2) Periodic processing is easier to analyze

Periodic processing is easier to analyze from the theoretical point of view.

3) Packet losses vs latency

There are situations where sometimes you prefer to lose packets.

Assumes: k:udp, k:tcp

4) Latency

➡ Latency numbers every programmer should know

7.7. Further reading

- A reference on all things event-based (control and signal processing) [23]
- A simple discussion of event-based control by K. J. Åström [24]

For a couple of different possible models for event-based processing:

- Synchronous Data Flow [25], in which each actor consumes a fixed amount of messages on each port.
- Delta Data Flow [26]

UNIT G-8

Middleware

Assigned to: Andrea

KNOWLEDGE AND ACTIVITY GRAPH

Results: Knowledge of middleware, marshalling, service discovery.

8.1. Why using middleware

- Reusable code
- Abstraction

Assumes: k:code-reusability, k:code-abstraction, k:code-encapsulation.

8.2. What middleware provides

1) Marshalling

Also known as “serialization”. In Python, this is called “pickling”.

2) Service discovery

Service discovery: do I know who to contact

Pinging `.local` addresses

3) Scheduling

Deciding when to do some processing

8.3. A few alternatives of robotics middleware

- ROS
- DDS
- LCM

8.4. Trade-offs and design considerations

1) Starting curve vs long-term productivity

TODO: to write

2) Performance vs convenience

TODO: to write

UNIT G-9

Modularization and Contracts

Assigned to: Andrea Censi

9.1. Monolithic vs modular design

Why do we want to separate things into modules?

- Now we can divide the work among different people.

Now we need interfaces

9.2. Contracts

We call "contracts" the guarantees that modules give to each other.

We organize the discussion as follows:

- API: Types and signatures
- Timing: Latency, frequency, availability
- Resources: Computation and memory
- Semantics
- Quality: Reliability / probability

9.3. Level 0: API (Types and signatures)

TODO: to write

9.4. Level 1: Timing: latency, frequency, availability

TODO: to write

9.5. Level 2: Resources: Computation and memory

TODO: to write

9.6. Level 3: Semantics

TODO: to write

e.g. reference frames

9.7. Level 4: Quality

Reliability, probability

UNIT G-10

Configuration management

Assigned to: Andrea

KNOWLEDGE AND ACTIVITY GRAPH

Results: How to deal with configuration effectively.

10.1. Motivation

1) Aside: Python

We use Python examples but the general concepts are applicable to any programming languages. Some of the idioms here are good for dynamic object-oriented languages.

10.2. Stages of configuration management: from clueless to pro

We will look at the typical story of how one deals with parameters.

TODO: summary

10.3. The primordial stage (clueless developer)

At the beginning you have:

```
def f(x):
    return x * 10 + 1
```

Then it becomes:

```
def f(x):
    # return x * 10 + 1
    return x * 11 + 1
```

10.4. Recognition that parameters are important

The next step is making sure that the parameters are extracted from the code

After this step, the code looks much better:

```
def f(x):
    a = 10
    b = 1
    y = x * a + b
    return y
```

Why is it better?

- It separates the "business logic" from the parameters.
- It gives meaningful name to the parameters

1) Aside: legibility of code

Note that it is not true that having fewer lines of code means the code is better! Legibility first.

> Code is for humans to read, and only tangentially for machines to interpret. -???

➡ Python manifesto

10.5. Parameters in interfaces

The next step is recognizing the parameters in the interface of the function.

In this case, we might write something like this:

```
def f(x, a=10, b=1):
    y = x * a + b
    return y
```

The next step is encapsulation of the object.

TODO: parameters and defaults

10.6. Encapsulation of functionality objects

For example, consider an image resizing function:

```
def resize(y, w, h):
    y = cv2.imresize(x, (w, h))
    return y
```

Probably the better way is the following. We create an `ImageResizer` class that is parametrized.

```python
class ImageResizer():

    def __init__(self, w, h):
        self.w, self.h = w, h

    def __call__(self, x):
        y = cv2.imresize(x, (self.w, self.h))
        return y
```

Now we can call this object as follows:

```
image_filter = ImageResizer(w=200,h=320)

x = ...
y = image_filter(x)
```

Notice that now we have abstracted the interface from the implementation: after the object is created, we can call it an "image filter", with the implication that it is "something that given an image into another".

TODO: For example, we can now write a generic `apply_filter` function:

1) In Duckietown

TODO: In Duckietown...

10.7. Convenient interfaces

Now, we still have the problem of how the user specified these parameters. We need some sort of glue that goes from the user interface, which might be the command line interface, a graphical interface, or configuration files.

It makes sense that this functionality is implemented consistently across the system.

10.8. Abstracting the configuration

The next step is understanding that a user never wants to deal with single parameters, but rather we need to give names to entire configurations.

For example, we wight want to say something like:

```
image_resizer_large:
    w: 640
    h: 480

image_resizer_small:
    w: 320
    h: 320
```

From the point of view of the user, it is easy to experiment.

From the point of view of the developer, it promotes an approach in which one writes more generic code.

If there

```
apply_filters:
    filter_names:
    - image_resizer_large
    - increase_contrast
    - crop_bottom
```

For example, suppose that there is a magic function

```
def instance(name):
    """ Returns an instantiation of the named object """
    ...
```

Then, one could implement a filter combinations like in the following code. The initialization parameter is the names of the filters, which are then instantiated and stored. The `__call__()` function calls all the filters in the that where defined

```
class ApplyFilters():
    def __init__(self, filter_names):
        self.filters = map(instance, filter_names)

    def __call__(self, image):
        for f in self.filters:
            image = f(image)
        return image
```

10.9. Duckietown solution

We now look at the Duckietown solution for the previous problems.

The code is in EasyAlgo.

10.10. Beyong: Customization, History tracking

UNIT G-11
Duckiebot modeling

Assigned to: Jacopo

TODO: put prettier figures

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

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

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

Different methods can be followed to obtain the Duckiebot model, namely the Lagrangian or Newton-Euler, we choose to describe the latter as it arguably provides a clearer physical insight. Showing the equivalence of these formulations is an interesting exercise, and the reader may refer to [27], from which this chapter is mostly taken, for detailed insight.

KNOWLEDGE AND ACTIVITY GRAPH

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

Requires: k:intro-kinematics

Requires: k:intro-dynamics

Suggested: k:intro-ODEs-to-LTIsys

Results: k:diff-drive-robot-model

11.1. Preliminaries

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

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

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

Note: The robot is assumed to be a rigid body, symmetric, and x_r coincides with ax-

is of symmetry.

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

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

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

$$\mathbf{q}^{\mathbf{I}} = \begin{pmatrix} x_A \\ y_A \\ \theta \end{pmatrix},$$

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

$$\mathbf{X}^{\mathbf{I}} = \mathbf{R}(\theta)\mathbf{X}^{\mathbf{r}}. \tag{5}$$

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

$$\mathbf{R}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{6}$$

Note: Remember that the orthogonality condition implies that $\mathbf{R}^T(\theta)\mathbf{R}(\theta) = \mathbf{R}(\theta)\mathbf{R}^T(\theta) = \mathbf{I}$, hence:

$$\mathbf{R}^T(\theta) = \mathbf{R}^{-1}(\theta) \tag{1}$$

Note: $\mathbf{R}(\theta)$ is not a function of time, but only of the orientation. Hence, by denoting time derivatives as $\dot{(\cdot)}$ we can obtain the relation between velocities in the two reference systems:

$$\dot{\mathbf{X}}^{\mathbf{I}} = \mathbf{R}(\theta)\dot{\mathbf{X}}^{\mathbf{r}}. \tag{7}$$

Figure 11.1 summarizes the notations introduced.

Figure 11.1. Relevant notations for modeling a differential drive robot

11.2. Kinematics

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

1) Differential drive robot kinematic constraints

The kinematic constraints are derived from two assumptions:

- *No lateral slipping motion*: the robot cannot move sideways, but only in the direction of motion, i.e., its lateral velocity in the robot frame is zero, i.e.:

$$\dot{y}_A^r = 0. \tag{2}$$

By inverting (5), this constraint can be expressed through the inertial frame variables, yielding:

$$\dot{y}_A \cos\theta - \dot{x}_A \sin\theta = 0. \tag{3}$$

- *Pure rolling*: the wheels never slips or skids (Figure 11.2). Hence, letting $\dot{\varphi}_l, \dot{\varphi}_r$ be the angular velocities of the left and right wheels respectively, the velocity of the ground contact point P is given by:

Figure 11.2. Pure rolling (no slipping) kinematic constraint

$$\begin{cases} v^r_{P,r} &= R\dot{\varphi}_r \\ v^r_{P,l} &= R\dot{\varphi}_l \end{cases} \tag{8}$$

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

$$\begin{cases} \dot{x}_{P,r} &= \dot{x}_A + L\dot{\theta}\cos\theta \\ \dot{y}_{P,r} &= \dot{y}_A + L\dot{\theta}\sin\theta \end{cases}, \tag{9}$$

and

$$\begin{cases} \dot{x}_{P,l} &= \dot{x}_A + L\dot{\theta}\cos\theta \\ \dot{y}_{P,l} &= \dot{y}_A + L\dot{\theta}\sin\theta \end{cases}. \tag{10}$$

By recalling (1) and (2), the expression of left and right wheel velocities in the robot frame can be summarized in the *pure rolling constraint* equation:

$$\begin{cases} \dot{x}_{P,r}\cos\theta + \dot{y}_{P,r}\sin\theta &= R\dot{\varphi}_r \\ \dot{x}_{P,l}\cos\theta + \dot{y}_{P,l}\sin\theta &= R\dot{\varphi}_l \end{cases}. \tag{11}$$

11.3. Differential drive robot kinematic model

In a differential drive robot, controlling the wheels at different speeds generates a rolling motion of rate $\omega = \dot{\theta}$. In a rotating field there always is a fixed point, the *center of instantaneous curvature* (ICC), and all points at distance $\boldsymbol{d}$ from it will have a velocity given by $\omega\boldsymbol{d}$, and direction orthogonal to that of the line connecting the ICC and the wheels (i.e., the *axle*). Therefore, by looking at Figure 11.3, we can write:

Figure 11.3. By controlling the rotation rates of the wheel independently, a differential drive robot can

make turns. Figure adapted from [](#bib:Dudek10).

TODO: change labels in pic to match previously used conventions. R in figure is d in this text, and L in text is l/2 in pic.

$$\begin{cases} \dot{\theta}(d-L) &= v_l \\ \dot{\theta}(d+L) &= v_r \end{cases}, \tag{12}$$

from which:

$$\begin{cases} d &= L\frac{v_r+v_l}{v_r-v_l} \\ \dot{\theta} &= \frac{v_r-v_l}{2L} \end{cases}. \tag{13}$$

A few observations stem from (13):

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

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

By recalling the *no lateral slipping motion* (2) hypothesis and the *pure rolling* constraint (8), and noticing that the translational velocity of A in the robot frame is $\dot{x}_A^r = \dot{\theta}d$ we can write:

$$\begin{cases} \dot{x}_A^r &= R(\dot{\varphi}_R + \dot{\varphi}_L)/2 \\ \dot{y}_A^r &= 0 \\ \dot{\theta} &= \omega = R(\dot{\varphi}_R - \dot{\varphi}_L)/(2L) \end{cases}, \tag{14}$$

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

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

$$\begin{bmatrix} x_A^r \\ y_A^r \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{R}{2} & \frac{R}{2} \\ 0 & 0 \\ \frac{R}{2L} & -\frac{R}{2L} \end{bmatrix} \begin{bmatrix} \dot{\varphi}_R \\ \dot{\varphi}_L \end{bmatrix}. \tag{15}$$

Finally, by using (6), we can recast (15) in the inertial frame.

> **Note:** The *forward kinematics* model of a differential drive robot is given by:

$$\dot{\mathbf{q}}^I = \mathbf{R}(\theta) \begin{bmatrix} x_A^r \\ y_A^r \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{R}{2}\cos\theta & \frac{R}{2}\cos\theta \\ \frac{R}{2}\sin\theta & \frac{R}{2}\sin\theta \\ \frac{R}{2L} & -\frac{R}{2L} \end{bmatrix} \begin{bmatrix} \dot{\varphi}_R \\ \dot{\varphi}_L \end{bmatrix}. \tag{16}$$

11.4. Differential drive robot dynamic model

While kinematics studies the properties of motions of geometric (i.e., massless) points, dynamical modeling takes into account the actual material distribution of the system. Once mass comes into play, motion is the result of the equilibrium of

forces and torques. While different approaches can be used to derive these equations, namely the Lagrangian or Newtonian approaches (former based on energy considerations, latter on equilibrium of generalized forces), we choose to follow the Newtonian one here for it grants, arguably, a more explicit physical intuition of the problem. Obviously both methods lead to the same results when the same hypothesis are made.

1) Notations

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

TABLE 11.1. NOTATIONS FOR DYNAMIC MODELING OF A DIFFERENTIAL DRIVE ROBOT

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

2) Free body diagram

The next step, and definitely the most critical, is writing the free body diagram of the problem (Figure 11.4). In this analysis the only forces acting on the robot are those applied to the wheels.

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

Figure 11.4. Free body diagram of a differential drive robot

3) Rotating frames in polar coordinates: a recap

Let the center of mass of the robot be identified by the vector $\mathbf{r}(t)$, as shown in Figure 11.4). The polar coordinated allow us to express $\mathbf{r}(t)$ using the complex notation:

$$\mathbf{r}(t) = r(t)e^{j\theta(t)}. \tag{4}$$

By differentiating in time (4), and recalling the chain rule of derivatives, it is straightforward, although arguably boring to obtain the expression of radial velocity and acceleration, respectively:

$$\begin{aligned}\dot{\mathbf{r}}(t) &= \dot{r}(t)e^{j\theta(t)} + jr(t)\dot{\theta}(t)e^{j\theta(t)} \\ \ddot{\mathbf{r}}(t) &= \ddot{r}(t)e^{j\theta(t)} + 2j\dot{r}(t)\dot{\theta}(t)e^{j\theta(t)} + jr(t)\ddot{\theta}(t)e^{j\theta(t)} - r(t)\dot{\theta}^2 e^{j\theta(t)}.\end{aligned} \tag{17}$$

By simplifying and writing the lateral components explicitly, (17) becomes:

$$\begin{aligned}\dot{\mathbf{r}}(t) &= v_u(t)e^{j\theta(t)} + v_w(t)e^{j\left(\theta(t)+\frac{\pi}{2}\right)} \\ \ddot{\mathbf{r}}(t) &= a_u(t)e^{j\theta(t)} + a_w(t)e^{j\left(\theta(t)+\frac{\pi}{2}\right)},\end{aligned} \tag{18}$$

where:

$$\begin{aligned}v_u(t) &= \dot{r}(t) \\ v_w(t) &= r(t)\dot{\theta}(t) \\ a_u(t) &= \ddot{r}(t) - r(t)\dot{\theta}^2 \\ a_w(t) &= 2\dot{r}(t)\dot{\theta}(t) + r(t)\ddot{\theta}.\end{aligned} \tag{19}$$

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

4) Equilibrium of forces and moments

We derive the dynamic model by imposing the simultaneous equilibrium of forces along the longitudinal and lateral directions in the robot frame with the respective inertial forces, and of the moments around the vertical axis (coming out of the.. screen) passing through the center of mass of the robotic vehicle.

$$\begin{aligned} Ma_u(t) &= F_{u_L} + F_{u_R} \\ Ma_w(t) &= F_{w_L} - F_{w_R} \\ \ddot{\theta}(t) &= \frac{L}{J}(F_{u_R} - F_{u_L}) + \frac{c}{J}(F_{w_R} - F_{w_L}) \end{aligned} \tag{20}$$

By substituting the (19) in (20), the equilibrium of forces equations are expressed in terms of accelerations of the center of mass in the robot frame:

$$\dot{v}_u(t) = v_w\dot{\theta}(t) + \frac{F_{u_L} + F_{u_R}}{M} \tag{21}$$

$$\dot{v}_w(t) = -v_u\dot{\theta}(t) + \frac{F_{w_L} - F_{w_R}}{M} \tag{22}$$

$$\ddot{\theta}(t) = \frac{L}{J}(F_{u_R} - F_{u_L}) + \frac{c}{J}(F_{w_R} - F_{w_L}). \tag{23}$$

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

5) Imposing the kinematic constraints

Equations (20) of motion can be decoupled by imposing the kinematic constraints (2) and (8). In particular, to impose the no lateral slipping hypothesis (2), we first need to express the velocity of the center of mass of the robot in the inertial frame, then derive the velocity of point A as a function of that in C, and finally impose the lateral velocity to be zero. To do so, we first need to notice that, in the inertial frame:

$$\begin{aligned} x_C &= x_A + c\cos\theta \\ y_C &= y_A + c\sin\theta \end{aligned} \tag{24}$$

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

$$\begin{bmatrix} \dot{x}_C \\ \dot{y}_C \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} v_u \\ v_w \end{bmatrix}. \tag{25}$$

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

By using this condition in (21) and (22), and combining with (23), we obtain the dynamical equations of a differential drive robot under the aforementioned non holonomic constraints:

$$\begin{aligned} \dot{v}_u(t) &= c\dot{\theta}^2 + \frac{1}{M}(F_{u_L} + F_{u_R}) \\ \ddot{\theta} &= \frac{L}{Mc^2 + J}(F_{u_R} - F_{u_L}) + \frac{Mcv_u}{Mc^2 + J}\dot{\theta}. \end{aligned} \tag{26}$$

Although (26) describes the dynamics of the robot, the input forces are difficult to measure. As it will be clearer from the next section where a model of the DC motor will be provided, it is more practical to consider the torques $(\tau_R, \tau_L) = (RF_{u_R}, RF_{u_L})$

in (26) as inputs to this system. By implementing this consideration and rearranging in matrix form:

$$\begin{bmatrix} M & 0 \\ 0 & Mc^2 + J \end{bmatrix} \begin{bmatrix} \dot{v}_u \\ \ddot{\theta} \end{bmatrix} + \begin{bmatrix} 0 & -Mc\dot{\theta} \\ Mc\dot{\theta} & 0 \end{bmatrix} \begin{bmatrix} v_u \\ \dot{\theta} \end{bmatrix} = \frac{1}{R} \begin{bmatrix} 1 & 1 \\ L & -L \end{bmatrix} \begin{bmatrix} \tau_R \\ \tau_L \end{bmatrix}. \tag{27}$$

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

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

11.5. DC motor dynamic model

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

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

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

Figure 11.5 shows a diagram of a typical DC motor.

Figure 11.5. Diagram of a DC motor

Having a relation between the applied voltage and torque, in addition to the dy-

namic and kinematic models of a differential drive robot, allows us to determine all possible state variables of interest.

> **Note:** torque disturbances acting on the wheels, such as the effects of friction, can be modeled as additive terms (of sign opposite to $\boldsymbol{\tau}$) in the DC motor equations.

11.6. Conclusions

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

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

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

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

UNIT G-12

Odometry Calibration

Assigned to: Jacopo

UNIT G-13
Computer vision basics

Assigned to: Matt

For the interested reader, there are several excellent books that discuss fundamental topics in computer vision. These include (in no particular order), Computer Vision: Algorithims and Applications (available online), Computer Vision: A Modern Approach and Multiple View Geometry in Computer Vision.

1) Computer vision: What and why?

Topics:

- **Objective**: Discover what is present in the world, where things are, and what actions are taking place from image sequences. In the context of robotics, this corresponds to learning a (probabilistic) world model that encodes the robot's environment, i.e., building a representation of the environment.
- History of computer vision
  - Summer Vision Project
  - OCR (license plates, signs, checks, mail, etc.)
  - Face detection
  - Structure from motion
  - Dense reconstruction
  - Augmented reality
  - Applications to self-driving cars and beyond

UNIT G-14

Camera geometry

Assigned to: Matt

Note: We could break this up into separate sub-sections for (prospective) projection and lenses.

Topics:

- Pinhole cameras (tie into eclipse viewing, accidental cameras)
- Geometry of projection: equations of projection; vanishing points; weak perspective; orthographic projection;
- Review of 3D transformations (translation and rotation); homogeneous transformations
- Perspective projection in homogeneous coordinates

1) Lenses

- Lenses: Why?; first-order optics (thin lens); thick lens
- First-order optics (thin lens)

Unit G-15
Camera calibration

Assigned to: Matt

Note: The discussion of intrinsic and extrinsic models could be moved up to the geometry subsection

Topics

- Why calibration?
- Intrinsic parameters: idealized perspective projection; pixel scaling (square and non-square); offset; skew.
- Intrinsic model expressed in homogeneous coordinates
- Extrinsic parameters: translation and rotation of camera frame (non-homogeneous and homogeneous)
- Combined expression for intrinsic and extrinsic
- Calibration targets
- Calibration models

UNIT G-16

Image filtering

Assigned to: Matt

Background: Discrete-time signal processing

Note: We currently don't have a preliminary section on signal processing and it's probably sufficient to point readers to a good reference (e.g., Oppenheim and Schafer).

Topics

Note: May want to stick to linear filtering

- Motivation (example)
- Convolution: definition and properties (including differentiation)
- Linear filters: examples (sharpening and smoothing)
- Gaussian filters
  - kernels
  - properties (convolution with a Gaussian, separability)

Unit G-17
Illumination invariance

Assigned to: Matt

UNIT G-18

Line Detection

Assigned to: Matt

Topics:

- What is edge detection?
- Image derivatives
- Finite difference filters
- Image gradients
- Effects of noise and need for smoothing
- Derivative theorem of convolution
- Derivative of Gaussian filters
- Smoothing vs. derivative filters
- Line detection as an edge detection problem

UNIT G-19

Feature extraction

Assigned to: Matt

UNIT G-20

Place recognition

Assigned to: Matt

Unit G-21
Filtering 1

> Assigned to: Liam

TODO: Markov Property

TODO: Bayes Filter

TODO: Graphical representation

TODO: Kalman Filter

TODO: Extended Kalman Filter

UNIT G-22

Filtering 2

> **Assigned to:** Liam

TODO: MAP estimation

TODO: Laplace Approximation

TODO: Cholesky Decomposition?

TODO: Incrementalization - Givens Rotations - iSAM

Unit G-23
Mission planning

Assigned to: ETH

UNIT G-24

Planning in discrete domains

Assigned to: ETH

Unit G-25
Motion planning

Assigned to: ETH

UNIT G-26

RRT

Assigned to: ETH

UNIT G-27

Feedback control

Assigned to: Jacopo

UNIT G-28

PID Control

Assigned to: Jacopo

UNIT G-29

MPC Control

Assigned to: Jacopo

UNIT G-30

Object detection

Assigned to: Nick and David

UNIT G-31

Object classification

Assigned to: Nick and David

UNIT G-32

Object tracking

Assigned to: Nick and David

Unit G-33

Reacting to obstacles

Assigned to: Jacopo

UNIT G-34

Semantic segmentation

Assigned to: Nick and David

UNIT G-35

Text recognition

Assigned to: Nick

UNIT G-36

SLAM - Problem formulation

Assigned to: Liam

UNIT G-37

SLAM - Broad categories

Assigned to: Liam

Unit G-38

VINS

Assigned to: Liam

UNIT G-39

Advanced place recognition

Assigned to: Liam

UNIT G-40

Fleet level planning (placeholder)

Assigned to: ETH

UNIT G-41

Fleet level planning (placeholder)

Assigned to: ETH

Unit G-42
Bibliography

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

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

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

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

[7] Albert Einstein. Zur Elektrodynamik bewegter Körper. (German) On the electrodynamics of moving bodies. *Annalen der Physik*, 322(10):891–921, 1905. DOI

[15] Ivan Savov. Linear algebra explained in four pages. https://minireference.com/static/tutorials/linear_algebra_in_4_pages.pdf, 2017. Online; accessed 23 August 2017.

[14] Kaare Brandt Petersen and Michael Syskind Pedersen. The matrix cookbook. .pdf

[17] Matrix (mathematics). Matrix (mathematics) — Wikipedia, the free encyclopedia, 2017. Online; accessed 2-September-2017. www:

[18] Reference bib:norms not found.

[19] Athanasios Papoulis and S. Unnikrishna Pillai. *Probability, Random Variables and Stochastic Processes*. McGraw Hill, fourth edition, 2002.

[20] David J. C. MacKay. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, New York, NY, USA, 2002.

[21] H. Choset, W. Burgard, S. Hutchinson, G. Kantor, L. E. Kavraki, K. Lynch, and S. Thrun. *Principles of robot motion: Theory, agorithms, and implementation*. MIT Press, June 2005.

[22] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, New York, NY, USA, 2006.

[23] M. Miskowicz. *Event-Based Control and Signal Processing*. Embedded Systems. CRC Press, 2015. http

[24] Karl J. Aström. *Event Based Control*, pages 127–147. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. DOI http

[25] Edward A Lee and David G Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987.

[26] Rajit Manohar and K Mani Chandy. Delta-dataflow networks for event stream processing. pages 1–6, June 2010.

[27] Rached Dhaouadi and A. Abu Hatab. Dynamic modelling of differential-drive mobile robots using

lagrange and newton-euler methodologies: A unified framework. *Advances in Robotics & Automation*, 2(2):1–7, 2013.

[28] Richard Szeliski. *Computer Vision: Algorithms an Applications*. Springer, 2010. http

[29] David Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*. Prentce Hall, Upper Saddle River, NJ, 2011.

[30] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004.

Part H
Exercises

These exercises can guide you from the status of a novice coder to experienced roboticist.

Unit H-1
Exercise: Basic image operations

Assigned to: Andrea Daniele

1.1. Skills learned

- Accessing command line arguments.
- Reading and writing files.
- Working with pixel-based image representations.
- OpenCV and `duckietown_utils` API for reading/writing images.

1.2. Instructions

Create an implementation of the program `dt-image-flip0`, specified below.

If this exercise is too easy for you, skip to Unit H-2 - Exercise: Basic image operations, adult version.

1.3. Specification of `dt-image-flip0`

The program `dt-image-flip0` takes as an argument a JPG file with extension `.jpg`:

```
$ dt-image-flip0 file.jpg
```

and creates a file called `file.flipped.jpg` that is flipped around the horizontal axis.

(a) The original picture. (b) The flipped output

Figure 1.1

Example input-output for the program `dt-image-flip`.

1.4. Useful APIs

1) Load image from file

The OpenCV library provides a utility function called `imread` that loads an image from a file.

2) Flip an image

+ comment

This is the kind of thing that they need to figure out how to do with pixels. -AC

3) Write an image to a file

The `duckietown_utils` package provides the utility function `write_image_as_jpg()` that writes an image to a JPEG file.

Unit H-2
Exercise: Basic image operations, adult version

Assigned to: Andrea Daniele

2.1. Skills learned

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

2.2. Instructions

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

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

2.3. `dt-image-flip` specification

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

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

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

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

If the file exists, then the script must create:

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

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

(a) The original picture.

(b) The output `flip.jpg`

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

Figure 2.1. Example input-output for the program `image-ops`.

2.4. Useful APIs

1) Images side-by-side

+ comment

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

An image loaded using the OpenCV function `imread` is stored in memory as a NumPy array. For example, the image shown above (Figure 2.1a - The original picture.) will be represented in memory as a NumPy array with shape `(96, 96, 3)`. The first dimension indicates the number of pixels along the `Y-axis`, the second indicates the number of pixels along the `X-axis` and the third is known as *number of channels* (e.g., Red, Green, and Blue).

+ comment

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

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

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

We provide a script called `image-ops-tester` that can be used to make sure that you wrote a conforming `dt-image-flip`.

Use it as follows:

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

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

`image-ops-tester` will return 0 if the program exists and conforms to the specification (Section 2.3 - `dt-image-flip` specification).

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

Bottom line

Things that are not tested are broken.

Unit H-3
Exercise: Simple data analysis from a bag

Assigned to: Andrea Daniele

3.1. Skills learned

- Reading Bag files.
- Statistics functions (mean, median) in Numpy.
- Use YAML format.

3.2. Instructions

Create an implementation of `dt-bag-analyze` according to the specification below.

3.3. Specification for `dt-bag-analyze`

Create a program that summarizes the statistics of data in a bag file.

```
$ dt-bag-analyze bag file
```

Compute, for each topic in the bag:

- The total number of messages.
- The minimum, maximum, average, and median interval between successive messages, represented in seconds.

Print out the statistics using the YAML format. Example output:

```
$ dt-bag-analyze bag file
''topic name'':
    num_messages: value
    period:
        min: value
        max: value
        average: value
        median: value
```

3.4. Useful APIs

1) Read a ROS bag

A bag is a file format in ROS for storing ROS message data. The package `rosbag` defines the class `Bag` that provides all the methods needed to serialize messages to and from a single file on disk using the bag format.

2) Time in ROS

In ROS the time is stored as an object of type `rostime.Time`. An object `t`, instance of `rostime.Time`, represents a time instant as the number of seconds since epoch (stored in `t.secs`) and the number of nanoseconds since `t.secs` (stored in `t.nsecs`). The utility function `t.to_sec()` returns the time (in seconds) as a floating number.

3.5. Test that it works

Download the ROS bag `example_rosbag_H3.bag`. Run your program on it and compare the results:

```
$ dt-bag-analyze example_rosbag.bag
/tesla/camera_node/camera_info:
  num_messages: 653
  period:
    min: 0.01
    max: 0.05
    average: 0.03
    median: 0.03

/tesla/line_detector_node/segment_list:
  num_messages: 198
  period:
    min: 0.08
    max: 0.17
    average: 0.11
    median: 0.1

/tesla/wheels_driver_node/wheels_cmd:
  num_messages: 74
  period:
    min: 0.02
    max: 4.16
    average: 0.26
    median: 0.11
```

Unit H-4
Exercise: Bag in, bag out

Assigned to: Andrea Daniele

4.1. Skills learned

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

4.2. Instructions

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

4.3. Specification of `dt-bag-decimate`

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

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

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

4.4. Useful new APIs

1) Create a new Bag

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

For example:

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

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

2) Write message to a Bag

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

4.5. Check that it works

To check that the program works, you can compute the statistics of the data using the program `dt-bag-analyze` that you have created in Unit H-3 - Exercise: Simple data

analysis from a bag.

You should see that the statistics have changed.

Unit H-5
Exercise: Bag thumbnails

Assigned to: Andrea Daniele

5.1. Skills learned

- Reading images from images topic in a bag file.

5.2. Instructions

Write a program `dt-bag-thumbnails` as specified below.

5.3. Specification for `dt-bag-thumbnails`

The program `dt-bag-thumbnails` creates thumbnails for some image stream topic in a bag file.

The syntax is:

```
$ dt-bag-thumbnails bag topic output dir
```

This should create the files:

```
output dir/00000.jpg
output dir/00001.jpg
output dir/00002.jpg
output dir/00003.jpg
output dir/00004.jpg
...
```

where the progressive number is an incremental counter for the frames.

5.4. Test data

If you don't have a ROS bag to work on, you can download the test bag `example_ros-bag_H5.bag`. You should be able to get a total of 653 frames out of it.

5.5. Useful APIs

1) Read image from a topic

The `duckietown_utils` package provides the utility function `rgb_from_ros()` that processes a ROS message and returns the RGB image it contains (if any).

2) Color space conversion

In OpenCV, an image can be converted from one color space (e.g., BGR) to another supported color space (e.g., RGB). OpenCV provides a list of supported conversions. A `ColorConversionCode` defines a conversion between two different color spaces. An exhaustive list of color conversion codes can be found here. The conversion from a color space to another is done with the function `cv.cvtColor`.

Unit H-6
Exercise: Instagram filters

Assigned to: Andrea Daniele

6.1. Skills learned

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

6.2. Instructions

Create `dt-instagram` as specified below.

6.3. Specification for `dt-instagram`

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

The syntax to invoke the program is:

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

where:

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

The list of filters is given in Subsection 6.3.1 - List of filters.

For example, the result of the command:

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

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

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

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

1) List of filters

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

- `flip-vertical`: flips the image vertically

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

6.4. Useful new APIs

1) User defined filters

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

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

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

Unit H-7
Exercise: Bag instagram

> Assigned to: Andrea Daniele

7.1. Instructions

Create `dt-bag-instagram` as specified below.

7.2. Specification for `dt-bag-instagram`

Write a program `dt-bag-instagram` that applies a filter to a stream of images stored in a ROS bag.

The syntax to invoke the program is:

```
$ dt-bag-instagram bag in topic filters bag out
```

where:

```
- `bag in` is the input bag;
- `topic` is a string containing the topic to process;
- `filters` is a string, which is a colon-separated list of filters;
- `bag out` is the output bag.
```

7.3. Test data

If you don't have a ROS bag to work on, you can download the test bag `example_ros-bag_H5.bag`.

7.4. Useful new APIs

1) Compress an RGB image into a `sensor_msgs/CompressedImage` message

The `duckietown_utils` package provides the utility function `d8_compressed_image_from_cv_image()` that takes an RGB image, compresses it and wrap it into a `sensor_msgs/CompressedImage` ROS message.

7.5. Check that it works

Play your `bag out` ROS bag file and run the following command to make sure that your program is working.

```
$ rosrun image_view image_view image:=topic _image_transport:=compressed
```

Unit H-8
Exercise: Live Instagram

Assigned to: Andrea Daniele

8.1. Skills learned

- Live image processing

8.2. Instructions

Do the first ROS tutorial at Unit J-6 - Minimal ROS node - `pkg_name`. That tutorial is about listening to text messages and writing back text messages. Here, we apply the same principle, but to images.

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

8.3. Specification for the program `dt-live-instagram`

Create a program `dt-live-instagram` that takes a string argument:

```
$ dt-live-instagram filters
```

This program should do the following:

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

8.4. Check that it works

Run your program

```
$ dt-live-instagram filters
```

It might be a good idea to print out the topic selected by your program. Let's call it `topic`. Run the following command to make sure that your program is working.

```
$ rosrun image_view image_view image:=''topic/filters'' _image_transport:=compressed
```

UNIT H-9

Exercise: Git and conventions

TODO: move here the exercises we had last year about branching.

UNIT H-10

Exercise: Use our API for arguments

10.1. Skills learned

- Learn about the command-line API that we have in Duckietown,

10.2. Instructions

We have a useful API that makes it easy to create programs with command line arguments.

TODO: to write

10.3. Useful new API learned

- Our API for command line arguments.

UNIT H-11

Exercise: Bouncing ball

11.1. Skills learned

- Show how to visualize data on a bag.
- Programmatic generation of images.
- Timestamps generation.

11.2. Instructions

Create a program

```
$ bag-bounce --fps fps --speed speed
```

that shows a bouncing ball on the screen, as if it were a billiard

Useful new API learned

- Our API for command line arguments.

Unit H-12
Exercise: Visualizing data on image

12.1. Skills learned

- Show how to superimpose data on an image.

12.2. Instruction

Write an implementation of `bag-mark-spots`.

12.3. Specification for `bag-mark-spots`

Create a program that for each image, finds the pixels that are closest to a certain color, and creates as the output a big red, yellow white spot on them.

```
$ bag-mark-spots --input bag in --mark "[[255,0,0],255,0,0]," --size 5 --output [bag out]
```

UNIT H-13

Exercise: Make that into a node

13.1. Learned skills

- Abstracting code in interfaces that can be reused.
- Launch files.

13.2. Instructions

Abstract the analysis above in a way that the same analysis code can be run equally from a bag or on the laptop.

Make a ROS node and two launch files:

- One runs everything on the Duckiebot, and the output is visualized on the laptop.
- One runs the processing on the laptop.

Unit H-14
Exercise: Instagram with EasyAlgo interface

14.1. Learned skills

- Use of our Duckietown API for abstracting algorithms.

14.2. Instructions

TODO: Do the above using our API for filters.

We define the interface `InstagramFilter` and the EasyAlgo configuration files.

14.3. See documentation

TODO: pointer to EasyAlgo

14.4. Use in your code

Write a node using the EasyNode framework that decides which filters to run given the configuration.

TODO: Also introduce the DUCKIETOWN_CONFIG_SEQUENCE.

UNIT H-15

Milestone: Illumination invariance (anti-instagram)

TODO: Make them run our code, and also visualize what's going on

UNIT H-16

Exercise: Launch files basics

16.1. Learned skills

- Launch files

UNIT H-17

Exercise: Unit tests

17.1. Learned skills

- Write unit tests that can be integrated in our framework.

17.2. Unit tests with nosetests

- Unit tests

17.3. Unite tests with ROS tests

- Integration with ROS tests

17.4. Unite tests with comptests

UNIT H-18

Exercise: Parameters (standard ROS api)

18.1. Learned skills

- Reading parameters
- Dynamic modification of parameters

UNIT H-19

Exercise: Parameters (our API)

19.1. Learned skills

- Use Duckietown API

Unit H-20
Exercise: Place recognition abstraction

20.1. Learned skills

- ...

20.2. Instructions

We use the following interface:

```
class FeatureDescription():

    def feature(self, image):
        """ returns a "feature description" """

    def feature_compare(feature1, feature2):
        pass
```

We also provide a basic skeleton.

20.3. Try a basic feature:

Simplest feature: average color.

```
class AverageColor(FeatureDescription):

    def feature(self, image):
        return np.mean(image)

    def feature_mismatch(f1, f2)
        return np.abs(f1-f2)
```

Compute the similarity matrix for the

```
$ similarity --feature average_color --input input.bag  --output dirname
```

20.4. Bottom line

TODO: to write

UNIT H-21

Exercise: Parallel processing

21.1. Learned skills

- Do things faster in parallel.

21.2. Instructions

We introduce the support for parallel processing that we have in the APIs.

UNIT H-22

Exercise: Adding new test to integration tests

TODO: to write

22.1. Learned skills

- Do things faster in parallel.

22.2. Instructions

TODO: to write

22.3. Bottom line

TODO: to write

22.4. Milestone: Lane following

TODO: to write

UNIT H-23

Exercise: localization

TODO: to write

UNIT H-24

Exercise: Ducks in a row

TODO: to write

UNIT H-25

Exercise: Comparison of PID

TODO: to write

UNIT H-26

Exercise: RRT

TODO: to write

UNIT H-27

Exercise: Who watches the watchmen? (optional)

27.1. Skills learned

- Awareness that this is a losing game.

27.2. Instructions

A good exercise is writing `image-ops-tester` yourself.

However, we already gave you a copy of `image-ops-tester`, which you used in the previous step, so there is not much of a challenge. So, let's go one level up, and consider...

27.3. `image-ops-tester-tester` specification

Write a program `image-ops-tester-tester` that tests whether a program conforms to the specification of a `image-ops-tester` given in Section 2.5 - Testing it works with `image-ops-tester`.

The `image-ops-tester-tester` program is called as follows:

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

This must return:

- 0 if the candidate conforms to the specification;
- 1 if it doesn't;
- another error code if other errors arise.

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

We provide you with a helpful program called `image-ops-tester-tester-tester` that makes sure that a candidate script conforms to the specification of an `image-ops-tester-tester`.

Use it as follows:

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

This should return 0 if everything is ok, or different than 0 otherwise.

Bottom line

Even if things are tested, you can never be sure that the tests themselves work.

Part I
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.

UNIT I-1

Ubuntu packaging with APT

1.1. `apt install`

TODO: to write

1.2. `apt update`

TODO: to write

1) `apt dist-upgrade`

TODO: hold back packages

1.3. `apt-key`

TODO: to write

1.4. `apt-mark`

TODO: to write

1.5. `add-apt-repository`

TODO: to write

1.6. `wajig`

TODO: to write

1.7. `dpigs`

TODO: to write

UNIT I-2

GNU/Linux general notions

Assigned to: Andrea

2.1. Background reading

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

UNIT I-3

Every day Linux

3.1. `cd`

TODO: to write

3.2. `sudo`

TODO: to write

3.3. `ls`

TODO: to write

3.4. `cp`

TODO: to write

3.5. `mkdir`

TODO: to write

3.6. `touch`

TODO: to write

3.7. `reboot`

TODO: to write

3.8. `shutdown`

TODO: to write

3.9. `rm`

TODO: to write

UNIT I-4
Users

4.1. `passwd`

TODO: to write

UNIT I-5

UNIX tools

5.1. `cat`

TODO: to write

5.2. `tee`

TODO: to write

5.3. `truncate`

TODO: to write

Unit I-6
Linux disks and files

6.1. `fdisk`

TODO: to write

6.2. `mount`

TODO: to write

6.3. `umount`

TODO: to write

6.4. `losetup`

TODO: to write

6.5. `gparted`

TODO: to write

6.6. `dd`

TODO: to write

6.7. `sync`

TODO: to write

6.8. `df`

TODO: to write

6.9. How to make a partition

TODO: to write

UNIT I-7

Other administration commands

7.1. `visudo`

TODO: to write

7.2. `update-alternatives`

TODO: to write

7.3. `udevadm`

TODO: to write

7.4. `systemctl`

TODO: to write

UNIT I-8
Make

8.1. `make`

TODO: to write

UNIT I-9

Python-related tools

9.1. `virtualenv`

TODO: to write

9.2. `pip`

TODO: to write

Unit I-10
Raspberry-PI commands

10.1. raspi-config

TODO: to write

10.2. vcgencmd

TODO: to write

10.3. raspistill

TODO: to write

10.4. jstest

TODO: to write

10.5. swapon

TODO: to write

10.6. mkswap

TODO: to write

UNIT I-11

Users and permissions

11.1. `chmod`

TODO: to write

11.2. `groups`

TODO: to write

11.3. `adduser`

TODO: to write

11.4. `useradd`

TODO: to write

Unit I-12
Downloading

12.1. `curl`

TODO: to write

12.2. `wget`

TODO: to write

12.3. `sha256sum`

TODO: to write

12.4. `xz`

TODO: to write

UNIT I-13

Shells and environments

13.1. `source`

TODO: to write

13.2. `which`

TODO: to write

13.3. `export`

TODO: to write

Unit I-14
Other misc commands

14.1. pgrep

TODO: to write

14.2. npm

TODO: to write

14.3. nodejs

TODO: to write

14.4. ntpdate

TODO: to write

14.5. chsh

TODO: to write

14.6. echo

TODO: to write

14.7. sh

TODO: to write

14.8. fc-cache

TODO: to write

UNIT I-15

Linux resources usage

15.1. Measuring CPU usage using `htop`

You can use `htop` to monitor CPU usage.

```
$ sudo apt install htop
```

TODO: to write

15.2. Measuring I/O usage using `iotop`

Install using:

```
$ sudo apt install iotop
```

TODO: to write

15.3. How fast is the SD card?

➡ Section 16.1 - Testing SD Card and disk speed.

Unit I-16
SD Cards tools

16.1. 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.

16.2. How to burn an image to an SD card

Knowledge and activity graph

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

1) Finding your device name for the SD card

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

Insert the SD Card in the slot.

Run the command:

```
$ sudo fdisk -l
```

Find your device name, by looking at the sizes.

For example, the output might contain:

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

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

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

2) Unmount partitions

Before proceeding, unmount all partitions.

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

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

3) Burn the image

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

Let the image file be `image file`.

Burn the image using the command `dd`:

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

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

Note: dd comand with status=progress parameter only work for dd –version 8.24 ubuntu16.04.2

16.3. How to shrink an image

KNOWLEDGE AND ACTIVITY GRAPH

Requires: An image file to burn.

Requires: 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.

It might be useful to compress it, before distribution:

```
$ xz image file
```

Unit I-17
Networking tools

Assigned to: Andrea

Knowledge and activity graph

Preliminary reading:

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

➡ XXX (ref to find).

TODO: to write

Make sure that you know:

17.1. `hostname`

TODO: to write

17.2. Visualizing information about the network

1) `ping`: are you there?

TODO: to write

2) `ifconfig`

TODO: to write

```
$ ifconfig
```

UNIT I-18

Accessing computers using SSH

> **Assigned to:** Andrea

18.1. Background reading

TODO: to write

- Encryption
- Public key authentication

18.2. 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

18.3. 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
$ vim ~/.ssh/config
```

+ comment
laptop or duckiebot? - LP

Then add the following lines:

```
HostKeyAlgorithms ssh-rsa
```

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

18.4. 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`.

18.5. Creating an SSH keypair

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

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

Use the convention

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

where:

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

An SSH key can be generated with the command:

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

The session output will look something like this:

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

At this point, tell it to choose this file:

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

Then:

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

Press enter; you want an empty passphrase.

```
Enter same passphrase again:
```

Press enter.

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

18.6. How to login without a password

KNOWLEDGE AND ACTIVITY GRAPH

Requires: You have two computers, called "`local`" and "`remote`", with users "`local-user`" and "`remote-user`". Here, we assume that `local` and `remote` are complete hostnames (such as `duckiebot.local`.).

Requires: The two computers are on the same network and they can ping each other.

Requires: You have created a keypair for `local-user` on `local`. This procedure is described in Section 18.5 - Creating an SSH keypair.

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 ping the remote one:

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

Note that, above, `remote` should be a complete hostname, such as `duckiebot.local`. 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.

18.7. 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/*
```

18.8. ssh-keygen

TODO: to write

Unit I-19
Wireless networking in Linux

19.1. iwconfig

TODO: to write

19.2. 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.

UNIT I-20

Moving files between computers

20.1. SCP

TODO: to write

1) Download a file with SCP

TODO: to write

20.2. RSync

TODO: to write

Unit I-21
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").

21.1. External documentation

➡ A VIM tutorial.

21.2. Installation

Install like this:

```
$ sudo apt install vim
```

21.3. `vi`

TODO: to write

21.4. Suggested configuration

Suggested `~/.vimrc`:

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

21.5. Visual mode

TODO: to write

21.6. 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.

UNIT I-22

Atom

22.1. Install Atom

Following the instructions here:

```
$ sudo add-apt-repository ppa:webupd8team/atom
$ sudo apt update
$ sudo apt install atom
```

Unit I-23
Liclipse

23.1. Why using IDEs

To be productive in coding, you need to have a proper IDE.

Look at how quickly you can create Python files by using an IDE (Figure 23.1). In this case, the editor is importing the required symbols. Think about how long would it take to do it without an IDE.

Figure 23.1. Eclipse editor capabilities

23.2. Other alternatives

In addition to LiClipse, the one that is suggested for this class, there exist:

- PyCharm
- Eclipse

23.3. Installing LiClipse

Follow the instructions at this page.

At the moment of writing, these are:

```
$ wget http://www.mediafire.com/file/rwc4bk3nthtxcvv/liclipse_4.1.1_linux.gtk.x86_64.tar.gz
$ tar xvzf liclipse_4.1.1_linux.gtk.x86_64.tar.gz
$ sudo ln -s `pwd`/liclipse/LiClipse /usr/bin/liclipse
```

Now you can run it using `liclipse`:

```
$ liclipse
```

When it runs for the first time, choose “use this as default” and click “launch”.

Choose “Import-> General -> Existing project into workspace”. Select the folder `~/duckietown`.

+ comment

Only Import -> General -> Projects from Folder or Archive, selecting `~/duckuments` worked for me. JT

+ comment
This is not about the duckuments, it's for `duckietown` - AC

If it asks about interpreters, select "auto config".

When it shows "uncheck settings that should not be changed", just click OK.

23.4. Set shortcuts for LiClipse

Go to "window/preferences/General/Keys".

Find "print", and unbind it.

Find "Quick switch editor", and set it to Ctrl-P. (This is now the same as Atom.)

Find "Previous tab" and assign Ctrl-Shift-[(This is now the same as Atom.)

Find "Next tab" and assign Ctrl-Shift-]. (This is now the same as Atom.)

Find "Show in (PyDev package explorer)" and assign Ctrl-Shift-M.

23.5. Shortcuts for LiClipse

Make sure that you can do the following tasks:

- Use the global browser: Press Cmd-Shift-T, type "what". It should autocomplete to `what_the_duck`. Press enter; it should jump to the file.
- Switch among open editors with Ctrl-P.
- Switch between tabs with Ctrl-Shift-], -[.
- See the current file in the directory, using Cmd-Shift-M.

TABLE 23.1. LICLIPSE COMMANDS

On Linux	On Mac	
Ctrl-Shift-T	Cmd-Shift-T	Globals browser
Ctrl-P	Cmd-P	Quick editor switch
Ctrl-Shift-[	Cmd-Shift-[	Previous tab (needs to be configured)
Ctrl-Shift-]	Cmd-Shift-]	Next tab (needs to be configured)
Ctrl-Shift-M	Cmd-Shift-M	Show in (PyDev package explorer)
Ctrl-1	Cmd-1	Find symbol

23.6. Other configuration for Liclipse

From the "Preferences" section, it's suggested to:

- Get rid of the minimap on the right.
- Get rid of spellchecking.

Then, there is the issue of "code completion". This is a love-it-or-hate-it issue. The choice is yours.

Unit I-24
Slack

24.1. Installing the Slack app on Linux

TODO: to write

24.2. Disabling Slack email notifications

Most importantly, please take the time to disable email notification for Slack.

The point of Slack is that you don't get email. You can work on Duckietown only when you have the time to do so.

TODO: to write how

24.3. Disabling Slack pop-up notification on the desktop

Also remove pop up notifications from the app. It should be a discrete notification that says "hey, when you have some time, look at Twist", not "HEY HEY HEY PAY ATTENTION TO ME NOW".

TODO: to write procedure

UNIT I-25

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.)

25.1. Advantages of using Byobu

TODO: To write

25.2. Installation

On Ubuntu, install using:

```
$ sudo apt install byobu
```

25.3. Documentation

☀ See the screencast on the website http://byobu.co/.

25.4. 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 25.1. 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 25.2. COMMANDS FOR PANES

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

Other commands:

TABLE 25.3. OTHER

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

25.5. Commands on OS X

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

Highlight using alt

UNIT I-26

Source code control with Git

> **Assigned to:** Andrea

26.1. Background reading

- Good book
- Git Flow

26.2. 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.

26.3. 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
```

26.4. Git tips

1) Delete branches

Delete local:

```
$ git branch -d branch-name
```

Delete remote:

```
$ git push origin --delete branch-name
```

Propagate on other machines by doing:

```
$ git fetch --all --prune
```

2) Shallow clone

You can clone without history with the command:

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

26.5. 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
```

26.6. git

TODO: to write

26.7. hub

1) Installation

Install `hub` using the instructions.

2) Creating pull requests

You can create a pull request using:

```
$ hub pull-request -m "description"
```

Unit I-27
Git LFS

This describes Git LFS.

27.1. Generic installation instructions

See instructions at:

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

27.2. 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
```

27.3. 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.

27.4. 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
```

UNIT I-28

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.

28.1. 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 28.1).

Figure 28.1

Go to your inbox and verify the email.

28.2. 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.

28.3. 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.

KNOWLEDGE AND ACTIVITY GRAPH

Requires: A public/private keypair already created and configured. This procedure is explained in Section 18.5 - Creating an SSH keypair.

Results: You can access Github using the key provided.

Go to settings (Figure 28.2).

Figure 28.2

Add the public key that you created:

Figure 28.3

Figure 28.4

Figure 28.5

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.

UNIT I-29

ROS installation and reference

> **Assigned to:** Liam

29.1. Install ROS

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

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

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

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

Fetch the ROS repo:

```
$ sudo apt update
```

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

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

There's more to install:

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

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

> XXX: not done in aug20 image:

Initialize ROS:

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

29.2. `rqt_console`

TODO: to write

29.3. roslaunch

TODO: to write

29.4. rviz

TODO: to write

29.5. rostopic

TODO: to write

1) rostopic hz

TODO: to write

2) rostopic echo

TODO: to write

29.6. catkin_make

TODO: to write

29.7. rosrun

TODO: to write

29.8. rostest

TODO: to write

29.9. rospack

TODO: to write

29.10. rosparam

TODO: to write

29.11. rosdep

TODO: to write

29.12. roswtf

TODO: to write

29.13. rosbag

A bag is a file format in ROS for storing ROS message data. Bags, so named because of their .bag extension, have an important role in ROS. Bags are typically created by a tool like `rosbag`, which subscribe to one or more ROS topics, and store the serialized message data in a file as it is received. These bag files can also be played back in ROS to the same topics they were recorded from, or even remapped to new topics.

1) rosbag record

The command `rosbag record` records a bag file with the contents of specified topics.

2) rosbag info

The command `rosbag info` summarizes the contents of a bag file.

3) rosbag play

The command `rosbag play` plays back the contents of one or more bag files.

4) rosbag check

The command `rosbag check` determines whether a bag is playable in the current system, or if it can be migrated.

5) rosbag fix

The command `rosbag fix` repairs the messages in a bag file so that it can be played in the current system.

6) rosbag filter

The command `rosbag filter` converts a bag file using Python expressions.

7) rosbag compress

The command `rosbag compress` compresses one or more bag files.

8) rosbag decompress

The command `rosbag decompress` decompresses one or more bag files.

9) rosbag reindex

The command `rosbag reindex` re-indexes one or more broken bag files.

29.14. `roscore`

TODO: to write

29.15. 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 18.3 - Local configuration.

29.16. Other materials about ROS.

- A gentle introduction to ROS

Part J
Software development guide

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

Unit J-1
Python

1.1. Background reading

- Python
- Python tutorial

1.2. Python virtual environments

Install using:

```
$ sudo apt install virtualenv
```

1.3. Useful libraries

```
matplotlib
seaborne
numpy
panda
scipy
opencv
...
```

1.4. Context managers

TODO: to write

1.5. Exception hierarchies

TODO: to write

TODO: how to catch and re-raise

1.6. Object orientation - Abstract classes, class hierarchies

TODO: to write

1.7. Downloading resources

Use this recipe if you need to download things:

```
from duckietown_utils.download import download_if_not_exist
url = 'https://www.dropbox.com/s/bzezpw8ivlfu4b0/frame0002.jpg?dl=0'
f = 'local.jpg'
download_if_not_exist(url, f)
```

(Do not commit JPGs and other binary data to the `Software` repository.)

TODO: actually use `urls.yaml`

1.8. IPython

How to enter IPython:

```
from IPython import embed()

a = 10
embed() # enters interactive mode
```

1.9. Idioms

`segment_list = copy.deepcopy(segment_list)`

➡ add_duckietown_header

➡

Unit J-2
Working with YAML files

YAML files are useful for writing configuration files, and are used a lot in Duckietown.

2.1. Pointers to documentation

TODO: to write

2.2. Editing YAML files

TODO: to write

2.3. Reading and writing YAML files in Python

TODO: the yaml library

TODO: the ruamel.yaml library

2.4. Duckietown wrapping API

TODO: to write

UNIT J-3

Duckietown code conventions

3.1. Python

1) Tabs

Never use tabs in Python files.

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) The encoding line

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

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

5) Sha-bang lines

Executable files start with:

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

6) Comments

Comments refer to the next line.

Comments, bad:

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

Comments, better:

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

3.2. Logging

For logging, import this logger:

```
from duckietown_utils import logger
```

3.3. Exceptions

```
DTConfigException

raise_wrapped
compact = True
```

3.4. 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 *
```

UNIT J-4

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` (Subsection 8.1.3 - The `what-the-duck` program) 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.

4.1. Environment variables (updated Sept 12)

You need to have set up the variables in Table 4.1.

> **Note:** The way to set these up is to add them in the file `~/.bashrc` (`export var='value'`). Do not modify the `environment.sh` script.

TABLE 4.1. ENVIRONMENT VARIABLES USED BY THE SOFTWARE

variable	reasonable value	contains
`DUCKIETOWN_ROOT`	`~/duckietown`	Software repository
`DUCKIEFLEET_ROOT`	`~/duckiefleet`	Where to look for class-specific information (people DB, robots DB).
`DUCKIETOWN_DATA`	`~/duckietown-data`	The place where to look for logs.
`DUCKIETOWN_TMP`		If set, directory to use for temporary files. If not set, we use the default (`/tmp`).
`DUCKIETOWN_CONFIG_SEQUENCE`	`defaults:baseline:vehicle:user`	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.

4.2. The "scuderia" (vehicle database)

The system needs to know certain details about the robots, such as their host names, the name of the users, etc.

This data is contained in the `${DUCKIEFLEET_ROOT}` directory, in files with the pattern `robot name.robot.yaml`.

The file must contain YAML entries of the type:

```yaml
owner: ID of owner
username: username on the machine
hostname: host name
description: generic description
log:
    date: comment
    date: comment
```

A minimal example is in Listing 4.6.

```yaml
owner: censi
hostname: emma
username: andrea
description: Andrea's car.
log:
    2017-08-01: >
        Switched RPI2 with RPI3.
    2017-08-20: >
        There is something wrong with the PWM hat and the LEDs.
```

Listing 4.6. Minimal scuderia file `emma.robot.yaml`

Explanations of the fields:

- `hostname`: the host name. This is normally the same as the robot name.
- `username`: the name of the Linux user on the robot, from which to run programs.
- `owner`: the owner's globally-unique Duckietown ID.

4.3. The `machines` file

Make sure you already set up ROS (Section 10.3 - Set up the ROS environment on the Duckiebot).

Activate ROS:

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

The `machines` file is created from the scuderia data using this command:

```
$ rosrun duckieteam create-machines
```

4.4. People database

> Assigned to: Andrea

TODO: Describe the people database.

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.

+ comment

There is no Slack username anymore, so we should change this to some other convention. -AC

4.5. 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 4.2. 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`

UNIT J-5

Node configuration mechanisms

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

UNIT J-6

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.

6.1. 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 XXX

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 *
```

6.2. 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 11.1 - chmod.

1) Header

Header:

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

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

Every ROS node in Python must start with this line.

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

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

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

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

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

2) Main

This is the main file:

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

    # Create the NodeName object
    node = Talker()

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

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

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

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

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

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

The line `rospy.spin()` blocks to keep the script alive. This makes sure the node stays

alive and all the publication/subscriptions work correctly.

6.3. The Talker class

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

1) Constructor

In the constructor, we have:

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

saves the name of the node.

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

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

The line:

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

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

➡ For more details see rospy overview: publisher and subscribers.

The line:

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

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

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

6.4. Launch File

You should always write a launch file to launch a node. It also serves as a documentation on the I/O of the node.

Let's take a look at `launch/test.launch`.

```
<launch>
    <node name="talker" pkg="pkg_name" type="talker.py" output="screen">

        <param name="~pub_timestep" value="0.5"/>

        <remap from="~topic_b" to="~topic_a"/>
    </node>
</launch>
```

For the `<node>`, the `name` specify the name of the node, which overwrites `rospy.init_node()` in the `__main__` of `talker.py`. The `pkg` and `type` specify the package and the script of the node, in this case it's `talke.py`.

Don't forget the `.py` in the end (and remember to make the file executable through `chmod`).

The `output="screen"` direct all the `rospy.loginfo` to the screen, without this you won't see any printouts (useful when you want to suppress a node that's too talkative.)

The `<param>` can be used to set the parameters. Here we set the `~pub_timestep` to `0.5`. Note that in this case this sets the value of `talker/pub_timestep` to `0.5`.

The `<remap>` is used to remap the topic names. In this case we are replacing `~topic_b` with `~topic_a` so that the subscriber of the node actually listens to its own publisher. Replace the line with

```
<remap from="~topic_b" to="talker/topic_a"/>
```

will have the same effect. This is redundant in this case but very useful when you want to subscribe to a topic published by another node.

6.5. Testing the node

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

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

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

```
$ rospack profile
```

Now you can launch the node by the launch file.

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

You should see something like this in the terminal:

```
... logging to /home/username/.ros/log/d4db7c80-b272-11e5-8800-5c514fb7f0ed/
roslaunch-robot name-15961.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is 1GB.

started roslaunch server http://robot name.local:33925/

SUMMARY
========

PARAMETERS
 * /rosdistro: $ROS_DISTRO
 * /rosversion: 1.11.16
 * /talker/pub_timestep: 0.5

NODES
  /
    talker (pkg_name/talker.py)

auto-starting new master
process[master]: started with pid [15973]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to d4db7c80-b272-11e5-8800-5c514fb7f0ed
process[rosout-1]: started with pid [15986]
started core service [/rosout]
process[talker-2]: started with pid [15993]
[INFO] [WallTime: 1451864197.775356] [/talker] Initialzing.
[INFO] [WallTime: 1451864197.780158] [/talker] ~pub_timestep = 0.5
[INFO] [WallTime: 1451864197.780616] [/talker] Initialzed.
[INFO] [WallTime: 1451864198.281477] [/talker] Goodbye, duckietown.
[INFO] [WallTime: 1451864198.781445] [/talker] Hello, duckietown.
[INFO] [WallTime: 1451864199.281871] [/talker] Goodbye, duckietown.
[INFO] [WallTime: 1451864199.781486] [/talker] Hello, duckietown.
[INFO] [WallTime: 1451864200.281545] [/talker] Goodbye, duckietown.
[INFO] [WallTime: 1451864200.781453] [/talker] Goodbye, duckietown.
```

Open another terminal and run:

```
$ rostopic list
```

You should see

```
/rosout
/rosout_agg
/talker/topic_a
```

In the same terminal, run:

```
$ rosparam list
```

You should see the list of parameters, including `/talker/pub_timestep`.
You can see the parameters and the values of the `talker` node with

```
$ rosparam get /talker
```

6.6. 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.

6.7. Guidelines

- Make sure to put all topics (publish or subscribe) and parameters under the namespace of the node with `~`. This makes sure that the IO of the node is crystal clear.
- Always include the name of the node in the printouts.
- Always provide a launch file that includes all the parameters (using `<param>`) and topics (using `<remap>`) with each node.

UNIT J-7

Makefile system

> **Assigned to:** Andrea

We use Makefiles to describe frequently-used commands.

7.1. User guide

The command

```
$ make all
```

displays the help for each command. This help is also included here as Section 7.3 - Makefile help.

7.2. Makefile organization

There is one Makefile at the root of the `Software` repository, which includes other makefiles in the directory `Makefiles/`:

```
Makefile
Makefiles/
    Makefile.build.mk
    Makefile.demos.mk
    Makefile.docker.mk
    Makefile.generate.mk
    Makefile.hw_test.mk
    Makefile.maintenance.mk
    Makefile.openhouse.mk
    Makefile.stats.mk
    Makefile.test.mk
```

Each child Makefile is called

```
Makefile.section.mk
```

and it should contain only targets of the form `section-name`.

For example, `Makefile.stats.mk` contains the targets "`stats`, `stats-easy_node`, `stats-easy_logs`".

The target called `section` should provide an help for the section.

For example, when you run `make build`, you see:

```
### Building commands

Commands to build the software.

- `make build-machines`       :  Builds the machines file.
- `make build-machines-clean` :  Removes the machines file.
- `make build-clean`          :  Clean everything.
```

The output should be valid Markdown, so that it can be included in this documentation.

7.3. Makefile help

1) Statistics

These provide statistics about the data and the configuration.

- `make stats-easy_node`: Prints summary of declared nots using the EasyNode frameworks.
- `make stats-easy_logs`: Prints summary of available logs.
- `make stats-easy_algo`: Prints summary of available algorithms.

2) Testing

These commands run the unit tests.

- `make test-all`: Run all the tests.
- `make test-circle`: The tests to run in continuous integration. .
- `make test-catkin_tests`: Run the ROS tests.
- `make test-anti_instagram`: Run the `anti_instagram` tests.
- `make test-comptests`: Run the `comptests` tests.
- `make test-comptests-clean`: Run the `comptests` tests.
- `make test-comptests-collect-junit`: Collects the JUnit results.
- `make test-download-logs`: Downloads the logs needed for the tests.

3) Building commands

Commands to build the software.

- `make build-catkin` : Runs `catkin_make`.
- `make build-catkin-parallel` : Runs `catkin_make`, with 4 threads.
- `make build-catkin-parallel-max` : Runs `catkin_make`, with many threads.
- `make build-machines` : Builds the machines file.
- `make build-machines-clean` : Removes the machines file.
- `make build-clean` : Clean everything.

4) Docker commands

For using Docker images

- `make docker-build`: Creates the image.

- `make docker-upload`: Uploads the image.
- `make docker-clean`: Removes all local images.

5) Automated files generation

Generation of documentation

- `make generate-all`: Generates everything.
- `make generate-help`: Generates help.
- `make generate-easy_node`: Generates the easy node documentation.
- `make generate-easy_node-clean`: Cleans the generated files.

6) Demos

These are simple demos

TODO: to write

7) Hardware tests

To perform hardware tests:

- `make hw-test-camera` : Testing Camera HW by taking a picture (smile!).
- `make hw-test-turn-right`: Calibration right turn
- `make hw-test-turn-left`: Calibrating left turn
- `make hw-test-turn-forward`: Calibrating forward turn

8) Maintenance

A couple of utilities for robot maintenance.

- `make maintenance-fix-time`: Fixes the time.
- `make maintenance-clean-pyc`: Removes pyc files.

9) Open house demos

These were the open house demos.

TODO: to write

Unit J-8
ROS package verification

Assigned to: Andrea

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

8.1. Naming

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

8.2. `package.xml`

□ There is a `package.xml` file.

✔ Checked by `what-the-duck`.

8.3. Messages

□ The messages are called ....

8.4. Readme file

□ There is a `README.md` file

✔ Checked by `what-the-duck`.

8.5. Launch files

□ there is the first launch file

8.6. Test files

TODO: to write

UNIT J-9

Duckietown utility library

9.1. Images

This sections contains the documentation about the utility functions used for image processing available in the `duckietown_utils` Python package.

1) Function `write_image_as_jpg`

Description: Takes an BGR image and writes it as a JPEG file.

+ comment

Are we sure that the encoding is right? -AC

Prototype:

```
write_image_as_jpg( image, filename )
```

Defined in: `image_writing.py`.

Arguments:

Name	Type	Description
image	numpy.ndarray	The BGR image to save as JPEG file.
filename	str	The path of the JPEG file.

Returns: `None`.

2) Function `rgb_from_ros`

Description: Takes a ROS message containing an image and returns its RGB representation.

Prototype:

```
rgb_from_ros( msg )
```

Defined in: `image_conversions.py`.

Arguments:

Name	Type	Description
msg	sensor_msgs.Image or sensor_msgs.CompressedImage	Message containing the image to extract.

Returns: `numpy.ndarray` :: RGB representation of the image contained in the ROS message `msg`.

3) Function `d8_compressed_image_from_cv_image`

Description: Takes a OpenCV image (BGR format), compresses it and wraps it into a ROS message of type `sensor_msgs.CompressedImage`.

Prototype:

```
d8_compressed_image_from_cv_image( image_cv )
```

Defined in: `image_jpg_create.py`.

Arguments:

Name	Type	Description
image_cv	numpy.ndarray	BGR representation of the image to compress.

Returns: `sensor_msgs.CompressedImage` :: A ROS message containing a compressed version of the input `image_cv`.

UNIT J-10

Creating unit tests with ROS

```
catkin_make -C catkin_ws/ --pkg easy_logs
```

UNIT J-11
Continuous integration

These are the conventions for the Duckietown repositories.

11.1. Never break the build

The `Software` and the `duckuments` repository use “continuous integration”.

This means that there are well-defined tests that must pass at all times.

For the `Software` repository, the tests involve building the repository and running unit tests.

For the `duckuments` repository, the tests involve trying to build the documentation using `make compile`.

If the tests do not pass, then we say that we have “broken the build”.

We also say that a branch is “green” if the tests pass, or “red” otherwise.

If you use the Chrome extension Pointless, you will see a green dot in different places on Github to signify the status of the build (Figure 11.1).

Figure 11.1. The green dot is good.

11.2. How to stay in the green

The system enforces the constraint that the branch `master` is always green, by preventing changes to the branches that make the tests fail.

We use a service called CircleCI. This service continuously looks at our repositories. Whenever there is a change, it downloads the repositories and runs the tests.

(It was a lot of fun to set up all of this, but fortunately you do not need to know how it is done.)

At this page you can see the summary of the tests. (You need to be logged in with your Github account and click “authorize Github”).

Figure 11.2. The CircleCi service dashboard, available at this page.

11.3. How to make changes to `master`: pull requests

It is not possible to push on to the master branch directly.

> ➡ See the Github documentation about pull requests to learn about the general concept.

The workflow is as follows.

(1) You make a private branch, say `your name-devel`.

(2) You work on your branch.

(3) You push often to your branch. Every time you push, CircleCI will run the tests and let you know if the tests are passing.

(4) When the tests pass, you create a "pull request". You can do this by going to the Github page for your branch and click on the button "compare and pull request" (Figure 11.3).

Figure 11.3. Compare and pull request button

(5) You now have an opportunity to summarize all the changes you did so far (Figure 11.4). Then click "create pull request".

Figure 11.4. Preparing the pull request

(6) Now the pull request has been created. Other people can see and comment on it. However, it has not been merged yet.

At this point, it might be that it says "Some checks haven't completed yet" (Figure 11.5). Click "details" to see what's going on, or just wait.

Figure 11.5. Wait for the checks to finish

When it's done, you will see either a success message (Figure 11.6) or a failure message

(Figure 11.7).

Figure 11.6. The tests are done

Figure 11.7. The tests have failed

(7) At this point, you can click "squash and merge" to merge the changes into master (Figure 11.8).

Figure 11.8. The final step

1) Troubleshooting

If you see a message like "merge failed" (Figure 11.9), it probably means that somebody pushed into master; merge master into your branch and continue the process.

Figure 11.9. Merge failed

Part K
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 J - Software development guide.

We do give links to the ROS packages implementing the functionality.

Unit K-1
Teleoperation

TODO: add video here

1.1. Implementation

Drivers:

- Unit N-1 - Package `adafruit_drivers`
- Unit N-4 - Package `pi_camera`

Operator interface:

- Unit N-3 - Package `joy_mapper`

1.2. Camera

TODO: to write

1.3. Actuators

TODO: to write

1.4. IMU

TODO: to write

UNIT K-2

Parallel autonomy

TODO: to write

UNIT K-3

Lane control

TODO: video here

3.1. Implementation

Perception:

- Unit O-1 - Package `anti_instagram`
- Unit O-3 - Package `ground_projection`
- Unit O-7 - Package `line_detector`, Unit O-6 - Package `line_detector2`
- Unit O-5 - Package `lane_filter`

Control:

- Unit O-4 - Package `lane_control`
- Unit N-2 - Package `dagu_car`

UNIT K-4
Indefinite navigation

TODO: add video here

4.1. Implementation

The packages involved in this functionality are:

- Unit P-2 - Package `apriltags_ros`
- Unit P-3 - Package `fsm`
- Unit P-4 - Package `indefinite_navigation`
- Unit P-5 - Package `intersection_control`
- Unit P-6 - Package `navigation`

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

UNIT K-5

Planning

TODO: add video here

5.1. Implementation

The packages involved in this functionality are:

- Unit Q-2 - Package `localization`
- Unit Q-1 - Package `duckietown_description`

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

Unit K-6
Coordination

TODO: add video here

6.1. Implementation

- Unit R-1 - Package `led_detection`
- Unit R-2 - Package `led_emitter`
- Unit R-3 - Package `led_interpreter`
- Unit R-4 - Package `led_joy_mapper`
- Unit R-6 - Package `traffic_light`
- Unit R-5 - Package `rgb_led`

UNIT K-7

Duckietown ROS Guidelines

7.1. Node and Topics

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

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

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

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

7.2. Parameters

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

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

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

Note: The above is deprecated. The configuration is handled differently.

7.3. Launch file

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

Each elemental launch file must only launch one node.

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

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

A node should always be launched by calling its corresponding launch file instead of

using `rosrun`. This ensures that the node is put under the correct namespace and all the necessary parameters are provided.

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

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

Part L
Fall 2017

Welcome to the Fall 2017 Duckietown experience.

Unit L-1
The Fall 2017 Duckietown experience

This is the first time that a class is taught jointly across 3 continents!

There are 4 universities involved in the joint teaching for the term:

- ETH Zürich (ETHZ), with instructors Emilio Frazzoli, Andrea Censi, Jacopo Tani.
- University of Montreal (UdeM), with instructor Liam Paull.
- TTI-Chicago (TTIC), with instructor Matthew Walter.
- National C T University (NCTU), with instructor Nick Wang.

This part of the Duckiebook describes all the information that is needed by the students of the four institutions.

At ETHZ, UdeM, TTIC, the class will be more-or-less synchronized. The materials are the same; there is some slight variation in the ordering.

Moreover, there will be some common groups for the projects.

The NCTU class is undergraduate level. Students will learn slightly simplified materials. They will not collaborate directly with the other classes.

1.1. The rules of Duckietown

The first rule of Duckietown

The first rule of Duckietown is: you don't talk about Duckietown, *using email*.

Instead, we use a communication platform called Slack.

There is one exception: inquiries about "meta" level issues, such as course enrollment and other official bureaucratic issues can be communicated via email.

The second rule of Duckietown

The second rule of Duckietown is: be kind and respectful, and have fun.

The third rule of Duckietown

The third rule of Duckietown is: read the instructions carefully.

Do not blindly copy and paste.

Only run a command if you know what it does.

UNIT L-2

First Steps in Duckietown

2.1. Onboarding Procedure

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

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

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

1) Github sign up

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

➡ Github signup page

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

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

2) Questionnaire

Next, fill in this questionnaire:

Preliminary Student Questionnaire

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

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

3) Accept invite to Github organization Duckietown

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

The invite should be waiting for you at this page.

4) Accept the invite to Slack

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

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

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

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

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

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

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

example Andrea Censi - Zurich

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

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

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

2.2. (optional) Add Duckietown Engineering Linkedin profile

This is an optional step.

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

2.3. Laptops

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

You need a laptop with these specifications:

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

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

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

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

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

those problems.

The issues that you will encounter are of two types.

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

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

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

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

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

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

2.4. Next steps for people in Zurich

1) Get acquainted with class journal and class logistics

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

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

➡ Unit L-10 - Zurich branch diary

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

➡ Unit L-3 - Logistics for Zürich branch

2) Make sure you can edit the Duckuments

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

➡ See the instructions in this section.

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

2.5. Next steps for people in Chicago

TODO: to write

Unit L-3
Logistics for Zürich branch

This section describes information specific to Zürich.

1) The local staff

These are the local TAs:

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

Please contact them on Slack, rather than email.

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

3.1. HR

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

3.2. Website / class journal

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

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

➡ Unit L-10 - Zurich branch diary

3.3. Duckiebox

The point of contact for Duckiebox distribution is Shiying Li.

3.4. Duckietown room access

The local Duckietown room is ML J 44.2.

TODO: write opening hours and rules

+ comment
double check the room number -AC

3.5. Extra spaces

There will be extra lab space available.

Space-time coordinates TBD.

Unit L-4
Logistics for Montréal branch

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

4.1. Website

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

4.2. Class Schedule

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

4.3. Lab Access

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

4.4. The Local Staff

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

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

4.5. Storing Your Robot

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

Unit L-5
Logistics for Chicago branch

> Assigned to: Matt

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

1) Website

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

2) Class Schedule

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

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

3) Lab Access

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

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

4) The Local LAs

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

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

Unit L-6
Git usage guide for Fall 2017

6.1. Repositories

These are the repositories we use.

1) Software

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

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

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

2) Duckiefleet

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

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

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

3) Duckuments

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

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

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

4) Lectures

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

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

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

5) Exercises

The exercises repository contains the solution to exercises.

The URL to clone is:

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

Only TAs have write permissions to this repository.

6.2. Git policy for homeworks

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

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

1) Duckiefleet file structure

You should put your homework files in folder at:

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

2) To submit your homework

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

Command line:

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

Through github:

1. Click on the "Releases" tab
2. Click "Create a new Release"
3. Add a version (e.g. 1.0)
4. Release title put `XX_homework_name`

5. In the text box put "@duckietown/fall-2017-instructors-and-tas homework complete"
6. Click "Publish release"

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

3) Merging things back

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

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

6.3. Git policy for project development

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

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

UNIT L-7
Organization

The following roster shows the teaching staff.

Andrea Censi

Liam Paull

Jacopo Tani

First Last

Emilio Frazzoli

First Last

First Last

First Last

First Last

First Last

First Last

First Last

First Last

First Last

First Last

Greta Paull

Kirsten Bowser

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

7.1. The Activity Tracker

TABLE 7.1. THE STUDENT ACTIVITY TRACKER

Student Activity Tracker : DO NOT USE YET Task List

Link to Activity Tracker

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

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

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

You have two options:

- Only use the sheet as a reference;
- Use the sheet actively to track your progress. To do this, send a message to Kirsten

with your gmail address, and add yourself.

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

The colored boxes have the following meaning:

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

7.2. The Areas sheet

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

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

Unit L-8
Getting and giving help

8.1. *Who* to ask for help

1) Primary points of contact

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

2) Point of contacts for specific documents

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

8.2. *How* to ask for help

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

1) Case: You find a mistake in the documentation

Action: Please fix it.

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

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

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

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

Feel free to add additional explanations.

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

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

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

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

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

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

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

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

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

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

4) Case: You are having a technical issue related to building the documentation

Action: Open an issue on the `duckuments` github page and provide all necessary information to reproduce it.

5) Case: You have found a well-defined defect in the software.

Action: open an issue on the `Software` repository github page and provide all necessary information for reproducing the bug.

Unit L-9
Slack Channels

This page describes all of the helpful Slack channels and their purposes so that you can figure out where to get help.

9.1. Channels

You can also easily join the ones that you are interested in by clicking the links in this message.

Table 9.1. Duckietown Slack Channels

Channel	Purpose
`help-accounts`	Info about necessary accounts, such as Slack, Github, etc.
`help-assembly`	Help putting your robot together
`help-camera-calib`	Help doing the intrinsic and extrinsic calibration of your camera
`help-duckuments`	Help compiling the online documentation
`help-git`	Help with git
`help-infrastructure`	Help with software infrastructure, such as Makefiles, unit tests, continuous integration, etc.
`help-laptops`	Help getting your laptop setup with Ubuntu 16.04
`help-parts`	Help getting the parts for the robot or replacement parts if you broke something
`help-robot-setup`	Help getting the robot setup to do basic things like be driven with a joystick
`help-ros`	Help with the Robot Operating System (ROS)
`help-wheel-calib`	Help doing your odometry calibration

+ comment

Note that we can link directly to the channels. (See list in the org sheet.) -AC

UNIT L-10

Zurich branch diary

10.1. Wed Sep 20: Welcome to Duckietown!

This was an introduction meeting.

1) Material presented in class

These are the slides we showed:

- PDF
- Keynote (huge)

2) Feedback form

Please help us making the experience better by providing feedback (can be anonymous)

3) Pointers to reading materials

Read about Duckietown's history and watch the Duckumentary.

➡ Part A - The Duckietown project

Start learning about Git and Github.

➡ Unit I-26 - Source code control with Git

Montreal, Chicago? What's happening?

➡ Unit L-1 - The Fall 2017 Duckietown experience

10.2. Monday Sep 25: Introduction to autonomy

1) Material presented in class

These are the slides we presented:

a - Logistics: Keynote, PDF.

a - Autonomous Vehicles: Keynote, PDF.

c - Autonomous Mobility on Demand: Keynote, PDF.

d - Plan for the next months: Keynote, PDF.

2) Feedback form

Please help us making the experience better by providing feedback (can be anonymous)

3) Questions and answers

Q: MyStudies is not updated; I am still on the waiting list. What should I do?

Answer: Nothing. Don't worry, if you have received the onboarding email, you are in the class, even if you still appear in the waiting list. We will figure this out with the de-

partment.

Q: What version of Linux do I need to install?

Answer: 16.04.* (16.04.03 is the latest at time of this writing)

Q: Do I need to install OpenCV, ROS, etc.?

Answer: Not necessary. We will provide instructions for those steps.

Q: My laptop is not ready. I'm having problems installing Linux on a partition.

Answer: Don't worry, take a Duckie, and, take a breath. We have time to fix every issue. Start by asking for help in the #help-laptops channel in Slack. We will address the outstanding issues in the next classes.

Q: How much space do I need on my Linux partition?

Answer: At least 50 GB; 200 GB are recommended for easy processing of data (logs) later in the course. If you have less space (say ~100GB), it might be wise to acquire an external hard drive to use as storage.

Q: Are there going to be Linux training sessions?

Answer: Maybe. We didn't plan for it, but it seems that there is a need. Subject to figuring out the logistics, we might organize an extra "lab" session or produce a support video.

10.3. Monday Sep 25, late at night: Onboarding instructions

At some late hour of the night, we sent out the onboarding instructions.

➡ Section 2.1 - Onboarding Procedure

Please complete the onboarding questionnaire by Tuesday, September 26, 15:00.

10.4. Wednesday Sep 27: Duckiebox distribution, and getting to know each other

Today we distribute the Duckieboxes and we name the robots. In other words, we perform the *Duckiebox ceremony*.

- getting to know each other;
- naming the robots;
- distribute the Duckieboxes.

> Note: If you cannot make it to this class for the Duckiebox distribution, please inform the TA, to schedule a different time.

1) Preparation, step 1: choose a name for your robot

Before arriving to class, you must think of a name for your robot.

Here are the constraints:

- The name must work as a hostname. It needs to start with a letter, contains only letters and numbers, and no spaces or punctuation.
- It should be short, easy to type. (You'll type it a lot.)
- It cannot be your own name.
- It cannot be a generic name like "robot", "duckiebot", "car". It cannot contain brand names.

2) Preparation, step 2: prepare a brief elevator pitch

As members of the same community, it is important to get to know a little about each other, so to know who to rely on in times of need.

During the Duckiebox distribution ceremony, you will be asked to walk up to the board, write your name on it, and introduce yourself. Keep it very brief (30 seconds), and tell us:

- what is your professional background and expertise / favorite subject;
- what is the name of your robot;
- why did you choose to name your robot in that way.

You will then receive a Duckiebox from our senior staff, a simple gesture but of sempiternal glory, for which you have now become a member of the Duckietown community. This important moment will be remembered through a photograph. (If in the future you become a famous roboticist, we want to claim it's all our merit.)

Finally, you will bring the Duckiebox to our junior staff, who will apply labels with your name and the name of the robot. They will also give you labels with your robot name for future application on your Duckiebot.

3) Feedback form

Please help us making the experience better by providing feedback (can be anonymous)

4) Material presented in class

- Duckiebot parts

10.5. (Template for every lecture) Date: Topic

> Assigned to: Name of TA

1) Preparation

Things that the students should do before class.

2) Material presented in class

Link to PDF and Keynote/Powerpoint materials.

3) Pointers to reading materials

Links to the units mentioned in the slides, and additional materials.

4) Questions and answers

Write here the FAQs that students have following the lecture or instructions.

Unit L-11
Montréal branch diary

11.1. Wed Sept 6

Class (11:30)

Slides:

- Duckietown History Future (keynote) (pdf)
- Duckietown Intro (keynote) (pdf)
- Autonomy Overview (keynote) (pdf)
- Autonomous Vehicles (keynote) (pdf)

Book materials:

- Part A - The Duckietown project
- Unit G-1 - Autonomous Vehicles
- Unit G-2 - Autonomy overview

11.2. Friday Sept 8

Acceptance emails sent

11.3. Sun Sept 10

- Onboarding email sent to accepted students

11.4. Mon Sept 11

Class (10:30)

> Note: This class we are meeting in rm. 2333 Pavillion André Aisenstadt.

- Logistics
- The Duckiebook
- Slack (Unit L-9 - Slack Channels)
- Git repos (Unit L-6 - Git usage guide for Fall 2017)
- How to get and give help (Unit L-8 - Getting and giving help)
- Grading scheme
- Student/Staff Introductions
- Duckiebox distribution.
- Go through the Duckiebox parts
- Unit L-14 - Checkoff: Duckiebot Assembly and Configuration initiated.

Deadline: **Mon Sept. 25**

11.5. Wed Sept 13

Class canceled. Continue working on Unit L-14 - Checkoff: Duckiebot Assembly and Configuration.

11.6. Mon Sept 18

Class (10:30 - 11:30)

- General discussion (how are things going? Sorry I was away last week.. anyone need anything?)
- Intro to robotics - Modern robotic systems
- The robot as a system - System architectures
- Decomposing the robotics sytems into smaller pieces - autonomy architectures
- Agreeing on the language that the different pieces "talk" - representations
- Background on basic probability theory?

Slides:

- Modern Robotic Systems (keynote) (pdf)
- System Architecture (keynote) (pdf)
- Autonomy Architectures (keynote) [(pdf)]((https://github.com/duckietown/lectures/blob/master/2_given/2017-09-18-udem-autonomy_architectures.pdf)
- Representations for Robotics (keynote) (pdf)

Book Materials:

- Modern Robotic Systems
- System Architecture Basics
- Autonomy Architectures
- Representations
- Probability Basics

Lab (11:30 - 12:30)

- Liam will be in 2333 setting up network and building Duckietown.

11.7. Wed Sept 20

Class (11:30 - 12:30)

- Robotics middlewares - what are they and basic concepts
- Introduction to the Robot Operating System (ROS)

Slides:

- Software Architectures (keynote) (pdf)
- Introduction to ROS (keynote) (pdf)

Book Materials:

- Introduction to ROS
- Middlewares

Lab (12:30 - 1:30)

Homeworks and Checkoffs:

- Unit L-14 - Checkoff: Duckiebot Assembly and Configuration deadline extended to **Monday Sept 25**. Submit by clicking here.

11.8. Mon Sept 25

Class (10:30-11:30)

- Microelectronics (pptx) (pdf)
- Modern Signal Processing (keynote) (pdf)

- Intro to Networking (keynote) (pdf)

Book Materials (still rough drafts, will be completed soon):

Lab (11:30 - 12:30)

- Liam will be in 2333
- Any final help needed for Unit L-14 - Checkoff: Duckiebot Assembly and Configuration
- Finalize Duckietown map setup

Homeworks and Checkoffs:

- Unit L-14 - Checkoff: Duckiebot Assembly and Configuration due by 11pm.
- New checkoff: "Taking a log" will be linked later today (due Monday Oct 2)
- New homework: "Data processing" will be linked later today (due Monday Oct 2)

11.9. Wed Sept 27

- Kinematics
- Dynamics

UNIT L-12

Chicago branch Diary

Classes take place on Mondays and Wednesdays from 9am-11am in TTIC Room 530.

12.1. Monday September 25: Introduction to Duckietown

Slides:

- Duckietown Course Intro (Keynote)(PDF)

Book Materials:

- Part A - The Duckietown project
- Unit G-1 - Autonomous Vehicles
- Unit G-2 - Autonomy overview

12.2. Tuesday September 26

Acceptance and onboarding e-mails sent

12.3. Wednesday, September 27

- Logistics
- The Duckiebook
- Slack (Unit L-9 - Slack Channels)
- Git repos (Unit L-6 - Git usage guide for Fall 2017)
- How to get and give help (Unit L-8 - Getting and giving help)
- Grading scheme
- Student/Staff Introductions
- Duckiebox distribution.
- Go through the Duckiebox parts
- Unit L-14 - Checkoff: Duckiebot Assembly and Configuration initiated.

Unit L-13
Guide for TAs

13.1. Dramatis personae

These are the TAs.

At ETHZ:

- Shiying Li (shili@student.ethz.ch)
- Ercan Selçuk (ercans@student.ethz.ch)
- Miguel de la Iglesia Valls (dmiguel@student.ethz.ch)
- Khurana Harshit (hkhurana@student.ethz.ch)
- Lapandic Dzenan (ldzenan@student.ethz.ch)
- Marco Erni (merni@ethz.ch)

At TTIC:

- Andrea F. Daniele (afdaniele@ttic.edu)
- Falcon Dai (dai@ttic.edu)
- Jon Michaux (jmichaux@ttic.edu)

At Montreal:

- Florian Golemo (fgolemo@gmail.com)

13.2. First steps

Here are the first steps for the TAs.

Note that many of these are not sequential and can be done in parallel.

1) Learn about Duckietown

Read about Duckietown's history; watch the Duckumentary.

➡ Part A - The Duckietown project

2) Online accounts

You have to set up:

- A personal Github account
- A Twist account
- A Slack account
- A Google Docs account (Gmail address)

Send an email to Kirsten Bowser (akbowser@gmail.com), with your GMail address and your Github account. She will give you further instructions.

Point of contact: Kirsten Bowser

3) Install Ubuntu

Install Ubuntu 16.04 on your laptop, and then install ROS, Atom, Liclipse, etc.

➡ Unit C-7 - Installing Ubuntu on laptops

4) Duckuments

Install the Duckuments system, so you can edit these instructions.

➡ Part B - Duckumentation documentation.

Point of contact: Andrea

5) Learn about Git and Github

Start learning about Git and Github. You don't have to read the entirety of the following references now, but keep them "on your desk" for later reference.

➡ Good book

➡ Git Flow

Point of contact: Liam?

6) Continuous integration

Understand the continuous integration system.

➡ Documentation on continuous integration.

Point of contact: Andrea

7) Duckiebot building

Build your Duckiebot according to the instructions.

➡ Part C - Operation manual - Duckiebot

Point of contact: Shiying (ETH)

Point of contact: ??? (UdeM)

Point of contact: ??? (TTIC)

As you read the instructions, keep open the Duckuments source, and note any discrepancies. You must note any unexpected thing that is not predicted from the instruction. If you don't understand anything, please note it.

The idea is that dozens of other people will have to do the same after you, so improving the documentation is the best use of your time, and it is much more efficient than answering the same question dozens of times.

8) Other documentation outside of the Duckuments

We have the following four documents outside of the duckuments:

1. Organization chart: This is where we assign areas of responsibility.
2. Lecture schedule
3. Checkoff spreadsheet
4. The big TODO list: Where we keep track of things to do.

Unit L-14
Checkoff: Duckiebot Assembly and Configuration

The first job is to get your Duckiebot put together and up and running.

14.1. Pick up your Duckiebox

Slack channel: #help-parts

There is a checklist inside. You should go through the box and ensure that the parts that are supposed to be in it actually are inside.

If you are missing something, contact your local responsible and ask for help on Slack in the appropriate channel.

These sections describe the parts that are in your box.

- Unit C-2 - Acquiring the parts for the Duckiebot `DB17-wjd`
- Unit E-1 - Acquiring the parts for the Duckiebot `DB17-1c`

14.2. Soldering your boards

Depending on how kind your instructors/TAs are, you may have to solder your boards.

- Unit C-3 - Soldering boards for `DB17`
- Unit E-2 - Soldering boards for `DB17-1`

14.3. Assemble your Robot

Slack channel: #help-assembly

You are ready to put things together now.

- Unit C-5 - Assembling the Duckiebot `DB17`
- Unit E-4 - Bumper Assembly
- Unit E-3 - Assembling the Duckiebot `DB17-wjd1c`

14.4. Optional: Reproduce the SD Card Image

If you are very inexperienced with Linux/Unix/networking etc, then you may find it a valuable experience to reproduce the SD card image to “see how the sausage is made”.

- Unit C-6 - Reproducing the image

14.5. Setup your laptop

Slack channel: #help-laptops

The only officially supported OS is Ubuntu 16.04. If you are not running this OS it is recommended that you make a small partition on your hard drive and install the OS.

Related parts of the book are:

- Section 6.9 - How to make a partition if you want to make a partition
- Unit C-7 - Installing Ubuntu on laptops

14.6. Make your robot move

Slack channel: #help-robot-setup

Now you need to clone the software repo and run things to make your robot move.

First initialize the robot:

- Unit C-8 - Duckiebot Initialization

Then get it to move!

- Unit C-10 - Software setup and RC remote control

+ comment

there is a deadline at this link, but each institution will have different deadlines.
Might be a source of entropy -JT

UNIT L-15

Checkoff: Take a Log

KNOWLEDGE AND ACTIVITY GRAPH

Requires: Unit L-14 - Checkoff: Duckiebot Assembly and Configuration

Results: A verified log in rosbag format updoaded to Dropbox

15.1. Take a Log

Take a log following the procedure described here: Unit C-17 - Taking a log. Make sure to also verify that it's good Section 17.1 - Verify a log.

TODO: naming conventions

15.2. Upload the log

TODO: add link to file request

UNIT L-16

Homework: Data Processing

KNOWLEDGE AND ACTIVITY GRAPH

Requires: Unit L-15 - Checkoff: Take a Log

Requires: Unit I-29 - ROS installation and reference

Results: Ability to perform basic operations on images

Results: Build your first ROS package and node

Results: Ability to process imagery live

UNIT L-17

Guide for mentors

Assigned to: Liam?

UNIT L-18

Project proposals

TODO: to write

UNIT L-19

Template of a project

TODO: to write

Part M
Packages - Infrastructure

TODO: to write

Unit M-1
Package duckieteam

1.1. Package information

Link to package on Github

Essentials

Author: Andrea Censi (maintainer)

Description

Code for handling the DB of people and robots.

1.2. create-machines

This program creates the machines file, using the data contained in the scuderia database.

Run as follows:

```
$ rosrun duckieteam create-machines
```

1.3. create-roster

TODO: this program is unfinished

UNIT M-2

Package duckietown

2.1. Package information

Link to package on Github

Essentials

Maintainer: Mack

Description

The duckietown meta package

Unit M-3
Package duckietown_msgs

3.1. Package information

Link to package on Github

Essentials

Maintainer: Mack

Description

TODO: Add a description of package `duckietown_msgs` in `package.xml`.

UNIT M-4

Package easy_algo

4.1. Package information

Link to package on Github

Essentials

Author: Andrea Censi (maintainer)

Description

A package to make it easy to abstract about algorithm configuration and create regression tests.

4.2. Motivation

The problem that this package solves is the need to easily refer to the configuration of algorithms, and objects in general.

Here's one case: a node requires a line detector; a line detector has a dozen parameters. Do we want to put a dozen parameters in the node?

Or, think about the case where there are different line detector classes. Now, there must be a parameter in the node that tells which class to instantiate.

All of this business can be simplified by the features of EasyAlgo.

4.3. Usage

The approach is to declare that there is a "family" of things, and to provide the description of what those things are in YAML files. These files can be anywhere in `catkin_ws`.

To declare the family of line detectors we create a file `line_detector.easy_algo_family.yaml`, with the following content:

```
interface: line_detector.LineDetectorInterface
description: These are the line detectors.
```

This tells the system that there exists a family called `line_detector` (from the filename) and that the objects in the family are instances of line_detector.LineDetectorInterface.

Then, we can write the configuration files that describe the particular instances. These files have the pattern `instance name.family name.yaml`.

Continuing the example, suppose that there is a `line_detector.ConcreteLineDetector` with two parameters, `alpha` and `beta`.

Then we can define a `baseline` object by creating a file called `baseline.line_detector.yaml`, containing the following:

```
description: The baseline detector
constructor: line_detector.ConcreteLineDetector
parameters:
    alpha: 1.0
    beta: 1.0
```

Similarly, we can define an `experiment` object in a file called `experiment.line_detector.yaml`:

```
description: My experiment with wild parameters
constructor: line_detector.ConcreteLineDetector
parameters:
    alpha: 2.0
    beta: 13.0
```

These configuration files are parsed by EasyAlgo.

4.4. User interface

The user can see the list of family by using:

```
$ rosrun easy_algo summary
```

To see the list of instances available for a family, use:

```
$ rosrun easy_algo summary family name
```

4.5. The developer's point of view

From the developer's point of view, it is possibly to access the code using the EasyAlgoDB class.

Create an instance using get_easy_algo_db():

```
from easy_algo import get_easy_algo_db

algo_db = get_easy_algo_db()
```

Then create an instance like the following:

```
line_detector_instance = algo_db.create_instance('line_detector', 'baseline')
```

It is also possible to query the database using the function `query`:

```
# iterate over all possible line detectors
available = algo_db.query('line_detector', '*'
for name in available:
    line_detector = algo_db.create_instance('line_detector', name)
```

Unit M-5
Package easy_logs

5.1. Package information

Link to package on Github

Essentials

Author: Andrea Censi (maintainer)

Description

The `easy_logs` packages manages the Duckietown logs.
It indexes them and has a simple interface to query the DB.
Tests should use the `easy_logs` DB to know what logs to work on.

5.2. Command-line utilities

The package includes a few programs to query the database: `find`, `summary`, and `details`.
They all take the same form:

```
$ rosrun easy_logs program  query
```

where `query` is a query expressed in the selector language (Section 5.3 - Selector language).
Use the `summary` program to display a summary of the logs available:

```
    $ rosrun easy_logs summary ''*dp3tele*''
    | #     Log name                          date          length    vehicle name    valid
    | --    ------------------------------    ----------    ------    ------------
-----------
    | 0     20160504-dp3tele1-thing           2016-05-04     294 s    thing           Yes.
    | 1     20160506-dp3tele1-nikola          2016-05-06     321 s    nikola          Yes.
    | 2     2016-04-29-dp3tele-neptunus-0     2016-04-29     119 s    neptunus        Yes.
    | 3     20160503-dp3tele1-pipquack        2016-05-03     352 s    pipquack        Yes.
    | 4     20160503-dp3tele1-redrover        2016-05-03     384 s    redrover        Yes.
    | 5     20160504-dp3tele1-penguin         None          (none)    None            Not
indexed
    | 6     20160430-dp3tele-3-quackmobile    2016-04-30     119 s    quackmobile     Yes.
```

The `find` command is similar to `summary`, but it only displays the filenames:

```
$ rosrun easy_logs find vehicle:oreo
.../logs/20160400-phase3-dp3-demos/dp3auto_2016-04-29-19-58-57_2-oreo.bag
.../logs/20160400-phase3-dp3-demos/dp3auto_2016-04-29-19-56-57_1-oreo.bag
.../logs/20160400-phase3-dp3-demos/dp3tele_2016-04-29-19-29-57_1-oreo.bag
.../logs/20160210-M02_DPRC/onasafari/201602DD-onasafari-oreo-RCDP5-log_out.bag
.../logs/20160400-phase3-dp3-demos/dp3tele_2016-04-29-19-27-57_0-oreo.bag
.../pictures/M03_04-demo_or_die/onasafari/oreo_line_follow.bag
.../logs/20160400-phase3-dp3-demos/dp3auto_2016-04-29-19-54-57_0-oreo.bag
.../logs/20160400-phase3-dp3-demos/dp3tele_2016-04-29-19-31-57_2-oreo.bag
```

The `details` command shows a detailed view of the data structure:

```
$ rosrun easy_logs details dp3auto_2016-04-29-19-58-57_2-oreo
- - [log_name, dp3auto_2016-04-29-19-58-57_2-oreo]
  - [filename, ...]
  - [map_name, null]
  - [description, null]
  - [vehicle, oreo]
  - [date, '2016-04-29']
  - [length, 112.987947]
  - [size, 642088366]
  - - bag_info
    - compression: none
      duration: 112.987947
      end: 1461960050.639
      indexed: true
      messages: 15443
      size: 642088366
      start: 1461959937.651054
      topics:
      - {messages: 107, topic: /diagnostics,
         type: diagnostic_msgs/DiagnosticArray}
      - {messages: 8, topic: /oreo/LED_detector_node/switch,
         type: duckietown_msgs/BoolStamped}
      - {messages: 9, topic: /oreo/apriltags_global_node/apriltags,
         type: duckietown_msgs/AprilTags}
      - {messages: 9, topic: /oreo/apriltags_global_node/tags_image,
         type: sensor_msgs/Image}
  ...
```

5.3. Selector language

Here are some examples for the query language (Table 5.1).

Show all the Ferrari logs:

```
$ rosrun easy_logs summary vehicle:ferrari
```

All the logs of length less than 45 s:

```
$ rosrun easy_logs summary "length:<45"
```

All the invalid logs:

```
$ rosrun easy_logs summary "length:<45,valid:False"
```

All the invalid logs of length less than 45 s:

```
$ rosrun easy_logs summary "length:<45,valid:False"
```

TABLE 5.1. QUERY LANGUAGE

expression	example	explanation
`attribute:expr`	`vehicle:ferrari`	Checks that the attribute `[attribute` of the object satisfies the expression in `expr`
`>lower bound`	`>10`	Lower bound
`<upper bound`	`<1`	Upper bound
`expr1,expr2`	`>10,<20`	And between two expressions
`expr1+expr2`	`<5+>10`	Or between two expressions
`pattern`	`*ferrari*`	Other strings are interpreted as wildcard patterns.

5.4. Automatic log download using `require`

The command-line program `require` downloads requires logs from the cloud.

The syntax is:

```
$ rosrun easy_logs download log name
```

For example:

```
$ rosrun easy_logs download 20160223-amadoa-amadobot-RCDP2
```

If the file `20160223-amadoa-amadobot-RCDP2.bag` is not available locally, it is downloaded.

The database of URLs is at the file `dropbox.urls.yaml` in the package `easy_node`.

A typical use case would be the following, in which a script needs a log with which to work.

Using the `require` program we declare that we need the log. Then, we use `find` to find the path.

```
#!/bin/bash
set -ex

# We need the log to proceed
rosrun easy_logs require 20160223-amadoa-amadobot-RCDP2

# Here, we know that we have the log. We use `find` to get the filename.
filename= rosrun easy_logs find 20160223-amadoa-amadobot-RCDP2`

# We can now use the log
vdir $filename
```

5.5. Browsing the cloud

How do you know which logs are in the cloud?
Run the following to download a database of all the logs available in the cloud:

```
$ make cloud-download
```

You can query the DB by using `summary` with the option `--cloud`:

```
$ rosrun easy_logs summary --cloud '*RCDP6*'

| #    Log name                                  date         length    vehicle name
| --   ----------------------------------------   ----------   ------    ------------
| 0    20160122-censi-ferrari-RCDP6-catliu       2016-02-28    194 s    ferrari
| 1    20160122-censi-ferrari-RCDP6-joe-wl       2016-02-27    196 s    ferrari
| 2    20160228-sanguk-setlist-RCDP6-sangukbo    2016-03-02    193 s    ferrari
| 3    20160122-censi-ferrari-RCDP6-eharbitz     2016-02-27    198 s    ferrari
| 4    20160122-censi-ferrari-RCDP6-teddy        2016-02-28    193 s    ferrari
| 5    20160122-censi-ferrari-RCDP6-jenshen      2016-02-29     83 s    ferrari
```

Then, you can download locally using:

```
$ rosrun easy_logs download 20160122-censi-ferrari-RCDP6-teddy
```

Once it is downloaded, the log becomes part of the local database.

5.6. Advanged log indexing and generation

1) Shuffle

`expr/shuffle` shuffles the order of the logs in `expr`.
Give me all the `oreo` logs, in random order:

```
$ rosrun easy_logs summary  vehicle:oreo/shuffle
```

2) Simple indexing

`expr/[i]` takes the i-th entry.
Give me the first log:

```
$ rosrun easy_logs summary  "vehicle:oreo/[0]"
```

Give me a random log; i.e. the first of a random list.

```
$ rosrun easy_logs summary  "vehicle:oreo/shuffle/[0]"
```

3) Complex indexing

You can use the exact Python syntax for indexing, including `[a:]`, `[:b]`, `[a:b]`, `[a:b:c]`, etc.
Give me three random logs:

```
$ rosrun easy_logs summary  "all/shuffle/[:3]"
```

4) Sorting

TODO: To implement.

5) Time indexing

You can ask for only a part of a log using the syntax:

```
expr/{start:stop}
expr/{start:}
expr/{:stop}
```

where `start` and `stop` are in time relative to the start of the log.
For example, "give me all the first 1-second intervals of the logs" is

```
all/{:1}
```

Cut the first 3 seconds of all the logs:

```
all/{3:}
```

Give me the interval between 30 s and 35 s:

```
all/{30:35}
```

5.7. How to use a file from the cloud

This applies to any resource, not only logs.

First, put the file in the `duckiedown-data-2017` directory on Dropbox.

This is the link.

You need to ask Liam/Andrea because only them have write access.

Then, get the public link address and put it in the file `dropbox.urls.yaml` in the package `easy_node`. Remember to have `?dl=1` instead of `?dl=0` in the url.

In the code, use the function `require_resource()`:

```
from duckietown_utils import require_resource

zipname = require_resource('ii-datasets.zip')
```

The storage area is in `${DUCKIETOWN_ROOT}/cache/download`.

If the file is already downloaded, it is not downloaded again.

(So, if the file changes, you need to delete the cache directory. The best practice is to change the filename every time the file changes.)

The function `require_resource()` returns the path to the downloaded file.

Also note that you can put any URL in the file `dropbox.urls.yaml`; but the convention is that we only link things to Dropbox.

Unit M-6
Package easy_node

6.1. Package information

Link to package on Github

Essentials

Author: Andrea Censi (maintainer)

Description

`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.

6.2. 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.

This is the smallest example of an empty configuration:

```
description: My node
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 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

6.3. 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:

```python
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:

```yaml
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 6.6 - Benchmarking).
- 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 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()`.

6.4. 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:

```yaml
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:

```yaml
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.

6.5. Visualizing the configuration database

There are a few tolls 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


 Subcriptions

    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 [..]
```

6.6. 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
```

6.7. 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 Unit N-3 - Package `joy_mapper`.

6.8. Parameters and services defined for all packages

(Generated from configuration `easy_node.easy_node.yaml`.)
These are properties common to all nodes.

Parameters

No parameters defined.

Subscriptions

No subscriptions defined.

Publishers

No publishers defined.

6.9. Other ideas

(Add here other ideas that we can implement.)

UNIT M-7

Package `easy_regression`

7.1. Package information

Link to package on Github

Essentials

Author: Andrea Censi (maintainer)

Description

A package to make it easy to abstract about algorithm configuration and create regression tests.

7.2. Design goals

TODO: to write

7.3. Formalization

A regression test is defined by the following:

1. A set of logs on which to operate.
2. Zero or more "processors". A processor takes a bag as input and write a bag in output.
3. Zero or more "analyzers": these are objects that analyze the results.
4. Zero or more "acceptance conditions". These are the conditions on the results of the analyzers.

Let's go more in detail, to obtain a formalization that can be used to parallelize the processing.

This is in the spirit of a map/reduce framework.

➡ Map-reduce framework XXX

We assume that a processor has the following signature:

$$\textbf{processor} : \textbf{Bag} \to \textbf{Bag}$$

An analyzer is something that returns a statistics of type `Stat`:

$$\textbf{analyzer} : \textbf{Bag} \to \textbf{Stat}$$

We also assume to have an operation `merge` that allows to merge the results of two statistics:

$$\textbf{merge} : \textbf{Stat} \times \textbf{Stat} \to \textbf{Stat}$$

Finally, the acceptance condition is a test on the statistics. We want to be careful

about the possible conditions, so instead of having only true/false, we give the following results:

$$\textbf{acceptance} : \textbf{Stat} \to \{\texttt{ok}, \texttt{fail}, \texttt{nodata}, \texttt{abnormal}\}$$

The semantics is as follows:

- `ok` means that the conditions is satisfied;
- `fail` means that the condition is not satisfied;
- `nodata` means that there is not enough data to check the condition. For example, there are conditions that reference the results of regression tests at another date; that data might be not available;
- `abnormal` covers all abnormal conditions, including, for example, mistakes in writing the testing conditions.

The regression test engine does the following.

It reads the log test. It makes the logs available. (This might imply downloading the logs.)

A physical log is a physical .bag file. These are generated

```
regression1.regression_test.yaml

description:
Simple regression test.
a
logs:
- "vehicle:ferrari,length:<10"
processors:
- transformer: Identity
stats:
visualizers:



identity.job_processors.yaml
description:
constructor: IdentityProcessor
parameters:


count:


S is a dict of YAML

processor: Bag -> Bag
analyzer: Bag -> S
reduce: S x S -> S
display_test: S -> Display
```

A regression test is characterized by

7.4. Example of configuration file

Using an EasyAlgo configuration file, we can describe a simple regression tests that counts the number of messages in a bag in a file `example.regression_test.yaml` as follows:

```
description: Simple regression test
constructor: easy_regression.RegressionTest
parameters:
    logs:
    - 20160223-amadoa-amadobot-RCDP2
    processors: []
    analyzers:
    - count_messages
    checks:
    - desc: The number of messages read should remain the same.
      cond: |
        v:count_messages/20160223-amadoa-amadobot-RCDP2/num_messages == 5330
```

In this example, there is only one log, no processors, and one analyzer.

There is one condition, that checks that the output of `count_messages` on that log is 5330.

```
v:count_messages/20160223-amadoa-amadobot-RCDP2/num_messages == 5330
```

We can run this regression test using:

```
$ rosrun easy_regression run --tests example
```

7.5. Language for the conditions to check

There is a flexible language that allows to check conditions.

The simplest checks regarding checking the values:

```
v:analyzer/log/statistics == value
v:analyzer/log/statistics &gt;= value
v:analyzer/log/statistics &lt;= value
v:analyzer/log/statistics &lt; value
v:analyzer/log/statistics &gt; value
```

Check that it is in 10% of the value:

```
v:analyzer/log/statistics ==[10%] value
```

Use `@date` to reference the test result at that date. For example,

```
v:analyzer/log/statistics ==[10%] v:analyzer/log/statistic@2017-08-01
```

Use `~branch@date` to reference the value of a branch at a certain date:

```
v:analyzer/log/statistics ==[10%] v:analyzer/log/statistic~master@2017-08-01
```

Use `?commit` to reference the value of a branch at a specific commit:

```
v:analyzer/log/statistics ==[10%] v:analyzer/log/statistic?e9aa5f4
```

UNIT M-8

Package what_the_duck

8.1. Package information

Link to package on Github

Essentials

Author: Andrea Censi (maintainer)

Description

`what-the-duck` is a program that tests *dozens* of configuration inconsistencies that can happen on a Duckiebot.

3) The what-the-duck program

To use it, first compile the repository, and then run:

```
$ ./what-the-duck
```

+ comment

Where is this file? why not use rosrun so we can run from anywhere? LP

8.2. Adding more tests to what-the-duck

The idea is to add to `what-the-duck` all the tests that can be automated.

The documentation about to do that is not ready yet.

8.3. Tests already added

Here is the list of tests already added:

```
✓  Camera is detected
✓  Scipy is installed
✓  sklearn is installed
✓  Date is set correctly
✓  Not running as root
✓  Not running as ubuntu
✓  Member of group sudo
✓  Member of group input
✓  Member of group video
✓  Member of group i2c
✓  ~/.ssh exists
✓  ~/.ssh permissions
✓  ~/.ssh/config exists
✓  SSH option HostKeyAlgorithms is set
✓  At least one key is configured.
✓  ~/.ssh/authorized_keys exists
✓  Git configured
✓  Git email set
✓  Git name set
✓  Git push policy set
✓  Edimax detected
✓  The hostname is configured
✓  /etc/hosts is sane
✓  Correct kernel version
✓  Messages are compiled
✓  Shell is bash
✓  Working internet connection
✓  Github configured
✓  Joystick detected
✓  Environment variable DUCKIETOWN_ROOT
✓  ${DUCKIETOWN_ROOT} exists
✓  Environment variable DUCKIETOWN_FLEET
✓  ${DUCKIETOWN_FLEET} exists
✓  ${DUCKIETOWN_FLEET}/scuderia.yaml exists
✓  ${DUCKIETOWN_FLEET}/scuderia.yaml is valid
✓  machines file is valid
✓  Wifi network configured
✓  Python: No CamelCase
✓  Python: No tab chars
✓  Python:  No half merges
```

8.4. List of tests to add

Please add below any configuration test that can be automated:

- Editor is set to `vim`.
- Syntax on in `~/.vimrc`
- They put the right MAC address in the network configuration
- Ubuntu user is in group video, input, i2c (even if run from other user.)
- There is at least X.YGB of free disk space.

- If the SD is larger than 8GB, the disk has been resized.

Part N
Packages - Teleoperation

TODO: to write

UNIT N-1

Package `adafruit_drivers`

1.1. Package information

Link to package on Github

Essentials

Author: Dmitry Yershov

Maintainer: Mack

Description

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?

UNIT N-2

Package dagu_car

2.1. Package information

Link to package on Github

Essentials

Author: Dmitry Yershov
Author: Shih-Yuan Liu
Maintainer: Mack

Description

TODO: Add a description of package `dagu_car` in `package.xml`.

2.2. 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.

2.3. 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.

2.4. 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.

2.5. 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.

2.6. 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.

2.7. 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.

UNIT N-3

Package `joy_mapper`

3.1. Package information

Link to package on Github

Essentials

Maintainer: Mack

Description

The joy_mapper package for duckietown. Takes sensor_msgs.Joy and convert it to duckietown_msgs.CarControl.

3.2. Testing

Connect joystick.

Run:

```
$ roslaunch launch/joy_mapper_test.launch
```

The robot should move when you push buttons.

3.3. Dependencies

- `rospy`
- `sensor_msgs`: for the `Joy.msg`
- `duckietown_msgs`: for the `CarControl.msg`

3.4. 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 `steer_angle_gain`: `int`; default value: `1`

TODO: Missing description for entry "`steer_angle_gain`".

Parameter `simulated_vehicle_length`: `float`; default value: `0.18`

TODO: Missing description for entry "`simulated_vehicle_length`".

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 `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 `anti_instagram`: topic `anti_instagram_node/click` (`BoolStamped`)

TODO: Missing description for entry "`anti_instagram`".

Publisher `e_stop`: topic `wheels_driver_node/emergency_stop` (`BoolStamped`)

TODO: Missing description for entry "`e_stop`".

Publisher `avoidance`: topic `~start_avoidance` (`BoolStamped`)

TODO: Missing description for entry "`avoidance`".

UNIT N-4

Package `pi_camera`

4.1. Package information

Link to package on Github

Essentials

Maintainer: Andrea Censi

Description

TODO: Add a description of package `pi_camera` in `package.xml`.

4.2. 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.

4.3. Node `camera_node_continuous`

(Generated from configuration `camera_node_continuous.easy_node.yaml`.)

Camera driver.

Parameters

Parameter `framerate`: `float`; default value: `60.0`

Frame rate

Parameter `res_w`: `int`; default value: `320`

Resolution (width)

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`".

4.4. 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 computaon.

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.

4.5. 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.

4.6. 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 `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`".

Parameter `image_type`: `str`; default value: `'compressed'`

TODO: Missing description for entry "`image_type`".

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 O
Packages - Lane control

TODO: to write

UNIT O-1

Package `anti_instagram`

1.1. Package information

Link to package on Github

Essentials

Author: mfe

Maintainer: Mack

Description

TODO: Add a description of package `anti_instagram` in `package.xml`.

1.2. 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
```

1.3. Unit tests needed external files

These are other unittest that require the logs in DUCKIETOWN_DATA:

```
$ rosrun anti_instagram annotations_test.py
```

1.4. 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.

UNIT O-2

Package `complete_image_pipeline`

2.1. Package information

Link to package on Github

Essentials

Maintainer: Mack

Description

This package contains functions that allow to call the entire message pipeline programmatically, as a Python function, without having to instantiate any ROS node.

This is useful for regression tests, and quick tests with new logs.

2.2. Program `single_image_pipeline`

This runs the entire pipeline:

```
$ rosrun complete_image_pipeline single_image_pipeline --image image --line_detector
detector --image_pre image_prep
```

where `image` can be a filename or a URL.

For example, this:

```
$ rosrun complete_image_pipeline single_image_pipeline --image "https://www.dropbox.com/s/
bzezpw8ivlfu4b0/frame0002.jpg?dl=1"
```

results in an output as in Figure 2.1.

Figure 2.1. Output of `single_image_pipeline`

UNIT O-3

Package `ground_projection`

3.1. Package information

Link to package on Github

Essentials

Author: Changhyun Choi

Maintainer: Mack

Description

TODO: Add a description of package `ground_projection` in `package.xml`.

3.2. Node `ground_projection_node`

(Generated from configuration `ground_projection.easy_node.yaml`.)

TODO: node description for ground_projection

Parameters

No parameters defined.

Subscriptions

No subscriptions defined.

Publishers

No publishers defined.

Unit O-4
Package `lane_control`

4.1. Package information

Link to package on Github

Essentials

Author: steven
Author: Shih-Yuan Liu
Maintainer: Liam Paull

Description

TODO: Add a description of package `lane_control` in `package.xml`.

4.2. `lane_controller_node`

(Generated from configuration `lane_controller_node.easy_node.yaml`.)

> **Note:** there is some very funny business inside. It appears that `k_d` and `k_theta` are switched around.

Parameters

Parameter `v_bar`: `float`
Nominal linear velocity (m/s).
Parameter `k_theta`: `float`
Proportional gain for $\boldsymbol{\theta}$.
Parameter `k_d`: `float`
Propertional gain for $\boldsymbol{d}$.
Parameter `d_thres`: `float`
Cap for error in $\boldsymbol{d}$.
Parameter `theta_thres`: `float`
Maximum desired $\boldsymbol{\theta}$.
Parameter `d_offset`: `float`
A configurable offset from the lane position.

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`".

Unit O-5
Package `lane_filter`

> Assigned to: Liam

5.1. Package information

Link to package on Github

Essentials

Maintainer: Liam Paull

Description

TODO: Add a description of package `lane_filter` in `package.xml`.

5.2. `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 `mean_d_0`: `float`; default value: `0.0`

TODO: Missing description for entry "`mean_d_0`".

Parameter `mean_phi_0`: `float`; default value: `0.0`

TODO: Missing description for entry "`mean_phi_0`".

Parameter `sigma_d_0`: `float`; default value: `0.0`

TODO: Missing description for entry "`sigma_d_0`".

Parameter `sigma_phi_0`: `float`; default value: `0.0`

TODO: Missing description for entry "`sigma_phi_0`".

Parameter `delta_d`: `float`; default value: `0.02`
(meters)

Parameter `delta_phi`: `float`; default value: `0.0`
(radians)

Parameter `d_max`: `float`; default value: `0.5`

TODO: Missing description for entry "`d_max`".

Parameter `d_min`: `float`; default value: `-0.7`

TODO: Missing description for entry "`d_min`".

Parameter `phi_min`: `float`; default value: `-1.5707`

TODO: Missing description for entry "`phi_min`".

Parameter `phi_max`: `float`; default value: `1.5707`

TODO: Missing description for entry "`phi_max`".

Parameter `cov_v`: `float`; default value: `0.5`

Linear velocity "input".

> XXX which units?

Parameter `cov_omega`: `float`; default value: `0.01`

Angular velocity "input".

> XXX which units?

Parameter `linewidth_white`: `float`; default value: `0.04`

TODO: Missing description for entry "`linewidth_white`".

Parameter `linewidth_yellow`: `float`; default value: `0.02`

TODO: Missing description for entry "`linewidth_yellow`".

Parameter `lanewidth`: `float`; default value: `0.4`

TODO: Missing description for entry "`lanewidth`".

Parameter `min_max`: `float`; default value: `0.3`

Expressed in nats.

Parameter `use_distance_weighting`: `bool`; default value: `False`

For use of distance weighting (dw) function.

Parameter `zero_val`: `float`; default value: `1.0`

TODO: Missing description for entry "`zero_val`".

Parameter `l_peak`: `float`; default value: `1.0`

TODO: Missing description for entry "`l_peak`".

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 `use_max_segment_dist`: `bool`; default value: `False`

For use of maximum segment distance.

Parameter `max_segment_dist`: `float`; default value: `1.0`

For use of maximum segment distance.

Parameter `use_min_segs`: `bool`; default value: `False`

For use of minimum segment count.

Parameter `min_segs`: `int`; default value: `10`

For use of minimum segment count.

Parameter `use_propagation`: `bool`; default value: `False`

For propagation.

Parameter `sigma_d_mask`: `float`; default value: `0.05`

TODO: Missing description for entry "`sigma_d_mask`".

Parameter `sigma_phi_mask`: `float`; default value: `0.05`

TODO: Missing description for entry "`sigma_phi_mask`".

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 `lane_pose`: topic `~lane_pose` (`LanePose`)

TODO: Missing description for entry "`lane_pose`".

Publisher `belief_img`: topic `~belief_img` (`Image`)

TODO: Missing description for entry "`belief_img`".

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`".

Publisher `switch`: topic `~switch` (`BoolStamped`)

TODO: Missing description for entry "`switch`".

UNIT O-6

Package `line_detector2`

6.1. Package information

Link to package on Github

Essentials

Author: Andrea Censi (maintainer)

Description

A re-implementation of the line detector node.

This is a re-implementation of the package `line_detector` using the new facilities provided by `easy_node`.

6.2. 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** if the output bag.

> **Note:** you always need to use absolute paths for bag files.

You can let this run for a few seconds, then stop using Ctrl-C.

You can then inspect the result using:

```
$ roscore &
$ rosbag play -l bag out
$ rviz &
```

In `rviz` click "add", click "by topic" tab, expand "`line_detector`" and click "`image_with_lines`".

Observe on the result that:

1. There are *lots* of detections.
2. Predominantly white detections (indicated in black) are on white lines, yellow detections (shown in blue) are on blue lines, and red detections (shown in green) are on red lines.

These are some sample logs on which to try:

```
$ wget -O 160122-manual1_ferrari.bag https://www.dropbox.com/s/8bpi656j7qox5kv?dl=1

https://www.dropbox.com/s/vwznjke4xvnhi9o/160122_manual2-ferrari.bag?dl=1

https://www.dropbox.com/s/y7ulj198punj0mp/160122_manual3_corner-ferrari.bag?dl=1

https://www.dropbox.com/s/d4n9otmlans4i62/160122-calibration-good_lighting-tesla.bag?dl=1
```

Sample output:

```
From cmd:roslaunch duckietown camera.launch veh:=${VEHICLE_NAME}
[INFO] [WallTime: 1453839555.948481] [LineDetectorNode] number of white lines = 14
[INFO] [WallTime: 1453839555.949102] [LineDetectorNode] number of yellow lines = 33
[INFO] [WallTime: 1453839555.986520] [LineDetectorNode] number of white lines = 18
[INFO] [WallTime: 1453839555.987039] [LineDetectorNode] number of yellow lines = 34
[INFO] [WallTime: 1453839556.013252] [LineDetectorNode] number of white lines = 14
[INFO] [WallTime: 1453839556.013857] [LineDetectorNode] number of yellow lines = 29
[INFO] [WallTime: 1453839556.014539] [LineDetectorNode] number of red lines = 2
[INFO] [WallTime: 1453839556.047944] [LineDetectorNode] number of white lines = 18
[INFO] [WallTime: 1453839556.048672] [LineDetectorNode] number of yellow lines = 28
[INFO] [WallTime: 1453839556.049534] [LineDetectorNode] number of red lines = 2
[INFO] [WallTime: 1453839556.081400] [LineDetectorNode] number of white lines = 13
[INFO] [WallTime: 1453839556.081944] [LineDetectorNode] number of yellow lines = 34
[INFO] [WallTime: 1453839556.082479] [LineDetectorNode] number of red lines = 1
```

The output from `rviz` looks like Figure 6.1.

Figure 6.1. RViz output

6.3. Quantitative tests

TODO: Something more quantitative (to be filled in by Liam or Hang)

6.4. 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 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.

Parameter img_size: not known

TODO: Missing description for entry "img_size".

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.

Subscriptions

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 Unit O-1 - Package anti_instagram.

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.

Publishers

Publisher edge: topic ~edge (Image)

TODO: Missing description for entry "edge".

Publisher color_segment: topic ~colorSegment (Image)

TODO: Missing description for entry "color_segment".

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".

UNIT O-7

Package `line_detector`

7.1. Package information

Link to package on Github

Essentials

Author: Hang Zhao

Maintainer: Mack

Description

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.

Part P
Packages - Indefinite navigation

TODO: to write

Unit P-1
AprilTags library

1.1. Package information

Link to package on Github

Essentials

Author: Michael Kaess

Author: Hordur Johannson

Maintainer: Mitchell Wills

Description

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)

UNIT P-2

Package `apriltags_ros`

AprilTags for ROS.

build passing

+ Resource error

```
I will not embed remote files, such as https://api.travis-ci.org/RIVeR-Lab/
apriltags_ros.png:
```

2.1. Package information

Link to package on Github

Essentials

Author: Mitchell Wills

Maintainer: Mitchell Wills

Description

A package that provides a ROS wrapper for the apriltags C++ package

Unit P-3
Package fsm

3.1. Package information

Link to package on Github

Essentials

Author: Michael Misha Novitzky
Maintainer: Mack

Description

The finite state machine coordinates the modes of the car.

3.2. 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 `global_transitions`: `dict`; default value: ``

TODO: Missing description for entry "`global_transitions`".

Parameter `initial_state`: `str`; default value: `''`

TODO: Missing description for entry "`initial_state`".

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`".

Unit P-4
Package `indefinite_navigation`

4.1. Package information

Link to package on Github

Essentials

Maintainer: Liam Paull

Description

TODO: Add a description of package `indefinite_navigation` in `package.xml`.

UNIT P-5

Package `intersection_control`

5.1. Package information

Link to package on Github

Essentials

Author: Michael Misha Novitzky

Maintainer: Mack

Description

The intersection_control package implements a dead reckoning controller until we do something smarter.

UNIT P-6

Package `navigation`

6.1. Package information

Link to package on Github

Essentials

Author: igor

Maintainer: Mack

Description

TODO: Add a description of package `navigation` in `package.xml`.

UNIT P-7

Package `stop_line_filter`

7.1. Package information

Link to package on Github

Essentials

Maintainer: Liam Paull

Description

TODO: Add a description of package `stop_line_filter` in `package.xml`.

Part Q
Packages - Localization and planning

TODO: to write

UNIT Q-1

Package `duckietown_description`

1.1. Package information

Link to package on Github

Essentials

Author: Teddy Ort

Maintainer: Mack

Description

TODO: Add a description of package `duckietown_description` in `package.xml`.

UNIT Q-2

Package `localization`

2.1. Package information

Link to package on Github

Essentials

Author: teddy

Maintainer: Mack

Description

TODO: Add a description of package `localization` in `package.xml`.

Part R
Packages - Coordination

TODO: to write

UNIT R-1
Package `led_detection`

1.1. Package information

Link to package on Github

Essentials

Maintainer: Andrea Censi

Description

TODO: Add a description of package `led_detection` in `package.xml`.

TODO: add authors

1.2. 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:

```
$ roslaunch duckietown camera.launch veh:=robot name
```

In a separate terminal, fire up the LED detector and the custom GUI by running:

```
$ roslaunch led_detector LED_detector_with_gui.launch veh:=robot name
```

> Note: to operate without a GUI:

```
$ roslaunch led_detector LED_detector.launch veh:=robot name
```

The LED_detector_node will be launched on the robot, while LED_visualizer (a simple GUI) will be started on your laptop. Make sure the camera image from the observer-bot is visualized and updated in the visualizer (tip: check that your camera cap is off).

Hit on Detect and wait to trigger a detection. This will not have any effect if LED_detector_node is not running on the duckiebot (it is included in the above launch file). After the capture and processing phases, the outcome will look like:

The red numbers represent the frequencies directly inferred from the camera stream, while the selected detections with the associated signaling frequencies will be displayed in green. You can click on the squares to visualize the brightness signals and the Fourier amplitude spectra of the corresponding cells in the video stream. You can also click on the camera image to visualize the variance map.

1.3. Unit tests

To run the unit tests for the LED detector, you need to have the F23 rosbags on you hard disk. These bag files should be synced from [this dropbox link](https://www.dropbox.com/sh/5kx8qwgttu69fhr/AAASLpOVjV5r1xpzeW7xWZh_a?dl=0).

For the test to locate the bag files, you should have the `DUCKIETOWN_DATA` environment variable set, pointing to the location of you duckietown-data folder. This can be achieved by:

```
$ export DUCKIETOWN_DATA=local-path-to-duckietown-data-folder
```

All the available tests are specified in file `all_tests.yaml` in the scripts/ folder of the package led_detection in the duckietown ROS workspace. To run these, use the command:

```
$ rosrun led_detection unittests.py algorithm name-of-test
```

Currently, `algorithm` can be either 'baseline' or 'LEDDetector_plots' to also display the plot in the process.

To run all test with all algorithms, execute:

```
$ rosrun led_detection unittests.py '*' '*'
```

More in general:

```
$ rosrun led_detection unittests.py tests algorithms
```

where:

- `tests` is a comma separated list of algorithms. May use "`*`".
- `algorithms` is a comma separated list of algorithms. May use "`*`".

The default algorithm is called "`baseline`", and its tests are invoked using:

```
$ rosrun led_detection <script> '*' 'baseline'
```

Unit R-2
Package led_emitter

2.1. Package information

Link to package on Github

Essentials

Maintainer: Andrea Censi

Description

TODO: description for `led_emitter` package.

2.2. 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:

```
$ roslaunch led_joy_mapper led_joy_with_led_emitter_test.launch veh:=robot name
```

This launches the joy controller, the mapper controller, and the led emitter nodes. You should not need to run anything external for this to work. Use the joystick buttons A, B and C to change your duckiebot's LED's blinking frequency.

Button A broadcasts signal CAR_SIGNAL_A (2.8hz), button B broadcasts signal CAR_SIGNAL_B (4.1hz), and button CAR_SIGNAL_C (Y on the controller) broadcasts signal C(5hz).The LB button will make the LEDs all white, the RB button will make some LEDs blue and some LEDs green, and the logitek button (middle button) will make the LEDs all red

Repeat this for each vehicle at the intersection that you wish to be blinking. Use previous command replacing `robot name` the names of the vehicles and try command different blinking patterns on different duckiebots.

(optional tests) For a grasp of the low level LED emitter, run:

```
$ roslaunch led_emitter led_emitter_node.launch veh:=robot name
```

You can then publish to the topic manually by running the following command in another screen on the duckiebot:

```
$ rostopic pub /robot name/led_emitter_node/change_to_state std_msgs/Float32 float-value
```

Where `float-value` is the desired blinking frequency, e.g. 1.0, .5, 3.0, etc. If you wish to run the LED emitter test, run the following:

```
$ roslaunch led_emitter led_emitter_node_test.launch veh:=robot name
```

This will cycle through frequencies of 3.0hz, 3.5hz, and 4hz every 5 seconds. Once done, kill everything and make sure you have joystick control as described above.

UNIT R-3

Package `led_interpreter`

3.1. Package information

Link to package on Github

Essentials

Author: qlai

Maintainer: Mack

Description

TODO: Add a description of package `led_interpreter` in `package.xml`.

UNIT R-4

Package `led_joy_mapper`

4.1. Package information

Link to package on Github

Essentials

Author: lapentab

Maintainer: Mack

Description

TODO: Add a description of package `led_joy_mapper` in `package.xml`.

Unit R-5
Package `rgb_led`

5.1. Package information

Link to package on Github

Essentials

Author: Dmitry Yershov
Maintainer: Mack

Description

TODO: Add a description of package `rgb_led` in `package.xml`.

5.2. 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
```

UNIT R-6

Package `traffic_light`

6.1. Package information

Link to package on Github

Essentials

Maintainer: Mack

Description

TODO: Add a description of package `traffic_light` in `package.xml`.

Part S
Packages - Additional functionality

TODO: to write

UNIT S-1

Package mdoap

1.1. Package information

Link to package on Github

Essentials

Author: Catherine Liu

Author: Sam

Author: Ari

Maintainer: Mack

Description

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

UNIT S-2

Package `parallel_autonomy`

2.1. Package information

Link to package on Github

Essentials

Maintainer: Liam Paull

Description

TODO: description for the parallel autonomy package

UNIT S-3

Package `vehicle_detection`

3.1. Package information

Link to package on Github

Essentials

Author: Andrea Tacchetti

Maintainer: Mack

Description

TODO: Add a description of package `vehicle_detection` in `package.xml`.

Part T
Packages - Templates

These are templates.

UNIT T-1

Package `pkg_name`

1.1. Package information

Link to package on Github

Essentials

Author: Shih-Yuan Liu

Maintainer: Andrea Censi

Description

The package `pkg_name` is a template for ROS packages.

For the tutorial, see Unit J-6 - Minimal ROS node - `pkg_name`.

1.2. How to use this template

This package is a template that contains code

UNIT T-2

Package `rostest_example`

2.1. Package information

Link to package on Github

Essentials

Author: Teddy Ort

Maintainer: Mack

Description

TODO: Add a description of package `rostest_example` in `package.xml`.

Part U
Packages - Convenience

These packages are convenience packages that group together launch files and tests.

UNIT U-1

Package duckie_rr_bridge

1.1. Package information

Link to package on Github

Essentials

Author: greg

Maintainer: Mack

Description

The duckie_rr_bridge package. Creates a Robot Raconteur Service to drive the Duckiebot.

UNIT U-2

Package duckiebot_visualizer

2.1. Package information

Link to package on Github

Essentials

Author: Shih-Yuan Liu

Maintainer: Mack

Description

TODO: Add a description of package `duckiebot_visualizer` in `package.xml`.

2.2. 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`".

UNIT U-3

Package duckietown_demos

3.1. Package information

Link to package on Github

Essentials

Maintainer: Liam Paull

Description

TODO: description of `duckietown_demos`

UNIT U-4

Package duckietown_logs

4.1. Package information

Link to package on Github

Essentials

Maintainer: Andrea Censi

Description

Package to create videos from ROS logs.

4.2. 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
```

UNIT U-5

Package duckietown_unit_test

5.1. Package information

Link to package on Github

Essentials

Maintainer: Mack

Description

The `duckietown_unit_test` meta package contains all the unit test launch files for Duckietown.

UNIT U-6

Package `veh_coordinator`

6.1. Package information

Link to package on Github

Essentials

Author: Michal Cap

Maintainer: Mack

Description

The simple vehicle coordination package

I think is used to fake the vehicle coordination for the FSM.

Unit U-7
Last modified

- Tue, Sep 26: Unit I-18 - Accessing computers using SSH (Liam Paull)
- Tue, Sep 26: Unit C-8 - Duckiebot Initialization (Liam Paull)
- Tue, Sep 26: Unit G-11 - Duckiebot modeling (Andrea Censi)
- Tue, Sep 26: Unit C-9 - Networking aka the hardest part (Liam Paull)
- Tue, Sep 26: Unit L-14 - Checkoff: Duckiebot Assembly and Configuration (liampaull)
- Tue, Sep 26: Unit L-6 - Git usage guide for Fall 2017 (liampaull)
- Tue, Sep 26: Unit L-7 - Organization (liampaull)
- Tue, Sep 26: Unit L-15 - Checkoff: Take a Log (liampaull)
- Tue, Sep 26: Unit L-16 - Homework: Data Processing (liampaull)
- Tue, Sep 26: Unit C-10 - Software setup and RC remote control (Liam Paull)
- Tue, Sep 26: Unit L-12 - Chicago branch Diary (Matthew Walter)
- Tue, Sep 26: Unit L-11 - Montréal branch diary (liampaull)
- Tue, Sep 26: Unit C-2 - Acquiring the parts for the Duckiebot `DB17-wjd` (Jacopo Tani)
- Tue, Sep 26: Unit C-3 - Soldering boards for `DB17` (Jacopo Tani)
- Tue, Sep 26: Unit C-5 - Assembling the Duckiebot `DB17` (Jacopo Tani)
- Mon, Sep 25: Unit L-2 - First Steps in Duckietown (Kirsten Bowser)
- Mon, Sep 25: Unit L-10 - Zurich branch diary (Andrea Censi)
- Mon, Sep 25: Unit L-3 - Logistics for Zürich branch (Andrea Censi)
- Mon, Sep 25: Unit L-4 - Logistics for Montréal branch (Andrea Censi)
- Mon, Sep 25: Unit L-5 - Logistics for Chicago branch (Andrea Censi)
- Mon, Sep 25: Unit E-4 - Bumper Assembly (Jacopo Tani)
- Mon, Sep 25: Unit C-1 - Duckiebot configurations (Jacopo Tani)
- Mon, Sep 25: Unit C-17 - Taking a log (liampaull)
- Sun, Sep 24: Part C - Operation manual - Duckiebot (Jacopo Tani)
- Sun, Sep 24: Unit C-4 - Preparing the power cable for `DB17` (Jacopo Tani)
- Sun, Sep 24: Unit E-1 - Acquiring the parts for the Duckiebot `DB17-lc` (Jacopo Tani)
- Sun, Sep 24: Unit E-2 - Soldering boards for `DB17-l` (Jacopo Tani)
- Sun, Sep 24: Unit E-3 - Assembling the Duckiebot `DB17-wjdlc` (Jacopo Tani)
- Sun, Sep 24: Unit C-16 - Camera calibration (Jacopo Tani)
- Sun, Sep 24: Unit C-6 - Reproducing the image (Jacopo Tani)
- Sun, Sep 24: Unit C-7 - Installing Ubuntu on laptops (Jacopo Tani)
- Sun, Sep 24: Unit C-11 - Reading from the camera (Jacopo Tani)
- Sun, Sep 24: Unit C-12 - RC control launched remotely (Jacopo Tani)
- Sun, Sep 24: Unit C-13 - RC+camera remotely (Jacopo Tani)
- Sun, Sep 24: Unit C-14 - Interlude: Ergonomics (Jacopo Tani)
- Sun, Sep 24: Unit C-15 - Wheel calibration (Jacopo Tani)
- Sun, Sep 24: Part E - Operation manual - Duckiebot with LEDs (Jacopo Tani)
- Sun, Sep 24: Unit E-5 - `C1` (LEDs) setup (Jacopo Tani)
- Sun, Sep 24: Unit L-9 - Slack Channels (Jacopo Tani)
- Sun, Sep 24: Unit H-3 - Exercise: Simple data analysis from a bag (Andrea F. Daniele)
- Sun, Sep 24: Unit H-5 - Exercise: Bag thumbnails (Andrea F. Daniele)
- Sun, Sep 24: Unit H-6 - Exercise: Instagram filters (Andrea F. Daniele)

- Sun, Sep 24: Unit J-9 - Duckietown utility library (Andrea F. Daniele)
- Sun, Sep 24: Unit H-1 - Exercise: Basic image operations (Andrea Censi)
- Sun, Sep 24: Unit H-2 - Exercise: Basic image operations, adult version (Andrea Censi)
- Sun, Sep 24: Unit H-4 - Exercise: Bag in, bag out (Andrea Censi)
- Sun, Sep 24: Unit H-7 - Exercise: Bag instagram (Andrea Censi)
- Sun, Sep 24: Unit H-8 - Exercise: Live Instagram (Andrea F. Daniele)
- Sun, Sep 24: Unit J-1 - Python (Andrea F. Daniele)
- Sat, Sep 23: Unit G-7 - Modern signal processing (Andrea Censi)
- Sat, Sep 23: Part L - Fall 2017 (Andrea Censi)
- Sat, Sep 23: Unit L-1 - The Fall 2017 Duckietown experience (Andrea Censi)
- Sat, Sep 23: Unit L-13 - Guide for TAs (Andrea Censi)
- Sat, Sep 23: Unit L-17 - Guide for mentors (Andrea Censi)
- Sat, Sep 23: Unit I-29 - ROS installation and reference (Andrea F. Daniele)
- Sat, Sep 23: Unit F-21 - Time series (Andrea Censi)
- Sat, Sep 23: Unit F-24 - Computer science concepts (Andrea Censi)
- Fri, Sep 22: Unit A-2 - Duckietown history and future (Andrea Censi)
- Thu, Sep 21: Part G - A course in autonomy (Andrea Censi)
- Thu, Sep 21: Part H - Exercises (Andrea Censi)
- Thu, Sep 21: Unit H-9 - Exercise: Git and conventions (Andrea Censi)
- Thu, Sep 21: Unit H-27 - Exercise: Who watches the watchmen? (optional) (Andrea Censi)
- Thu, Sep 21: Unit F-22 - Transformations (Falcon Dai)
- Thu, Sep 21: Unit F-1 - Chapter template (Jacopo Tani)
- Thu, Sep 21: Unit F-15 - From ODEs to LTI systems (Jacopo Tani)
- Thu, Sep 21: Unit G-12 - Odometry Calibration (Jacopo Tani)
- Thu, Sep 21: Unit F-18 - Dynamics (Jacopo Tani)
- Thu, Sep 21: Unit F-5 - Complex numbers (Jacopo Tani)
- Thu, Sep 21: Unit F-6 - Linearity and Vectors (Jacopo Tani)
- Thu, Sep 21: Unit F-16 - Probability basics (Jacopo Tani)
- Thu, Sep 21: Unit F-17 - Kinematics (Jacopo Tani)
- Wed, Sep 20: Unit G-2 - Autonomy overview (Jon Michaux)
- Tue, Sep 19: Unit F-3 - Sets (Jacopo Tani)
- Mon, Sep 18: Unit G-4 - System architectures basics (Andrea Censi)
- Mon, Sep 18: Unit F-20 - Reference frames (Falcon Dai)
- Sun, Sep 17: Unit B-4 - Using LaTeX constructs in documentation (Andrea Censi)
- Sun, Sep 17: Unit F-19 - Coordinate systems (Andrea Censi)
- Sun, Sep 17: Unit G-5 - Autonomy architectures (Andrea Censi)
- Sun, Sep 17: Unit B-10 - Learning in Duckietown (Andrea Censi)
- Sun, Sep 17: Unit I-23 - Liclipse (Andrea Censi)
- Sun, Sep 17: Unit F-7 - Matrices basics (Jacopo)
- Sat, Sep 16: Unit B-8 - The Duckuments bot (Andrea Censi)
- Sat, Sep 16: Unit G-3 - Modern Robotic Systems (Andrea Censi)
- Fri, Sep 15: Unit B-5 - Advanced Markduck guide (Andrea Censi)
- Thu, Sep 14: Unit G-6 - Representations (Matthew Walter)
- Thu, Sep 14: Unit G-1 - Autonomous Vehicles (Andrea Censi)
- Thu, Sep 14: Unit G-8 - Middleware (Andrea Censi)
- Thu, Sep 14: Unit M-6 - Package `easy_node` (Andrea Censi)
- Wed, Sep 13: Unit F-8 - Matrix inversions (Andrea Censi)
- Wed, Sep 13: Unit F-9 - Matrices and vectors (Andrea Censi)

- Wed, Sep 13: Unit F-10 - Matrix operations (basic) (Andrea Censi)
- Wed, Sep 13: Unit F-13 - Linearization (Andrea Censi)
- Wed, Sep 13: Unit F-14 - Norms (Andrea Censi)
- Wed, Sep 13: Unit G-9 - Modularization and Contracts (Andrea Censi)
- Wed, Sep 13: Unit G-10 - Configuration management (Andrea Censi)
- Wed, Sep 13: Unit G-13 - Computer vision basics (Andrea Censi)
- Wed, Sep 13: Unit G-15 - Camera calibration (Andrea Censi)
- Wed, Sep 13: Unit G-16 - Image filtering (Andrea Censi)
- Wed, Sep 13: Unit G-17 - Illumination invariance (Andrea Censi)
- Wed, Sep 13: Unit G-18 - Line Detection (Andrea Censi)

Page left blank