EZ-RASSOR Onboarding

Follow the steps below before you start working on the EZ-RASSOR. Tackle them in order and assume that each is absolutely required unless otherwise specified.

1. Install Ubuntu 18.04 (Bionic Beaver)

The EZ-RASSOR runs primarily on Ubuntu 18.04 LTS because ROS (a major part of our technology stack) only officially supports Ubuntu Long-Term Support (LTS) releases. Install Ubuntu 18.04 on a virtual machine or install it on bare metal (directly on your system).

It is *highly* preferred that you choose the latter for several reasons:

1. More of your system's resources will be available for the simulation.
2. The simulation works better when it can directly interface with physical hardware.
3. Bare metal installs are easier to perform.
4. Developing on a local machine is much easier.
5. A bare metal install forces you to become familiar with Linux faster.
6. You should already be using Linux for everything anyway. ;)

2. Become Comfortable with Python

Our system is written almost entirely in Python2. Start learning Python with this YouTube series (and/or this condensed video). Practice writing code within your Ubuntu install so you become familiar with Ubuntu and Python at the same time. *This step is not required if you already know Python.*

Here are some exercises for practice:

1. Write a Python function that calculates the Nth Fibonacci number in O(1) space and without recursion.

2. Create a Deque (doubly-linked list) Python class that supports append, prepend, search, and removal from the front and back of the list.

3. Become familiar with Python modules and create a simple Caesar Cipher module.

4. Write a Python list comprehension that satisfies these conditions:
    1. Start with the range of numbers [0, 100].
    2. Cube each number.
    3. Remove any even cubes except cubes under and including the number 50.

5. Write a generator function that takes an arbitrary number of iterables and interweaves them. Here's an example:

```
iterable1 = [1, 10, 20]
iterable2 = (1.0, 2.0, 3.14, 1.337)
iterable3 = “string”
print(list(your_function(iterable1, iterable2, iterable3)))
# prints [1, 1.0, “s”, 10, 2.0, “t”, 20, 3.14, “r”, 1.337, “i”, “n”, “g”]
```

3. Read PEP8
This short document describes preferred Python style and is widely accepted by the global Python community. All code in this project **must** be PEP8 compliant. Grossly non-compliant code will be rejected. Don't waste your own time by writing non-PEP8 Python.

4. Become Comfortable with POSIX/Shell/Linux
This project includes several shell scripts that make the development process easier. Before you read those scripts or try to use them, familiarize yourself with shell scripting and POSIX compliance. Pick up some basic Linux commands along the way. *This step is not required if you already know how to write **good** shell scripts.*

Here are some exercises for practice:

1. Write "Hello World" in Java, then create a shell script that automatically builds the program into a separate `build` directory. Next, have your script run the program and delete the contents of the `build` directory afterwards.

2. Allow users to pass flags to the above script like `--build`, `--run`, or `--clean`. This lets users execute only the parts of the process that they want to execute.

3. Write a script that plays the FizzBuzz game for a given range of numbers.

4. Write a script that generates 100 files which each contain 10000 garbage alphanumeric characters, then determine how many times the combination "atoz" appears in these files using this famous Linux command.

5. Become Comfortable with reStructuredText
We use reStructuredText (rST) for all of our important documentation (readme, guidelines, wiki pages, etc). This markup language is similar to Markdown, but with a few small differences that make it more powerful and beautiful. Briefly review the specification for reStructuredText. Any changes you make to our documentation must be done using rST, not Markdown. *This step is not required if you already know how to write in rST.*

6. Install Blender
Blender is a 3D modeling program that our team uses to make changes to our simulation robot model. Install it on your Ubuntu machine and check out this tutorial series. You should also take a look at this wiki page and play around with the model in Blender after you finish this guide.

7. Install ROS
This project is made up of about a dozen or so small programs that perform simple tasks like rotating the robot's wheels or listening for HTTP requests from the mobile application. All of these programs communicate with each other via ROS: the Robot Operating System. Read more about ROS and install ROS Melodic on your Ubuntu machine. Note that our installation script provides an easy way to install ROS quickly for end users, but you should install it manually the first time so that you learn about the process.

8. Read the ROS Wiki
Read the introduction pages of the ROS wiki and work through the tutorial (on your Ubuntu machine). The tutorial will help you set up a Catkin workspace (where you can write all of your practice code).

After you've completed the tutorial, take a crack at this exercise:

1. Create a new Catkin package in the workspace you set up during the tutorial.
2. Write a node that publishes every year in the range [1337, 9001] to the topic `/years`.
3. Write another node that listens to `/years` and sends all leap years to `/leap_years`.
4. Write a third node that prints each year published in `/leap_years` to the screen.
5. Write a launch file that starts all three nodes.

9. Read the EZ-RASSOR Wiki
Our wiki contains information that will help you understand our system's overall architecture and each individual ROS node within the system. Read **every** page of the wiki in the order implied by the wiki sidebar (from top to bottom).

10. Review Additional EZ-RASSOR Documentation
Read our contributing guidelines, the readme, and the license (but don't install anything yet). Feel free to skim the design document but understand that most of this document is no longer relevant. *The wiki is the authoritative, up-to-date description of our project.*

11. Install the EZ-RASSOR Project
Installation instructions are located in the contributing guidelines. **Do not use the readme instructions** (they do not properly set up the development environment required to make code changes).

12. Become Comfortable with the Project
Now that the project is present on your machine, read all of the code for every package, read the mobile application code, and read the shell scripts. If you're inclined, review this old autonomy code that got cut from the final version (it may provide inspiration).

13. Install the EZ-RASSOR Mobile Application
Our application can be found in the Google Play Store. Review this wiki page if you plan to make changes to the application.

14. Manually Test the Project
Finally, manually test the system. Ensure that you've properly set up your environment by following the instructions in the contributing guidelines. You also need to build the packages with the development script. You might have to source the `setup.bash` file inside `~/.workspace/devel` as well. With all of that done, go through each of the examples on this wiki page to manipulate the EZ-RASSOR in the simulation. You should also try operating the robot by directly publishing to the movement topics when the system is running, and by sending HTTP requests to the controller server with curl.