Untitled

Jonathan Garcia-Mallen

??? August 2016

Contents

1 Background

Duckietown (2.166) is a graduate class on advanced autonomy taught at MIT, first taught in Spring 2016. It is a hands-on, project-focused course focusing on self-driving vehicles and high-level autonomy. Its students work to solve the underlying problem of designing the Autonomous Robo-Taxis System for the (fictional) City of Duckietown. Its students are diverse, coming from multiple departments with different backgrounds.

With this diversity in mind, the first week ***or two*** is dedicated to bringing everyone on the same page, and doling out the robo-taxis to be programmed: Duckiebots. A Raspberry Pi 2 is at the center of these machines *(cite duck site)*. To program them, students learn to log in remotely from their laptops to the robot's Pi and launch programs the same way, or by sending a command directly from their laptops without logging into their robot. The students had no way of running a program on their duckiebots without using their laptops.

Picture this scenario. A grad student is testing some new autonomy on their laptop. Their reseach advisor *(is this the right word?)* happens to walk by on her way to a meeting and asks him "how's your duckiebot doing?" The student rushes to power on his robot taxi (named batmobile), wait for it to connect to the network, and rushes the following incantation into his laptop's terminal: *(change 'dat-grad-student' to 'you'?)*

```
dat-grad-student@duckietop4:~$ ssh batmobile
ssh: Could not resolve hostname batmobile.local: Name or service not known
dat-grad-student@duckietop4:~$ ping batmobile.local
ping: unknown host batmobile.local
dat-grad-student@duckietop4:~$ ping batmobile.local
PING batmobile.local (18.314.159.265)
64 bytes from localhost (18.314.159.265): icmp_seq=1 ttl=64 time=396.039 ms
64 bytes from localhost (18.314.159.265): icmp_seq=2 ttl=64 time=F I X
T H E S E   N U M B E R S
^C
--- localhost ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.022/0.029/0.039/0.008 ms
dat-grad-student@duckietop4:~$ ssh batmobile
.
.
.
# F I L L   T H E   R E S T   O F   T H I S
ubuntu@batmobile:~$ roslaunch duckietown dat-grad-students-demo.launch veh:=batmobile

or he does the other thing where he still has to ssh in but still sees all the above er

you know, the remote roslaunch
dad-grad-student@duckietop4:~$ roslaunch duckietown dat-grad-students-demo.launch veh:=
```

This is clearly a worst-case scenario. It is not the only scenario. A well-planned demo for a barely-technical audience would demand questions such as "Why do you need a laptop, if this is an autonomous vehicle?" or "Is the code running on the robot, or your computer?" And in any case, a laptop and the corresponding WiFi network necessary is yet another potential point of failure. A laptop should not be necessary in order to begin an autnonmous routine on the duckiebot. This 6.UAP project remedies this.

2 Requirements and Design Goals

The purpose of this project is to create a quick and easy means to start any ROS program on the duckiebot. There is a clear primary requirement: this system must let the user (researcher or student) start a program of their choosing on the duckiebot without using any device external to the duckiebot. The duckiebot has two inputs that may be considered: a Raspberry Pi Cam 2, and a Logitech Joystick controler.

Three goals guided the fulfillment of this requirement. The system must be reliable. It cannot fail when the user is in front of an audience. It must be easy to use and require as little interaction as is possible. Users shouldn't have a hard time interfacing with it, or have to push more buttons than either rosrun method. Lastly, this implementation must be future-proof *(is there a more formal word?)*. The duckietown software will soon have to migrate to a different version of ROS and Ubuntu. The utilities produced by this project must be usable even as ROS and Ubuntu change.

3 Existing technologies used here

3.0.1 apt-get isntall joystick;

3.0.2 jstest;

3.0.3 js_linux.py/python code from github

3.0.4 init.d/linux

3.0.5 supervisord

4 Implementation

4.1 Joystick Daemon

4.1.1 joystick-daemon.py; startuptest

4.1.2 start-response.sh

4.2 Reliable startup script

4.2.1 (runlevels, if need more padding)

4.2.2 init.d

4.2.3 rc.local

4.2.4 supervisord

5 Conclusions and Future work

- results - yo. it ran like 10 times in a row, successfully. It be good. - It shouldn't have taken this long. It's not worth figuring out why rc.local didn't work, though. Who knows why. - Make it prettier. - Make it more user friendly. - we use ROS, arguably the most popular robotics middleware around. That didn't matter much at all for this project. This entire system could be used for a system running on MOOS, used by LAMSS, or LCM, used by the Robot Locomotion group. So long as a joystick is being used as input, the only file that would change would be start-response.sh.

5.1 Acknowledgements

- John Leonard, CEO(???) of Duckietown Engineering Co., for generously advising this and many others of my works
- Liam Paull, COO of Duckietown Engineering Co., for advising me directly and patiently
- Alex Chernovsky, of SIPB, for recommending supervisord
- Anders, of pika, for giving me advice, though I forgot that advice
- that person from office of EECS undergrads, for helping me get an incomplete when jleonard was busy
- Kelly Shen, for planning advice, writing examples, and prayers.

5.2 References