Untitled

Jonathan Garcia-Mallen

??? August 2016

Contents

1 Background

Duckietown (2.166) is a graduate class on advanced autonomy taught at MIT. It was first taught Spring 2016. It is a hands-on, project-based 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 and with different backgrounds.

With this diversity in mind, the first two weeks were 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. 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 have no way of running a program on their duckiebots without using their laptops.

Picture this scenario. A grad student is testing a new autonomy on his laptop. His research advisor happens to walk by on her way to a meeting and asks him "how's your duckiebot doing?" The student hurries excitedly to power on his robot taxi (named batmobile), waits for it to connect to the network, and rushes the following incantation into his laptop's terminal:

```
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 cepillo.local (10.42.0.62) 56(84) bytes of data.
64 bytes from 10.42.0.62: icmp_seq=1 ttl=64 time=1.02 ms
64 bytes from 10.42.0.62: icmp_seq=2 ttl=64 time=0.971 ms
^C
--- batmoble.local ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.971/1.000/1.029/0.029 ms
dat-grad-student@duckietop4:~$ ^C
dat-grad-student@duckietop4:~$ ssh batmobile
```

```
ubuntu@batmobile:~$ roslaunch duckietown dat-grad-students-demo.launch  veh:=batmobile
```

First, `ssh batmobile` is run to log into the duckiebot named batmobile. Failing that, the student `ping`s his robot to see if it is on the network. On the second ping attempt, batmobile has completed enough of its bootup sequence to respond to pings. The student `ssh`'s into batmobile and launches his demo. But by this time, his advisor has already walked on to her meeting.

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?" A laptop and the corresponding WiFi network necessary are other potential points of failure. Trying to demonstrate your robot in a lab with twenty other duckiebots clogging the same 2.4GHz channel can lead to frustrations best kept outside of the scope of 2.166. A laptop should not be necessary in order to begin an autonomous 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 a computer offboard the robot. Its Raspberry Pi has two inputs that may be considered: a Raspberry Pi Cam 2, and a Logitech Gamepad F710 joystick controller. The computer itself runs Ubuntu 14.04.

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 when they type in roslaunch. Lastly, this implementation must be future-proof. The duckietown software will migrate to different versions of ROS and Ubuntu. The utilities produced by this project must be usable even as ROS and Ubuntu change.

3 Existing technologies used here

We must start a program at an unexpected time. Using input directly to the Raspberry Pi. We use its joystick. Input from the joystick must always be monitored. This monitor, this daemon, must start up on the duckiebot by itself.

To interface with the joystick, we included this python module from github. The initialization system initializes every process that runs on Ubuntu, directly or indirectly. But both used by Ubuntu 14.04 have been marked for death. Supervisord is a python package that can also initialize and manage processes. It is actively developed by ***x qntty of devs on gith*** We explain these further now.

3.1 js_linux.py

Nearly everything is a file in a UNIX operating system such as Ubuntu. This includes inputs from the joystick. When a button is pressed or a joystick is tilted, the events are written to a file `/dev/input/js0`. We use python to directly open and read this file:

```
# Open the joystick device.
fn = '/dev/input/js0'
print('Opening %s...' % fn)
jsdev = open(fn, 'rb')
```

This snippet is from js_linux.py [1]. This script reads and also parses this. When run, it prints to the terminal what buttons are being pressed and how much a joystick's axis is being tilted. We modify js_linux.py in order to import it as a module.

This script is based on the more standard C api for reading events written by the joystick device. The python script was chosen over the C api so that more people would be able to understand how these modules work, and modify them if they see it necessary. More MIT students are familiar with python than with C, especially when one considers undergraduate students.

3.2 System V

Linux initializes. We use SystemV. SystemV (and upstart) will soon die, so we use supervisord instead.

When the duckiebot is powered on, it must begin listening for joystick commands. This background listening program, this daemon, must be launched on system startup. Ubuntu 14. 04 supports two initialization (init) systems, called by the Linux kernel to start every other process necessary to run the operating system. They are Upstart *textbfcite something* and System V (sysv). The directories /etc/init and /etc/init.d correspond to either, respectively. Both init systems have been marked for death. Their replacement, systemd, is not readily available for Ubuntu 14.04. This poses a problem for any future-proof implementation. but they could not be avoided. We proceeded with sysv, as it has much greater support than upstart.

This init script does not start the listening program. It is used to start supervisord.

3.3 supervisord

We use supervisord version 3 as it allows us to run processes as a specific user. It is unclear how this would be done in init scripts. We installed it via pip **textitare you sure about that?**because the Ubuntu 14.04 package repositories only contains older versions. This feature is crucial, and is not available in older versions.

Installing via pip rather than apt-get required us to make an init script for it. This is not required in newer versions of Ubuntu. I consider it "future-proof."

Supervisord is configured in the `/etc/supervisor/supervisor.conf` file.

4 Implementation

4.1 Joystick Daemon

The joystick daemon consists of three parts: js_linux.py, joystick-daemon.py, and bash scripts corresponding to the buttons on the joystick.

4.1.1 joystick-daemon.py

We have modified js_linux.py for joystick-daemon.py to import it as a module. joystick-daemon.py runs an infinite loop based off of the example in js_linux.py. Each iteration, it records a translated portion of the joystick input written to /dev/input/js0. This portion is one event from the joystick. If the event is a button press, it checks which button was pressed and calls the shell script corre-

sponding to this buffer. For example, if the start button is pressed, joystick-daemon.py will call start-response.sh.

4.1.2 start-response.sh

This shell script is called by joystick-daemon.py whenever the start button is pressed. In order to run a ROS program, one usually performs the following invocation while logged into the duckiebot:

```
1  source /home/ubuntu/duckietown/environment.sh;
   source /home/ubuntu/duckietown/set_ros_master.sh;
3  roslaunch duckietown joystick.launch veh:=cepillo;
```

The first two lines set up ROS to be used, and the third finally launches the desired program. In this case, the program launched is a joystick demo for driving the duckiebot around with a joystick. On all duckiebots the entire duckietown software stack is located at /home/ubuntu/duckietown, where 'ubuntu' is the username for all duckiebots.

Since the software is located and set up for this user, it must be run as this user. Numerous attempts were made to run joystick.launch as the root user, to no avail.

4.2 Reliable startup script

The startup script has two requirements: launch joystick-daemon.py when the duckiebot powered on, and launch joystick-daemon.py as the user 'ubuntu.'

4.2.1 rc.local

The go-to method of running any startup script in ubuntu is by calling a script in rc.local ***cite stackoverflow, askraspi to show that this is the go-to method of running a startup script***. This requires only one line of bash. This was the initial method of implementing the startup script.

It failed. rc.local runs the programs it calls as the root user. We circumvented it by doing

```
1  su ubuntu -c "python /home/ubuntu/.duckietown/joystick-daemon/joystick-daemon.py >> /l
```

The `su` command lets run a command as another user. In this case, su is called by root to run python as ubuntu. This successfully called the startup script at about 50% of all boot sequences. We were not able to find out why it failed half the time, so we investigated other methods.

4.2.2 init.d

Creating a sysv init script would be the easiest and cleanest means to startup the joystick daemon.CONCLUSIONS We attempted to create a sysv init script /etc/init.d/duckietown_joystickd. It did not function. The details surrounding its failure are well lost. So we went to supervisord.

Ideally, a sysv init script would be made that calls joystick-daemon.py directly, rather than calling supervisord as we do now.

I pulled an example sysv init script for supervisord from somewhere on the internet, tweaked, it, and it worked /// We briefly explain sysv's interface. (/etc/init.d/README) (/etc/init.d/skeleton). skeleton is 160 lines. This is pretty much all we use:

```
# Short-Description:  Start/stop supervisor
# Description:        Start/stop supervisor daemon and its configured
#                     subprocesses.
### END INIT INFO

. /lib/lsb/init-functions

PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
DAEMON=/usr/local/bin/supervisord
NAME=supervisord
DESC=supervisor
```

4.2.3 supervisord

Supervisord was an unfortunate addition to the system, but it works smoothly. It started when its init script is run by the init system. Supervisord in turn calls joystick-daemon.py in a program block within the configuration file /etc/supervisor/supervisor.conf:

```
[program:joystickd]
command=/usr/bin/python /home/ubuntu/.duckietown/joystick-daemon/joystick-daemon.py >>
umask=022
user=ubuntu
```

Thus, it succeeds in initializing the python listening script.

5 Conclusions and Future work

The base goal of a reliable, future-proof interface was tested by cutting and reestablishing power to the duckiebot ten times and pressing the start button on the joystick to launch the joystick demo. Running the joystick demo lets the user drive the duckiebot using the joystick. Each time, pressing the start button and waiting about fifteen seconds let the user drive the duckiebot with the joystick. The program started 10/10 times.

It should not have taken this long to do this project. Had rc.local been reliable, this likely would have been finished in may. Unfortunately, no.

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.

Future Work:

consider using sysv init. I did not consider it at all. However, supervisord is a very large dependency; an oversized cludgeon for the little nail of a problem we have. Considerations: What are the differences between 14.04 init and 16.04 init?

The dependency of supervisord must be removed. The latest this should happen is when Duckietown switches to Ubuntu 16.04.

It should be more user friendly.

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

(1) rdb had a nice gist on github.com https://gist.github.com/rdb/8864666

(2) original C api for the joystick https://www.kernel.org/doc/Documentation/input/joystick-api.txt