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'?)*

```
1    dat−grad−student@duckietop4:~$ ssh batmobile
     ssh: Could not resolve hostname batmobile.local: Name or service not known
3    dat−grad−student@duckietop4:~$ ping batmobile.local
     ping: unknown host batmobile.local
     dat−grad−student@duckietop4:~$ ping batmobile.local
6    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
9    ^C
     −−− localhost ping statistics −−−
     4 packets transmitted, 4 received, 0% packet loss, time 2999ms
12   rtt min/avg/max/mdev = 0.022/0.029/0.039/0.008 ms
```

```
    dat−grad−student@duckietop4:~$ ssh batmobile

    .
15  .

    .

    #F I L L    T H E    R E S T    O F    T H I S
18  ubuntu@batmobile:~$ roslaunch duckietown dat−grad−students−demo.launch
    veh:=batmobile

1   or  he  does  the  other  thing  where  he  still  has  to  ssh  in  but  still  sees  all
    the  above  errors  that  come  up  whenever  you've  not  thinking  straight,   running  commands  y

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

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

code/shell should definitely be moved from this section to the next

We *¡–pl person?* 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. init.d initializes EVERYTHING on Ubuntu 14.04. But it's been marked for death. Supervisord is a python package that can also initialize thingies, and is actively developed by **x qntty of devs on gith** We explain these further now.

3.0.1 js_linux.py

Nearly everything is a file in UNIX. This includes inputs from the joystick. They are written to a file `/dev/input/js0`, which is opened and read as such in python:

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

This snippet is from js_linux.py. This one-file script interacts is based off of the standard linux c interface. Using C would have narrowed greatly how many people can improve this code, so python here we go.

Plus, we got this code from github. For free!

3.0.2 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.

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:

```
1    # PATH should only include /usr/* if it runs after the mountnfs.sh script
     PATH=/sbin:/usr/sbin:/bin:/usr/bin
3    DESC="Description of the service"
     NAME=daemonexecutablename
     DAEMON=/usr/sbin/$NAME
6    DAEMON_ARGS="--options args"
     PIDFILE=/var/run/$NAME.pid
     SCRIPTNAME=/etc/init.d/$NAME
9
     # Exit if the package is not installed
     [ -x "$DAEMON" ] || exit 0
```

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

3.0.3 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:

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

Thus, it succeeds in initializing the python listening script.

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   init.d

4.2.2   rc.local

4.2.3   supervisord

5   Conclusions and Future work

CONCLUSIONS - 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. FUTURE WORK. - Make it prettier. - once we switch to ubuntu 16.04, we should try to remove the dependency of supervisord. It's just not a good idea to depend on another thing. For now, it is a bridge to smooth such a transition. - 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. - 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?

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) https://www.kernel.org/doc/Documentation/input/joystick-api.txt

(2) rdb had a nice gist on github.com