kr_docs Documentation

Release 0

KumarRobotics

Oct 02, 2019

Contents

This is the index for the KR github docs. This documentation will cover the following topics:

- Guidelines when using the organization repository.
- Recommended practices when developing in ROS.
- Useful tips and tricks for streamlining your development process.

The documentation is very much a work in progress and will continue to be updated as time goes by. Report inaccuracies to the maintainer [Chao].

Contents:

CHAPTER 1

Suggested Readings

For running experiments on real robotics platforms a lot of tools can come in handy. Following are some of the software/tools to get familarized with before diving into running robots in the lab.

Please follow it thoroughly and enjoy the learning!

1.1 Ubuntu

Most software for UAVs and robots in lab runs on Ubuntu 18.04 LTS. Please install this particular version, a lot of other software specifically depends on it. We strongly suggest to install a native system, don't virtualize. Do not use Virtual Box, although it works fine, the drop in performance is really noticeable, especially when simulating a real-time flying drone.

1.2 Bash

Bash is the standard/default shell in Ubuntu. Most of the time, when you work in the terminal, you use bash to run programs, for scripting and to manage your file system.

Skills needed:

- moving through a file system
- writing and running simple shell scripts
- mounting devices (`fdisk`, `mount`, `sync`, `umount`)
- **common commands:**
  - `grep`, `|`, `echo`, `ssh`, `scp`, `rsync`, `cat`

1.3 GIT - code versioning system

GIT is a powerful versioning system. All our code is stored and versioned using GIT. It allows collaborative work between many people and can be managed completely from the terminal. Our repositories are hosted at

Tutorials:

-
-

Skills needed:

- cloning repositories
- committing changes
- pushing and pulling changes to a server
- branching
- merging
- fixing conflicts (will come as they appear :-))

1.4 TMUX - terminal multiplexer

Tmux is a command line utility that allows splitting a terminal to multiple panels and creating windows (tabs). It is similar to e.g. Terminator, but runs completely in the command line. Thus it can be used over ssh.

It is scriptable, which makes it ideal for automating simulations, where multiple programs are running in parallel.

-
-

1.5 Tmuxinator - automating tmux

Tmux itself is very powerful, tmuxinator is just adding some cream on it. Tmuxinator uses .xml files containing the description of a tmux session. It allows to define and automate complex multi-terminal setups for e.g. development (one session per program) and simulations.

-

1.6 Vim

Everyone should use a tool that is right for the job. Well, for our purposes (C++, ROS, python, bash), Vim is very well suited. A lot of the time, you will find yourself in need of editing a code remotely (over ssh), and Vim can provide IDE-like features even in that situation.

Learning Vim is about changing the paradigm of programming - it's more about controlling a machine (synthesizer) that edits a text, rather than moving a cursor with a mouse and then typing the text.

Working in the terminal, using e.g. tmux and vim can also help you put away your mouse. Yes, a mouse is not an ideal tool for programming, though it has its use in gaming, 3D modeling, video editing and so on. Without a mouse, you will become much more productive, and your carpal tunnels will thank you.

Give it a chance, have a look at following videos:

-
-

1.7 ROS - Robot Operating System

ROS is a middleware between Ubuntu and C++. Thanks to it, our programs can talk to each other asynchronously, even though they don't have to "know" each other necessarily. It also allows simple control of your software from the terminal. A lot of robotic stuff has been already programmed in ROS, including sensor drivers, visualization, planning, etc., thus we don't need to reinvent the wheel. Getting into ROS is simple, just follow tutorials on their webpage and don't be afraid to experiment. You can't break the drone in simulation :-).

Tutorials:

Required skills: The more, the better, most of it will come as your start working on a project.

CHAPTER 2

Software

2.1 C++

2.1.1 Modern C++

Targeting c++11

c++11 is the most recent ISO standard for the c++ language. It adds numerous features which improve the language and the standard library. We recommend adopting c++11. Some packages in the KR github may depend on your compiler having support for the new specification.

Since version 4.8, GCC implements all of the c++11 features. GCC 4.8 is **pre-installed in Ubuntu 14.04** (Trusty) and works with ROS Indigo out of the box.

Setting up GCC 4.8 on Ubuntu 12.04

The Ubuntu 12.04 (Precise) repositories do not contain a package for GCC 4.8. PPAs for toolchain test builds are available. In order to install `gcc` and `g++` version 4.8, execute the following commands:

```
sudo add-apt-repository ppa:ubuntu-toolchain-r/test
sudo apt-get update
sudo apt-get install gcc-4.8 g++-4.8
```

You should then update the links in `/usr/bin` so that they point to the newer version of gcc. This is achieved by executing:

```
sudo update-alternatives --remove-all gcc
sudo update-alternatives --remove-all g++
sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.8 20
sudo update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-4.8 20
sudo update-alternatives --config gcc
sudo update-alternatives --config g++
```

You will then need to update boost to 1.54 in order to use the new compiler.

Source: ubuntuhandbook.org

Enabling c++11 in CMake

c++11 features are enabled with the compiler switch: `-std=c++11`. Adding this option to your CMakeLists.txt file is very simple:

```
add_definitions("-std=c++11")
```

2.1.2 C++ IDE

2.1.3 C++ Style Guide

Google C++ Style Guide ROS C++ Style Guide

2.2 Git Workflow

2.2.1 Git Branching Model

For simplicity, we use GitHub Flow, which will be summarized here.

GitHub Flow

- Anything in the `master` branch is deployable
- To work on something new, create a descriptively named branch off the master (ie: `feature/handle-frame-drops`)
- Commit to that branch locally and regularly push your work to the same named branch on the server
- When you need feedback or help, or you think the branch is ready for mergin, open a pull request
- After someone else has reviewed and signed off on the feature, you can merge it into master
- Once it is merged and pushed to *master*, you can and should deploy immediately

`Master` branch

The `origin/master` is the main branch where the code of `HEAD` always reflects a demo-ready state. We treat the `master` branch as *read-only*, such that no code should be directly committed to it. Instead, the `master` branch moves forward only by merging from topic branches.

Topic branches

The topic branches always have a limited life time, since they will be removed eventually. Try to keep the commits in a single branch tightly related, and give it a descriptive name.

We recommend using the following prefixes in branch names:

- `feature/` - New features.

- `fix/` - Bug fixes.
- `refactor/` - Code movement or restructuring.

Pull requests and code review

We follow the shared repository model. You do not need to fork the repository. Simply use topic branches to sandbox new work. Use the following workflow for development:

1. Create a topic branch off of `master` for new work:

```
$ git checkout -b fix/handle-frame-drops
```

2. Develop your code inside the branch. Commit liberally to track your changes.
3. Regularly push your branch to the server (`origin`):

```
$ git push origin fix/handle-frame-drops
```

4. When you are ready for a code review, or simply want some feedback, initiate a pull request. Go to the repository that you are working on on Github, switch to your branch, and click the Pull Request button.
5. Github defaults to merging into `master`. You can click the Edit button and change the merge target to `some-other-branch`.
6. Get someone else to review your changes. Revise & iterate until convergence.
7. Merge your branch. You can merge directly through Github by clicking the `Merge pull request` button.

   To merge locally:

```
# Checkout the changes if you don't already have the branch locally
git fetch origin
git checkout fix/handle-frame-drops

# Make sure the branch is up-to-date and resolve any conflicts
git merge master

# Fast-forward master, and update the server
git checkout master
git merge fix/handle-frame-drops
git push origin master
```

8. Optionally, tidy up defunct branches you do not intend to develop anymore.

   Delete local topic branch:

```
git branch -d fix/handle-frame-drops
```

   If you are dealing with pull request on Github, it will prompt you whether to delete the branch or not after merging. Otherwise, you can do this to clean up the remote topic branch:

```
# assume remote branch not deleted by Github
git push origin --delete fix/handle-frame-drops
git remote prune origin
```

2.3 Test

Google Test

2.4 LaTeX

2.4.1 Tikz and PGFPlots

Subsample plots

If you have too much data to plot, your document may take quite a while to compile. To speed things up, you can subsample the data. To plot only a subset, you can follow the discussion here and modify the `addplot` command in your `tikz` file.

```
\addplot[<your options here>,each nth point={100}] ...
```

CHAPTER 3

Hardware

There are 4 sections under hardware

3.1 Motion Capture

3.1.1 Indoor Motion Capture

The instruction for using indoor Vicon system can be found .

3.1.2 Outdoor Motion Capture

The instruction for using indoor Qualisys system can be found .

3.2 Network

3.2.1 Network Setup

Indoor Network Setup

To ensure your robot has continuous and reliable network connection, you should use `wpa_supplicant` if you don't have a display for the robot.

Here are the steps for setting up `wpa_supplicant`.

0. First of all, ask for a static ip for your robot in the mrsl network.
1. add `wpa_supplicant.conf` to `/etc/wpa_supplicant/`:

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
ap_scan=1
fast_reauth=1

network={
    ssid="mrsl_airrouterhp"
    scan_ssid=1
    key_mgmt=NONE
}
```

2. modify `/etc/network/interfaces` so that it looks like:

```
auto lo
iface lo inet loopback

# auto start wlan0 interface
auto wlan0
# use static ip for wlan0
iface wlan0 inet static
# your desired ip
address 192.168.129.xyz
gateway 192.168.129.1
netmask 255.255.255.0
wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf

auto eth0
iface eth0 inet static
address 192.168.1.abc
netmask 255.255.255.0
```

3. The next time your power on your robot, it will automatically connect to the router you specified in `wpa_supplicant.conf`. And if will reconnect even if you lose connection temporarily.
4. A sample `interfaces` file can be found here.

Outdoor Network Setup

For large-scale outdoor experiments, we need ad reliable long-range wifi connection between the robot and the ground station. To this end, we use Ubiquiti wireless products. Both *bullet M* and *PicoStation M* are good choices.

We recommend using *PicoStation* simply because it is easier to strip the case off.

We use Ubiquiti products mainly for its airMax technology, which is designed for outdoors wireless communication.

Below we describe in detail how to setup *airMax* connection between two *Ubiquiti* devices. We use a *Bullet M* in **Access Point** mode to serve as a ground station and a *PicoStation M* in **Station** mode, which can be seen as a client.

Access Point (Ground Station)

1. On your computer, add a new `Ethernet` connection from **Ubuntu Network Manager**.
2. In **IPv4 Settings** tab, change **Method** to `Manual`
3. Add an **Address**, here we use `192.168.1.200`, with **Netmask** `255.255.255.0`. You can leave the **Gateway** and **DNS servers** empty.
4. Power up the *Bullet M* and connect to your computer via Ehternet.

5. Make sure you are on the network configuration you just setup. Then open a browser window with the following **ip address** `192.168.1.20`, which is the default address for *Bullet M*.
6. The default **Username** and **Password** are both `ubnt`.
7. In the first tab left to **MAIN**, select `Enable` for **airMAX**, then click **Change** and **Apply**. Notice that all the changes you made will be cancelled if you switch tab without applying those changes.
8. In **WIRELESS** tab, make the following changes. The rest can be left as default.

Item	Value
Wireless Mode	Access Point
SSID	ubnt (anyname)
Channel Width	20 MHz (recommended)

9. In **NETWORK** tab, make the following changes.

Item	Value
Network Mode	Bridge
Management IP address	Static
IP Address	192.168.1.20
Netmask	255.255.255.0
Gateway IP	192.168.1.1

Station (Client)

1. On your robot, make sure your ethernet interface (eth0) is on the same subnet as the **Access Point**, which should be `192.168.1.abc`.
2. Power up the *PicoStation M* and connect to your robot via Ethernet.
3. Open a browser window with the folloing **IP address** `192.168.1.20` (default).
4. Again, he default **Username** and **Password** are both `ubnt`.
5. In **WIRELESS** tab, make the folloing changes. The rest can be left as default.

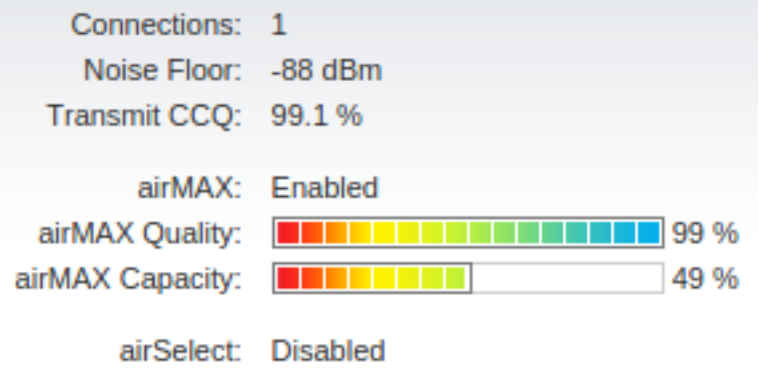
Item	Value
Wireless Mode	Station
SSID	ubnt (same as AP)
Channel Width	Auto 20/40 MHz

6. In **NETWORK** tab, make the following changes. Note that **IP Address** should be something other than `20` and `abc`.

Item	Value
Network Mode	Bridge
Management IP address	Static
IP Address	192.168.1.21
Netmask	255.255.255.0
Gateway IP	192.168.1.1

7. In **MAIN** tab, remember the **WLAN0 MAC** of this device. Then go back to the **Access Point** setup page on your computer. In **WIRELESS** tab, select `Enable` for **MAC ACL** and `Allow` for **Policy**. And add the above MAC to the list by clicking on **ACL...**.

Now you should have established **airMAX** connection between the *Bullet M* and the *PicoStation M*. In **MAIN** tab of the **Access Point**, you will see something like this.



Internet Sharing from Your Computer

Often, it is convenient to have internet access on your robot.

1. Add the following two lines to your ethernet interface in `/etc/network/interfaces`:

```
gateway 192.168.1.200 # IP address of your laptop
dns-nameservers 192.168.129.1 8.8.8.8  # for internet access
```

2. Put the following function in your `.bashrc`, then you can enable and disable sharing via `sharenet on/off`:

```
function sharenet()
{
    if [ $# -eq 0 ]; then
        echo "usage: sharenet <on/off>"
        return 0
    fi

    local if_from=wlan0
    local if_to=eth0
    # check command-line commands
    cmd=$1
    case $cmd in
        on )
            sudo su -c "echo 1 > /proc/sys/net/ipv4/ip_forward"
            echo "Enable sharing internet from $if_from to $if_to"
            sudo /sbin/iptables -A FORWARD -i $if_to -o $if_from -j ACCEPT
            sudo /sbin/iptables -A FORWARD -i $if_from -o $if_to -m state --state␣
↪RELATED,ESTABLISHED -j ACCEPT
            sudo /sbin/iptables -t nat -A POSTROUTING -o $if_from -j MASQUERADE
            ;;
        off )
            sudo su -c "echo 0 > /proc/sys/net/ipv4/ip_forward"
            echo "Disable sharing internet from $if_from to $if_to"
            ;;
        * )
```

(continues on next page)

(continued from previous page)

```
            echo "sharenet: $1: invalid command"
            echo "usage: sharenet <on/off>"
            ;;
    esac
}
```

3.2.2 Wifi Test

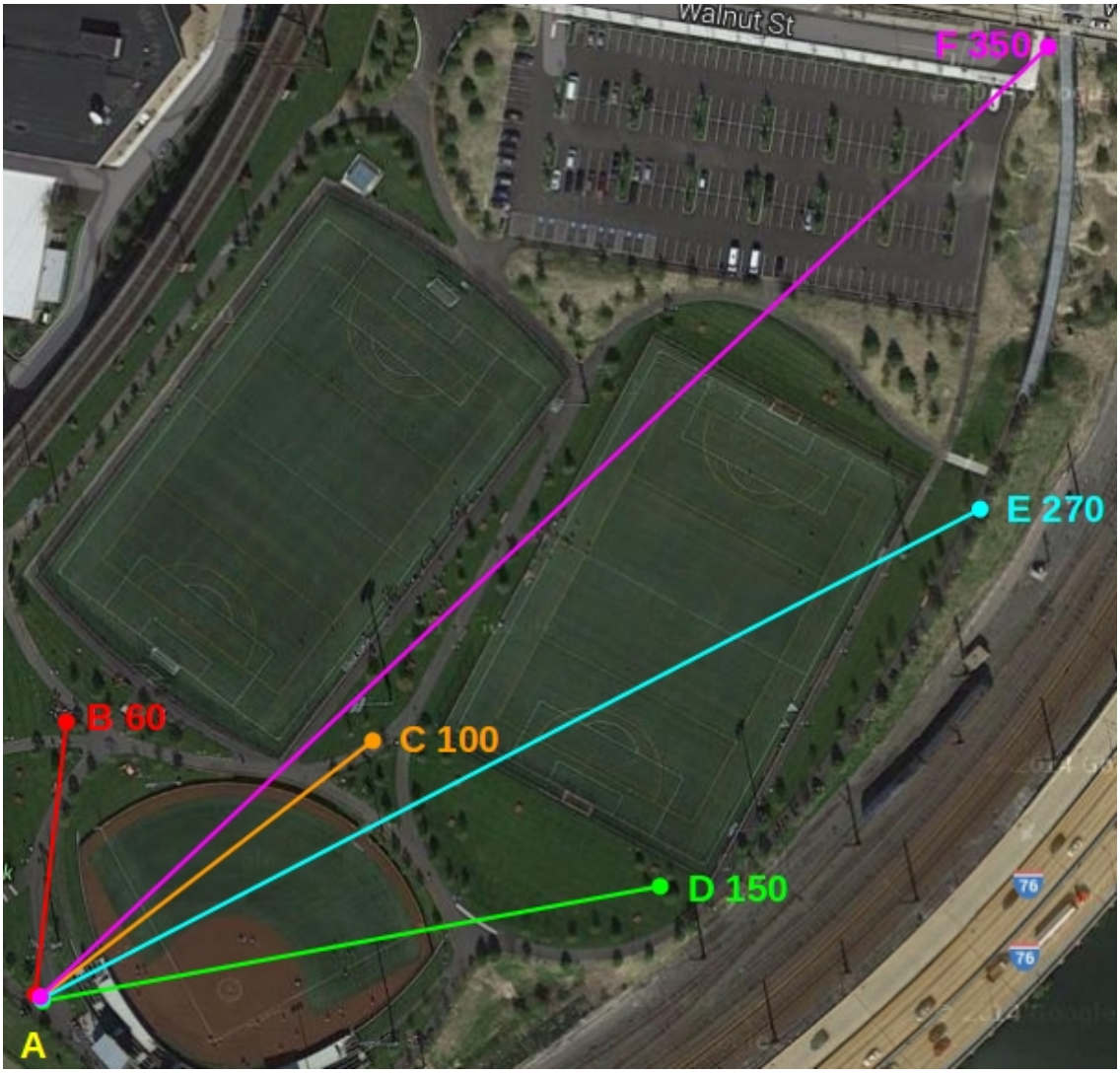
On 2014-12-17, we conducted a test comparing bullet airmax connection to normal wifi connection.

Test Setup

The test uses 1 base station, which is a bullet-M2HP with an EnGenius EAG-2408 antenna. There are three clients who connect to the station:

1. Bullet-M2HP with small antenna
2. Bullet-M2HP with medium antenna
3. Intel 7260HMW IEEE 802.11AC, dual-band, 2x2 Wi-Fi

The bullets are connected to the base station via Airmax, while the Intel card is connected to the base station via normal wireless connection. The *Channel Spectrum Width* is set to 40 MHz, which gives a max TX rate of 150 Mbps.

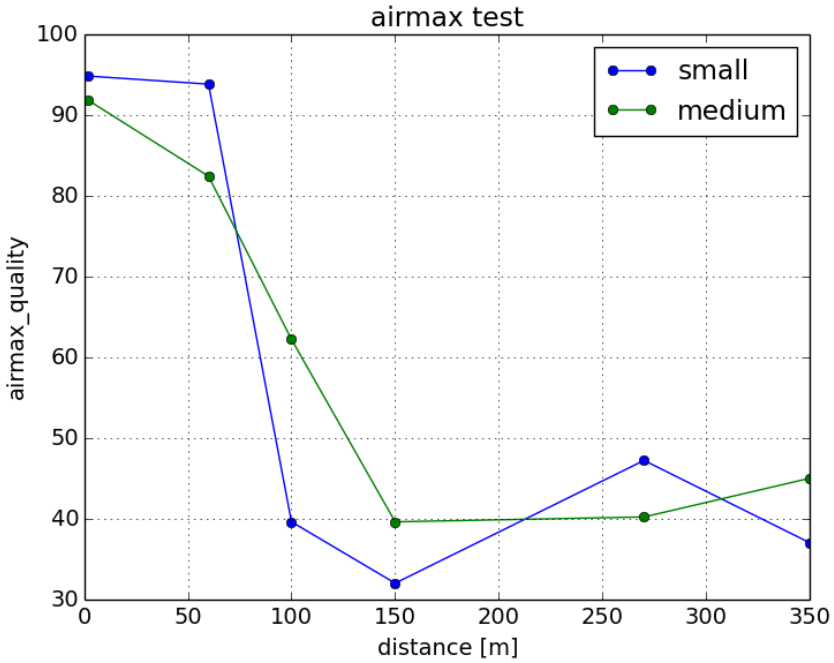


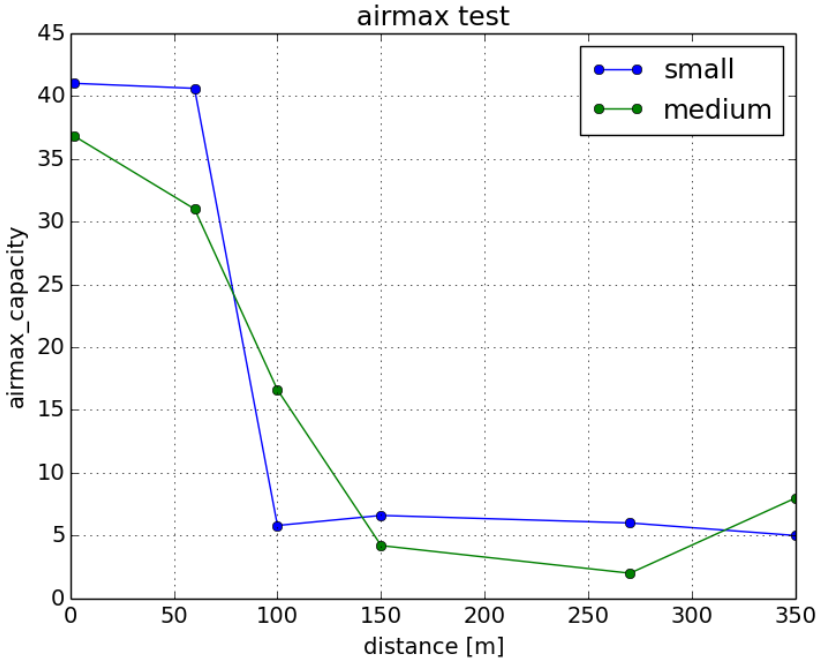
We picked 6 points at Penn Park with various distances to the base station (which is located at point A), and we test the connection at those points with the three configuration mentioned above.

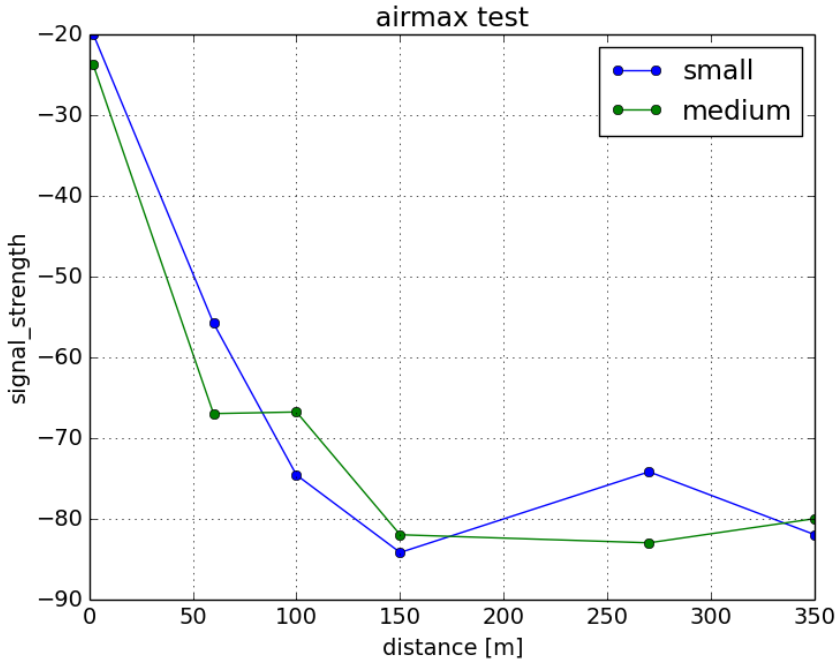
Test Results

airmax

The following images show comparision of airmax capacity, quality and signal strength at different distances using two different antenna with airmax enabled.


airmax test
small
medium
airmax_quality
distance [m]
100
90
80
70
60
50
40
30
0
50
100
150
200
250
300
350

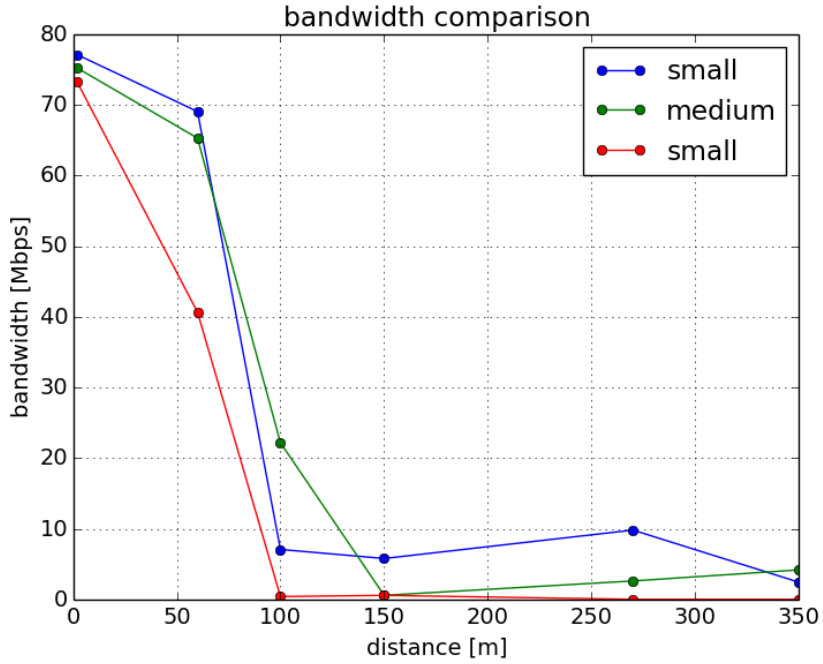

airmax test
small
medium
airmax_capacity
distance [m]
45
40
35
30
25
20
15
10
5
0
0
50
100
150
200
250
300
350



Note that even though Point D is significantly closer to base station than Point E, it experienced worse connection. That is because D is below A while E is on a bridge and above A.

bandwidth

The following image shows comparision of bandwidth at different distances using the above configuration and normal wifi.



Summary

It seems like the client's antenna size does not make a huge difference in coneection quality.

- If you are doing experiments in open space and within 100 meters, then you can probably get away with normal wireless connection.
- If your robot needs to go further away and you want to monitor its status, then you should switch to *Ubiquiti* **airMAX** connection.
- If you need good connection even at more than 400 meters away, then you should buy a better station/antenna.

We will do some more tests in 2015 in Biglervielle and California and will update this report later.

3.2.3 Chrony

Chrony lets you syncronize the clocks of multiple devices so you can have consistent timestamps.

To install chrony run `sudo apt install chrony`

If you don't know where chrony.conf is run `whereis chrony`

Setting up chronyd (server)

To allow devices with all ip adress in the network `192.168.1.`, add the following line to `chrony.conf`:

```
allow 192.168.1.1/24
```

Setting up chronyc (client)

Assuming master's ip `192.168.1.55` and client ip will be `192.168.1.115`

chrony.conf

```
server 192.168.1.55 iburst
makestep 0.0001 10
maxchange 1000000.000001 1 1
rtcsync
```

/etc/network/interfaces

```
source /etc/network/interfaces.d/*

auto lo
iface lo inet loopback

# auto eth0
iface eth0 inet dhcp
iface default inet dhcp
auto wlan0
#iface wlan0 inet dhcp
iface wlan0 inet static
```

(continues on next page)

(continued from previous page)

```
        address 192.168.1.115
        netmask 255.255.255.0
        gateway 192.168.1.1
        dns-nameservers 192.168.129.5
        wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
```

/etc/wpa_supplicant/wpa_supplicant.conf

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=ssh
ap_scan=1

network={
        ssid="windy"
        proto=RSN
        key_mgmt=NONE
        pairwise=CCMP TKIP
        group=CCMP TKIP
}
```

3.3 Computation

3.3.1 Intel NUC

Intel NUC is a common computation platform being used at KumarRobotics. They are usually mounted on bigger platform such as Pelican and DJI S900/1000.

This page is dedicated to record problems and solutions that we encountered while working with the NUC.

Ubuntu Installation

The NUC boots to the Ubuntu live USB and installing the OS to the internal mSATA SSD completes without an issue. The problem can be that the NUC wouldn't see the drive as a EFI boot target, and it would refuse to boot from the drive. The NUC has problems with custom EFI boot locations, the NUC expects there to be a file named bootx64.efi located in the /EFI/BOOT folder on the EFI partition, and if that file is named something else and/or located elsewhere, the computer can't recognize the drive as a boot target. The Debian distros will usually try to place a bootloader named grubx64.efi in a folder named /EFI/[distro name], so the NUC wouldn't try to boot from the drive.

From Ubuntu live USB drive, finish the operating system installation but don't reboot. Instead, open a terminal window and type the following.

```
sudo mount /dev/sda1 /mnt
sudo mkdir /mnt/EFI/BOOT
sudo cp /mnt/EFI/ubuntu/* /mnt/EFI/BOOT
sudo mv /mnt/EFI/BOOT/grubx64.efi /mnt/EFI/BOOT/bootx64.efi
```

This assumes `sda1` is your EFI boot drive partition. It might be different if you have multiple hard drives/usb connected. Hopefully this issue would be fixed with an updated BIOS release.

Improper Shutdown

If your NUC shutdowns due to low battery power, then it may not boot up properly the next time. This is described in detail here.

The reason why this is happening is because in `/boot/grub/grub.cfg`, the following logic sets the `timeout` to `-1` after a power failure.

```
if [ "${recordfail}" = 1 ] ; then
    set timeout=-1
else
    ...
```

But we don't want to modify this file directly because it will be overwritten after every kernel update. The workaround to fix this problem is by setting a config variable `GRUB_RECORDFAIL_TIMEOUT` which will be read by `/etc/grub.d/00_header`.

Open `/etc/default/grub`, add the following line in the file.

```
GRUB_RECORDFAIL_TIMEOUT=5
```

Here `5` is just a random number I picked. After this modification run

```
sudo update-grub
```

Then reboot and verify that in `boot/grub/grub.cfg` the `set timeout=-1` has been successfully changed to `set timeout=5`.

Waiting for Network Configuration

Sometimes when you boot up Ubuntu, it will stuck at the booting screen for a long time saying it is waiting for network configuration. This can be annoying. Here's how to disable it.

1. Open `/etc/init/failsafe.conf`
2. Change the first 3 `sleep` command to `sleep x` where `x` is some small value, like `5`.
3. Or if you don't want to wait at all, simply comment out the following lines.

```
$PLYMOUTH message --text="Waiting for network configuration..." || :
sleep 40

$PLYMOUTH message --text="Waiting up to 60 more seconds for network configuration.
↪.." || :
sleep 59
```

3.3.2 Pixhawk

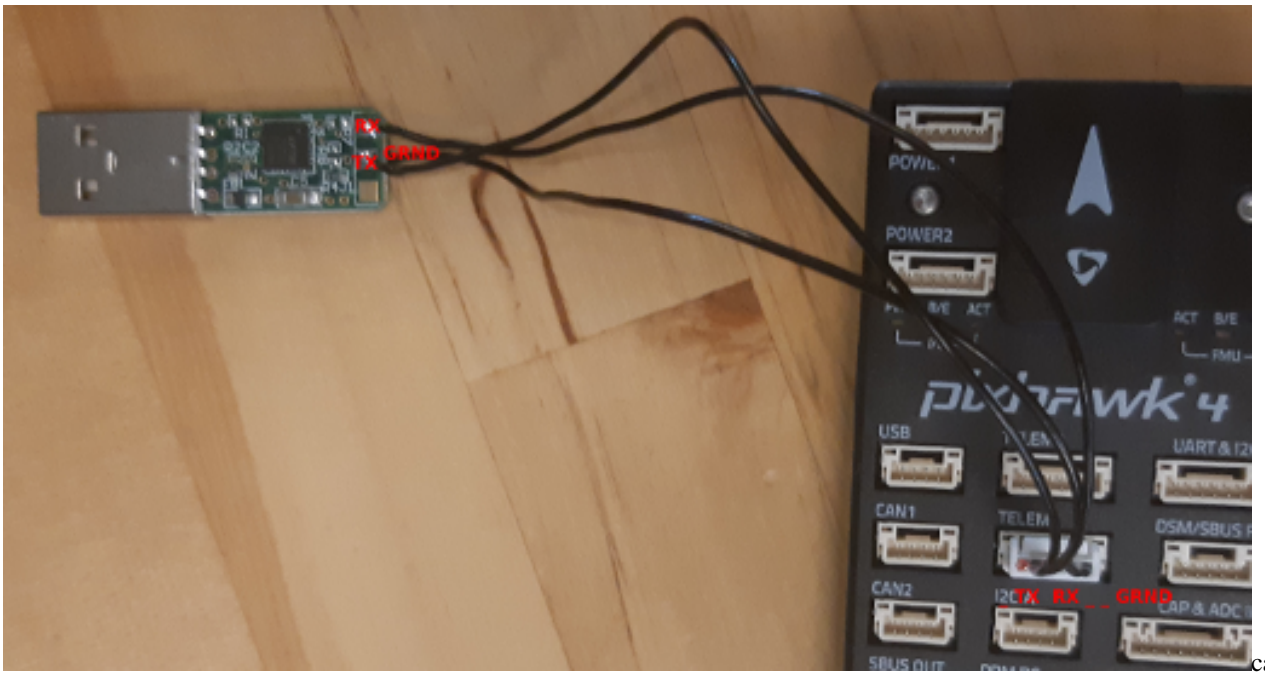
Setting up a new Pixhawk for autonomous flight with MAVROS

Cable (UART to USB)

Tx - Rx

Rx - Tx

Grnd - Grnd



cables

Linux

Create a new udev rule

```
cd /etc/udev/rules.d/
sudo vim 99-pixhawk.rules
```

Add the following line:

```
SUBSYSTEM=="tty", ATTRS{idVendor}=="xxxx", ATTRS{idProduct}=="yyyy",
ATTRS{serial}=="zzzz", SYMLINK+="ttyPixhawk", RUN+="/bin/stty -F /dev/
ttyPixhawk 921600 raw -echo"
```

You can find your idVendor and idProduct running `lsusb`, identify which one is your device. The device will be either the pixhawk itself or the UART adapter. **Do this using the interface you will use for autonomy later**. If you're using more than one device with same ids add serial as well.

`Bus 002 Device 007:  ID 26ac:0032` In this case `idVendor = 26ac` and `idProduct = 0032`.

```
udevadm info -a -n /dev/ttyUSB1 | grep -E '{serial}|{idVendor}|{idProduct}' |
head -n 3
```

Reboot you computer: `sudo reboot now`

Pixhawk (QGC)

Connect your Pixhawk to a computer using the micro-usb port and open QGroundControl.

Go To *vehicle setup* -> *params* and set:

- **MAV_1_CONFIG** = TELEM 2 (reboot pixhawk after changing this)
- **MAV_1_MODE** = Onboard

- **SER_TEL2_BAUD** = 921600

Test

```
<launch>
        <!-- vim: set ft=xml noet : -->
        <!-- example launch script for PX4 based FCU's -->

        <arg name="fcu_url" default="/dev/ttyPixhawk:921600"/>
        <arg name="gcs_url" default="" />
        <arg name="tgt_system" default="1" />
        <arg name="tgt_component" default="1" />
        <arg name="log_output" default="screen" />
        <arg name="fcu_protocol" default="v2.0" />
        <arg name="respawn_mavros" default="false" />

        <include file="$(find mavros)/launch/node.launch">
                <arg name="pluginlists_yaml" value="$(find mavros)/launch/autonomy/
↪px4_pluginlists.yaml" />
                <arg name="config_yaml" value="$(find mavros)/launch/autonomy/px4_
↪config.yaml" />

                <arg name="fcu_url" value="$(arg fcu_url)" />
                <arg name="gcs_url" value="$(arg gcs_url)" />
                <arg name="tgt_system" value="$(arg tgt_system)" />
                <arg name="tgt_component" value="$(arg tgt_component)" />
                <arg name="log_output" value="$(arg log_output)" />
                <arg name="fcu_protocol" value="$(arg fcu_protocol)" />
                <arg name="respawn_mavros" default="$(arg respawn_mavros)" />
        </include>
</launch>
```

Useful links

http://wiki.ros.org/mavros

Pixhawk docs for setting up companion computer

3.3.3 Odroid

Some introductions to odroid goes here.

Update Kernel

Updating the kernel: Follow the instructions from here or run the following

```
sudo -s
wget -O /usr/local/bin/odroid-utility.sh https://raw.githubusercontent.com/mdrjr/
↪odroid-utility/master/odroid-utility.sh
chmod +x /usr/local/bin/odroid-utility.sh
odroid-utility.sh
```

3.4 Robot

3.4.1 Scarab

The instruction for using scarabs can be found .

3.4.2 Snapdragon Flight

The instructions for setting up Snapdragon Flight boards can be found .

3.4.3 Clearpath Jackal

3.4.4 Fetch

3.5 Sensor

3.5.1 Serial Devices Permission and Ownership

Some ROS driver which access resources in `/dev` will fail to launch, stating that they cannot open the device or access the resource.

Perform `ls /dev/tty*` to see if there is a serial device connected. The default could be `/dev/ttyACM0` or `/dev/ttyUSB0`. If the device is connected, then you probably need to fix a permission issue.

By default, TTY devices in `/dev` are owned by the root user. You can do the following to grant yourself access.

Simple fix: giving temporary access to `<username>`:

```
sudo chown <username> /dev/ttyACM0
```

Permanent fix: adding `<username>` to dialout group:

```
usermod -a -G dialout <username>
```

3.6 Thrust Stand Testing

There comes a time in every engineer's life, when they must measure thrust coefficients and efficiency from propeller-motor combinations. This is that.

3.6.1 RC Benchmark Series 1580 Thrust Stand and Dynamometer

Necessary hardware:

- RC Benchmark 1580 Board
- RC Benchmark structure (including two 2kg load cells and one 5kg load cell)
- Calibration hardware (200g mass, beam, and screws)
- Mini USB Type-B Cable

- Computer with RC Benchmark software installed
- Motor and Propeller
- ESC (appropriately spec'd for motor/prop combo)
- Power Supply (with appropriate voltage/current/power ratings)
- Power Cables (stripped on one end)
- Non-reflective tape
- Reflective tape
- Optical Sensor and cable
- 4x motor mounting screws
- Corresponding screw driver
- 2x C-clamps

Performing a Thrust Test:

1. Assuming the thrust stand has already been assembled, clamp down the bottom of the structure to a flat surface.
2. Download the RC Benchmark Software here: https://rcbenchmark.gitlab.io/docs/en/dynamometer/software/dynamometer-software-download.html
3. Connect the RC Benchmark board to your computer via the Mini-USB Type B cable and press the "Connect" button on the left side of the RC Benchmark software.
4. Under the "Utilities" tab, perform a thrust calibration and, if necessary, a torque calibration. When you are done, reclamp the thrust stand to your work surface.
5. Connect the power cables from your ESC to the respective positive and negative screw terminals labeled, "ESC", on top of the PCB.
6. Connect the signal cables from your ESC to the header pins labeled, "ESC". (Note: Colors may vary, but Black will "always" be ground/negative.)
7. Connect the power cables from your power supply to the respective positive and negative screw terminals labeled, "Power Input" and "Ground", respectively.
8. Tape the motor's side with non-reflective tape. Usually, the motor's metal housing will already be reflective, so you may or may not need to tape an additional small strip of reflective tape on the motor.
9. Mount the propeller/motor combination to the thrust stand. (Note: for small motors, you may need to create an mounting adapter.)
10. Mount the optical RPM sensor relatively close the the motor. Connect the sensor cables to one of the headers labeled, "S1", "S2", or "S3". (Note: The board does not by default provide power to sensor. You will need to provide an external 5V source from your ESC, power supply, or with a wire bridge from one of the board's 5V sources.)
11. Optional: To use the electrical RPM sensor, connect a jumper cable from the pin labeled, "RPM probe", to one of the ESC leads. Then, under "Utilities", adjust the value for the "Number of Poles" setting to match your motor.
12. Connect the motor leads to the ESC.
13. Turn on your power supply and adjust the voltage and current limit accordingly.
14. Under the "Setup" tab, set working units to the system you prefer, select a folder to be your working directory, and flash the latest firmware if it is not up-to-date.
15. Under the "Safety Cutoffs" tab, adjust the cutoffs to values appropriate to your prop/motor combination.

16. Under the “Manual Control” tab, navigate to the “ESC” scrollbar, where you can adjust the duty cycle to check the direction the motor is spinning. If the motor is spinning the wrong way, switch the order of two of the three motor cables.
17. Under the “Automatic Control” tab, click the “Script” drop box and select “Sweep - continuous”. Here, you can adjust the minimum and maximum PWM values, and total time, among other things. When you are ready, zero the load cells by pressing “Tare Load Cells”, then press the “Run” button followed by the “Start” button to begin the test.

Important: Make sure that the propeller is either pointed toward or away from you in case the propeller breaks. For larger propellers, you will want to use the caged enclosure located in the motion capture space at PERCH. Lastly, always wear safety glasses when doing thrust tests.

Tips & Troubleshooting:

- Make sure to fasten your ESC and power cabling securely with cable ties. Dangling power cables can affect the readings.
- For testing multiple motors, it may be a good idea to solder screw terminals to your ESC to avoid repeatedly soldering connectors to the motor leads.
- It is a good idea to periodically perform a calibration on the ESC. First, remove the propeller from the motor and, under “Manual Control”, uncheck the box under the ESC scrollbar. Set the scrollbar to its maximum and check the box. The ESC should audibly indicate via the motor that it is in calibration mode. Uncheck and recheck the box, then set the scrollbar to its minimum position. The ESC will cause the motor to emit a tone if the calibration was successfull. You can check this by adjusting the scroll bar.
- To obtain a thrust/rpm curve, the voltage supplied to the ESC is not extremely important, and you can just use the nominal battery voltage of your system. If you would like map the duty cycle(therefore voltage) to thrust more accurately, you can run tests at your maximum and minimum voltages and interpolate.
- For further details beyond this quick-start guide, RCBenchmark’s online documentation can be found here: https://rcbenchmark.gitlab.io/docs/en/series-1580.html

CHAPTER 4

Setting up your build environment

4.1 catkin

Catkin is the recommended dependency manager for use with ROS. Most packages added to the KR github will be intended for use with catkin. Instructions for setting up a catkin workspace are available on the ROS wiki. Adding a new package to your catkin workspace is straightforward.

4.1.1 Linking packages into your catkin workspace

When managing a large project with multiple packages, it is usually convenient to keep your workspace separate from the git folder. This is achieved by *symbolically linking* your packages into the catkin `src` folder.

Example

For the following example we suppose the following directory structure exists:

```
~/Documents/catkin  <-- Root of the catkin workspace
~/Documents/code    <-- Folder where you store your git repositories
```

If you wanted to clone the bluefox2 package and add it to your catkin workspace, you would do the following:

```
cd ~/Documents/code
git clone https://github.com/KumarRobotics/bluefox2
ln -s bluefox2 ~/Documents/catkin/src/bluefox2
```

Of course, you do not *have* to link to the root of a git repository. You can link to any folder which is itself a catkin package, or which contains catkin packages.

4.2 Qt Creator

Qt Creator is a fast, powerful and, cross-platform c++ development IDE. It is actively maintained by the Digia corporation, and is freely available under the LGPL licensing scheme. Qt integrates easily with cmake projects, and offers

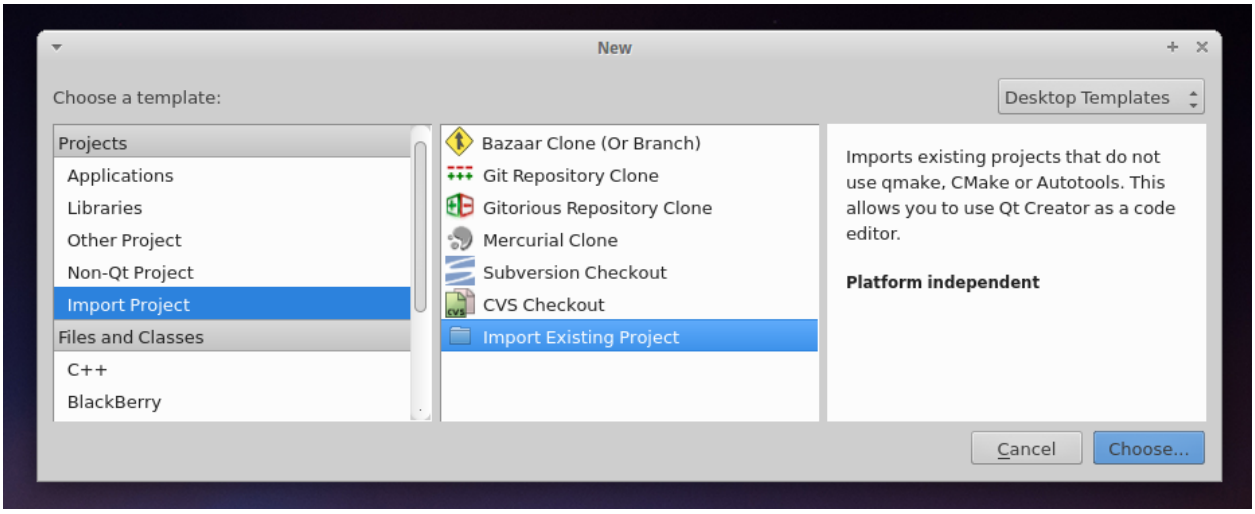
an elegant GUI development toolkit for Qt - the recommended platform for building graphical ROS applications.

4.2.1 Launching Qt Creator

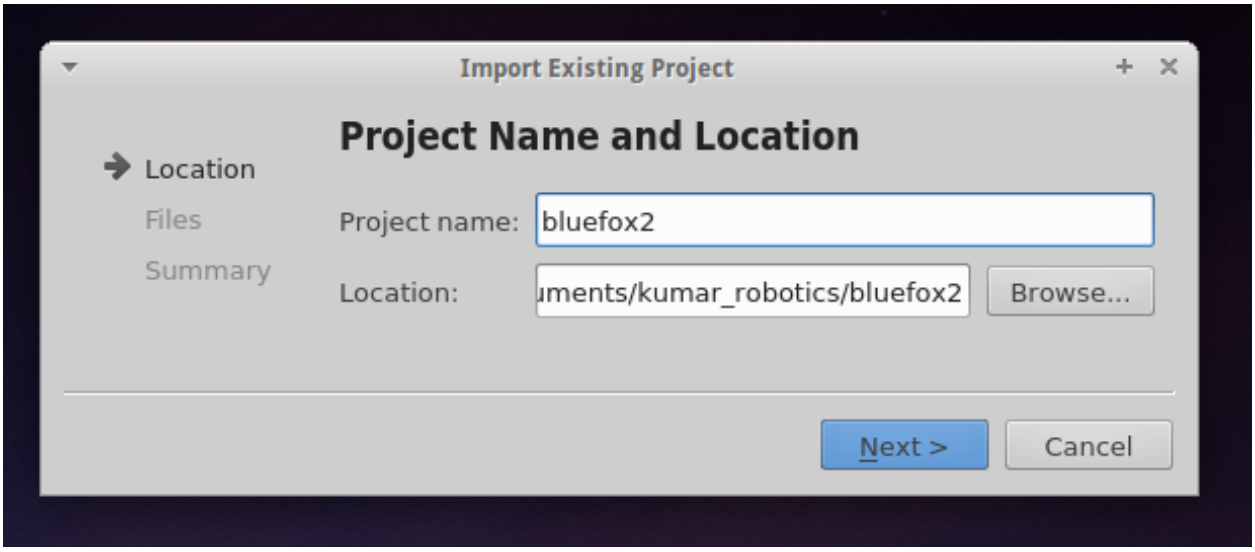
In order to access the necessary environment variables (`ROS_PACKAGE_PATH`, `ROS_MASTER_URI`, etc) Qt Creator must be started from the command line. Alternatively, you can make a desktop launcher by following the instructions here.

4.2.2 ROS Packages in Qt Creator

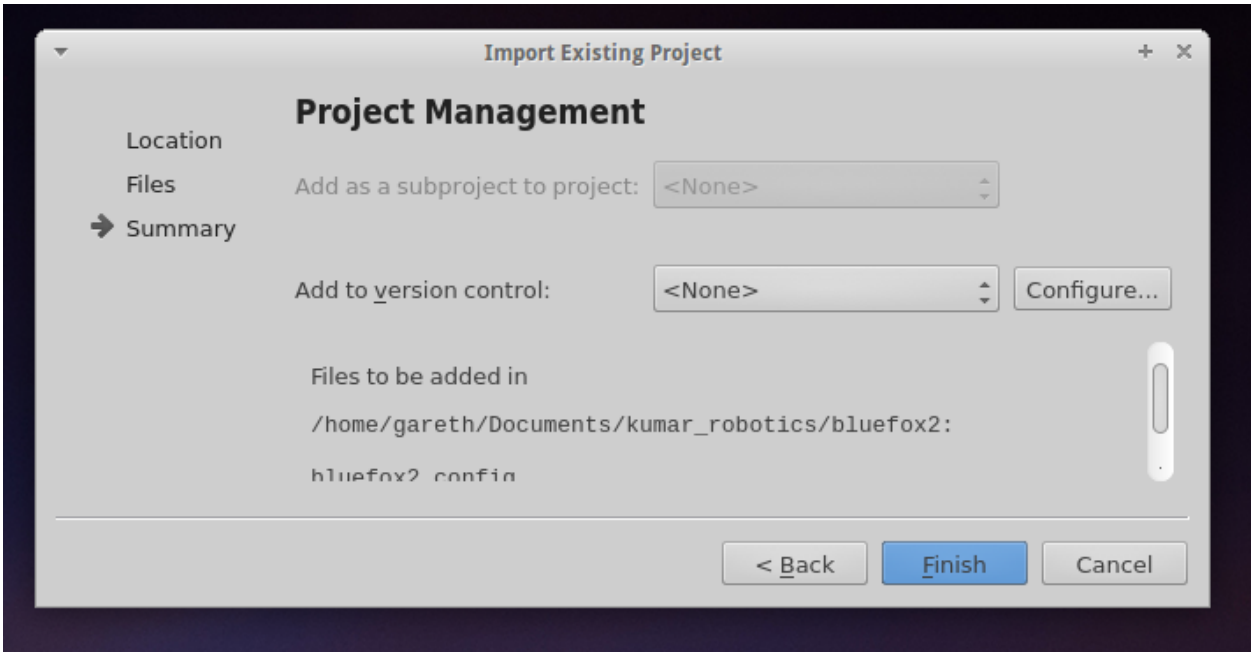
To use Qt Creator with your ROS package. Create a new project with the `File -> New File or Project` action. Select the `Import Project -> Import Existing Project`, as illustrated below:



Assign your project a sensible name, and select the location as the package directory (in which your `src` directory resides):

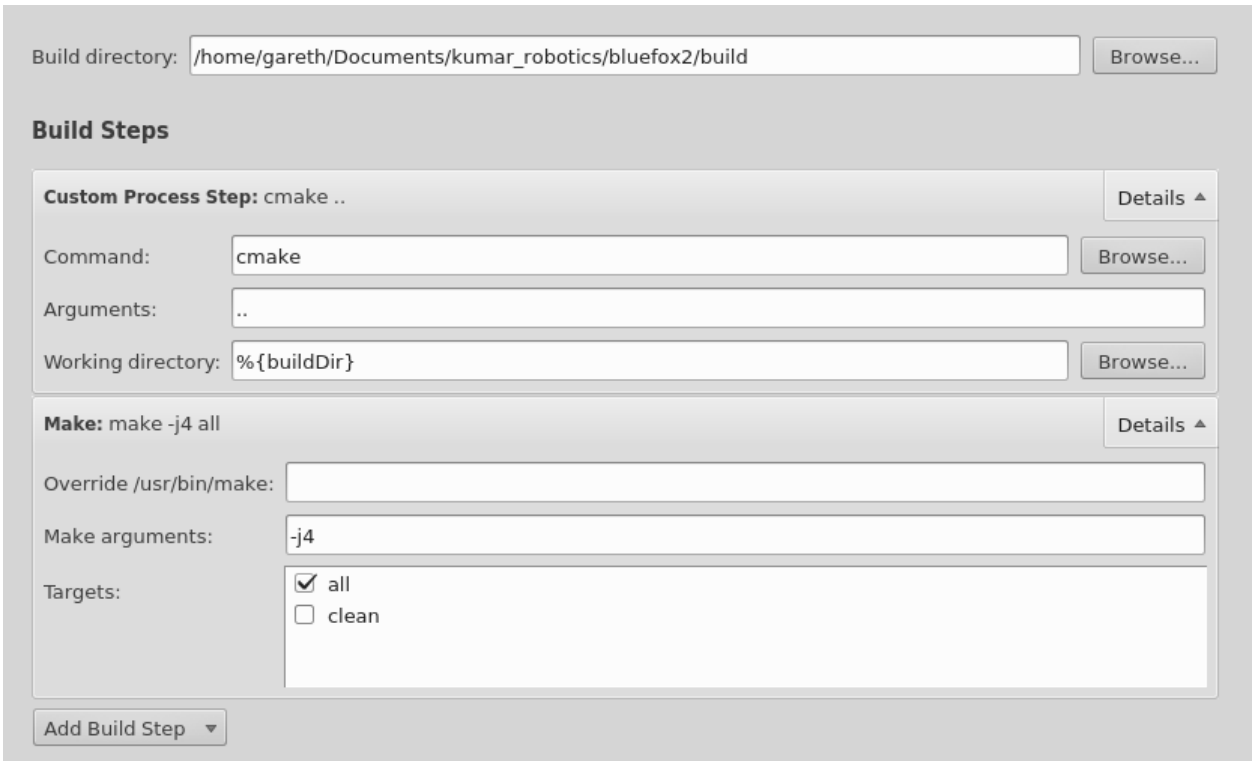


You will be asked to import files. Select directories which contain headers and source files. When prompted to select project options, you should **NOT** add the IDE files to version control - these are specific to your machine.



Lastly, you must configure the Qt Creator build settings correctly. Navigate to the `Projects` menu in the mode selector (accessible from `Window -> Show Mode Selector`).

Append `build` to the build directory (you may need to create this directory yourself). Then add a custom build step which performs the operation `cmake ..` (.. is the argument). You may also optionally add `-j4` to the make arguments in order to parallelize builds. See the image below for an example of valid settings.



If you wish for Qt Creator to index a set of headers that are not on the default path, add them to your `.includes` file in QtCreator. For example:

```
include
build/devel/include
```

will index your project headers and generated project headers (such as messages).

4.2.3 Better Qt Creator

Themes

Qt creator ships with aweful color schemes. You should change it to increase your productivity. Go get qtcreator-themes for better color schemes.

Beautifier

Beautifier is QtCreator plugin that helps you format your code. It supports astyle, clang-format and uncrustify. We recommend using clang-format. To get clang-format, just do:

```
sudo apt-get install clang-format-3.5
```

Go to `Help` -> `About Plugins...`, under `C++` click `Beautifier`. Then restart.

After restart, go to `Tools` -> `Options` -> `Beautifier`, and setup your clang-format.

CHAPTER 5

Common Issues

5.1 Common ROS Problems

5.1.1 GCC fails to find headers for generated messages

You may wish to compile a package (in catkin) which contains **msg** files to be used by other packages (or a package which depends on said messages). The order of compilation is important here. If dependencies are improperly specified, GCC may fail to compile when you include generated message headers.

For example, you might have the package structure:

```
my_workspace/
  my_library/
    msg/MyMessage.msg
    src/my_library.cpp
    CMakeLists.txt
    package.xml
    ...
  my_node/
    src/my_node.cpp
    CMakeLists.txt
    package.xml
    ...
```

Evidently, `my_node` must declare a dependency on `my_library` in both `CMakeLists.txt` and `package.xml`. However, `my_library` and `my_node` should **also** both declare a dependency on the exported targets of `my_library`. Otherwise, catkin may attempt to compile `my_library.cpp` and `my_node.cpp` before the header for `MyMessage.msg` has been generated, causing build failure. Exported targets include the messages and services defined by a package.

This additional dependency is specified by adding the following to both make files:

```
add_dependencies(${PROJECT_NAME} ${catkin_EXPORTED_TARGETS})
```

You may optionally specify the exported targets of only one package, like so:

```
add_dependencies(${PROJECT_NAME} ${my_library_EXPORTED_TARGETS})
```

See here for details.

CHAPTER 6

Indices and tables

- genindex
- search