POLITECNICO DI TORINO

Master of Science degree in MECHATRONIC ENGINEERING

GUIDELINES FOR PIC4SER USERS

PX4 autopilot customization for non-standard gimbal and UWB peripherals

Supervisor:
Marcello Chiaberge

Thesis advisors:
Dott. Ing. Gianluca Dara
Dott. Ing. Simone Silvestro

Candidate:
Francesco Malacarne
s260199

ACADEMIC YEAR 2019-2020

Contents

List of Figures

1 Introduction

2 How to install PX4 1
- 2.1 System I'm working on 1
- 2.2 General advices 1
- 2.3 Installation guide 1

3 How to replicate my tests 5
- 3.1 Preamble 5
- 3.2 Servo testing 5
- 3.3 Driver testing 7
  - 3.3.1 DWM1001 testing description 9
  - 3.3.2 Fault tolerance testing using Arduino 14

4 General advices 17
- 4.1 Tips for custom driver development 17
- 4.2 Subscribing to a topic reading information 18
- 4.3 How to add a simple firmware module 21
- 4.4 How to add a new topic 21
- 4.5 How to integrate my firmware customization into the newest PX4 version? 22
- 4.6 How to start a module at startup 22
- 4.7 How to log a topic on the SD card 23
- 4.8 How to run a bash file 23

5 Useful links 25
- 5.1 Doxygen 25
- 5.2 Official PX4 guides 25
- 5.3 Links related to my work 25

List of Figures

2.1 Forking info 2
2.2 Error during installation 3
2.3 Error example 3

3.1 Connections between Pixracer and servomotor. 6
3.2 Pixracer interfaces 6
3.3 Pixracer port schematic. 7
3.4 QGroundControl AUX1 configuration for RC passthrough. 7
3.5 Driver testing setup using Arduino. 9
3.6 Decawave app configuration needed to run the test. 10
3.7 TELEM2 focus on Pixracer. 11
3.8 DWM1001-dev connection with Pixracer. 11
3.9 PuTTy settings to monitor DWM1001-dev serial communication. 12
3.10 Test output. 13
3.11 Example of PuTTy output during a test. 13
3.12 Real test of the UWB driver performing a circular path. 13
3.13 Connection between Arduino and Pixracer. 15

Introduction

This is a document containing some advices for PIC4SeR users willing to delve into PX4 and/or trying to replicate the tests I performed during my master thesis work. The goal of my project was the customization of the open-source autopilot PX4, for the integration of non-standard peripherals focusing on the implementation of a lightweight camera gimbal on a drone mounting Pixracer flight controller, and the creation of a custom driver to leverage UWB technology for future works about indoor positioning and swarm navigation. I hope you fill find some value in these advices.

My customization

All the details about these components can be found in my thesis, whereas the code can be read freely either on my GitHub private profile (`@francimala`) or on PIC4SeR GitHub profile.

Stabilization module is named `servo_control` and can be found in directory: `Firmware/src/modules/servo_control`.

DWM1001 driver is named `dwm1001` and can be found in directory: `Firmware/src/drivers/distance_sensor/dwm1001`

`rc_update` has been slightly modified, it publishes on a different topic (`actuator_control_rc` instead of `actuator_control`). It can be found in directory: `Firmware/src/modules/rc_update`

`mission_block.cpp` was slightly modified to support 1-axis gimbal. It can be found in directory: `Firmware/src/modules/navigator`

Moreover, 3 topics have been added:

- `actuator_control_rc`
- `dwm1001`
- `dwm1001_raw`

Important note: if you want to integrate these components into the new PX4 version, follow guidelines here: 4.5.

How to install PX4

2.1 System I'm working on

- Linux distro: Ubuntu 18.04.5 LTS
- ROS: installed during procedure but not used
- QGroundControl v4.0.6
- Python2: 2.7.17
- Python3: 3.6.9

2.2 General advices

Before starting I would like to give some general advices based on my experience:

1. It is strongly recommended to use a real machine rather than a virtual machine. It is possible to work in VM, but you will see a huge difference compared to a standard installation, especially if you don't have a top PC.
2. The installation process could be slightly different from the one I'm showing right now due to many factors. If you encounter any error different from the ones I'm going to share in this guide, try to google it finding the solution (I actually did it to install PX4 and to make this guide).
3. Having 8 GB of RAM is suggested, but if you have only 4 GB do not worry, you will probably need to increase the SWAP memory because during the installation you will need more than 4 GB of RAM. During my first installation I used a 4 GB PC but I was not able to complete the set up due to insufficient RAM, then I increased the SWAP memory and I accomplished the installation.
4. Always rely on the official developer guide. It is a very powerful tool although a bit confusing sometimes. Here the link: `https://dev.px4.io/master/en/`.

2.3 Installation guide

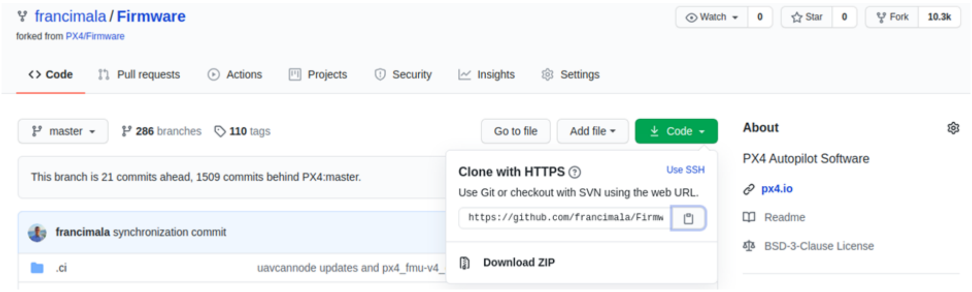
This guide is based on the installation toolchain for Ubuntu users that can be found at the following address: `https://dev.px4.io/master/en/setup/dev_env_linux_` `ubuntu.html`. The first step to install PX4 is to clone the official repository from GitHub. There are different ways to do so, but I'm going to suggest a variation with respect to the one presented in the developer guide, since you will probably need your own GitHub repo containing the changes you made to the original code.

1. If you do not have an account, go on github.com and create one. Being a polito student you have access to some premium features, so check it out (you can also register with your private email and then register as a student adding a second certified email).

2. Fork the official PX4 repository: go here `https://github.com/PX4/Firmware` and press "fork" on the top right corner of the screen. You will create a copy of the firmware into your own repository so that any change you make will not affect the original repository.

Once you forked the repo you can start with the procedure.

1. Go on your GitHub page and copy the address of the forked repo: click on code and press the copy button close to the address.



This branch is 21 commits ahead, 1509 commits behind PX4:master.

francimala synchronization commit

uavcannode updates and px4_fmu-v4_

Clone with HTTPS

Use Git or checkout with SVN using the web URL.

`https://github.com/francimala/Firmw`

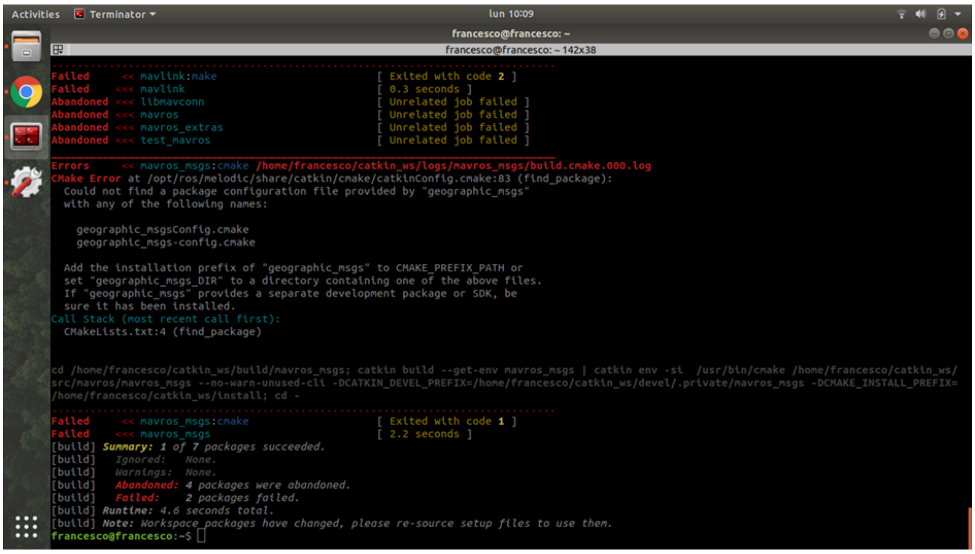
PX4 Autopilot Software

Figure 2.1: Forking example.

2. Now you have to download the Firmware so that you will be able to make changes locally on your PC without interfering with anyone else. You have to choose a location into your PC, I personally installed it into my "home" directory. Open a new terminal and browse to the location you want to install PX4 to, then launch the following command:
   `git clone address --recursive`.
   This command will clone the repo into the directory you decided.

3. After cloning the repository, you can proceed with the installation. Since we may need to use ROS, it is suggested to follow these instructions: `https://dev.px4.io/master/en/setup/dev_env_linux_ubuntu.html#rosgazebo`

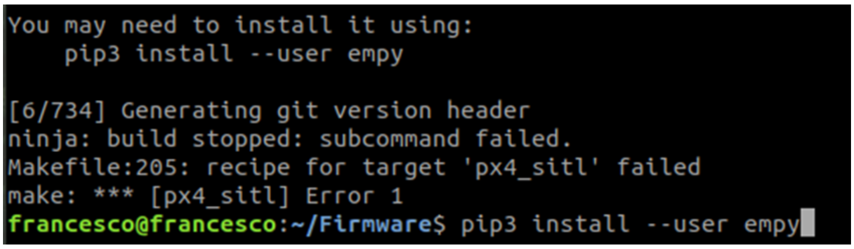
Procedure ended with some errors, but they are not going to create problems.



```
.............................................................................
Failed     << mavlink:make                          [ Exited with code 2 ]
Failed    <<< mavlink                               [ 0.3 seconds ]
Abandoned <<< libmavconn                            [ Unrelated job failed ]
Abandoned <<< mavros                                [ Unrelated job failed ]
Abandoned <<< mavros_extras                         [ Unrelated job failed ]
Abandoned <<< test_mavros                           [ Unrelated job failed ]
_____________________________________________________________________________
Errors     << mavros_msgs:cmake /home/francesco/catkin_ws/logs/mavros_msgs/build.cmake.000.log
CMake Error at /opt/ros/melodic/share/catkin/cmake/catkinConfig.cmake:83 (find_package):
  Could not find a package configuration file provided by "geographic_msgs"
  with any of the following names:

    geographic_msgsConfig.cmake
    geographic_msgs-config.cmake

  Add the installation prefix of "geographic_msgs" to CMAKE_PREFIX_PATH or
  set "geographic_msgs_DIR" to a directory containing one of the above files.
  If "geographic_msgs" provides a separate development package or SDK, be
  sure it has been installed.
Call Stack (most recent call first):
  CMakeLists.txt:4 (find_package)


cd /home/francesco/catkin_ws/build/mavros_msgs; catkin build --get-env mavros_msgs | catkin env -si  /usr/bin/cmake /home/francesco/catkin_ws/
src/mavros/mavros_msgs --no-warn-unused-cli -DCATKIN_DEVEL_PREFIX=/home/francesco/catkin_ws/devel/.private/mavros_msgs -DCMAKE_INSTALL_PREFIX=
/home/francesco/catkin_ws/install; cd -
.............................................................................
Failed     << mavros_msgs:cmake                     [ Exited with code 1 ]
Failed    <<< mavros_msgs                           [ 2.2 seconds ]
[build] Summary: 1 of 7 packages succeeded.
[build]   Ignored:   None.
[build]   Warnings:  None.
[build]   Abandoned: 4 packages were abandoned.
[build]   Failed:    2 packages failed.
[build] Runtime: 4.6 seconds total.
[build] Note: Workspace packages have changed, please re-source setup files to use them.
francesco@francesco:~$ 
```

Figure 2.2: Error during installation.

To test whether the installation was successful, it is enough to launch a simulation running the following commands:
`cd Firmware` (going to the directory where we forked PX4 firmware)
`make px4_sitl gazebo`
If everything was successful, Gazebo should start, otherwise some errors appear. Here I'm proposing the solutions to the errors I faced, after a graphical example of error. PS: every time you solve an error you have to test again the simulation with command:
`make px4_sitl gazebo`



```
You may need to install it using:
    pip3 install --user empy

[6/734] Generating git version header
ninja: build stopped: subcommand failed.
Makefile:205: recipe for target 'px4_sitl' failed
make: *** [px4_sitl] Error 1
francesco@francesco:~/Firmware$ pip3 install --user empy
```

Figure 2.3: Error example.

1. sudo apt install python3-pip
2. pip3 install –user empy
3. pip3 install –user pyros-genmsg
4. pip3 install –user toml
5. pip3 install –user numpy
6. pip3 install –user jinja2
7. sudo apt-get install libgstreamer-plugins-base1.0-dev

As you can see, the first errors are related to python, whereas the last one is related to a missing package needed to compile the code. If you run both bash files presented into the developer guide you will probably face no errors (I ran only the second one because the guide was not clear).

Moreover, you may need to install fastRTPS; follow instructions here: `https://dev.px4.io/master/en/setup/fast-rtps-installation.html#fast-rtps`.

A general suggestion: whenever you make a modification to the code, you have to run the make command to actually test it in both simulation and real hardware. Moreover, each platform has its own build requirements (if you make Pixracer code but you have Pixhawk, the code won't work), so take care of the platform you are using and the command you are typing to build the system.

Once the simulation is performed correctly you should also check that the compilation for Pixracer and Pixhawk are fine. For doing this run the command:

```
make px4_fmu-v4_default
```

If you get no errors you are fine, otherwise you have to solve them. Here some of the errors I got with the corresponding solution.

- Error related to "'__ULong' does not name a type" → install the correct version for the gcc-arm compiler from `https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads`. Download it into the home directory, extract it using the command:

  ```
  tar xjf gcc-arm-none-eabi-9-2020-q2-update-x86_64-linux.tar.bz2
  ```

How to replicate my tests

3.1 Preamble

If you want to replicate my test, you need my Firmware version. You can fork it from `https://github.com/francimala/Firmware`. Remember to build it before using it, otherwise you won't see anything.

3.2 Servo testing

Goal and expected result

The goal of this test is to check the servomotor behavior with respect to the pitch variation of the Pixracer. The expected result is a servomotor movement opposite to the pitch variation of the flight controller. Moreover, setting AUX1 passthrough to channel 7 and moving the knob it is possible to see a variation of the stabilization point.

Needed gear

- Pixracer.
- Servomotor.
- 5V source (Arduino in my case, but it could be whatever source able to provide 5V).
- Radio controller.

Setup

The first step is the connection between the servo and the Pixracer using the 3 equipped cables. According to my setup:

- Orange cable: 5V.
- Yellow cable: signal carrying the position.
- Brown cable: ground.

The orange cable should be connected to the 5V source. In my case such a voltage is provided by the Arduino board, so it is directly connected to the 5V pin available on Arduino. The brown cable must be connected with both the source GND and the

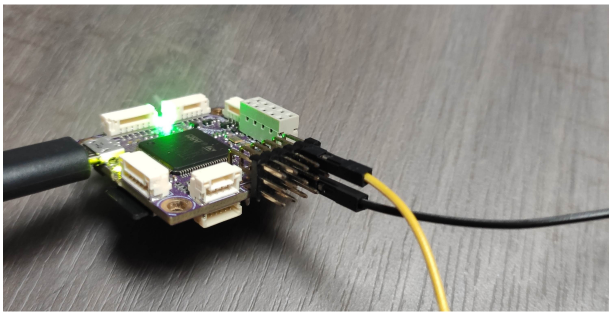


Figure 3.1: Connections between Pixracer and servomotor.

Pixracer GND, so that the three devices (flight controller, servomotor and source) will have the same reference (this is a key point, if missed the test won't work). Lastly, the yellow cable must be connected with the Pixracer signal pin, that is the top one (according to the figure). Once the connection between the servo and the Pixracer is performed, it is possible to plug the flight controller to the PC with the micro USB cable and to open QGC. Running the command << `make px4_fmu-v4_default upload` >> the firmware will be loaded into Pixracer and the module `servo_control` will automatically start. As a consequence, after a few seconds it will be possible to manually change the Pixracer pitch observing the variation of the servo position.

The next two figures are an useful representation of the communication layout.

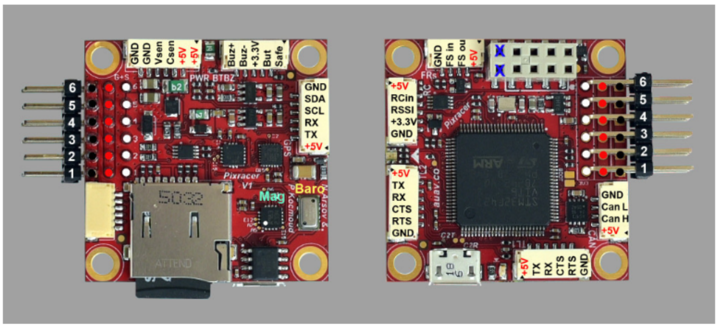


Figure 3.2: Pixracer interfaces. The 6 pins on the right represent the 6 PWM outputs; white is the signal, red is the 5V (present only if there is a battery connected to Pixracer) and black is ground.

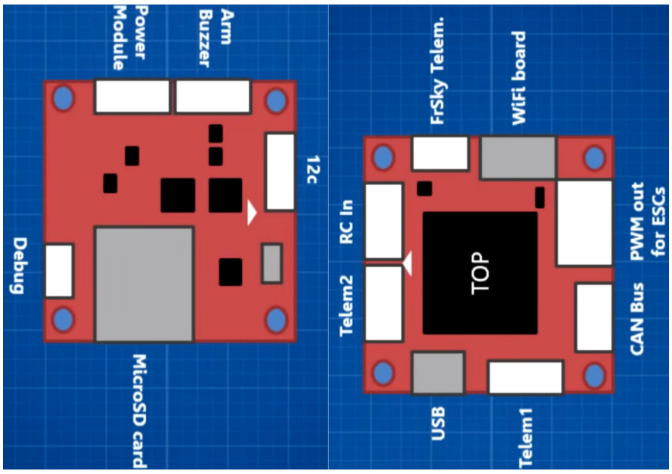


Figure 3.3: Pixracer port schematic.

Lastly, for testing the RC contribution, go on Radio setup in QGroundControl and set channel 7 as AUX1 passthrough (figure 3.4). Switch the RC on and after a while start moving the associated knob; you should start seeing the servo changing its stationary point.

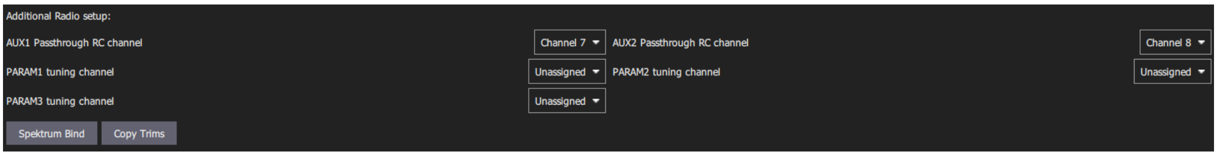

Additional Radio setup:
AUX1 Passthrough RC channel: Channel 7
AUX2 Passthrough RC channel: Channel 8
PARAM1 tuning channel: Unassigned
PARAM2 tuning channel: Unassigned
PARAM3 tuning channel: Unassigned

Figure 3.4: QGroundControl AUX1 configuration for RC passthrough.

3.3 Driver testing

Testing phase was highly affected by the COVID-19 pandemic, which strongly constrained the access to the research lab for master thesis students. Instead of giving up, postponing such a challenging and important part of the work, I decided to find an alternative way to carry the testing part. Being this driver based on a serial communication, I programmed an Arduino MEGA board to publish specific messages over one of its serial ports so that the autopilot could be connected to this port, simulating a connection to a DWM1001 module. The actual features to be tested were mainly 2: the capability of adapting to different number of anchors without losing consistency, and the capability of recognizing the presence of the estimated tag position within the payload.

Before delving into the test details, it is worth to mention the actual message sent over the topic dwm1001.

```
uint64 timestamp          # time since system start (microseconds)
float32[98] distances     # Distances between anchors and tag, each
                            anchor provides 4 measurements (x,y,z
                            coordinate of the anchor with respect
                            between the anchor and the tag).
float32[4] positions      # Position of the TAG into the reference
                            system defined by the anchors (x,y,z,Quality)
uint16 anchor_num         # Number of anchors
bool pos_detected         # Is POS present into the payload?
```

Arduino was programmed to send the following messages, each 4 second spaced:

1. `DIST,1,AN0,1151,5.00,8.00,2.25,6.44\r\n`

   Here the driver should set `anchor_num` equal to 1 and `distances` should have the first 4 values different from 0, whereas the remaining ones equal to 0. POS is not detected, therefore `pos_detected` should be false and `positions` set to 0.

2. `DIST,4,AN0,1151,5.00,8.00,2.25,6.44,AN1,0CA8,0.00,8.00,2.25,6.50,AN2, 111C,5.00,0.00,2.25,3.24,AN3,1150,0.00,0.00,2.25,3.19,POS,200.55,2.01, 100.24,100\r\n`

   Here the driver should set `anchor_num` equal to 4 and `distances` should have the first 16 values different from 0, whereas the remaining ones equal to 0. POS is detected, therefore `pos_detected` should be true and `positions` set to components read into the message.

3. `DIST,4,AN0,1151,5.00,8.00,2.25,6.44,AN1,0CA8,0.00,8.00,2.25,6.50,AN2, 111C,5.00,0.00,2.25,3.24,AN3,1150,0.00,0.00,2.25,3.19\r\n`

   Here the driver should set `anchor_num` equal to 4 and `distances` should have the first 16 values different from 0, whereas the remaining ones equal to 0. POS is not detected, therefore `pos_detected` should be false and `positions` set to 0. This test was made to check the ability of adapting to POS variations. For whatever reason, modules could stop estimating the position and the drone should be able to detect such a condition.

4. `DIST,3,AN0,1151,5.00,8.00,2.25,6.44,AN1,0CA8,0.00,8.00,2.25,6.50,AN2, 111C,5.00,0.00,2.25,3.24,POS,300.55,1.01,6.24,100\r\n`

   Here the driver should set `anchor_num` equal to 3 and `distances` should have the first 12 values different from 0, whereas the remaining ones equal to 0. POS is detected, therefore `pos_detected` should be true and `positions` set to read components. This test was made to check the ability of adapting to number of anchor variations; in a real application it could happen that an anchor stops working or just go outside of coverage, the driver should detect this variation and clear the previous values.

5. `DIST,3,AN0,1151,15.00,800.25,2.25,6.44,AN1,0CA8,100.00,8.00,2.25, 600.50,AN2,111C,5.00,0.00,2.25,3.24\r\n`

   Here the driver should set `anchor_num` equal to 3 and `distances` should have the first 12 values different from 0, whereas the remaining ones equal to 0. POS is not detected, therefore `pos_detected` should be false and `positions` set to 0. This test was made to check the ability of adapting to variable distances (more than 100 meters, even though it is very unlikely that an anchor has such a high coverage).

Tests highlighted the driver capability of adapting to different number of anchors, the presence of POS and the variability of the detected distance.
The used setup is shown in figure 3.5. It is worth to say that contrarily to the connection between Pixracer and DWM1001, the serial link between Pixracer and Arduino was made in a standard approach: Arduino TX connected to Pixracer RX, Arduino Rx connected to Pixracer Tx.

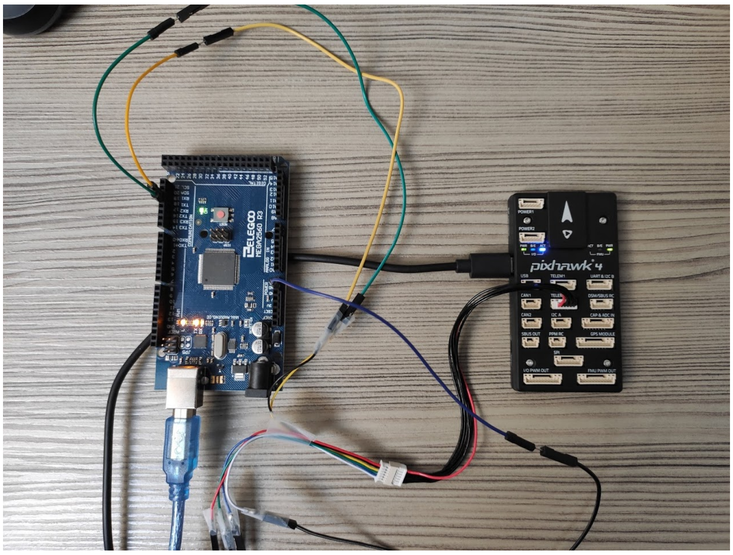


Figure 3.5: Driver testing setup using Arduino.

3.3.1 DWM1001 testing description

Goal and expected results

The goal of this test is to check the driver functionalities using two real DWM1001-dev modules. The expected result is that once the driver is started, the active tag will be programmed to send the relative distance from the anchor publishing this value to the topic `dwm1001`.

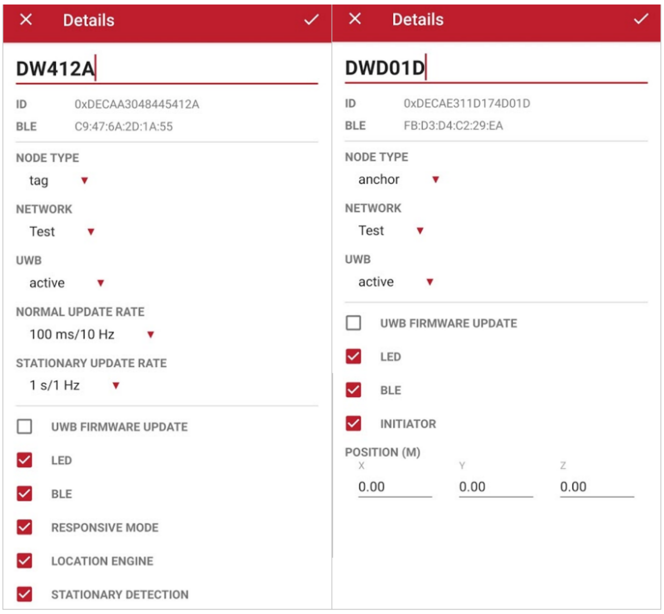
Needed gear

- 2 DWM1001-dev modules.
- Android phone or tablet (not too old) with Decawave app installed.
- Pixracer/Pixhawk board.
- PC with my version of the PX4 Firmware (https://github.com/francimala/Firmware) and QGC.
- 2 micro USB cables.

Setup

The first operation is setting up the 2 DWM1001-dev modules using the Decawave Android App. If you haven't installed it yet, you can download it from this link: `https://www.decawave.com/source-code-for-the-android-application/`. I highly recommend using an up-to-date smartphone because I tried on a 2015 Huawei model and I was not able to complete the setup, although having full compatibility. One module should be set as an active tag (left picture), whereas the other one should be set as an active anchor (initiator, right picture). The anchor should be plugged to a fixed position (like the main network) whereas the tag to the flight controller, supplied by whatever source, either a PC through micro USB cable, or a power bank or a battery (in this case pay attention to the voltage, it must be 3.3V, not 5V!!). My suggestion is to connect it to a PC through micro USB cable so that PuTTy can be used to check what is going on over the serial port.

Important note: if you have any problem with the DRTLS app (you don't see a tag or an anchor) you should try to flash again the firmware on the DWM1001 modules. It happens that if you use a different smartphone from one test to another, it somehow remember the old configuration and the new smartphone will not detect the modules in the correct way. If you need to flash again the firmware, follow page 15 and 16 of the guide that you can find here: `https://www.decawave.com/wp-content/uploads/2019/01/DWM1001-Firmware-User-Guide-2.1.pdf`.



DW412A

ID 0xDECAA3048445412A
BLE C9:47:6A:2D:1A:55

NODE TYPE: tag
NETWORK: Test
UWB: active
NORMAL UPDATE RATE: 100 ms/10 Hz
STATIONARY UPDATE RATE: 1 s/1 Hz

☐ UWB FIRMWARE UPDATE
☑ LED
☑ BLE
☑ RESPONSIVE MODE
☑ LOCATION ENGINE
☑ STATIONARY DETECTION

DWD01D

ID 0xDECAE311D174D01D
BLE FB:D3:D4:C2:29:EA

NODE TYPE: anchor
NETWORK: Test
UWB: active

☐ UWB FIRMWARE UPDATE
☑ LED
☑ BLE
☑ INITIATOR

POSITION (M)
X 0.00
Y 0.00
Z 0.00

Figure 3.6: Decawave app configuration needed to run the test.

Connect flight controller to the DWM1001-dev module using the TELEM2 port of the Pixracer/Pixhawk (within the code the default port I used is /dev/ttyS2, but you can change it if you need it). Figures 3.3, 3.2 and ?? may help you finding the right port on the board, especially Pixracer, because Pixhawk is pretty clear.

The connection should be the following one:

- Black cable is the Pixracer/Pixhawk ground and must be connected to DWM1001-dev GND (third pin starting from above).
- Yellow cable is Pixracer/Pixhawk TX and must be connected to DWM1001-dev TX (fourth pin).
- Green cable is Pixracer/Pixhawk RX and must be connected to DWM1001-dev RX (fifth pin).

Now it is possible to start powering devices up.

1. Plug the anchor to the main net.
2. Plug the tag to the PC.

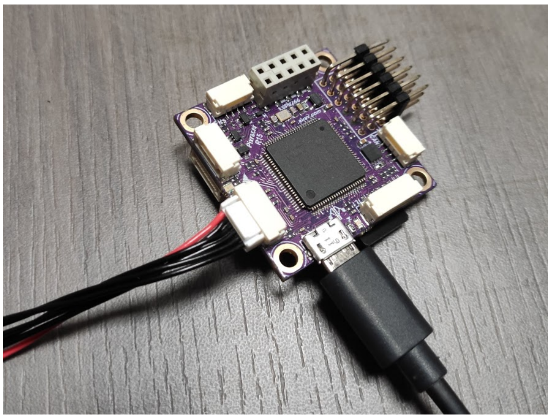


Figure 3.7: TELEM2 focus on Pixracer.

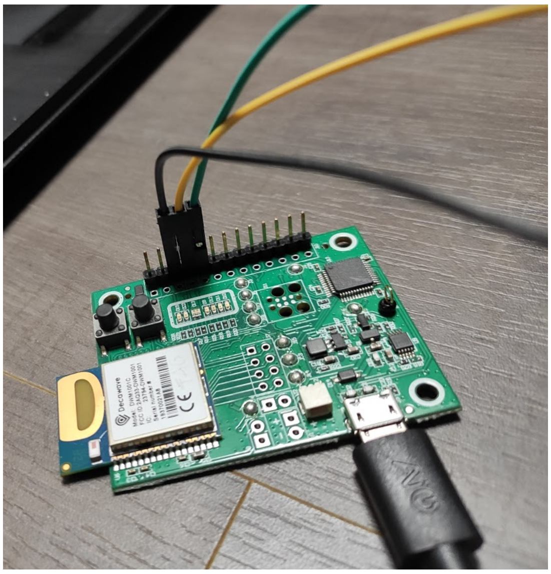


Figure 3.8: DWM1001-dev connection with Pixracer.

3. Plug the flight controller to the PC.

   If you don't need PuTTy you can jump to point 4, otherwise If you want to use PuTTy to monitor what's going on the DWM1001-dev module you can open a serial communication following these 3 steps:

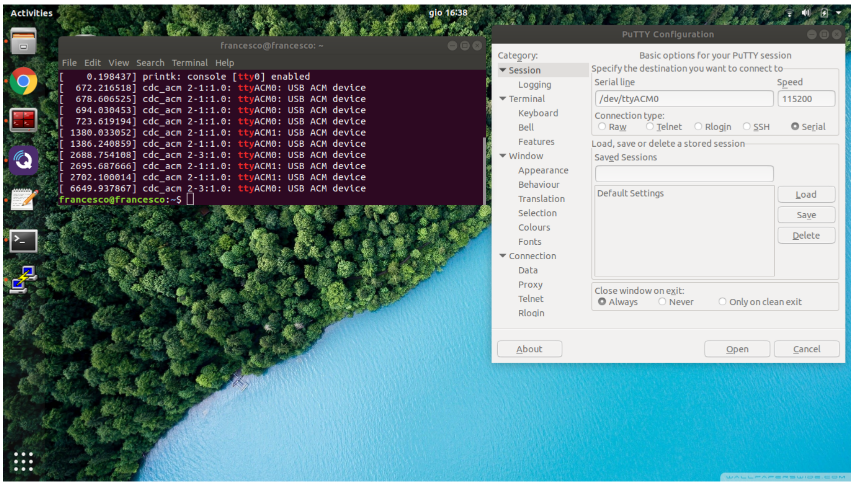
   (a) Open PuTTy and click on "Serial". Then, if you don't know the device name follow point 2, otherwise jump to point 3.

   (b) Open a new terminal and launch command `<< dmesg | grep tty >>`: the last two devices should be the flight controller and the DWM1001-dev module; in principle you should use the last-but-one, but if you want to be sure you can unplug the flight controller and launch again the command `<< dmesg | grep tty >>`, the last device should be the correct one (ACM... or ttyS...).

   (c) Write the name you discovered and set the baudrate to 115200.

4. Open QGroundControl.

5. Go into the MAVLINK console.



```
[    0.198437] printk: console [tty0] enabled
[  672.216518] cdc_acm 2-1:1.0: ttyACM0: USB ACM device
[  678.606525] cdc_acm 2-1:1.0: ttyACM0: USB ACM device
[  694.030453] cdc_acm 2-1:1.0: ttyACM0: USB ACM device
[  723.619194] cdc_acm 2-1:1.0: ttyACM0: USB ACM device
[ 1380.033052] cdc_acm 2-1:1.0: ttyACM1: USB ACM device
[ 1386.240859] cdc_acm 2-1:1.0: ttyACM0: USB ACM device
[ 2688.754108] cdc_acm 2-3:1.0: ttyACM0: USB ACM device
[ 2695.687666] cdc_acm 2-1:1.0: ttyACM1: USB ACM device
[ 2702.100014] cdc_acm 2-1:1.0: ttyACM1: USB ACM device
[ 6649.937867] cdc_acm 2-3:1.0: ttyACM0: USB ACM device
francesco@francesco:~$ 
```

Basic options for your PuTTY session

Specify the destination you want to connect to

Serial line: /dev/ttyACM0

Speed: 115200

Connection type: ○ Raw ○ Telnet ○ Rlogin ○ SSH ● Serial

Load, save or delete a stored session

Saved Sessions:

Default Settings

Close window on exit: ● Always ○ Never ○ Only on clean exit

Figure 3.9: PuTTy settings to monitor DWM1001-dev serial communication.

6. Type `<< dwm1001 start >>` to start the driver. You should see something happening on PuTTy if you have prepared it. If you read on the MAVLink console *Got the beginning!* it means that the module was programmed correctly and the serial messages started to flow.

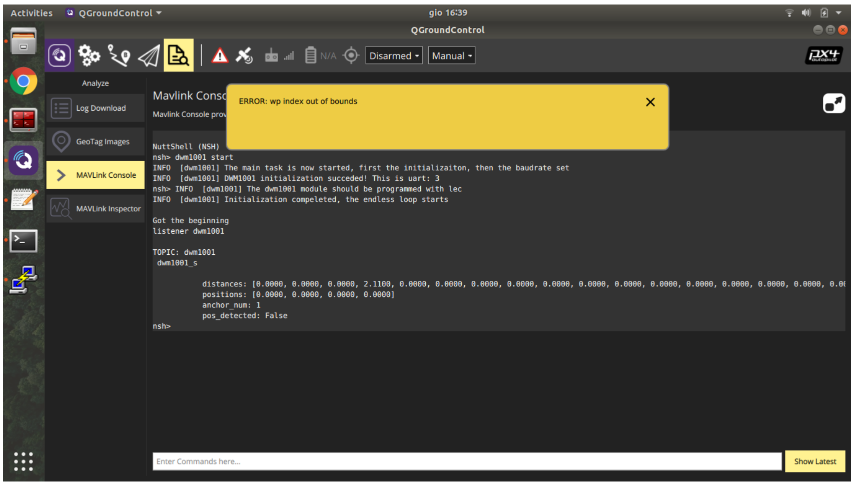
7. Type `<< listener dwm1001 >>` to check whether the driver is working correctly.

The test is now complete and it should be possible to see results similar to the ones in figures 3.10 and 3.11.

Testing 4 anchors and 1 tag

Only at the end of my working period I had the opportunity to test the basic system on the complete setup (4 anchors and 1 tag). Testing setup is basically the same as the one presented in this subsection, but with more anchors. The tag should be always connected to the drone and set as active tag. One anchor should be set as active and initiator, whereas the 3 other anchors shall be set as simple active anchors (no initiator). Moreover, indications about the relative position between anchors should be set by means of the DRTLS Android App; this is an important step, as all the measurements will depend on this setting. For doing this in the best way, it is convenient to place them in a rectangular shape all at the same level. However, the Android app will guide you through the installation step, so do not worry. The test starts again with the same command: `<< dwm1001 start >>` and the result should be the same as the one obtained in the Arduino testing subsection.

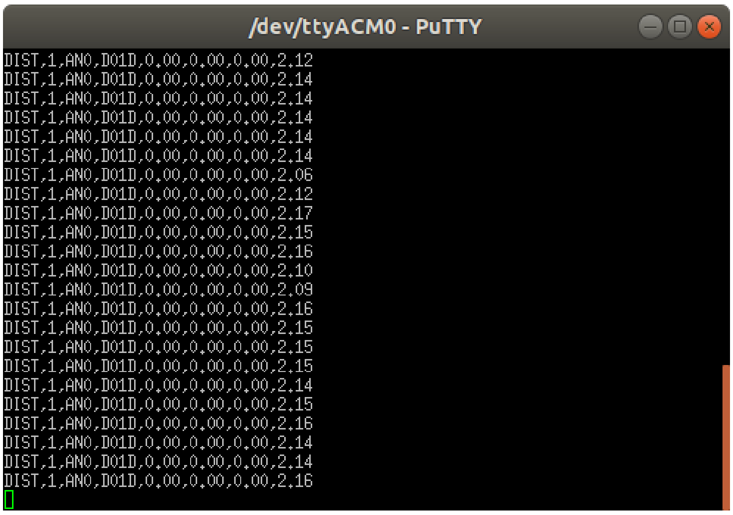
For testing purposes I forced logging on SD of the dwm1001 topic so that the estimated position could be seen and compared with visual measurements. An example showing a circular path is presented in figure 3.12.



ERROR: wp index out of bounds

```
NuttShell (NSH)
nsh> dwm1001 start
INFO  [dwm1001] The main task is now started, first the initializaiton, then the baudrate set
INFO  [dwm1001] DWM1001 initialization succeded! This is uart: 3
nsh> INFO  [dwm1001] The dwm1001 module should be programmed with lec
INFO  [dwm1001] Initialization compeleted, the endless loop starts

Got the beginning
listener dwm1001

TOPIC: dwm1001
 dwm1001_s

           distances: [0.0000, 0.0000, 0.0000, 2.1100, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0
           positions: [0.0000, 0.0000, 0.0000, 0.0000]
           anchor_num: 1
           pos_detected: False
nsh>
```

Figure 3.10: Test output.



```
DIST,1,AN0,D01D,0.00,0.00,0.00,2.12
DIST,1,AN0,D01D,0.00,0.00,0.00,2.14
DIST,1,AN0,D01D,0.00,0.00,0.00,2.14
DIST,1,AN0,D01D,0.00,0.00,0.00,2.14
DIST,1,AN0,D01D,0.00,0.00,0.00,2.14
DIST,1,AN0,D01D,0.00,0.00,0.00,2.14
DIST,1,AN0,D01D,0.00,0.00,0.00,2.06
DIST,1,AN0,D01D,0.00,0.00,0.00,2.12
DIST,1,AN0,D01D,0.00,0.00,0.00,2.17
DIST,1,AN0,D01D,0.00,0.00,0.00,2.15
DIST,1,AN0,D01D,0.00,0.00,0.00,2.16
DIST,1,AN0,D01D,0.00,0.00,0.00,2.10
DIST,1,AN0,D01D,0.00,0.00,0.00,2.09
DIST,1,AN0,D01D,0.00,0.00,0.00,2.16
DIST,1,AN0,D01D,0.00,0.00,0.00,2.15
DIST,1,AN0,D01D,0.00,0.00,0.00,2.15
DIST,1,AN0,D01D,0.00,0.00,0.00,2.15
DIST,1,AN0,D01D,0.00,0.00,0.00,2.14
DIST,1,AN0,D01D,0.00,0.00,0.00,2.15
DIST,1,AN0,D01D,0.00,0.00,0.00,2.16
DIST,1,AN0,D01D,0.00,0.00,0.00,2.14
DIST,1,AN0,D01D,0.00,0.00,0.00,2.14
DIST,1,AN0,D01D,0.00,0.00,0.00,2.16
```

Figure 3.11: Example of PuTTy output during a test.

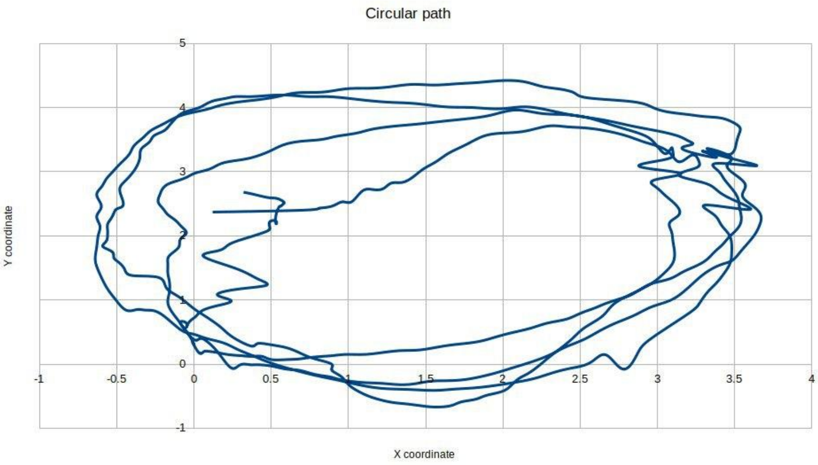


Figure 3.12: Real test of the UWB driver performing a circular path.

To summarize:

1. Make sure the firmware is correctly uploaded on Pixracer (`make px4_fmu-v4_default upload`).

2. Power on all the modules and program them with the DWM app: set all the initial positions (measuring them) and check that one anchor is the initiator (active), and the tag is an active tag.

3. Power on the Pixracer and the stabilization module `servo_control` will automatically start. The driver will not be initialized automatically, it must be started manually using a specific command within the MAVLink console. However, before starting it, the logger shall be started.

4. Start the logger using `logger on`.

5. Start the driver using `dwm1001 start`. You will be sure about the initialization when you see ”Got the beginning” on the MAVLink console. If you don’t, the driver did not recognize the serial message coming from the tag, which probably means that the connection is not correct.

6. When the flight mission is over, you can read the log file accessing the microSD log directory. For the conversion use `pyulog` (`https://github.com/PX4/pyulog`). Copy the correct .ulg file in a directory you know and run the command `ulog2csv ulog file.ulg` in a new terminal. This procedure will create a csv human-friently document in that folder containing all the logs. Then you can use Matlab or Python to plot the desired values.

Driver explained (italian): `https://youtu.be/0GsJtd7PwT8`
UWB testing: `https://www.youtube.com/watch?v=FuytZULFfAo&ab_channel=FrancescoMalacarne`
Connection between Arduino and DWM1001-dev: `https://github.com/francimala/Arduino-MEGA-to-DWM1001`

3.3.2 Fault tolerance testing using Arduino

Goal and expected results

The goal of this test is to make a sort of fault tolerance test of the system simulating wrong messages sent over the serial port. The expected result is a the same as the DWM1001-dev: PX4 should think of being receiving messages coming from DWM1001-dev.

Needed gear

- Arduino MEGA (or whatever Arduino having more than 1 RX-TX channels, Arduino UNO cannot be used).

- Pixracer

- PC with my version of the PX4 Firmware (`https://github.com/francimala/Firmware`) and QGC.

Setup

Before plugging anything to the PC, it is necessary to connect Arduino and PX4. For doing this, only three cables are needed: TX, RX and GND (the same GND must be guaranteed linking them together). Conversely with respect to the previous test where TX cables had to be linked together, here the connection is slightly different:

- Black cable is the Pixracer/Pixhawk ground and must be connected to Arduino MEGA GND.
- Yellow cable is Pixracer/Pixhawk TX and must be connected to Arduino MEGA RX.
- Green cable is Pixracer/Pixhawk RX and must be connected to Arduino MEGA TX.

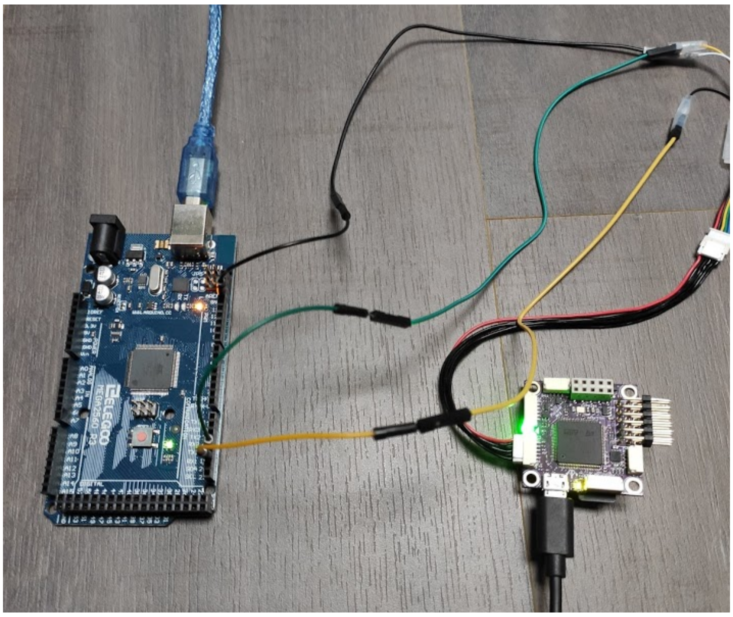


Figure 3.13: Connection between Arduino and Pixracer.

Once completed the connection, both Arduino and PX4 can be plugged to the PC using their specific cables. Arduino should be programmed with the sketch I wrote that is downloadable at this link. Whereas Pixracer must be equipped with my PX4 version. Once everything is uploaded, it is possible to start the test by opening QGC and starting the driver into the MAVLINK console (using the same command used in the previous test `<< dwm1001 start >>`). The result should be the same as before, but with different payloads periodically.

General advices

4.1 Tips for custom driver development

This is a small general guide that could be useful for writing a custom driver. A driver is that part of the firmware that allows you to either use data coming from a specific hardware, or to actuate a control strategy by programming peripherals to work in a certain way. Unfortunately, as stated into the official developer guide in this page `https://dev.px4.io/master/en/middleware/drivers.html`, the easiest way to implement new drivers is to start from an already existing one, because there is no a universal procedure to do so. A list of all the driver's source codes can be found at `https://github.com/PX4/Firmware/tree/master/src/drivers`. Before starting to read and understand all the drivers, it is suggested to have a clear idea about the driver to be developed. In particular, answering these questions may be a good starting point:

1. Which is the communication protocol used for the communication between the hardware and the flight controller?

2. Is it sufficient to power up the peripheral or is it necessary to send specific commands to let it work?

3. How does the payload look like?

The answers to these questions can be generally found inspecting the hardware datasheet. Once the idea about the driver to be developed is clear, I would start developing a first raw version to be deployed on Arduino (if the communication protocol is suitable), due to its simplicity (the code I created for Arduino was about 30 lines, whereas the final PX4 driver is more than 400). Then, you should take a look at the user guide under the peripheral section, because you can find the list of the components for each communication protocol `https://docs.px4.io/master/en/peripherals/`; this is useful because you can directly dive into the relevant drivers instead of wasting time under other codes not inherent with your work. For instance I only looked for the UART drivers so that I2C, UAVCAN etc. drivers could be avoided. Once you have a clear idea of the drivers to be used as reference, you can start analysing these codes line by line. At the end of this analysis you should be able to understand the purpose of each function and whether it is useful for your application. Lastly you can start writing your driver implementing a similar version to the one you wrote with Arduino.

4.2 Subscribing to a topic reading information

Goal

The goal is to subscribe to a topic. A topic is a sort of information container where different modules can publish information and of course can subscribe to. Whenever you subscribe to a topic, you basically read all the information inside it. Whenever you publish into a topic, you update its information; after having updated information a subscriber can read updated information.

Procedure

This guide is created based on the official guide at `https://dev.px4.io/v1.9.0/en/apps/hello_sky.html`.

In order to create a new script to subscribe to a topic and read the information exchanged within the messages, it is firstly necessary to create the file into the correct directory:

1. Create a new directory `Firmware/src/examples/directory_name`.
2. Create a C file named `directory_name.c` within the created directory.

In order to understand better, an example is carried on. The name for the directory is `px4_camera_attitude` instead of `directory_name`, and the subscription topic is `mount_orientation.h`.

After the first declaration part (entirely shown in the complete code), it is necessary to include the uORB library and the topic we want to subscribe to; in this case, since we want to subscribe to `mount_orientation.h`, we add the following two lines:

```
#include <uORB/uORB.h>
#include <uORB/topics/mount_orientation.h>
```

The entire list of topics and messages can be found at the following links:

- Messages: `https://github.com/PX4/Firmware/tree/master/msg/`
- Topics in the local directory: `Firmware/build/px4_sitl_default/uORB/topics`

Once completed the declaration, the main function can be exported and created. The name of the main function must be equal to the name of the directory, adding the writing `main`, in this case: `px4_camera_attitude_main`. Then write the code you need according to the application you have to write.

To resume, a practical procedure to subscribe to a specific topic may be the following one:

1. Create a new directory `Firmware/src/examples/directory_name`.
2. Create a C file named `directory_name.c` within the created directory.
3. Copy the entire file you can find after this list.
4. Use the tool “Find and replace” to replace all the writings `mount_orientation` with the topic you want to subscribe to.

5. Use the tool “Find and replace” to replace all the writings `camera_attitude` with the name of your directory (`directory_name`).
6. At line 69 you have to change `attitude_euler_angle` with the actual value you want to read within the message `mount_orientation.msg`.
7. Notice that the structure `mount_orientation_s` defined at line 65 is specified within the file `mount_orientation.h`.

The following code is used to get the orientation of the camera (euler angles) from the topic `mount_orientation.h`.

```
1  #include <px4_platform_common/px4_config.h>
2  #include <px4_platform_common/tasks.h>
3  #include <px4_platform_common/posix.h>
4  #include <unistd.h>
5  #include <stdio.h>
6  #include <poll.h>
7  #include <string.h>
8  #include <math.h>
9
10 #include <uORB/uORB.h>
11 #include <uORB/topics/mount_orientation.h>
12
13 __EXPORT int px4_camera_attitude_main(int argc, char *argv[]);
14
15 int px4_camera_attitude_main(int argc, char *argv[])
16 {
17   PX4_INFO("I'm going to read the mount orientation.");
18
19   /* subscribe to mount_orientation topic */
20   int sensor_sub_fd = orb_subscribe(ORB_ID(mount_orientation));
21   /* limit the update rate to 5 Hz */
22   orb_set_interval(sensor_sub_fd, 200);
23
24
25   /* one could wait for multiple topics with this technique,
26   just using one here */
27   px4_pollfd_struct_t fds[] = {
28     { .fd = sensor_sub_fd,   .events = POLLIN },
29     /* there could be more file descriptors here,
30     in the form like:
31     * { .fd = other_sub_fd,   .events = POLLIN },
32     */
33   };
34
35   int error_counter = 0;
36
37
38   for (int i = 0; i < 1; i++) {
39
40     /* wait for sensor update of 1 file descriptor for
41     1000 ms (1 second) */
42     int poll_ret = px4_poll(fds, 1, 2000);
43
44     if(poll_ret == 0) {
      // None of our providers provided us data
      PX4_ERR("Got no data within a second");
    }

    else if (poll_ret < 0) {

      /* this is seriously bad - should be an emergency */
      if (error_counter < 10 || error_counter % 50 == 0) {
        /* use a counter to prevent flooding
        (and slowing us down) */
        PX4_ERR("ERROR return value from poll(): %d", poll_ret);
      }

      error_counter++;

    }

    else {
      if (fds[0].revents & POLLIN) {
        /* obtained data for the first file descriptor */
        struct mount_orientation_s raw;
        /* copy sensors raw data into local buffer */
        orb_copy(ORB_ID(mount_orientation), sensor_sub_fd, &raw);
        PX4_INFO("Camera attitude:\t%8.4f\t%8.4f\t%8.4f",
          (double)raw.attitude_euler_angle[0],
          (double)raw.attitude_euler_angle[1],
          (double)raw.attitude_euler_angle[2]);
      }
    }
  }

  PX4_INFO("exiting");

  return 0;
}
```

The application is now complete. In order to run it you first need to make sure that it is built as part of PX4. Applications are added to the build/firmware in the appropriate board-level cmake file for your target:

- PX4 SITL (Simulator): `Firmware/boards/px4/sitl/default.cmake`.
- Pixhawk v1/2: `Firmware/boards/px4/fmu-v2/default.cmake`.
- Pixracer (px4/fmu-v4): `Firmware/boards/px4/fmu-v4/default.cmake`.
- cmake files for other boards can be found in `Firmware/boards/`.

To enable the compilation of the application into the firmware create a new line for your application somewhere in the cmake file: `examples/directory_name`, or `examples/px4_camera_attitude` for our case.

```
px4_add_module(
    MODULE examples__px4_camera_attitude
    MAIN px4_camera_attitude
    STACK_MAIN 2000
    SRCS
        px4_camera_attitude.c
    DEPENDS
    )
```

4.3 How to add a simple firmware module

The goal of this section is to understand how to introduce a new piece of firmware in PX4. In this example I'm going to add a new module named `px4_simple_app` into the examples directory. However, the procedure is the same to add either new modules (like `servo_control` in my thesis) or new drivers (like `dwm1001` in my thesis). This guide was created based on the section "Writing your first application" of the PX4 developer guide (`https://dev.px4.io/master/en/apps/hello_sky.html`).

1. Create a new directory where you want your file to be, in this case: `Firmware/src/examples/px4_simple_app`.
2. Create a new C file in that directory named with the name you want the commands to be called, in this case: `px4_simple_app.c`
3. Insert the header and write your code.
4. Before testing it you need to perform two more operations. The first one is to create a CMakeList.txt file where you insert the code placed into the guide.
5. The application is now complete. In order to run it you first need to make sure that it is built as part of PX4. Applications are added to the build/firmware in the appropriate board-level cmake file for your target:
    - PX4 SITL (Simulator): `Firmware/boards/px4/sitl/default.cmake`
    - Pixhawk v1/2: `Firmware/boards/px4/fmu-v2/default.cmake`
    - Pixracer (px4/fmu-v4): `Firmware/boards/px4/fmu-v4/default.cmake`
    - cmake files for other boards can be found in `Firmware/boards/`
6. To enable the compilation of the application into the firmware create a new line for your application somewhere in the cmake file:
   `examples/px4_simple_app`

More information on how to integrate a new module (italian): `https://youtu.be/IgG5OOK1Tyw`

4.4 How to add a new topic

This guide is based on the uORB part of the PX4 developer guide: `https://dev.px4.io/master/en/middleware/uorb.html#adding-a-new-topic`. To add a new topic, you need to first create a new .msg file in the `msg/` directory and add the file name to the `msg/CMakeLists.txt` list. From this, the needed C/C++ code is

automatically generated. In fact, the actual topic located into the build directory (`Firmware/build/.../uORB/Topics`) will be automatically created based on the message file definition.
Have a look at the existing msg files for supported types.
To each generated C/C++ struct, a field `uint64_t timestamp` will be added. This is used for the logger, so make sure to fill it in when publishing the message.
To use the topic in the code, include the header:
`#include <uORB/topics/topic_name.h>`

4.5 How to integrate my firmware customization into the newest PX4 version?

The first step is downloading the new firmware version from the official PX4 repo: `https://github.com/PX4/Firmware`. Then, the integration of the new modules and the new topic must be completed.
Go into the my modules directory (`Firmware/src/modules`) and copy the `servo_control` directory into the modules directory of the new firmware. Then, move into my `rc_update` module and copy and paste the .cpp and .h files in the same folder of the new version. Then, go into the distance sensor directory `Firmware/src/drivers/distance_sensor` and copy and paste the dwm1001 directory. So far, all the new modules have been physically added, however, PX4 will not compile them when launching the build command, because they are not specified into the `default.cmake` file. In order to compile them, browse to the platform of interest (in the case of Pixracer go into `Firmware/boards/px4/fmu-v4`), open the `default.cmake` and add under the section DRIVERS the name dwm1001, and servo_control under the MODULES section; this step must be replicated for all the used platforms (also Pixhawk, fmu-v5). At this stage all modules can be correctly compiled. Nevertheless, some topics must be added to the default ones. For doing this, browse to the `msg` directory and copy and paste from my firmware version the following .msg files:

- `actuator_control_rc`
- `dwm1001`
- `dwm1001_raw`

Similarly to the module integration, these names must be added into the `CMakeLists.txt` file located in the same folder. All the customization have now been introduced.

How to integrate my customization (italian): `https://youtu.be/V-wbH9gnAvw`

4.6 How to start a module at startup

Procedure taken from `https://dev.px4.io/master/en/concept/system_startup.html#starting-additional-applications`. You have to connect the microSD card of the Pixracer/Pixhawk board to a PC, go into the etc directory (if not present create it) then edit the file `extras.txt` (if not present create it) inserting the on of the two following lines:

- If you want to abort boot if this module is not started correctly:
  ```
  custom_app start
  ```
- If you don't want to abort boot if this module is not started correctly:
  ```
  set +e
  custom_app start
  set -e
  ```

In my case I did not want to abort boot if any error occurred with `servo_control`, therefore I went for the second option.

4.7 How to log a topic on the SD card

Procedure taken from `https://dev.px4.io/master/en/log/logging.html`. Logging a topic is a fundamental feature that PX4 offers. For doing this, you have to specify the topics you want to log in a specific file, named `logger_topics.txt`, placed in the microSD card into the directory `etc/logging/logger_topics.txt`. Logging starts when arming the drone, but you can start it before, using the command `logger on`.

Once you have the ULog file (.ulg), you can convert it into csv (you will obtain a number of csv files equal to the number of topic you wrote into the topic logger textual file). For the conversion use `pyulog` (`https://github.com/PX4/pyulog`). Copy the correct .ulg file in a directory you know and run the command `ulog2csv ulog_file.ulg` in a new terminal in that directory to obtain the csv files related to each topic.

General remark: always publish the current time in your topic messages using `hrt_absolute_time()` function for the timestamp. For instance, if you have a structure named `example`, always add the line `example.timestamp = hrt_absolute_time()` before publishing the message.

4.8 How to run a bash file

There are multiple ways to run bash files, here I'm presenting the one I'm more familiar with. Write the bash script, then save it as `file_name.sh`, open the terminal, change the directory into the one where the bash file was saved, then use the following commands:

```
chmod +x ./file_name.sh
./file_name.sh
```

Useful links

5.1 Doxygen

Doxygen is an extremely powerful tool useful for understanding relationships between modules and topics. A practical example that I used during my work was the following one: I had two modules publishing on the same topic (`actuator_control`), but I only knew one of them. To discover the other one I wrote `actuator_control` in Doxygen and I discovered that module `RC_update` was the second module publishing on `actuator_control`. You can find it here: `https://px4.github.io/Firmware-Doxygen/dc/d61/md_src_drivers_uavcan_uavcan_drivers_stm32__r_e_a_d_m_e.html`, just type what you need into the search bar on the top right corner.

5.2 Official PX4 guides

Developer guide: `https://dev.px4.io/master/en/`
User guide: `https://docs.px4.io/master/en/`
GQroundControl user guide: `https://docs.qgroundcontrol.com/master/en/index.html`
MAVLink guide: `https://mavlink.io/en/`

5.3 Links related to my work

My firmware version: `https://github.com/francimala/Firmware`
Stabilization module explained (italian): `https://youtu.be/IgG5OOK1Tyw`
Driver explained (italian): `https://youtu.be/0GsJtd7PwT8`
How to integrate my customization (italian): `https://youtu.be/V-wbH9gnAvw`
How does mixing work in PX4? `https://www.youtube.com/watch?v=N97bxWtoPJ8&ab_channel=FrancescoMalacarne`
UWB testing: `https://www.youtube.com/watch?v=FuytZULFfAo&ab_channel=FrancescoMalacarne`
Connection between Arduino and DWM1001-dev: `https://github.com/francimala/Arduino-MEGA-to-DWM1001`