Offboard control PX4 MPC

Prakrit Tyagi REx Lab

November 2022

1 Installation

Note: These steps were followed on a fresh **Ubuntu 22.04.3 LTS** install and I have also noted down some errors that I encountered.

Webpages from PX4 docs to refer PX4-ROS 2 User Guide, PX4 Ubuntu Development Environment setup and Git PX4 examples. Always read the docs corresponding to your PX4 firmware version. Px4 docs are different for each firmware version. If you enter the PX4 docs from browser you will enter into main branch of doc which corresponds to the current in-development version of PX4. Just bookmark your version of doc for easy access.

Initially : Python 3.10.12 available, gcc g++ not available, Java JDK not available. As this is a fresh ubuntu install.
Lets begin our setup!!!

1. Install git

```
~$ sudo apt install git
```

2. Clone the PX4 firmware. I used **1.14 release** which used uXRCE-DDS middleware for Ros2 communication.

```
~$ mkdir Developer
~$ cd Developer
~/Developer/$ git clone https://github.com/PX4/PX4-Autopilot.git
~/Developer/$ cd PX4-Autopilot
~/Developer/PX4-Autopilot$ git fetch origin release/1.14
~/Developer/PX4-Autopilot$ git checkout release/1.14
```

3. Run ubuntu.sh file to setup the development. This will install all the dependencies given in requirements.txt and other tools for simulation (Gazebo) and cmake, gcc, g++, Java, eigen etc.

```
~/Developer/PX4-Autopilot$ bash ./Tools/setup/ubuntu.sh
```

4. Reboot Computer or re-login before next step. Then update submodules.

```
~/Developer/PX4-Autopilot$ make submodulesclean
```

This takes a while without any output on terminal. Leave it for 15mins to finish.

5. Install the QGroudControl application. http://qgroundcontrol.com/

6. Install Ros 2 Humble and Ros 2 Dev tools. To install ROS 2 "Humble" on Ubuntu 22.04 follow offical docs.

   - Always source the setup.bash file of ROS2 in each terminal before using ROS2.

   ```
   $ source /opt/ros/humble/setup.bash
   ```

   Also ROS2 comes with FAST DDS which is the middleware used by PX4.

7. Some Python dependencies must also be installed (using pip or apt)

```
pip install --user -U empy pyros-genmsg setuptools
```

8. Setup XRCE-DDS Agent and Client. For ROS 2 to communicate with PX4, a XRCE-DDS client must be running on PX4, connected to an XRCE-DDS agent running on the companion computer. The PX4 come with the XRCE-DDS client automatically.

   - Open a terminal
   - Enter the following commands

```
git clone https://github.com/eProsima/Micro-XRCE-DDS-Agent.git
cd Micro-XRCE-DDS-Agent
mkdir build
cd build
cmake ..
make
sudo make install
sudo ldconfig /usr/local/lib/
```

   - Test

```
MicroXRCEAgent udp4 -p 8888
```

9. Troubleshooting : If there are missing dependencies run these. Though ubuntu bash and ros 2 should have installed all dependencies.

```
$ sudo apt install python3-colcon-common-extensions
$ sudo apt install ros-foxy-eigen3-cmake-module
```

10. Test: Checkpoint 1

    - Start the agent with settings for connecting to the XRCE-DDS client running on the simulator:

```
$ MicroXRCEAgent udp4 -p 8888
```

    - Open a new terminal in the root of the PX4 Autopilot repo that was installed above and run command below. For humble start a PX4 Gazebo simulation using:

```
make px4_sitl gz_x500
```

    Note: To understand more about PX4 builds look at this page Building PX4 Software.

11. **Building ROS 2 Workspace for our projects**. This section shows how to create a ROS 2 workspace hosted in your home directory (modify the commands as needed to put the source code elsewhere). The px4_ros_com and px4_msgs packages are cloned to a workspace folder, and then the colcon tool is used to build the workspace.

    - Create a workspace directory using

```
$ mkdir -p ~/ws_project_name/src/
$ cd ~/ws_project_name/src/
```

    - Clone the ROS 2 bridge packages px4_ros_com and px4_msgs to the /src directory (the master branch is cloned by default). You should match the versions of PX4 firmware and PX4 msgs versions:

```
$ git clone -b release/v1.14 https://github.com/PX4/px4_ros_com.git
$ git clone -b release/1.14 https://github.com/PX4/px4_msgs.git
```

    Commit ids (Editor please check these....)

      – For px4_msgs

```
8362da45380032ec8efe993ba6fb941bf2fcde64
```

      – For px4_ros_com

```
b1014f354777177711d9e907c664f9e2c4e710f4
```

    - Source the ROS 2 development environment into the current terminal and compile the workspace using

```
cd ..
source /opt/ros/humble/setup.bash
colcon build
```

Now source the setup.bash file before using any binary from this project.

12. Test: Checkpoint 2

  (a) Start the sim from root folder of PX4-Autopilot

```
cd Developer/PX4-Autopilot
make px4_sitl gz_x500
```

  (b) Start Agent

```
$ MicroXRCEAgent udp4 -p 8888
```

  (c) Open a new terminal and start a "listener" using the provided launch file:

```
$ ros2 launch px4_ros_com sensor_combined_listener.launch.py
```

  If the bridge is working correctly you will be able to see the data being printed on the terminal/console where you launched the ROS listener. Now debugvector and offboard control examples can be run. We can also make our own ros2 examples and test them in simulation before implementing on the hardware.

2 Directly publish to motors using ROS2 for Simulation

- Adding actuator motor topic to firmware:

  We need to make changes to dds_topics.yaml file present in src/modules/uxrce_dds_client folder. Add the below lines in the file so that PX4 firmware can subscribe to his topic when we publish to it using ROS2.

```
- topic: /fmu/in/actuator_motors
type: px4_msgs::msg::ActuatorMotors
```

  Not every topic is present in this file. So add any topic that you want to use. All topic PX4 topics

- disabled lockstep if using gazebo-classic

  We need to make a change to a file iris.sdf.jinja present in Tools/simulation/gazebo-classic/models/iris folder. Disable lockstep by changing 1 to 0.

- write a new node file to publish to actuator motors

  github link to file............

Error : Landing detected when running LQR.cpp: Temp fix Change value for the param

```
COM_DISARM_LAND 2->0
```

3 LQR implementation

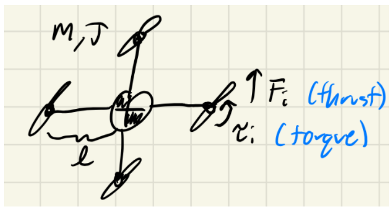


Figure 1: quadcopter

Quadcopter model: Below is mathematics of a standard quadcopter model with control as thrust values of individual motors i.

$$X = \begin{bmatrix} {}^{N}r \\ {}^{N}q^{B} \\ {}^{B}V \\ {}^{B}\omega \end{bmatrix}, F_i = k_T * u_i, \tau = k_\tau * u_i \tag{1}$$

where,

- position is in N frame.
- attitude is B to N frame.
- linear velocity is in B frame.
- angular velocity is in B fram.
- Z axis is up.

Kinematic equations:

$$
\begin{aligned}
{}^N\dot{r} &= {}^N V = Q\,{}^B V \\
\dot{q} &= \frac{1}{2} q \times \hat{\omega} = \frac{1}{2} L(q) H\,{}^B\omega = \frac{1}{2} G(q)\,{}^B\omega
\end{aligned}
$$

Translational dynamics:

$$m\,{}^N\dot{V} = {}^N F$$

Now we need to rotate the V in B frame. so .....

$$
\begin{aligned}
&{}^N V = Q\,{}^B V \\
&\Rightarrow {}^N\dot{V} = \dot{Q}\,{}^B V + Q\,{}^B\dot{V} = Q\hat{w}\,{}^B V + Q\,{}^B\dot{V} \\
&\Rightarrow {}^B\dot{V} = Q^T\,{}^N\dot{V} - {}^B\omega \times {}^B V \\
&\Rightarrow {}^B\dot{V} = \frac{1}{m}\,{}^B F - {}^B\omega \times {}^B V \\
&{}^B F = Q^T \begin{bmatrix} 0 \\ 0 \\ -m_g \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ k_T & k_T & k_T & k_T \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}
\end{aligned}
$$

rotation dynamics: This equation comes from Euler's equation to give total torque.

$$
\begin{aligned}
J\,{}^B\dot{\omega} + {}^B\omega \times J\,{}^B w &= {}^B\tau \\
{}^B\dot{\omega} &= J^{-1}\left({}^B\tau - {}^B\omega \times J\,{}^B\omega\right)
\end{aligned}
$$

$$
{}^B\tau = \begin{bmatrix} lK_T\left(u_2 - u_4\right) \\ lk_T\left(u_3 - u_1\right) \\ k_2\left(u_1 - u_2 + u_3 - u_3\right) \end{bmatrix}
$$

Now I have made some changes before using these equations as the ros topics that give back feedback are in NED frame. As defined below.

- position is in NED global frame
- quaternion is body to global frame
- velocity is in NED global frame
- angular velocity is is NED body frame.

Also the angular velocity equation depends on how your motors are numbered and what configuration of quad are you flying(X ? +). Keep this in mind.

changes:

- change the sign of gravity in global frame.
- change the input equation according to your configuration in body frame.

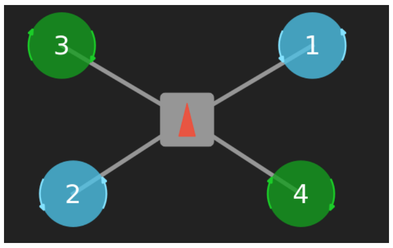


Figure 2: Configuration of actuators

This tells us how to change the dynamics equations.

$$^BF = Q^T \begin{bmatrix} 0 \\ 0 \\ m_g \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ k_T & k_T & k_T & k_T \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}$$

$$^B\dot{\omega} = J^{-1} \left(^B\tau - {}^B\omega \times J^B\omega \right)$$

$$^B\tau = \begin{bmatrix} -l * k_\tau & l * k_\tau & l * k_\tau & -l * k_\tau \\ l * k_\tau & -l * k_\tau & l * k_\tau & -l * k_\tau \\ k_\tau & k_\tau & -k_\tau & -k_\tau \end{bmatrix} * u_{4\times1}$$

- Finally when using the feedback from ros convert the velocity from global frame(N) to body frame(B).

NEXT do linearization and discretization to get A and B matrix.

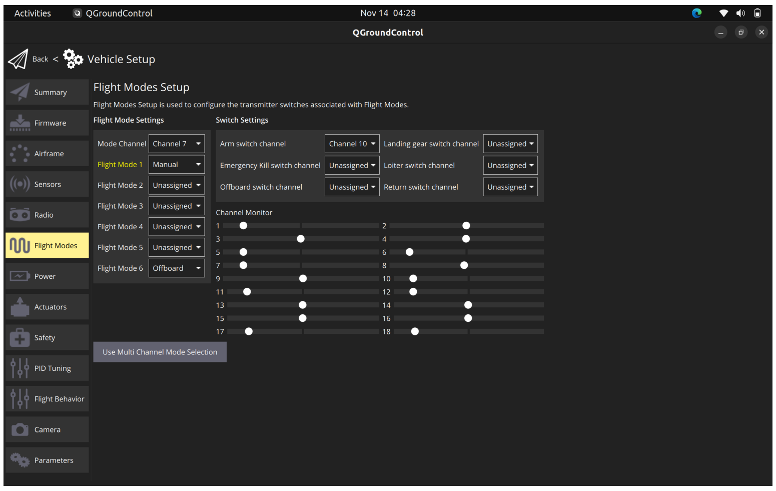
A:

B:

4 Hardware implementation

I tried this on a fresh Ubuntu 22 desktop install on Raspberry pi 4 with 4G of swap space.

1. Flash Pixhawk with PX4. REPO: release/1.14. We need to make more changes on top of what we did above to run it on hardware for our project. It would be best to create two branches one for sim and one for hardware.
   (a) First get the release branch: (repeated again for clarity)
      i. Clone the PX4-Autopilot repo and navigate into PX4-Autopilot directory:
      ```
      git clone https://github.com/PX4/PX4-Autopilot.git
      cd PX4-Autopilot
      ```
      Note:You can reuse an existing repo rather than cloning a new one. In this case clean the build environment (see changing source trees):
      ```
      make clean
      make distclean
      ```
      ii. Fetch the desired release branch. For example, assuming you want the source for PX4 v1.14:
      ```
      git fetch origin release/1.14
      ```
      iii. Checkout the code for the branch
      ```
      git checkout release/1.14
      ```
      iv. Update submodules:
      ```
      make submodulesclean
      ```
   (b) Build Px4 for hardware and upload it to pixhawk or any PX4 compatible hardware. I used a FMUv5 based pixhwak 4. So the build command for this flight controller is
   ```
   make px4_fmu-v5_default
   ```
   connect to pixhawk using a usb and first upload a clean build to check for any issues. Use the below command.
   ```
   make px4_fmu-v5_default upload
   ```
   (c) Open QGroundControl and run through the setup and calibrate the sensors. Make sure that the drone can be armed by the RC transmitter.
2. Establish Communication: First we need to establish communication between Raspberry pi and pixhwak. Follow this **Link**.
   ```
   sudo MicroXRCEAgent serial --dev /dev/serial0 -b 921600
   ```



Flight Modes Setup

Flight Modes Setup is used to configure the transmitter switches associated with Flight Modes.

Flight Mode Settings

Mode Channel	Channel 7
Flight Mode 1	Manual
Flight Mode 2	Unassigned
Flight Mode 3	Unassigned
Flight Mode 4	Unassigned
Flight Mode 5	Unassigned
Flight Mode 6	Offboard

Switch Settings

Arm switch channel	Channel 10	Landing gear switch channel	Unassigned
Emergency Kill switch channel	Unassigned	Loiter switch channel	Unassigned
Offboard switch channel	Unassigned	Return switch channel	Unassigned

Figure 3: Setup the flight modes exactly like this.

```
sudo apt-get install openssh-server
```

ssh into raspberry pi to run the above command. Make sure you are connected to the same network.

```
ssh cmurexlab@192.168.123.149
```

Check with ros2 topics list if you are able communicate. You will get an output like this.

```
cmurexlab@cmurexlab:~$ ros2 topic list
        /fmu/in/obstacle_distance
        /fmu/in/offboard_control_mode
        /fmu/in/onboard_computer_status
        /fmu/in/sensor_optical_flow
        /fmu/in/telemetry_status
        /fmu/in/trajectory_setpoint
        /fmu/in/vehicle_attitude_setpoint
        /fmu/in/vehicle_command
        /fmu/in/vehicle_mocap_odometry
        /fmu/in/vehicle_rates_setpoint
        /fmu/in/vehicle_trajectory_bezier
        /fmu/in/vehicle_trajectory_waypoint
        /fmu/in/vehicle_visual_odometry
        /fmu/out/failsafe_flags
        /fmu/out/position_setpoint_triplet
        /fmu/out/sensor_combined
        /fmu/out/timesync_status
        /fmu/out/vehicle_attitude
        /fmu/out/vehicle_control_mode
        /fmu/out/vehicle_local_position
        /fmu/out/vehicle_odometry
        /fmu/out/vehicle_status
        /parameter_events
        /rosout
```

3. Now makes these changes and flash pixhawk again.
   (a) In the file dds_topics.yaml add topics: rc_channels to read and actuator_motors to publish using ros2. Also make sure vehicular angular velocity topics is uncommented.
   (b) In the ControlAllocator.cpp file make these changes to run() function.

```
// Publish actuator setpoint and allocator status
// publish_actuator_controls();
if(_rc_channels_sub.updated())
{
```

```
rc_channels_s rc_channels{};
_rc_channels_sub.copy(&rc_channels);
rc_channel_7 = rc_channels.channels[6]; // rc channel 7 value
}
if (rc_channel_7 < 0.0f)
{
/* code */
publish_actuator_controls();
}
```

corresponding change is ControlAllocator.h file.

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

uORB::Subscription _rc_channels_sub{ORB_ID(rc_channels)};
float rc_channel_7; /**< rc channel 7 value */
```

Test this by running actuator_publisher.cpp node from raspberry pi. Start the publisher and use you channel 7 in rc to change modes.

4. Streaming Mocap data(Optitrack): Install mocap_ros2 package in your local computer. Now start streaming mocap data from a mocap computer on a network. Run mocap2ros publisher in your local computer by using given commands. Local computer should be on the same network.

```
ros2 launch mocap_optitrack_driver optitrack2.launch.py
```

```
ros2 lifecycle set /mocap_optitrack_driver_node activate
```

Connect raspi to same wifi and check the published topics

5. If mocap2ros publisher is working and the bridge connection between raspi and Pixhwak is made then check the output of

```
ros2 topic list
```

be something like this...

```
cmurexlab@cmurexlab:~$ ros2 topic list
/fmu/in/actuator_motors
/fmu/in/obstacle_distance
/fmu/in/offboard_control_mode
/fmu/in/onboard_computer_status
/fmu/in/sensor_optical_flow
/fmu/in/telemetry_status
/fmu/in/trajectory_setpoint
/fmu/in/vehicle_attitude_setpoint
/fmu/in/vehicle_command
/fmu/in/vehicle_mocap_odometry
/fmu/in/vehicle_rates_setpoint
/fmu/in/vehicle_trajectory_bezier
/fmu/in/vehicle_trajectory_waypoint
/fmu/in/vehicle_visual_odometry
/fmu/out/failsafe_flags
/fmu/out/rc_channels
/fmu/out/sensor_combined
/fmu/out/timesync_status
/fmu/out/vehicle_angular_velocity
/fmu/out/vehicle_attitude
```

```
/fmu/out/vehicle_control_mode
/fmu/out/vehicle_local_position
/fmu/out/vehicle_odometry
/fmu/out/vehicle_status
/markers
/mocap_control
/mocap_environment
/mocap_optitrack_driver_node/transition_event
/parameter_events
/rigid_bodies
/rosout
```

NOTE: Use screen in ssh for multiplexing through shell. If screen not installed install it in raspi.

6. When building px4_ros_com in raspberry pi dont forget to add mocap_msg package in src.

5 EKF2 PX4 fusion

Mocap only gives location and attitude. To get full state we need to fuse this data with PX4 sensors to get full state feedback.

6 Params set or changed

```
MAV_1_CONFIG = 0 (Disabled)
UXRCE_DDS_CFG = 102 (TELEM2)
SER_TEL2_BAUD = 921600
```

Augmented Lagragian was not fast.
New Approach: ADMM(alternating direction method of multipliers). Check stephan boid's book.

```
cd ~/Downloads/cmake-3.12.0-rc3/   # or wherever you downloaded cmake
./bootstrap --prefix=$HOME/cmake-install
make
make install
export PATH=$HOME/cmake-install/bin:$PATH
export CMAKE_PREFIX_PATH=$HOME/cmake-install:$CMAKE_PREFIX_PATH
```

error while installing mocap4ros2_optitrack

```
solution:
sudo apt install ros-humble-ament-clang-format ros-humble-ament-cmake-clang-format
```

7 Flight Checklist

1. First establish communication between raspi and pixhawk
2. Run the optitrack pub so that GPS can be bypassed
3. then calibrate sensors. After every crash recalibrate sensors.
4. Put props.
5. Wear eye glases and fly at safe distance.