Flying a Pentacopter with Loss of a Motor

Tao Du
taodu@csail.mit.edu

July 23, 2018

Contents

1 Introduction

This document describes my attempt to upgrading my pentacopter demo so that it can recover from the loss of an motor during the flight. The final flight test video is here. In that video, the copter first took off with all of its five motors working. At roughly 00:09 I turned off the rotor closest to the camera and the copter became equivalent to a nonsymmetric quadcopter. I then turned the rotor on and turned it off again at roughly 00:25. There were glitches during the transition but overall the copter was still controllable and was not greatly affected by the sudden changes in motors.

This document is mostly about explaining all the changes I made to the C++ code in ArduCopter to make this demo work (No changes were made in the hardware platform). Moreover, this document also briefly describes how to fly this upgraded pentacopter in the flight test with/without the extra motor.

2 Code Structure

The whole work-flow is actually pretty straightforward:

- The copter first operates normally as a pentacopter;
- At some point, I use a discrete switch (channel 6 in my experiment) in my RC transmitter to notify the copter that I want to shut down a motor;

- There is a loop (100Hz if I recall correctly) in ArduCopter that regularly checks the radio input from my transmitter. When it captures the change in channel 6, it calls my function to update the motor and the controller accordingly;
- In that function, I disable a certain motor in ArduCopter so that it cannot spin. This simulate the loss of a motor during the flight;
- In the meantime, I change the frame class from a pentacopter to a customized, asymmetric quadcopter. Concretely, this is done by recalculating the contributions of each of the remaining, working motors;
- The copter now flies as if it is a quadcopter.

The final firmware code is here. Once you check out the `holodeck-demo` branch, please check out the `loss_of_motor_demo` tag and follow the instructions in `README` to compile and upload the firmware to your flight controller. Below I will describe all the changes I made to the C++ code step by step.

2.1 Monitor the radio signal

You need a transmitter that has at least 6 channels. Usually, channel 1 to 4 send continuous signals to control roll, pitch, throttle, and yaw of your copter. Channel 5 is a switch that can send out discrete signals and is typically reserved for changing flight modes. Channel 6 is also a discrete channel and can be used for your own purposes (Figure 1), so I picked this channel to turn on/off the extra motor.

Figure 1: Channel 6 in our RC transmitter. It is a three-position switch but we only need two positions in this experiment.

The RC loop that listens to all the channels is located in ArduCopter/ArduCopter.cpp:

```
// rc_loops - reads user input from transmitter/receiver
// called at 100hz
void Copter::rc_loop()
{
```

```
    // Read radio and 3-position switch on radio
    // ------------------------------------------
    read_radio();
    read_control_switch();
    // Tao Du
    // taodu@csail.mit.edu
    // Jul 22, 2018
    // trigger the switch of 'frame_class' in pentacopter to simulate the loss of one rotor.
    read_frame_class_switch();
}
```

Listing 1: Adding our switch function in the RC loop

This `rc_loop` function is called at 100Hz and it is the natural place to insert our switch function. In the `read_frame_class_switch` function, we need to check the latest signal from Channel 6, compare it to its old value to see if it changes, and plan according. This function is implemented in `ArduCopter/switches.cpp`:

```
void Copter::read_frame_class_switch()
{
    // calculate position of flight mode switch
    int8_t switch_position;
    uint16_t rc6_in = RC_Channels::rc_channel(CH_6)->get_radio_in();
    if      (rc6_in < 1231) switch_position = 0;
    else if (rc6_in < 1750) switch_position = 1;
    else switch_position = 2;

    // We assume 0 -> penta, 1 -> penta with loss of motor (quad), 2 -> ignore.
    bool frame_class_switch_changed = switch_position != frame_class_switch_state;
    AP_Motors::motor_frame_class current_class = (AP_Motors::motor_frame_class)g2.frame_class.get();
    if ((current_class != AP_Motors::MOTOR_FRAME_PENTA && current_class != AP_Motors::MOTOR_FRAME_PENTA_QUAD) || !frame_class_switch_changed || switch_position == 2)
    return;

    // Update the frame class.
    if (switch_position == 0) {
        // Reset everything to penta.
        g2.frame_class.set(AP_Motors::MOTOR_FRAME_PENTA);
    } else {
        // Reset everything to penta_quad.
        g2.frame_class.set(AP_Motors::MOTOR_FRAME_PENTA_QUAD);
    }
    motors->reset_frame_class_and_type((AP_Motors::motor_frame_class)g2.frame_class.get(), (AP_Motors::motor_frame_type)g.frame_type.get());
    frame_class_switch_state = switch_position;
}
```

Listing 2: Triggering the change of motors

Note that we introduced two macro definitions specifically for our pentacopter demo: `MOTOR_FRAME_PENTA` represents the regular pentacopter with five working motors, and `MOTOR_FRAME_PENTA_QUAD` is for the pentacopter after losing one rotor. We do not want our changes to influence the other frame classes (e.g., quad, hexa, octa, etc.), so we proceed only if our current frame class is one of them and the switch position has changed, which is a clear sign that we need to turn on/off one motor and update the controller.

2.2 Change the frame class

Ideally, we would want to auto-detect the motor failure from ESCs, or from abnormal IMU responses, or even from a optical sensor next to each motor. This turned out to be a lot harder than I had thought before. As a result, I simply send signals to turn on or off a motor and "simulate" the loss of its power in the air, and switch to the other controller at the same time. The corresponding code is shown below:

```
void AP_MotorsMatrix::reset_frame_class_and_type(motor_frame_class frame_class,
    motor_frame_type frame_type)
{
    // exit immediately if armed or no change
    // Tao Du
    // taodu@csail.mit.edu
    // Jul 22, 2018
    // allow for the change of frame_class if it is our pentacopter demo.
    if (frame_class != MOTOR_FRAME_PENTA &&
        frame_class != MOTOR_FRAME_PENTA_QUAD &&
        (armed() || (frame_class == _last_frame_class && _last_frame_type ==
    frame_type))) {
        return;
    }
    _last_frame_class = frame_class;
    _last_frame_type = frame_type;

    // setup the motors
    setup_motors(frame_class, frame_type);
}
```

Listing 3: Resetting the frame class

As can be seen here, this `reset_frame_class_and_type` function simply reinitializes all of the motors by calling `setup_motors`. This is almost the same as `set_frame_class_and_type` in the same file, except that the `set_` function additionally initializes the rotor output frequency by calling `set_update_rate`. In my experiment, I noticed that calling `set_update_rate` multiple times resulted in crazily oscillating rotor output signals at very high frequency. This is why I introduced a separate `reset_` function here. So now `set_` is called only once when the `Copter` class initializes everything, and `reset_` is called whenever the signal from Channel 6 is changed.

The remaining task is to add a new frame class that represents the pentacopter losing one motor in `setup_motors`. This is straightforward:

```
        case MOTOR_FRAME_PENTA_QUAD: {
            // This describes the same pentacopter as above but with motor 3 disabled.
            add_motor(AP_MOTORS_MOT_1, 72, -1, 1.44721364f, 1);
            add_motor(AP_MOTORS_MOT_2, 144, 1, 0.55278636f, 2);
            add_motor(AP_MOTORS_MOT_4, -144, -1, 0.55278636f, 3);
            add_motor(AP_MOTORS_MOT_5, -72, 1, 1.44721364f, 4);
            success = true;
            break;
        }
```

Listing 4: Adding a new frame class

Compared to the original pentacopter, this new frame disables motor 3. We choose to disable this motor instead of others because the remaining quadcopter is still fully controllable. Disabling some other motors, e.g., motor 2, leaves us 3 clockwise rotors and 1 counterclockwise rotor only. In this case, the resulting quadcopter is not fully controllable and the best thing we can do is giving up on

yaw control and leveling itself and maintaining its altitude. This sounds like a risky experiment so I did not touch it this time.

3 Flight Test

The flight test video is here. The only new operation during this flight test was that I needed to toggle the Channel 6 switch on my transmitter. I noticed that the copter was flying quite stably when I turned motor 3 off at 00:09 and 00:25 (Figure 2). However, when I turned it on again, you can see from the video that there was a disruptive glitch at 00:13 and 00:35. The copter ended up stabilizing itself though, but I needed to leave some space for this adjustment. I guess a smoother blend instead of a sudden switch between the two motor classes could partially alleviate this issue but I didn't try it in this experiment.

Figure 2: Screenshot of the copter that turned off motor 3 in the air.

4 Acknowledgment

Thank Jie Xu, my labmate, for taking the video.