Crash Course Paparazzi 2016
Creating code

Lesson 3. . . Making decisions

Roland Meertens, Christophe de Wagter, Guido de Croon

Februari 2016

Introduction

Last week you could to let your drone follow a flight plan. You were manually placing your waypoints, and manually selecting what block to fly. This week will spice things up by letting the drone decide what to do. To do this we will try writing code, but not before more explanations of Paparazzi are given.

Goals of this week

- Know how and where to get help
- Adding a module
- Learning how to debug
- Making decisions onboard
- Moving waypoints

How to get help

As you start working with the Paparazzi autopilot you will get loads of questions. Paparazzi is written in the C language, and it can be a big hurdle to start programming in this language. I recommend reading the book "Head First C". Although the book is full with strange images, the explanations are very clear.
Here is where you can get help if you are stuck:

- If you can't find something in Paparazzi use the "Find in file" function of Eclipse. This will help you find the files or variables you are searching for.
- If you can't find it with Eclipse you can try searching the Paparazzi wiki. It is a huge collection of information and almost everything is documented there. If you find something that is not yet documented: please add it yourself to the wiki and help your fellow students.
- If you still can't find a specific Paparazzi function chat with the developers of Paparazzi on gitter.im/paparazzi.
- If you have questions about the C language: check Stackoverflow for answers to your question.
- You can also ask questions on the course's forum.

Adding a module

A tutorial on how to add a module to your program can be found here: `http://wiki.paparazziuav.org/wiki/Modules`. First check that the module you want to have does not exist already. Be sure to check the create module tool that is already available to you in Paparazzi (paparazzi/create_module.py). Create a module now, add it to your airframe file, and recompile everything.

Learning how to debug

When programming your drone you could encounter that the drone does not do what you expected the first time around. In that case there are two ways to check what the values of variables on your drone are:

- Paparazzi messages
- Printing to the terminal

To learn more about paparazzi messages check this webpage: `http://wiki.paparazziuav.org/wiki/Telemetry` Create a new message now and send an arbitrary value. Check if you receive it by running the messages program in Paparazzi. As for printing to the terminal: you can use the following in your code:

```
printf("Integer variable: %d\n",myIntegerVariable);
```

To see this text appear open a terminal on your laptop and type:

```
telnet 192.168.42.1
cd data/ftp/internal_000/paparazzi
killall -9 ap.elf
./ap.elf
```

Your program is now restarted, and your message will appear in your terminal.

Making decisions onboard

As you saw last week you can leave blocks in your flight plan by adding an exception. Paparazzi allows you to use a variable of your module as a trigger for an exception. When the variable you selected is not equal to zero, the exception will occur and a next block is selected. Let's try to create the following behaviour:

- Start at waypoint one, and enter your newly created block.
- The drone will now fly towards waypoint two.
- As soon as the drone spent five seconds flying towards this waypoint (or was there for too long) we fly back to waypoint one.

Checking if the drone spent a long time in a block can be done in the block itself with the line:

```
<exception cond="LessThan(5, block_time)" deroute="Standby" />
```

However, we will create this as a function in your module.h file as such:

```
extern bool spentLongTimeInBlock(int timeInBlock){
return timeInBlock > 5;
}
```

And add it to our block:

```
<exception cond="spentLongTimeInBlock(block_time)" deroute="Standby"/>
```

Fly and verify that your program works. Now you know that you can add complex functions in your module, and call them from your flightplan. You are now also able to create exceptions yourself!

Exercise: moving waypoints

Now that we can make decisions we can also move waypoints. The goal of this exercise is to fly in the pattern of a house inside the arena. We do this by flying to a waypoint (WP_GOAL) and when we reach this waypoint we move the waypoint to a new place.

1. Start by creating a function that takes a waypoint as input (note: the waypoint variables can be found in firmwares/rotorcraft/navigation.h. Note that if you want or need to find out how the flightplan xml is converted into c-code, you can find that in var/aircrafts/bebop/generated/flight_plan.h).

2. Look at the functions that are already in Paparazzi that do something with waypoints in the file sw/airborne/firmwares/rotorcraft/navigation.h.

Now you should be able to set a waypoint in one corner of the arena(away from the net), and move this waypoint in the shape of a square or house by placing it at different points in the arena as soon as you reach them.
You can either move a waypoint relative to its old position, or you can move it relative to your current position and heading. To do the latter you should make a placeRelative function that takes a waypoint and a relative move as input and moves the waypoints taking the yaw of the drone into account. This function will surely be of much use in the next few weeks. If you choose to also change your heading while flying this pattern, (find function for it in navigation.h) then take into account that after a significant yaw angle command, drones may need some time to execute it and might move laterally in case of asymmetry or calibration issues. Always be careful not to make brutal yaw changes close to obstacles.