Developer's Guide for software from Coordination of Autonomous Aerial Vehicles, ACCESS Summer Project 2015

Erik Berglund, Paul Rousse, Benjamin Summ and Johan Sundin
at SML, KTH

September 5, 2015

Contents

1 Introduction

This document describes the different parts of the program for quadcopters developed on the Access Summer Project, 2015, and before, in the Smart Mobility Lab (SML), KTH.

The quadcopter type used, and for which this guide is aimed, is the 3DR IRIS$^+$. As a motion capture system Qualisys, `http://www.qualisys.com/`, was used. The quadcopters use the Pixhawk autopilot. Most parts of the program is written in Python.

2 Preparations

This section describes how to set up the different parts of the program on a fresh computer. Subsec. 2.1 describes the installation procedure and Subsec. 2.2 the required configuration procedure. Finally, some advice are outlined in Subsec. 2.3.

2.1 Installation

For the main part of the software that was used in the project, the Linux operating system Ubuntu is recommended, and currently version 14.04 LTS (Trusty). However, some software require a Windows installation. A short description and the installation procedure for each required program is found below. Scripts and other files from the summer project 2015 can be found at `https://github.com/XXXXXXXXXXXXXXXXXXX`.

ROS, Robot Operating System, is a framework aimed at develop software for robots. ROS (currently Indigo Igloo) can be installed, together with a simulator based on RotorS, using Gazebo and the PX4 Firmware, by following the instructions in the summer project 2015 github `README.md` file. This file also contains other information about the installation procedure. To run the simulator with several quadcopters, Docker is required. It can be installed following the instructions at `https://docs.docker.com/installation/ubuntulinux/`.

The github repository should be cloned into the `/src` folder in the ROS workspace. Then, go to the workspace top directory (e.g. `/catkin_ws`). Thereafter, source and build the workspace by first running `source devel/setup.bash` and then `catkin_make`.

To calibrate and change settings of the quadcopters, Mission Planner can be used. It runs on Windows only. Configuration of the radio can be done with the 3DR Radio Configuration Tool, also good for checking the connection strength (RSSI). It is found at `http://ardupilot.com/downloads/?did=89`.

2.2 Configuration

Calibrations of the accelerometers of the quadcopters are mandatory and can be done in Mission Planner. When performing the calibration, use a water level to setup a flat place. In lack of this, a recommendation is to use a wall. It can be checked that a calibration was well performed by going to *Flight Data→Status*. If a_x and a_y are less than 1% of $g = 9.8$ and $a_z = -g$ the calibration is satisfactory.

Special conditions apply when flying indoors. Most importantly, the GPS will stop working. How to disable the GPS and other tips for flying indoors can be found at `http://copter.ardupilot.com/wiki/common-use-cases-and-applications/indoor-flying/`. Also, be sure to set the parameter `GPS_TYPE` to `0`, so that it is disabled.

For the radio configuration, instructions can be found at `http://copter.ardupilot.com/wiki/common-optional-hardware/common-telemetry-landingpage/common-3dr-`

`radio-version-2/`. Check that parameters are exactly the same. Also, check that the signal strength is good (RSSI) with the 3DR Radio Configuration Tool. To connect the correct antenna to the correct IRIS$^+$ copy `/scenarios/launch/99-usb-serial.rules` to the `/etc/udev/rules.d/` folder. If only using the antennas in the SML this is enough (the name of the serial interface of one specific antenna is always the same). Otherwise, or to add a new antenna, see `http://hintshop.ludvig.co.nz/show/persistent-names-usb-serial-devices/` or run `udevadm info -a -n /dev/ttyUSB0 | grep 'serial' | head -n1` and add a line with the result to `99-usb-serial.rules`.

For radio calibration, use the values of the parameters specified in the document `/scenarios/launch/iris/default_param_override.param` in the git folder.

2.3 Advice

There are some general advice when working with IRIS$^+$:

- Do not do any modification after an unpacking.
- Do not open the IRIS$^+$ if it can be avoided.

3 Architecture

The program is using ROS to connect different parts. A basic knowledge of ROS is assumed. The architecture used in the program is shown in Fig. 1. Circles do not correspond to ROS nodes strictly, but describe the different parts of the program. Rectangles, however, correspond to ROS topics. ROS packages are also written in the figure. Arrows with black heads correspond to publication to a topic, if the arrow points from a circle to a topic, and subscription to a topic, if the arrow points from a topic to a circle. Names in the following paragraphs refer to Fig. 1.

The goal of the program is to process commands from the user, forwarded through the GUI, so that the quadcopter, either in the real world or in the simulator, acts as the user intends. The GUI can start different ROS launch files, in turn starting scripts that publishes points for the quadcopter to follow. These scripts are located in the `trajectory_generator` package, and for example publishes points for a line or an arc, see Sec. 7. Also, the GUI can publish specific points itself, not using a predefined script. For more information of the GUI, see Sec. 5.1. Apart from commands from the user, information about the quadcopter's location, speed, acceleration, pitch, roll and yaw is also needed. This is provided by Qualisys, taken into the ROS framework by the `mocap` package.

A controller can be applied, since both the target point and the current point are known. This is done in the `controller` package, see Sec. 10. The current point is processed by the Security Guard. If not the quadcopter is within certain safety limits, etc., the Lander is told to land the quadcopter by the Security Guard. If it is, the Blender is told to further process the data. For more information about the Security Guard, see Sec. 8. The Blender got it's name because it can "blend" outputs of different controllers and collision avoidance, see Sec. 9. The outputs of the controllers are then transformed to roll, pitch, throttle and yaw in the Blender. This is sent to Mavros on the topic /irisX/mavros/rc/override/ (X $= 1, 2, 3, 4, \ldots$) and finally to the quadcopter.

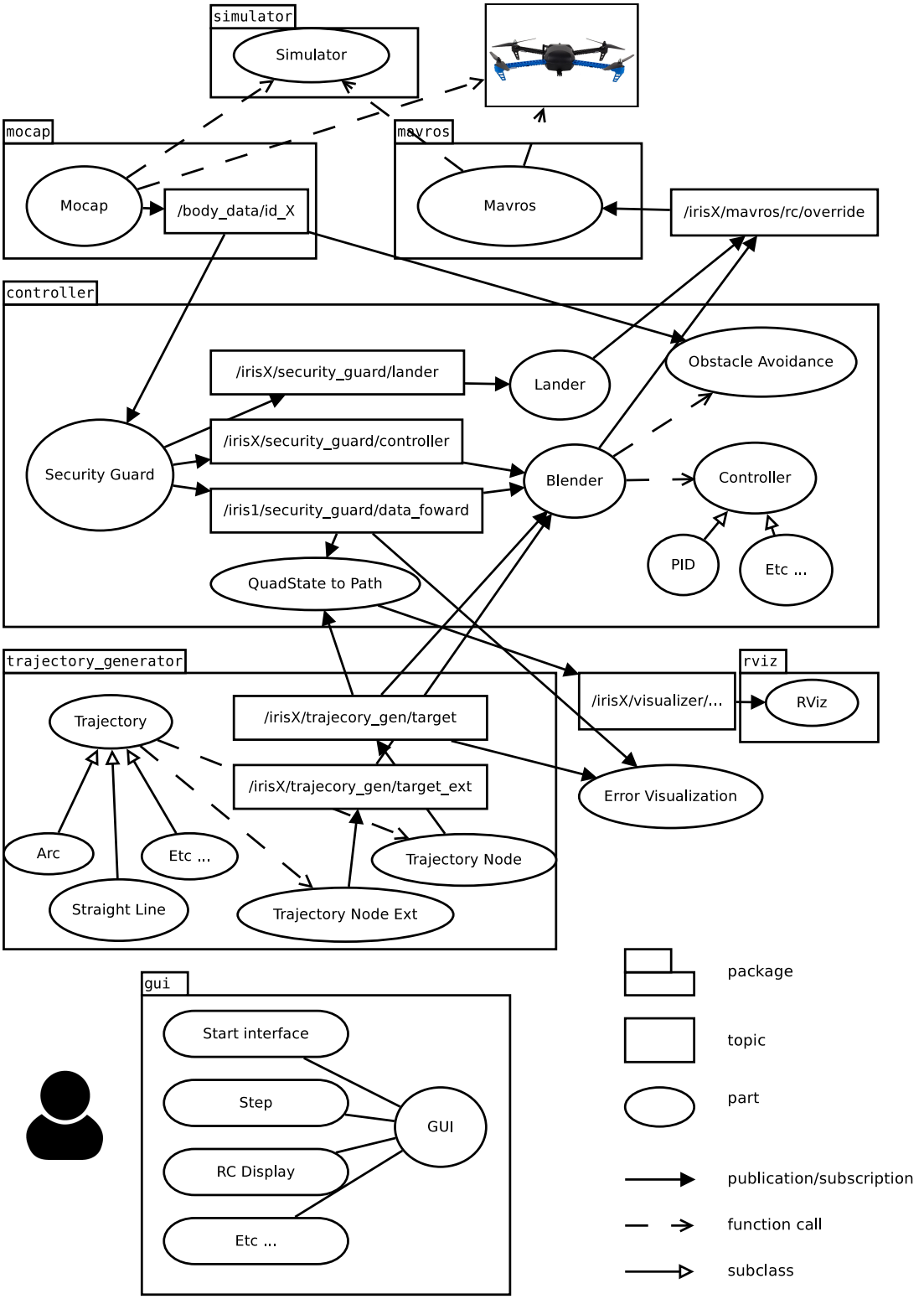

Figure 1: The architecture used in ROS. Only main parts and connections are shown. "Etc ..." indicates that other similar parts easily can be added. X = 1, 2, 3, ...

4 Documentation

To get the documentation of all the packages open a terminal and go to your catkin workspace. In the workspace go to `src/kampala` and write `sh gen_doc.sh`. The documentation will be auto-generated and when the generation is done a message is shown which tells you where the documentation is to be found. The path will be similar to `home/user/workspace_name/doc`. Here you find a folder for each package. To access the documentation of the package `package` open the folder `package` in the folder `doc` and go to `html`. Here open the file `annotated.html`, for example.

5 GUI

The GUI consists of several plugins in rqt, a framework for GUI development in ROS based on the application framework Qt. It is started with the command `roslaunch scenarios rqt.launch`. To launch the GUI properly, one needs to be in a catkin workspace with the `kampala` directory cloned from GitHub in its src directory, as the path from the directory where you launch the GUI to the files in the `kampala` is hard-coded in several of the plugins. When launched, a window with the header `Default - rqt` should appear. If the GUI hasn't been used before, the window should be empty, but you can add plugins to it by clicking on the `Plugins` tab. A dropdown menu with different folders will appear and the plugins added specifically for the quadcopters will be in the Iris folder. The most important plugin is the `Start interface`, which is needed to start the quadcopter. Its use is described in the section `How to fly`. Another important plugin is the `Tabbed GUI plugin`. It contains four tabs with one function each. They should be fairly self-explanatory but otherwise one should refer to the documentation in the code behind them, which can be found in the `gui/src/gui` directory. The files with the name `rqt iris` define the `Start Interface`, the `RCDisplay` files define the `RC and battery display`, the `pointInput` files define the `Instruction input`, the `saver` files define the `Data recorder` and the `positionPlot` files define the `Plot windows` tab.

5.1 Advice on modifying the GUI

The GUI has been created mostly by using the editor `QtCreator`. An installation guide for the editor can be found here: `https://wiki.qt.io/Install_Qt_5_on_Ubuntu`. To modify a plugin's appearance with this editor, double-click on the corresponding ui-file. With the editor, you can drag and drop widgets like buttons and text fields from a menu to add them to the plugin. Double-clicking on a widget lets you rename it. The logic in the GUI plugins is not defined in the ui file but in a python file that integrates the ui file. It is also possible to add widgets programmatically by importing them in the python file and adding them there. Refer to the file `positionPlot.py` for an example of that. At the end of the `README` file in the `gui` directory, a link to a ROS tutorial about creating rqt plugins can be found. This tutorial also links to other tutorials about integrating ui-files into python files.

The most important piece of advice when writing the code for the rqt-plugins is the following: `Do not try to modify Qt objects from a thread other than the main thread`. Qt objects are not thread safe and if you try to reach them from another thread, the GUI plugin can randomly freeze, or even worse, the whole GUI can crash. Trying to modify Qt objects in `rospy.Subscriber` callbacks counts as this, since a new thread is created for the callback function. What you can do is emit a signal from the other thread and connect it to a function that does the required modifications of the Qt

object, since this function will be called from the main thread. Do as little as possible in that function and don't call it too often, as tasks executed in the main thread slow the GUI down. The following link shows some examples of using signals in python after importing the module PySide: `https://wiki.qt.io/Signals_and_Slots_in_PySide`

6 How to fly

For experimentation using the GUI two different launch files have to be started. These are `mocap.launch` in the `mocap` package and `rqt.launch` in the `scenarios` package. To start the mocap one opens a terminal and writes `real` and then `roslaunch mocap mocap.launch`. To start the GUI one writes `real` and then `roslaunch scenarios rqt.launch`. To connect to the drone a new terminal is opened by clicking on *New Terminal* in the GUI. Then one has to click *Param* and *Connect*. *Param* has to be clicked in order to load all the parameters of the drone from the corresponding launch file. When the connection has been made one can press *Arm* and the drone should arm. Then different parts of the GUI can be used to start the drone depending on what one wants to achieve. To run the simulator the procedure is repeated, but with writing `sim` instead of `real`.

It is very important to have the correct Qualisys IDs for the quadcopters. The ID of a quadcopter can change, for example if a registered object is removed in Qualisys. Then, there are a few places in the code where this needs to be changed. For the Mocap to import the correct data, the parameter `body_array` needs to be changed in `mocap.launch`. (Note that there are two versions of this parameter – one for `real` and one for `sim`. Only the one for `real` needs to be changed.) The ID also needs to be updated in the file `irisX.launch` with the corresponding X (X $= 1, 2, 3, 4, \ldots$) in the `scenarios` package. The concerned parameters are `body_id` and `my_id` (yet again one version for `real` and one for `sim`). This is done so that each iris is connected to the correct ID.

6.1 Advice

When arming the drone one sometimes gets the error message: `Throttle below FS`. This tells you that the throttle is too low, but usually it is not. One should try to kill the Mavros node, reconnect and arm again. This almost always does the trick.

If the drone lands because of a problem in the program, the next time it is armed it might start to go up straight away without having any controller running. It seems to be possible to avoid this by simply unplugging the battery and plugging it back in.

If the landing button fails: DO NOT KILL MOCAP. The mechanism to land via the landing button and mocap is the same and if the landing button fails it is probable that the whole mechanism has failed. Killing mocap will only make the drone behave more violently. Instead of killing mocap disarm the drone manually by holding down the safety button on the drone. Do not push it again while holding the drone as it will arm. Shut down and restart all ROS nodes. Generally, if weird behavior is observed: shut down all the ROS nodes and restart them to see if the weird behavior stops.

There are some error messages given by Mavros that can be disregarded. These are `DCM bad heading` and `FCU variance`. The `DCM bad heading` will make the lamp at the back of the drone blink yellow and red. This can be disregarded. However, if there is some strange behavior one should check the status of the drone on Mission Planner.

An unlikely but possible problem is that the combination of markers on a drone is not unique, i.e. there is another object in the lab that is confused with the drone by

mocap. This is unlikely, but it has happened.

The performance of the drones seems to be quite sensitive to the battery level, especially in the z-direction. This has led to a need of retuning the gravity cancelling constant for different battery levels. It is rather annoying but necessary. If one manages to get convergence in the z-direction the integral part stabilizes the drone in a certain interval of battery levels.

There has been strange behaviour that has remained unexplained. It is probable that this behaviour emerged from different people messing with the same code and failing to merge it properly. It is advised to never work on the same code at the same time as to avoid such problems.

There has been a need for recalibrating the accelerometers a few times. It is good to be able to do this calibration rather quickly.

Using the GUI in connection with the launch files enables you to change the parameters given in the launch file while the drone is in the air. This is very useful for the tuning of parameters, especially in connection with the tracking visualizations that are started when arming the drone. To change the parameters while flying, change them in the launch file, save it and press *Param* in the GUI. If new parameters are added and you want to use this feature for those parameters you will have to change the functions that update the parameters of the blender, PID, obstacle avoidance, etc. If you want to use this feature for a controller of your own you can add a ROS service to your controller that calls a function that updates the parameters. To get this function connected with the GUI you have to add it in the function called Param() in the script `rqt_iris.py` in `gui/src/gui`. As this is used in the Blender and the PID these can be used as examples.

It is advised that somebody operates the GUI at all times in order to make the drone land in emergency situations.

7 Trajectory generation and trajectory following

In the `trajectory_generator` package there are different scripts that are meant to be used together with a controller in the `controller` package (mainly the PID controller). These scripts generate a reference for the controller that respects the controller's interface. This means that not only target positions are generated, but also target velocities and accelerations. A controller uses these together with the data of the motion capture system to compute the control output. The interface of the trajectory generator scripts is defined by the abstract class called `Trajectory`. All trajectory generators are subclasses of this class and use its interface.

There are some classes with class names that end with `_ext`. These are meant to be used in connection with controllers that also need a jerk (third derivative of position) and snap (fourth derivative of position) reference, for example the load lifting controller (that at the moment of writing unfortunately is not working).

To calculate the references certain continuous time laws are used. These depend, of course, on the trajectory to be performed. The time laws are then discretized using the publishing frequency of the trajectory node. Each trajectory uses a trajectory node to publish its data and only one such node should be used within one script. This assures that the same node is publishing the points all the time and hence there is no risk of interference between different nodes.

The target points are published on the topic `/irisX/trajectory_gen/target` (X = $1, 2, 3, 4, \ldots$). The blender subscribes to this topic and passes the necessary information to the PID.

A useful class in the package `trajectory_generator` is the `TrajectoryGenerator` class. It contains a collection of different functions that have turned out to be useful when writing code for trajectory generation. One nice feature is a function that, given position, velocity and yaw, generates the corresponding ROS message.

There are also different classes used for leader following. Here, there also is an abstract base class that can be used for different ways of leader following.

7.1 Obstacle avoidance

In the launch file of each drone there is a parameter called `obstacle_avoidance`. This parameter can be set to true or false depending on whether obstacles should be avoided or not. The obstacles to be avoided are specified through the parameter `OBSTACLES_TO_AVOID`. Observe that an obstacle that is not registered with the motion capture system or outside the area of detection will be ignored.

The avoidance itself is done through a potential between the drone and the obstacle, pushing the drone away from the obstacle. Here the z-direction is ignored, meaning that the drone is pushed away radially outward from an infinitely long cylinder whose axis is along the z-direction in the SML-frame. This is done mainly because two drones should repel each other even when they are at different heights. Obviously, it would be of use to improve this algorithm as one might want the drones to avoid obstacles that are in some finite interval along the z-axis.

8 Security guard and lander

The Security Guard is used for several security features during experiments. It gives permission to publish on the `rc/override` topic either to the Blender or to the Lander. Permission is granted to the Lander if the motion capture signal is lost for more than 0.5 seconds. The Lander sets the flight mode of the quad to landing mode and publishes neutral commands on the `rc/override` topic. When the Lander is used, the landing is not controlled since there is no feedback. Therefore, the drone keeps moving in the xy-plane during landing. This should be fixed sooner or later. It was looked into using internal measurements to get feedback for the landing and there are some Mavros topics that provide these measurements, but with the firmware used on iris 3 and iris 4 these topics don't give any information. However, the firmware on iris 2 can access the information. Observe that this is a beta firmware and not the firmware pre installed on the iris and it was chosen to not change the firmware on iris 3 and iris 4 to avoid trouble. If the drone is outside of the safety area defined in the launch file `iris_nodes.launch` or the landing button is pushed, the landing is controlled with the PID, still using the motion capture data.

The important topics that manage the landing are `irisX/security_guard/controller` and `irisX/security_guard/lander`. As long as everything works and no landing is desired a zero is published on `irisX/security_guard/controller` and false is published on `irisX/security_guard/lander`. When the landing button is pushed a 2 is published on `irisX/security_guard/controller` and false is published on `irisX/security_guard/lander`. The drone goes to a height of 0.5m and then the landing mode is set. If the mocap signal is lost a 1 is published on `irisX/security_guard/controller` and true is published on `irisX/security_guard/lander`. In this case the controller is turned of and the flight mode is set to landing mode.

Since here potentially different nodes are publishing on the topic `rc/override` it is of importance that only one node publishes at a time. Two nodes publishing on this

topic can cause conflicting outputs and therefore strange behaviour during landing.

9 Blender

The Blender is found in the `controller` package. It is used for two things. The first is taking in acceleration outputs from different controllers and blending these according to some scheme. The second is to calculate the control outputs after blending the accelerations and then publish these on the topic `irisX/mavros/rc/override` in order to control the drone.

As of now, the Blender takes an acceleration from a controller and one from the obstacle avoidance and performs a convex combination of these. The constant with which the acceleration from the obstacle avoidance is weighted is dependent on the distance to the obstacle. The Blender uses a method of the obstacle avoidance to get this constant. Of course, if obstacle avoidance is turned off the output is only calculated from the output of the controller.

Currently, there is a problem with this kind of blending of accelerations. The problem is that convergence to a goal point cannot be guaranteed in this way. In fact it can be easily thought of an example where convergence does not occur. Assume that the quadcopter is hovering at a certain point, which is its target, and another quadcopter is approaching. Furthermore, assume that the first quadcopter tries to avoid the second one, but the second one does not try to avoid the first. Then the first one will move away from its target and hover at a different point. Thus it is clear that a more sophisticated controller is necessary in order to do precision based tasks involving collision/obstacle avoidance.

10 Controllers

Mainly the PID controller, Sec. 10.1, has been used in this project. However, a new controller can easily be added by modifying the Blender, Sec. 9, and using the interface provided by the abstract base class `Controller`.

10.1 PID controller

The PID controller in the `controller` package is not just a normal PID controller. The proportional and derivative gains are coupled in a certain way. It is therefore not possible to change these gains independently by using the parameters given in the launch files. If they have to be changed independently one can either change them directly in the script of the PID without using the coupling between them or one can change the damping constant named `x_i` in the script of the controller. It has not really been necessary to do this, however. It should be sufficient to change the parameters `CONTROL_CANCEL_GRAVITY`, `PID_w`, `PID_w_z`, `PID_I_lim`, `PID_ K_i`, `PID_I_lim_z` and `PID_K_i_z`. The first parameter is crucial for performance in the z-direction as it is used to counterbalance gravity. It has been observed that this parameter might have to be retuned for different battery levels. The next two parameters are used to adjust both the proportional and the derivative gain in an optimal way with respect to each other. The `PID_w` is for x and y and `PID_w_` z is for z. The `I_lim` and `K_i` are the saturation and gain of the integral term. The ones with `_z` in the end are for the z-direction, the other ones for x and y. There is no need for a high saturation limit in the x and y direction but in the z direction it should be rather high to guarantee robustness against the change in battery level. The parameters that currently are in the launch files work

fine for iris2, iris3 and iris4. Observe that the value of the parameters `Kphi` and `Ktt` should be 637 for both.

11 Connecting servos

IRIS$^+$ uses the Pixhawk autopilot. On the Pixhawk there are outputs for servos, shown in Fig. 2. The outputs are divided in eight "main outputs", MAIN OUT 1-8, and six "auxiliary outputs", AUX OUT 1-6. MAIN OUT 1-4 are occupied by the the four motors. However, MAIN OUT 1-8 should be avoided for servos anyway since these update at a rate of 400 Hz by default. AUX OUT 1-6 update at 50 Hz – standard for servos.

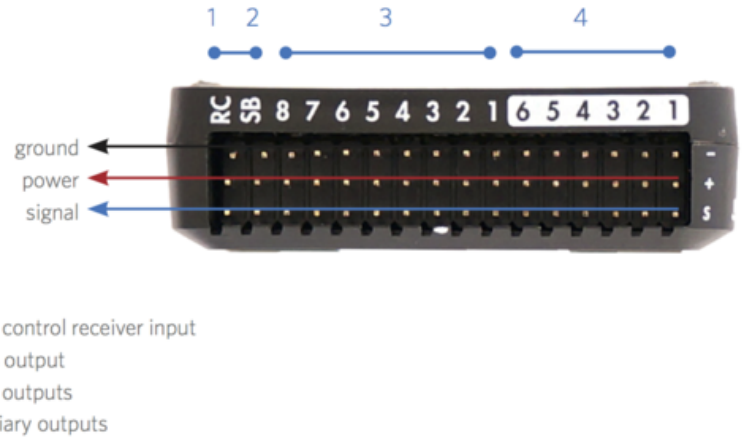

Figure 2: Pixhawk outputs for servos. Image from `http://copter.ardupilot.com/wiki/common-autopilots/common-pixhawk-overview/`.

AUX OUT 5-6 are, by default, set up as relays. The number of AUX OUT ports set up as servo outputs can be changed in Mission Planner with the `BRD_PWM_COUNT` parameter, in the way that setting `BRD_PWM_COUNT` to 6 gives servo output on AUX OUT 1-6.

The IRIS$^+$ transmitter and receiver has eight channels. This is reflected in the way that a vector of eight components can be sent to the quad via the topic `/irisX/mavros/rc/override/`. The first four components are for roll, pitch, throttle and yaw, leaving the other four for servos. In addition to that, the different channels need to be linked to the different servo outputs on the Pixhawk. This can be done in the Mission Planner. In Mission Planner, the eight channels are named RC1-RC8. The different servo ports are named in a similar way: RC1-RC8 corresponds to MAIN OUT 1-8 and RC9-RC14 corresponds to AUX OUT 1-6. Unfortunately, in the current version of Mission Planner, there is no general way of connecting channel RCX to RCY, X $= 1 \ldots 8$, Y $= 1 \ldots 14$. If three or less servos are needed, the built in settings for a camera gimbal can be used.[1]

The Pixhawk does not provide power to the servos itself, so these must be powered in other ways. A BEC or ESC, providing 5 V, can be used, for example. It can be connected to the servo outputs on the Pixhawk or to a servo directly. By default, the

[1] Which signals to be sent to which servo output are controlled by the parameter `RCX_FUNCTION`, X $= 1 \ldots 14$, in Mission Planner. For these parameters, there is an option called `Passthrough`, passing channel X to output X, but since there are eight channels, only MAIN OUT 1-8 can be reached by this option, not AUX OUT 1-6.

ground pins are connected to the ground of the battery, by a wire to MAIN OUT 1 ground pin. Also, on the bottom of the IRIS$^+$ there are a black cable connected to AUX OUT 6 ground pin and a white cable connected to AUX OUT 1 signal pin. In addition to this, there is a red power cable and another black ground wire connected to the battery of the IRIS$^+$, providing 12 V. If the voltage is lowered to around 5 V this can be used. So, if only one servo is needed, the quadcopter does not even needs to be opened!

For additional tips and tricks, see `http://copter.ardupilot.com/wiki/common-optional-hardware/common-servo/` and `https://learn.adafruit.com/quadcopter-spray-can-mod/`. A wiring diagram for the Pixhawk can be found in `http://copter.ardupilot.com/wiki/advanced-pixhawk-quadcopter-wiring-chart/`.

APPENDIX

A "Fix my IRIS" procedure

If the IRIS$^+$ starts to behave strange, a standard procedure is outlined below, fixing many of the possible errors.

- Put a fully charged battery in the drone.
- Is the drone unstable? In that case, try to control it with the RC transmitter.
  - In case of good behavior, check the controller.
    - Is hovering term too high?
    - Is the gains too high?
  - If drone fails to arm, take a look at messages in Mission Planner.
  - If propellers have different speeds, do a ESC Calibration.
  - If not stable with RC transmitter:
    - Redo accelerometer calibration.
    - Redo compass calibration.
  - If the range of the RC transceiver is low:
    - Redo RC transmitter calibration.
    - Change the battery of the drone.
    - Change the RC transmitter batteries.

  Otherwise, redo calibration in Mission Planner and be careful that there is no offset in the RC command (neutral position must be equal to 1500).
- Does Mavros fail to connect?
  - In case of green or blue light blinking, check 3DR radio configuration.
  - Check USB connection.
  - Check configuration of the launching file (e.g. `iris1.launch`).
  - Don't get parameters?
    - Reboot Mavros or reboot the drone.
    - Lower the air speed parameter of the 3DR radio.
    - As an alternative, take the time and wait for all the parameters.
  - In Mission Planner, check compass health and read `http://ardupilot.com/forum/viewtopic.php?f=48&t=10478` (don't move the quad and don't close the battery slot during gyro initialisation).
- ULTIMATE FIX: Change the drone! (Drones are complex systems – it might be difficult to find the error.)

B 3DR Radio communication interference

It seems that the radios interfere when they are close, so try to have at least 1 meter between each of them. However, it is not sufficient; if you look at the `/diagnostics` topic, then you will see that there are many Rx errors. It is the same in Mission Planner: by looking at the logs, we see that many errors append. One of the consequence that is directly visible is that that parameters are much slower to be loaded at the initialisation of Mavros.

A quick fix that has been tested is just decreasing the timeout variable in the `mavros` package (file in `mavros/src/plugins/param.cpp`, `PARAM_TIMEOUT_MS` has been changed to 100 ms instead of 1000 ms and `RETRIES_COUNT` has been increased to 5). In this way the initialisation is faster, however it does not solve this Rx trouble, and it might have some consequences on the controller if too many errors happens. Try to reduce the numbers of data that are transmitted by using the "rosrun mavros mavsys rate" command.

A good fix would be to find a firmware were the ECC can correct more that 25% of errors through the communication protocol. However, this is not possible without getting inside the firmware code. I (Paul) did not managed to make the "LBT Rssi" work (listen before talk); it could help to reduce the number of Rx errors. A good parameter to tune is the Protocol, setting up to Raw Data might increase the rate at which we can transmit data through `rc/override`. However, as long as I am not sure that every thing work well, I will leave it on Mavlink. The Tx power does not have to be extremely high, the 3DR radio is made so that it can transmit data up to 500 m, since we are about 10 m of the quad, it is okay to set it to 5.

Some people have been facing similar problems. However, none of the fixes worked (trying with the 1.7 firmware for the 3DR radio, setting different `SYSID_THISMAV`, separate every frequencies for each coupled radios). Some links are
`http://copter.ardupilot.com/wiki/common-optional-hardware/common-telemetry-landingpage/common-3dr-radio-version-2/`
`http://ardupilot.com/forum/viewtopic.php?f=22&t=8488`
`http://www.rcgroups.com/forums/showthread.php?t=2077354`
`http://ardupilot.com/forum/viewtopic.php?t=10000&p=24499`
`http://diydrones.com/forum/topics/question-on-using-multiple-3dr-radios-to-control-multiple-drones`

Documentation can be found at `http://copter.ardupilot.com/wiki/common-optional-hardware/common-telemetry-landingpage/common-3dr-radio-advanced-configuration-and-technical-information/#Upgrading-radio-firmware`

To get any version of the Firmware, clone `https://github.com/Dronecode/SiK` and get back to another commit (the one of the version you want), install sdcc with `apt-get` and do a `make install` in the Firmware directory. The hex file (equal to the firmware) is the `Firmware/obj/hm_trp/radio~hm_trp/radio~hm_trp.ihx`.