Postprocessing UAV Footage

This documents provides a basic workflow of the georeferencing process of Solo/Pixhawk flight data as well as the optimized workflow for deriving high quality point clouds 3D models and orthoimages. In detail it covers the topics:

- data acquisition constraints
  1. geotagging of the images
  2. high quality image alignment
  3. high quality point cloud generation
  4. high quality orthoimages generation
  5. high quality dense point clouds
     - importing and exporting data from photoscan
  6. Agisoft Photoscan scripting
  7. interaction with the uavRst package

  This is a preliminary draft. Please note that significant changes can still be expected

Constraints of data acquisition

The base for good micro-remote sensing products are the images takes by the drone. They will pass through a post processing workflow where the original picture quality determines the quality in every further step. It is pretty simple and successful if one keep in mind some very basic constraints during data acquisition.

Actually you should take care of two requirements:

- sufficient overlap, provide photos where at least 70% of the photo is on focus while avoiding scale leaps (surface following flight paths). Consider on which footprint the overlap is calculated. If you have different structure layers in different distances to the camera, the overlap differs for every of this layers as they have all their own footprint dimensions.
- diffuse but bright light, best is at noon with kind of high thin cirrus clouds scattering the light perfectly and reduce shadows….

Prerequisites

The following workflow requires some tools: You need themapillary_toolswhich is a very useful bunch of python scripts dealing with image placement on web maps. To run them you also need to install `gpxpy`, `pillow`, `exifread`,`pyexiv2` . If running windows please install first a pip installer derivative, running Linux you can install pip with `sudo apt-get install pip`. Then you may install the dependencies like following:

```
pip install
git+[[https://github.com/mapillary/mapillary_tools.git|https://github.com/ma
pillary/mapillary_tools.git]]
```

```
pip install gpxpy

pip install exifread

pip install Pillow

sudo apt-get install python-pyexiv2 (Linux)
```

For Windows you will find at launchpad pyexif2.

Next (if not already installed) you need to install the gpsbabel binaries.

```
sudo apt-get install gpsbabel\\
```

For the visualization of the gpx data you may use QGIS or mapview and plotRGB in R.

Adding geotags to the images

Almost no standalone camera has a *GPS* or an *aGPS* onboard. As a result no image will have a spatial reference at all. Georeferenced images can be aligned better and faster and, as a side effect, one will have a fairly good georeference (~1-5 meters). So even if this task is a bit cumbersome it will help a lot.

The most prominent way to apply a spatial position is to add geotags to the images. This needs a semi-automated post processing. The best software doing so is geosetter, which is unfortunately not running very stable under Linux and OSX (besides some other shortcomings).

Actually, while adding geotags you have to deal with three effects:

- time shift between logger and camera time
- automatic timezone adaption of the data
- identification of a start and stop image within the time series

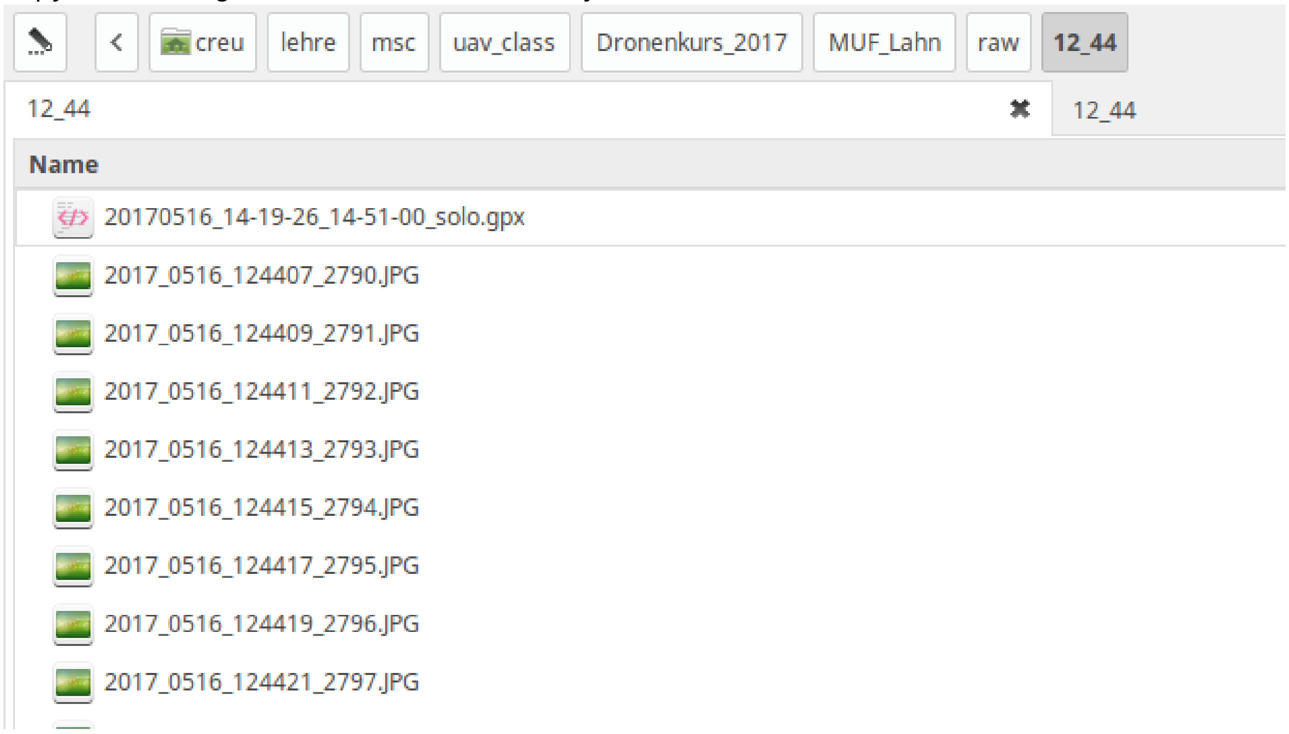
For our needs we want to align the images along the flight track within a accuracy of 2-5 seconds (according to lapse rate). It seems to be kind of senseless to automate this process due to the fact that you always have to start and stop the cameras manually and you always will take pictures on the ascending, descending going to and coming back from the core task.

Note: You have to find out when the timestamp of the image is created. Shooting the picture or saving the file (what might be a few seconds later, e.g. for Mapir-cameras the timestamp is created in that moment the file is saved).

Preparation of the data

If not done you need to organize some basic things. It is more than helpful to separate each partial flight an the corresponding gpx file in a subdirectory. Very straightforward it may look like below.

- separate all image files according to the corresponding flight (log file)
- copy the GPX-log file into the same directory



creu lehre msc uav_class Dronenkurs_2017 MUF_Lahn raw 12_44

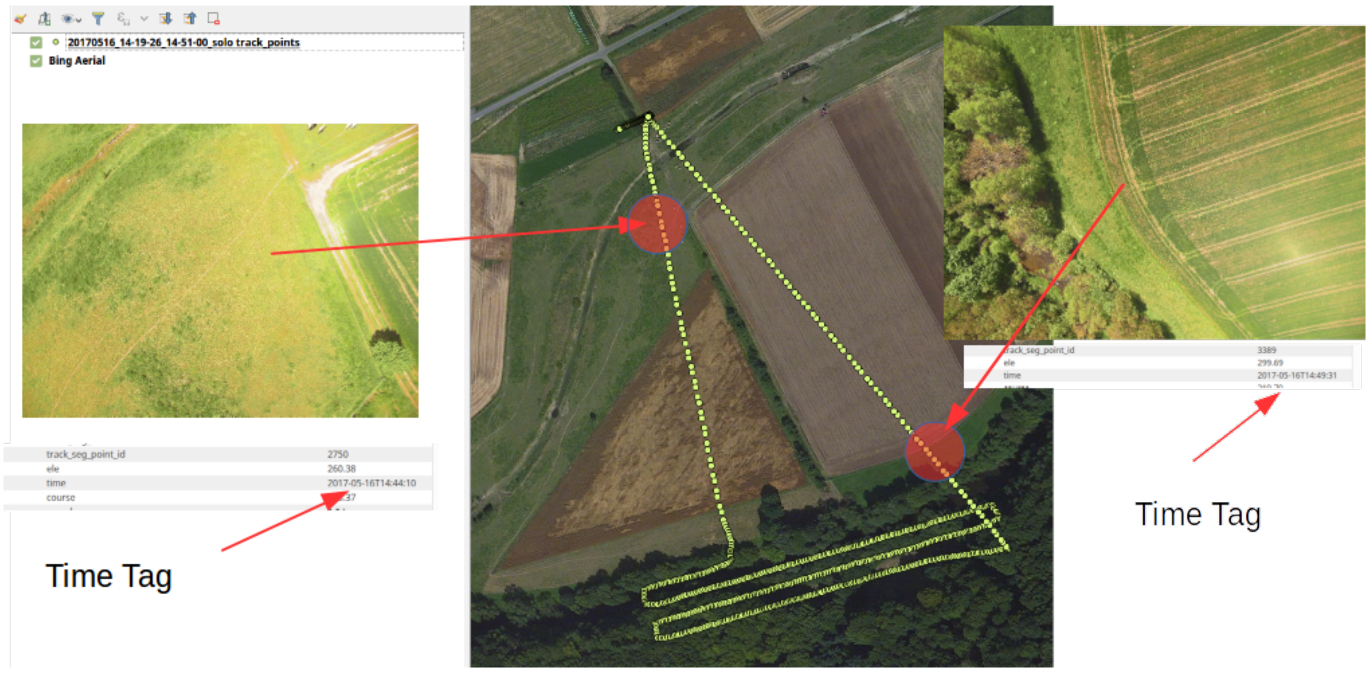
12_44

12_44

Name
20170516_14-19-26_14-51-00_solo.gpx
2017_0516_124407_2790.JPG
2017_0516_124409_2791.JPG
2017_0516_124411_2792.JPG
2017_0516_124413_2793.JPG
2017_0516_124415_2794.JPG
2017_0516_124417_2795.JPG
2017_0516_124419_2796.JPG
2017_0516_124421_2797.JPG

The basic concept

You have to find the start and stop image of the partial task. Load the task in QGIS open the Bing aerial map and the first and last image. In most cases you can easily identify the rough position of the camera. It is by no means necessary to get an extremely exact result. You just need to to generate a correct sequence according to the gpx track points as described below.



track_seg_point_id	2750
ele	260.38
time	2017-05-16T14:44:10
course	[illegible].37

track_seg_point_id	3389
ele	299.69
time	2017-05-16T14:49:31

The workflow

Open a terminal and navigate to the image/gpx folder.

Write fake exif date and time tag

```
python~/dev/R/uavRst/inst/python/add_fix_dates.py.'2017-05-16 16:44:12'**2**
```

Please note:

1. the location of the scripts is arbitrary

```
 - you start with the best fitting track point according to your first
image
 - the **dot** is obligatory to point to the actual directory
 - the 2 is the time increment of the cameras lapse rate in seconds
 - take the time of the GPX-time-tag (red 16)real GPS time is 14
```

</note>

==== Cut the gpx log file ====

```
gpsbabel-t -i gpx -f current_solo.gpxtrack,
start=20170516144408,stop=20170516144930-o gpx-F clean_task.gpx
```

Please note:

1. gpsbabel may re-interpret the time tag to GPS time so usually you have to set the time shifted by the time zone and summertime
2. just add for convenience two lapse rate steps in the beginning and end Write the derived geotags to each image file.

```
python ~/dev/R/uavRst/inst/python/geotag_from_gpx.py.clean_task.gpx
```

You are done with the preparation now. Time to start Photoscan.