Contents

Chapter 1

design

1.1 Plan

Below is the plan for storing the ENRAM workflow (i.e. the whole process from reading the radar data to analysis and visualization of the data):

- get all the C code and the IDL code, store it in a new repository 'enram' at `https://github.com`[1]
  - also store a selection of the data, e.g. one sample from every station, for unit testing

    Q: on what machine are the tests run? Whose license is IDL using then?
- set up a virtual machine ('ENRAMVM') containing a Linux operating system

  Q: what version of Linux? Ubuntu/Lubuntu? does it matter?
- copy the data to inside the virtual machine
- within the virtual machine, install IDL
  - get an idl license
- within the virtual machine, install hdfview, plus any other software that you need
- set the necessary paths
- check out the code from the enram repository using `git clone`

[1] We prefer a public project with a license like the ones we already have in place for existing projects. Check that this is OK with Martin, Hidde, Adriaan.

- save the virtual machine on the external hard drive
- also store the virtualbox installer on the external harddrive

1.2 End user steps to reproduce the ENRAM workflow

These are the steps that Willem needs to take in order to reproduce the ENRAM workflow.

- Plug in the external USB harddrive
- In Windows, open a file explorer, navigate to the USB drive, click on the virtualbox installer.
- In Windows, start virtualbox. Load/open the virtual machine 'ENRAMVM' located on the external haddrive.
- Log in if needed (can I skip logging in/maybe use a guest session or something?)
- Open a terminal
- `cd` to the directory that holds the local copy of the github/enram repository.
- In the terminal, start IDL
- At the IDL prompt, run `.r bird_call.pro` or any other program that you want.

1.3 Drawbacks

What can I not do with this setup? Are there any other drawbacks?

- external harddrive

  Q: (would it make sense to make a harddisk image that is then stored somewhere on, say, gridstorage?)

  A: yes

- unit testing/integration testing not allowed under the license...not possible?

Chapter 2

notes

package hdfview on linux ubuntu

- describe the METEOR 360AC (IEEE) C-Band setup/what the radar actually measures
- what the raw data look like
- what the converted data look like
- read the data
- what is a cluttermap
- how cluttermaps are calculated
- PPI = http://en.wikipedia.org/wiki/Plan_position_indicator
- http://en.wikipedia.org/wiki/C_band

Selex SI Gematronik rainbow

2.1 History

1. Adriaan's vol2bird C program
2. Martin's IDL code plus C-library

2.2 Overview of files and folders

```
dir1='/home/daisycutter/enram/flysafe2-readonly/'
```

When I navigate to `$dir1`, I see these dirs:

1. `idl/`
2. `idl/clutter/`
3. `idl/io/`
4. `idl/vol2bird/`
5. `process/`
6. `process/log/`
7. `usr/people/graafdem/IDL/flysafe/*`
8. `visualisation/map/`
9. `visualisation/map/loop_gif/`
10. `visualisation/profile/`
11. `visualisation/profile/all_gif/`
12. `visualisation/profile/test_gif/`

Below is a more detailed description of what each directory contains:

1. `idl`
   - contains 3 IDL scripts: 'common_definition.pro', 'radar_definitions.pro', and 'radar_names.pro'.
   - 'common_definitions.pro' defines a number of global (`COMMON`) constants like the maximum number of elevation scans, the radius of the Earth[1], a conversion factor to get from Z [2] to cm^2/km^3 at C-band [3] [4,5]
   - 'radar_definitions.pro' contains 5 hardcoded, absolute paths[6,7,8]:
     (a) `RAW_DATA_PATH` is the full path to the raw data, as delivered by the various radar operatives. Its current value is `/usr/people/graafdem/FLYSAFE/process/data/raw/`, which does not exist on my system or the virtual machine.

[1] `RADIUS43` is not used anywhere though...?
[2] reflectivity power?
[3] Q:there are a couple of integer variables that have an L after their value, e.g. `NLAYER=30L;`, according to http://www.exelisvis.com/docs/IDL_Data_Types.html this means long integer in IDL. Maybe the programmer adopted a 'better safe than sorry' approach? (Normal integers are 16 bit, i.e.-32768 to +32767). A: has to do with consistent size of integers between C and IDL
[4] I should check 'common_definitions.pro' again sometime when I have a better idea of what each variable is.
[5] Also, the COMMON block in 'idl/vol2bird/bird_call.pro' is not exactly equal to that found in 'idl/common_definition.pro'. Is it necessary to have both versions? A: may have something to do with different versions of Adriaan's software, check his email communication with Martin.
[6] maybe check out http://www.physics.wisc.edu/~craigm/idl/archive/msg06372.html
[7] Q: does IDL need hardcoded paths or could it use relative paths? A: No.
[8] Q: am I free to re-organize the directory structure? A: Yes.

(b) `IO_PATH` is the full path to the IDL io definitions. Its current value is `/usr/people/graafdem/FLYSAFE/idl/io/`, which does not exist on my system or the virtual machine.

(c) `INPUT_DATA_PATH` is the full path to the harmonized, ODIM format data. Its current value is `/usr/people/graafdem/FLYSAFE/process/data/odim/`, which does not exist on my system or the virtual machine.

(d) `CLUTTER_DATA_PATH` is the full path to the cluttermap files. Its current value is `/usr/people/graafdem/FLYSAFE/process/data/cluttermaps/`, which does not exist on my system or the virtual machine.

(e) `BIRD_DATA_PATH` is the full path to the output data directory. Its current value is `/usr/people/graafdem/FLYSAFE/process/data/bird/`, which does not exist on my system or the virtual machine.

In brief this is how I think the components from these directories work together:

(a) The starting point is of course the data in `RAW_DATA_PATH`;

(b) The data are stored in various formats and styles, so in order to access them, one needs to know how to read each specific format and style, which is defined in the algorithms in `IO_PATH`. After the data have been 'harmonized', i.e. converted to a uniform format (ODIM/HDF5), they are saved in `INPUT_DATA_PATH`;

(c) Clutter (unwanted return signals from buildings, vegetation, as well as foreign signals that fall within the radar's receiver wavelength) needs to be removed from the data with the use of so called cluttermaps (i.e. masks). Before that can be done, the cluttermaps need to be calculated, essentially by picking a time interval during which all return signals can be attributed to clutter. The resulting maps are stored in directory `CLUTTER_DATA_PATH`;

(d) By using the cluttermaps from `CLUTTER_DATA_PATH` and the harmonized data from `INPUT_DATA_PATH` <some algorithm[1]> is used to calculate how many birds are airborne in the vicinity of each radar.

- 'radar_names.pro' contains a function that returns and optionally prints an alphabetical list of radar station codes when given either the full name of a country, the country code, a list of radar station codes, or an arbitrary combination.

2. `idl/clutter`

   contains amongst other things, an IDL script 'statistics.pro' that is able to accumulate reflectivity power over time (useful for calculating cluttermaps). Further contains a couple more cluttermap related scripts, as well as a test script.

3. `idl/io`

   contains the input/output routines for the different radar systems, per country (i.e. it is assumed that all systems from one country can be handled equally).

4. `idl/vol2bird`

[1] probably `idl/vol2bird`?

- contains 'bird_call_ori.pro' ('ori' suggests 'original', not sure in what sense) and 'bird_call.pro', these are almost equal, different RANGMAXSTDEV, DBZFACTOR, and SIGMABIRD[1] as well as a couple more small changes.
- also contains 'vol2bird_routines_ori.c' ('ori' suggests 'original', not sure in what sense) and 'vol2bird_routines.c'. These are equal.

5. `process`
   - contains 'flysafe2.pro', and 'flysafe2_day.pro', pretty much the same functionality. 'flysafe2_day.pro' is initialized to run for one day only (15-Aug-2011). Both scripts extract bird density profiles from European weather radar data.
   - contains 'histogram.pro', an algorithm that is able to iterate over a time range, collect the relevant files, and accumulate the reflectivies in support of determining cluttermaps.
6. `process/log`

   contains the standard out and error from running the algorithms in `process`
7. `usr/people/graafdem/IDL/flysafe/`

   this directory contains a couple of IDL libraries. It looks like these libraries are integral versions of their respective packages, I think most scripts/functions remain unused.
8. `visualisation/map`

   contains
9. `visualisation/map/loop_gif`

   contains
10. `visualisation/profile`

    contains
11. `visualisation/profile/all_gif`

    contains
12. `visualisation/profile/test_gif`

    contains

```
dir2='/media/daisycutter/49f14219-4570-4c14-87b3-d7e502ed3736/users/
graafdem/FLYSAFE2/'
```

When I plug in the external USB drive, these 12 directories currently exist at `$dir2`:

[1] Q:same as with the common blocks?

1. bird/
2. cluttermaps/
3. knmi/
4. odim/
5. rainbow/
6. raw/
7. sens_clutter/
8. software/
9. statistics/
10. web/
11. zero_bird/

Here's a comparison between the contents of two dirs based on a recursive diff:

1. $dir1/idl/ and $dir2/software/idl are functionally equal. Differences are whitespace characters.
2. $dir1/process/ and $dir2/software/process
3. $dir1/visualisation/ and $dir2/software/visualisation: $dir1 has a visualisation/profile/multi_prof_test.pro and a profile/test_gif

2.3 Proposed directory structure

1. data
2. data/raw
3. data/harmonized
4. data/harmonized/cluttermaps
5. data/harmonized/odim
6. data/harmonized/bird
7. src
8. src/clutter
9. src/io
10. src/vol2bird

2.4 About the C-routine

Adriaan Dokter made a C program that calculates bird volumes[1] based on radar data[2]. Martin de Graaf has converted this C-program into a library that can be called from other tools such as Fortran, MATLAB, R, Java, Python and so on. The C-library is located at `enram/src/vol2bird/vol2bird_routines.c`. The routines can no longer be used as a standalone program, but rather must be called from IDL.

In the C-library, 2 structures are important: `scanmeta` and `cellprop`. `scanmeta` contains metadata like the time at which the scan was made, the pulse length that was used, etc.[3]. It seems `cellprop` contains a couple of 2-D (?) arrays that hold the actual data[4].

```
struct scanmeta {
    int date;                /*Date of scan data in YYYYMMDD.*/
    int time;                /*Time of scan data in HHMMSS.*/
    float heig;              /*Height of radar antenna in km.*/
    float elev;              /*Elevation of scan in deg.*/
    int nrang;               /*Number of range bins in scan.*/
    int nazim;               /*Number of azimuth rays in scan.*/
    float rscale;            /*Size of range bins in scan in km.*/
    float ascale;            /*Size of azimuth steps in scan in deg.*/
    int azim0;               /*Ray number with which radar scan started.*/
    float zoffset;           /*Offset value of quantity contained by scan.*/
    float zscale;            /*Scale of value of quantity contained by scan.*/
    int missing;             /*Missing value of quantity contained by scan.*/
    float PRFh;              /*High PRF used for scan in Hz.*/
    float PRFl;              /*Low PRF used for scan in Hz.*/
    float pulse;             /*Pulse length in microsec.*/
    float radcnst;           /*Radar constant in dB.*/
    float txnom;             /*Nominal maximum TX power in kW.*/
    float antvel;            /*Antenna velocity in deg/s.*/
};

struct cellprop {
    int imax;
    int jmax;
    float dbz;
    float tex;
    float cv;
    float area;
```

[1] densities?
[2] reflectivity?
[3] but it does not seem to hold any information with which the radar can be uniquely identified—this is probably taken care of in the IDL code that uses the C-library.
[4] I'm currently not sure what every variable means.

```
    float clutterarea;
    float max;
    int index;
    char drop;
};

typedef struct scanmeta SCANMETA;
typedef struct cellprop CELLPROP;
```

2.4.1 Overview of the functions that make up the C-routine

Below is an overview of the functions that reside in enram/src/vol2bird/vol2bird_routines.c:

1. functions that have no body:
   - `read_h5_scan`
   - `string2datetime`
   - `printhelp`
2. functions that have no IDL gateway function[1]:
   - `dist`: calculates the distance in km between two gates[2]
   - `sortcells`: Sorting of the cell properties using cell area or mean. Assume an area or mean equal to zero for cells that are marked 'dropped'
   - `updatemap`: updates the cellmap by dropping cells and reindexing the map. Leaving index 0 unused, will be used for assigning cell fringes
3. functions that have an IDL gateway function:
   - `texture`: computes a texture parameter based on a block of (`ntexrang` x `ntexazim`) pixels. The texture parameter equals the local standard deviation in the velocity field.
   - `findcells`: detects the cells in `image` using an integer threshold value of `thres` and a non-recursive algorithm which looks for neighboring pixels above threshold. On return the marked cells are contained by `cellmap`. The number of detected cells/highest index value is returned.
   - `analysecells`: analyses the cellmap found by the `findcells` procedure. Smalls cells are rejected and the cells are re-numbered according to size. The final number of cells in the cellmap is returned as an integer.

[1] does that mean that these functions can only be called by other parts of the C-code?
[2] what is a gate?

- `fringecells`: enlarges cellmap by additional fringe. First a block around each pixel is searched for pixels within a distance equal to `fringe`.
- `classification`: classifies the range gates, distinguishing between clutter, rain, fringes, empty and valid gates. It returns the classification and layer counters
- `vvp`: computes the wind velocity components.

4. gateway functions between IDL and C. The mechanism for all of these is the same: the `call_blabla` function has as input arguments `(int argc,void *argv[])`, or the number of variables `argc` that need to be communicated, followed by an array of pointers `*argv[]` to the corresponding variables. Due to this mechanism, the C-code can only operate directly on variables (memory regions, really) that are defined elsewhere, in this case, by IDL.

   - `call_texture`: receives the arguments from IDL for the call to texture, and calls `texture`. It returns[1] the updated texture array `vtex`.
   - `call_findcells`: receives the arguments from IDL for the call to `findcells`, and calls `findcells`. It returns the number of cells `Ncell` and the updated `cellmap` array.
   - `call_analysecells`: receives the arguments from IDL for the call to `analysecells`, and calls `analysecells`. It returns the number of valid cells and the updated `cellmap`.
   - `call_fringecells`: receives the arguments from IDL for the call to `fringecells`, and calls `fringecells`. For efficiency the increase in of indices by 3 and index change -1 to 0 is incoporated here as well. It returns the updated `cellmap`.
   - `call_classification`: receives the arguments from IDL for the call to `classification`, and calls `classification`. It returns `zdata`, `nzdata`, and various range gate classification collections `Npts*`[2] NOTE: 2-D arrays are used, but simply passed as if they were 1D arrays. The translation is: `x[i,j]` (IDL) = `x[i+jlayer]`, where `jlayer = j*NDATA` (C). No `REFORM()` is necessary in IDL, just declare as `x=REPLICATE(NLAYER,NDATA)`
   - `call_vvp`: receives the arguments from IDL for the call to `vvp`, and calls `vvp(int argc,void *argv[])`

5. `svd_vad1func`
6. `svd_vad2func`
7. `svd_vvp1func`
8. `svd_vvp2func`
9. `svd_vvp3func`

[1] I'm not sure it actually returns anything, because it changes the original variables (?).
[2] not sure this is correct.

10. There is also a function without a prototype: `azimuth_gap`: detects gaps in the azimuthal distribution of the radial velocity data. For this, a histogram of the azimuths of the available velocity data is made using `NGAPBIN` azimuth bins. Subsequently, the number of data points per azimuth bin is determined. When two consecutive azimuth bins contain less than `NGAPMIN` velocity points, a gap is detected. The number of velocity datapoints is `Npnt` and the dimension of each x-point is `Ndx` (`x[0...Npnt*Ndx-1]`). The azimuth coordinate is assumed to be the first x-coordinate. The function returns true (`1`) when a gap is detected and false (`0`) when no gap is found.

call_vap: receives the arguments from IDL for the call to vap, and calls vap(int argc,void *argv[])

call_fit: receives the arguments from IDL for the call to azimuth_gap and svdfit

definitions:

1. 'bins'
2. 'rays' ?= 'azimuths'
3. 'scans'

Section 2.2 from (Dokter et al., 2009) describes the reflectivity concept. Apparently, reflectivity is larger than zero, even when there are no birds. Should this 'background' value then always be subtracted?

what are unfolded radial velocities?

what is Doppler spectrum width?

2.5 Definition of terms

1. attenuation: absorbtion of energy by an object
2. azimuth: direction that the radar is pointed in, in degrees relative to north (ϕ).
3. dBZ: reflectivity, expressed in decibels according to $10 \cdot \log_{10}(\mathrm{Z})$[1]
4. elevation: angle between a horizontal plane and the direction that the radar is looking in (θ)
5. λ: wavelength of the radar. For C-Band, $\lambda \approx 5.3$ cm, for S-Band, $\lambda \approx 10$ cm.
6. PRF: pulse repeat frequency, i.e. 1/PRT
7. PRT: pulse repeat time

[1] why factor 10?

8. PPI (plan position indicator): traditional polar plot of reflectivities (think 'Das Boot'). Note that even when the radar array is positioned horizontally (elevation angle $\theta = 0$), the height above mean sea level increases with increasing distance from the radar due to curvature of the earth.

9. pseudo-PPI or CAPPI: constant-altitude PPI, traditional polar plot of reflectivities, constructed by interpolating from scans taken at various elevations.

10. range: straight-line distance from the radar to the scatterer (R)

11. range gate: range distance interval [L]

12. scatterer: the object that reflects the radar signal (it is generally assumed that there is no attenuation)

13. σ in Eq. 2.3 of (Dokter et al., 2009): assumed (effective) size of the scatterer? ...measured in L^2?

14. VVP: volume velocity processing, see (Dokter et al., 2009, p.28).

15. VAD: velocity azimuth display, see (Dokter et al., 2009, p.32).

16. QPE: quantitative precipitation estimate, see (Dokter et al., 2009, p.32).

17. Z: reflectivity factor (Dokter et al., 2009, p.11)

Appendices

Bibliography

Adriaan M. Dokter, Felix Liechti, and Iwan Holleman. Bird detection by operational weather radar. Technical Report 6, KNMI, 2009.

Index