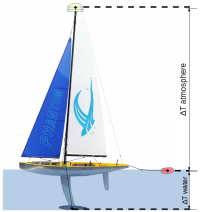


Lake thermodynamic fluxes monitoring with OSHW and FOSS4G

Yann Chemin

October 22th, 2016

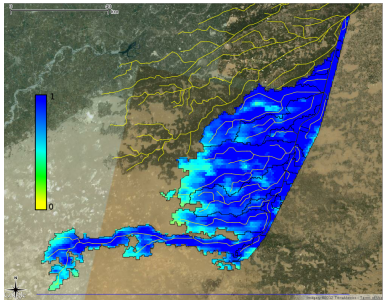
The Scientific Problem:

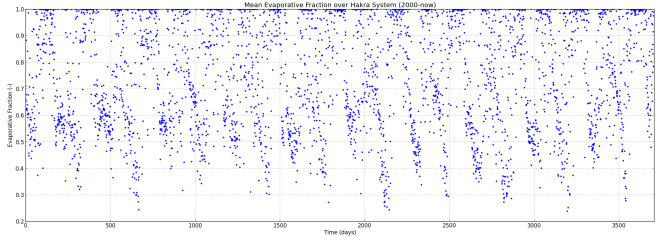
Monitoring Evapotranspiration from Space for Water Management

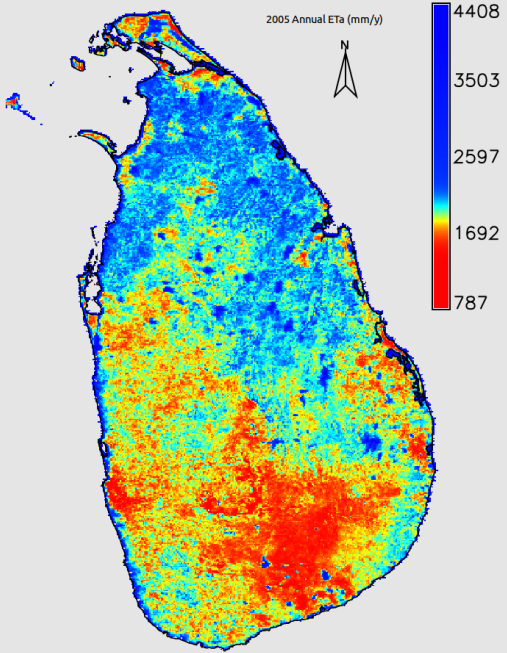
- ET = E+T
- Evaporation + Transpiration
- Evapotranspiration calibration & modeling
- Lakes of various sizes
- controlled, ungauged
- Rural tanks in Asia

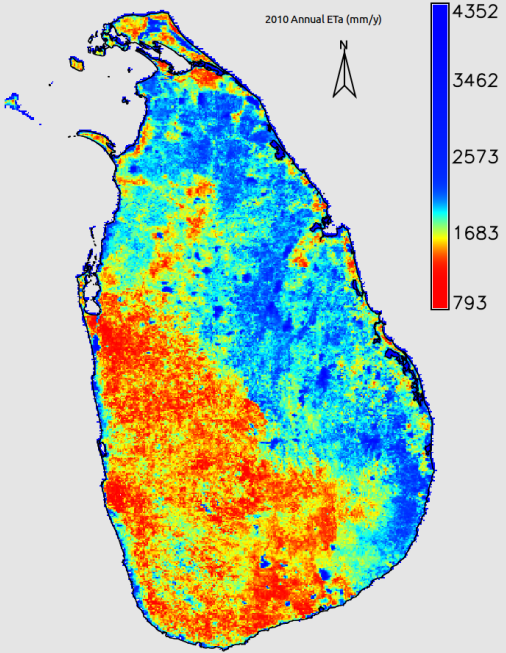
Irrigation water monitoring & management

- Map: Uniform colour is equity of water distribution
- Graph: Irrigation system equity (mm/d, daily, 12 years)

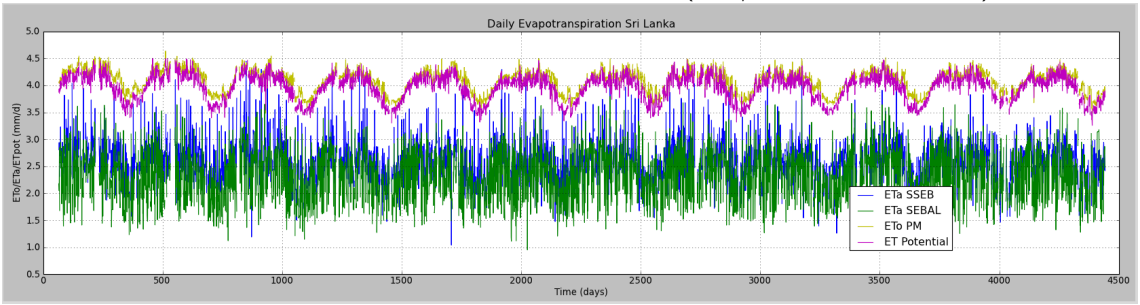





2005 Annual ETa (mm/y)
N
4408
3503
2597
1692
787


2010 Annual ETa (mm/y)
N
4352
3462
2573
1683
793

Average ET for Sri Lanka, Daily 2000-2012 (mm/d, daily, 12.3 years)

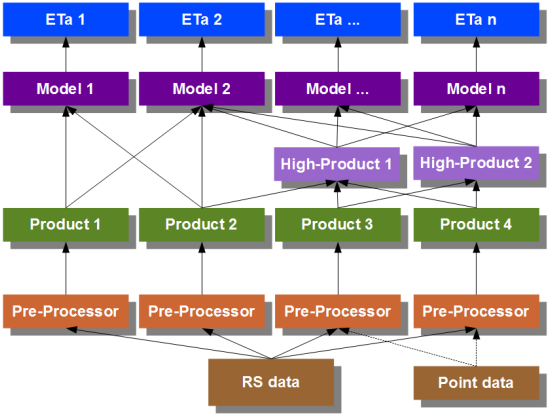


Comparison

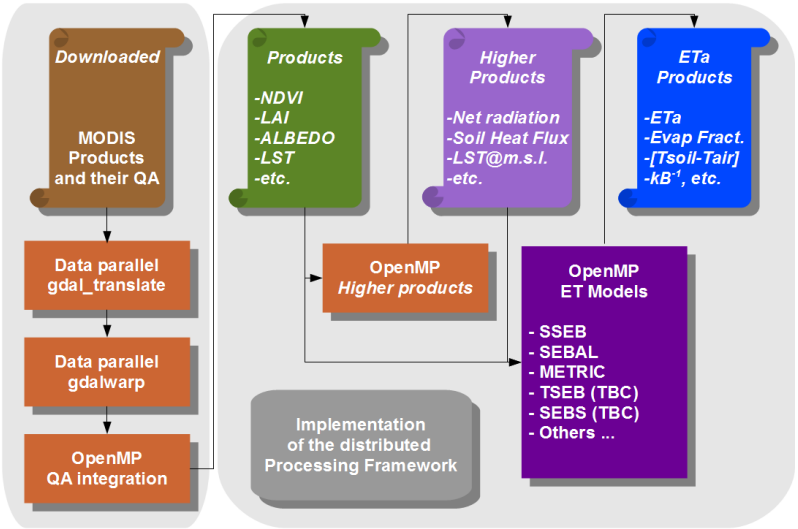
- ETo & ETpot (rad) are similar, expected.
- ETa models are not so similar, expected.
- ETo & ETpot (rad) are higher than ETa models, expected.

Chain processing
has a fundamental impact on remote sensing work:

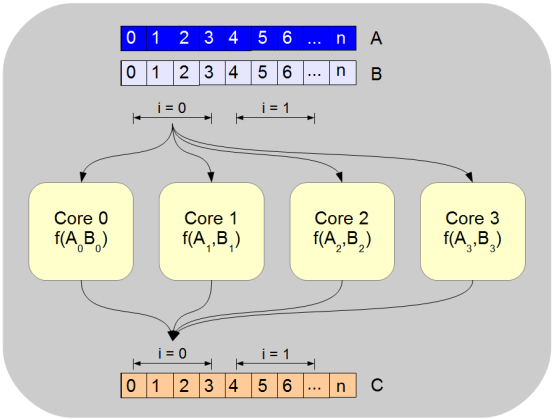
- Standardization limits bugs
- Less prone to human error
- Simpler parameterization access
- Permits to apply any number of modules to all target images
- Ensures maximum quality of generated images



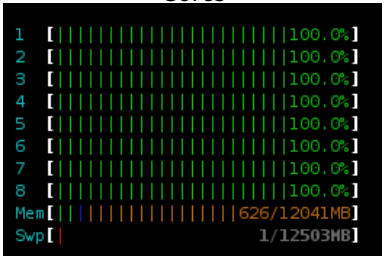
- GDAL+[C+OpenMP]
- GRASSGIS+pyGRASS+[C+OpenMP]

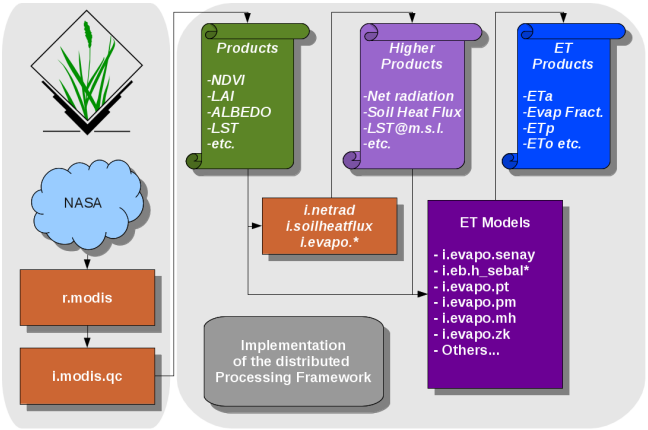

Downloaded
MODIS Products and their QA
Data parallel gdal_translate
Data parallel gdalwarp
OpenMP QA integration
Products
-NDVI
-LAI
-ALBEDO
-LST
-etc.
Higher Products
-Net radiation
-Soil Heat Flux
-LST@m.s.l.
-etc.
ETa Products
-ETa
-Evap Fract.
-[Tsoil-Tair]
-kB-1, etc.
OpenMP Higher products
OpenMP ET Models
- SSEB
- SEBAL
- METRIC
- TSEB (TBC)
- SEBS (TBC)
- Others ...
Implementation of the distributed Processing Framework

OpenMP Distribution



Cores




Products
-NDVI
-LAI
-ALBEDO
-LST
-etc.
Higher Products
-Net radiation
-Soil Heat Flux
-LST@m.s.l.
-etc.
ET Products
-ETa
-Evap Fract.
-ETp
-ETo etc.
NASA
r.modis
i.modis.qc
i.netrad
i.soilheatflux
i.evapo.*
ET Models
- i.evapo.senay
- i.eb.h_sebal*
- i.evapo.pt
- i.evapo.pm
- i.evapo.mh
- i.evapo.zk
- Others...
Implementation of the distributed Processing Framework

Pythonizing GRASS:
From Shell commands to Python functions

metaModule concept

1. **GRASS GIS:** Specific image processing modules
2. **PyWPS:** GRASS modules called by Python
3. **GRASS script:** GRASS modules called by Python: metaModule
4. **pyGRASS:** GRASS modules called as Python function: metaModule
5. **PyWPS v4:** pyGRASS metaModule used directly

Summary for Landsat pyGRASS metaModule

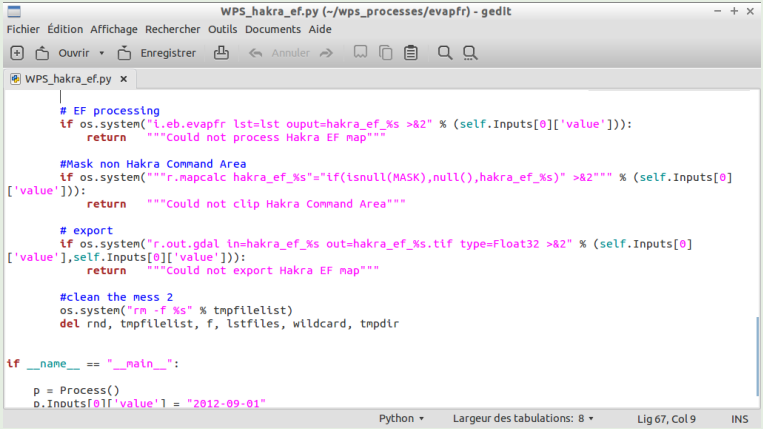
```
from grass import script as g
from grass.script import setup as gsetup
gisbase=os.environ['GISBASE']
gsetup.init(gisbase,gisdb,location,mapset)
from grass.pygrass.modules.shortcuts import raster as r
from grass.pygrass.modules.shortcuts import imagery as i
from grass.pygrass.modules.shortcuts import display as d

r.mapcalc(expression="vis=18",overwrite=OVR)
r.in_gdal(input=L7f,output=L7r,flags="e",overwrite=OVR)
r.proj(input="dem",location="Myanmar",memory=10000,resolution=90.0,overwrite=OVR)

i.landsat_toar(input_prefix=pref,output_prefix=outpref,
        metfile=metadata[0],sensor=LSENSOR,quiet=QIET,overwrite=OVR)

i.atcorr(input=b, elevation="dem", visibility="vis", parameters=prm,
        output=b_out, flags="ra", range=[0,1],quiet=QIET,overwrite=OVR)

i.landsat_acca(input_prefix=b_in,output=b_clouds,overwrite=OVR)
r.mask(raster=b_clouds,flags="i",overwrite=True)

i.vi(red=b3,nir=b4,output=b_ndvi,viname="ndvi",quiet=QIET,overwrite=OVR,finish_=False)
i.albedo(input=b_in,output=b_albedo,flags="lc",quiet=QIET,overwrite=OVR,finish_=False)
i.emissivity(input=b_ndvi, output=b_emissivity,quiet=QIET,overwrite=OVR,finish_=False)
```

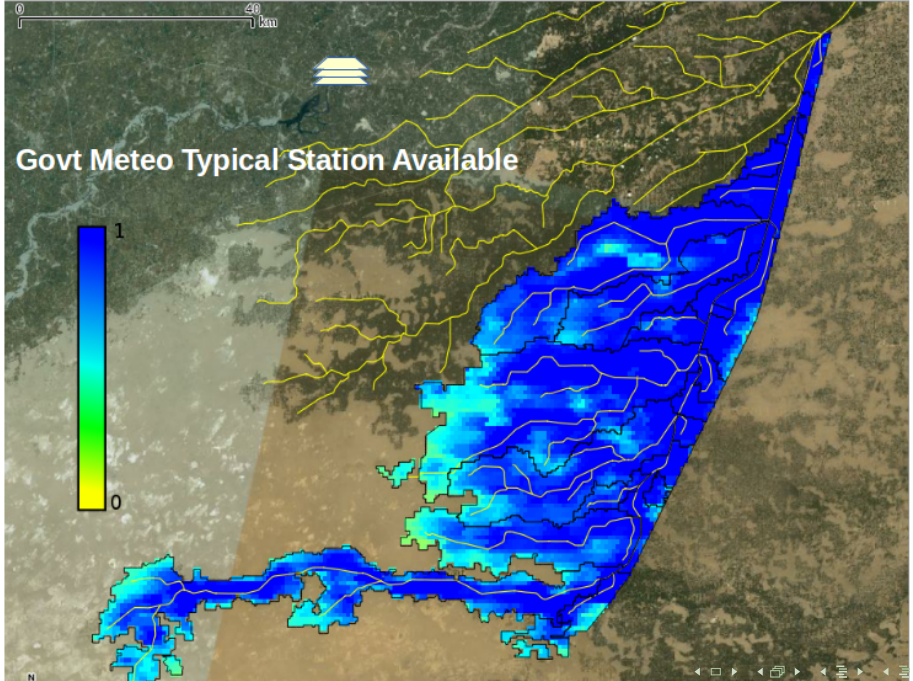
http://grasswiki.osgeo.org/wiki/Python/pygrass

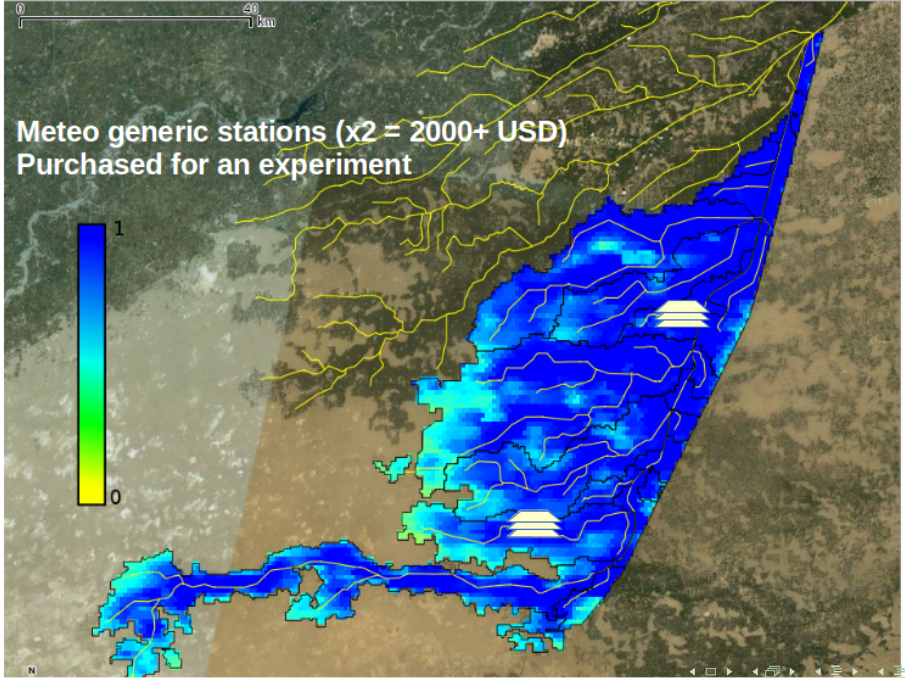
Developed by Jachym Cepicky (http://les-ejk.cz/)

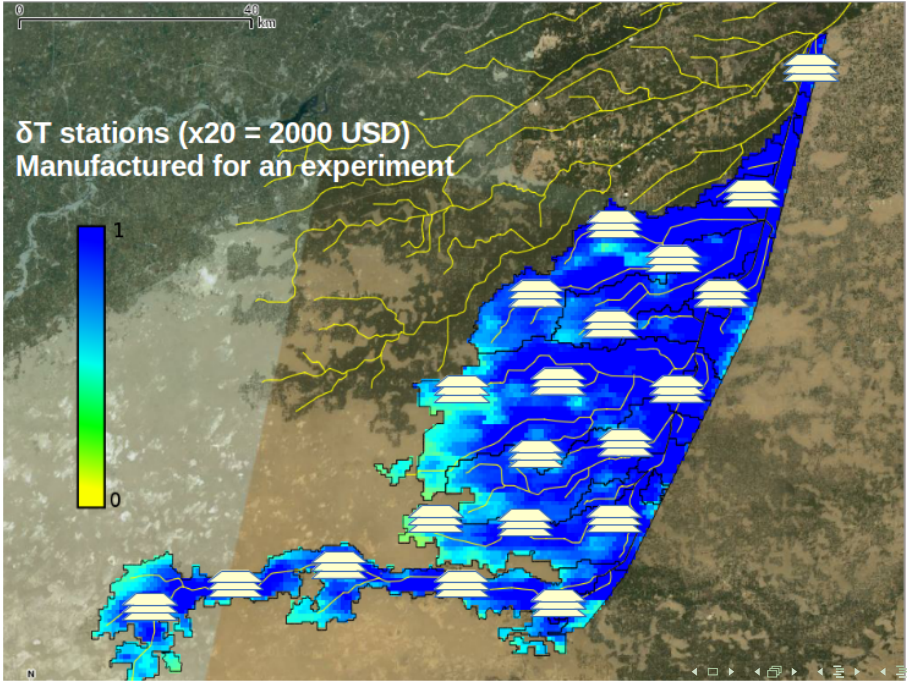
- OGC WPS standard
- Server side
- Written in Python Language
- Version 4 in the making
- v4 Low-level API: integration with GRASS GIS
- v4 Possible pyGRASS support

PyWPS v2 style



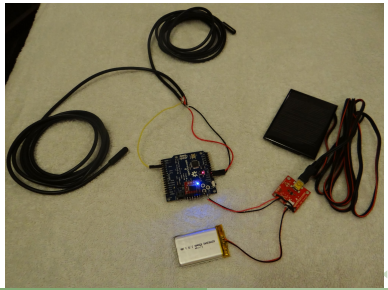

0
40
km
Govt Meteo Typical Station Available
1
0
N

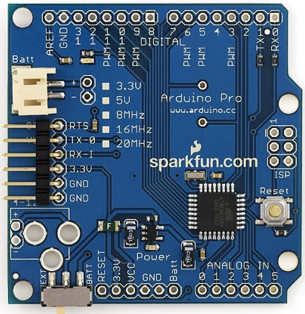

0
40
km
Meteo generic stations (x2 = 2000+ USD)
Purchased for an experiment
1
0
N

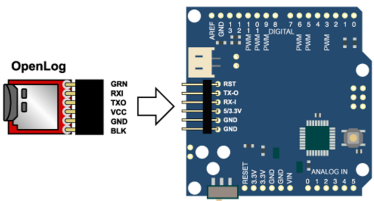

0
40
km
δT stations (x20 = 2000 USD)
Manufactured for an experiment
1
0
N

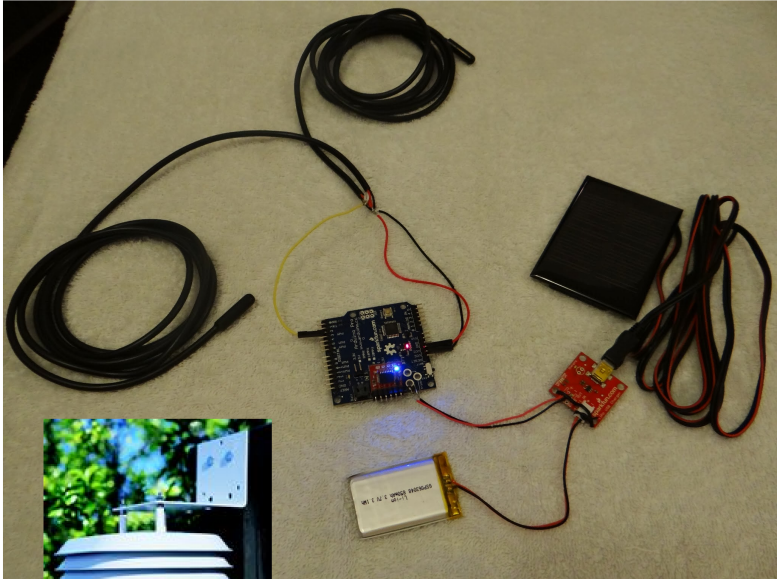
Micro Weather Station v1:
Temperature Profiler for ET models calibration

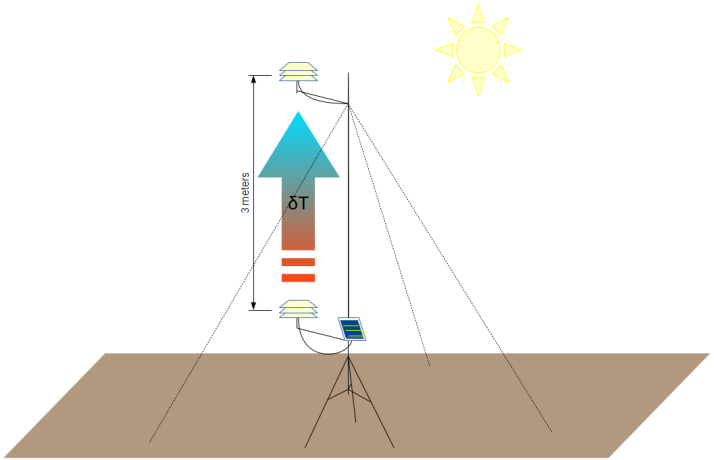
- Arduino Pro 3.3V
- Water-proof Digital Temperature Sensors
- Li-ion Battery + Solar Panel
- OpenLog data logger with SD card
- Cost $<$ 100 USD

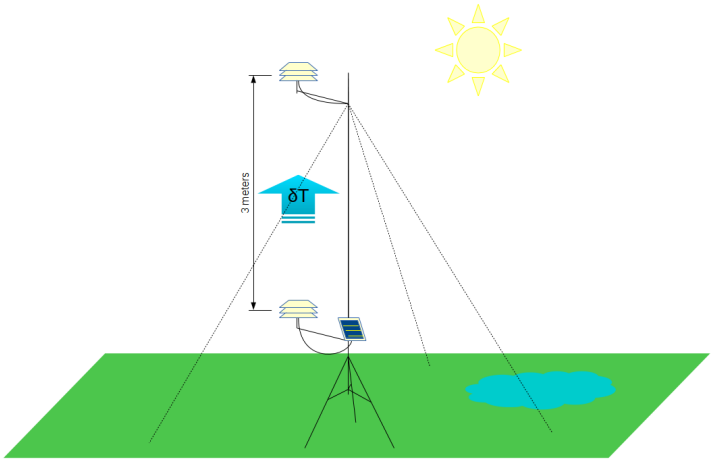



Batt
DIGITAL
Arduino Pro
www.arduino.cc
sparkfun.com
3.3V
5V
8MHz
16MHz
20MHz
RTS
TX→0
RX←1
3.3V
GND
GND
ISP
Reset
Power
ANALOG IN
RESET
3.3V
5V
GND
GND
RAW

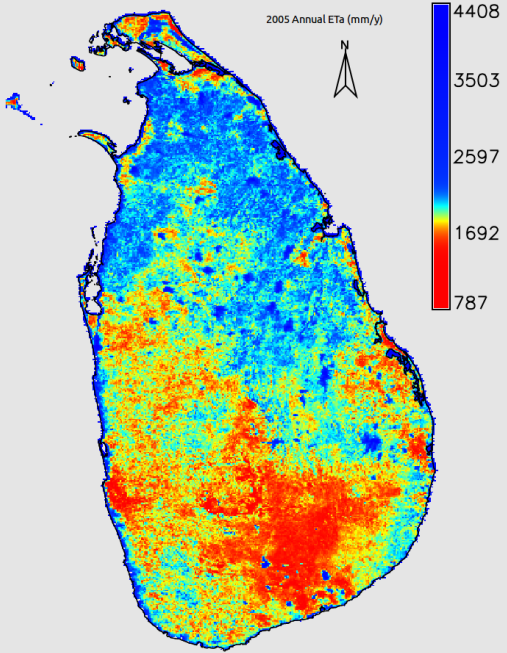

OpenLog
GRN
RXI
TXO
VCC
GND
BLK
RST
TX-O
RX-I
3V3
GND
GND
DIGITAL
ANALOG IN

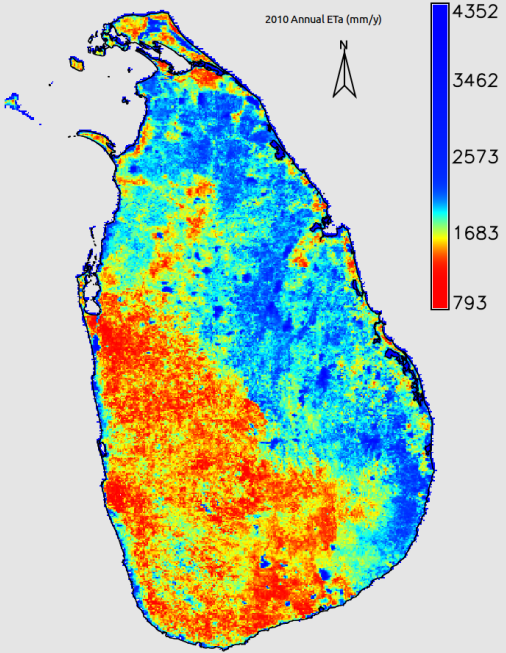



3meters
δT


3meters
δT

This is nice but what about the lakes, reservoirs, rural tanks?
How to calibrate them?

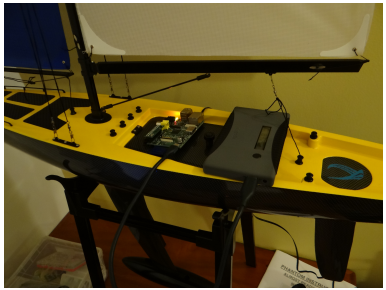

2005 Annual ETa (mm/y)
N
4408
3503
2597
1692
787

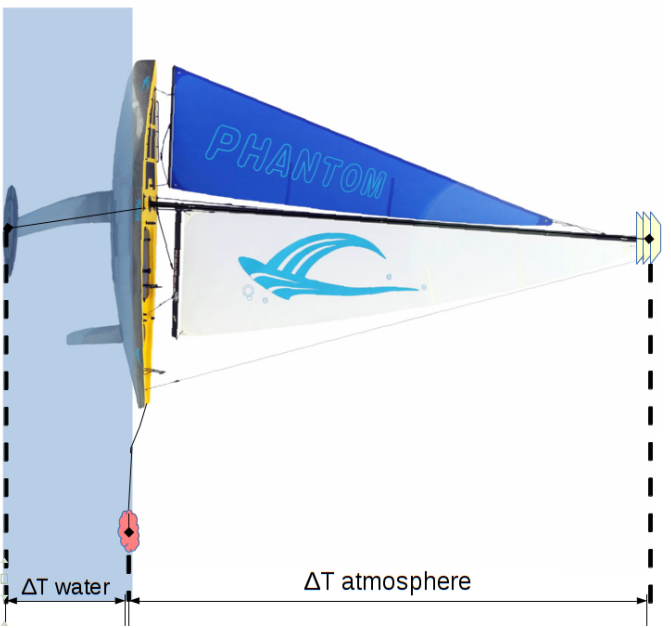

2010 Annual ETa (mm/y)
N
4352
3462
2573
1683
793

We need a floating weather station, for few days a year to collect data.

Amitomi is a 1m-class autonomous sailing boat
Designed to survey small tanks temperature gradient for calibrating Evaporation models

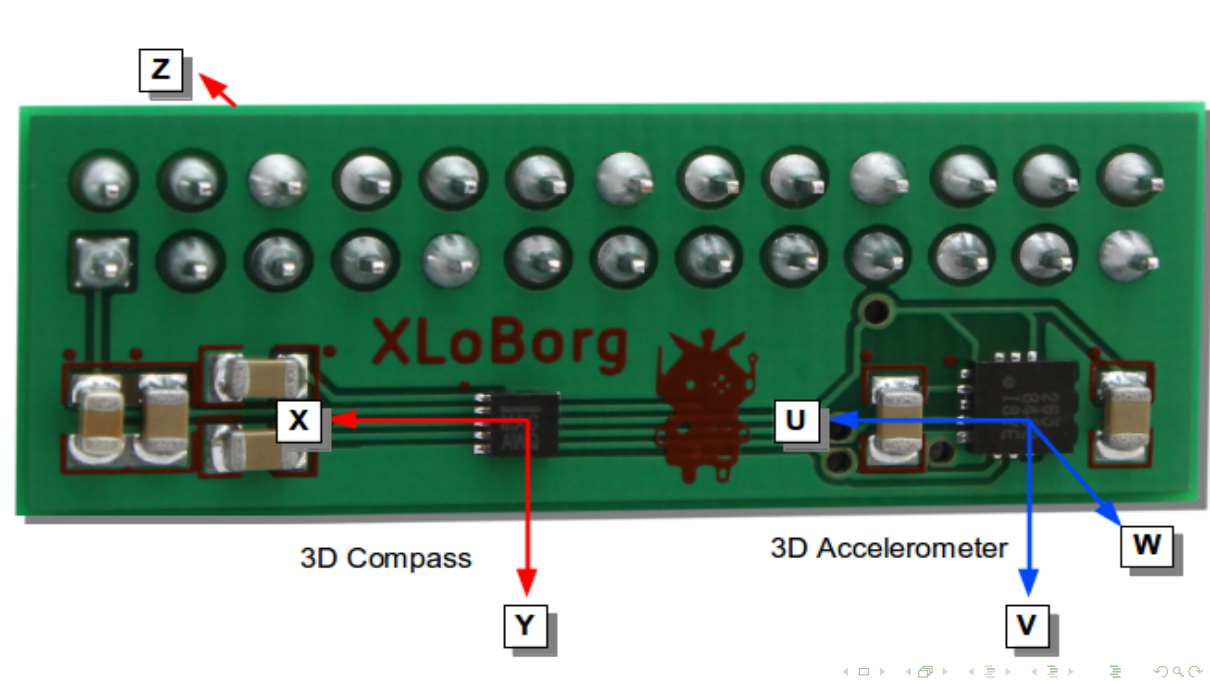
https://sites.google.com/site/amitomiautoboat




PHANTOM
ΔT water
ΔT atmosphere

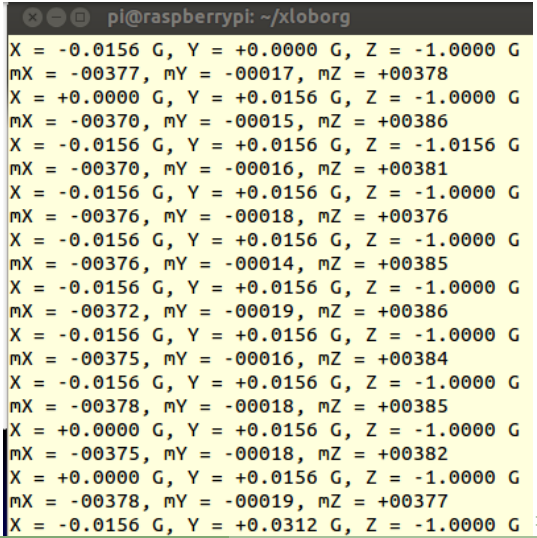
Components

- RaspberryPI v2
- + XloBorg Accelerometer
- + GPS shield
- + motor shield
- + Python code


Z
XLoBorg
X
U
3D Compass
3D Accelerometer
Y
V
W

Compass/Accelerometer software

- Python - XloBorg
- 3D Compass
- 3D Accelerometer
- Pitch/Yaw/Roll
- Azimuth (for Bearing)



Attitude software

- Python - XloBorg
- 3D Compass
- 3D Accelerometer
- Pitch/Yaw/Roll
- Attitude of the boat

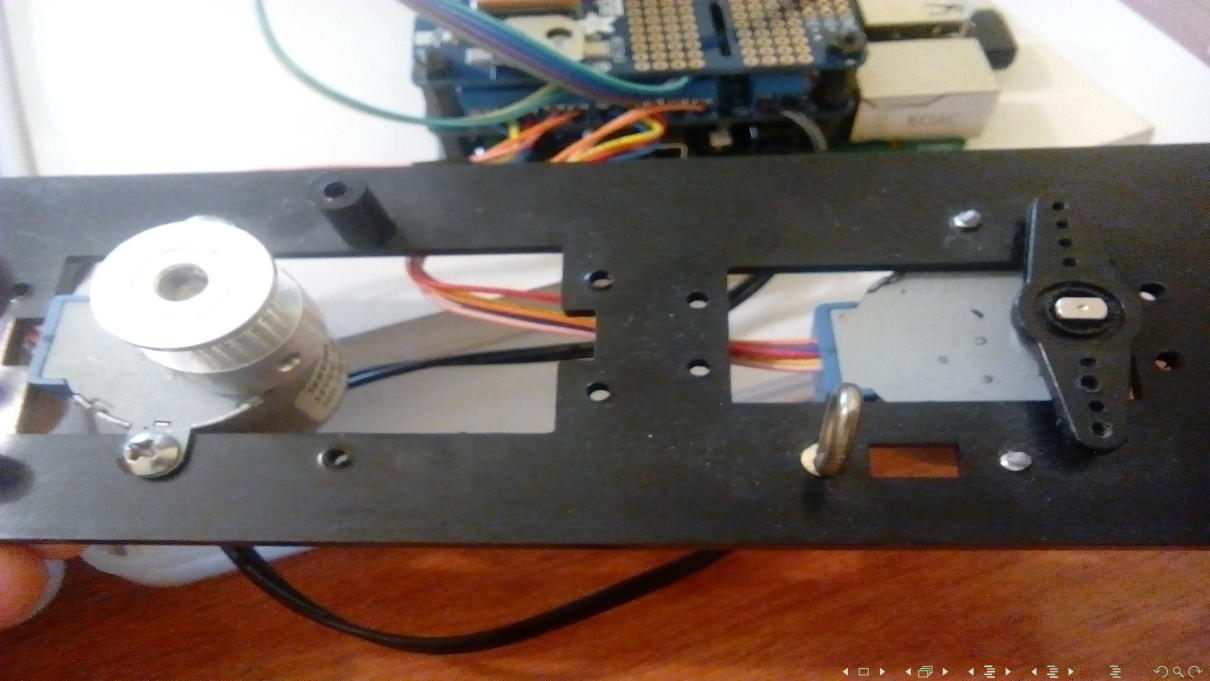
```
# Start the XLoBorg module (sets up devices)
XLoBorg.Init()

compass = arrow(pos=(0,0,0), axis=(0,0,0), shaftwidth=1, color=(1,0,0))
boat=arrow(pos=(0,0,0), axis=(0,0,0), shaftwidth=1, color=(0,0,1))
while True:
    time.sleep(1)
    # Read and display the raw accelerometer readings
    print 'X = %+01.4f G, Y = %+01.4f G, Z = %+01.4f G' % XLoBorg.ReadAccelerometer()
    u,w,v=XLoBorg.ReadAccelerometer()
    u *= 1000.0
    v *= 1000.0
    w *= 1000.0
    # Read and display the raw magnetometer readings
    print 'mX = %+06d, mY = %+06d, mZ = %+06d' % XLoBorg.ReadCompassRaw()
    x,z,y=XLoBorg.ReadCompassRaw()
    compass.visible=False
    compass = arrow(pos=(0,0,0), axis=(x,y,z), shaftwidth=1, color=(1,0,0))
    boat.visible=False
    boat=arrow(pos=(0,0,0), axis=(u,v,w), shaftwidth=1, color=(0,0,1))
```

XLoBorg
Ultimate GPS Pi HAT
PA6H1F1503
adafruit
INDUSTRIES
Ext Antenna
PPS
Fix
CR1220
+3V
+5V
GND

GPS/Bearing software

- Python - GPS
- GPS RPIv2 Hat
- Location
- Azimuth (for Bearing)

```
import math as M

def bearing(lat1, lat2, lon1, lon2):
    """
    http://stackoverflow.com/questions/17624310/geopy-calculating-gps-heading-bearing
    """
    dLon = lon2 - lon1
    y = M.sin(dLon) * M.cos(lat2)
    x = M.cos(lat1)*M.sin(lat2)-M.sin(lat1)*M.cos(lat2)*M.cos(dLon)
    brng = M.degrees(M.atan2(y, x))
    if (brng < 0):
        brng += 360
    return brng
```



Motors software

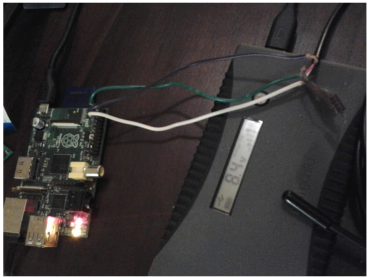
- Python - motor hat
- Motor RPIv2 Hat
- Stepper Winch (sails)
- Servo Rudder

```
from Adafruit_MotorHAT import Adafruit_MotorHAT, Adafruit_StepperMotor
import time
import atexit
import threading
import random

# create a default object, no changes to I2C address or frequency
mh = Adafruit_MotorHAT()

# create empty threads (these will hold the stepper 1 and 2 threads)
st1 = threading.Thread()
st2 = threading.Thread()


# recommended for auto-disabling motors on shutdown!
def turnOffMotors():
        mh.getMotor(1).run(Adafruit_MotorHAT.RELEASE)
        mh.getMotor(2).run(Adafruit_MotorHAT.RELEASE)
        mh.getMotor(3).run(Adafruit_MotorHAT.RELEASE)
        mh.getMotor(4).run(Adafruit_MotorHAT.RELEASE)

atexit.register(turnOffMotors)

#Stepper 1 is the SAIL WINCH, FASTER
S1 = mh.getStepper(64, 1)       # 200 steps/rev, motor port #1
#Steppet 2 is the RUDDER, SLOW
S2 = mh.getStepper(64, 2)       # 200 steps/rev, motor port #2
#Set Speed
S1.setSpeed(6400)               # 30 RPM
S2.setSpeed(30)                 # 30 RPM
```

AmiTomi's brain is the RaspberyPI python code:

- Skipper: the captain/navigator software
- Waypoint sorter: optimizer for route
- Sensor datalogger: simultaneous sensing
- Mapper: import data and 3D interpolation

RaspberryPI GPIO connecting to temperature digital sensors (2m cables)



Skipper software

- Python
- Rolls through the whole mission
- “Captain on the Boat”
- Use the Waypoints to navigate bearing
- Adjust rudder to keep bearing
- Adjust sail tension to optimize speed

```
def bbox(waypoints):
    #Step 1: create a bounding box for the waypoints coordinates

def bbox_center(bnd):
    #Step 2: define the central point of the bounding box

def bbox_half_diag_length(bnd,bnd_center):
    #Step 3: define the half diagonale length

def upwind_virtual_waypoint(upwinddir,bnd_center,radius):
    #step 4: Define upwind virtual waypoint with radius distance from bbox center

def upwind_waypoint_find(waypoints,upw):
    #step 5: find most upwind waypoint by distance to virtual upwind waypoint

def waypoint_dist_to_firstwp(waypoints,first_waypoint):
    #step 6: waypoints distance to first waypoint

def distangle(waypoints,upwinddir,upw):
    #step 7: Create matrices of distance and angle from wind

#Boat should start sailing to first waypoint from here...

def route(waypoints, distM, angleM, first_waypoint):
    #step 8: Create a downwind route through the waypoints by ordering them

#End of routing steps, the final route has been computed and stored on disk


#Navigational elements
def getbearing():
    # Read and display the raw magnetometer readings

def makeactualpos(last_bearing, last_true_gps_pos, last_kalman_pos, last_time, last_velocity, last_attitude):


#Still Working on those...
def navigate(mode,waypoint):


def survey(mode,waypoint):
    #set course to that waypoint
```

Waypoints sorting software

- Python-openopt
- Assess waypoints
- Guess the wind direction
- Compute upwind mission start
- Compute route through downwind waypoints

```
#!/usr/bin/env python
import numpy as np
#Navigation test engine
#Loads fake datasets and runs the skipper through the data

releasepoint = np.genfromtxt('releasepoint001.csv',delimiter=',')
waypoints = np.genfromtxt('waypoints001.csv',delimiter=',')
upwinddir = np.genfromtxt('upwinddir001.csv',delimiter=',')
print 'releasepoint = ', releasepoint
print 'waypoints = w', waypoints
print 'upwinddir = ', upwinddir, '[0.0=North]'

#set training velocity at 1m/s (motorbike dataset)
v=1.0

#Sort the waypoints by most upwind first
#Step 1: create a bounding box for the waypoints coordinates
def bbox(waypoints):
    #First Column is latitude
    bbox_north=waypoints[:,0].max()
    bbox_south=waypoints[:,0].min()
    bbox_east=waypoints[:,1].max()
    bbox_west=waypoints[:,1].min()
    print 'BOUNDING BOX'
    print '\t',bbox_north
    print bbox_west,bbox_east
    print '\t',bbox_south
    return(bbox_north,bbox_south,bbox_east,bbox_west)

bnd = bbox(waypoints)

#Step 2: define the central point of the bounding box
def bbox_center(bnd):
    lat = bnd[1] + (bnd[0]-bnd[1])/2.0
```

Mapping software

- Python-OGR (GDAL)
- pyGRASS
- Receive 3 temperature data with Location
- Convert to vector data
- Compute 3 interpolation layers (in water, on water, in air)
- Compute conduction and convection flux
- Model evaporation maps

FOSS and FOSS4G software

- Python-gps (GPS data)
- Python-i2ctools (Compass/Temperature data)
- Python-XIoBorg (Compass data)
- Python-openopt (Waypoints downwind sorting openopt.org)
- Python-MotorPiTX (servo control for sails & rudder)
- (py)GRASS (live processing of 3D GIS data)
- If online: PyWPS, SOS/network reporting.

FOSS4G natural extension is Open Source Hardware

- **RaspberryPI:** Small PC (ARM v8, Linux)
- **Arduino:** Micro-controller
- **OpenLog:** Data Logger
- **GDAL/OGR:** Flexible sensor raw data manipulation
- **GRASS GIS:** Mobile FOSS4G powerhouse
- **PyWPS:** Online GRASS GIS processing
- **Together:** Flexible all-in-one sensor-to-map solutions

Thank you