Terminator VII : Robots of the deep blue

ENSTA BZH - benoit.zerr@ensta-bretagne.org

Getting started with DDBOAT

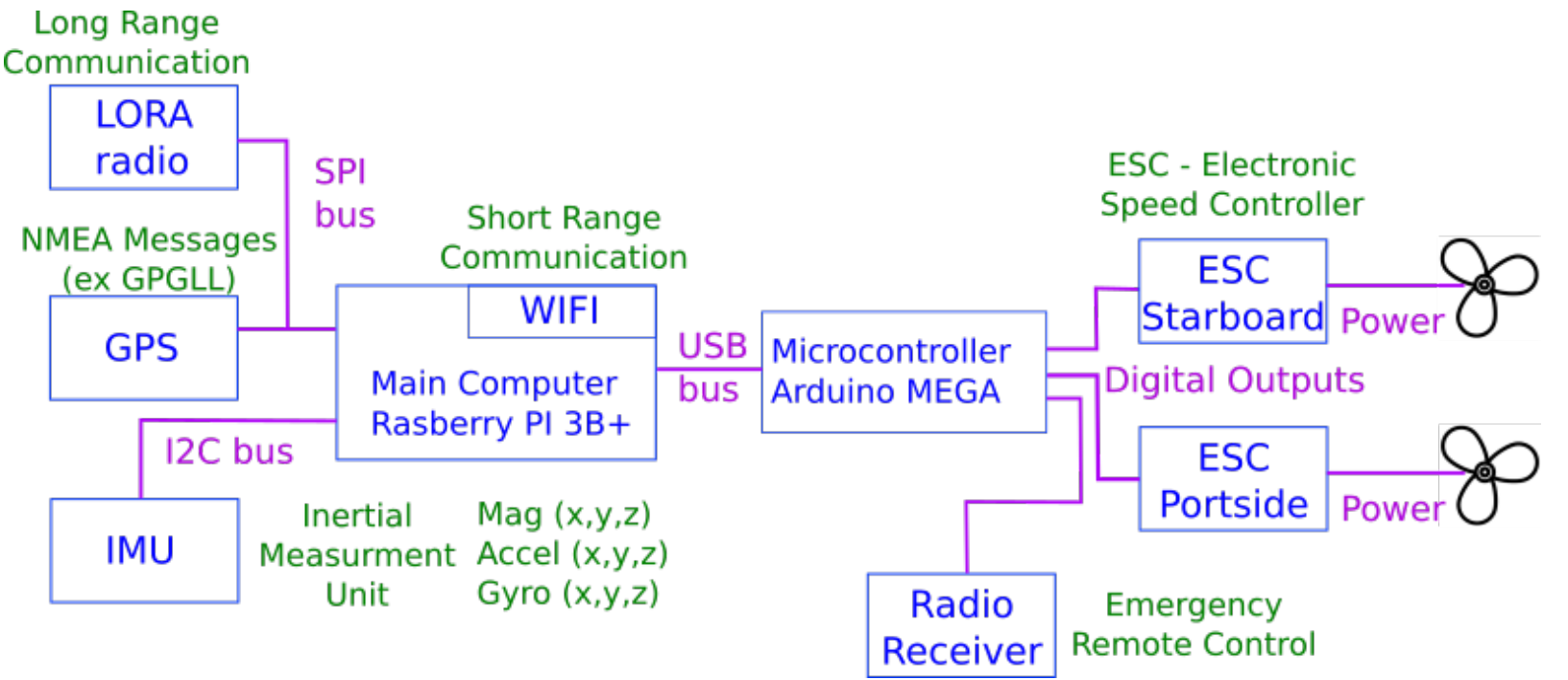
- Building the teams
- The DDBOAT architecture
- Powering the DDBOAT
- Remote control
- Talking to the DDBOAT
- First very simple program
- A bit more complex, program getting the GPS position
- Acquiring GPS data and displaying it on a map (e.g. Google Earth)
- Acquiring data from the IMU (Inertial Measurement Unit)
- Calibrating the compass
- Powering the propellers

Building the Teams

- Each team will consist of 3 students (7 teams)
- At least one student should have experience with Python programming
- Every team should have a laptop
- Every team will be given a DDBOAT

The DDBOAT Architecture

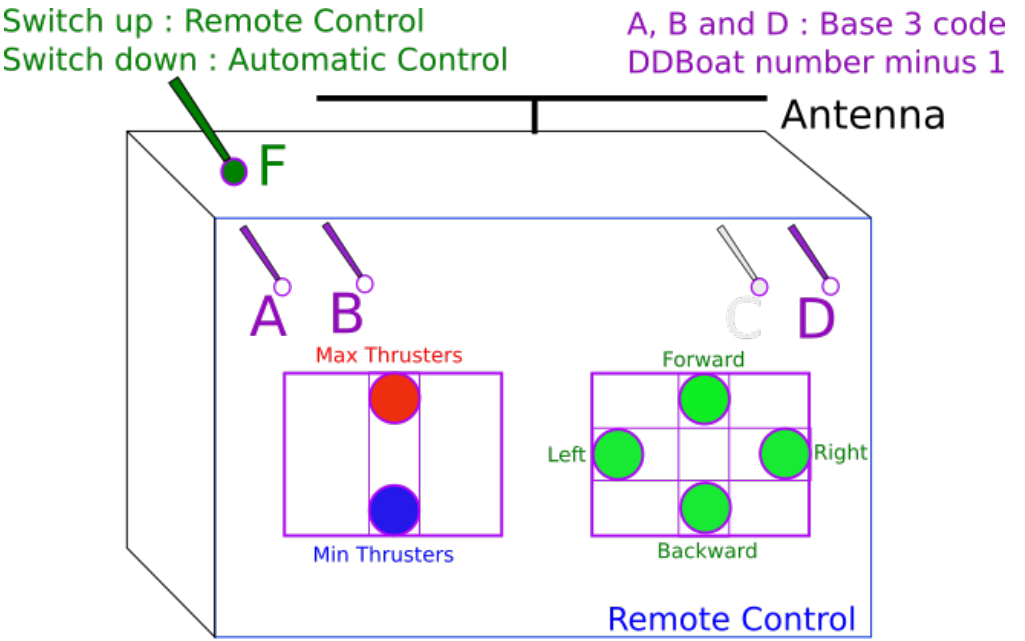
DDBoat : **D**ifferential **D**rive **Boat** - More intuitive than classical propeller+ruder boats.



Powering the DDBOAT

- Check and install the batteries :
  - The DDBOAT is powered by two LIPO 2S batteries (motors + electronics)
  - Check the voltage of the batteries before starting a mission
    - The voltage should be greater than 7.5V for a short (standard) mission
    - The voltage should be greater than 8.0V for a long mission
  - Attach (scratches) the batteries on the 2 sides of the electronic box
  - Plug the batteries to the electronic box using the connectors
- Power on the electronic box (switch 1/0 on top of the box) :
  - The yellow LED on the electronic box should be on (electronics powered)
  - Simple melody should be played (motors powered)
  - After the melody has played 3 times, the DDBOAT is ready to be controlled

Remote control



3 remote controls can be used depending on the color tape on the electronic box (black, grey or green). Choose the right one !

Remote control (cont.)

Select the right (your !) DDBoat :

- The left joystick controls the power (set to low to medium at the beginning)
- The right joystick controls the direction (forward, backward, left, right)
- The switch F has 2 positions : up (remote control) and down (autonomous mode)
- The switches A,B and D have 3 positions indicating the code of the DDBoat in base 3
- The code of the DDBoat is the boat number minus 1, coded in base 3
- Example : DDBoat 12 has the code 11 in decimal and 102 in base 3
- 102 corresponds $1 * 3^2 + 0 * 3^1 + 2 * 3^0 = 11$ in decimal
- The positions of the switches are : A=2, B=0, D=1

Talking to the DDBOAT

- The DDBoat is connected to the DARTAP WIFI network (for short range communication)
- DARTAP pswd is ABACA (network deliberately unsecure !)
- Each DDBoat has a fix IP address (172.20.25.2xx) with xx the boat number on 2 digits
- Example: for DDBoat 2, the IP address is 172.20.25.202

Talking to the DDBOAT (cont.)

Connection test :

- Connect your laptop to the DARTAP WIFI network
- Open a terminal (powershell on Windows) and type :
  `ping 172.20.25.2xx` (replace xx by the boat number)
- You should see the ping responses from the DDBoat with a ping time less than 100ms
- Connect to the DDBoat main computer (Raspberry Pi) using SSH :
  `ssh best@172.20.25.2xx` (password is `best`)
- type `ls -l` to see the files on the DDBoat, you should see a folder named `drivers-ddboat-v2`, a file named `calib.py` and other files ...
- Keep the SSH connection open, we will use it later to execute Python programs on the DDBoat

First very simple program

The program `test.py` will just print the message "Hi!" on the screen.

- Open the file `test.py` with the editor of your choice
- Type the following code :

```
print("Hi!")
```

- Copy the program from your laptop to the DDBoat using the command :
  `scp test.py best@172.20.25.2xx:`
- The program is now on the DDBoat, go back to the previous terminal where you are connected to the DDBoat
- Run the program with the command : `python3 test.py`
- You should see the message "Hi!" on the screen

A bit more complex program

The program gps.py will print the GPS position of the DDBoat.

- Open the file gps.py and type the following code :

```
import sys
import os
import time
# access to the drivers
sys.path.append(os.path.join(os.path.dirname(__file__), 'drivers-ddboat-v2'))
import gps_driver_v2 as gpddrv
gps = gpddrv.GpsIO() # create a GPS object
gps.set_filter_speed("0") # allowing GPS measures to change even if the DDBoat is not moving
cnt = 5 # takes 5 GPS measures
while True:
    gll_ok,gll_data=gps.read_gll_non_blocking()
    if gll_ok: # GPGLL message received
        print (gll_data)
        cnt -= 1
        if cnt==0:
            break
    time.sleep(0.01)
```

- Use the same procedure as before to copy the program to the DDBoat and run it

Acquiring GPS data and displaying it on a map

- `gll_data` is a list of 5 values : latitude, east/west, longitude, north/south, time
- The latitude and longitude are in degrees and decimal minutes : DDMM.MMMM
- The east/west and north/south values are 'E' or 'W' and 'N' or 'S'
- The time is in the format HHMMSS.SSS
- You can convert the latitude and longitude to decimal degrees using the function `cvt_gll_ddmm_2_dd(gll_data)`
- Convert the (lat,lon) coordinates to meters (x,y) using a projection (UTM)
- Take a reference point (lat_{ref},lon_{ref}) on the lake and convert it to meters (x_{ref},y_{ref})
- Now we can compute the distance and the heading to the reference point :
  $dx = x - x_{ref}$, $dy = y - y_{ref}$, $d = \sqrt{dx^2 + dy^2}$, $\theta = \arctan 2(dy, dx) * 180/\pi$
- Be careful, trigonometric angles and geographic angles are different ! $\Theta_{geo} = 90 - \Theta_{trig}$

Acquiring GPS data and displaying it on a map (cont.)

Function to convert latitude and longitude to decimal degrees :

```
def cvt_gll_ddmm_2_dd (st):
    ilat = st[0]
    ilon = st[2]
    olat = float(int(ilat/100))
    olon = float(int(ilon/100))
    olat_mm = (ilat%100)/60
    olon_mm = (ilon%100)/60
    olat += olat_mm
    olon += olon_mm
    if st[3] == "W":
        olon = -olon
    return olat,olon
```

Acquiring GPS data and displaying it on a map (cont.)

Function to convert (lat,lon) to (x,y) in meters using an UTM projection :

At the beginning of the program, add the following lines ti define the projection :

```
from pyproj import Proj, transform
projDegree2Meter = Proj("+proj=utm +zone=30, +north +ellps=WGS84 +datum=WGS84 +units=m +no_defs")
```

After each GPS measurement, convert it to meters :

```
lat,lon = cvt_gll_ddmm_2_dd(gll_data) # convert DDMM.MMMM to DD.DDDDD
x,y = projDegree2Meter(lon, lat) # convert to meters
```

Acquiring GPS data and displaying it on a map (cont.)

Add a reference point at the beginning of the program, and convert it to meters :

```
# reference point "Guerledan's lake"
reference_lat = 48.199111
reference_lon = -3.014930
reference_x,reference_y = projDegree2Meter(reference_lon,reference_lat)
```

After each GPS measurement, compute the distance and the heading to the reference point :

```
dx = x - reference_x
dy = y - reference_y
d = (dx**2 + dy**2)**0.5
theta = 90 - np.atan2(dy,dx)*180/math.pi
```

Convert the heading from trigonometric to geographic angle :

```
theta_geo = 90 - theta
```

Acquiring GPS data and displaying it on a map (cont.)

Program example (part 1/3):

```
import sys
import os
import time
from pyproj import Proj, transform
import numpy as np
# access to the drivers
sys.path.append(os.path.join(os.path.dirname(__file__), 'drivers-ddboat-v2'))
import gps_driver_v2 as gpddrv
gps = gpddrv.GpsIO() # create a GPS object
gps.set_filter_speed("0") # allowing GPS measures to change even if the DDBoat is not moving
# define the projection from WGS84 (lon,lat in degrees) to UTM (meters)
# UTM zone for Brittany is 30N
projDegree2Meter = Proj("+proj=utm +zone=30, +north +ellps=WGS84 +datum=WGS84 +units=m +no_defs")
# reference point "Guerledan's lake"
reference_lat = 48.199111
reference_lon = -3.014930
reference_x,reference_y = projDegree2Meter(reference_lon,reference_lat)
print ("ref",reference_x,reference_y,reference_lon,reference_lat)
```

Acquiring GPS data and displaying it on a map (cont.)

Program example (part 2/3):

```
# convert lat,lon from DDMM.MMMM to DD.DDDDD
def cvt_gll_ddmm_2_dd (st):
    ilat = st[0]
    ilon = st[2]
    olat = float(int(ilat/100))
    olon = float(int(ilon/100))
    olat_mm = (ilat%100)/60
    olon_mm = (ilon%100)/60
    olat += olat_mm
    olon += olon_mm
    if st[3] == "W":
        olon = -olon
    return olat,olon
```

Acquiring GPS data and displaying it on a map (cont.)

Program example (part 3/3):

```
cnt = 5 # takes 5 GPS measures
while True:
    gll_ok,gll_data=gps.read_gll_non_blocking()
    if gll_ok: # GPGLL message received
        lat,lon = cvt_gll_ddmm_2_dd(gll_data) # convert DDMM.MMMM to DD.DDDDD
        x,y = projDegree2Meter(lon, lat) # convert to meters
        lat_check,lon_check = projDegree2Meter(x,y,inverse=True) # check conversion OK
        dx = x - reference_x
        dy = y - reference_y
        distance = np.sqrt(dx*dx+dy*dy)
        heading_trigo = np.degrees(np.arctan2(dy,dx))
        heading_geo = 90.0 - heading_trigo  # convert from trigonomety to geographic
        print ("lat=%.4f lon=%.4f (check %.4f %.4f) x=%.2f y=%.2f dx=%.2f, dy=%.2f, distance=%.2f, heading=%.2f"%
        (lat,lon,lat_check,lon_check,x,y,dx,dy,distance,heading_geo))
        cnt -= 1
        if cnt==0:
            break
    time.sleep(0.01)
```

Acquiring GPS data and displaying it on a map (cont.)

Log the GPS data in a log file (example GPX).

Add the following lines at the beginning of the program :

```
import simplekml
# Create a KML object
kml = simplekml.Kml()
```

After each GPS measurement, log the (lat,lon) data in the KML file :

```
pnt = kml.newpoint(name="GPS", coords=[(lon,lat)])
```

At the end of the program, save the KML file :

```
kml.save("gps_data.kml")
```

Example of gps_data.kml GPS log file

```
<kml xmlns="http://www.opengis.net/kml/2.2" xmlns:gx="http://www.google.com/kml/ext/2.2">
    <Document id="1">
        <Placemark id="3">
            <name>GPS</name>
            <Point id="2">
                <coordinates>-3.0142666666666664,48.19931,0.0</coordinates>
            </Point>
        </Placemark>
        <Placemark id="5">
            <name>GPS</name>
            <Point id="4">
                <coordinates>-3.0142666666666664,48.19931,0.0</coordinates>
            </Point>
        </Placemark>
        <Placemark id="7">
            <name>GPS</name>
            <Point id="6">
                <coordinates>-3.0142650000000004,48.19931,0.0</coordinates>
            </Point>
        </Placemark>
        <Placemark id="9">
            <name>GPS</name>
            <Point id="8">
                <coordinates>-3.0142650000000004,48.19931,0.0</coordinates>
            </Point>
        </Placemark>
        <Placemark id="11">
            <name>GPS</name>
            <Point id="10">
                <coordinates>-3.014263333333333,48.19930833333333,0.0</coordinates>
            </Point>
        </Placemark>
    </Document>
</kml>
```

This file can be opened with Google Earth or Geoportail to display the GPS data on a map.

Acquiring GPS data and displaying it on a map (cont.)

Log the GPS data in a log file (example GPX).

Add the following lines at the beginning of the program :

```
import gpxpy.gpx
# Create a GPX object
gpx = gpxpy.gpx.GPX()
gpx_track = gpxpy.gpx.GPXTrack()
gpx.tracks.append(gpx_track)
gpx_segment = gpxpy.gpx.GPXTrackSegment()
gpx_track.segments.append(gpx_segment)
```

After each GPS measurement, log the (lat,lon) data in the GPX file :

```
gpx_segment.points.append(gpxpy.gpx.GPXTrackPoint(lat, lon))
```

At the end of the program, save the GPX file :

```
with open("gps_data.gpx", "w") as f:
    f.write(gpx.to_xml())
```

Example of gps_data.gpx GPS log file

```
<?xml version="1.0" encoding="UTF-8"?>
<gpx xmlns="http://www.topografix.com/GPX/1/0" xsi:schemaLocation="http://www.topografix.com/GPX/1/0 http://www.topografix.com/GPX/1/0/gpx.xsd"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1.0" creator="gpx.py -- https://github.com/tkrajina/gpxpy">
<trk>
<trkseg>
<trkpt lat="48.199358333333336" lon="-3.0141483333333334"></trkpt>
<trkpt lat="48.19935666666667" lon="-3.0141483333333334"></trkpt>
<trkpt lat="48.19935666666667" lon="-3.01415"></trkpt>
<trkpt lat="48.199355" lon="-3.01415"></trkpt>
<trkpt lat="48.19935333333333" lon="-3.0141516666666672"></trkpt></trkseg></trk></gpx>
```

This file can be opened with Google Earth or Geoportail to display the GPS data on a map.

Acquiring data from the IMU (Inertial Measurement Unit)

The IMU is a 9 degrees of freedom sensor (accelerometer, gyroscope, magnetometer).

- it gives the acceleration in 3 directions (x,y,z) on 16 bits (can be transformed to m/s^2, but not necessary)
- it gives the angular speed in 3 directions (x,y,z) on 16 bits (can be transformed to rad/s, but not necessary)
- it gives the magnetic field in 3 directions (x,y,z) on 16 bits (can be transformed to microTesla, but not necessary)
- The IMU is a low cost IMU9V5 from Pololu `https://www.pololu.com/product/2738`
- magnetometer is in a LIS3MDL chip
- accelerometer + gyroscope are in a LSM6DS33 chip

Acquiring data from the IMU (Inertial Measurement Unit (cont.)

How to get the IMU data :

```
import sys
import os
import time
# access to the drivers
sys.path.append(os.path.join(os.path.dirname(__file__), 'drivers-ddboat-v2'))
import imu9_driver_v2 as imudrv
imu = imudrv.Imu9IO() # create an ARduino object
# get the magnetic raw data (raw  = not calibrated !)
xmag,ymag,zmag = imu.read_mag_raw()
# get the accelerometer date
xaccel,yaccel,zaccel = imu.read_accel_raw()
# get the gyroscope data
xgyro,ygyro,zgyro = imu.read_gyro_raw()
```

Calibrating the compass

We need to calibrate the compass to compensate the sensor errors (gain, offset), the possible wrong axis orientation and the static magnetic perturbations.

The raw magnetic data can be calibrated using the following formula :

$$\begin{bmatrix} x_{calib} \\ y_{calib} \\ z_{calib} \end{bmatrix} = A^{-1} \cdot \left(\begin{bmatrix} x_{mag} \\ y_{mag} \\ z_{mag} \end{bmatrix} + \mathbf{b} \right)$$

with matrix A and vector b defined by the calibration process (explained in tutorial of Prof. Luc Jaulin this morning).

Calibrating the compass (cont.)

On each DDBoat, a python code `calib.py` is available to calibrate the compass and get A and b using 4 poses :

- North pose : the DDBoat is oriented to the North
- South pose : the DDBoat (turned upside down) is oriented to the South
- West pose : the DDBoat is oriented to the West
- Up pose : the DDBoat is oriented vertically (towards the sky)

A beep code will tell you when to change the pose.

If you are courageous, you can, for each pose make the average of several measures to reduce the noise !

Powering the propellers

- Propellers are powered by 2 brushless motors controlled by an electronic speed controller (ESC).
- The ESC are connected to an Arduino MEGA 2560 micro controller board.
- Speed command goes from -255 (max reverse speed) to 255 (max forward speed)
- After DDBoat startup, wait for the melody to play 3 times before commands sent to the ESC become effective
- Example of in place rotation for 2 seconds :

```
import sys
import os
import time
# access to the drivers
sys.path.append(os.path.join(os.path.dirname(__file__), 'drivers-ddboat-v2'))
import arduino_driver_v2 as arddrv
ard = arddrv.ArduinoIO() # create an ARduino object
left_speed = 100
right_speed = -left_speed
ard.send_arduino_cmd_motor(left_speed,right_speed) # in place turn
time.sleep(2)
ard.send_arduino_cmd_motor(0,0) # stop
```