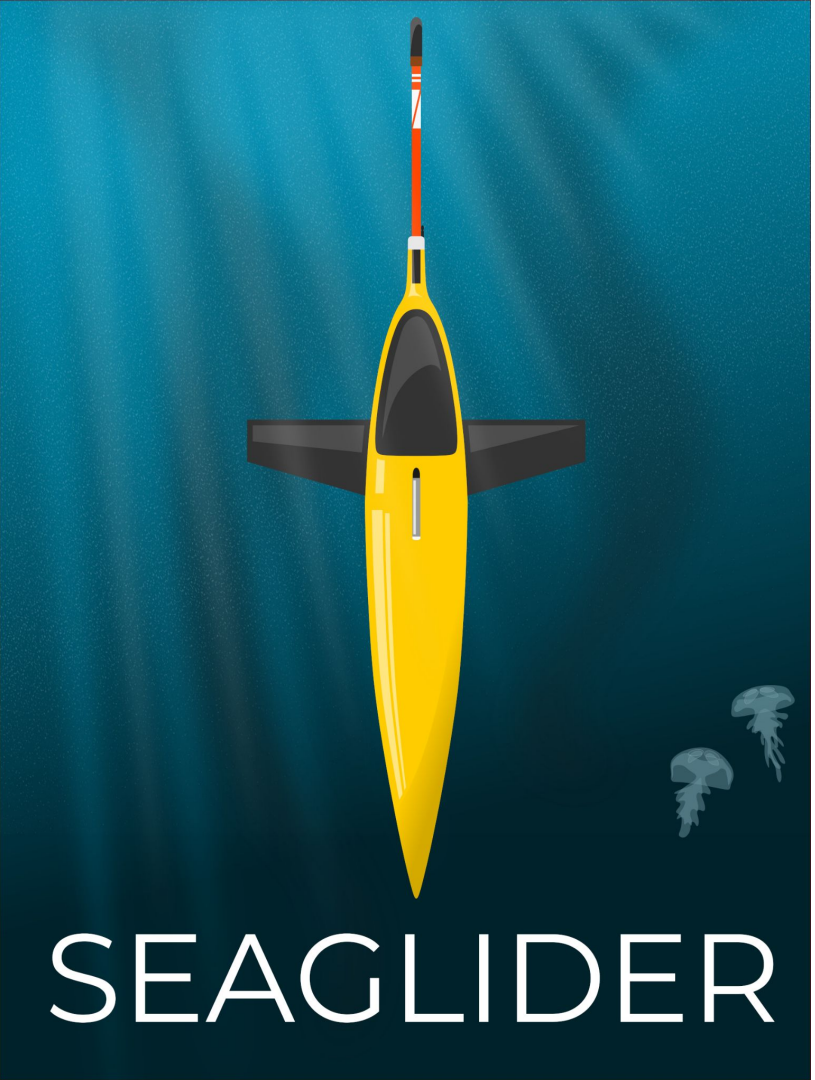

IOP office hours

Integrative Observational Platforms group (IOP)

iopsg@uw.edu

July 29, 2026

UNIVERSITY of WASHINGTON
Applied Physics Laboratory

What we are going cover/who asked us questions

1. Changes to installing basestation software (v3.0.9 and later)
2. Ballasting for large density ranges
3. *“Flight model - It is still unclear for me, not the model by itself but the implementation on the glider. How does the glider uses it to choose the pitch and buoyancy? How the pilot can adjust a,b pairs? Why does the basestation decide to recalculate the model? What is the trigger? Why does the model recalculation require ALL mission files, and why does it take a lot of time? How to do it manually?”*

1. Installing basestation software using UV

- The next two slides only apply if you setup or maintain an actual basestation server - we'll cover the reprocessing case in slide 5.
- As outlined in email send last week to `seaglider_community`, the requirements for the basestation software have changed. Starting with commit ead188c (pushed on 2026/07/28), python 3.14 and newer versions of numpy and scipy are required.
- The primary implication is that if you have been updating your basestation sources by `git pull origin master`, you may find yourself without a working basestation
  - seaglider.pub has not yet been updated to this version of code (probably the week of Aug 3. - this should be transparent to seaglider.pub users)
- The only tool set supported to install basestation source code is `uv` (https://docs.astral.sh/uv/)
  - It may be possible to use other tools to install python and the support libraries - you are on your own with that path

Installing basestation software using UV - cont.

Demonstration time

1. Starting with a docker container (basically, a lightweight virtual machine) that is has the sources loaded into the /usr/local/basestation3 and system prepped per the ReadMe.md
2. Install uv: `curl -LsSf https://astral.sh/uv/install.sh | sh`
3. Step through the rest of the Readme.md installation directions

Installing basestation software using UV for reprocessing

This is even easier:

1. Install UV (macOS and linux):

```
curl -LsSf https://astral.sh/uv/install.sh | sh
```

2. Grab a copy of the sources:
   a. `mkdir -p ~/tmp`
   b. `git clone https://github.com/iop-apl-uw/basestation3.git ~/tmp/basestation3`
   c. `cd ~/tmp/basestation3`
   d. `uv run Base.py --help`
   e. `uv run Reprocess.py --mission_dir <seaglider_home_directory> --force --reprocess_plots --backup_flight`

2. Ballasting for a large density range

Rule of thumb:

- For SG: a ~50 cc change in glider volume produces a 1 σ_θ density change
- For SGX: a ~70 cc change in glider volume produces a 1 σ_θ density change

Review of buoyancy and ballasting

The VBD contains ~800 cc's of moveable oil, i.e., available buoyancy (same for SG and SGX). To determine the density range (stratification) where your glider can operate:

- Start with the total buoyancy (cc's of oil) available
- Subtract the cc's of oil needed to lift the antenna out of the water (150 cc)
- Subtract the cc's of oil needed for negative buoyancy (thrust), usually specified at the deepest/densest water to be encountered).
- Divide the remaining cc's of oil by the rule of thumb (50 or 70 cc/σ_θ):

Total VBD oil available	800 cc
Positive buoyancy required to expose antenna	– 150 cc
Negative thrust in densest water	– 250 cc
VBD remaining	400 cc

400 ccs of buoyancy is equivalent to:
8 σ_θ units of stratification for SG
5.5 σ_θ units of stratification for SGX

Ballasting for a density range near the limit (e.g., 8 σ_θ units for SG, 5.5 σ_θ units for SGX)

- Pay attention to your vehicle type (SG vs SGX) and the density range you are operating in.
- Look closely at the expected potential density profile in the time and region of the deployment, including variability (not just the climatology)
- Err on the side of too much buoyancy - use the minimum thrust you can tolerate.
- In our experience, using a ballast tank to estimate volmax can have a ±100 cc uncertainty - probably not acceptable. It is more accurate to estimate volmax from sea-trial data (we require three dives to 150 m with decent flight for our ballasting).

3. Flight Model, Hydro Model, and all that stuff

What is the “hydro model”?

- The hydro (hydrodynamic) model is a mathematical model of the glider’s movement through still water that takes as input the glider’s buoyancy and observed pitch, rho and two tuning parameters - **hd_a** (lift), **hd_b** (drag) - and returns a forward velocity and angle of flight. It is an iterative solver and is not guaranteed to converge to produce a solution.
    - An important note: there is a third parameter, **hd_c**, that historically has been a variable but now is treated as a constant of 5.7e-06

Where are there implementations of it?

- Three locations: (a) on the glider, (b) in the basestation in **RegressVBD.py** and (c) in the basestation in **Hydro.py**
- (a) and (b) are very close cousins, (c) is a bit different in that it attempts to capture more of the dynamics of the glider’s flight.

Where is it used?

- (a) is used on-board the glider for helping with next dive planning and estimating depth-averaged current. All inputs are taken from the glider’s parameters. (Note - the full details of how the glider uses the hydro model to select buoyancy is a topic for a later session)
- (b) is used in the **vert_vel_new** plot and the VBD regression tool in vis as part of a muti-dive regression to estimate vbd bias, volmax, better **HD_A** and **HD_B**, and **C_VBD**. Input is from the glider’s parameters (**HD_A**, **HD_B**, etc.)
- (c) is used in the rest of the basestation3 - particularly in **TempSalinityVelocity.py** for correcting a Seaglider unpumped CTD, **MakeDiveProfiles.py** for calculating the glider’s velocity for displacement estimates and DAC calculation, and **FlightModel.py** when the Flight Model System (FMS) is running. For everything except the FMS, input depends on the state of the `--ignore_flight_model` flag (false by default):
    - If false and FMS output is available (more on the next slide), then use the FMS **HD_A**, **HD_B**, etc.
    - If true, use the values in `sg_calib_constants.m`

Flight Model, Hydro Model and all that stuff - cont.

What is the Flight Model System?

- The FMS is a system that is part of the basestation that estimates the HD_A, HD_B, vbdbias, volmax (and a number of other parameters) for each dive of the mission. (Recall this is different than a single set of values for an entire mission if in the `sg_calib_constants.m` file)
- These are housed in single file, `flight.pkl`, in the `flight` subdirectory of the mission directory, but some of the key values reside in the `fm_dddd.m` files in that directory.
- The FMS processes dives as they arrive. Depending on what the system observes in the characteristics of new dives, it may go back and “restate” (update) its previous estimates for these dives (and by default, regenerate the associated netCDF files for the dive)
- These estimates are by default used as input to the other science processing (**TempSalinityVelocity.py** and **MakeDiveProfiles.py**) where a forward velocity is generated from the **HydroModel.py**::hydro_model.

Flight Model, Hydro Model and all that stuff - cont.

That's all great, but what am I supposed to do?

- By default, basestation processing runs the FMS, produces estimates of HD_A, HD_B, and so on that it uses internally for science processing on the basestation. You should let it do that. (Note: by default, `--ignore_flight_model` is false and the basestation will ignore HD_A, HD_B, etc in the sg_calib_constants.m file)
- For tuning the glider, use the **vert_vel_new** plot and/or the VBD regression tool in vis to generate estimates of HD_A and HD_B to be set in the cmdfile.
- The FMS system (if running) will occasionally send out notifications about changing HD_A and HD_B - treat those as a heads up that things might be changing on the glider (the FMS is doing a lot of its own trend analysis): have a look at the **mission_volmax**, **FM_ab_dives**, **FM_vbdbias** plots to see if the overall vehicle status is changing, and use **vert_vel_new** to help with the actual changes if needed (via the cmdfile).