Frequently Asked Questions about the CUAUV Controller

Valid as of March 19, 2016

Contents

1 How can I tune a PID loop?

For this, I refer you to Jeff Heidel's Fall 2012 report on `controld3` found on SVN. See page 12.

2 Where is the thruster configuration?

The thrusters are configured in the vehicle configuration files found in `conf`. See `conf/thor.conf` for Thor's configuration file. Don't forget to rebuild after editing the file so the generated JSON file can be updated.

3 How can I add/remove a thruster?

To add a thruster, add a new thruster to the thruster list, as is done for the other thrusters in `conf/VEHICLE.conf`.

To remove a thruster, simply remove or comment out the line that adds it to the thrusters list in `conf/VEHICLE.conf`.

4 One of the thrusters on the vehicle is broken and I want to run without it. How can I tell the controller not to use it?

Remove the thruster from the controller as described above.

5 One of the thrusters on the vehicle is now in a different position. How can I tell the controller where it is?

There are two things needed: the position and the orientation.

The position is given as a 3 dimensional vector with the origin at the vehicle's center of mass. Positive x is the fore direction, Positive y is the starboard direction, and positive z is down. The position should be a vector to the center of the thruster's prop.

The orientation is inputted as two angles, a yaw (heading) in the range 0 degrees to 360 degrees and a pitch in the range -90 degrees to 90 degrees. Yaw and pitch are angles to rotate about the positive z and positive y axis respectively. The thruster initially starts with the prop facing the aft direction (propelling the sub forward with positive PWM) and yaw (heading) is applied before pitch.

Some examples:

An aft sway thruster with positive PWM pushing the vehicle starboad would have a position of roughly (-1, 0, 0) a yaw angle of 90 degrees, and a pitch angle of 0 degrees.

```
{ "name": "sway_aft", "pos": [-1, 0, 0], "heading_pitch": [90, 0] }
```

A fore port depth, pitch, and roll controlling thruster (positive PWM pushes the vehicle down):

```
{ "name": "fore_port", "pos": [1, -1, 0], "heading_pitch": [0, -90] }
```

6 What else can I configure about a thruster?

- **Thrust characterization curve**
  Currently unconfigurable since the switch to the `conf` system. This functionality coming soon...
  ~~The thrust characterization curve is used to convert between the PWM sent to the thruster and the thrust it produces. The dictionary `thruster_models` near the top of `control/thrusters.py` containts the filenames of the thrust curves for every thruster, mapped by name. The files are searched for in a directory given by the variable `MODEL_DIR`. The files are expected to be a in a format as outputted by the bollard pull test script.~~

- **Turn-on PWM value**
  Currently unconfigurable since the switch to the `conf` system. This functionality coming soon...
  ~~This is determined by the class type of the thruster, i.e. all SeaBotix are assumed to have the same minimum PWM. Alter if needed.~~

- **Efficiency**
  The `drag` variable can be set to a value between zero and one to indicate the percentage of thrust the thruster actually a delivers. For example, if a thruster is 90% blocked by a piece of foam, the drag should be set to 0.1. The term "drag" is very improper. The value is the ratio of thrust delivered to thrust delivered under normal operation.

7 How can I tell the controller where the center of buoyancy is?

The variable `center_of_buoyancy` in `conf/VEHICLE.conf` is a vector that points from the vehicle's center of mass to the center of buoyancy. For a perfectly trimmed vehicle, this should be (0, 0, -x), where x is the vertical distance from the center of buoyancy to the center of mass. The larger this value is, the more passive pitch and roll control the vehicle is expected to have.

In addition, there are variables that control the strength of the buoyancy force (the weight of the water displaced by the vehicle when fully submerged) and the strength of the gravitational force (the weight of the vehicle).

8 How can I tune the buoyancy force?

The buoyancy force is calculated in `control/vehicle.py`. Most of the proposed tuning method below relies heavily on trial and error and has not been tested.

First the `gravity_force` variable (representing the strength of the gravitational force) should be set to the vehicle's weight (in Newtons, everything is in metric units).

Afterwords, the strength of the buoyancy force can be determined by examining the behavior of the depth of the vehicle. There are many ways to do this; here is one:

Assuming the vehilce is positively buoyant, set the `buoyancy_force` variable to a value slightly higher than the vehicle's weight (`gravity_force`). To determine if that is the correct value, drive the vehicle to a depth below the surfce (e.g. 1 meter) and make sure it is stable. Then, either make note of the integral term on the depth PID loop, or turn off the depth PID loop (`<a> <o>` in control-helm) and make note of the vehicle's depth. If the integral term is positive, the buoyancy force is underestimated and you need to increase it accordingly. If the integral term is negative, the buoyancy force is overestimated and you need to decrease it. Similarly, if the vehicle rises toward the surface when the depth PID loop is turned off, the buoyancy force is underestimated. If the vehicle dives, the buoyancy force is overestimated.

Once the strength of the buoyancy force has been tuned, tuning the center of buoyancy is the only thing left. First, if the vehicle is properly trimmed (i.e. pitch and roll are near zero when stationary), the center of buoyancy should have x and y components of zero as described above.

If the vehicle is not trimmed properly, you can attempt to add x and y components to the center of buoyancy such that the vehicle is steady and flat in the water with the controller running. An example: the vehicle lost a weight in the back and is now passively pitched downward. The center of buoyancy has shifted backward (more so the center of mass shifted forwards but we'll ignore that...) and you should add a small negative x component to the center of buoyancy.

Tuning the z component of the center of buoyancy can be tricky. First, it should always be negative (this means the sub attempts to passively right itself if pitched or rolled). Second, it helps if the feedback loops responsible for orientation control have no intergral term. This is always true for the quaternion loop but the individual PID loops for heading, pitch, and roll, may have a non zero integral term. A zero intergral term allows you to more easily see the error due to mis-estimation of the buoyancy force.

Slowly increase the z component until you are satisfied with the pitch and roll response of the vehicle. The vehicle should be able to steadily maintain a desired pitch and roll of up to 45 degrees with a properly tuned center of buoyancy. Be careful not to overestimate the buoyancy force!

Telltale signs of an overestimated buoyancy force: the sub has a hard time returning to 0 pitch or 0 roll, the sub overshoots its desired pitch or roll. Likewise an underestimated buoyancy force will result in the sub undershooting its desired pitch or roll.

Clarification on OVERESTIMATED in this context: This means that the distance between the center of buoyancy and the center of mass is smaller in real life, i.e. the magnitude of the center of buoyancy vector is too large, i.e. you probably made the z component of the center of buoyancy vector too negative.

9 I want to use buoyancy force modeling for orientation control, but I don't want to use it for depth control. What can I do?

In `control/vehicle.py`, set `gravity_force` equal to the `buoyancy_force`. This tells the controller that the sub is neutrally buoyant and the calculated depth force due to buoyancy will always be zero. Induced torques due to buoyancy will still be non zero (assuming that the center of buoyancy is not the origin/center of mass).

10 I don't want the controller to model the buoyancy force at all!

Set `settings_control.buoyancy_forces` to zero. The controller will no longer compensate for forces and torques induced on the vehicle due to the buoyancy force, but that's no fun.

11 How can I compensate for the drag on the vehicle?

Coming soon...

12 I don't want the controller to model any drag forces!

Set `settings_control.drag_forces` to zero. The controller will no longer compensate for forces and torques induced on the vehicle due to drag.

13 I don't like the vehicle's control. What can I do?

The first step in trying to solve the problem is figuring out WHY there is a problem in the first place. To do this, one can examine the values of several helpful shared memory variables that relate to control.

14 Which shared memory variables are useful for debugging the controller?

Here we will list shared memory groups that provide useful information about what is going on in the controller.

- `control_internal_X` (`X` a DOF)
  - `out`: the force on the vehicle in the `X` degree of freedom as desired by the `X` PID loop. If `X` is a rotational degree of freedom, this value is only valid if the quaternion PID loop is NOT being used..
  - `out_limited`: the force on the vehicle desired from the optimizer / thrusters in the `X` degree of freedom. Essentially, `out_limited` = LIMIT(`out`−`control_passive_forces.X`).
- `control_passive_forces`
  This group is a wrench in sub space (body frame) that is the wrench determined to be acting on the vehicle from sources other than thrusters. This is calculated in `control/vehicle.py` and currently incorporates buoyancy and linear drag.
- `control_internal_opt_errors`
  This group is a wrench in sub space (body frame) that is the wrench difference between the desired wrench output requested by the PID loops and the actual (computed) twrench output produced by the thrusters. All of the values here should be 0 (or negligibly small) if every outputted thruster value is inside (-255, 255). A non zero value for a particular degree of freedom indicates that the optimizer is not faithfully fulfilling the requests of the PID loops. Two of the most common and probable reasons for this are: 1. The desired PID loop output is too large in magnitude to be realized by the vehicle's thrusters and 2. The optimizer is

sacrificing accuracy in one degree of freedom to fulfill accuracy in another (usually one that is demanding heavily).

Example of the first case: The vehicle is moving at about 0.5 m/s, and the x velocity PID loop is asking the vehicle to accelerate (positive PID outout) to reach a speed of 99.0 m/s. However, 0.5 m/s is about the top speed of the vehicle and, assuming drag forces are reasonably correct, the maximum output of the vehicle in the x direction will equal the drag force at about 0.5 m/s. One expects `control_internal_opt_errors.forward` to be very large in such a scenario.

Example of the second case: The vehicle is moving forward at some speed, and the user requests a 180 degree change in heading (U-turn). Say that the error in the forward direction is 0 initially (possible with a speed less than 0.5 m/s). During the U turn, one would expect the forward error to jump up, if our heading / orientation PID loop is agressive often, as is often the case. Adding priority to torques will further accentuate this effect (see question on priorities).

15 Which shared memory variables affect the operation of the controller?

Here we will list shared memory groups that the user can use to tweak the controller. We will not deal with the traditional PID loops.

- `settings_control`
  - `buoyancy_forces`
    This boolean variable indicates to the controller whether or not to use the buoyancy force in passive force calulation. Setting this and `drag_forces` both to zero has the effect of removing passive force modeling from the controller. Modeling buoyancy forces helps with maintaining a non zero pitch and roll and is rumored to help get rid of steady state roll and pitch oscillations.
  - `drag_forces`
    This boolean variable indicates to the controller whether or not to use the drag force in passive force calculation. Setting this and `buoyancy_forces` both to zero has the effect of removing passive force modeling from the controller. Modeling drag forces allows for anticipating torques induced by certain linear movements, provided that the correct drag planes are modeled (see question on drag planes).
  - `quat_pid`
    This boolean variable indicates to the controller whether or not to use a quaternion feedback loop for orientation control. Settings this to 0, reverts to using the traditional heading, pitch, and roll PID loops to control orientation. Pressing the (q) key in `auv-control-helm` will toggle this flag. Note that the implemented quaternion feedback loop is a PD controller, i.e. there is no integral action.
- `settings_quat`
  This group containts controller gains for the quaternion feedback loop. These are only used if `settings_control.quat_pid` is set to 1. Note that the `kI` variable has no effect as there is no integral action in the quaternion feedback loop. Currently, there is no way to edit these from control helm, but in practice I have had little reason to tune these.

Reasonable values for `kP`: 4.0 to 15.0
Reasonable values for `kD`: 0.5 to 3.0
Roughly, others may work better.

- `control_internal_priority`
  This group contains weights for the optimizer's objective function. These only have an effect when there are thrusters that are saturated, i.e. the PID loops are demanding more than the vehicle's thrusters can supply.

  IMPORTANT: The only variable that will have any effect is `torque`.