Day 3: Autonomy

Before you start, update your branch to `post-lecture3` which includes all the code changes I made during the lecture today:

```
cd goby3-course
git fetch
git checkout post-lecture3
```

Assignment 1:

Goal: Develop a simple "helm" application and use it to control the vehicle through `goby_frontseat_interface`.

Task:

There's a lot of work in autonomy these days and we would like to lower the barrier of entry to get a new autonomy algorithm or system onto real vehicles. Using the `goby_frontseat_interface`, we have a straightforward way to connect new autonomy systems to deployed vehicles.

In this task, you will replace the pHelmIvP-based autonomy system on the USV with a (simple) one of your own. We want to command the USV to:

- At a fixed speed of 1.5 m/s:
  - Follow a fixed course due east for a configurable duration of time.
  - Follow a fixed course due south for a configurable duration of time.
  - Follow a fixed course northwest and stop once the vehicle is no longer getting closer to its starting point.

The result will be (approximately, depending on the durations and currents) a triangle path.

A few suggestions for getting started:

- Create a new SingleThreadApplication, either using the `pattern` files or from scratch. Let's call it `goby3_course_helm`.
- Refer to the `goby_frontseat_interfaces` interface (`goby3/build/share/goby/interfaces/goby_frontseat_interface_interface.yml`):
  - Publish "HELM_DRIVE" to the `helm_state` group of `goby_frontseat_interface`, say at 1 Hz.
  - Publish to the `desired_course` group of `goby_frontseat_interface` for your setpoints, say at 1 Hz.
  - Read (subscribe to) the group `node_status` for the vehicle's current position.
- Disable the IvPHelm code.
  - Comment out the launch of pHelmIvP in `usv.launch`.
- Add the `goby3_course_helm` to the `usv.launch` file, add a template for its configuration, and enable the configuration generation in `usv.pb.cfy.py`.

You can test the `goby3_course_helm` by running the topside and USV (or the whole mission with `all.launch`) and visualize it on Google Earth or OpenCPN.

Assignment 2:

Goal: Once your new autonomy engine (Assignment 1) reaches the end of its mission, create a RECOVER command to send to the AUVs. Upon receipt, the AUVs will enter a RECOVER state and come to the surface.

Task:

Once our USV completes the third (northwest) leg of its mission and stops, we want to send a command to the AUVs to RECOVER (drive to the surface and aggregate at a single point).

Your task is to:

- Create a new DCCL message for the AUV command (You can use dccl id 127).
- Publish the AUV command on the USV.
- Modify the IvPHelmTranslation plugin (`src/lib/moos_gateway`) for `goby_moos_gateway` to subscribe for the AUV command and publish an appropriate MOOS variable for recovery.
- Update the AUV behavior file (`launch/trail/config/templates/auv.bhv.in`) to change behavior and recover when the command is received. You can use the StationKeep behavior (https://oceanai.mit.edu/ivpman/pmwiki/pmwiki.php?n=Helm.BehaviorStationKeep) along with a ConstantDepth behavior of 0 m (you can either send an update to the existing `deploy_depth` behavior or create a new one that activates only on recovery).

Make sure you test this with the complete `./all.launch` mission.

Wrap up

Excellent work - now we've shown that we can develop and deploy our own autonomy behaviors and use the `goby_frontseat_interface` to talk to the actual vehicle. At this point you'd be ready to put it in the water with any vehicle for which you or someone else has written a driver for (currently Iver3, GD Bluefin, and Waveglider SV2 vehicles).

In addition, we've developed a second command for multivehicle autonomy, where one vehicle reaching a mission goal (the USV reaching its end of mission) can trigger a change in autonomous behavior for a fleet of other vehicles (the AUVs going into recovery).

Solutions (Toby)

My solutions are pushed to the `post-homework3` branch of goby3-course. Please reference the code together with this text.

Assignment 1:

Created `src/bin/helm` from the `single_thread` pattern files. Added it to the `src/bin/CMakeLists.txt`. Changed the class name in `app.cpp`, `config.proto`, and binary name in `CMakeLists.txt`.

Added configuration values for east and south durations in `config.proto`, as well as speed as an optional value defaulted to 1.5 m/s.

Run loop() at 1 Hz.

Created enumerations for each leg and an `update_leg()` method that checks how long we've been on each leg, and if we've exceeded the configured duration, switches to the next leg. I left the northwest leg for now, and went to test the first two.

In loop(),

- `update_leg()`
- Send HELM_DRIVE to `helm_state`
- Send DesiredCourse to `desired_course`

Added my configuration template to `launch/trail/config/templates/goby3_course_helm.pb.cfg.in`, added a block to `launch/trail/config/usv.pb.cfg.py`, and addded a launch line to `usv.launch`. Commented out the MOOS code from `usv.launch`.

Now I ran it:

```
cd goby3-course/launch/trail
./topside.launch
goby3_course_n_auvs=0 ./usv.launch
```

Once I got that working, I added member variables to keep track of the start position, the latest position (both as `goby::middleware::frontseat::protobuf::NodeStatus`), and the closest distance to the start we've seen thus far (`min_dr_`).

Then, I subscribed for `goby::middleware::frontseat::groups::node_status` to store the starting position of the vehicle, and keep tracking of the current position. Using this information, I calculate the range to the start (`dr`) while we're on the NORTHWEST leg. When this starts increasing, I put the USV into recovery mode. I had to filter the `dr` to keep the vehicle from entering recovery as it turns the last corner.

Assignment 2:

To the existing `src/lib/messages/command_dccl.proto`, I added an `AUVCommand` message.

```
>dccl -a -f command_dccl.proto -m goby3_course.dccl.AUVCommand
||||||| Dynamic Compact Control Language (DCCL) Codec |||||||
1 messages loaded.
Field sizes are in bits unless otherwise noted.
============= 127: goby3_course.dccl.AUVCommand =============
Actual maximum size of message: 4 bytes / 32 bits
        dccl.id head.....................8
        user head.....................0
        body.....................18
        padding to full byte.....................6
Allowed maximum size of message: 32 bytes / 256 bits
--------------------------- Header ---------------------------
dccl.id head.....................8 {dccl.default.id}
---------------------------- Body ----------------------------
goby3_course.dccl.AUVCommand.....................18 {dccl.default3}
        1. time.....................17 {dccl.time2}
        2. desired_state.....................1 {dccl.default3}
```

Then I update the behavior file (`launch/trail/config/templates/auv.bhv.in`) to add a StationKeep behavior for the recovery. I also added an updates field for the `deploy_depth` behavior so I can set the depth to 0. Finally, I changed `DEPLOY` into `AUV_DEPLOY_STATE`, which we'll set to "TRAIL" or "RECOVER".

Then, I added a new group `goby3_course::groups::auv_command`, which I will use to publish the command on the USV and subscribe to it on the AUVs.

In the `src/bin/helm/app.cpp` I publish the recover command upon completing the mission. In the `src/bin/manager/auv/app.cpp`, I added a `subscribe_commands()` method where I do a subscription to the `auv_command` that is nearly identical to the equivalent method in the USV manager.

From there, I added the command handling to the IvPHelmTranslation (`src/lib/moos_gateway`).