User manual

Search and Rescue - Underwater

Version 1.0

Author: Alexander Roser
David Andersson
Hampus Frick
Isac Lundin
Ken Dahl
Oscar Holm
Oskar Philipsson

Date: December 19, 2022

Status

Reviewed	-	-
Approved	-	-

Project Identity

Group E-mail:	searchandrescueunderwater2022@gmail.com
Homepage:	http://www.grphomepage.se
Orderer:	Gustav Zetterqvist, ISY
	E-mail: gustav.zetterqvist@liu.se
Customer:	Andreas Gällström, Saab Dynamics
	E-mail: andreas.gallstrom@saabgroup.com
Customer:	Jonatan Olofsson, Saab Dynamics
	E-mail: jonatan.olofsson@saabgroup.com
Supervisor:	Daniel Bossér, ISY
	E-mail: daniel.bosser@liu.se
Supervisor:	Philip Andersson, Saab Dynamics
	E-mail: philip.e.andersson@saabgroup.com
Supervisor:	Erik Söderberg, Saab Dynamics
	E-mail: erik.soderberg@saabgroup.com

Group Members

Name	**Responsibility**	**E-mail**
Alexander Roser (AR)	Head of testing	alero223@student.liu.se
David Andersson (DA)	Head of documentation	davan957@student.liu.se
Hampus Frick (HF)	Project leader	hamfr643@student.liu.se
Isac Lundin (IL)	Head of hardware	isalu162@student.liu.se
Ken Dahl (KD)	Head of software	kenda091@student.liu.se
Oscar Holm (OH)	Head of information	oscho082@student.liu.se
Oskar Philipsson (OP)	Head of design	oskph717@student.liu.se

Document History

Version	Date	Changes made	Sign	Reviewer
0.1	2022-12-10	First draft.	DA, IL, OH, OP, AR, HF, KD	DA
0.2	2022-12-16	Second draft.	DA, IL, OH, OP, AR, HF, KD	DA
1.0	2022-12-19	First version.	DA, IL, OH, OP, AR, HF, KD	DA

Contents

1 Introduction

This document is the user manual for the project Search and Rescue Underwater in the course TSRT10, Reglerteknisk projektkurs, CDIO, given at Linköping University.

1.1 Definitions

- UAS - Unmanned aircraft system, i.e., drone.
- GUI - Graphical user interface.
- ROS2 - Robot operating system.
- Session - The time span from which the UAS is turned on until the UAS is turned off.
- Mission - Contains all tasks produced by the task planner. Multiple missions can be executed in one session.
- UUV - Unmanned underwater vehicle.
- Emergency pinger - Transmitter of the emergency signal relayed by the UUV.
- Semi-autonomous mode - A mode where the user is coordinating where the UAS should go.

2 System overview

The system is using a UAS with a hydrophone in order to collect measurements in the water in order to find an UUV with an emergency pinger.

Because of some unresolved issues regarding the hardware, the usage of the simulation environment and the real hardware are slightly different. Once the hardware issues are resolved, the instructions for the real hardware will be much more similar to the simulation environment.

2.1 Software

The software is architectured according to Figure 1. Each small rectangle represents a ROS node. They are organized in two different docker containers. One for the UAS and one for the base station. Software in the loop testing is used, so the real hardware can be replaced by simulated parts. One program is acting as the drone simulator and one ROS node is acting as the pinger simulator.

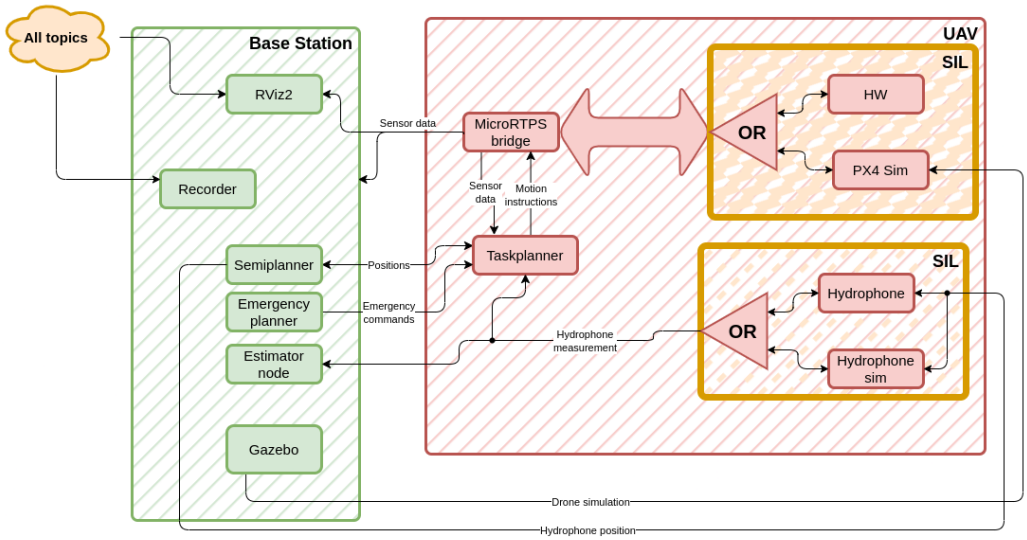

Figure 1: Software architecture overview

The nodes choreograph a mission in the following way. Planner nodes send what coordinate the drone should move to in order to dip the hydrophone. The planner asks the sound simulator node to simulate measurements. Then the measurements are sent to the estimator node, which processes them and comes up with the estimated position. Then the drone moves to the next position and so on. When done, the drone lands and the planner orders the estimator to plot the likelihood function of the position estimate.

2.2 Real data collection

For the real data collection, QGroundControl is used for controlling the drone instead of ROS. A real sound card is used instead of a simulated one. The sound card is set to record continuously and later the measurement positions are mapped to the correct time in the recording. The estimation is done manually through scripts instead of automatically by a ROS node.

3 Simulations

This section will outline how to operate the simulation environment and the important ROS nodes.

3.1 Install and setup

First, you will need to install Linux on your machine. Using a virtual machine is not recommended because of bad virtual machine performance in Windows. Ubuntu is the recommended distribution, but most should work since everything is done inside docker containers (Fedora has been tested to work without any major problems). Dual boot on the same hard disk as Windows or install Linux on a separate USB drive is recommended (Ask SAAB to buy immediately if they have not already done that).

Follow the instructions in the README in the base of the repo in order to clone the repo and sub-repos correctly, and install docker according to the instructions at `doc/startup_guides/docker_startup.md`. Continue following the instructions in the README to build the docker containers. While you wait for them to build, read ALL documentation in the `doc` folder.

When you have successfully followed the README, you should have the simulator Gazebo open and a drone should have spawned close to the origin.

3.2 Usage

This section will explain how to start an entire mission in the simulation and how to individually control the important ROS nodes

3.2.1 Entire mission

Make sure you have successfully completed the setup. You do not need to rebuild the docker containers, but you need at least two terminals in each container. One for building the ROS packages and one for running commands. When Gazebo is up and running and the drone is visible, you can start the programmed mission by starting a new terminal in the `base_station` container by running

```
docker exec -it base_station bash
```

then in the new docker terminal run the following command

```
ros2 launch planner planner.launch.py
```

which should result in the drone taking off, dipping the hydrophone at the predefined points, taking measurements, sending to the estimator node, landing when done, printing the estimates in the terminal after each dipping position and plotting the likelihood functions after the final dipping position.

The set of predefined points can be changed by editing the variable `self.coordinates` in the **SemiPlanner** class in the file `src/planner/planner/semiplanner.py`.

3.2.2 Sound simulation node

To only run the sound simulation node, Gazebo does not need to be running. Run the following command in the uav_sim container to start the node

```
ros2 run hydrophone_sim hydrophone_talker
```

To simulate a sound, run the following command to publish the hydrophone position to the `/sound_rec/start` topic.

```
ros2 topic pub --once /sound_rec/start geometry_msgs/msg/Pose \
"{position: {x: 1, y: 2, z: 3}, orientation: {x: 0, y: 0, z: 0, w: 1}}"
```

3.2.3 Estimation node

Starting the estimator node, Gazebo is not needed. It listens on the same topic that the sound simulator node publishes to.

```
ros2 run estimator likleyhood_estimator
```

To plot the estimator's likelihood, use this command

```
ros2 topic pub --once /estimator/plot_likleyhood std_msgs/msg/Bool "{data: true}"
```

3.2.4 Planner nodes

To start the main collection of planner nodes run the following command. This requires Gazebo and the PX4 simulator to be running.

```
ros2 launch planner planner.launch.py
```

When only working with sound, simulating the drone flying takes unnecessary time and CPU. To only publish measurement points to the sound simulator, the calibration planner can be run using this command.

```
ros2 run planner calibration_planner
```

3.2.5 Controlling the drone

This section explains the ROS topics used to communicate with the drone. This requires Gazebo to be running.

Use this command to arm the drone.

```
ros2 topic pub --once /uav_arm_command std_msgs/msg/Bool "{data: true}"
```

Use this command to make the drone move to the wanted position.

```
ros2 topic pub --once /uav_trajectory_setpoint_pose geometry_msgs/msg/Pose \
"{position: {x: 1, y: 2, z: 3}, orientation: {x: 0, y: 0, z: 0, w: 1}}"
```

3.3 Customizing external software

This section will specify how to make changes to the external software, i.e., Gazebo and PX4.

3.3.1 Adding and changing simulation world

A submodule for the simulation worlds is located in `sbd_uw/src/sbd-sim`. In the branch `SAR-UW`, the currently used simulation world can be found and modified in `worlds/waterandland.world`.

To add a new simulation world, simply create a new `.world` file in `sbd_uw/src/sbd-sim/worlds/`. To change the default world to simulate in, change the default value for the `WORLD` variable in `sbd_uw/Makefile` to the new file name.

3.3.2 Adding and changing UAS model

The simulation environment contains four different models of the UAS, each with different number of hydrophone segments, and are located in `sbd_uw/uav/models`. To specify which model to use, add `HYDRO_SEG=<n_segments>` when running `make uav_sim` in the `uav_sim container` (see README), where `n_segments` can be 0, 1, 2, or 9. For example,

```
make uav_sim HYDRO_SEG=9
```

To modify a model, it is recommended to spawn and modify it in Gazebo. Save a copy of the model to the `src/` folder since it is synced with your local `src/` folder through a Docker volume. The generated code in the saved file can then replace the contents of the corresponding `.sdf.jinja` file in `sbd_uw/uav/models`.

Note: changing or adding a model will require rebuilding the `sim_model` dockerfile. This is because the model files are added to Gazebo in the dockerfile, which the image of is then built. A better approach would be to replace the `.sdf.jinja` files with simple `.sdf` files and add a Docker volume to keep the model files in the Docker container synced with the local files. That way, rebuilding `sim_model` would not be necessary each time a change is made.

3.3.3 Customizing the PX4 code

A submodule containing all code for the PX4 is located in `/sbd_uw/src/uav/PX4-Autopilot`. To make changes to a file, create a copy of the file and place it in `/sbd_uw/`. Then, overwrite the existing file with your own in `/sbd_uw/docker/sim_model.dockerfile`. Some files have already been replaced, see `sim_model.dockerfile` for examples.

Note: Docker volumes could be used here too. However, when modifying a file, the last row in `sim_model.dockerfile`,

```
RUN DONT_RUN=1 make px4_sitl_rtps gazebo
```

has to be run for generating executable files either way, so there is really no point.

4 Gathering real measurements

In theory, the method for simulation and reality should be the same. In reality, however, this is not the case.

To perform a mission in reality, *QGroundControl* QGC [1](Available for both Windows and Linux) is used instead of the ROS2 program, to control the UAS.

There are two ways to connect the computer to the PX4; either with a USB-C cable or wireless with telemetry. In the second case, connect the telemetry radio to the computer with USB and make sure the PX4 is powered. No extra software is needed, except QGC. The following sections will describe how to perform a mission in reality in order to gather measurements.

4.1 Remote Controller

During flight, the remote controller should always be in hands of a remote pilot. The remote controller has two joysticks and three active switches. Figure 2 shows the remote controller from atop. The left joystick controls throttle (up and down) and yaw (left and right). The right joystick controls pitch (up and down) and roll (left and right).

Figure 2: The remote controller from above. The left and right joystick are used to maneuver the UAS

Three switches are used to arm and disarm the UAS, switch flight mode and kill the UAS if needed. Figure 3 shows the three switches. To arm the UAV, move the relevant switch to the bottom position in the figure. Do the opposite to disarm the UAV. By moving the kill switch, named SB, towards the screen, the UAV's motors are killed. The mode switch

is used to change between manual- and altitude mode [3]. The difference is that altitude mode will keep the UAS at the same altitude if the throttle is in a neutral position, which is the middle position of the left joystick. Moving the joystick to the upmost and down most position will make the UAS to slowly climb and descend respectively. For manual mode, the same input is equivalent with full throttle and no throttle, respectively.

Figure 3: The remote controllers switches. The switches are labeled *ARM*, *KILL* and *Mode*

It is possible to visualize the remote control output in QGC under ”radio”.

4.2 Set up a mission in QGC

The set up of a mission is done before the startup procedure of the UAS. The first step is to choose a location where everything will be proceeded from in QGC. First we need to define where the mission should start “Mission start” and some standard settings for the UAS such as flight speed and altitude, these values will be used during the mission if nothing else is stated. Then the first command is “Takeoff” to make the UAS fly. After that you should choose some “Waypoints” where the UAS should fly to, you can choose as many as you like. When you want the UAS to land, you could either use “Land” to define exactly where you want to land or “Return to home” which makes the UAS land at the same point it started. After you have set up a full mission you need to upload it to the drone. Also a good thing to check after the setup is the safety parameters for UAS that can be found in the settings tab in QGC [2]. Some parameters are:

- Geo-fence
- low battery level-actions
- lost connection-actions
- landing procedure

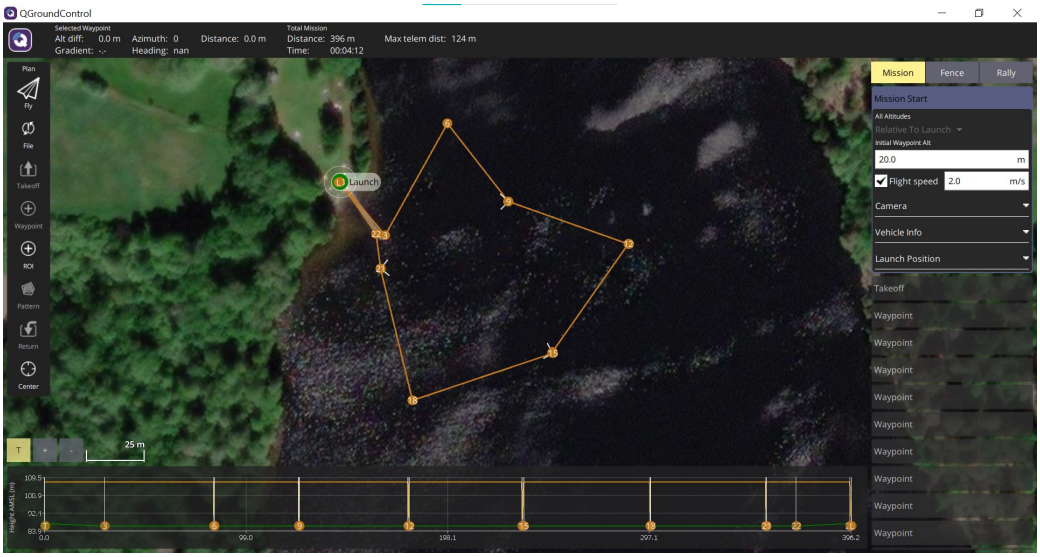

Figure 4: An example picture of a mission that is fully set up in QGC.

4.3 Prepare the UAS

To make the UAS ready for a mission, start by attaching the Battery, without connecting it. Connect the hydrophone cable to channel one on the sound card. Mount the end of the hydrophone cable under the battery in such a way that the stress on the sound card output is reduced.

Mount the propellers and make sure they are not loose. Lastly, connect the battery to the UAS power input, located on the bottom of the body. Make sure that the remote controller is on before the battery is connected.

4.4 Start the mission

To start a mission, first you need to power the sound card on the UAS and start recording by sliding the recording switch to the record position, which can be shown in Figure 5. A red light should then be visible at the switch. The mission itself is started by sliding a bar in QGC, after that, no more input is needed until the UAS has finished the mission and landed again. During the mission, some things need to be kept tract of. First is the UAS and which coordinates it flies to, secondly is how long time in the mission the measurements are executed. To be able to match the data from the sound card to the coordinates.

4.5 After the mission

When the mission is done, the sound card should still be turned on and indicate that it is recording. Turn off the recording by moving the switch the same way as before. Then enter the menu on the sound card and enable data transfer [4]. Connect the sound card to a computer through a USB-cable and download the recording on *.wav* format.

Figure 5: Image of the sound card, showing the location of the record switch.

4.6 Signal processing

The collected measurements are then paired with their corresponding coordinates. To save time, a suggestion is to manually keep track of the time when the hydrophone has been submerged.

The sound card records the measurements as *WAV* files. These can be imported in the script *main.m* located in the git folder *sbd_uw/Signal Estimation/Signal procesing of gathered data/*. The main file is then used with the file *get_meas.m* to retrieve the strength of the signal, explained in the section 7.1 in *Technical Documentation*. The parameters P^0 and β are then calibrated, explained in section 7.2 in the *Technical Documentation*. This is done in the file *estimate_P0_Beta_meas*, done through *main.m*. The parameters P^0 and β with the power and location of each measurement are finally implemented in *estimator.py* to estimate the pingers location with the maximum likelihood function.

One can also estimate the location of the pinger by minimizing the loss function with the Gauss-Newton algorithm in 2D, as explained in section 7.3 in the *Technical Documentation*. In that case, run the script *estimate_pos.m* which is located in the git folder *sbd_uw/Signal Estimation/Signal processing of gathered data/Gauss_Newton_2d*. Note, the parameters P^0 and β first needs to be calibrated as explained above.

References

[1] Qgroundcontrol installation, . URL https://docs.qgroundcontrol.com/master/en/getting_started/download_and_install.html. Accessed: 22-12-14.

[2] Qgroundcontrol safety, . URL https://docs.px4.io/main/en/config/safety.html. Accessed: 22-12-14.

[3] Px4 flight modes overview. URL https://docs.px4.io/main/en/getting_started/flight_modes.html. Accessed: 22-12-02.

[4] F3. URL https://zoomcorp.com/en/de/handheld-video-recorders/field-recorders/f3/f3-support/. Accessed: 22-12-07.