User Manual
Search and Rescue - Land

Version 1.0

Author: Rickard Wretlind
Date: December 5, 2022

Status

Reviewed	Jakob Åslund	2022-12-05
Approved	Jakob Åslund	2022-12-05

Course name:	Reglerteknisk projektkurs, CDIO	E-mail:	oplandlubber@gmail.com
Project group:	OWL	Document responsible:	Rickard Wretlind
Course code:	TSRT10	Author's E-mail:	ricwr413@student.liu.se
Project:	Search and Rescue - Land	Document name:	User_Manual.pdf

Project Identity

Group E-mail:	oplandlubber@gmail.com
Homepage:	https://tsrt10.gitlab-pages.liu.se/2022/sbd/
Orderer:	Jakob Åslund, Linköping University
	E-mail: jakob.aslund@liu.se
Customer:	Torbjörn Crona, Saab Dynamics
	E-mail: torbjorn.crona@saabgroup.com
Course Responsible:	Daniel Axehill, Linköping University
	E-mail: daniel.axehill@liu.se
Advisors:	Anja Hellander, Linköping University
	E-mail: anja.hellander@liu.se
	Linus Wiik, Saab Dynamics
	E-mail: linus.wiik@saabgroup.com
	Joel Wikner, Saab Dynamics
	E-mail: joel.wikner@saabgroup.com
	Åke Johansson, Saab Dynamics
	E-mail: ake.johansson1@saabgroup.com

Group Members

Initial	Name	Responsibility	E-mail (@student.liu.se)
DG	Daniel Goderik	Project Manager	dango893
AL	Anton Larsson	Head of Design	antla594
DS	Daniel Sandvall	Head of Hardware - Drone	dansa201
SF	Sebastian Fagerst-edt	Head of Testing	sebfa953
ES	Eric Sevonius	Head of Software	erise263
RW	Rickard Wretlind	Head of Documentation	ricwr413
AW	Albin Westlund	Head of Hardware - Rover	albwe662

Document History

Version	Date	Changes made	Sign	Reviewer
0.1	2022-11-28	First draft	DG	Jakob Åslund
0.2	2022-11-30	Second draft	DG	Jakob Åslund
0.3	2022-12-05	Third draft	ES, AL, SF	Jakob Åslund
0.4	2022-12-05	Fourth draft	DS, SF	Jakob Åslund
1.0	2022-12-05	First version	RW	Jakob Åslund

Contents

1 Introduction **1**

1.1 System Description . . . 1

1.2 Definitions . . . 1

2 Installation **3**

2.1 Docker . . . 3

2.2 Raspberry Pi Configuration . . . 3

2.3 Using Docker Images and Containers . . . 3

2.4 Base Station . . . 4

2.5 Rover . . . 5

2.5.1 Docker Setup . . . 5

2.5.2 Raspberry Pi Configuration . . . 5

2.6 UAV . . . 5

2.6.1 Docker Setup . . . 5

2.6.2 Raspberry Pi Configuration . . . 5

3 Qualisys **6**

3.1 Defining bodies . . . 6

4 Base Station **7**

4.1 Mission Setup . . . 7

4.1.1 Hardware . . . 7

4.1.2 Simulation . . . 7

4.1.3 No-fly Zones . . . 8

4.2 Spawning Distressed Persons . . . 8

4.2.1 Controlling Distressed Persons . . . 9

4.3 Starting the mission . . . 10

4.4 Restarting mission . . . 10

5 Rover **11**

5.1 Hardware . . . 11

5.1.1 Powering and Charging . . . 11

5.1.2 RC Controller . . . 11

5.1.3 Usage . . . 11

5.2 Simulation . . . 12

5.2.1 Usage . . . 12

6 UAV **13**

6.1 Hardware . . . 13

6.1.1 Powering and Charging . . . 13

6.1.2 RC Controller . . . 13

6.1.3 Propellers . . . 15

6.1.4 Usage . . . 15

6.2 Simulation . . . 16

6.2.1 Usage . . . 16

1 Introduction

This document is the user manual for the *Search and Rescue project* conducted in the course *Reglerteknisk Projektkurs - TSRT10* at Linköping University during autumn 2022.

The document contains information on how an operator should install and interact with required software, as well as how to interact with the hardware. In order to give the reader insight and some basic knowledge, it also contains information of the system and how it is constructed.

1.1 System Description

The entire system is run inside multiple *Docker containers*[2], which in of themselves contain the required software for their specific tasks. The project is built in three different subsystems: Base Station, Rover and UAV. Each subsystem has two corresponding containers, one for hardware and one for simulation. The user will always specify which container should be run, as it contains the software needed for its intended purpose. The systems also utilize software in the loop, which means that the same functionality works on both hardware and software.

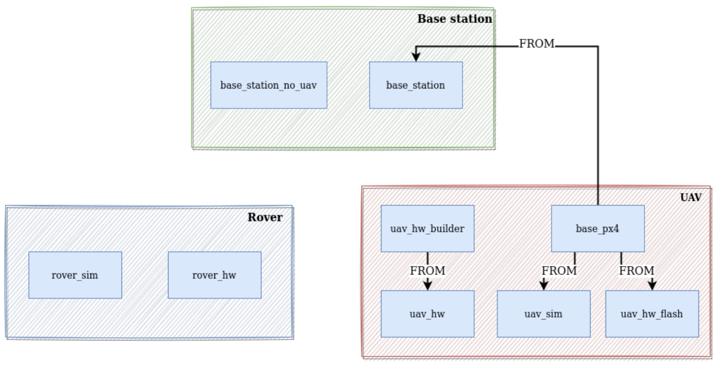

Figure 1: Overview of the structure for the containers

The UAV is equipped with an onboard flight controller (Pixhawk 4) which is connected to an RPi, and a camera used for image processing. There is also an RC-controller connected to the UAV for optional manual control of the UAV

The Rover is equipped with an RPi, an Arduino for actuator control, a Lidar for mapping and localization, a camera for image processing as well as an IMU (which is not used).

The Base Station is able to control the entire mission as well as displaying a simulation environment and Rviz2-window. Rviz2 visualizes the mapping and localization for the Rover, the planned paths from the motion planner, No-fly Zones for the UAV as well as the image feed from the vehicles' cameras.

1.2 Definitions

Below some definitions and acronyms are explained which are recurring in this document.

- **Rover** - An unmanned tracked vehicle.
- **UAV** - An unmanned aerial vehicle.
- **Agent** - A participant in a mission, Rover and/or UAV.
- **Base Station** - A computer that handles the information from the Rover and UAV.
- **Distressed person** - In simulation, this is a virtual marker that should be found by the Rover and UAV. When doing real tests, this will be RC-cars colored with bright colors.
- **SLAM** - Simultaneous Localization and Mapping.
- **LIDAR** - Light Detection and Ranging.
- **SIL** - Software In The Loop.
- **Qualisys** - Sensor system in the room Visionen that uses cameras and reflective targets to deliver position data.
- **ROS2** - "Robot Operating System", Framework for robot software development.
- **No-fly zone** - A zone where the UAV is restricted from flying into.
- **PDDL** - Planning Domain Definition Language.
- **RPi** - Raspberry Pi.
- **Pixhawk** - The flight controller *Pixhawk 4* that is mounted on the UAV.
- **HW** - Hardware.
- **SW** - Software.
- **Rviz2** - A visualization manager that displays the generated map and agent positions during the mission.
- **Gazebo** - Simulation environment.
- **RC-car** - Small cars controlled by the user that are used to simulate distressed persons.
- **$** - Denotes the start of a shell command, that is, an input command to a terminal. It should not be included when typing a command in the terminal.

2 Installation

This section describes the necessary installations that have to be done before running the programs. The first thing that must be done is cloning the Git repository. This is done by typing the following in a terminal

```
$ git clone --recurse-sub <project_ssh_path>
```

2.1 Docker

The system is heavily reliant on Docker, and installing this is therefore an important step before running the system. Installing Docker and Docker-Compose is done with the following commands in a terminal.

```
$ curl -fsSL https://get.docker.com -o get-docker.sh
$ sudo sh get-docker.sh
$ sudo groupadd docker
$ sudo gpasswd -a $USER docker
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt install docker-compose
$ sudo usermod -aG docker $USER
$ sudo service docker restart
```

2.2 Raspberry Pi Configuration

- Install Raspberry Pi OS (64-bit), preferably Debian Bullseye or later.
- Enable SSH with a static IP when connected to the Wi-Fi in Visionen [1].
- Configure the camera
  - Run **sudo raspi-config**
  - Select Interface Options and Legacy Camera
  - Select YES
  - Reboot
- Clone the git repository sbd and initiate submodules.

  ```
  $ git clone --recurse-sub <project_ssh_path>
  ```
- Install docker and docker-compose according to Section 2.1.

2.3 Using Docker Images and Containers

This section describes how to build docker images, create docker containers, start docker containers, and source the ROS2 workspace in containers. To create new images and containers, Docker-Compose is used. In the docker-compose file (available at the top level of the SBD git-repo), the different entities used during a mission (the Base Station, Rover and UAV) are defined as services. Services in docker-compose represent a named configuration of a docker container; which files to link as volumes, which ports to make available, environment variables and more. Broadly, there are two services for each entity: one for running in simulation, and one to running on hardware.

To build a new image for a service, run the command

```
$ docker-compose build <service_name>
```

Where "< *service_name* >" should be replaced by the name of the service. To create a new container for a service, run the command

```
$ docker–compose run <service_name>
```

This will also start the container. To start an existing container, run the command

```
$ docker start –i <container_name>
```

Where "< *container_name* >" should be replaced by the name of the container. To get the name of all created containers, run the command

```
$ docker ps –a
```

To compile the ROS2 workspace in a container, run the command

```
$ make build
```

When running the system, it will be required to have multiple terminals connected to the containers. To run the ROS2-code the terminals need to source the ROS2-worskpace, which can only be done after the code has been compiled with make. To open an already running container in a new terminal run

```
$ docker exec –it <container_name> bash
```

Then to source the ROS2 workspace in the terminal, run

```
$ . install/setup.bash
```

Warning: do not source and run any ROS2-code in the same workspace as you built the workspace with make, this can lead to unexpected errors and is an inherent part of the ROS2 build-tool colcon.

To launch any applications with a graphical interface (including the simulation) in a container, the xhost-server needs to be configured to be accessed inside. To do so, open a terminal on your system and run

```
$ sudo xhost +
```

This change will hold until you restart your computer, and therefore needs to re-run every time the computer has been turned off.

2.4 Base Station

The installation for the base station is a matter of building the required docker images. There is a base_image (shared with the UAV in simulation) which needs to be built before building the images for the base. To build and run the required images, follow Section 2.3 with the service and container names declared in Table 1.

	Hardware	**Simulation**
Base service name	base_px4	base_px4
Service/Container name	base_station_host	base_station

Table 1: Service- and container names for the Base Station

Instead of creating a new container each time, one can start existing containers in the same way as described in section 2.3.

2.5 Rover

This section describes the required installations for the Rover. This includes building docker images and Raspberry Pi configuration.

2.5.1 Docker Setup

There are different docker images, that have to be built, depending on if the Rover will be run on HW or in simulation. Follow Section 2.3 with the service and container names declared in Table 2

	Hardware	**Simulation**
Service/Container name	rover_hw	rover_sim

Table 2: Service- and container names for the Rover

Instead of creating a new container each time, one can start existing containers in the same way as described in section 2.3.

2.5.2 Raspberry Pi Configuration

Complete all steps in Section 2.2.

2.6 UAV

This section describes the required installations for the UAV. This includes building docker images and Raspberry Pi configuration.

2.6.1 Docker Setup

There are different docker images, that have to be built, depending on if the UAV will be run on HW or in simulation. For simulation, the base_px4 image has to be built first, while base_uav_hw has to be built before running on hardware. Follow Section 2.3 with the service and container names declared in Table 3.

	Hardware	**Simulation**
Base service name	base_uav_hw	base_px4
UAV service name	uav_hw	uav_sim

Table 3: Service- and container names for the UAV

2.6.2 Raspberry Pi Configuration

- Do all steps as in Section 2.2
- Configure the UART pins so that the RPi uses the PL011 UART [5].

3 Qualisys

When running the system on hardware in Arena Visionen it is vital to set up Qualisys in a correct manner. Qualisys is a system in Visionen which consists of multiple cameras which together with corresponding software (called QTM) are able to track objects defined with markers.

The first one will have to do is to log in to the stationary computer, start a new project in QTM and calibrate the system according to the Qualisys documentation [3].

3.1 Defining bodies

To get position data from Qualisys, the bodies need to be defined. To do this

- Select the markers to be defined.
- Right click and select "Define rigid body (6DOF)" and name it with an appropriate name.

4 Base Station

To run the Base Station, either start an existing base_station container or create one according to Section 2.4 and Section 2.3.

4.1 Mission Setup

4.1.1 Hardware

To start a mission on hardware, start a base_station_host container. Compile the code using

```
$ make build
```

Source a second terminal as described in Section 2.3. After this, compile the code in the second terminal, typing

```
$ make base_station_hw <arguments>
```

The base_station_hw make command has the following arguments.

Argument	values	default	explanation
NAME	String	'rover1'	Optional and used for namespacing the rover in Rviz2.
RVIZ	true/false	true	Specify if a Rviz2 window should be opened, and configured to the Rover.
RECORD	true/false	false	Records all ROS2 topics into a ROS2 bag.

4.1.2 Simulation

To prepare the base station for simulation, start a base_station container. Compile the code using

```
$ make build
```

Source a second terminal as described in Section 2.3. After this, make the base_station in the second terminal, typing

```
$ make base_station <arguments>
```

The base_station make command has the following arguments.

Argument	values	default	explanation
NAME	String	'rover1'	Optional and used for namespacing the rover in Rviz2.
SIM	true/false	true	Specify if Gazebo should be started.
RVIZ	true/false	true	Specify if a Rviz2 window should be opened, and configured to the Rover.
RECORD	true/false	false	Records all ROS2 topics into a ROS2 bag.
WORLD	String	'slam.world'	Specify a .world file to be loaded into Gazebo.

4.1.3 No-fly Zones

No-fly zones as specified with a .png-file. In order to fit with the dimensions of Visionen it should be 120x120 pixels, where 10 pixels represents 1 meter. All areas where the UAV should be allowed to fly should be colored white, whereas No-fly zones should be colored entirely black. If one needs to change the map, it should be named: *map_test2.png* and the current image should be replaced with the new one. The path to the image inside the git repository is:

```
$ src/bringup/cfg/map_test2.png
```

The software will interpret the black pixels as No-fly-zones and create inflation layers around it. In order to keep a safety distance to the outer boundaries, i.e. walls, one should consider colouring the pixels at the edges black.

4.2 Spawning Distressed Persons

Spawning of Distressed persons in simulation is done in the base station. Once the Base Station is built and a Base Station container is active, distressed persons can be spawned.

- In one terminal, start a Base Station docker container.
- Build base station by typing

```
$ make build
```

- Open a second terminal, source it, and compile the code by typing

```
$ docker exec -it <base_station container name> bash
$ . install/setup.bash
$ make base_station
```

- Once the Base Station has started the simulation in Gazebo, source a third terminal as above.
- Now Distressed persons can be spawned with the following line

```
$ make dp <arguments>
```

- The following make file arguments are available. If BLUE_line, BLUE_circle, YELLOW_line, YELLOW_circle all are false, both robot will be stationary if not controlled manually according Section 4.2.1

Argument	values	default	explanation
dpID	Integer	1	Optional and used for namespacing.
dpID2	Integer	2	Like dpID but for a second Distressed person.
BLUE	true/false	true	Spawn a blue Distressed person. If false at the same as YELLOW, nothing will spawn.
BLUE_line	true/false	false	Spawn a blue Distressed person moving on a line. BLUE argument has to be true. BLUE_circle argument has to be false.
BLUE_circle	true/false	false	Spawn a blue Distressed person moving in a circle. BLUE argument has to be true. BLUE_line argument has to be false.
YELLOW	true/false	true	Spawn a yellow Distressed person. If false at the same as BLUE, nothing will spawn.
YELLOW_line	true/false	false	Spawn a yellow Distressed person moving on a line. YELLOW launch argument has to be true. YELLOW_circle argument has to be false.
YELLOW_circle	true/false	false	Spawn a yellow Distressed person moving in a circle. YELLOW launch argument has to be true. YELLOW_line argument has to be false.

4.2.1 Controlling Distressed Persons

When wanting to control a Distressed person other than in a circle or in a straight line. A package called "rqt_robot_steering" can be used. With this package, the linear and angular velocities can be controlled. This package is installed in the base station docker file. The following commands can be used to start and use the package:

- Source a base station container
- Write

```
$ ros2 run rqt_robot_steering rqt_robot_steering
```

- A GUI for controlling the Distressed persons will appear. Change the velocity topic to "dp1/cmd_vel" or "dp2/cmd_vel" to control a blue or yellow Distressed person, respectively.

4.3 Starting the mission

When all the parts of the system have been started, the mission can be started with the following command. Remember to run ". install/setup.bash" first, otherwise an error will occur.

```
$ ros2 topic pub --once /StartMission \
  custom_messages/msg/MissionInfo \
  "{cmd: Take off, color:[BLUE,YELLOW]}"
```

The color key tells which distressed person you want to search after in the mission. Only blue and yellow is supported.

4.4 Restarting mission

If you have finished a mission, you can restart the mission by deleting the rover in Gazebo, abort the processes in the terminal in which 'make rover' was run, and rerun the command. Then the mission can be restarted with the following command:

```
$ ros2 topic pub --once /StartMission \
  custom_messages/msg/MissionInfo \
  "{cmd: Abort to init, color: [YELLOW,BLUE]}"

$ ros2 topic pub --once /StartMission \
  custom_messages/msg/MissionInfo \
  "{cmd: Start phase 1, color: [YELLOW,BLUE]}"
```

5 Rover

The Rover is run either on hardware or in simulation, and the configuration process differs slightly. Below, the two processes are explained.

5.1 Hardware

The Rover is equipped with an RPi (RPi 4 Model B) as its main computer, powered by a USB type C cable. Connected to it is an Arduino that sends signals to the motors, gets rotation data from the wheel sensors, and gets signals from the RC remote controller that it passes to the RPi. The Remote Controller (Futaba 4PLS) is used to manually drive the Rover.

5.1.1 Powering and Charging

The batteries should always be charged while still inside the Rover. Switch off the main power and plug in the power cable (XT60). Make sure that the red and black parts of the connection match.

5.1.2 RC Controller

Manual driving of the Rover can be done by using the RC controller. Start the Rover normally, and power on the RC controller. The DL1 dial can be used to switch between the manual and the autonomous mode. It is done by checking the dial value mod 2, so a single tick on the dial should do the trick. When in manual mode, the Rover can drive back and forward with the main trigger, and rotate with the steering wheel.

5.1.3 Usage

To use the Rover on hardware, the following steps have to be made.

- Place the Rover in Visionen in the origin of the coordinate system, pointing along the x-axis. Confirm that the Rover is defined in the Qualisys system.
- Connect to the RPi from a laptop to create a docker container, compile the code, and launch the software.

```
$ ssh pi@192.168.0.12
$ cd sbd
$ docker-compose run rover_hw
  -- In docker container --
$ make build
```

  then open a new terminal. Connect to the UAV via SSH and source a new terminal to start the simulation.

```
$ ssh pi@192.168.0.12
$ cd sbd
$ docker exec -it rover_hw_container bash
  -- In docker container --
$ source install/setup.bash
$ make rover_hw
```

- The Rover should now appear in Rviz2 with SLAM mapping the surrounding area. The Rover is now ready to be run.

5.2 Simulation

Before starting a simulation of the Rover, the Base Station has to be started according to Section 4.1.2.

5.2.1 Usage

This section will cover all the necessary steps in simulating the Rover.

- Start or create a **rover_sim** docker container. If creating a new container, **docker-compose run rover_sim** should be run instead. In the second terminal, the name of the container will be **sbd_rover_sim_run_...** instead of **rover_sim_cont**.

```
$ cd sbd
$ docker start -i rover_sim_cont
  -- In docker --
$ make build
  -- Second terminal --
$ docker exec -it rover_sim_cont bash
  -- In docker --
$ source install/setup.bash
$ make rover_sim
```

- The Rover should now appear in both Gazebo and Rviz2. It should instantly start mapping the surrounding area with SLAM, which can be seen in Rviz2. The Rover is now ready to start a mission or perform any action the user would want it to do.

6 UAV

The UAV is run either on hardware or in simulation, and the configuration process differs slightly. Below, the two processes are explained.

6.1 Hardware

The UAV consists of a Raspberry Pi 4, a Pixhawk 4, an RC-receiver, a camera, a power module, an ESC (Electronic Speed Controller), four actuators, and a battery. All the components are mounted on a Luminer QAV-R airframe. There are also four reflective balls mounted on the UAV so that the Qualisys system can locate it. An RC-controller is also available to fly the UAV.

6.1.1 Powering and Charging

The UAV is powered by an 11.1V LiPo battery connected to the power module, which is connected to the ESC that powers all the actuators. The RPi is connected to the power module through a voltage controller converting the voltage to 5V, and the Pixhawk receives its power from the RPi through a USB cable.
The battery is a three-celled LiPo battery with a maximum voltage of 12.6V. LiPo batteries need to be handled with care since they can easily be damaged. The nominal voltage of a LiPo cell is 3.7V and if the voltage drops below 3.0V the battery can be damaged. To prevent this, a voltage measurement device must be connected to the battery at all times when connected to the UAV. This device will signal if the voltage drops below a certain voltage, which is now set to 3.6V. This limit can be changed with a button on top of the device. If the voltage alarm goes off when flying, the UAV should immediately be manually landed and the battery changed [4].
LiPo fires are self-oxidizing and therefore hard to extinguish. Furthermore, hydrogen fluoride gas and other gases are produced which are both toxic and lethal. LiPo batteries should never be charged or used unattended. If a battery is swollen, makes a gas-leaking sound, or is abnormally warm, the battery should be disconnected and safely discarded. The voltage alarm should always be connected when using the battery.
The batteries should always be stored in LiPo-safe containers/bags when not used or when charging. Both connections on the battery must be connected to the charger and the charging current should be set to 1A.

6.1.2 RC Controller

The RC controller is used to manually fly the UAV and can switch between the different modes. The following modes are available: Manual, Altitude, Position, and Offboard. The three first modes are manual modes where the user flies the UAV, and in the last mode, the RPi flies it. When the controller is in manual mode, the roll, pitch, yaw, and thrust are controlled with the sticks on the RC controller. To simply fly the UAV, the recommended mode is Altitude mode, where the barometer is used to maintain altitude. If the Pixhawk receives position data from Qualisys, Position mode is recommended instead, since it holds the position when both sticks on the controller are centered. There is also an arm switch that makes the propellers start spinning, as well as a kill switch that immediately cuts all power and stops the propellers. Before starting the software on the UAV, the RC controller must be switched on and the user must be ready to switch to manual mode and take control of the flight. The UAV will not be able to fly if not connected to the controller. When flying autonomously, the UAV will arm itself and switch to Offboard

mode when the takeoff service is called. It is also impossible for the UAV to override the kill switch.
In Figure 2 the RC-controller is shown with all the switches named. The different functions are mapped as follows:

- 1: Right joystick x-axis: Controls roll
- 1: Right joystick y-axis: Controls pitch
- 2: Left joystick y-axis: Controls throttle
- 2: Left joystick x-axis: Controls yaw
- 3a and 3b: Control flight mode
- 4: Kill switch, immediately cuts power to engines
- 5: ARM switch
- 6: Offboard switch

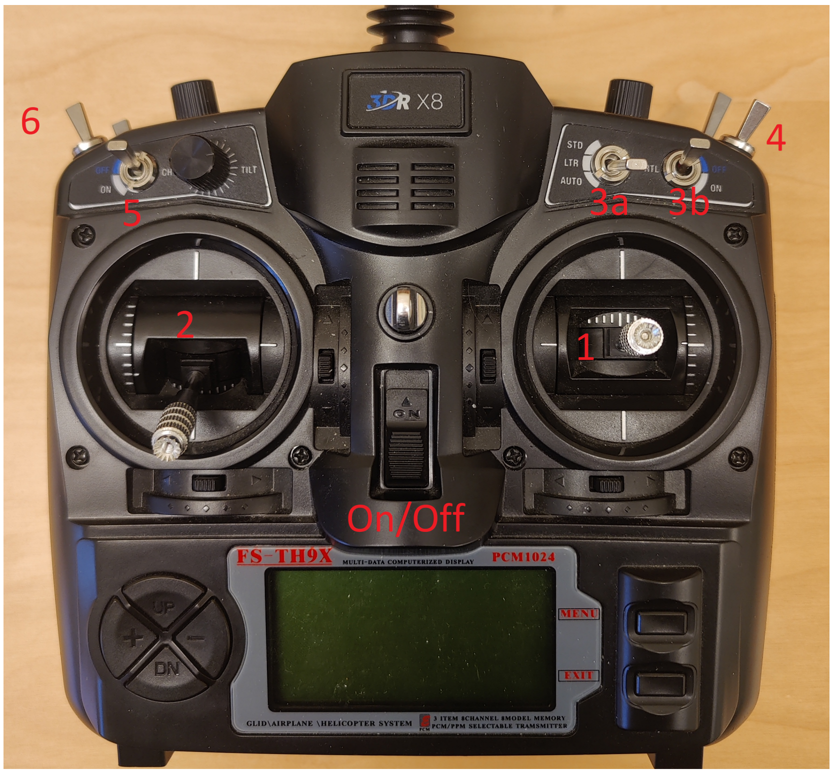

Figure 2: The RC-controller used to fly the UAV, with markings for all the used switches.

To fly in Altitude mode, switch 3a should be set to its most downward position and 3b set upwards. To switch to position mode, the 3b should be switched down and 3a can be in any position.

6.1.3 Propellers

The four propellers must be mounted correctly according to Figure 3. There are two propellers designed to spin clockwise and two designed to spin anti-clockwise, and each pair is to be mounted in the opposing corners.

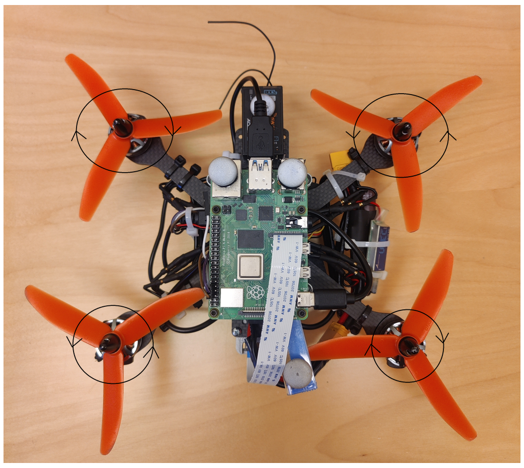

Figure 3: The UAV with the directions of the propellers marked.

6.1.4 Usage

This section will cover all the necessary steps in launching the UAV when running on hardware.

- Confirm that all propellers are mounted correctly and that they do not hit any cables.
- Calibrate Qualisys according to Section 3.
- Place the UAV in Visionen in the origin of the coordinate system, pointing along the x-axis. Confirm that the UAV is defined in the Qualisys system by making sure that the markers tagged with UAV pops up on the Qualisys computer.
- Connect the voltage measurement device to the battery and connect the battery to the UAV.
- Switch on the RC-controller.
- Connect to the RPi from a laptop and create a docker container, compile the code, and launch the software.

```
$ ssh uavpi@192.168.0.28
$ cd sbd
$ docker-compose run uav_hw
  -- In docker --
$ make build
$ source install/setup.bash
$ make uav_hw
```

- If there already exists a container on the RPi where *make build* has been run and no code has been changed on the UAV, an existing container can be started instead.

```
$ ssh uavpi@192.168.0.28
$ docker start -i uav_hw_cont
  -- In docker --
$ source install/setup.bash
$ make uav_hw
```

- The UAV will now begin flying if the take-off service is called.

6.2 Simulation

Before simulating the UAV, the image **base_px4** needs to be built, and the base station needs to be started according to Section 4.1.2.

6.2.1 Usage

This section will cover all the necessary steps in simulating the UAV.

- Start or create a **uav_sim** docker container according to Section 2.3. In a second terminal, use the command

```
docker ps -a
```

to get the correct name for the container instead of **uav_sim_cont** below)

```
$ cd sbd
$ docker start -i uav_sim_cont
  -- In docker --
$ make build
  -- Second terminal --
$ docker exec -it uav_sim_cont bash
  -- In docker --
$ source install/setup.bash
$ make uav_sim
```

References

[1] Pieter Beulque. *Automatically connect a Raspberry Pi to a Wifi network.* `https://weworkweplay.com/play/automatically-connect-a-raspberry-pi-to-a-wifi-network/`. (Visited on 11/28/2022).

[2] Docker Inc. *Use containers to Build, Share and Run your applications.* `https://www.docker.com/resources/what-container/`. (Visited on 12/02/2022).

[3] Qualisys. *Calibrating your system.* `https://docs.qualisys.com/getting-started/content/getting_started/running_your_qualisys_system/calibrating_your_system/calibrating_your_system.htm`. (Visited on 12/05/2022).

[4] Brian Schneider. *A Guide to Understanding LiPo Batteries.* `https://www.rogershobbycenter.com/lipoguide`. 2021-10-09. (Visited on 11/28/2022).

[5] AB Electronics UK. *Serial Port setup in Raspberry Pi OS.* `https://www.abelectronics.co.uk/kb/article/1035/serial-port-setup-in-raspberry-pi-os`. 2022-11-10. (Visited on 11/28/2022).