Technical Report
Search and Rescue - Land

Version 1.0

Author: Rickard Wretlind
Date: January 4, 2023

Status

Reviewed	Jakob Åslund	2023-01-03
Approved	Jakob Åslund	2023-01-04

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:	Technical_Report.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 Fagerstedt	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	2022-12-07	First draft	RW	Jakob Åslund
0.2	2022-12-12	Second draft	RW, DG	Jakob Åslund
0.3	2022-12-14	Third draft	DG	Jakob Åslund
0.4	2022-12-20	Fourth draft	DG	Jakob Åslund
0.5	2022-12-21	Fifth draft	SF, AL	Jakob Åslund
0.6	2022-12-22	Sixth draft	DS	Jakob Åslund
1.0	2023-01-04	First version	DS	Jakob Åslund

Contents

1 Introduction

This document is the technical report for the Search and Rescue - Land project, in the course TSRT10 Automatic Control - Project Course. It presents the specifications for the design of the system.

1.1 Parties

There are principally two different parties involved in this project: Linköping University and SAAB Dynamics. In the context of the course, the SAAB Dynamics party acts as customers interested in a Search and Rescue-platform, and the University party acts as a company tasked with delivering the product.

The party from Linköping university consists of: Anja Hellander acting as advisor, Jakob Åslund acting as orderer, and the project group which consists of seven master students. Furthermore, the party from SAAB Dynamics consists of: Torbjörn Crona fulfilling the role of customer, and Linus Wiik, Joel Wikner and Åke Johansson acting as additional advisors.

1.2 Purpose and goal

The main purpose of this project is to develop a Search and Rescue-system consisting of an unmanned ground vehicle, called the Rover, and an Unmanned Aerial Vehicle (UAV) collaborating to identify, track and supply people in distress. To accomplish this, the system should be able to map and navigate an area with people in distress. Specifically, the Rover should use a LIDAR-sensor to map the environment, whilst, using cameras, collaboratively searching with the UAV for distressed people.

Thus, the goal is to deliver a system according to the aforementioned specifications. The system should also follow design guidelines such as adhering to Google's code standard, developing the system in ROS2 and extending the docker integration of the current project.

1.3 Usage

As mentioned above, the intended use case of the product is deployment in a complex environment with distressed people to save. However, the finished product will only function in an environment as described in "Project_Plan.pdf"[9].

1.4 Background Information

To search for and rescue distressed people by foot can be dangerous and time-consuming. In order to increase the safety and improve the chance of finding and rescuing distressed persons, robots, such as rovers and UAV:s, can be used instead.

This project has been developed for many years at Linköping university and has had different development goals through the years. Today the system consists of one UAV and one Rover which are equipped with different sensors, cameras, and LIDAR which makes it possible to map the environment and create motion plans in search and rescue missions.

The project has a sister project called "Search and Rescue – Underwater". The cooperation between the two projects is slight, while there are visions to unify the projects in the distant future.

1.5 Definitions

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

- **Rover** - An autonomous tracked vehicle.
- **UAV** - An autonomous quadcopter.
- **Agent** - 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.
- **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.

2 System Overview

This section aims to give an overview of the entire system. The system utilizes SIL, which means it can either be run on hardware or in simulation without subsystems knowing which mode it is run on. Furthermore, the system is divided into different subsystems and three of those subsystems, if not run in simulation mode, are hardware based. Those subsystems include the Base Station, the Rover and the UAV. A schematic overview of those can be seen in Figure 1.

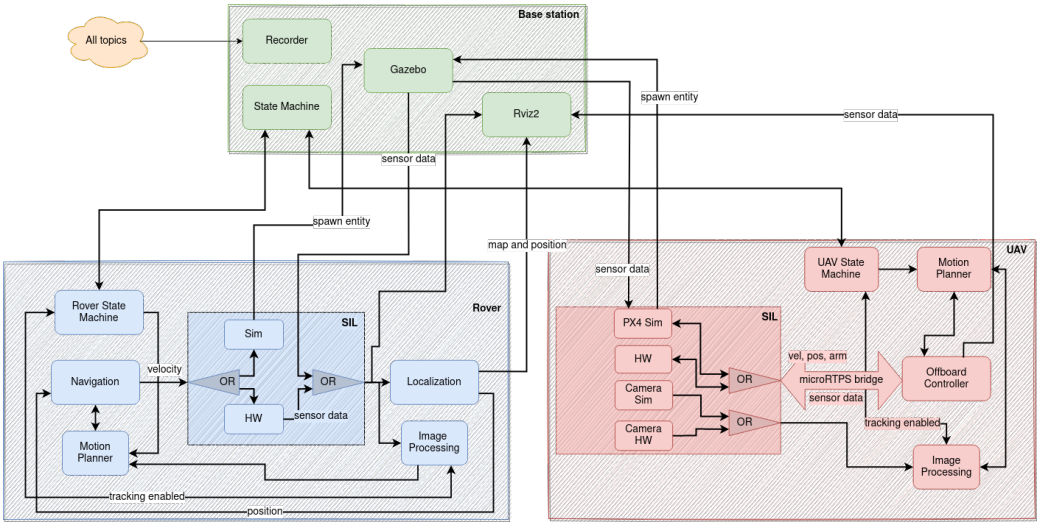

Figure 1: An overview of the entire system with Base Station, UAV and Rover.

The system uses the Base Station as an intermediator for communication between the Rover and the UAV. The Base Station runs a state machine that handles communication via a ROS2 network, and coordinates the Rover and UAV during a mission. During a mission the state machine tells the robots which state they should be in, and then the robots plan and execute the tasks in that state by themselves.

Not visualized in Figure 1 is that the system uses Qualisys implemented in the arena Visionen. Qualisys provides an estimated position (similar to a GPS) for the Rover and UAV.

2.1 Subsystems

The system is divided into the following subsystems:

- Base Station
- Rover
- UAV
- Simulation Environment

2.2 Mission

Briefly summarized, the mission is to search an unknown area with the Rover and UAV until all distressed persons are found. At first the Rover will map the area while searching for distressed persons and the UAV will search for distressed persons. If the Rover finds a distressed person it will start tracking it and the UAV will fly to collect and deliver supplies. If the UAV finds the distressed person it will track the person and the Rover will start to drive to its position. Once the Rover sees the person it will take over the tracking and the UAV will fly to collect and deliver supplies. When all distressed persons have been found, the Base Station will signal mission complete.

2.3 Communication

The primary software used for communications, both between subsystems and inside a subsystem, is ROS2. ROS2 enables communications via:

- Topics
- Services
- Actions

Topics will be primarily used in the communication utilizing ROS2. Furthermore, the communication between the Base Station, Rover and UAV is accomplished through a network (which enables the use of ROS2 topics). While running simulations on a single computer, a bridge network is employed to communicate between the different systems locally. While running on hardware, each system acts as a host on their computing platform and communicate over Wi-Fi.

3 Image Processing and Position Estimation

The Rover and the UAV both use the same type of image processing. To search for distressed persons, an image recognition algorithm is running. In our system, distressed persons are modeled as blue and yellow rc cars. The image recognition is therefore calibrated to look for those specific colors. When a distressed person has been identified, they instead run an image tracking algorithm to follow the distressed person in the image.

3.1 Image Recognition

While the robots are not tracking a distressed person, an image recognition algorithm is running, tasked with identifying distressed persons. The algorithm uses a color model of the distressed persons to identify them. Since RGB-space is prone to noise, usually the HSV-space, which separates out hue (H), saturation (S), and value (V) from an image, is used instead [2]. Specifically, every incoming image from the camera is converted to the HSV-space and masked to only include hue-values in a range around the hue value of the color model. Afterwards, the algorithm searches for contours in the resulting masked image, and any resulting contour of a size greater than a predefined threshold is taken to be a distressed person.

3.2 Image tracking

The tracking algorithm combines the Camshift Algorithm [2] with an extended Kalman Filter using a coordinated turn motion model for the target. The classical Camshift Algorithm is an improvement of the Meanshift Algorithm. The Meanshift Algorithm is a non-parametric clustering algorithm, which, given an initialization, uses an iterative search to find the nearest dominant peak in a probability distribution. The idea of the algorithm is illustrated in Figure 2. More rigorously, given n samples $x_i, i = 1, 2, ..., n$ in R^d, then the mean shift vector $m(x)$ is defined as

$$m(x) = \frac{\sum_{x_i \in S_h} K(x_i - x)x_i}{\sum_{x_i \in S_h} K(x_i - x)}, \tag{1}$$

where S_h is a closed ball in R^d, centered on x, and K is a kernel function which weights points x_i inside S_h. Thus, the mean shift vector forms an average, with respect to the kernel function $K(x)$, of the samples in a neighborhood S_h around a point x. Now consider the difference $m(x)-x$, which corresponds to a *mean shift*, i.e., a shift to the densest region in S_h. The iterative Meanshift Algorithm computes the mean shift given some initial point x_0, and then iteratively recomputes the mean shift using the output mean shift from the previous iteration as a new point in the algorithm, $x_{k+1} = m(x_k) - x_k$. The method converges to some local optimum with respect to the density of the samples in the data.

In the target tracking setting, an image with an initial window containing the target is used to create a histogram. While more of the dimensions can be used, the standard algorithm is based on the hue, and thus it is the hue which is sampled from the target in the image and stored as a 1-d histogram.

A probability distribution is computed using the histogram in a process called *back projection*. A back projection of an image given a model histogram is computed by, for each pixel value, finding the corresponding bin in the histogram and assigning the back projected image the value stored in the bin at the sampled point. Thus, an image, $I(x, y)$, is obtained which represents a probability distribution of the target in the image.

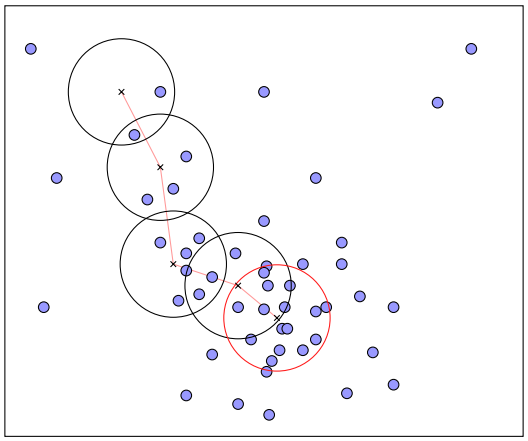

Figure 2: The Figure shows an example of the Meanshift Algorithm. The blue dots represent samples of some distribution, and each black circle represents the closed ball in an iteration of the algorithm, which terminates at the red circle having reached some goal threshold.

The back projected image $I(x, y)$ can then be used as an input to the Meanshift Algorithm, along with an input window representing the closed ball, in order to locate the target. Most commonly, the kernel function is chosen as an *image moment*, $M_{i,j}$, in the Meanshift Algorithm for images. The image moment represents a weighted average of the pixel intensities of an image $I(x, y)$, and is computed as

$$M_{i,j} = \sum_x \sum_y x^i y^j I(x, y). \tag{2}$$

Specifically, the first moments $M_{1,0}$ and $M_{0,1}$ are typically used for the x- and y-coordinate respectively. From (1) and (2), it follows that x- and y-coordinates are normalized by the zeroth moment $M_{0,0}$. Using the back projected image $I(x, y)$ and the Meanshift Algorithm with the image moments kernel, the center position (x_c, y_c) is obtained for every iteration as

$$(x_c, y_c) = \Big(\frac{M_{1,0}}{M_{0,0}}, \frac{M_{0,1}}{M_{0,0}}\Big). \tag{3}$$

The Meanshift Algorithm terminates once the initial window has been moved to a local optima in the back projected image. This process is then repeated for every incoming image from a video feed, where the previous converged position is used as initial position in the subsequent frame.

The Meanshift Algorithm works well in a setting where the tracked target has a constant size, that is, it struggles with targets moving to and from the camera, deformed targets, or partially occluded targets. The Camshift Algorithm's contribution to the Meanshift Algorithm is in tackling this problem by adaptively changing the frame size, s, of the target as a function of the zeroth order moment $M_{0,0}$. In the original paper [2] the window size s was updated as

$$s = 2\sqrt{\frac{M_{0,0}}{256}}. \tag{4}$$

This frame size is employed in the tracking algorithm. To estimate the location of a distressed person, the center position, the width, and the height of the frame, measured

in pixels, is used from the Camshift algorithm. The Camshift Algorithm is summarized as a flowchart in Figure 3.

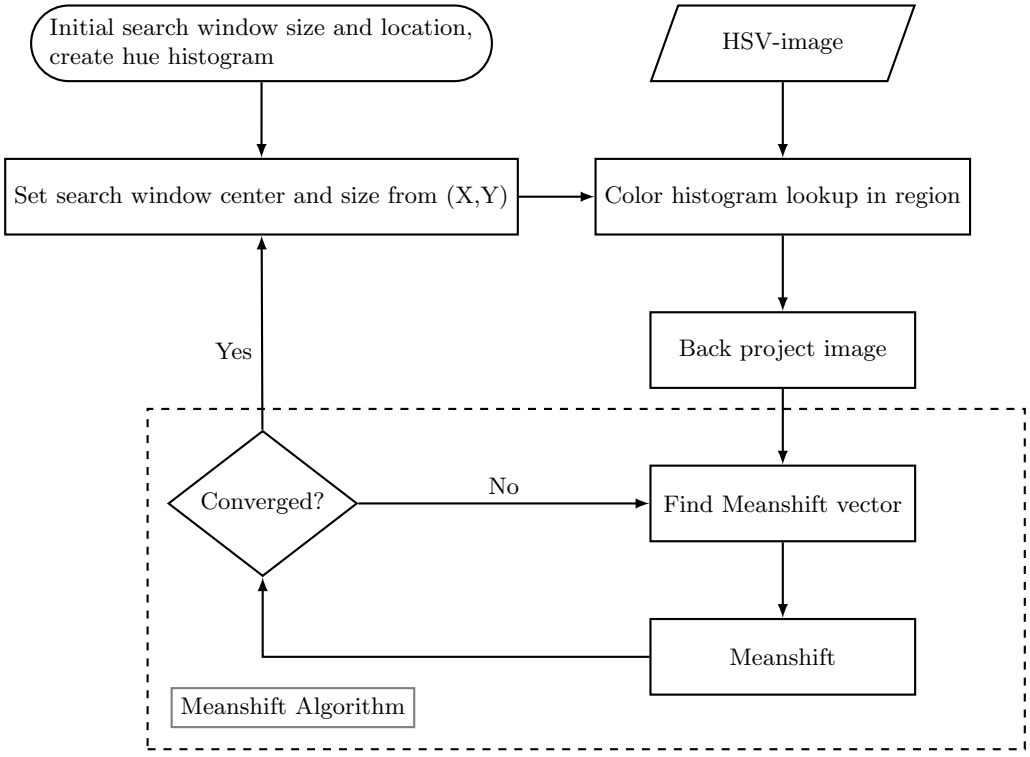

Figure 3: The Figure shows a flowchart of the Camshift Algorithm. The dashed box represents the steps in the Meanshift Algorithm. The algorithm starts at the rounded rectangle and terminates once there are no new HSV-images to process.

3.3 Extended Kalman filter

The Camshift Algorithm is extended to include an extended Kalman Filter in order to achieve better tracking in the presence of occlusion of the target or other disturbances. For an overview of the EKF filtering topic, see the chapter on the topic in [5]. The target is described by a coordinated turn motion model, which is used in the time update of the Kalman Filter to estimate the target in a dead-reckoning fashion when it is occluded. The state vector used in the extended Kalman filter is $\begin{bmatrix} x & y & \dot{x} & \dot{y} & \omega \end{bmatrix}^T$ where x and y are the global x- and y-positions, $\dot{x}$ and $\dot{y}$ are the global velocities in the x- and y-directions, ω is the angular velocity. The state transition model is a non-linear model, since a coordinated turn model is used, and is defined by

$$\mathbf{x_{t+1}} = \mathbf{f}(\mathbf{x_t}) + \mathbf{w_t}, \tag{5}$$

where $\mathbf{f}(\mathbf{x_t})$ is

$$\mathbf{f}(\mathbf{x_t}) = \begin{bmatrix} x + \frac{\dot{x}}{\omega} sin(\omega T) - \frac{\dot{y}}{\omega}(1 - cos(\omega T)) \\ y + \frac{\dot{x}}{\omega}(1 - cos(\omega T)) - \frac{\dot{y}}{\omega} sin(\omega T) \\ \dot{x} cos(\omega T) - \dot{y} sin(\omega T) \\ \dot{x} sin(\omega T) + \dot{y} cos(\omega T) \\ \omega \end{bmatrix},$$

$\mathbf{x_t}$ are the states, and $\mathbf{w_t}$ is the process noise. This model is then linearized at every state using

$$\mathbf{x_{t+1}} = \begin{bmatrix} 1 & 0 & \frac{sin(\omega_t T)}{\omega_t} & -\frac{1-cos(\omega_t)}{\omega_t} & 0 \\ 0 & 1 & \frac{1-cos(\omega_t T)}{\omega_t} & \frac{sin(\omega_t)}{\omega_t} & 0 \\ 0 & 0 & cos(\omega_t T) & -sin(\omega_t) & 0 \\ 0 & 0 & sin(\omega_t T) & cos(\omega_t) & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{x_t} + \mathbf{w_t}. \tag{6}$$

The measurement model used is defined by

$$\mathbf{z_t} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \mathbf{x_t} + \mathbf{v_t}. \tag{7}$$

The x and y states in $\mathbf{x_t}$ are estimated as described in Section 5.5.1and Section 6.6.1 for the Rover and UAV, respectively. $\mathbf{v_t}$ is the observation noise. Little domain knowledge has been employed in selecting the initial covariance $\mathbf{P_0}$, process noise covariance $\mathbf{Q}$, and the observation noise covariance $\mathbf{R}$ in the filters. In addition, the correlation between the position and the velocity was not known. Thus, the initial covariance is simply a diagonal matrix of the form

$$\mathbf{P_0} = diag([10, 10, 1, 1, 1]),$$

and the process- and observation noise are assigned identity. Further work could include fine-tuning these covariances, investigating the correlation between the position and velocities, and weighing the noise in different magnitudes.

The presence of occlusion of the target is measured using the Bhattacharyya coefficient [4], which is a measure of the overlap between two statistical samples. In this case, the statistical samples consist of the original histogram of the target, $p(i, j)$, and a histogram from the converged Camshift Algorithm. Given these quantities, the Bhattacharyya coefficient, $C_B(p, q)$, is defined as

$$C_B(p, q) = \sum_{(i,j) \in X} \sqrt{p(i, j)q(i, j)}, \tag{8}$$

where X is the set of hue-bins in the histograms, and it holds that $0 \leq C_B(p, q) \leq 1$. A low value of $C_B(p, q)$ indicates similar samples, and a high value indicates dissimilarity between the samples. The coefficient is used by fixing a threshold α, beyond which the object is said to be occluded. When the target is visible, the output of the Camshift Algorithm is used as measurement update in the extended Kalman Filter. The time update is used to update the starting position of the Camshift Algorithm between frames. See Figure 4 for an overview of how the combined Camshift and Kalman algorithm works.

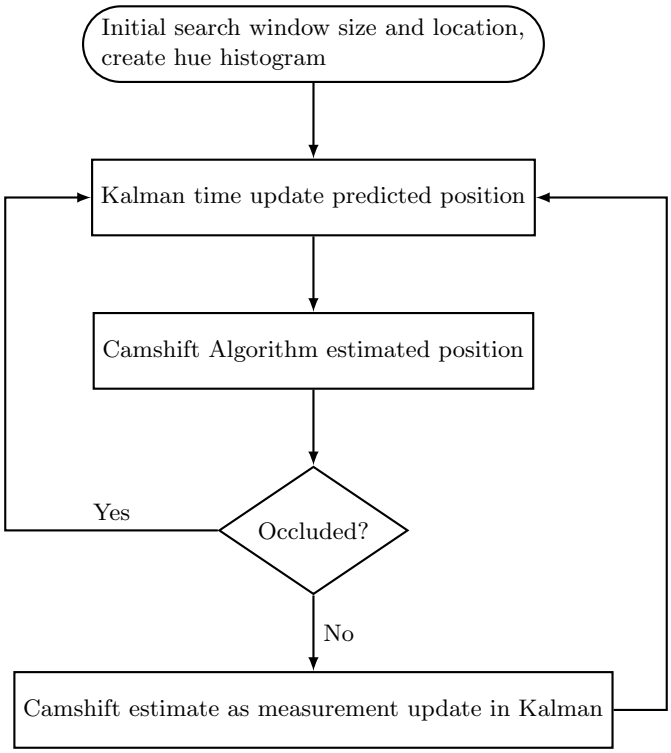

Figure 4: An overview of the Camshift Kalman algorithm to estimate the position of the distressed persons. Occlusion is measured using the Bhattacharyya coefficient and a predetermined threshold.

4 Base Station

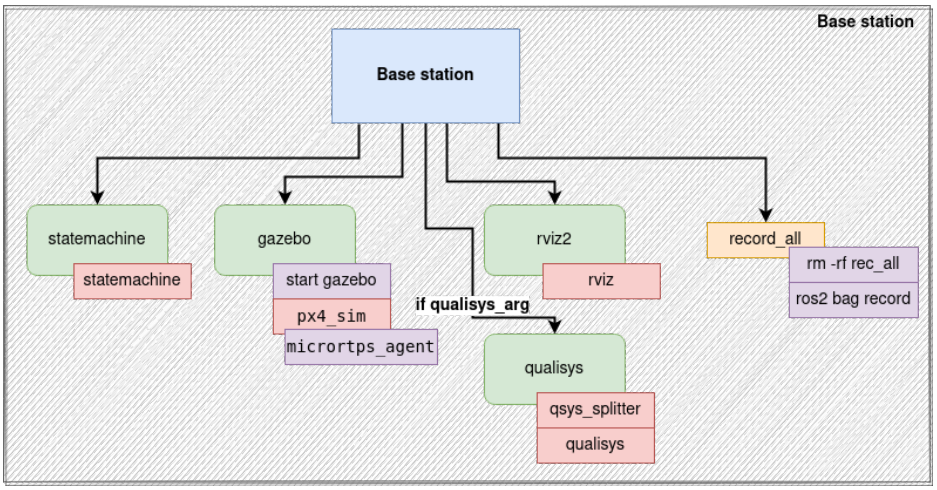

Figure 5: Schematic view of the Base Station's launch file.

The purpose of the Base Station is to plan and coordinate the missions. During a mission, it sends instructions to the Rover and the UAV. Figure 5 shows an overview of the Base Station's responsibilities.

4.1 State Machine

The State Machine is responsible for coordinating the UAV and the Rover during the mission. It is basically a directed graph where each vertex is a state and each edge is a transition between states. For a given state, only transitions to certain states are allowed. E.g., the UAV cannot start searching before it has taken off. After system launch, the state machine starts in an init state. The graph is shown in Figure 6.

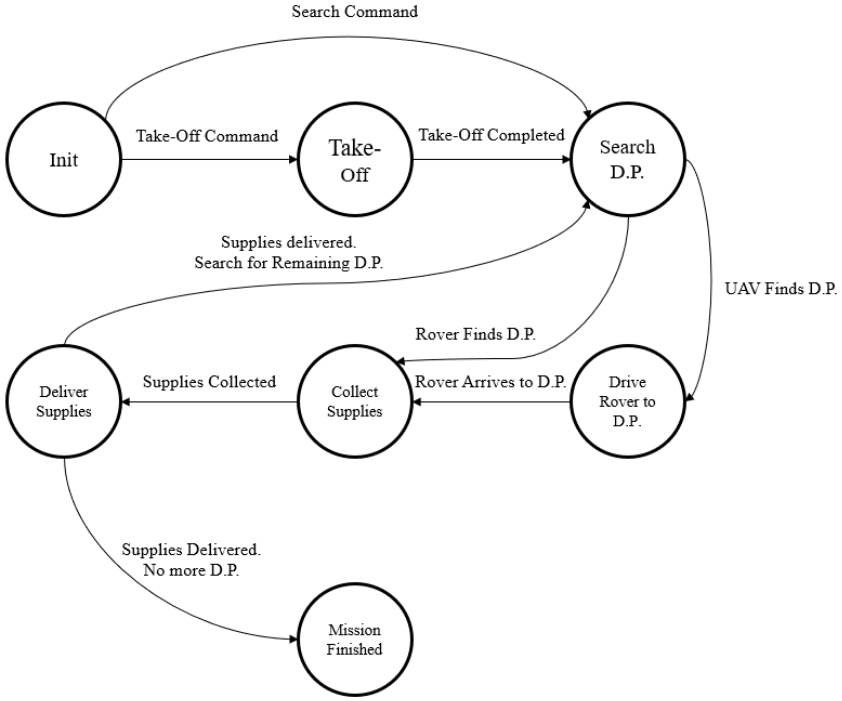

Figure 6: This figure shows the graph used by the state machine during a mission.

The missions states are illustrated in Figure 6 and a detailed explanation of the states can be seen below:

- **Initial state** - The mission is defined or reset and the Rover and UAV are ready to be deployed.
- **Take-Off** - The UAV takes off and the Rover should be still until the take-off is done.
- **Search D.P.** - The Rover and the UAV search after distressed persons. The Rover drives in such a way that it explores and maps the world and the UAV searches the world in a predefined pattern.
- **Drive Rover to D.P.** - The UAV tracks a found distressed person, and the Rover drives towards the distressed person until it can track it.
- **Collect supplies** - The Rover tracks the found distressed person while the UAV collects supplies.
- **Deliver supplies** - The rover tracks the found distressed person while the UAV delivers supplies to the found distressed person.

The transitions between states are explained below.

- **Take-Off Command** - A command typed into the terminal to arm and launch the UAV.

- **Search Command** - A command typed into the terminal to start the search state.
- **Take-Off Completed** - The UAV has taken off successfully.
- **UAV Finds D.P.** - The UAV has spotted a distressed person with its camera.
- **Rover Finds D.P.** - The Rover has spotted a distressed person with its camera.
- **Rover Arrives to D.P.** - The Rover spots the distressed person that is being tracked by the UAV and starts tracking it.
- **Supplies Collected** - The UAV has finished its Collect Supplies script.
- **Supplies delivered. Search for Remaining D.P.** - The UAV has delivered supplies to the distressed person and it is considered saved. The robots start searching for the next person
- **Supplies Delivered. No more D.P.** - The UAV has finished delivering supplies to the last distressed person.

4.1.1 States

Listed below are the states that the robots can be in. The state machine decides which state the robots should be in.

Rover The Rover has the following states.

- **Map area** - The Rover will run a discovery algorithm to map up the area. When complete, it will inform the Base Station that phase one is completed.
- **Track a person** – The Rover will follow the distressed person until the UAV arrives with supplies.
- **Drive Rover to DP** - The Rover will drive to the location of the distressed person.

UAV The UAV has the following states.

- **Search** - The UAV will search the whole environment for distressed persons with a predefined search pattern.
- **Collect supplies** – The UAV will travel to the supply point and collect supplies.
- **Deliver supplies** – The UAV will deliver the supplies to the distressed person.
- **Land** – The UAV will perform a landing maneuver on the Rover or on the ground.
- **Track a person** – The UAV will follow the distressed person until the Rover arrives.
- **Take-off** - The UAV will perform a take-off maneuver from the ground.

4.2 Qualisys positioning system

The Base Station is responsible for distributing the Qualisys position data to the agents. A ROS2 package is used to get the Qualisys data published on a ROS2 topic [6]. The message received contains the name of the robot as defined in the Qualisys system. A node called "qualisys_data_split" listens to the topic, extracts the name from the message, and converts the data to a message which is sent to the corresponding agent.

4.2.1 Qualisys splitter node

The "qualisys_data_split" node acts as an intermediate between Qualisys and the agents. By separating the data before it reaches each agent, the amount of messages that needs to be handled by each agent is lowered. The splitter also converts the data to the correct message type and coordinate frame that each agent requires to utilize it. The coordinates received from Qualisys are in millimeters and are converted to meters. The quaternion received from Qualisys also needs to be converted since all angles are inverted with respect to the coordinate frame. Since the data received from the Qualisys ROS2 node is incorrect, the problem lies somewhere between Qualisys and the Qualisys splitter node. To compensate for this, the vector part of the received quaternion is multiplied by -1.

4.2.2 Rover positioning data

The positioning data is converted to an odometry message, that is forwarded to the Rover. The frames in the header are also set accordingly, and a timestamp is added to the message.

4.2.3 UAV positioning data

The UAV requires the positioning data to be a "VehicleVisualOdometry" message, which will be further explained in Section 6.5. Furthermore, the UAV uses an FRD (Front, Right, Down) frame when estimating its position and the received data needs to be transformed accordingly. The Qualisys coordinate system is defined with a positive z-axis (after converting it in the Qualisys splitter node), and the UAV frame is defined with a negative z-axis. The two frames also share x-axis. To convert coordinates, the y- and z-axis change sign, and to convert the quaternion p, it is rotated 180 degrees around the x-axis. This is done by multiplying with a quaternion q and its inverse q^{-1} representing a rotation of 180 degrees around the x-axis. These are defined using the basic quaternions $\mathbf{i}$, $\mathbf{j}$, $\mathbf{k}$ as

$$q = \cos\frac{\theta}{2} + (u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k})\sin\frac{\theta}{2} = /u_x = 1, u_y = u_z = 0, \theta = \pi/ = \mathbf{i}, \tag{9}$$

and

$$q^{-1} = \cos\frac{\theta}{2} - (u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k})\sin\frac{\theta}{2} = /u_x = 1, u_y = u_z = 0, \theta = \pi/ = -\mathbf{i}, \tag{10}$$

with the axis to rotate around defined as $u = (u_x, u_y, u_z)$ and the clockwise rotation θ around said axis. The rotated quaternion p' is then defined as

$$p' = qpq^{-1}, \tag{11}$$

with q and q^{-1} as defined above.

5 Rover

This chapter describes the design of the Rover. It describes both hardware and software, as well as different modes of operation. An overview of the Rover design for both hardware and simulation can be seen in Figure 7.

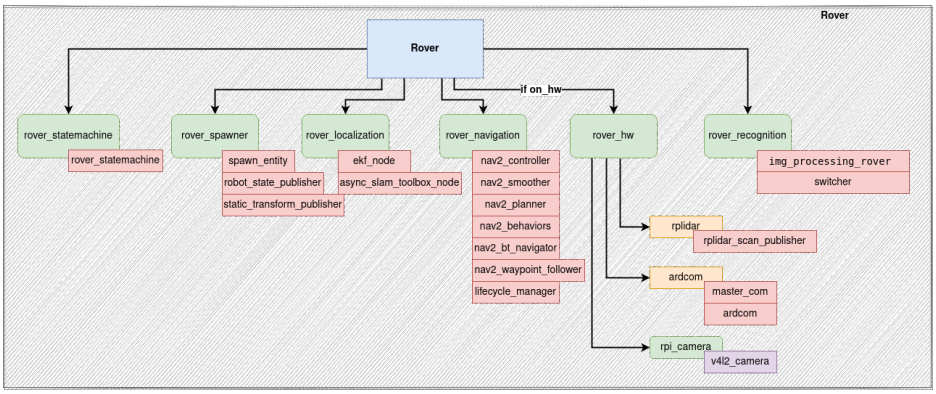

Figure 7: Schematic view of the launch file for the Rover.

5.1 Hardware

The Rover contains a multitude of individual hardware components. An overview showing which hardware components that communicate with each other can be seen in Figure 8. The included hardware and short descriptions are listed below.

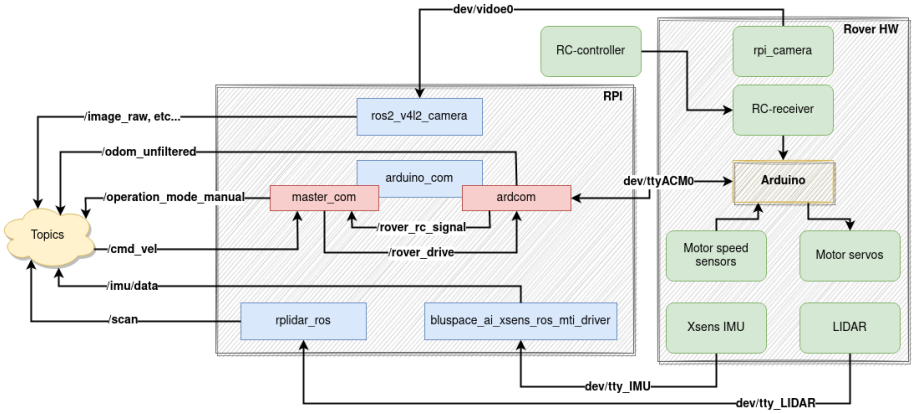

Figure 8: Overview of the hardware communication.

- **Raspberry Pi 4** - Onboard computer with Wi-Fi.
- **LIDAR, RPi LIDAR A2 from SLAMtech** - Rotating laser to obtain heading and range data that is used in SLAM.
- **RPi Camera** - Camera connected to the RPi.
- **Arduino** - Microcontroller to receive odometer data and control the tracks.
- **RC-receiver** - Receiver for manual drive.

- **RC-controller** - Controller to manually control the Rover.
- **Odometer** - Instrument used to calculate the distance the Rover has traveled since start.
- **Motor servos** - A pair of servo motors that drive the individual tracks.
- **IMU** - Inertial measurement unit, used for position estimation.
- **Qualisys balls** - Reflective markers needed for positioning in Qualisys.

5.2 SLAM

The Rover uses an algorithm for SLAM. This maps the area while keeping track of the Rover's relative location in the that area. The SLAM-algorithm uses data from the LIDAR and the positional data from Qualisys to map the world and localize the Rover. The ROS2-package Slam Toolbox is used to implement the SLAM algorithm, and it uses a graph-based 2D-implementation of SLAM. It is a continuation of SRIs old open source slam algorithm OpenKarto [7], which had multiple popular ROS1 implementations [10]. The package requires published LIDAR data and a valid transform from the vehicle (the base_link-frame) to the vehicle's starting position (the odom-frame). In turn, it publishes the map in the form of an occupancy grid and the estimated position of the vehicle in the map-frame.

5.2.1 LIDAR

The RPlidar_Ros2 package generates measurement data from the mounted LIDAR sensor, consisting of the distance and angle to the detected objects. This data is published on the topic \scan, and is used by Slam Toolbox. The package is maintained by Slamtech, the producers of the RPlidar product.

5.2.2 Robot Localization

The package Robot Localization is a classic ROS-package which has been patched to work with ROS2. It implements an EKF-filter which fuses data from the different odometry sources and publishes the filtered odometry data and the associated transform from the base_link-frame to the odom-frame, which Slam Toolbox requires. The package has support for fusing data from a plethora of odometry sensors, including: GPS, IMU and wheel encoders. There is no native support for the Qualisys message-type, however, the data from those messages can be converted in terms of either a GPS-signal or wheel-encoder data.

5.2.3 Occupancy Grid-Representation

As mentioned in Section 5.2, the Slam Toolbox package represents the map as a 2D-occupancy grid. An occupancy grid is a data structure which assigns a value between 0 and 1 to an ordered set of cells representing the map. A value of 0 indicates a completely unobstructed area, essentially air, and a value of 1 represents a fully occupied area, i.e., a wall. The occupancy grid-representation has a few advantages over a simple binary representation, in that it can model uncertainty in the mapping and a cost associated with being near objects. Mapping methods are not perfect, and thus, it is natural to represent uncertainties in the mapping using a value in-between occupied and unoccupied.

Furthermore, since estimation of the position of the vehicle is never perfect either, it is natural to represent the cost of entering areas around objects in much the same way. The motion planner and motion controller use the occupancy grid information as weights when planning paths and while following them, in a way that balances the costs associated to entering partially occupied areas with the potential to save time.

Both a local and global occupancy grid is maintained, the so-called local and global costmaps, with differing levels of resolution. The local costmap is centered around the vehicle and has a higher resolution, but is not maintained over the whole map. The global costmap, as the name implies, is a costmap spanning the whole explored area, but with a reduced resolution. These costmaps serve complementary roles; the local costmap is useful for the motion controller and motion planner in navigating the vehicle in real-time, whilst the global costmap is needed for the motion planner and mission planner to plan and coordinate the actions of the vehicle on the global scale.

5.3 Navigation

The Rover uses a motion planner to find the best path from its current position to the goal nodes decided in the search algorithm. To follow the path, a motion controller is used.

5.3.1 Motion Planner

The motion planner uses the goal nodes delivered by the different algorithm to plan out a path for the Rover. It uses information from SLAM to compute a path that does not collide with obstacles. The Rover uses a hybrid-A* motion planner from Nav2 called Smac Planner.

5.3.2 Motor Controller

A motor controller is implemented on the Rover. It uses a pair of PID-controllers to control the motor output of the Rover. One is responsible for the heading speed and the other for the yaw speed. The controller is needed for the motor output, since the dynamics of the Rover is nonlinear and not particularly well-approximated with a linear equation. Especially, the power needed to overcome the initial friction while turning in place has deadzone-characteristics. However, after the initial friction is overcome, the relationship between the motor output and achieved velocity is far closer to a linear relationship. The PID has been tuned to get satisfactory results both when turning in place and driving forward.

5.4 Discovery Algorithm

During the search phase, the Rover maps the area using a discovery algorithm called Frontier-Based exploration [11]. This algorithm classifies boundaries between open space and unexplored space as frontiers by looking at the costmap from the Rover's Lidar data. The idea is then to move towards the closest frontier. While moving, it uses SLAM to scan the nearby unexplored area and updates the map. By continuously doing so, the whole area is explored.

While mapping the area, the Rover also looks for the Distressed Persons using its camera. If it spots a distressed person, it will stop the mapping and start tracking.

5.5 Tracking

Tracking is used in order to identify and continuously estimate the position of a distressed person. The Rover's tracking algorithm is able to distinguish between different distressed persons by colors. This is so that when, e.g., the blue distressed person is being rescued, it will only track that person and not start tracking another person instead.

5.5.1 Location measurements of distressed persons

In order to calculate the location of distressed persons, firstly the depth to the distressed person has to be approximated. In order to incorporate the depth-dimension in the approximated position of a distressed person, a pinhole camera model is employed.[3]

Given the focal length of the camera, f, the pixel height of an observed object in a frame, h^p_{dp}, the pixel height of the image, h^p_I, the height of the camera lens, h_{cl}, and the height of the distressed person in meters, h_{dp}, the distance of the object to the camera can be calculated using

$$d = \frac{f\ h_{dp}\ h^p_I}{h_{cl}\ h^p_{dp}}. \tag{12}$$

To transform the depth of the distressed person from the Rover to a global position, firstly the angle between the local x-axis of the Rover and the distressed person, $\theta_{\tilde{x},\tilde{y}}$, is calculated as

$$\theta_{\tilde{x},\tilde{y}} = \frac{\tilde{y}^p F_{\tilde{x},\tilde{y}}}{w^p_I}, \tag{13}$$

where $F_{\tilde{x},\tilde{y}}$ is the field of view in the xy-plane, $\tilde{y}^p$ is local y-coordinates of the distressed person, expressed in pixels and w^p_I is the pixel width of the image.

The global yaw of the Rover, θ_{yaw}, combined with $\theta_{\tilde{x},\tilde{y}}$ transforms to a global yaw of a distressed person, θ_{dp}, in

$$\theta_{dp} = \theta_{yaw} - \theta_{\tilde{x},\tilde{y}}. \tag{14}$$

There is a negative sign in the equation, since $\theta_{\tilde{x},\tilde{y}}$ is negative if the distressed person is to the left of the Rover. This is also illustrated in Figure 9.

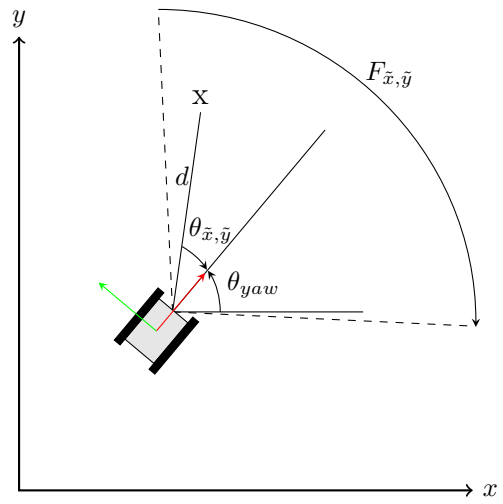

Figure 9: The Figure illustrates how the quantities d, $F_{\tilde{x},\tilde{y}}$, $\theta_{\tilde{x},\tilde{y}}$ and θ_{yaw} are defined. They are used to estimate the position of a distressed person, marked by an X in the Figure. The red arrow emanating from the Rover is the forward coordinate (local x-axis), and the green one represents the left coordinate (local y-axis) in a body-fixed FLU (front-left-up) coordinate system.

The global position of the distressed person is then calculated according to

$$x_{dp} = x + c_x cos(\theta_{yaw}) + dcos(\theta_{dp}), \tag{15}$$

and

$$y_{dp} = y + c_y sin(\theta_{yaw}) + dsin(\theta_{dp}), \tag{16}$$

where x and y is the x- and y-position of the Rover, c_x and c_y is the offset of the camera from the center of the rover in local x- and y-position, x_{dp} and y_{dp} is the estimated global x- and y-position of the distressed person. This estimated position is then used as a measurement update in the extended Kalman filter as explained in Section 3.3.

5.5.2 Following distressed persons

The estimated position is used to follow a distressed person. If the distressed person is visible in the camera image, Figure 10 explains the different vectors used to find a goal point to follow.

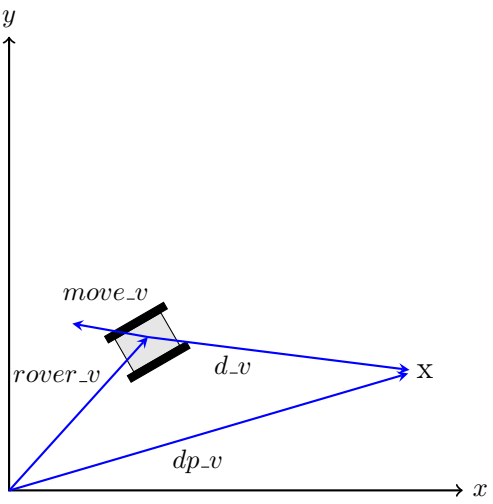

Figure 10: The Figure illustrates the vectors used to calculate the position which the Rover should move during tracking, given an estimated state of the Distressed Person, marked by an X in the Figure. $\theta_{yaw,dp}$ is the yaw angle of the Distressed Person in relation to the global coordinate system. The vectors $rover\_v$ and dp_v represent the global position of the rover and Distressed Person respectively. The vector d_v represents the relative position of the Distressed Person in relation to the Rover. Finally, $move\_v$, is the vector that is calculated using the other quantities, and the end-point of it is used as the next position for the Rover to move to.

As seen in the image, there are four vectors used. The vector from the origin to the Rover is represented by $rover\_v = \begin{bmatrix} x \\ y \end{bmatrix}$. The vector $dp\_v = \begin{bmatrix} x_{dp} \\ y_{dp} \end{bmatrix}$ is the estimated position received from the extended Kalman filter from the origin. The vector from the Rover to the distressed person is denoted $d\_v = \begin{bmatrix} dx \\ dy \end{bmatrix}$. Finally, the vector from the Rover to the goal pose during tracking is denoted $move\_v = \begin{bmatrix} x_{move} \\ y_{move} \end{bmatrix}$, and is defined by

$$\begin{bmatrix} dx \\ dy \end{bmatrix} = \begin{bmatrix} x_{dp} \\ y_{dp} \end{bmatrix} - \begin{bmatrix} x \\ y \end{bmatrix}, \tag{17}$$

and

$$\begin{bmatrix} x_{move} \\ y_{move} \end{bmatrix} = \begin{bmatrix} dx \\ dy \end{bmatrix} (1 - \frac{d_w}{\| \begin{bmatrix} dx \\ dy \end{bmatrix} \|}). \tag{18}$$

The parameter d_w in (18) is the wanted distance from the Rover to the distressed person. The goal pose sent to the Rover's motion planner is calculated with

$$x_{goal} = x + x_{move}, \tag{19}$$

and

$$y_{goal} = y + y_{move}. \tag{20}$$

When the Rover is too close to the distressed person, this will send a goal pose behind the Rover and since the Rover needs to have the distressed person in the image, the Rover

needs to be able to reverse. Since the Rover wants to rotate to the goal pose in other parts of the mission, e.g., in the discovery phase, some parameters in the controller are changed when the distressed person is too close to allow the Rover to reverse.

If the distressed person is lost from the image, for example when driving around a corner, the extended Kalman filter will still update the position of the distressed person for 12 seconds. Since the controller uses velocity scaling on approach to the goal pose, the Rover will move quicker towards a goal further away than to a goal very close. Thus, in order to catch up to the distressed person quickly, the goal position sent is changed to be the estimated position of the distressed person when it is outside of view.

5.6 Modes of Operation

There are two modes of operations for the Rover, manual and autonomous mode. On the RC controller there is a switch which is used to switch between the modes. The Arduino receives the RC-signals and transmits them to the master_com node, which decides if the Rover should drive autonomously or manually.

5.6.1 Manual Mode

If the Rover is in manual mode, the master_com node translates the RC signals into motor signals.

5.6.2 Autonomous Mode

In autonomous mode the Rover is driven autonomously, and the motor signals depend on controllers, planners, and search algorithms.

6 UAV

The overview of the UAV subsystem is visualized by the UAV launch file in Figure 11. When running the launch file, it initiates the UAV offboard controller, the camera and the image processing nodes, and the node for the motion planner. If the UAV is to be simulated, the launch file spawns the UAV in Gazebo and sends a command to the Base Station to start the simulation. If the UAV is running on HW, the microRTPS agent is initiated.

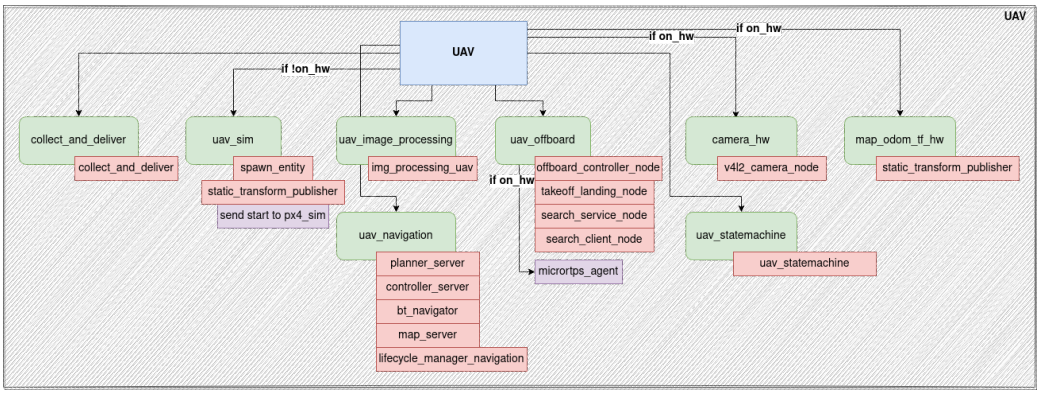

Figure 11: Schematic figure of the launch file for the UAV.

6.1 Hardware

The UAV is a quadcopter based on the *Lumenier QAV-R 2* frame with components listed in Section 6.1.1. An overview of the system can be seen in Figure 12, where both HW and SW components are illustrated.

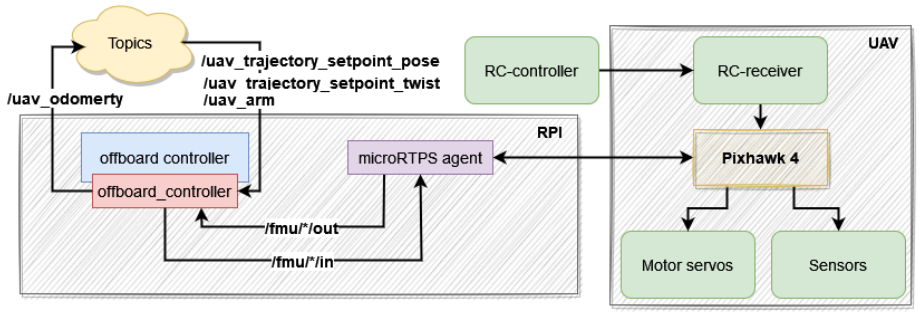

Figure 12: Overview of the UAV hardware.

6.1.1 Hardware Components

- **Pixhawk 4** - Flight controller with internal IMU and barometer
- **Raspberry Pi 4b** - Onboard computer with Wi-Fi
- **RPi camera** - Camera connected to the RPi

- **RC-receiver** - Receiver for manual flight
- **Motors and ESC** - Propeller motors and Electronic Speed Controller
- **Battery** - 11.1V Lithium Polymer (LiPo) battery
- **RC-controller** - Controller connected to the UAV for manual flight

6.1.2 Safety Features

The UAV has some safety features to minimize the risk of crashes and bodily injury. The Pixhawk has some built-in fail-safes that, when triggered, will land the UAV immediately. The fail-safes used are the following: An RC loss fail-safe that activates when the RC-signal to the manual controller is lost, and a fail-safe that activates when the Pixhawk loses connection to the RPi.

In addition to these fail-safes, a fail-safe for losing Qualisys positioning data is implemented. The UAV will activate the fail-safe used when loosing connection to the RPi if it has not received positioning data in the last 2 seconds. Then it will take an additional two seconds to land after the fail-safe is called.

The RC-controller also has a kill switch that stops all rotors, and the user is able to switch to manual control at any time.

6.2 Coordinate frames

The UAV has three different coordinate frames: Map, Odom, and Base Link. Map is the global frame that is defined by the coordinate system in Gazebo on SW and Qualisys on HW. The odom coordinate frame is defined differently in HW and SW. In the simulation, the odom frame has its origin in the starting position of the UAV, and its x-axis points north since the simulated UAV uses a compass to estimate its yaw. When starting the system on HW the odom frame is instead defined exactly like the map frame. Usually, the Pixhawk sets its take-off position as the origin, but since it receives positioning data from Qualisys it will use the same origin as the map. The third frame, base link, is defined to have its origin in the current position of the UAV with its x-axis pointing forwards. Coordinates can be transformed between these different frames using the "tf2" package. When transforming the coordinates, the transform between the frames needs to be looked up. Then the translation between the different frames on each axis can be subtracted from the coordinates to get the new coordinates in the desired frame. The different frames are illustrated in Figure 13.

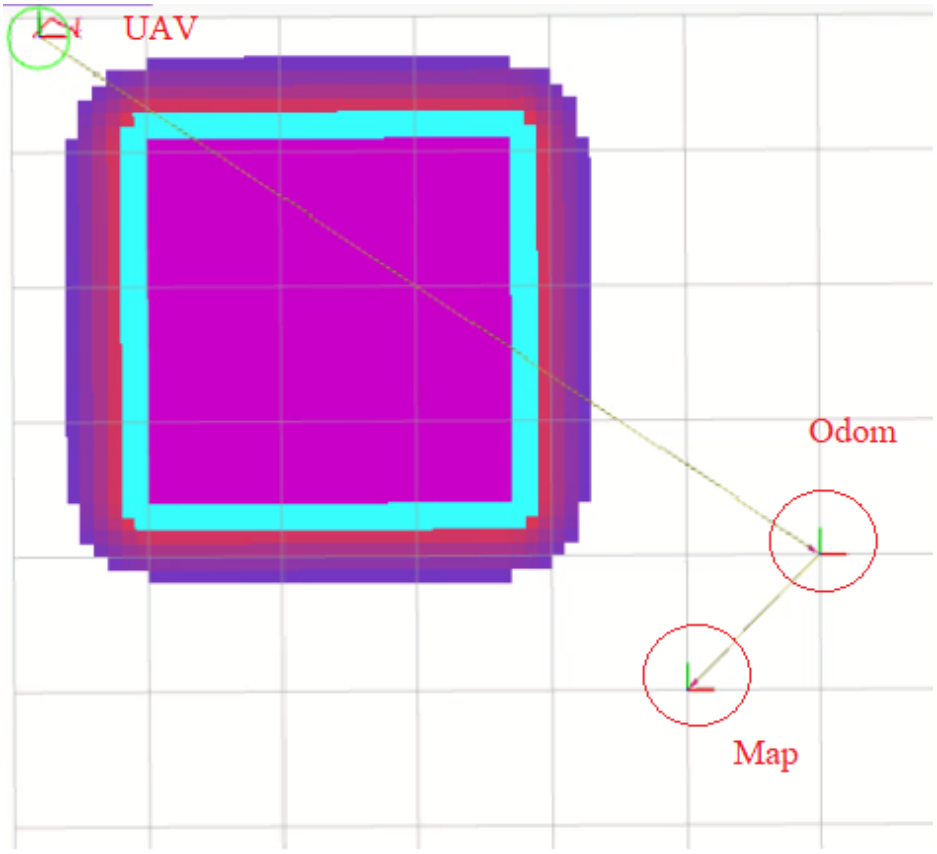

Figure 13: UAV, odom and map frames illustrated in Rviz2, enclosed in circles.

6.3 Onboard Communication

The communication between the RPi and the Pixhawk is carried out over a microRTPS bridge. This bridge converts ROS2 topics to uORB topics the Pixhawk can interpret. An agent on the RPi receives data from a ROS2 topic, converts it, and sends it to a client on the Pixhawk. If data is sent from the Pixhawk instead, it works the other way around and the agent publishes the received data on a ROS2 topic.

A ROS2 node called "offboard_control" acts as an intermediary between the agent and the rest of the system. The node needs to publish a position or velocity command to the Pixhawk with a frequency of at least 2 Hz, otherwise, a fail-safe will be initiated, and the UAV will land. The node will by default publish these commands with a frequency of 10 Hz.

6.4 Position and Velocity Commands

To control the UAV, a ROS2 "Pose" message containing a desired position and orientation will be sent. This message will be sent to the "offboard_control" node on the topic "uav_trajectory_setpoint_pose" and then sent to the Pixhawk. The UAV will then fly to the position and, with its internal PID controllers, hold that position. This will allow the user to send simple commands with a position to the UAV, which stabilizes around the point by itself.

An alternative to controlling the position of the UAV is to control its velocity. By sending a "Twist" message on the topic "uav_trajectory_setpoint_twist", the UAV will hold a specified velocity instead. Using the velocity control, the UAV will use its internal PID controllers to hold the velocity. Important to note is that setting the velocity to zero does not guarantee that the UAV stays in the same position, since the position is not part of the feedback loop when controlling the velocity. Both the position and the velocity must be expressed in the odom frame.

6.5 Position Feedback

The internal controllers in the Pixhawk need external positioning to close the feedback loop. In the simulation, a GPS module is mounted to the UAV, which provides it with position data. However, on the HW, the UAV depends on the Qualisys positioning system in Visionen.

The Pixhawk has support for receiving a "VehicleMocapOdometry" and a "VehicleVisualOdometry" message, each containing position, orientation, velocity, and angular velocity. This data is fused into one of the two available filters on the Pixhawk: EKF2 and LPE. EKF2 is an extended kalman filter that estimates attitude, position, and velocity. LPE filter is also an extended kalman filter that estimates position and velocity. To estimate attitude, an additional filter is used with the LPE filter. The EKF2 filter is recommended to use since the LPE filter is no longer supported [1]. To use the motion capture message, the LPE filter needs to be used. However, if the motion capture message is converted to a visual odometry message, the EKF2 filter can be used instead. By sending the motion capture data from Qualisys as a visual odometry message to the Pixhawk the EKF2 filter is used to estimate the position. To utilize the position data, the Pixhawk must receive it at 50Hz. If the frequency is too low, the filter will not fuse the data and the UAV will not hold its position while flying and risks crashing.

The odometry message is labeled to be in an FRD frame that is not aligned with north. It is possible to include covariance in the message, but instead a manual variance of the Qualisys data is set as a local parameter on the Pixhawk.

6.6 Tracking

The UAV, unlike the Rover, does not use a depth approximation. As long as the UAV does not tilt overly much, the camera plane of the UAV will line up with the desired tracking space (the 2D projected plane of the room onto the floor). Furthermore, the depth dimension of the UAV's camera is better represented by the measured height of the UAV.

The bird's eye view, combined with the higher degree of freedom of movement in comparison with the Rover, makes the UAV a better tracker in a free space. However, it is severely limited by the fact that the distressed persons can move into the no-fly zones, in which the UAV cannot enter. The extended Kalman filter will however, if the no-fly zone is not too big, eventually send a position outside the no-fly zone and try to intercept the distressed person.

6.6.1 Location measurements of distressed persons

To measure the location of a distressed person, the distance in local x- and y-axis is calculated. This is done using the height of the UAV and angle between the z-axis and the distressed person's x- and y-position. The angles are illustrated in Figure 14.

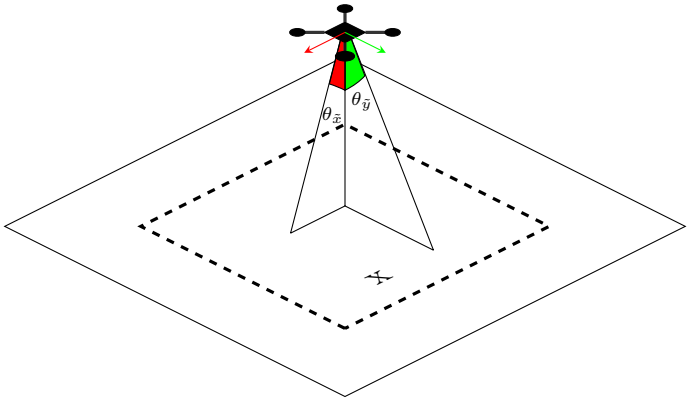

Figure 14: The Figure shows the view of the UAV as it passes over a distressed person. The distressed person is indicated with the X, the camera view with the dashed rectangle, and the red and green axes represent the front and left directions respectively. The position estimate of the distressed person is constructed with the help of the angles $\theta_{\tilde{x}}$ and $\theta_{\tilde{y}}$ in the figure.

The angles are calculated as

$$\theta_{\tilde{y}} = \frac{\tilde{y}^p F_{\tilde{y}}}{w_I^p}, \tag{21}$$

and

$$\theta_{\tilde{x}} = \frac{\tilde{x}^p F_{\tilde{x}}}{h_I^p}, \tag{22}$$

where $\theta_{\tilde{y}}$ and $\theta_{\tilde{x}}$ are the two angles, $\tilde{y}^p$ and $\tilde{x}^p$ are the pixel offset from the center of the image to the distressed person in the local y-axis and local x-axis respectively, $F_{\tilde{y}}$ and $F_{\tilde{x}}$ are the field of view in local x- and y-direction, w_I^p and h_I^p are the total pixel width and pixel height of the image.

These angles are then used together with the height of the UAV, z, to calculate the displacement along the UAV's local x- and y-axis in

$$\tilde{y} = tan(\theta_{\tilde{y}}) \cdot z, \tag{23}$$

and

$$\tilde{x} = tan(\theta_{\tilde{x}}) \cdot z. \tag{24}$$

The global position of a distressed person is calculated as

$$x_{dp} = x + \tilde{x}cos(\theta_{yaw}) - \tilde{y}sin(\theta_{yaw}), \tag{25}$$

and

$$y_{dp} = y + \tilde{x}sin(\theta_{yaw}) + \tilde{y}cos(\theta_{yaw}), \tag{26}$$

where x and y are the global x- and y-position of the UAV and θ_{yaw} is the global yaw of the UAV. The trigonometry used to transform the local positions to global positions is illustrated in Figure 15. The global position is the used as a measurement update in the extended Kalman filter.

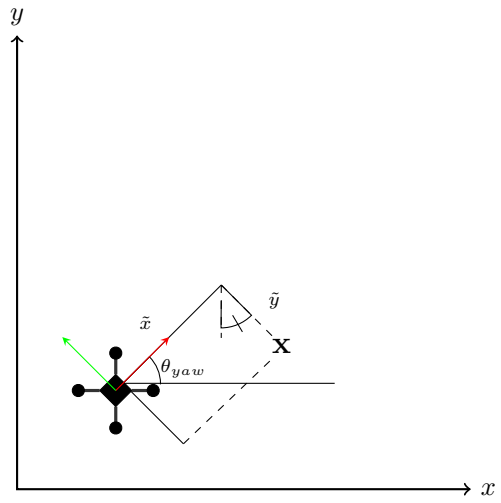

Figure 15: The Figure illustrates how the global yaw of the UAV, θ_{yaw}, is used to estimate the global position of a distressed person, marked as X in the Figure, with the help of the displacement in along local x- and y-axis fo the UAV. The local x- and y-axis is illustrated as a red and green arrow, respectively.

6.6.2 Following distressed persons

Different from the Rover, the UAV does not need to consider the distance from a distressed person since it flies at a constant height. Also, it does not need to consider flying around corners since there are not any obstacles in the air besides the no-fly zones. Therefore, the estimated position of the distressed person received from the extended Kalman filter is sent directly to the UAV's motion planner according

$$x_{goal} = x_{dp}, \tag{27}$$

and

$$y_{goal} = y_{dp}. \tag{28}$$

6.7 Motion Planner

The motion planner is implemented in Nav2. It takes in a static map, including no-fly zones, which is discussed in Section 6.7.2. The UAV flies at a constant altitude of 2 m. Therefore, the motion planner only plans paths in two dimensions, and not in altitude, during normal flight. When a goal is set, Nav2´s built in Dijkstra algorithm is used to calculate the optimal path to the goal, given the costmap.

6.7.1 Controller

To follow the path given by the motion planner, a variant of a pure pursuit controller has been written. Nav2 has a built-in pure pursuit controller, however it is designed for ground vehicles with wheels. The UAV can move in all directions, and therefore a new controller has been written for it.

The controller works as a traditional pure pursuit controller, in the sense that it has a look-ahead distance. This is a parameter in the controller, which represents a distance from the UAV. It chooses the point on the path closest to the look-ahead distance that is in the direction of the UAV:s motion. Then the UAV and tries to navigate to it. A traditional pure pursuit controller would send a desired rotation and velocity, so that the vehicle can turn and reach the point. But because the UAV can move in all directions, no rotation is calculated. Instead, a desired lateral and longitudinal velocity is calculated to reach the look-ahead point.

6.7.2 No-fly Zones

No-fly zones are areas of the map, where the UAV is not allowed to fly. This is meant to represent no-fly zones in real life, like zones around military areas or airports. These are specified to the UAV as .png-files, where the no-fly zones are drawn as a map. The .png-file is specified as a static map to Nav2. Nav2 will calculate a costmap for this map and will use this in the navigation. This allows the motion planner to plan paths around the no-fly zones.

The algorithm that calculates the search pattern must also avoid placing waypoints inside the no-fly zones. This is done with the help of the image processing tool OpenCV. The .png-file is read when creating the search pattern. When a waypoint is to be set, it simply checks if that position corresponds to a black pixel in the .png-file. If the pixel is black, another attempt is made one meter away from the original attempt. This goes on until a valid position is found, and a correct waypoint is set.

6.7.3 Safety Distance

The UAV will fly at a constant height of 2 meters. Thus, during normal flight, the UAV will not come close to any distressed persons. When delivering supplies, the UAV will only descend to an altitude of 0.7 meters to keep a safety distance of about 0.5 meters to the distressed person. When landing the UAV there is no safety feature that prevents it from landing too close to a distressed person.

6.8 UAV Search Algorithm

While the Rover maps the area, the UAV runs a search pattern over the entire environment, searching for distressed persons. This pattern follows a simple S shape that scans the entire environment, see Figure 16.

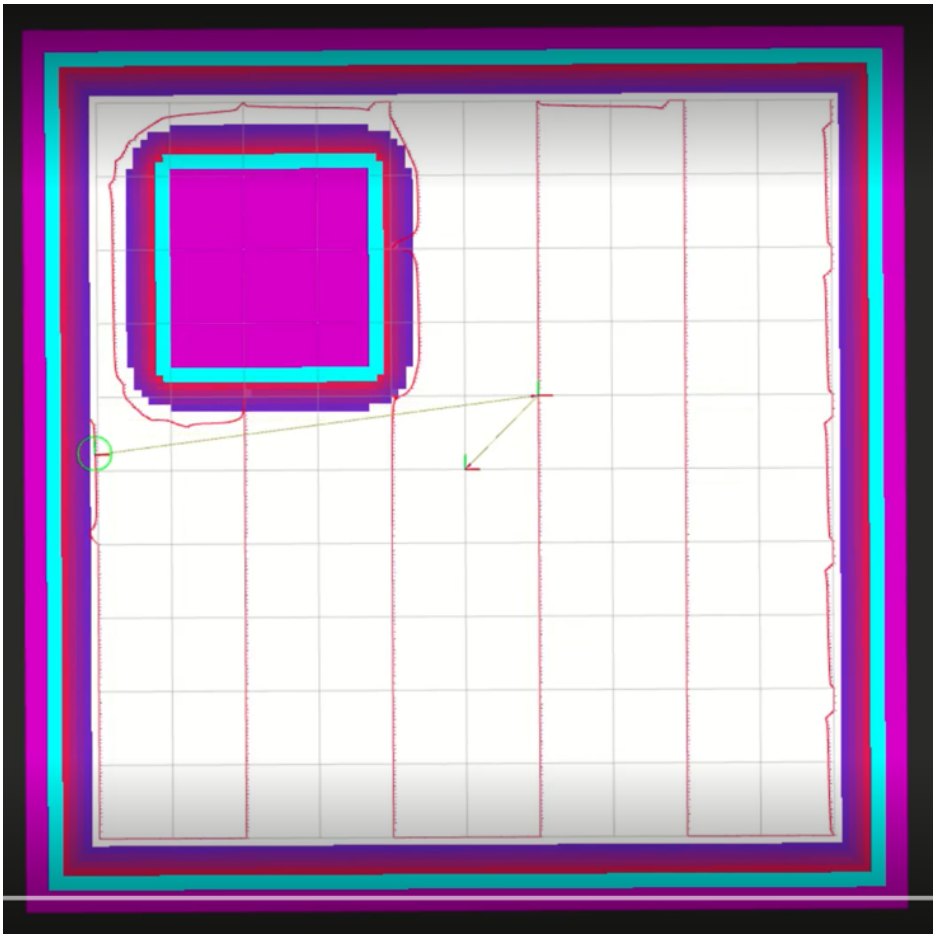

Figure 16: This figure shows the Rviz2 window for the UAV while it searches for distressed person. The red lines are the planned path and the coordinate system with the green circle is the UAV. The pink areas are obstacles; this includes the walls of the visionen and the No-fly zone. The colored areas around obstacles are inflation layers, these are layers that the motion planner avoids planning inside of.

6.9 Take-off and Landing Services

ROS2 services are implemented for taking off and landing the UAV. These services receive a start command, execute the action, and send back a result indicating that the UAV has finished its action.

The take-off service receives a take-off command and the UAV is armed, set into offboard mode, and starts flying up to an altitude of two meters, where it hovers. When the UAV reaches the correct altitude, the service sends a message back confirming the take-off is completed. The service utilizes the UAV's current position and adds two meters to the z component and publishes the new position to the UAV.

The landing service receives the global position of the landing site when it is called. The UAV navigates to a position above the landing site using the motion planner and starts to descend to about 0.1 meters above the ground, where the RPi connection fail-safe is started to ensure a landing. When the landing is complete, it sends a confirmation message that it has landed. The reason the fail-safe is used instead of setting the ground

level as the target when landing is that if the UAV does not reach the requested height even though it has reached the floor it may try to correct its position causing unexpected behavior. This problem was observed in the simulation, but has not been tested on HW. The method to fly vertically in the take-off is reused in the landing service.

6.10 Collect and Deliver Supplies

The UAV collects and delivers supplies to the distressed persons while the Rover tracks them. The UAV picks up supplies at a given supply point, whereupon it returns to the distressed person to deliver the supplies. Since the distressed person might be moving, the UAV will fly to the location that the Rover estimates the person to be at. When the UAV reaches the delivery position it descends to the height of one meter, hovers one second and then climbs back to its original height.

The collection of supplies is implemented as a ROS2 action which navigates the UAV to the supplies. The supplies location has a default value but can also be updated by publishing the desired location on a topic. When we have arrived to the supplies a request will be sent to a service which executes a supply signal. A supply signal is a motion that indicates that it picks up supplies. In a fully developed system this would need to include some form of hook that picks up the supplies. However, this motion has been simplified to only involve a drop in altitude followed by a rise to the original height.

The delivery of supplies is also implemented as a ROS2 action which navigates to the found distressed person. By subscribing to the topic */rover1/dp_state* the UAV gets continuously updates of the estimated position of the found distressed person and for every update the action changes the navigation goal. When the UAV is within a 0.2 m radius from the distressed person, the action will send a service request to perform a supply signal, similar to the collection of supplies.

The motion indicating collection and delivery of supplies is implemented as a service in the same way as the take-off and landing, as described in Section 6.9.

6.11 Modes of Operation

The UAV has two main modes of operation: Manual and Autonomous. When the UAV is in manual mode, it is controlled by the RC-controller, and in autonomous mode, the UAV is controlled by the onboard RPi.

6.11.1 Manual Mode

There are a few different types of manual modes: Manual/Stabilized, Altitude, and Position. The manual/stabilized mode lets the user fly the UAV fully manually. Altitude mode stabilizes the UAV at an altitude using the barometer to measure height. The position mode utilizes external positioning to hold a position that the user can change using the controller. The position mode will be used to verify that the UAV utilizes the position given by the Qualisys positioning system. When flying the UAV autonomously, the user can at any time switch to a manual mode to take over control. If the positioning data from Qualisys is poor, altitude mode should not be used when taking control of the UAV since the altitude estimation mainly uses external positioning. Instead, manual/stabilized mode is recommended since they give the user full control of the throttle.

6.11.2 Autonomous Mode

The autonomous mode, called offboard mode, implies that the UAV is fully controlled by the onboard RPi. If the UAV does not receive data from the RPi with at least 2 Hz, the UAV will enter fail-safe mode.

7 Simulation

An important part of the project is the simulation environment. The system uses SIL, which enables testing in simulation in contrast to only testing on hardware. This is crucial for the testing because of the limited testing-time, and minimizes the risk of damaging the hardware. The simulations are handled in the Base Station system.

7.1 Gazebo11

Gazebo11 is the simulation software that is used when running the project on SW. It is integrated into ROS2 and can handle communication via ROS2 topics. It can simulate the world and multiple robots, as well as handle the physics in the simulation.

7.2 Simulation of the Robots

Generally, when simulating robots in Gazebo11, SDF-files are used. An SDF-file includes the information that Gazebo11 needs to simulate the robots accurately, such as, dimensions, sensors, materials etc. The SDF-files for the UAV and the Rover have slightly different origins in how they are created and where they are obtained from.

7.2.1 Rover

The Rover's SDF-file is created using a URDF-file. The URDF-file is where the Rover model is described, including its physical measurements and the sensors used with associated positions. This is then used to create the SDF-file. The URDF-file was created to describe the project's Rover as closely as possible, and used the dimensions of the specific Rover used in this project.

7.2.2 UAV

The UAV has a different method for generating the SDF-file. It is generated when compiling the PX4-Autopilot git project. The SDF-file defines the structure of the UAV, including its sensors. Furthermore, the git project contains a software program that simulates a PX4 flight controller.

The simulated PX4 is able to communicate with the microRTPS bridge running on the simulated UAV. However, the simulated drone has different dimensions than the actual drone used in this project. This is because the submodule PX4-Autopilot, with the SIL simulation of the Pixhawk, contains a standard UAV with all the sensors and flight controller already in the model. Creating a UAV from scratch that has the same dimensions and structure would take time, and after analyzing the benefits of creating a new model, it was deemed to not be worth the effort.

The simulated UAV is slightly larger, but this should not have any major impact on the testing using the simulation, since the flight controller will control the different sizes of UAV similarly. Since the main purpose of the project is to execute a mission and not to look into controlling the UAV, the fact that the UAV will behave slightly different between simulation and real tests can be overlooked.

A camera is added to the UAV model and is defined as an SDF-file that imports the base model and adds a model of an RPi-camera. This file is added to the same folder as the base model as a volume in the docker-compose file.

7.3 Simulation of the Map

The tests in the project have been conducted in Visionen. Therefore, a simulation environment which is designed to look like Visionen has been created. The map has a white floor and walls with similar dimensions as Visionen. In the simulation environment, it is possible to add boxes as obstacles and No-fly zones.

7.4 Simulation of Distressed Persons

Similarly to the Rover, simulation of distressed persons is done with a URDF-file. It is possible to simulate a blue and a yellow distressed person, both with their own URDF-file, differing in colour. The URDF-file is then made into a SDF-file when the distressed person is simulated in Gazebo. When spawning a distressed person, it can be told to move in a circle or along a line. This is done in a launch file, sending a twist message to the distressed persons /cmd_vel topic. The launch file for the distressed persons is structured as described in Figure 17.

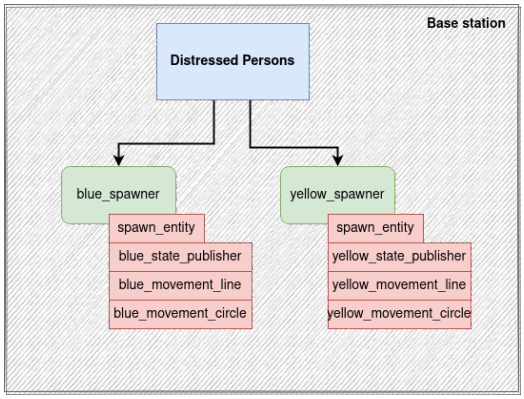

Figure 17: Schematic view of the Distressed persons' launch file.

8 Further Work

This section will discuss future work, that would need to be implemented for a more robust, and more functional system. These are mainly things that have not been implemented because of lack of time in the project.

8.1 Base Station

A major issue with the current system is the frequent crashes upon starting the system. This is most likely because everything starts at the same time causing the cpu to overload. Many nodes depend on other nodes, so before the other node has started properly many warning messages will be displayed in the terminal. The solution to this issue is to create a start-up sequence where the nodes are started in a predefined order, where each node waits until the previous node has started properly.

The current system suffers from additional performance issues. Even if the system launches successfully the simulation runs with a real-time factor of 30-40% on the Saab supplied computers. A real-time factor of 100% is preferable to decrease time spent on testing the system in simulation. To increase the performance, the system would need more efficient code. A major user of computational effort is the image tracking. In the current system, the image recognition runs at 60 Hz if it instead ran at 20 Hz, the performance of the entire system would increase significantly.

8.1.1 State Machine

The performance of the state machine and the mission time could be increased by saving the last known location of a distressed person, even when that person is not currently being tracked. For example, if the blue distressed person is being tracked and the UAV flies to collect the supplies and spots the yellow distressed person along the way, it could save the location of the yellow person. Once the supplies have been delivered, and the blue person has been saved, the robots can immediately go to the location where the yellow person was spotted, and search the surrounding area. This could also be implemented on the Rover, so that when it is tracking the blue person and spots the yellow person, it saves that location.

One issue with the state machine is during the Collect Supplies state. It assumes that the Rover will always track successfully. The system uses an Extended Kalman Filter to find a distressed person, if the Rover loses track of it. However, upon testing it is clear that this is not always successful. Especially if the person moves around a corner far away from the Rover. So, if the Rover is tracking the person and the UAV is collecting supplies, and the Rover loses track of the distressed person, there is no way for the system to find it again. To make the system more robust the state machine would have to make sure that the tracking is performed successfully and if not, it would need a "backup plan". This "backup plan" could for example be that the robots return to the search phase, or that they have a special search state where it searches the last known location of the distressed person.

8.2 Rover

The applied discovery algorithm in Section 5.4 has shown weakness in some environments/worlds. From the initial costmap generated by SLAM the algorithm sometimes cannot find frontiers. This has been solved by sending an initial point (1,1) to the Rover to get it moving. During this movement it will hopefully scan larger portions of the environment and generate frontiers. This solution is however not flawless. If there were to be an obstacle at that point, it would crash and most likely get stuck. A more robust solution is necessary for this issue.

When the world is fully explored the Rover will stop searching. This is because it doesn't have a second search pattern. In the Design Specification[8] we proposed one possible search pattern which we didn't have time to apply during the project. This search pattern can be used for both the Rover and the UAV once the world is fully explored and we believe this would be a good way for searching for distressed persons once the world is mapped.

8.2.1 Tracking

The way of sending goal poses during tracking needs to be upgraded. This is because the technique, explained in Section 5.5.2, does not take into account the surrounding map when sending a goal pose to the motion planner. If a point is sent into an inflation layer close to a wall, it can result in the Rover getting stuck. Also, when tracking around a corner, the Rover sometimes takes a shortcut and can get stuck to the wall or an inflation layer. The fact that the estimated position of the distressed person is sent, instead of the calculated goal position, when the distressed person is lost from the image helps to follow around corners. However, a better solution that takes the discovered map into account when deciding a goal point would be preferred.

Some tests were made to only use the technique explained in Section 5.5.2 when the Rover is too close to a Distressed Person, and instead send the position of the Distressed Person when the Rover is too far away. This had some improvements when driving around a corner, but also caused some choppy driving since the Rover stopped when coming too close to a distressed person. The worst problem occurred when running together with everything else in a mission. The reason for this is probably that the system could not handle changing parameters in the controller fast enough. This resulted in the Rover not wanting to reverse, which could lead to worse results than what is implemented now.

A final thing to be implemented in the tracking for both the Rover, and the UAV is to create a topic which is published to by the state machine that tells the robots that a Distressed Person is rescued. A variable in the recognizer script is implemented for this, called "color_done". If this parameter is changed to True via a topic, the tracking algorithm will ignore that color as a potential color to track.

8.3 UAV

The Qualisys data loss fail-safe currently implemented only detects complete loss of the position feedback. If the frequency of the data would be lower than needed for proper utilization of the data the system companion computer would not notice and the UAV will risk crashing. It would be a good idea to look at alternative solutions for detecting this, for example measuring the frequency or lowering the time set for the system to wait until landing.

Another problem with the current implementation of the motion capture data is the time-stamp set on the message. Since the Pixhawk and the ROS2 network do not share clock it is not so easy to set the timestamp correctly. At the moment, the timestamp on the message is set to the same timestamp as the last received odometry message from the Pixhawk. The actual timestamp update on the "timesync" topic has a lower frequency than the odometry message and can not be used as it is since the frequency of the motion capture messages is significantly higher. There is also a parameter called "EKF2_EV_DELAY" that defines the delay caused by the message being sent from Qualisys to the Pixhawk that can be used for this.

In the simulation, the odom frame is defined as pointing north resulting in a difference in orientation between the odom and map frame. To transform between these frames this orientation needs to be taken into consideration and this is not done in the current implementation. It has been possible to work around this problem, but this should be fixed to ensure it does not cause problems in the future.

When starting the mission on HW a new problem emerged. When both agents are running (or at least trying to run as the Rover did not successfully launch everything on HW) the motion capture data is received slower and the frequency is less consistent. This may be caused by too many messages being transmitted on the ROS2 network at the same time which can delay the messages from Qualisys. A possible solution to this could be a ROS2 domain ID. Using different ID:s, the different parts of the system only have access to the messages sent on its ID. If some nodes can be on two different domains these can send messages between the different subsystems and hopefully solve the problem.

References

[1] PX4 Autopilot. *Switching State Estimator*. https://docs.px4.io/main/en/advanced/switching_state_estimators.html. 2020-2-12. (Visited on 12/19/2022).

[2] Gary R. Bradski. *Computer Vision Face Tracking For Use in a Perceptual User Interface*. 1998.

[3] D. Burschka and G. Hager. "Vision-based control of mobile robots". In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*. Vol. 2. 2001, 1707–1713 vol.2. DOI: 10.1109/ROBOT.2001.932857.

[4] Weipeng Guan et al. "Visible Light Dynamic Positioning Method Using Improved Camshift-Kalman Algorithm". In: *IEEE Photonics Journal* 11.6 (2019), pp. 1–22. DOI: 10.1109/JPHOT.2019.2944080.

[5] Fredrik Gustafsson. *Statistical Sensor Fusion*. 3rd ed. Studentlitteratur AB, 2018, pp. 195–208.

[6] Ola Johansson. *mocap4ros_qualisys*. https://github.com/SkogsTomte/mocap4ros2_qualisys. 2022-11-14. (Visited on 12/07/2022).

[7] Kurt Konolige et al. "Efficient Sparse Pose Adjustment for 2D mapping". In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2010, pp. 22–29. DOI: 10.1109/IROS.2010.5649043.

[8] Operation Watch Landlubber with editor Rickard Wretlind. *Desing specification Search and Rescue - land*. https://tsrt10.gitlab-pages.liu.se/2022/sbd/documents/TSRT10_Design_Specification.pdf. Linköping, Sweden, 2022.

[9] Operation Watch Landlubber with editor Rickard Wretlind. *Project Plan Search and Rescue - land*. https://tsrt10.gitlab-pages.liu.se/2022/sbd/documents/TSRT10_Project_Plan.pdf. Linköping, Sweden, 2022.

[10] slam_karto. *slam_karto - ROS Wrapper and Node for OpenKarto*. http://wiki.ros.org/slam_karto. 2019. (Visited on 10/03/2022).

[11] Brian Yamauchi. *A Frontier-Based Approach for Autonomous Exploration*. Naval Research Laboratory Washington, DC 20375-5337, 1997. DOI: 10.1109/CIRA.1997.613851.