ARSI Deliverable D26.6

MAV Prototype

Table of contents

1. Introduction and scope 3

1.1 Objective and scope 3
1.2 Structure of the document 3
1.3 References 3
1.4 Acronyms and abbreviations 4

2. Hardware design 5

2.1 Onboard hardware 5
2.2 Landing gear 9
2.3 Motor protections 11
2.4 Dust & Water resistance 12
2.5 Wi-Fi router 13

3. Software design 14

3.1 Localization 15
3.2 Onboard navigation 20
3.3 Mission execution 24
3.4 Communications 26

1. Introduction and scope

1.1 Objective and scope

In this document we present the hardware and software designs of the ARSI MAV platform at the end of the phase II of the ECHORD++ Sewer Inspection PDTI. This is the platform that will be demonstrated at the phase II evaluation in the Mercado del Borne area of Barcelona in October.

The current platform is a combination of the system design developed in phase I and changes motivated by the lessons learned from our numerous tests throughout phase II, both in our laboratory and in the sewers in Barcelona.

1.2 Structure of the document

In section 2 we present the current hardware design on the ARSI MAV, which was already developed in large part on the phase I prototype. We give a detailed description of all onboard sensors and electronics, and the motivation behind changes in the sensor payload proposed in phase I.

The MAV airframe itself is included under hardware design, to describe how the various issues flagged after the phase I evaluation were addressed, in particular our platform's ability to withstand soft and hard collisions, and the ubiquitous humidity of the sewer tunnels.

In section 3 we focus on software design, which was almost entirely implemented during phase II. Our core task in this phase was to develop a real-time onboard navigation system enabling the MAV to fly safely and efficiently in the sewer network, within the limits specified in the Challenge Brief [1], but without any restrictions on the topology or configuration of the sewer tunnels themselves.

Note that there is significant overlap between the present document and ARSI D26.5 [4]. We did our best to avoid duplicating information, and referenced D26.5 whenever needed. As with all R&D projects, laboratory tests and field trials are key to develop experience and turn a theoretical design into a functioning prototype. A detailed description of the numerous tests conducted throughout phase II is given in D26.7 [5] along with test results which are also referenced throughout this document.

1.3 References

[1] Challenge Brief — Robots for the inspection and clearance of sewer networks
[2] ARSI D26.1 — Operation requirements and system design
[3] ARSI D26.4 — Operational procedures and sewer inspection service
[4] ARSI D26.5 — Prototype for sewer inspection
[5] ARSI D26.7 — MAV platform verification for sewer inspections requirements
[6] ARSI D27.8 — Autonomous navigation and data recording
[7] PDTI Sewer — Final evaluation phase I
[8] TeraRanger One performance over water

[9] The dynamic approach to obstacle avoidance
[10] RGBD-based Robot Localization in Sewer Networks
[11] Phase II motor protection tests (video)

1.4 Acronyms and abbreviations

- ARSI: Aerial Robot for Sewer Inspection
- MAV: Micro Aerial Vehicle
- Mavlink: MAV communication protocol
- Mavros: Mavlink-ROS interface
- PID: Proportional-Integral-Derivative controller
- PX4: Pixhawk flight stack
- RGB-D: RGB and Depth camera
- ROS: Robotic Operating System
- RVIZ: ROS visualization interface

2. Hardware design

In this section we describe the hardware configuration of our solution at the end of phase II. While the airframe and core onboard sensors remain unchanged from the proposed design, a number of hardware changes to the ARSI MAV were implemented throughout this phase. These changes were motivated by the experience and feedback from phase I, and by the lessons learned from our numerous tests in real sewer environments.

2.1 Onboard hardware

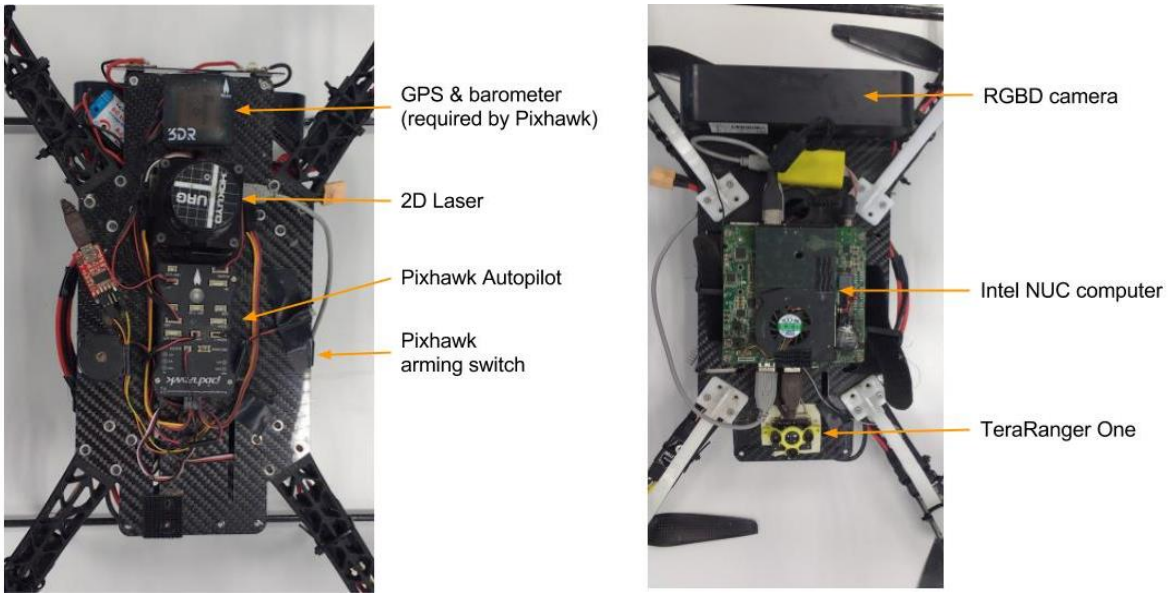

Figure 1: Onboard sensors mounted on the upper (left) and lower (right) parts of the ARSI MAV

Table 1 lists the onboard sensors on the ARSI MAV prototype at the end of phase II of the ECHORD++ Sewer Inspection PDTI. Figure 1 depicts the position of each sensor on the upper and lower parts of the ARSI MAV platform. The most notable changes from the phase I design [1] were the integration of a frontal RGB-D (color and depth) camera, replacing the monocular camera with fisheye lens originally proposed, and the infrared ranger for altitude control. These changes are discussed in the following sections.

Model	***Sensor type***	***Usage***
TeraRanger One	Infrared ranger	• Altitude control
Orbbec Astra Pro	RGB-D camera	• Inspection data collection • 3D reconstruction • Visual odometry • Trajectory planning • Obstacle avoidance
Hokuyo UST-10LX	2D scanning laser	• Trajectory planning • Obstacle avoidance

3DR Pixhawk	MAV Autopilot	• Flight control • Position/velocity estimation
Intel NUC i7	Onboard PC	• Sensor data collection • Real-time trajectory calculation • Mission control • Obstacle avoidance
XLamp XHP70	LED lights	• Illumination for video collection
Custom-made sensor	Gas detector	• Gas monitoring

Table 1: Onboard hardware on ARSI MAV at the end of phase II

RGB-D camera

This important change was motivated both by the phase I evaluation comments and the lessons learned during phase II:

- RGB-D cameras like the Orbbec Astra use infrared sensors to generate dense 3D point clouds on the fly. This 3D information contains crucial information about the surroundings of the drone, which can be used both to generate trajectories and to detect and avoid obstacles. As noted during the phase I evaluation, relying only on the 2D laser for obstacle avoidance would have been risky, given the limited horizontal field of view of the sensor.
- As presented in [4] the ARSI MAV uses visual odometry to estimate its position and velocity. The Orbbec Astra camera enables us to use visual odometry algorithms which take both color and depth information from the camera into account. Since the camera uses an infrared projector to build its depth map, the depth stream is largely independent from the environment or the light conditions, and constitutes a robust additional source of information for visual odometry.
- The RGB-D camera produces valuable structural information of the sewers, which can be used directly for inspection purposes (see Figure 2). It can also be used to help the dense 3D reconstruction generated in post-processing [6].

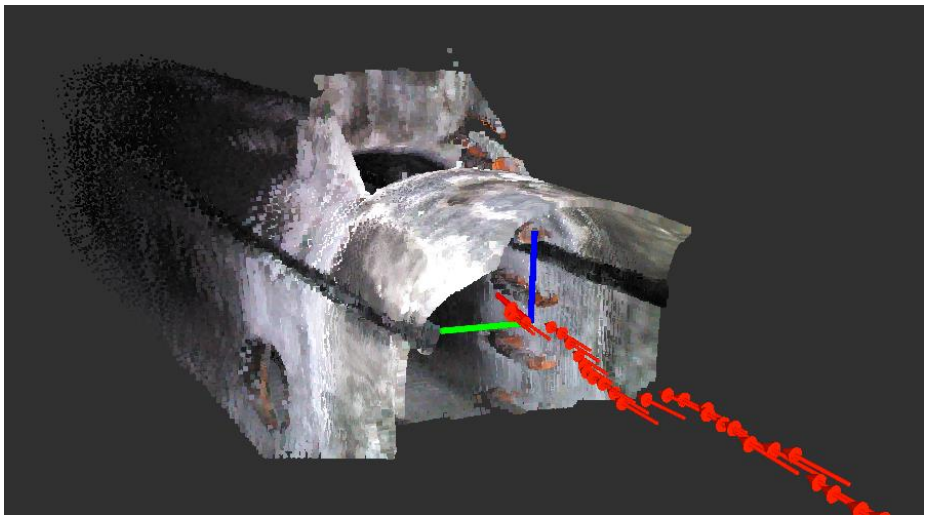

Figure 2: Real-time 3D model produced by the Orbbec Astra camera

The drawback of the RGB-D camera is its significantly increased weight: 160g including USB cable compared with the 45g each for the fisheye cameras selected in phase I. We experimented with RGB-D alternatives such as the Intel RealSense ZR300, which is not only significantly lighter, but also includes a fisheye camera and an IMU. Unfortunately, its short depth range (~2.5m compared to 8m for the Orbbec Astra) did not produce good odometry and would also make it less useful for obstacle avoidance.

Given the time constraints for phase II, we decided to remove the fisheye cameras and use the Orbbec Astra instead as our primary inspection sensor for phase II, postponing our investigation into lighter RGB-D alternatives to phase III (in particular the new RealSense D400, due for release later this year); the objective still being for the final MAV product to carry a fisheye camera for visual inspection purposes.

Infrared ranger

As detailed in section 3.2, the strategy for navigation and obstacle avoidance in the sewers is based on a 2D costmap populated by the laser rangefinder and the RGB-D camera. This means that real-time control requests are only issued in the 2D plane (x and y) as well as yaw, while the altitude remains constant. The altitude is pre-defined in the mission plan, and can also be adjusted during flights from the Mission Control interface (see [6]).

Figure 3: TeraRanger One infrared ranger

Since the odometry is expected to drift over time, it cannot be relied on for accurate altitude control, especially in environments as narrow as sewer tunnels. We decided to use the TeraRanger One infrared ranger as our primary altitude sensor. It is a lightweight (8g plus USB cable) high-frequency high-resolution sensor, well integrated with both ROS and Pixhawk. It is also known to perform well over water [8], which is particularly important for the ARSI MAV platform since it generally flies directly over the central sewage water canal.

As shown in Figure 1 the sensor is mounted underneath the ARSI MAV, facing directly towards the ground. The offset to the Pixhawk unit is recorded in a configuration file, so that the relevant coordinate transforms are applied when range data is used.

Onboard computer

In phase II we replaced the Odroid PX4 used in phase I with an Intel NUC i7. Though lighter, the Odroid PX4 was simply not powerful enough to run all the real-time embedded software required to operate the ARSI MAV whilst collecting inspection data. With a relatively low weight for its performance and a small form ratio, the Intel NUC is widely used in robotics.

Gas detector

In phase I of the project we worked with environmental measurement specialists Envira Sostenible to develop a prototype for a gas detector adapted to an MAV platform, able to measure concentrations of Hydrogen Sulfide (H2S) and Carbon Monoxide (CO). This prototype was demonstrated at the phase I evaluation in July 2016.

Figure 4: Gas detector developed in phase II

In phase II we worked again with Envira Sostenible to develop a second prototype (see Figure 4) able to perform a broader range of measurements, as required by the PDTI Challenge Brief, including Volatile Organic Compounds (VOCs) and Lower Explosive Limit (LEL).

The prototype was developed and calibrated in laboratory; a ROS driver was also developed to log measurements and record them into the ARSI inspection files. Unfortunately, with the integration of the RGBD camera and the NUC computer, the weight margins for the MAV payload became very narrow. As we encountered complex control issues in our initial tests in the sewers, we decided to not risk making the problem worse by overloading the MAV and so the integration of the gas detector was postponed to phase III.

2.2 Landing gear

In this phase we reviewed our landing gear, as the first prototype developed in phase I proved too fragile to withstand repeated landings. We tried to improve on our phase I design while also experimenting with various commercial products.

In our tests we found the DJI Flame Wheel landing gear to be an excellent solution for our platform: it is made of a single part, and crucially it is designed to be flexible in order to absorb shocks. This landing gear is also light (60g), cheap, easy to find in online stores, and directly compatible with our airframe (also from DJI).

Figure 5: DJI Flame Wheel landing gear

In the hundreds of landings that we performed throughout this phase, many of them poorly controlled, we only had to replace landing gear parts twice during violent crashes in the sewers. Even then, they were easily replaced and we could resume testing.

We modified the DJI landing gear in order to increase the landing surface, as the ARSI platform must be able to stand over the sewer's central canal for takeoff and landing. We achieved this by mounting two fiber carbon tubes through the landing gear legs as shown in Figure 6.

The 2 Wi-Fi antennas connected to the Intel NUC onboard PC are also attached to the landing gear. Its shape allows us to orient them towards the back of the platform, where the router is located according our concept of operations [3].

Figure 6: ARSI platform with its modified DJI landing gear

2.3 Motor protections

Like the landing gear, the 3D-printed motor protections used in phase I proved inadequate as they bended upon collision, providing little protection and interfering with the motors.

In this phase we first looked for commercial alternatives, and found a model which provided better protection due to its raised shape (see Figure 7). These protections are cheap, light (16g each), easy to mount, and proved capable during our numerous tests in the sewers. In the narrowest tunnels (80cm), some contact between the MAV and the sewer walls is unavoidable, and these protections proved robust enough to withstand soft shocks. When violent shocks occurred, they provided good protection but would likely break, though they were easily replaced and testing could continue. In summary these protection were a good short-term solution and allowed us to investigate a more robust solution in parallel.

Figure 7: Motor protections used initially in phase II

In phase II a motor protection frame specifically designed for the ARSI MAV was developed. As noted during the phase I evaluation, the two main weaknesses of our protections were 1) that they could only withstand soft impacts against the sewer walls, and 2) that they would bend upon contact, interfering with the propellers and the motors.

Figure 8: ARSI MAV motor protection

The ARSI MAV protection (see Figure 8) is designed around a frame of 5mm fiber carbon tubes rigidly attached to the MAV body using 3D-printed plastic brackets. Two thinner fiber carbon (3mm) tubes are bended around the motors and attached to the frame using 3D-printed connectors, providing 360 degrees protection to the ARSI MAV. This design ensures that no part of the protection can interfere with the motors upon impact, since the main frame is fully rigid, and the flexible cannot get in the way of the propellers. Our tests have shown that the protection is robust and able to withstand hard impacts without damage to the MAV propellers or motors (see video in [11]).

The motor protection in its current state does not provide protection for impacts from above. This is a weakness that was noted during the phase I evaluation since instability in altitude control can occur, especially when flying through a connection between sewer tunnels of different heights. The ARSI MAV should be able to withstand some contact with the roof to protect its motors and sensors (the laser in particular); but more importantly to avoid contact with the sewers and risk causing sparks in an environment where flamable gases can be found. This is a task that we definitely intend to complete during phase III.

2.4 Dust & Water resistance

As commented by the project evaluators in phase I, any robotic system designed for operation in the sewer systems must withstand a certain degree of exposition to water and dust.

In our experience operating the ARSI drone in the sewers we observed that any water coming in contact with the platform is likely to come from underneath, either from splash caused by air movement when flying at low altitude, or when landing unevenly in the sewer canal.

In this phase we concentrated on protecting the electronics of the sensors mounted underneath the drone using a hydrophobic spray. The spray leaves a thin layer of hydrophobic gel which isolates all the electronics and connectors from water without damaging them. We tested this

spray in I phase by submerging a USB hub in water whilst connected, and it remained fully functional. This gel also keeps dust out to a certain extent. We also mounted a plastic protection underneath the onboard Intel NUC PC, to shield from water and dust. This simple design proved effective so far in our numerous tests in the sewers.

It is worth mentioning that our priority in phase II was to develop the positioning and navigation system for the ARSI drone, as those are by far the most challenging aspects of this project. While we attempted to find a viable initial solution to the water and dust problem for this phase, we intend to investigate it more thoroughly in phase III.

2.5 Wi-Fi router

As described in ARSI D26.4 — Operational procedures and sewer inspection service [3], our concept of operations includes deploying a Wi-Fi router at the location where the ARSI drone takes off, in order to maintain communications throughout the flight and provide mission control and video feedback to the user via our control interface (see [6]).

Figure 9 depicts the router case we developed in phase II to facilitate deployment and operations. The Wi-Fi router is mounted inside a watertight case and connected to the external directional antennas. It is lowered into the sewers through standard manholes using a rope and placed above the water evacuation canal. By simply connecting to the router using an Ethernet cable, users are able to receive information from the ARSI drone during flights.

Figure 9: ARSI Wi-Fi router in its watertight case deployed in a sewer manhole

3. Software design

In this section we describe the design of the onboard software system on the ARSI MAV. First, we give an overview of the general onboard design, and the relationships between the different software modules as well as the communication protocols used. Then we give detailed descriptions of the core elements of the onboard system, namely localization, navigation and execution stacks.

General architecture

Onboard processing on the ARSI MAV is distributed between the Pixhawk autopilot unit and the Intel NUC computer. The Pixhawk runs the state estimators (attitude and position/velocity), as well as the flight controller (cascaded PIDs). The NUC computer runs all the other modules, including sensor drivers, Mavros, the navigation stack (costmap and local planner), as well as the mission execution nodes and the watchdog. Detail about all these modules is given in the following sections.

On the NUC, all drivers and data processing nodes communicate via Robotic Operating System (ROS) protocol, mainly through topics and services. The Pixhawk and the NUC on the other hand exchange data via the Micro Air Vehicle protocol Mavlink. The interface between Mavlink and ROS is abstracted out using the ROS Mavros library on the NUC, so that message conversions are done seamlessly and in a centralized manner. Figure 10 illustrates the exchange of information between the NUC and the Pixhawk.

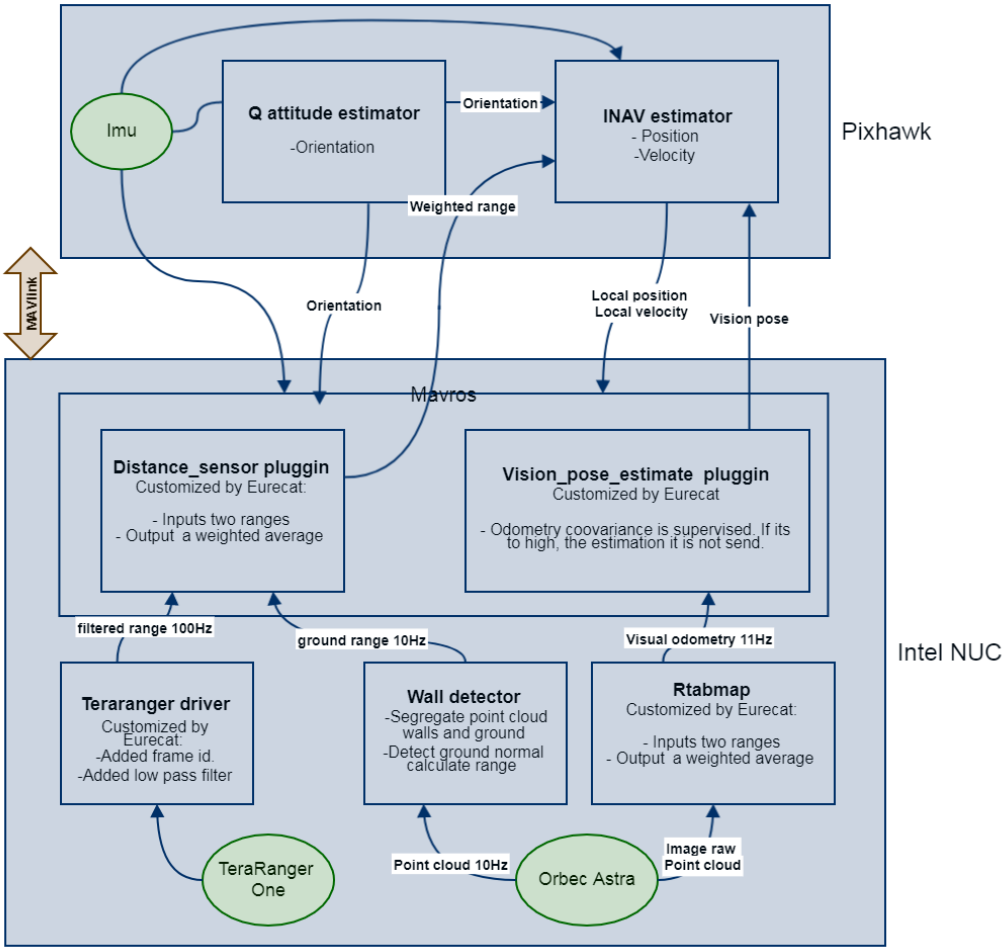

Figure 10: Data exchanged between the NUC computer and the Pixhawk autopilot

Sensor integration

All sensors on the ARSI MAV platform are integrated via ROS. Drivers for the Orbbec Astra, Hokuyo laser and TeraRanger are all provided by the manufacturers as open-source software modules. ROS is so widely used in robotics nowadays that most sensor manufacturers provide ROS support, so that the ARSI system is very much plug and play. For example, the ARSI MAV could be used with another type of RGB-D camera or without any software changes.

Nevertheless, some sensor integration work was required in phase II. For example, early tests with the TeraRanger in the sewers showed that measurements were noisy, likely due to the fact that the drone often flies directly over water flowing in the sewer canal. We added a configurable low-pass filter to the open-source TeraRanger driver [5] with good results.

3.1 Localization

The role of the localization modules in the ARSI system is to produce a real-time estimate of the MAV pose (position and attitude) and velocity, as it carries out inspections in sewer networks. Since no external information source (such as GPS) is directly available in the sewers, we must

rely only on our onboard sensors: the frontal RGB-D camera, the IMU embedded in the Pixhawk autopilot, and the TeraRanger for range measurements.

Figure 11 gives an overview of the localization system implemented on the ARSI MAV. At the center of our localization system is the INAV pose and velocity estimator running on the Pixhawk autopilot unit. The INAV estimator combines visual odometry from the RGBD camera; altitude (ground range) measurements from the TeraRanger; and attitude measurements from the IMU to build a real-time estimate of the MAV pose and velocity. Note that the estimator keeps functioning even if visual odometry or ground ranges become unavailable for short periods of time (e.g. if images become blurred during a quick yaw adjustment).

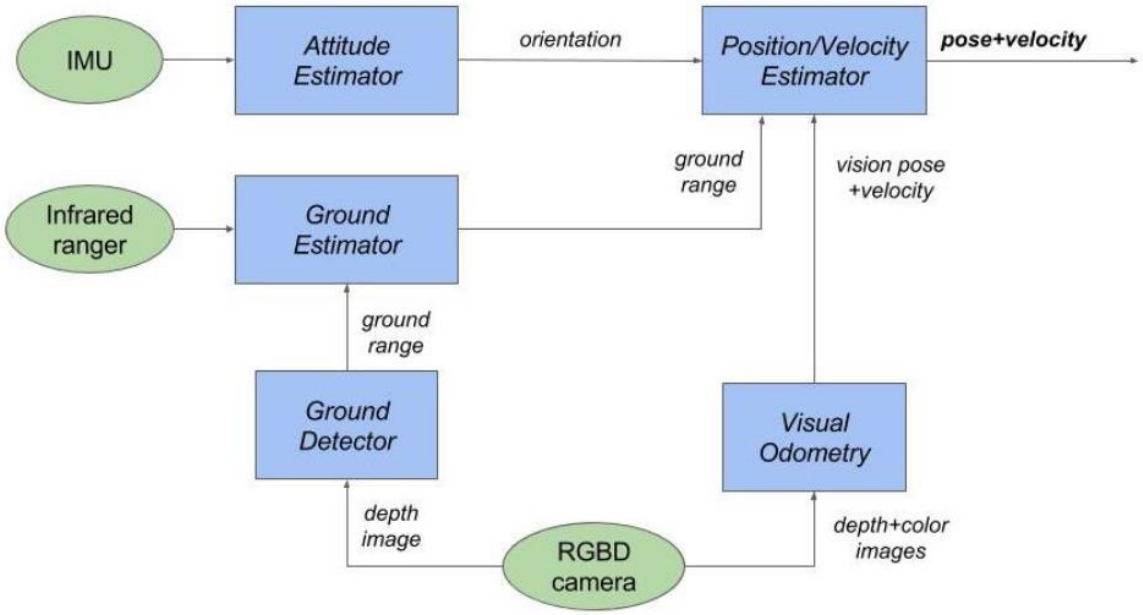

Figure 11: Architecture of the ARSI localization system

Visual odometry

The frontal RGBD camera is used to perform visual odometry. Broadly speaking, visual odometry algorithms identify stable features (e.g. corners, stable regions) in each frame, and match them against previous frames to find correspondences. Using the camera intrinsics and Computer Vision techniques, these correspondences allow estimating the rigid transform between frames, from which the instant velocity is derived. Velocities are integrated over time to produce an estimate of the MAV trajectory during sewer inspections.

As previously discussed, the RGB-D camera was selected in part because it produces an additional depth stream using a pair of infrared sensors (a projector and a receptor). Images from the RGB and infrared cameras are registered against each other (this is done in the camera firmware), meaning that the depth for each pixel in the RGB image is known. Using the camera intrinsics (also provided by the manufacturer), depths are converted into a 3D position for each pixel. Unlike monocular or stereo visual odometry algorithms, RGB-D odometry algorithms use

this additional 3D information to estimate the transform between successive frames more accurately, resulting in a more robust estimation.

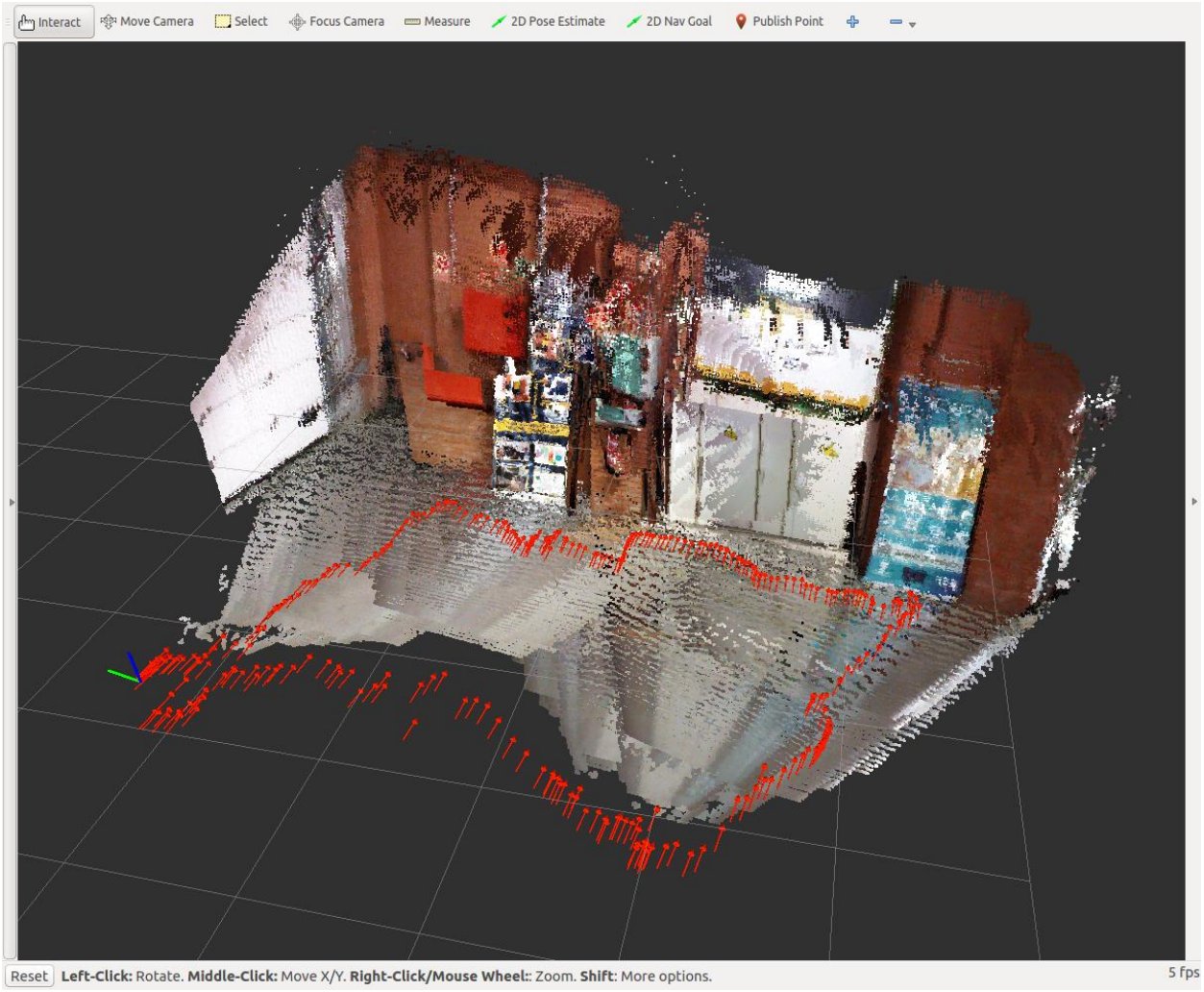

Figure 12: MAV trajectory estimated with RtabMap in our flying area

Broadly speaking, visual odometry algorithms try to solve two separate problems:

1) Calculating the robot trajectory by integrating instant velocity estimates

2) Reducing or bounding the drift caused by the integration of noisy velocities.

Visual odometry algorithms largely differ in their solution to problem 2) while 1) is generally solved using classic Computer Vision techniques all available from open-source libraries such as OpenCV. As already discussed in [4], in this phase II we chose to use the open-source RtabMap algorithm. Like most other algorithms, RtabMap's contribution lies not in the feature detection or matching (it uses external libraries such as OpenCV for this); it lies instead in its approach to providing a globally consistent solution using a graph-based SLAM method.

However, in the context of the ARSI concept of operations, producing a globally consistent solution is difficult because CPU and memory resources on the onboard computer are limited, and because we typically do not visit the same area twice in a single flight (to maximize coverage given our limited autonomy) which is key to loop-closing techniques in SLAM algorithms.

During flights of the ARSI MAV, we are really only interested in solving problem 1) to produce robust velocity estimates for the PID controllers. Position drift over time does not directly affect control, since the setpoint sent to the autopilot is recalculated at each iteration using local sensor data (depth and laser). In practice, we only use the visual odometry part of RtabMap when running onboard the MAV, leaving out the global consistency problem for post-processing (details about this are given in [6]).

Wall and ground detection

The Orbbec RGB-D camera integrated in phase II produces depth images at a frequency of 30Hz. These depth images can be backprojected using the camera intrinsics, to generate point clouds containing around 300,000 points. Since each pixel in the depth image has a corresponding pixel in the color image, we can apply to RGB value to each vortex in the point cloud to generate a raw 3D model (see Figure 2) of the vicinity of the ARSI MAV (the depth range is 8m).

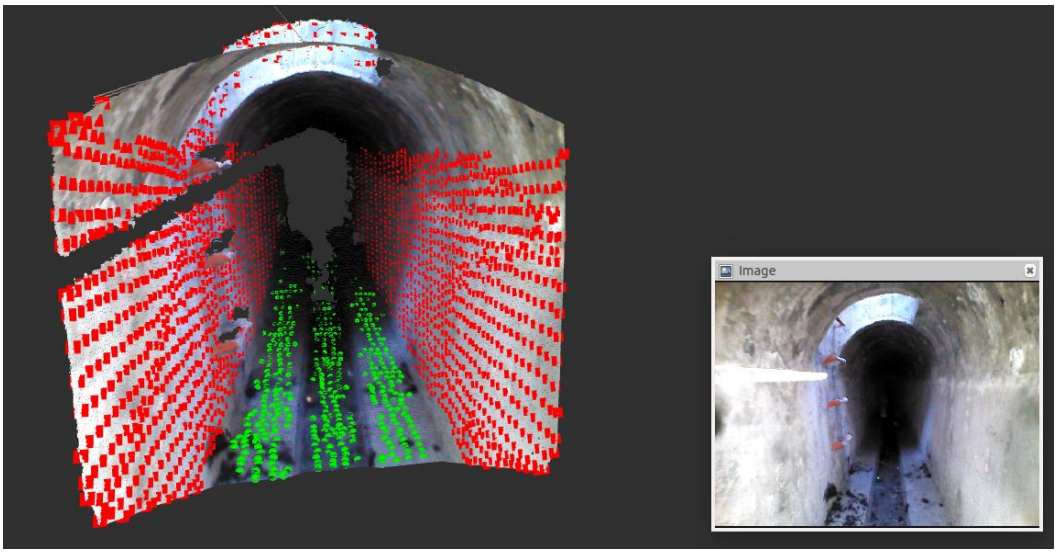

Figure 13: Walls (in red) and ground (in green) detection from RGB-D data

Clearly, the point clouds generated by the Orbbec camera carry detailed information about the surroundings. In this phase we used this information both in the costmap (see below) and in the altitude control.

First we developed an algorithm to segment the point cloud into walls and ground planes. For each point, the nearest neighbors are identified using a KD tree and the local surface normal is estimated. Normals are then clustered into planes using region growing and labeled based on their orientation, to identify ground and wall planes. Figure 13 illustrates this segmentation with

RGB-D data from Mercado del Born in Barcelona.

Pixels belonging to the walls can be used for trajectory generation alongside laser data (see costmap section). The detected ground plane on the other hand provides an estimation of the MAV altitude. This altitude estimation is more robust than that of the TeraRanger, since it is calculated from thousands of data points. However, these data points can only provide information about the area ahead of the drone, not the one it is currently flying over, since it is not visible in the camera. The altitude estimation derived from the ground detection is therefore only a prediction, based on the slope of the tunnel ahead. Although this prediction will be valid in the vast majority of cases, priority should always be given to the high-frequency TeraRanger measurements.

A new ROS node was developed to combine ground detections and TeraRanger measurements together in a weighted average filter, where a higher confidence is given to the ranger measurements. The weighted range is then passed to the Pixhawk firmware via Mavros with a very high confidence for odometry correction along the Z-axis. Our many tests have shown that the estimated altitude is consistent and stable, even when flying over uneven ground or sewage water.

Sensor fusion

The PX4 firmware onboard the Pixhawk autopilot module runs two highly configurable state estimator modules to produce attitude, velocity and position estimates (see Figure 11):

- Q attitude estimator: quaternion-based complementary filter for attitude. It can use information from accelerometers, magnetometers, gyroscopes and an external heading source.

- INAV position estimator: filter for 3D position and velocity states. It can use information from barometers, GPS, rangers for altitude estimation, and external position sources such as visual odometry or motion capture systems.

Both estimators are complimentary filters. They use the IMU as a core sensor, aided with the secondary sensors. The weight for each secondary source can be adjusted to reflect the level of confidence we have in their measurements.

For instance, we use these weights to disable the contribution of the GPS since there is no coverage in the sewer network. Also we noticed during our early tests that the magnetometer data often did not match the real trajectory of the ARSI platform in the sewer tunnels (we suspect due to the presence of metallic structures in tunnels). We therefore assigned a very low weight to the magnetometer in the state estimation configuration in order to minimize its effect on the overall solution.

Our experience from phase II however highlighted certain issues that we hope to investigate in the next phase:

- The quality of the illumination by the onboard LEDs has a direct impact on feature detection and the visual odometry. We should look for more powerful and power-efficient LEDs for the ARSI MAV.

- The limited field of view of the Orbbec Astra (60°x49.5°) results in some parts of the sewer not being visible in the image, depending on the altitude of the MAV. We should investigate the use of a fisheye camera for visual odometry, since it would provide more visual features, as well as better imagery for inspection purposes.

- Dust lifted by the MAV motors is sometimes visible in the imagery, especially when flying at low altitude, affecting the quality of the odometry. We should investigate other lighting sources and LED configurations on the MAV, to avoid illuminating dust particles in that are directly front of the camera.

GIS-referenced localization

While sewer tunnels are largely featureless, they do still exhibit a few landmarks, manholes and intersections in particular. Both types of landmarks are identifiable in the 3D point clouds generated by the 3D camera, and both have their locations recorded in the GIS maps of the sewers. If these landmarks can be detected and matched against GIS data, they can provide an unbiased position source for localization (similar to GPS).

During phase III, this solution will be investigated and implemented with our frontal camera to detect these landmarks using machine learning techniques (see in Figure 2, for instance, where manhole entry chutes are clearly visible in RGB-D point clouds). These detections will be matched against the GIS using a probabilistic model to estimate the robot location uncertainty, to derive the real robot location relative to the GIS landmark and correct for the drift caused by the odometry.

3.2 Onboard navigation

As already presented in ARSI D26.5 — Prototype for sewer inspection [4] we used a classic approach for our navigation system. We build a real-time 2D cost map of the local area from sensor data, and use it to derive an optimal trajectory to achieve our global goal while meeting our local requirements. In our case, the global goal is to fly from a given manhole to another; the local requirements are to fly safely by staying away from all obstacles and in a manner that generates high-quality inspection data.

Costmap

The ARSI costmap is derived from the standard ROS 2D costmap node, to which we added the following functionality:

- Additional controls to filter point cloud and laser data by range, altitude, or field of view
- In the ROS implementation, obstacles remain marked as long as they appear in the region of intersect, while we want them to be valid for a time-frame only. This is required to handle unreliable measurements such as dust particles visible in laser scans.

- Integration of the dust filtering described below.

The costmap is defined by a series of obstacle and inflation layers, as shown in the configuration file used on the ARSI MAV (Figure 14). We used a 4m by 4m map with a 1cm resolution. This provides enough visibility of incoming obstacles or intersections, and a good enough resolution for precise control in narrow tunnels, whilst managing the onboard CPU and memory resources.

The costmap acts as a rolling window following the MAV: obstacle data messages are pushed to the back of an observation queue, and removed at the front after their persistence period (100ms) has expired. The costmap is updated at 20Hz, and consists of 3 layers:

- An obstacle layer populated with scan data from the Hokuyo laser
  - Data is read directly from the Hokuyo driver LaserScan topic
  - We used a 30cm minimum range, and field of view is reduced to ±120 degrees to avoid seeing the MAV arms or propellers in the data
  - We use a 20cm radius and a minimum of 40 inliers for the radius-based outlier removal (see below)
- An obstacle layer populated with point cloud data of ground detections
  - Data is read from the PointCloud topic published by the ground detector
  - The Orbbec Astra has a 30cm minimum range for depth data
  - Incoming data is already downsampled by the ground detector using a voxel grid with a 5cm resolution
- An inflation layer
  - The inflation layer inflates every obstacle pixel in the costmap with a 30cm radius
  - It closes the gap between obstacles and defines an area that the MAV should never enter (and therefore that the local planner should not explore)

```
global_frame: odom
robot_base_frame: base_link
observation_persistence: 0.1

update_frequency: 5.0
publish_frequency: 20.0
static_map: false
rolling_window: true
width: 4.0
height: 4.0
resolution: 0.01

plugins:
    - { name: hokuyo, type: "costmap_2d::ObstacleLayer" }
    - { name: walls, type: "costmap_2d::ObstacleLayer" }
    - { name: inflation, type: "costmap_2d::InflationLayer" }
```

```
hokuyo:
    enabled: true
    combination_method: 1
    observation_sources: hokuyo_scan

    hokuyo_scan:
        data_type: LaserScan
        topic: /hokuyo/scan
        sensor_frame: laser
        min_range: 0.3
        min_angle: -120.0
        max_angle: 120.0
        filter_radius: 0.2
        filter_k_neighbors: 40

walls:
    enabled: true
    combination_method: 1
    observation_sources: walls_cloud

    walls_cloud:
        data_type: PointCloud2
        topic: /wall_detector/walls
        sensor_frame: base_link
        min_range: 0.3

inflation:
    inflation_radius: 0.3
    cost_scaling_factor: 1.0
```

Figure 14: Costmap configuration for ARSI MAV

Dust filtering

As described in [5], early flights in the sewers at Mercado del Born showed that the MAV motors unexpectedly lifted the dust from ground. Since the costmap is used to calculate a safe local path towards the global goal, any dust visible in the costmap is considered to be an obstacle that the MAV should avoid. If left unmanaged, the dust comes in and out of view, trajectories generated by the local planner become wildly unstable and safe flight becomes impossible.

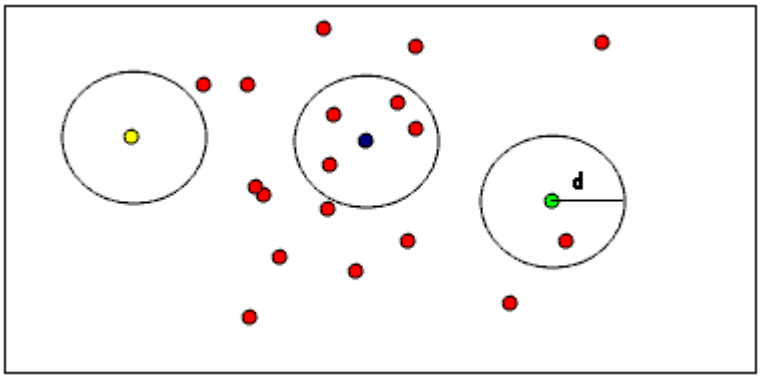

Figure 15: Radius-based outlier removal (image courtesy of PCL)

In this phase, we used a radius-based outlier removal technique. As shown in Figure 15, the neighbors within a certain radius for each laser data point are identified using a KD tree, and all points for which the number of neighbors is tool low are removed. In practice this filter works well for laser data of the sewers because points of the walls will typically have large support, while dust in the air will not.

In our tests we found that this filter performed well in all but extreme conditions. In phase III we hope to investigate methods combining RGB-D and laser for dust removal, as we observed that dust is much less visible in the depth images. Any dust detected in the laser would be matched against the corresponding region in the depth image for confirmation, and discarded if no matching obstacle is found.

Local planner

The local planner used for the ARSI MAV is based on the Dynamic Window Approach [9]. The general idea is to search the space around the MAV for possible trajectories which will advance the global goal whilst reacting to the environment to ensure safe flight.

Based on the costmap generated using the latest sensor data, the local planner iteratively evaluates all polyline trajectories within a search space defined by a rotation range and a prediction range. The rotation range is useful mainly to reduce the search space, since we know that we want the MAV to move forwards. The prediction range represents how far the planner should to look ahead. This range must be short enough so that it can react to changes in its immediate surroundings (e.g. an intersection, a manhole ladder coming in and out of view) whilst looking far enough ahead so that it starts following the correct trajectory before getting to the actual obstacle. This is particularly important in our case, as sudden yaw movements of the MAV cause images from the RGB-D camera to be blurry. Blurry images reduce the quality of the visual odometry and velocity estimates, which in turn affects control. In our tests we found that a prediction range of 1m was well adapted to navigate turns, intersections and change of sections (connections between tunnels of different dimensions) in the sewers.

Trajectories are scored based on 1) how they reduce the distance to the goal 2) how safe they are, that is how large the overall distance to all obstacles is (in particular the sewer walls).

Distance to obstacles is calculated using the distance transform of the costmap, so that each pixel value represents the smallest distance to any obstacle. Other criteria could be added in the future, for example following a predefined path or promoting certain flight dynamics. The trajectory score is a weighted sum of the different criteria, and the best trajectory is retained at each planner iteration (the planner runs at 20Hz).

Flight control

Flight control software for the ARSI MAV is executed by the PX4 flight stack onboard the Pixhawk autopilot unit. Based on the estimated MAV pose, PX4 uses cascaded PID controllers to control velocity, attitude and motor thrusts.

As detailed in [5] tuning the flight controllers for operation in the sewer tunnels proved very challenging in phase II. While numerous flight and control tests had been carried out in our laboratory environment, we found that the sewer conditions changed the flight dynamics so much that extensive retuning was required.

The process of tuning control in the sewers was made difficult by the fact that control instability often resulted in uncontrolled landings of the MAV. We did eventually achieve what we consider to be very good control considering the limited manoeuver space available in the sewers, the uneven ground, as well as turbulences and dust movement caused by the air flow generated by the MAV in such a confined space. Details about the tuning and testing process are given in [5].

3.3 Mission execution

The mission execution software developed in phase II defines each inspection flight as a series of pre-defined goals. The mission file in YAML format for a typical inspection flight is shown in Figure 16 and contains the following goals:

- Takeoff from current location
- Fly to chosen manhole (path planner)
- Land at the current location
- Disarm the motors

The mission is split into mission goals and an end sequence. Operators can abort the main mission at any moment from the Mission Control interface using the abort/land button [6]. A ROS service call is issued, and Mission Execution onboard the MAV is interrupted, going directly to the end sequence which typically contains lands the MAV and disarms the motors.

```
tolerances:
  xy: 0.2 # meters
  z: 0.1 # meters
  yaw: 5.0 # degrees

goals:
- { goal_type: takeoff, altitude: 0.7, tolerances: { z: 0.2 } }
```

```
- { goal_type: path_planner, x: 200.0, y: 50.0, cruise_altitude: 0.6,
low_altitude: 0.2, cruise_velocity: 0.5, slow_velocity: 0.2, max_yaw_error:
40.0, max_y_error: 0.4 }

end_sequence:

- { goal_type: landing, min_time_landed: 0.4, velocity_threshold: 0.1,
max_landed_altitude: 0.1, velocity_control: { min_z: -0.1, max_z: 0.1 } }

- { goal_type: disarm }
```

Figure 16: ARSI mission file

The takeoff, path planner and land goals all use the costmap and local planner to calculate their trajectories, so that the MAV dynamically stays away from the sewer walls or any other obstacle. At each iteration, the Mission Execution module requests a trajectory from the local planner, from which a control setpoint (target location) is derived. This dynamic setpoint (since it is constantly recalculated based on sensor data) is then issued to the Pixhawk control stack and executed. The cycle repeats until the MAV has reached its target destination and lands.

Goal	***Parameters***
Takeoff	• Takeoff altitude (relative to the MAV) • Optionally: adjusted goal tolerance along Z-axis
Path planner	• (x,y) goal location in MAV frame • Cruise altitude (m) • Low altitude (to avoid obstacles) • Cruise velocity (m/s) • Slow velocity (for narrow sections) • Max allowed errors in Y (m) and yaw (deg)
Landing	• Time on ground before disarming (s) • Velocity threshold to detect if MAV is moving • Maximum altitude when landed • Optionally: adjusted Z-velocity control maximums for smooth landing

Table 2: ARSI mission goals parameters

Mission files for each flight in an inspection campaign are currently prepared ahead of time and executed in turn. All goals are highly configurable, in particular path following, as shown in Table 2. In phase III, we will develop a user interface to display maps and GIS information of the area to inspect using the ARSI MAV. Using the interface, operators will be able to generate mission plans simply by selecting the entry and exit manholes for each flight. All mission plans will then be uploaded to the MAV, and the system will be ready to execute inspections directly from the Mission Control interface.

ARSI Watchdog

The Challenge Brief requires that operators on the surface have constant Wi-Fi link to the MAV during inspections. While our system meets this requirements in most cases, the possibility of losing signal during a flight exists and must be managed for safety reasons, so that the ARSI MAV is not allowed to continue flying autonomously without operator control.

The ARSI system includes a “watchdog” module which runs onboard the MAV and monitors communications to the Mission Control interface via a heartbeat message sent over UDP using ROS. If the watchdog detects that communications have been lost, it triggers an emergency sequence which commands the onboard Mission Execution module to land the MAV at its current location. If for any reason the landing sequence also fails, the watchdog will then issue a disarm command: this will stop the motors and the MAV will fall to the ground. While this will likely cause some damage on the MAV, it will ensure that it does not continue to fly unsupervised in the sewers.

The ARSI watchdog module runs on the onboard computer and communicates with the PX4/Pixhawk flight stack via a ROS interface (mavros). This link also implements a watchdog, so that the PX4 firmware will realize if communications are lost (within a 0.5s window), for example if the onboard computer loses power. PX4 will then enter a failsafe mode which triggers its own pre-programmed emergency landing and disarm sequence.

3.4 Communications

In deliverable D26.4 [3], we presented a detailed example of the ARSI MAV concept of operations for the area of Mercado del Borne in Barcelona. In this document, we defined a series of flights designed to maximize inspection coverage while keeping logistics simple for the inspection teams.

Our inspection plan is also designed to maximize Line-of-Sight (LoS) communications between the MAV and the Wi-Fi router connected to the Mission Control interface on the surface, since our tests have shown that Wi-Fi travels very reliably in straight lines using the waveguide properties of the sewer tunnels. In Mercado del Borne however, some travel around corners is unavoidable in order to cover the entire area. While the loss of LoS does not affect the autonomous flight itself, and while our tests have shown that Wi-Fi signal does travel beyond line of sight to a certain extent, it does make it more likely that communications to the base station will be partially lost.

Bandwidth management

In this section we describe the bandwidth management strategy developed to work around this issue. They key idea is to assign priorities to the different communication channels (typically ROS topics) between the MAV and the Mission Control interface.

In a standard ROS configuration, the ROS core runs on the onboard computer robot, while other computers (e.g. remote control stations) establish a connection to the onboard ROS core (by exporting the ROS master URI), to access messages on the different topics. During our tests in the sewers we observed that if left unmanaged, the communication link between the MAV and Mission Control can become saturated. The consequence is not only that feedback becomes unavailable to the operator, but also that commands such as landing, disarm, etc. can no longer

be issued to the MAV. This effectively means that the operator loses control over the MAV.

The communications module developed in phase II bypasses the standard ROS interface between the MAV and the Mission Control interface over the Wi-Fi link replacing it with a point-to-point connection over which we have total control. Instead, two ARSI communications nodes (a server and a client) communicate directly over a direct UDP link between the MAV and Mission Control.

On the MAV, the server node implements a system of topic priorities assigned in advance via a configuration file, so that even if some information is partially or even completely lost (e.g. the live video feed) we are still able to exchange low-bandwidth but more critical information with the MAV, in particular emergency services such as landing and disarm.

The server also allows us to greatly reduce the amount of data exchanged between the MAV and Mission Control. Images and laser scans, for example, are compressed and down-sampled to a manageable frequency, ensuring that they still remain useful both for flight monitoring and live inspection.

On the remote station, the client communications node converts the data streams received on the UDP link back to standard ROS topics, and provides seamless communication with any other ROS nodes running locally such as our Mission Control interface.