Autonomous Robotic Inspection and Maintenance on Ship Hulls and Storage Tanks

Deliverable report – D5.2

Context	
Deliverable title	Autonomous Trajectory Tracking and Obstacle Avoidance
Lead beneficiary	UPORTO
Author(s)	CNRS, UPORTO, UIB, UNI-KLU, NTNU
Work Package	WP05
Deliverable due date	31 September 2023 (M39)
Document status	
Version No.	1
Type	REPORT
Dissemination level	**Public**
Last modified	3 October 2023
Status	RELEASED
Date approved	6 October 2023
Approved by Coordinator	Prof. Cédric Pradalier (CNRS) Signature: [illegible]
Declaration	Any work or result described therein is genuinely a result of the BUGWRIGHT2 project. Any other source will be properly referenced where and when relevant.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 871260

TABLE OF CONTENTS

LIST OF FIGURES

REFERENCED DOCUMENTS

- Deliverable D2.3 – *MAV adaptation to BUGWRIGHT2 requirements*
- Deliverable D4.1 – *Localisation*
- Deliverable D4.2 – *Local Mapping and Obstacle Perception*
- Deliverable D5.1 – *Unified Control Interfaces*
- Stumm *et al.* – Stumm, E., Breitenmoser, A., Pomerleau, F., Pradalier, C., & Siegwart, R. (2012). Tensor-voting-based navigation for robotic inspection of 3D surfaces using lidar point clouds. *The International Journal of Robotics Research*, *31*(12), 1465-1488

The deliverables are stored on the file sharing site hosted by CNRS.

HISTORY OF CHANGES

Date	Written by	Description of change	Approver	Version No.
09/02/2023	UPORTO	Starting document		v0.0
29/03/2023	UIB, NTNU, CNRS	General text additions		v0.2
03/10/2023	CNRS	Crawler section additions		V1.0
06/10/2023	CNRS	Proofreading/ validation	CNRS	

ABBREVIATIONS

BS	Base Station
BW2	BUGWRIGHT2
MAV	Micro-Aerial Vehicle
SA	Supervised Autonomy
VTP	Virtual-Target Point
AUV	Autonomous Underwater Vehicle

Executive Summary

Delivery D5.2 is focus on the report on the online local navigation capabilities of BUGWRIGHT2's robots on or around vessel hulls and storage tanks. This task will tackle the problem of safe and precise autonomous local navigation of the robotic platforms on and around the inspected structure, using the data from task 4.2 and linking the unified control interfaces (task 5.1) to the higher-level mission planning aspects (task 5.3).

I. Introduction

The project BUGWRIGHT2 aims at the development of several robotic platforms oriented to making (ship) inspections easier and faster for the inspection crew. To this end, the robots need a suitable control architecture to solve the respective operating cases with the required level of autonomy. Among others, the control architecture may need representations of the environment at various levels to achieve the intended goals. These representations can be regarded as containers where the corresponding platform stores processed data collected by means of the onboard sensors as well as updates in accordance with new findings.

This work package 5 will develop the tools required for the autonomous navigation of the robotic platforms on or around the structure to be inspected, including control, safe trajectory tracking, global motion planning and coverage planning.

This task will tackle the problem of safe and precise autonomous local navigation of the robotic platforms on and around the inspected structure, using the data from task 4.2 and linking the unified control interfaces (task 5.1) to the higher-level mission planning aspects (task 5.3). On the MAV side, extending UIB's existing framework, UNI-KLU will handle wind gust and vessel motion, for safe and precise motion as well as autonomous take-off and landing. On RBP's crawlers and BEYE's AUV, the main challenge comes from the tether transporting power and data to the robot. For the crawlers, CNRS will include entanglement risk in the local navigation algorithm. On the AUV side, UPORTO and NTNU will integrate their time-tested local navigation framework for precise trajectory following and obstacle avoidance.

The rest of this document is organized as follows: Section II details the autonomous trajectory tracking, where Section II.1 entails the trajectory tracking for the MAVs, Section II.2 for the Pioneer X3s, and Section II.3 for the crawlers; Section III relates to obstacle avoidance, where Section III.1 focuses on the MAVs, Section III.2 on the Pioneers X3, and Section III.3 on the crawlers; finally, Section 0 summarizes and concludes deliverable D5.2.

II. Autonomous Trajectory Tracking

II.1. Trajectory Tracking for MAV

This section describes the trajectory following and the obstacle avoidance functionalities running onboard the inspection drone developed by the UIB.

It must be noted that the inspection drone, based on the commercial platform DJI Matrice M100, has been described at the hardware component level in deliverable D2.3 (MAV adaptation to BUGWRIGHT2's requirements), while a first view of the control architecture can be found in deliverable D5.1 (Unified Control Interfaces) and state estimation and localization issues have been treated in deliverable D4.1 (Localisation). Finally, the local mapping functionality is described in deliverable D4.2 (Local Mapping and Obstacle Perception).

II.1.1. Brief overview of the control architecture

The control architecture of the inspection drone has been designed around the *Supervised Autonomy* (SA) paradigm. This model defines a framework for human-robot interaction that aims at the alleviation of stress on human users through an appropriate level of instructions and feedback. In other words, the operator is not burdened with the complete control of the system, and hence can concentrate on the inspection task at hand. The SA framework comprises five concepts: *self-preservation*, which states that the robot itself is in charge of performing the necessary tasks to ensure its integrity; *instructive feedback*, used to provide the user with the same perception capabilities as the robot; *qualitative instructions*, to command the robot in an easy-to-understand manner; *qualitative explanations*, to show to the user what is happening using a language similar to the qualitative instructions; and a *user interface* to visualize the instructive feedback and allow the user to issue qualitative instructions. **Figure 1** illustrates graphically the full visual inspection system, designed in accordance to the SA paradigm as explained.

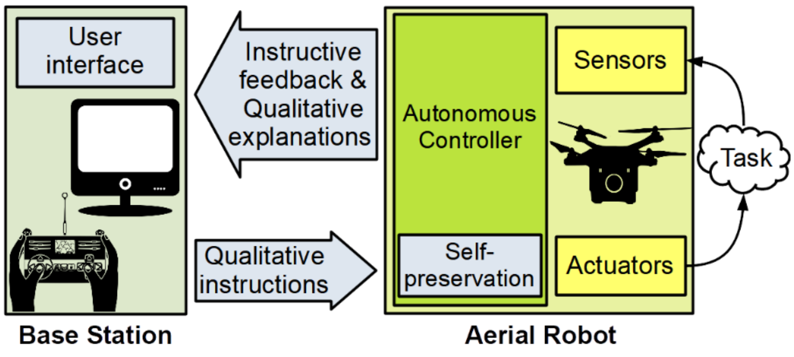

Figure 1: Overview of the UIB visual inspection system as designed around the SA paradigm.

To implement the SA framework, the system architecture comprises two separate agents: the vehicle and the Base Station (BS). On the one hand, the aerial platform, which is fitted with suitable sensors and actuators, is in charge of working out the control-related issues required to successfully carry out the specific task. Moreover, the same autonomous controller is also in charge of the self-preservation of the platform. On the other hand, the BS allows the operator/surveyor to feed the robot with qualitative instructions by means of human-robot interaction devices. At the same time, the BS provides the operator/surveyor with information about the mission's state and the robot status, using instructive feedback and qualitative explanations. The communication between both agents is performed via wireless connections. The autonomous controller comprises a set of behaviours that are in charge of accomplishing the intended task while ensuring the platform self-preservation. By way of illustration, a behaviour can be in charge of moving the platform as indicated by the user, another can prevent collisions with the surrounding obstacles, another can keep a constant distance to the inspected surface, etc.

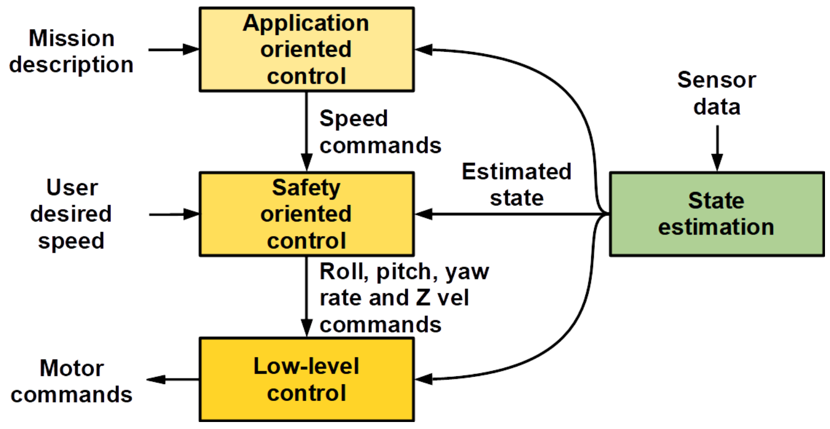

Figure 2: Control software of the UIB inspection drone shown as a layered architecture.

As shown in **Figure 2**, the control architecture of the robot assimilates to a hierarchical layered structure, where each layer implements a different level of control. Moreover, mid- and high-level control layers run different robot behaviours that contribute to the generation of the final motion control command. Additional details follow next:

- The low-level control layer comprises attitude and thrust control, as well as the behaviours that check the viability of the flight. We make use of the DJI FMU services, through the DJI SDK, to make available this functionality.
- The mid-level control layer accommodates safety-oriented control by including the *Safety manager* module, which comprises several robot behaviours that combine the user desired speed command with the available sensor data to obtain final and safe speed and height set-points that are sent to, respectively, the horizontal speed and height controllers.
- At the highest level of the hierarchy, the application-oriented control layer allows the execution of predefined missions by means of the *Mission manager* module, which is in charge of executing higher autonomy behaviours that implement missions defined as a set of waypoints to attain. The corresponding motion commands are generated in sequence and issued to the corresponding position controller, while monitoring waypoint achievement and overall correct mission execution.

Additionally, as shown in **Figure 2**, a state estimation module is transverse to all layers. This module is in charge of processing and fusing all the sensor data available on-board, to estimate with enough accuracy the platform state. The state estimate is employed in all control layers.

The inspection-oriented aerial platform developed by the UIB can perform inspection-related missions in fully autonomous mode, although the operator is allowed to take control whenever deemed necessary (e.g. to avoid an imminent crash or to solve a task that is not covered by the control software capabilities). These missions are handled by the *mission manager* and comprise:

1. inspection of a wall, so-called *sweeping*, a zig-zag like movement that goes from side to side and from top to bottom of a virtual rectangle in front of the surface to survey;
2. inspection of a vertical structure (such as a ladder or a vertical pipe, inspecting the structure from the left as the vehicle goes up, and from the right as it comes down);
3. inspection of an isolated structure, by surrounding it following a circular path, so-called *circular inspection*; and

4. the ability to go to a point, useful to bring the platform to a particular location previously memorized;
5. go home, which takes the platform to the point defined as home, e.g. the point where the vehicle took off.

Transversally to these autonomous missions, the operator can activate trajectory following to make the robot adhere to the respective intended paths; this functionality runs, hence, at the highest level of the control architecture hierarchy, together with the autonomous missions (see **Figure 2**). Collision prevention and obstacle avoidance, however, are an intrinsic part of the control architecture; indeed, the control architecture has been defined around platform self-preservation (as intended within the SA paradigm), and, thus, obstacle detection and avoidance run as part of the mid-level control layer (see **Figure 2** again).

II.1.2. Trajectory following

All autonomous missions consist of a list of waypoints that the vehicle must sequentially reach, and thus perform the appropriate trajectory to accomplish the mission. In more detail, the mission manager computes the next waypoint to be attained and provides it to the *position controller*. This module implements a set of standard PID controllers that provide the suitable velocity commands to make the platform reach the intended waypoint. Once the vehicle has reached the current waypoint (within a given tolerance WP_{tol} of e.g. 20 cm), the mission manager provides the next waypoint to the position controller. This process is repeated until the last waypoint is reached, and the mission is finished.

Although the described framework ensures that the vehicle attains all the waypoints that make up an autonomous mission, the trajectory followed when travelling from a waypoint to the next is not controlled. Indeed, the position controller is implemented as three independent, decoupled PID controllers, one for each axis of motion, what can derive into non-synchronized control actions, i.e. the control goal is achieved in one axis before than in the others. Furthermore, the aerial platform is exposed to external disturbances, such as wind, which can also affect the actual trajectory followed by the platform. As an example, **Figure 3** (left) shows the trajectory made by the aerial platform when performing a go-to-point mission in the presence of 5 m/s wind from the left. The figure shows in blue the desired trajectory (i.e. the straight line joining the current position of the MAV with the waypoint), while the actual trajectory followed by the MAV is shown in purple. As can be observed, the height controller reacts faster than the controllers in charge of moving the platform in the two other axes, what separates the platform from the desired path. Similarly, **Figure 3** (right) shows a sweeping mission performed under the same wind conditions, where the desired trajectory is indicated in green. As can be seen, in this experiment it is more noticeable how the wind pushes the aerial platform to the right, separating it from the desired trajectory.

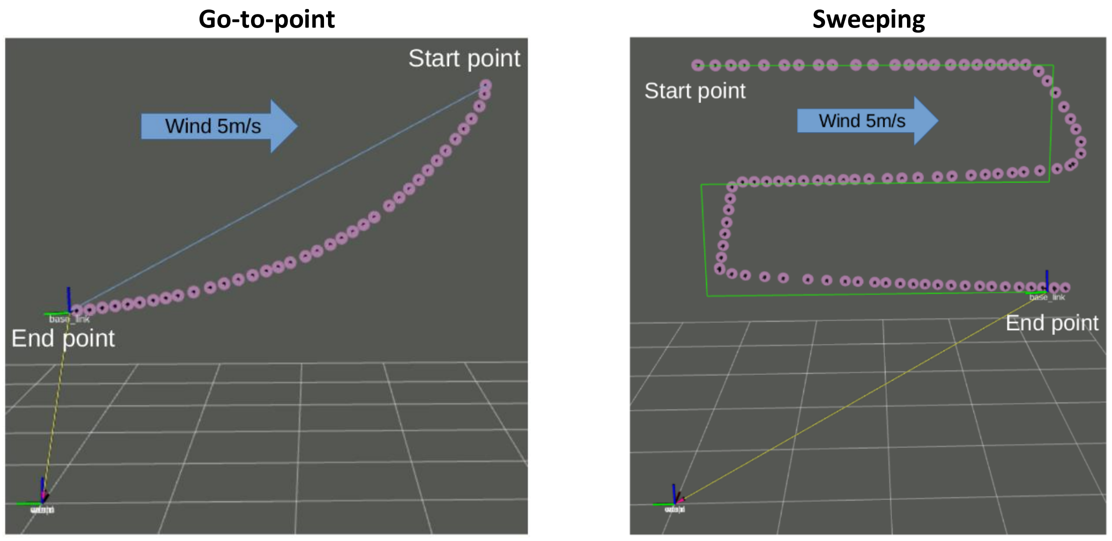

Figure 3: Trajectories not controlled when the vehicle performs two different missions. The blue/green lines indicate the desired trajectories, while the purple markers show the actual trajectories followed by the MAV.

In order to ensure that the vehicle advances over the desired trajectory (the one that joins the waypoints, with a certain margin of tolerance), a trajectory following functionality has been developed on the premise that it should be completely orthogonal to the missions. In this way, given two waypoints (the current and the previous one) the trajectory follower (if activated) periodically provides a *Virtual-Target-Point* (VTP) located on the line that joins them as an intermediate waypoint. In other words, during an autonomous mission, the VTP is continuously recomputed and fed into the position controller instead of the current waypoint.

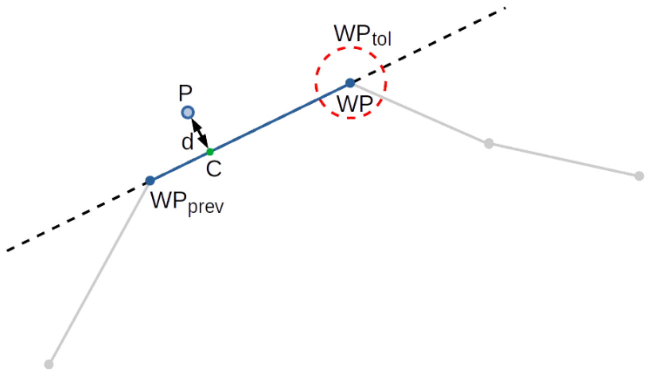

Figure 4: Waypoints involved in the computation of VTPs. ***P*** indicates the current location of the MAV, ***C*** indicates the point of the path which is closest to ***P*** and ***d*** is the distance between ***P*** and ***C***.

We explain next how the VTP is defined. To this end, let us consider the current waypoint WP and the previous waypoint WP_{prev} , and let P be the current location of the MAV. Then, we define (see **Figure 4**):

- C as the point closest to P and belonging to the line connecting WP and WP_{prev} , and
- d as the distance between P and C.

Being λ the maximum separation permitted with regard to the intended path, we consider up to five cases depending on the location of C:

1. **C is between WP and WP_{prev} and $d < \lambda$**

VTP is set at a distance L from C towards WP, where $L = \delta(1 - d/\lambda)$ and δ is the maximum distance to the VTP (see **Figure 5**). In this way, the platform moves towards the current waypoint WP but tries to stay on the path. In case L is larger than the distance between C and WP (the platform is closer to WP), the VTP is set to WP.

2. **C is between WP and WP_{prev} and $d \geq \lambda$**

   VTP is set to point C, as shown in **Figure 6**, so that the platform focuses on getting closer to the path.

3. **C is before WP_{prev} and outside WP_{tol}**

   VTP is set to point WP_{prev}, as shown in **Figure 7**. This is because the platform has moved backwards within the trajectory, so it must revisit the previous waypoint.

4. **C is before WP_{prev} and inside WP_{tol}**

   This case is illustrated in **Figure 8**. The VTP is computed as in case 1 or 2 depending on d.

5. **C is beyond WP**

   VTP is set to the current waypoint WP (see **Figure 9**). The platform has passed the current waypoint, so it must backtrack to reach it.

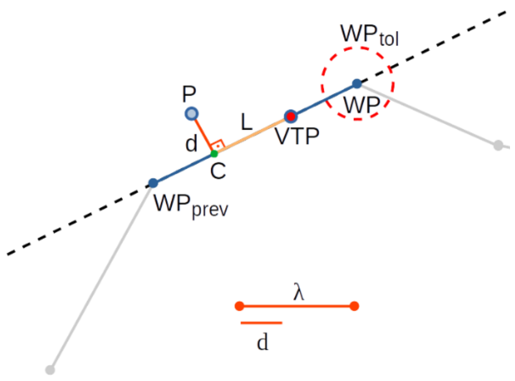

Figure 5: (Case 1) Computation of a VTP when $\boldsymbol{C}$ is between $\boldsymbol{WP}$ and $\boldsymbol{WP_{prev}}$ and $\boldsymbol{d} < \boldsymbol{\lambda}$.

$\boldsymbol{L = \delta(1 - d/\lambda)}$.

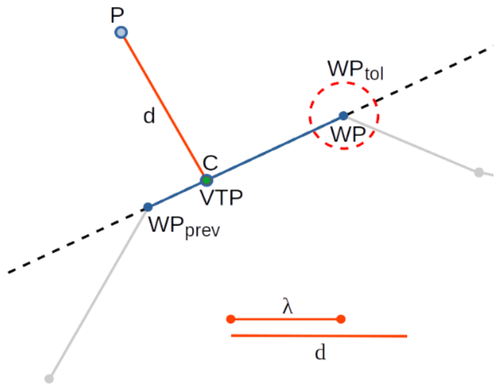

Figure 6: (Case 2) Computation of a VTP when $\boldsymbol{C}$ is between $\boldsymbol{WP}$ and $\boldsymbol{WP_{prev}}$ and $\boldsymbol{d} \geq \boldsymbol{\lambda}$.

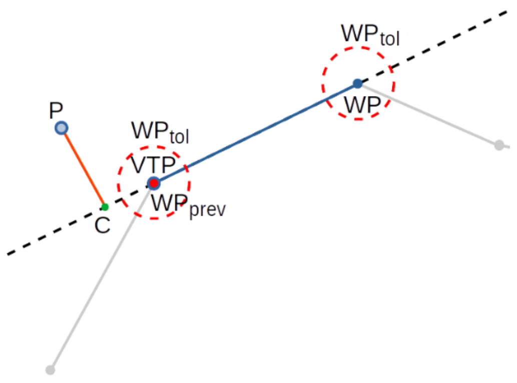

Figure 7: (Case 3) Computation of a VTP when $\boldsymbol{C}$ is before $\boldsymbol{WP_{prev}}$ and outside $\boldsymbol{WP_{tol}}$.

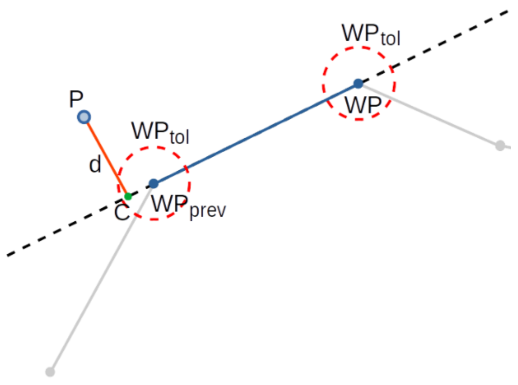

Figure 8: (Case 4) Computation of a VTP when $\boldsymbol{C}$ is before $\boldsymbol{WP_{prev}}$ and inside $\boldsymbol{WP_{tol}}$.

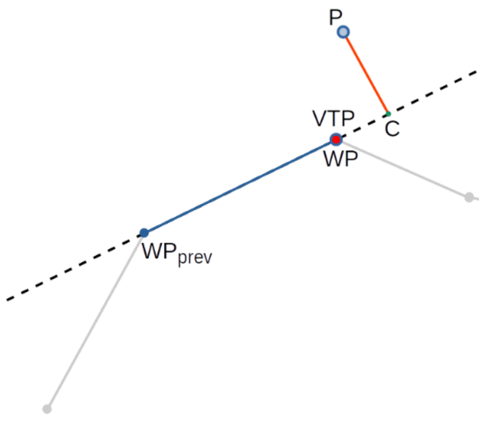

Figure 9: (Case 5) Computation of a VTP when $\boldsymbol{C}$ is beyond $\boldsymbol{WP}$.

Notice that during normal operation, the aerial platform is expected to be in **case 1** most of the time. To better understand the expression that applies in this case we can see how L, i.e. the distance from C to the VTP, evolves depending on the distance to the path d. **Figure 10** shows the relationship between these two distances using δ = 2.0 m and λ = 0.5 m. As can be observed, the closer the platform is to the path, the farther the VTP is placed from C towards WP.

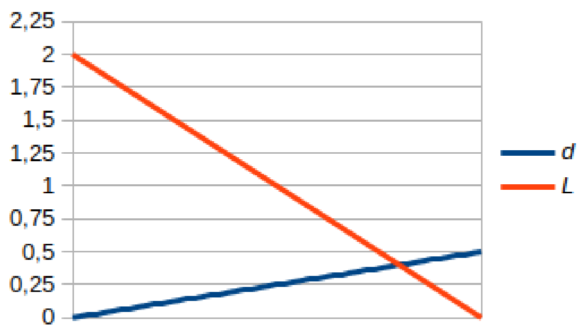

Figure 10: Relation between the distance $\boldsymbol{d}$ from the platform to the path and the distance $\boldsymbol{L}$ at which the VTP is set from $\boldsymbol{C}$, using $\boldsymbol{\delta}$ = 2.0 m and $\boldsymbol{\lambda}$ = 0.5 m.

Parameters λ and δ allow us to configure the trajectory follower to make the platform move as we wish, decoupling two different characteristics of the movement: while λ allows us to configure how close we want the platform to remain regarding the path, δ indicates the distance where we want to situate the VTP. Having in mind the position controller that resides in the control architecture of the vehicle, this distance has a key effect on the velocity at which the platform moves towards the waypoint. That is to say, a small δ will make the platform move slowly, while a larger value will make the platform move faster.

Figure 11, **Figure 12** and **Figure 13** show the effect of activating the trajectory follower while performing three different inspection-related operations in simulation mode[1], namely go-to-point (**Figure 11**), sweeping (**Figure 13**) and vertical inspection (**Figure 13**), in the presence of 5 m/s wind from the left. As can be observed in the images on the left, where the trajectory follower is deactivated, the platform is able to reach the waypoints despite the wind, but the paths followed are far from being as intended. A different behaviour is observed in the images on the right, where the trajectory follower is activated using δ = 2.0 m and λ = 0.2 m. In the three cases, the trajectory is closer to the desired one. Note what happens in the upper right corner of the last two experiments (sweeping [**Figure 12**] and vertical inspection [**Figure 13**]), where the platform inertia and the wind gusts push the platform out of the trajectory. In both situations, the trajectory follower operates in the aforementioned **case 2**, and focuses on approaching the platform to the intended trajectory before moving on to the next waypoint.

To finish, we illustrate the performance of the real drone with and without the trajectory follower activated: **Figure 14** and **Figure 15** illustrate the go-to-point operation, while **Figure 16** and **Figure 17** correspond to the sweeping operation, and **Figure 18** shows a vertical inspection mission. For a start, for the go-to-point case, the drone was first taken to waypoints A and B, separated around 11 m in X and around 3 m in Y (**Figure 15**), which were recorded once attained; later, the drone was commanded to go from point A to B twice, first without trajectory following and next with trajectory following. **Figure 14** shows a 3D view of the paths followed while **Figure 15** shows the projections onto planes ZY (front view) and XY (top view).

On the other side, **Figure 16** and **Figure 17** plot 3D and projected views of the path followed by the drone during a sweeping operation defined over a 9 x 5 m rectangle. As part of the normal operation to launch a sweeping mission, the drone was first taken to the upper-left corner of the rectangle, time during which the trajectory follower is logically deactivated, as well as between the end of the mission (lower-right corner, at about y = -4 and z = 1 in **Figure 17**[left]) and landing.

In third place, **Figure 18** shows 3D and projected views of the path followed by the drone during a 9 x 3 m vertical inspection operation.

As can be observed from the different plots, both in simulation and over the real platform, the drone paths adhere better to the intended paths when the trajectory following functionality is active, as expected.

[1] Using the Hardware-in-the-loop (HIL) simulation framework of the UIB

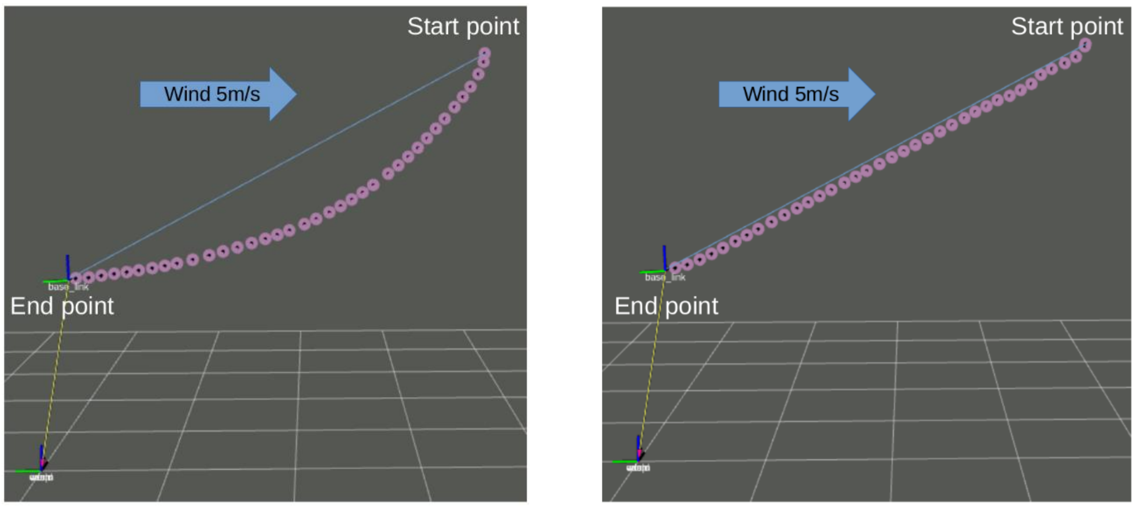

Figure 11: Illustration of the execution of a go-to-point operation in the presence of wind with the trajectory follower deactivated (left) and activated (right).

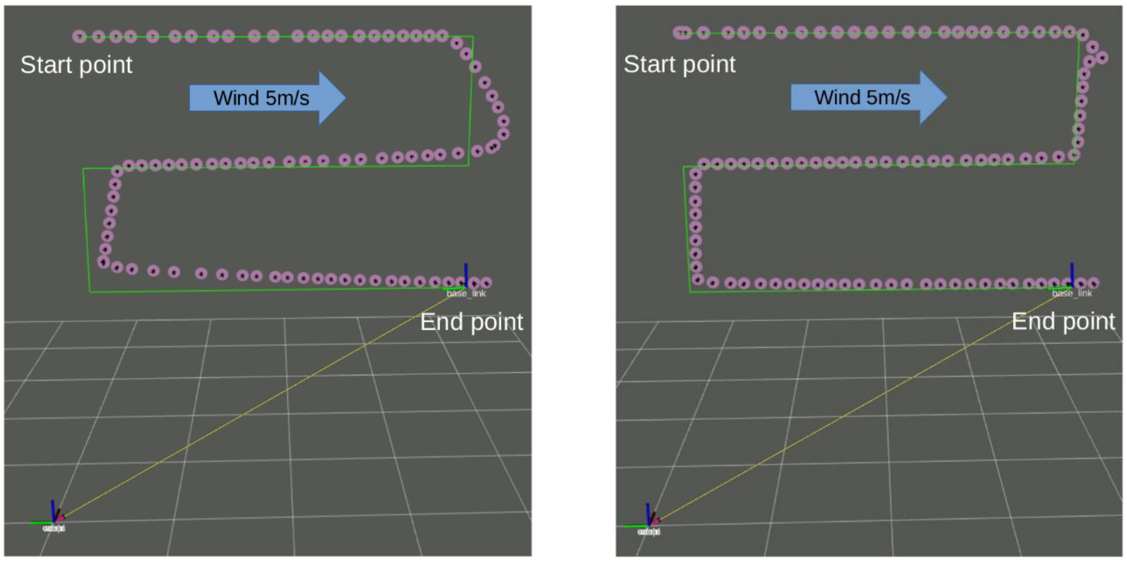

Figure 12: A sweeping operation with the trajectory follower deactivated (left) and activated (right).

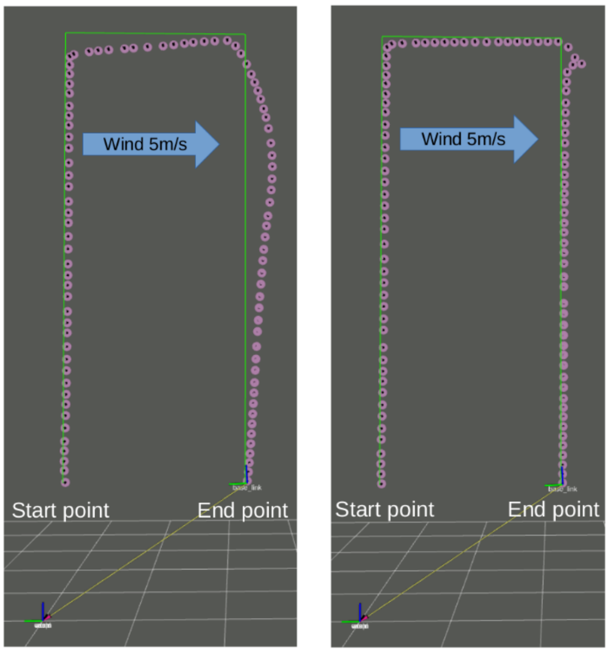

Figure 13: A vertical inspection operation with the trajectory follower deactivated (left) and activated (right).

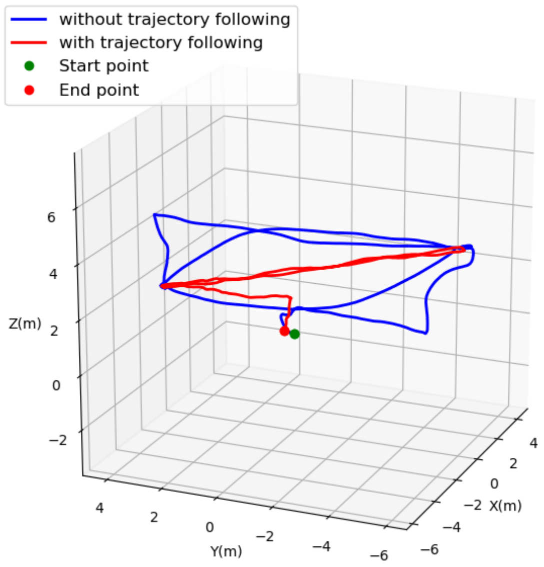

Figure 14: 3D view of a go-to-point operation performed by the real drone, showing the path followed with (red) and without the trajectory follower activated (blue).

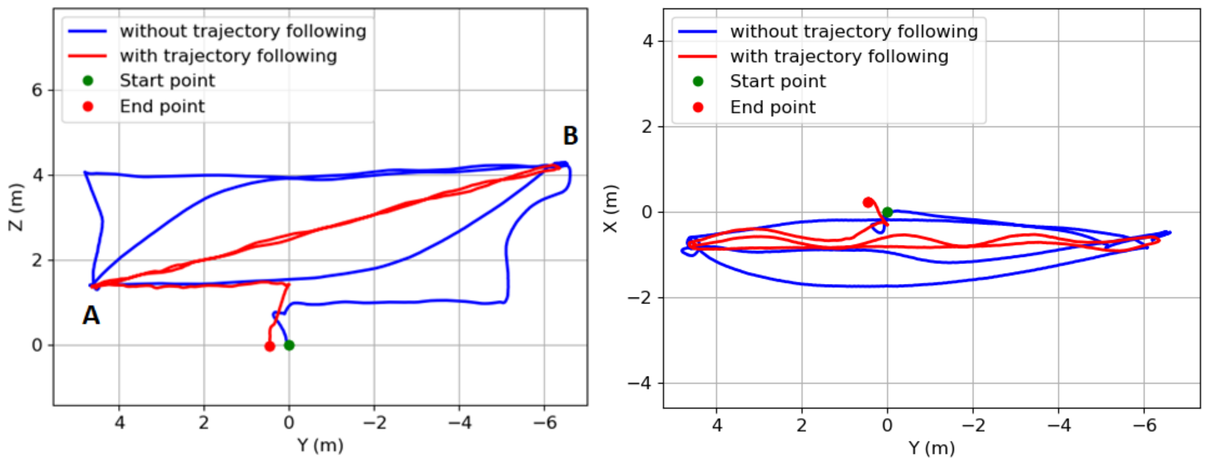

Figure 15: (left) Front and (right) top views of a go-to-point operation performed by the real drone, showing the path followed without (blue) and with the trajectory follower activated (red): the drone flew first to waypoints A and B, which were recorded; later the drone was commanded to go from A to B without using trajectory following and using trajectory following.

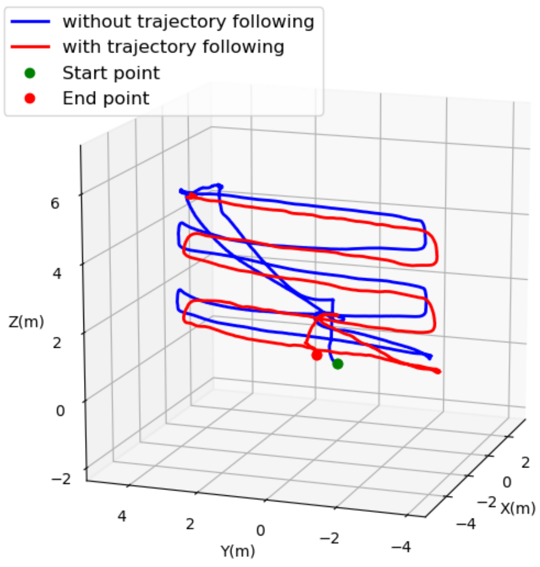

Figure 16: 3D view of an 8 x 5 m sweeping operation performed by the real drone, showing the path followed with (red) and without the trajectory follower activated (blue).

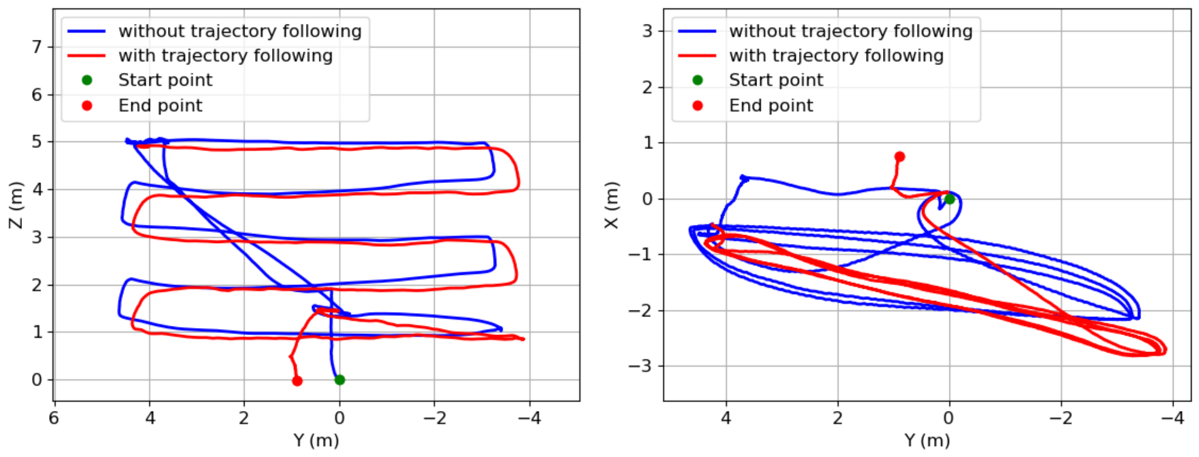

Figure 17: (left) Frontal and (right) top views of a sweeping operation performed by the real drone, showing the path followed without (blue) and with the trajectory follower activated (red): the drone flew first to the upper-left corner of the sweeping rectangle; next, an 8 x 5 m sweeping rectangle was defined and the mission launched first without trajectory following and later using trajectory following.

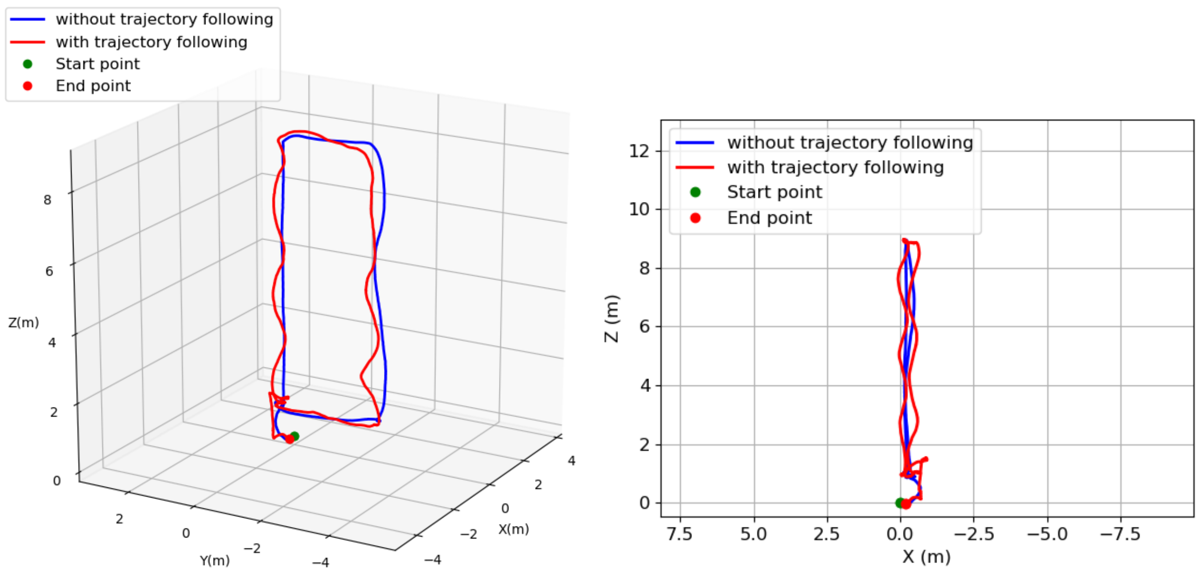

Figure 18: (left) 3D and (right) lateral views of a 9 x 3 m vertical inspection operation performed by the real drone, showing the path followed with (red) and without the trajectory follower activated (blue).

II.2. Trajectory Tracking for the Pioneer X3

In this section, we describe the methods developed for path planning and trajectory tracking for the Blueye X3 ROV.

Path planning is adaptive and done in multiple steps with minimum prior information. The first step is completed using the vessel length and draught. Since the precise ship geometry is not known, a classic vertical lawnmower pattern is first generated based on these two inputs. The goal of this path is to provide full visual coverage. It can be formed in two ways, with vertical or horizontal segments which divide this inspection area in vertical or horizontal slices. Therefore, the area the camera can cover at a specific distance and the wanted visual overlap need to be considered.

As soon as the vehicle starts to follow the pattern to inspect the hull, the path needs to be updated for the drone to keep a constant distance to the hull. It needs to be constant to ensure good visual coverage. If the distance varies, spots will be missed if the drone is too close or details could be missed if it is too far away. Over time, with a constant distance, the pattern should adapt to the actual shape of the ship. A multibeam forward-looking sonar is deployed to achieve this, imposing the constraint to always be facing the hull. By assuming that most parts of the structure are locally flat, it becomes possible to perform line detection based the acoustic image from the sonar measurement and similar to a wall following problem.

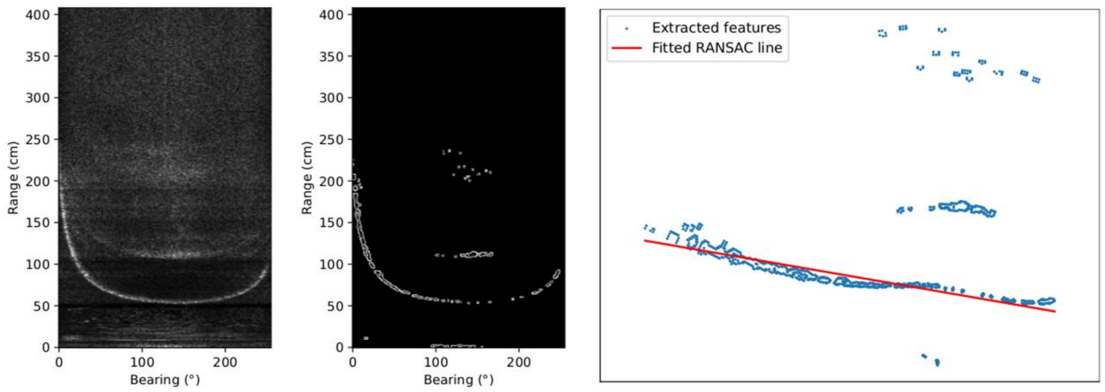

Figure 19: Processing of the sonar scans and line detection.

The estimation of the hull orientation and position with respect to the vehicle is done based on line detection methods on the acoustic data. We first apply an edge detection operator, The Canny edge detector, to sparsify the scan and obtain a better distribution of the features on the lines. Then the line is detected using RANSAC. Using the line's geometric details, representing the wall locally, it is possible to place it in the ROV reference frame and obtain the desired pose that satisfy the constraints.

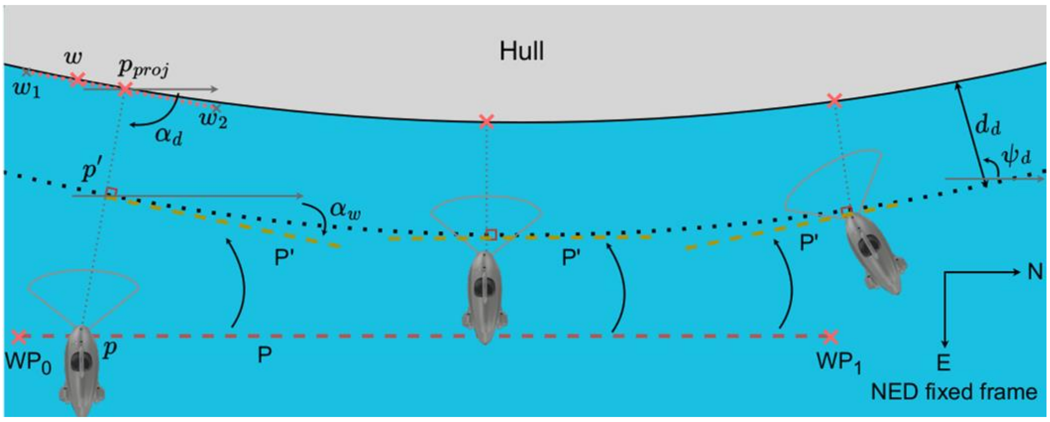

Figure 20: Path adaptation along the hull.

With the position of the ROV and the hull known, the initial path can be updated to a new path to adapt to the structure in front of the ROV. For each new sonar measurement, the path is translated and rotated. This is achieved by considering a desired distance to the hull and a normal orientation to the local wall.

The geometry that is involved is displayed in **Figure 20**, with a wall that has a slight curvature, requiring the vehicle to adapt. It is possible to observe the original path in red and the adapted one in yellow, in 3 different positions along the hull.

The vehicle is controllable in 4 DOF, where the heave motion is decoupled from the horizontal DOF and controlled using a vertical thruster. For the path-following problem, depth control is solved independently from the guidance problem in the horizontal plane.

Manoeuvring-based guidance with constant bearing nonlinear approach is used here. It allows precise control in slow speed. Also, it provides more convenient specification of the speed along the path as well as a better compensation of the heading and depth errors during the inspection. These two errors are especially important for the inspection procedure, and priority is given to them over positioning in the horizontal plane. When the errors are considered as too large, the inspection motion should be paused until they are corrected. This contributes to the inspection quality.

II.3. Trajectory Tracking for the Crawler

For the crawler, trajectory tracking is defined on the surface of the structure being inspected. The crawler receives its pose information (computed in T4.1) and works on a path defined on the triangle mesh representing the structure. As a result, a path is defined as a list of segments, each defined in the plane of a supporting triangular facet. This facet defines a normal vector, which, after a cross product with the direction of the segment, defines a local planar frame attached to the trajectory segment, with its origin at the start of the segment (i.e. the point closest to the start of the trajectory).

In opposition to the MAVs and AUVs, the crawlers are essentially 2D agent working on a local manifold. As such, their trajectory tracking control law is using a 2D controller inspired from Stumm *et al.*.

II.3.1. Definitions and Theory

Let us consider, at time t, the pose $T_3(t) \in SE(3)$ of the crawler in the world frame. Let us assume that the trajectory segment $Si = (Bi, Ei)$ in $\mathbb{R}^3 \times \mathbb{R}^3$ is currently being tracked, with an attached local frame $Ri = (Bi, qi)$ in $SE(3)$. We can now define $T_2(t) = T_3(t) \cdot Ri^{-1}$, the pose of the crawler in the local frame.

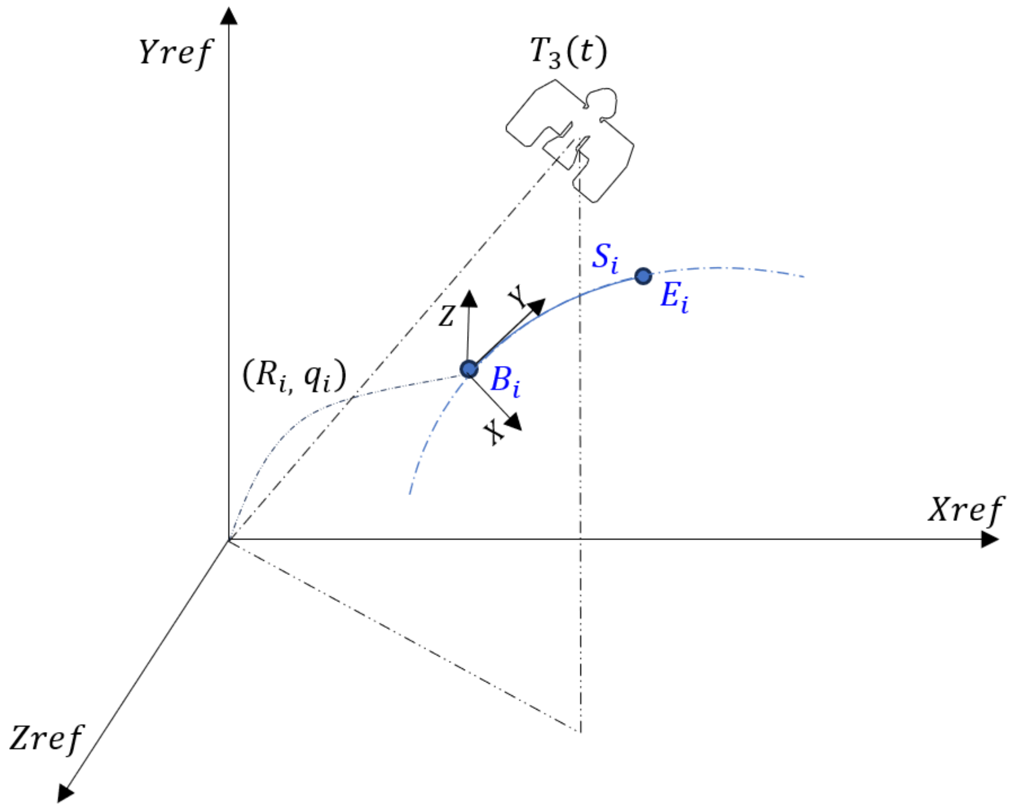

Figure 21: Visualization of the Crawler's Trajectory Segment and Local Frame in SE(3).

Let us extract the 3D translation coordinate and Euler angles from $T2(t) = [x, y, z, roll, pitch, yaw]$. If we assume that the surface is locally smooth and flat around Bi, then the z, roll and pitch component of $T2(t)$ can be neglected, and $T_2(t)$ can be approximated as $\widetilde{T_2}(t) \ = [x, y, 0,0,0, yaw]$. The objective of the path follower is to bring y and yaw to zero.

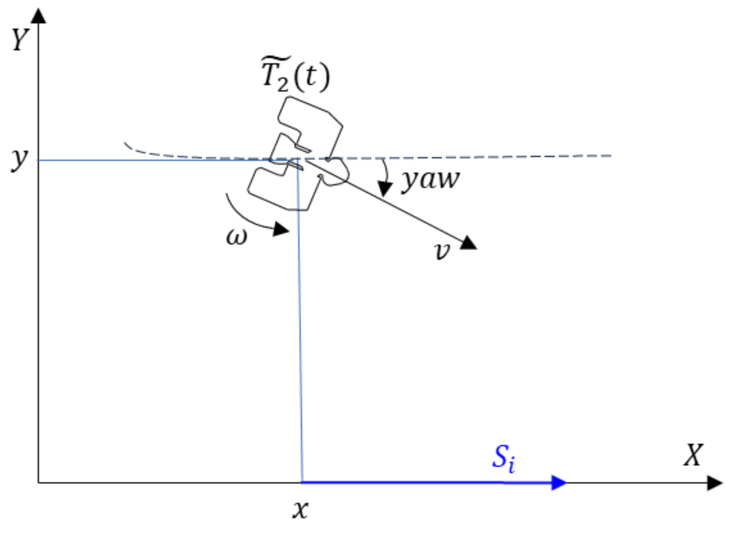

Figure 22: Visualization of 2D Path Definition for Path Following.

Following Stumm *et al.*, the control law we implement on the crawler is producing a pair (v, ω) of linear velocity and angular velocity, define as follows:

$$\omega = k_y \cdot y + k_{yaw} \cdot yaw$$

$$v = v_{ref} \cdot e^{-\left(\frac{\omega}{\omega_0}\right)^2}$$

This control law drives at constant speed while no rotations is required and brings y and yaw to zero on an infinite segment.

If $x > ||E\_i - B\_i||$, then the current path segment is considered complete and, if the path is not completed, the next segment is now considered as a reference.

II.3.2. Mission management

The *mission_client mission_server* enable inspection of rectangular surfaces in a 3D mesh using the *task_manager_crawler* package in ROS.

- *mission_server*: start_mission Service

  The mission_server contains inspection functions responsible for inspecting rectangular surfaces. The mission_server provides a ROS service called start_mission. It receives requests from the mission_client. The start_mission service takes input parameters such as x, y, z, width (W), and height (H) to define the rectangular surface to be inspected. Based on the specified 3D surface, the service calls the appropriate inspection function. If an invalid shape is provided, an error message is logged. Each function contains tuned parameters which have been validated on the specified the following 3D structures.

  - 'Storage tank': storage tank within the Gazebo environment.
  - 'JCColoumb': a scanned ship model imported to Gazebo environment.
  - 'Moebius': similar to the storage tank, the moebius strip has been virtually modelled and imported to Gazebo environment.

  rectangle_mission: The rectangle_mission function performs the inspection of a rectangular surface. It takes parameters such as x, y, z, width (W), height (H), step, and iterations. The function subscribes to the pose topic to receive the initial pose of the crawler. After obtaining the pose, the crawler moves to the designated starting position using the GoToMesh function. A series of vertical transects are then performed to fully inspect the rectangular surface. The number of iterations is determined based on the surface width, and the step size is adjusted accordingly. If the final step does not cover the entire surface, the step size is further adjusted to ensure complete coverage.

- *mission_client*: Sending Requests

  The mission_client imports the StartMission service from the task_manager_crawler.srv package. It sends a request to the start_mission service, providing the desired surface coordinates and dimensions. The success flag is returned as the service response.

handle_start_mission: The handle_start_mission function in mission_server is responsible for calling the appropriate inspection function based on the specified structure. It returns the success flag as the service response.

II.3.3. Experiments and results

II.3.3.1. On storage tank

In the simulation conducted on a virtual storage tank environment described in **Figure 21**, the automated inspection system showcased remarkable adaptability and accuracy. The 3D path plot in the **Figure 22** showed in a clear picture of how the crawler moved around the storage tank, following the desired inspection path. Similarly, the 2D path plot gave a bird's-eye view of the inspection path, demonstrating how the crawler navigate in the entire 6-meter width and 2-meter height of the surface.

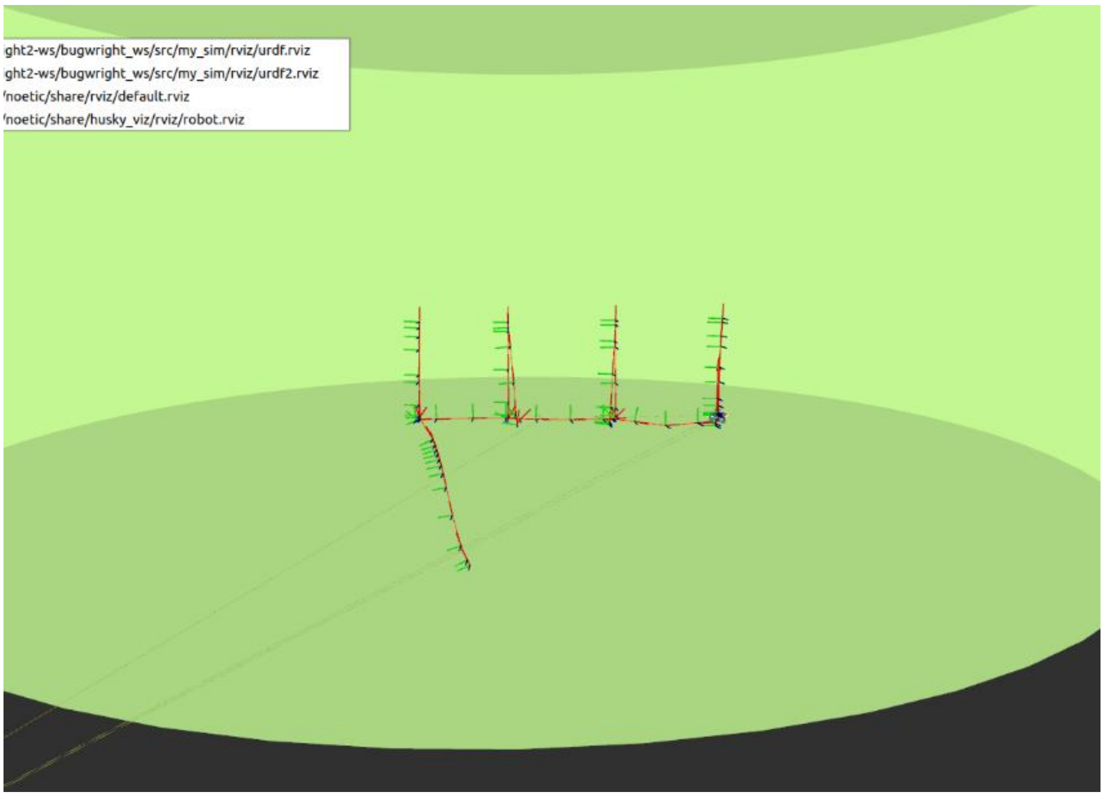
ght2-ws/bugwright_ws/src/my_sim/rviz/urdf.rviz
ght2-ws/bugwright_ws/src/my_sim/rviz/urdf2.rviz
'noetic/share/rviz/default.rviz
'noetic/share/husky_viz/rviz/robot.rviz

Figure 23: Mesh and Inspected Path Visualization for Storage Tank Inspection in Rviz.

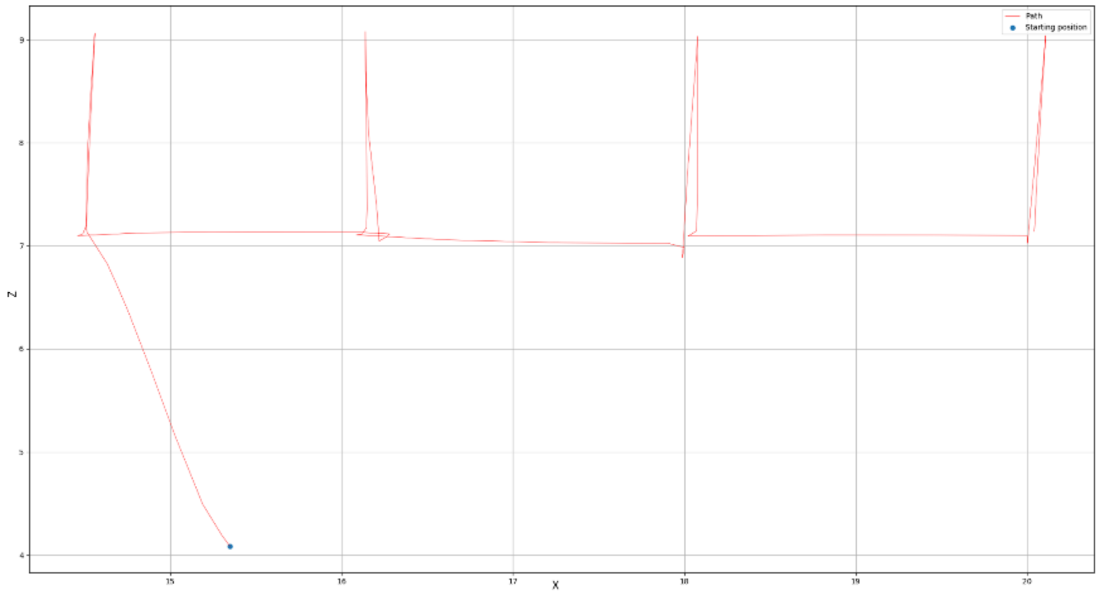

Figure 24: 2D Path Plot for Storage Tank Inspection.

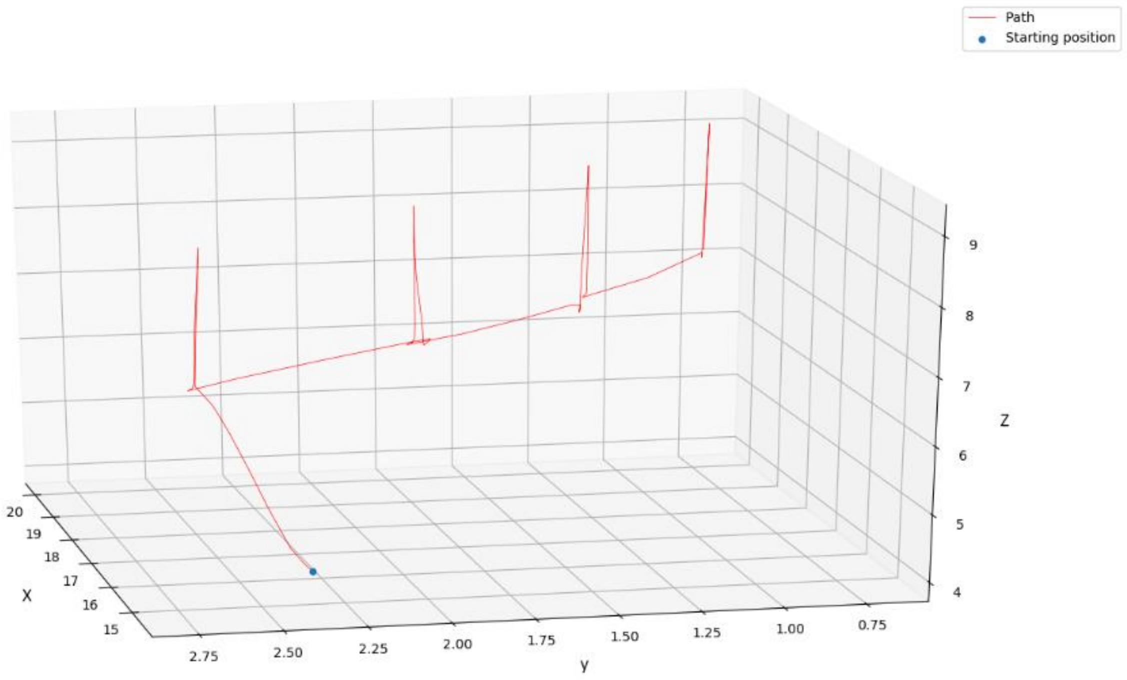

Figure 25: 3D Path Plot for Storage Tank Inspection.

II.3.3.2. On moebius strip

The simulation test conducted on a Moebius strip structure demonstrated the system's ability to tackle unconventional shapes. The 3D path plot in **Figure 26** showed the XYZ path the robot followed as it moved around the continuous loop of the moebius strip, effectively covering the desired inspection surface. The 2D path plot in **Figure 25** provided a unique perspective on the robot's movement, emphasizing its traversal across XZ axis. These results highlight how adaptable the system is at moving around surfaces that aren't typical or standard.

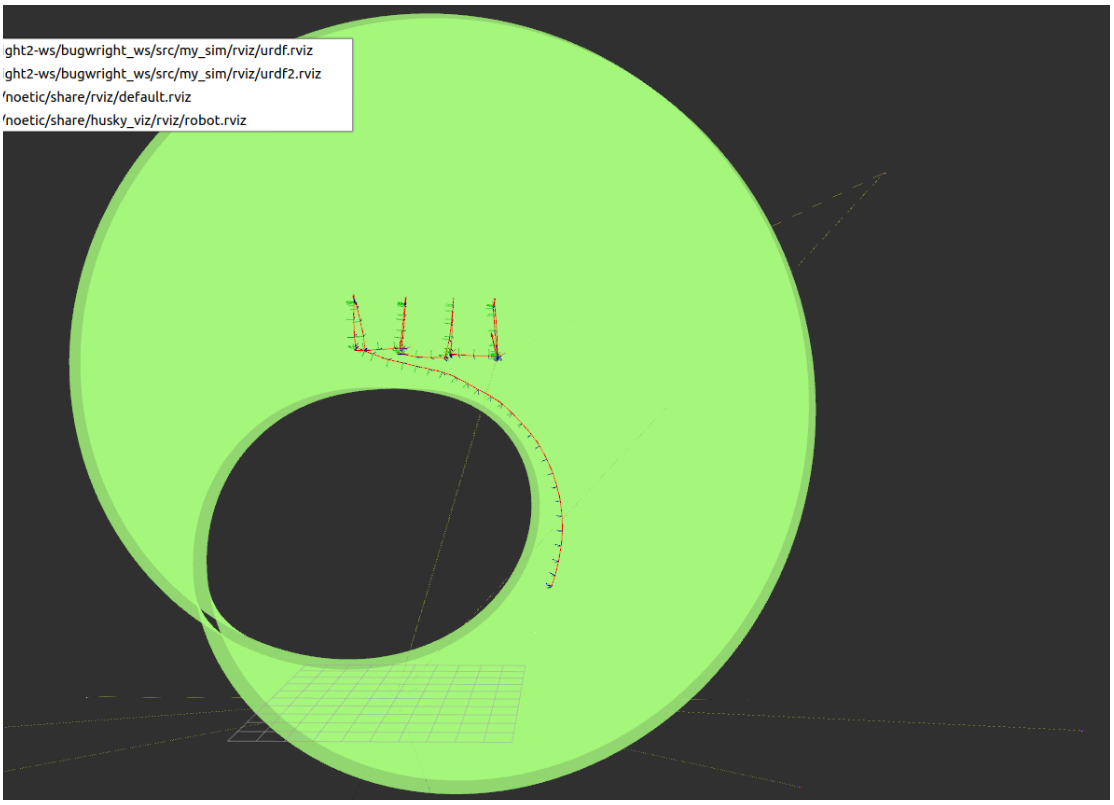
```
ght2-ws/bugwright_ws/src/my_sim/rviz/urdf.rviz
ght2-ws/bugwright_ws/src/my_sim/rviz/urdf2.rviz
/noetic/share/rviz/default.rviz
/noetic/share/husky_viz/rviz/robot.rviz
```

Figure 26: Mesh and Inspected Path Visualization for moebius strip Inspection in RViz.

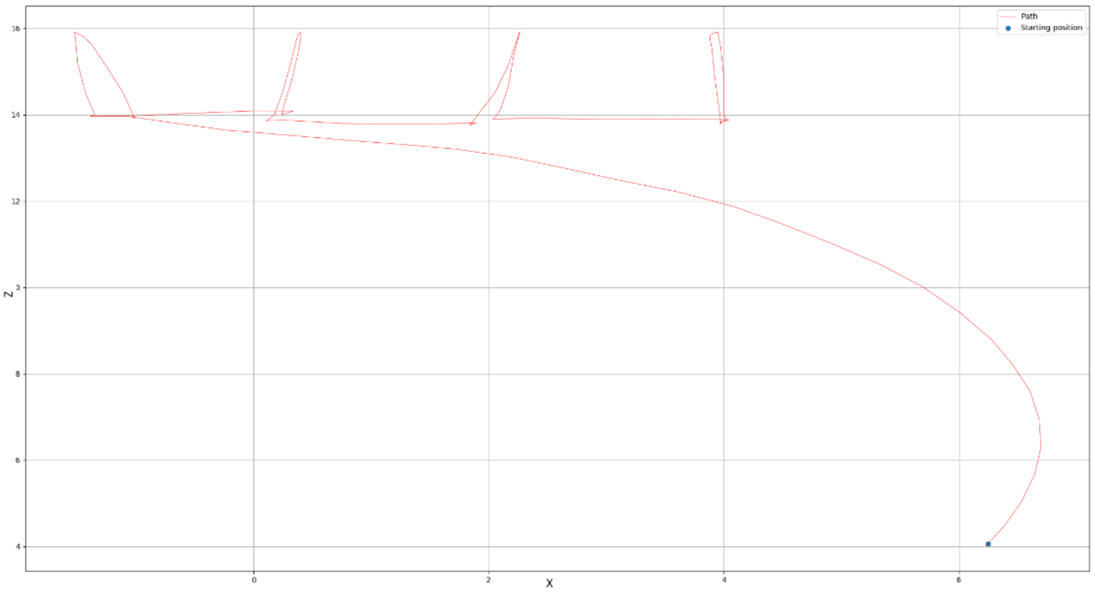

Figure 27: 2D Path Plot for moebius strip Inspection.

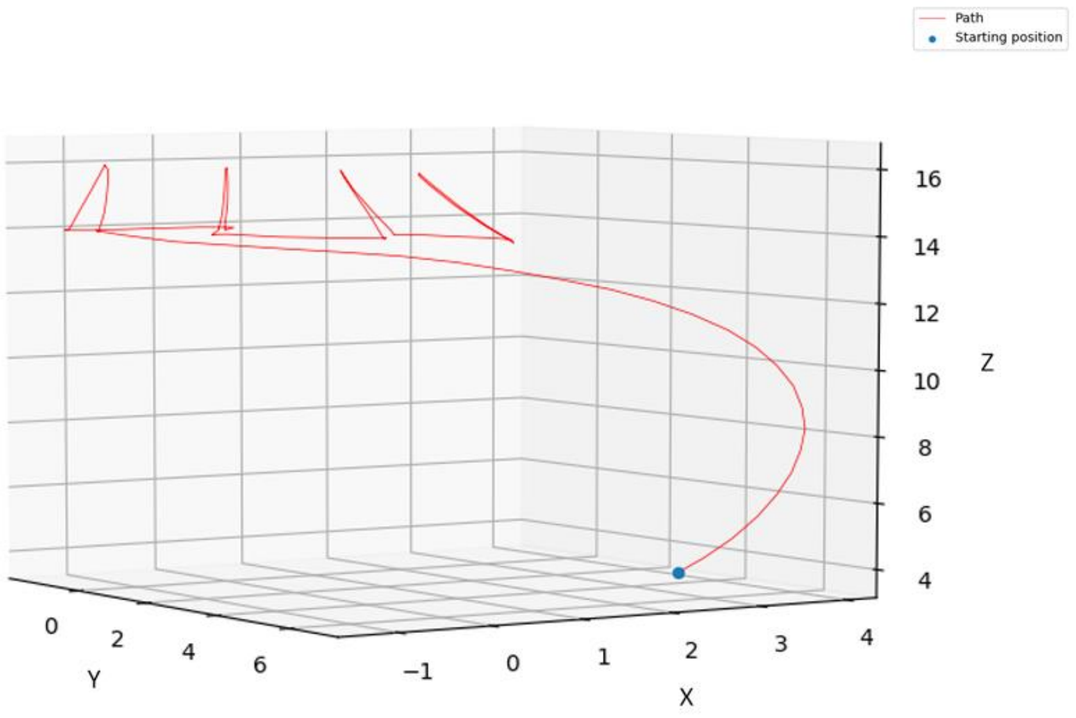

Figure 28: 3D Path Plot for moebius strip Inspection.

II.3.3.3. On the hull of the trawler JCCoulomb

The simulation test performed on a virtual ship structure, reconstructed from a real ship scan, highlighted the versatility of our robotic inspection system. The 3D path plot in **Figure 28** showcased the robot's seamless navigation across the ship's surface, capturing its movement along the length of the vessel. The 2D path plot in **Figure 27** illustrate the comprehensive coverage achieved during the inspection. The presence of slight drift in the simulation tests can be attributed to the inherent difficulty, and in some cases, the impossibility, of accurately simulating the behaviour of a magnetic robot operating on a surface while maintaining strict orthogonality to that surface. The results underline the system's robustness in adapting to varied geometries, thereby showcasing its potential for applications in ship maintenance and quality assessment.

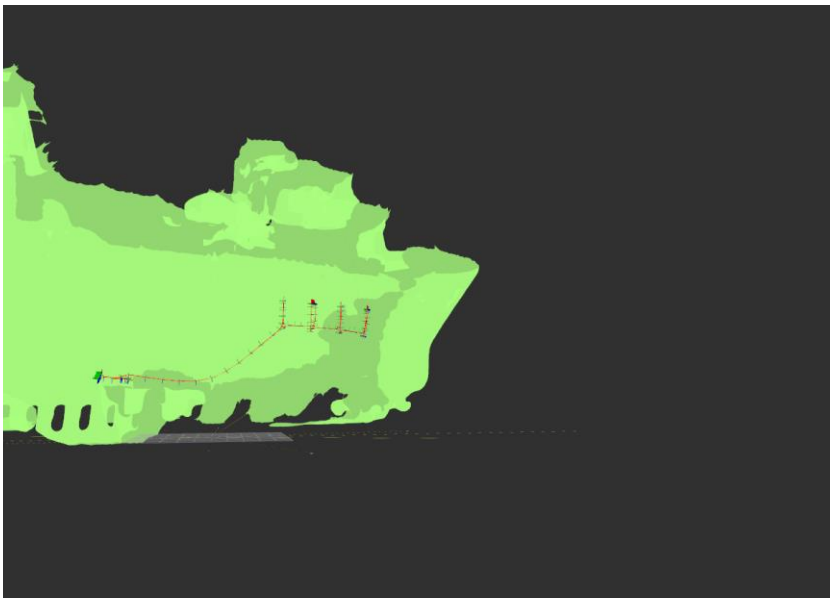
Figure 29: Mesh and Inspected Path Visualization for JCCouloumb Inspection in RViz.

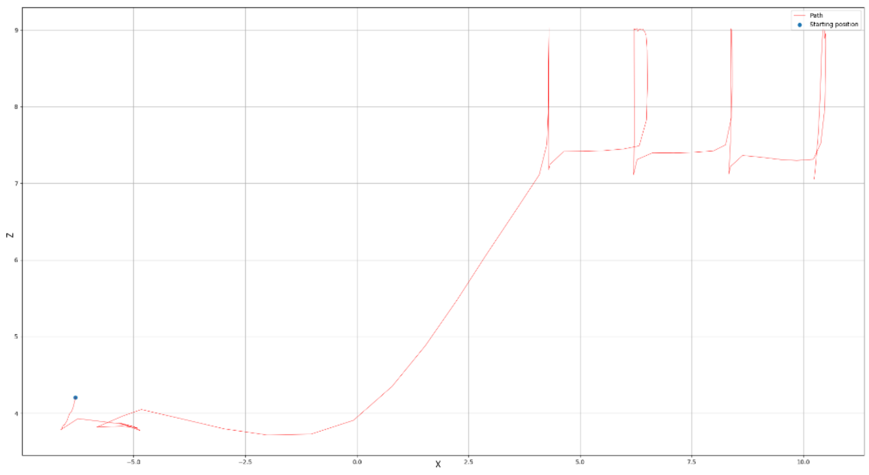

Figure 30: 2D Path Plot for JCColoumb Inspection.

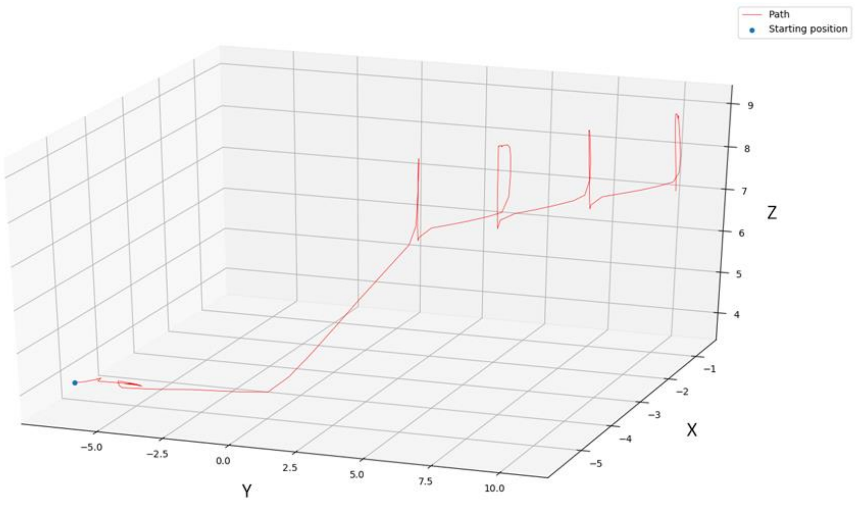

Figure 31: 3D Path Plot for JCColoumb Inspection.

II.3.4. Integration with the unified controller (T5.1)

As a reminder, for the crawlers, the unified controller is based on the ROS task manager developed by CNRS. This task scheduler combines a set of behaviours and control tasks that can be combined in various missions.

Specifically, the following tasks have been implemented:

- GoToMesh: using the path planner described in T5.3 and the controller defined above, the crawler can reach a specific pose on the mesh of the inspected structure.
- VerticalTransect: for very simple inspection missions, in particular on storage tanks, it is sufficient to just drive vertically for a known distance. The same controller as above is defined, but with a single segment as reference, and the IMU as main source of orientation.
- HorizontalTransect: also for simple inspection missions, this task allows driving horizontally on the inspected surface while maintaining a desired height, using the localization and the IMU as reference.
- DriveOnStiffener: the same control principle has been applied to the drive on the top face of stiffeners for hold inspections. The challenge here is that the stiffener width is similar to the robot width, so a very tight control is required. The stiffener itself is extracted from the LIDAR point cloud.
- DriveBetweenStiffeners: The surface between two stiffeners is named a “frame”. Using the LIDAR, the robot pose in this space can be computed and the same control law can be applied to conduct an autonomous inspection of the frame.
- DriveOnStiffenerSide: Similar to the operation on a stiffener, this control law drives on the side of the stiffener, which is a very narrow space that also need to be inspected.

Results for real crawler on a metal plate:

In the real-world experiment conducted to inspect a metal plate, the automated inspection system exhibited tangible performance. The 3D and 2D path plot in Figure 30 and Figure 31 illustrates the crawler's movements across the metal plate in the global frame, effectively inspecting the designated 5 meter by 2-meter area. it accentuated the systematic coverage achieved during the inspection process. These visualizations substantiate the system's efficacy in practical scenarios, affirming its potential for real-world application. It is important to acknowledge that the path during the inspection may not have been perfect due to the presence of noise in the UWB range measurements used in estimating the position of the crawler.

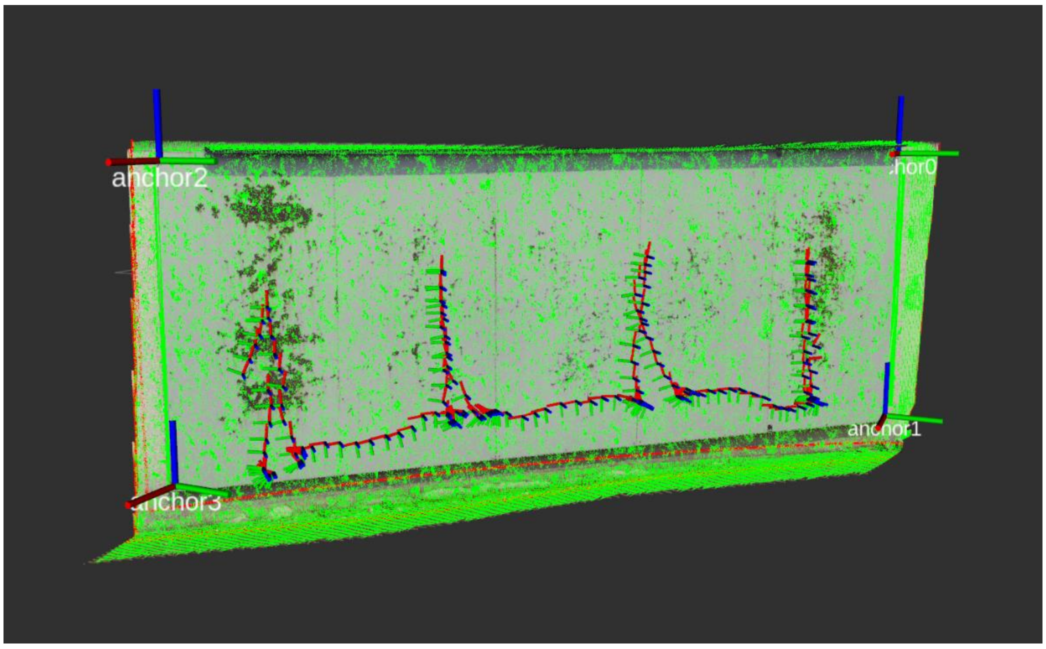

Figure 32: Mesh and inspected path visualization for metal plate inspection in RViz.

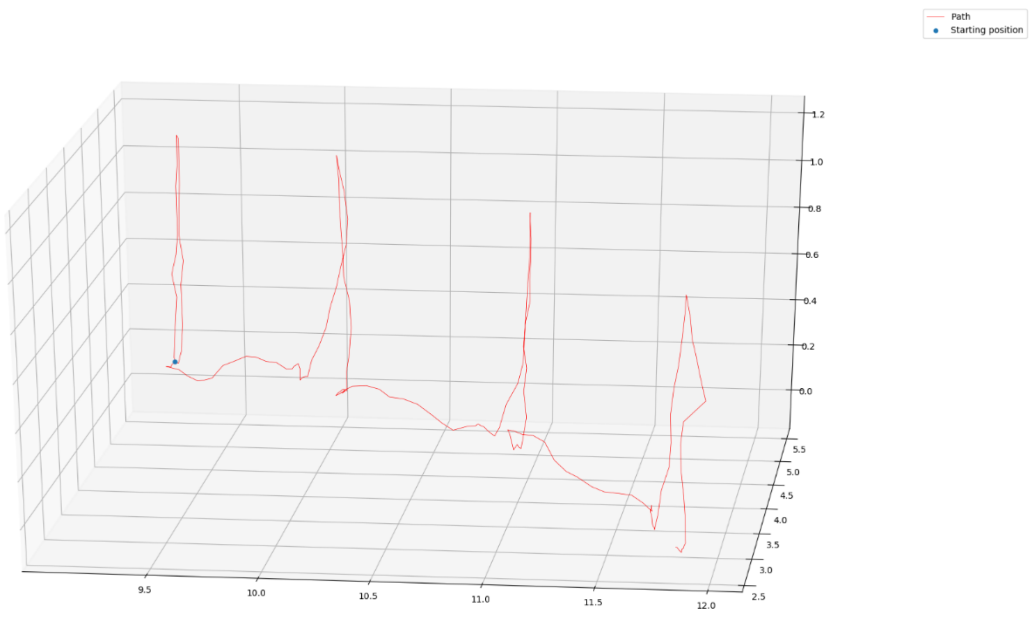

Figure 33: 3D path plot for metal plate inspection.

The following figures, presented as **Figure 32** and **Figure 33**, depict a view of the inspection path conducted as part of another experiment utilizing the IMU sensor. They illustrate the crawler's movement as it navigates the metal structure during this inspection, covering a distance of 2 meters in width and 1 meter in height.

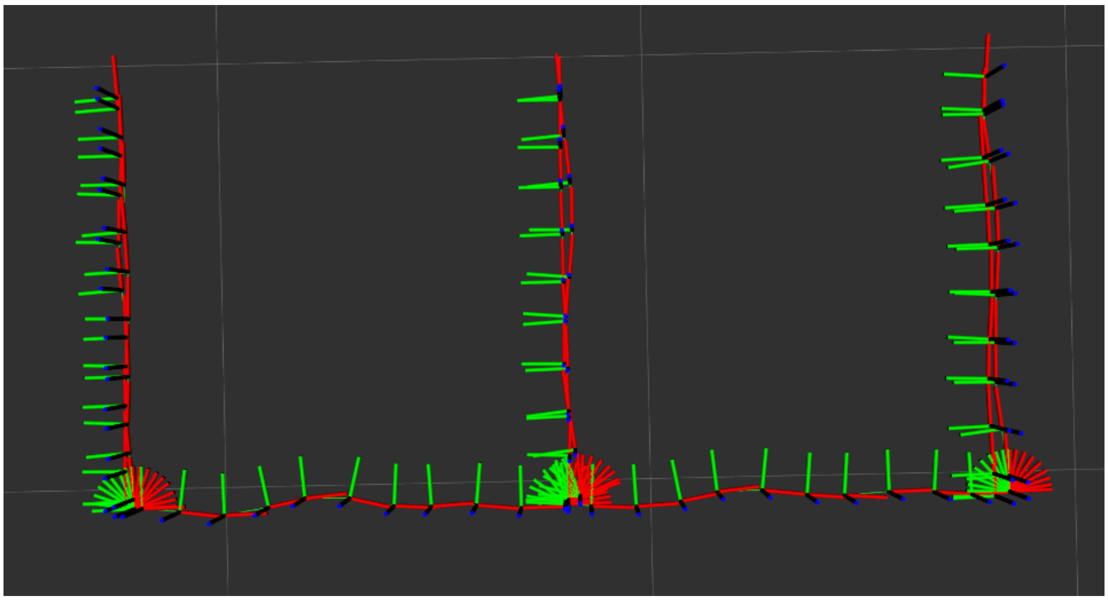

Figure 34: Inspected path visualization for metal plate inspection in RViz.

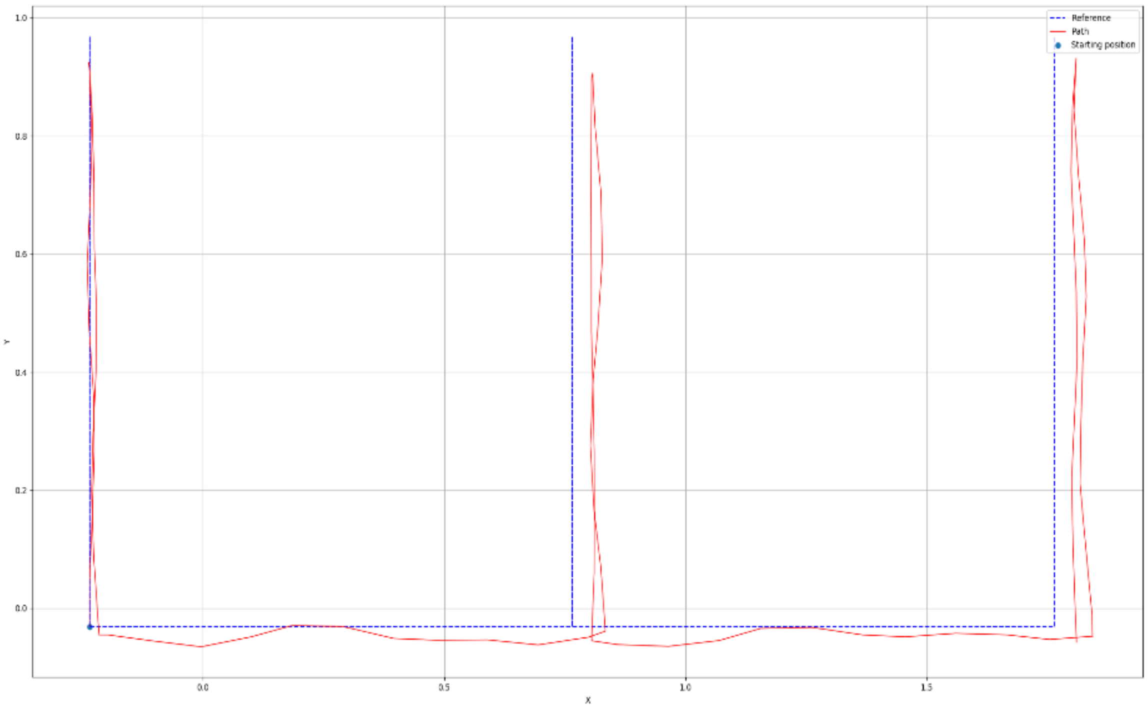

Figure 35: 2D path plot for metal plate inspection.

III. Obstacle Avoidance

III.1. Obstacle Avoidance for MAV

In this section, we describe the control side of the collision prevention and obstacle avoidance functionalities running onboard the inspection drone. The perception side can be found as part of deliverable D4.2. There we describe how the raw data from the obstacle-detection sensors, i.e. a 3D laser scanner for the case of the inspection drone, is processed to produce a suitable representation of the relevant obstacles in the vicinity of the platform during operation.

As already mentioned, the aerial platform is fitted with obstacle avoidance capabilities as part its control architecture based on robot behaviours and the SA paradigm. Among all the behaviours that collaborate in the generation of the final speed command, two are involved in the obstacle avoidance mechanism:

- **Attenuated go**. This behaviour is in charge of accomplishing the user intention so that it provides the velocity desired by the pilot/operator but attenuated towards zero when the platform approaches an obstacle. The closer the obstacle, the greater the attenuation, so that the user's desired speed is completely attenuated when the platform is very close to the obstacle.
- **Prevent collision**. This behaviour is in charge of separating the platform from the surrounding obstacles by means of a suitable repulsion velocity. To be precise, a repulsion vector is computed for all obstacles surrounding the platform. The closer the obstacle is, the greater the repulsion vector. Once all the repulsion vectors have been calculated (for all surrounding obstacles detected by the obstacle perception sensors, e.g. the 3D laser scanner the UIB inspection drone is fitted with) they are added together to obtain a single repulsion vector/velocity.

These two behaviours make use of the distances d_{att} and d_{min}, where $d_{att} > d_{min}$. The distance d_{att} is the distance from which the user's desired speed starts to be attenuated, while d_{min} is the minimum distance to obstacles allowed. **Figure 36** shows the three situations that can take place depending on where the aerial platform is situated in accordance to the two distance parameters.

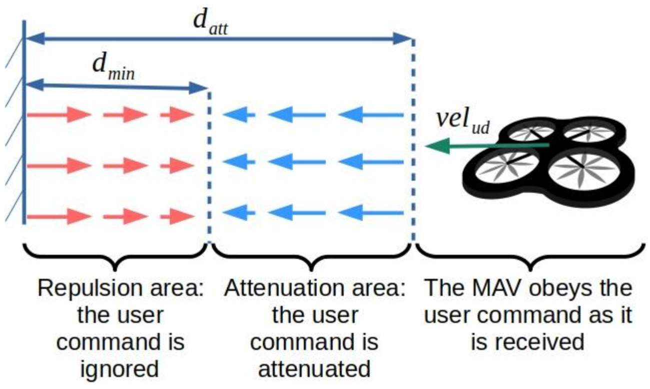
d_{att}
d_{min}
vel_{ud}
Repulsion area: the user command is ignored
Attenuation area: the user command is attenuated
The MAV obeys the user command as it is received

Figure 36: Distances involved in the collision prevention mechanism.

Given these two distances and considering a single obstacle situated at a distance d_o from the aerial platform, three different situations may arise:

1. $d_o > d_{att}$: the obstacle is not considered by the obstacle avoidance mechanism.
2. $d_{min} < d_o < d_{att}$: only the *attenuated go* behaviour is activated using the distance d_o to attenuate the user's desired speed in the case this points towards the obstacle. As a result, the platform can approach the obstacle with a speed attenuated by $\lambda \in [0,1]$, which is calculated as:

$$\lambda = min\left(1.0, max\left(0.0, \left(\frac{d_o - d_{min}}{d_{att} - d_{min}}\right)\right)\right)$$

so that the attenuated desired velocity results:

$$vel_{att} = \lambda \cdot vel_{ud}$$

3. $d_o < d_{min}$: both the *attenuated go* and the *prevent collision* behaviours are activated. The first one completely attenuates the user's desired speed, i.e. it becomes 0, in case this points towards the obstacle. The second behaviour makes use of the distance d_o to create a repulsion vector to move the platform away. As mentioned before, a repulsion speed vector rep_p is created for each obstacle, where the magnitude of the repulsion vector is computed as:

$$\|rep_p\| = K \cdot (d_{min} - d_p)$$

where K is the repulsion factor and the direction of the vector is pointing away from the obstacle. The final repulsion vector vel_{rep} is computed as the mean of all the repulsion vectors defined for each point separately. In addition, the magnitude of vel_{rep} is limited to the maximum speed allowed.

Considering both behaviours, the final velocity command provided by the obstacle avoidance mechanism results into:

$$vel_{cmd} = vel_{att} + vel_{rep}$$

Notice that these two behaviours not only prevent the platform from colliding with surrounding obstacles but also separates the aerial vehicle from dangerously approaching moving objects.

III.2. Obstacle Avoidance for the Pioneer X3

The use of the Pioneer X3 underwater ROV, its intended to do hull inspection maintaining a relative constant distance to the ship hull, facing it. The procedures described in Section II.2 already adds the ability to overcome, or divert, obstacles that are seen in the hull has long as they are in the field-of-view of the sensors (camera and FL-MBS).

Because the survey is done facing to the hull, there are two mitigations that can be added that will allow the trajectory tracking to avoid collisions:

1. Using all the collected data by all the robotic surveys, to feed potential obstacles to the robots; and
2. Using IMU data to identify out of the field-of-view collisions, and react to that to overcome it (e.g. reversing in order to “see” it and bypass it).

III.3. Obstacle Avoidance for the Crawler

The crawlers are relatively slow robot. They can start and stop instantaneously but their rotations is expensive in energy, potentially damaging for the paint of the structure, and a source of slippage. As a result, there is no need for a reactive obstacle avoidance on the crawler. The crawler also drives on a surface which is fairly well known beforehand and with very rare and sparse obstacles. No dynamic obstacles need to be considered, but both positive and negative static obstacles are possible.

Within these specifications, obstacle avoidance was implemented in a very simple and practical way, using three main components.

- Reactively, only a collision avoidance mechanism is required. Using the available sensors, lidar or simply ultrasonic rangers for the aerial crawler (seen in the **Figure 37** below), acoustic ranger (multi-beam or rotating) for the underwater crawler, we can detect the local obstacle (T4.2) and their distance in front of the crawler. A proportional velocity saturation is then applied to smoothly slow down when approaching an obstacle down to a null velocity to prevent collisions. This approach is sufficient to ensure the safety of the crawler while operating autonomously. Even though the state of the art proposes a lot of solution for obstacle avoidance, this is not something desired by practitioners in this application domain.
- When the crawler is stopped because an obstacle is on the path, we can either replan a globally valid path, or request operator control. Again, despite their low sophistication, these solutions are the preferred solutions for practitioners.

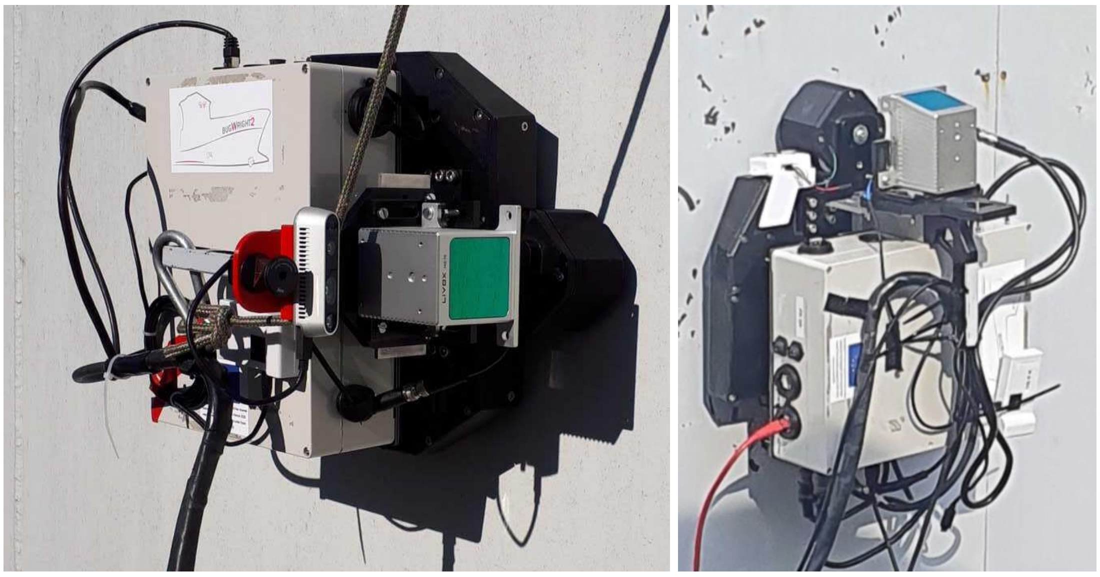
Figure 37 : Crawler Equipped with LiDAR and Ultrasonic Sensors for Collision Avoidance.

IV. Conclusions

Delivery D5.2 describes the different implementations for trajectory tracking and obstacle avoidance in the context of BUGWRIGHT2 hull inspection and storage tanks. This document describes the approach taken for the aerial drones (MAVs) where missions are a sequence of waypoints, and the trajectory is calculated to achieve the waypoints. Collision prevention and obstacle avoidance is embedded in the control loop.

For the Pioneer X3 the planning is done with a mix of onboard and offboard processing of the sensor data and prior information of the length and draught of the ship, generating a vertical lawnmower with the execution using the sensors to do the survey following the contour of the hull. Added obstacle detection and avoidance (also for off sensors field-of-view).

For the crawler trajectory tracking is defined on the surface of the structure being inspected with a path defined on the triangle mesh, resulting on a path defined as a list of segments, each defined in the plane of a supporting triangular facet. The crawlers being relatively slow robots, where rotations are expensive in energy with potential paint damage. For this reason, and with the surface being fairly well known beforehand, no dynamic obstacles need to be considered, but both positive and negative static obstacles are possible.