3D reconstruction improvement by path planning towards physical interaction with a UAV

P.D. (Patrick) Radl

MSc Report

Committee:
Dr. B. Sirmaçek
R.A.M. Hashem, MSc
Dr.ir. F. van der Heijden
Dr.ir. L.J. Spreeuwers

June 2019

021RAM2019
Robotics and Mechatronics
EE-Math-CS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Abstract

Physical interaction with objects by unmanned aerial vehicles (UAVs) is an emerging field in current research. To interact with objects accurately, without damaging them or the UAV itself, exact representations of objects are desirable.
Often predefined three-dimensional (3D) models are provided to meet this constraint. To not have to rely on predefined 3D models, it is necessary to gain them by the UAV "on-the-fly".
A common method to reconstruct online objects or environments is to attach sensors to UAVs and use the output as an input for simultaneous localization and mapping (SLAM) algorithms. In this research it has been investigated if the thereby created 3D models can be improved by multiple coverage.
At first it was necessary to determine an appropriate visual SLAM algorithm, which works reliable with a stereo camera in a simulation. Based on this different coverage path planning (CPP) methods were evaluated if multiple coverage of areas of the object of interest can improve a resulting 3D model. Two were thereby designed as online path planning algorithms to autonomously determine flight directions.
As SLAM algorithm ORB-SLAM2 by Mur-Artal and Tardos (2017) has been chosen and is shown to be suitable. Furthermore, results indicating that multiple coverage improves 3D models, represented as point cloud output of ORB-SLAM2, are provided. Moreover, potential for online path planning algorithms to improve flight time could be found.

Contents

List of acronyms

3D three-dimensional

BA bundle adjustment

CPP coverage path planning

CAD computer-aided design

GUI graphical user interface

FOV field of view

ROS Robot Operating System

SLAM simultaneous localization and mapping

UAV unmanned aerial vehicle

VFOV vertical field of view

1 Introduction

1.1 Context

For many years the usage of unmanned aerial vehicles (UAVs) has drastically increased. Originally mainly in military operations, it has also expanded to civilian fields (González-Jorge et al. (2017)). Today they are used for a wide number of fields, for instance, surveillance (Zaheer et al. (2016), Avola et al. (2017)), supporting precision agriculture (Hoffmann et al. (2016)) and visual inspection of civil infrastructure (Ham et al. (2016)).
For a while, this was limited to contactless flights using different sensors to orientate and avoid collisions with objects and other UAVs. Recently also tasks that include physical contact with objects are getting into the focus of research (Na and Baek (2016), Mattar et al. (2018)).
In the case of structural inspection of objects, like buildings or other big (partially) difficult to reach structures, some tasks require contact to their surfaces. As a consequence, people might have to reach those. For instance, rope access to wind turbines as shown in figure 1.1 is such a case. One easily can imagine the danger of personal injury as well as the very time-consuming efforts which have to be made to safely reach certain parts of the wind turbine. Therefore, it is beneficial to work towards replacing people in as much of such tasks as possible by using UAVs.
Nevertheless, UAV operators would still be necessary. By providing at least partial autonomous execution of tasks by UAVs, it is possible to reduce the operators' working schedule. Thereby, the number of required operators would be lower for the same number of UAVs in operation.
To not damage the UAV or the object of interest during autonomous operation, besides common collision avoidance to other objects, retrieving the distance to it as well as its three-dimensional (3D) representation at high detail is crucial.

Figure 1.1: Two maintenance engineers using rope access at a wind turbine (Schroeder (2012))

1.2 Problem Formulation and Research Questions

To gain a 3D model of the object of interest a common solution is to provide computer-aided design (CAD)-models on beforehand to path planning algorithms. This is common as especially for some industrial applications CAD-models are often available.

As this is not always the case and in general a limiting factor, it would be desirable to avoid it. To achieve this, one has to use simultaneous localization and mapping (SLAM)-algorithms to approximately gain a spatial representation as well as the pose of the UAV relative to it.
These are different in accuracy and not always available as open source and due to this the first research question is facing these issues:

> "Which SLAM algorithm would be useful to create 3D visual models on-the-fly?"

Solving the SLAM-problem, in general, does not restrict one to use visual sensors like cameras but in the case of aerial vehicles, those are a cost-efficient solution to provide contact-free information about the environment and therefore commonly used.
The resulting models as the output of such algorithms are dependent on several factors. One of these is the path along the object of interest is observed. Different poses at which it is investigated might lead to a different approximated model. As they affect the representation of the environment within the field of view (FOV) of the camera.
For being generally able to represent a complete model of an object of interest all its surfaces have to be once covered when flying the path. Covered means being visible within the FOV of the UAV's camera.
Those coverage paths often try to cover parts of the object only once, to keep the flight time short, and imply to gain thereby a sufficient level of detail in the object of interest's representation.
Due to this, if introducing multiple coverage of parts of the object of interest can improve the resulting model, currently represents a research gap.
Investigating it should answer the following research question:

> "Does multiple coverage of an object's surface lead to an improved 3D visual model of it?"

As rescanning using the same coverage path leads to a multiple in terms of flight duration, more time efficient methods would be desirable. These should avoid manual control input and therefore provide some degree of autonomy in making decisions about parts of the objects being rescanned. Formulated as a research question:

> "In order to autonomously improve the 3D visual model of an object, is it possible to determine suitable flight directions on-the-fly, which lead also to a reduction in flight time?"

All these investigations in this research are answered by using a simulated environment. One reason for this is the safety for not flying a UAV with experimental control input in the real-world. Moreover, creating the necessary environment in real would require allocating relatively huge indoor space.
To get close to the later real-world application with physical interaction the model of "betaX", shown in figure 1.2 is used as UAV in the simulation. It is a hexarotor, with fixed angle tilted rotors to gain full actuation, providing a manipulator for physical interaction at its front, rigidly connected to its base frame.
Also, the specifications of a real stereo camera, which is possible and intended to be mounted on the "betaX", are simulated, namely the Kayeton KYT-U130-3R60ND. For the simulation, two cubes are used as dummy representation of its two image sensors to identify their pose.
The object of interest, which is scanned to be reconstructed as a 3D-model, more precisely a point cloud, is a cylinder. A cylinder has homogeneous surfaces, which provide the possibility to keep a uniform distance when the UAV is flying concentric around it to inspect the lateral surface.

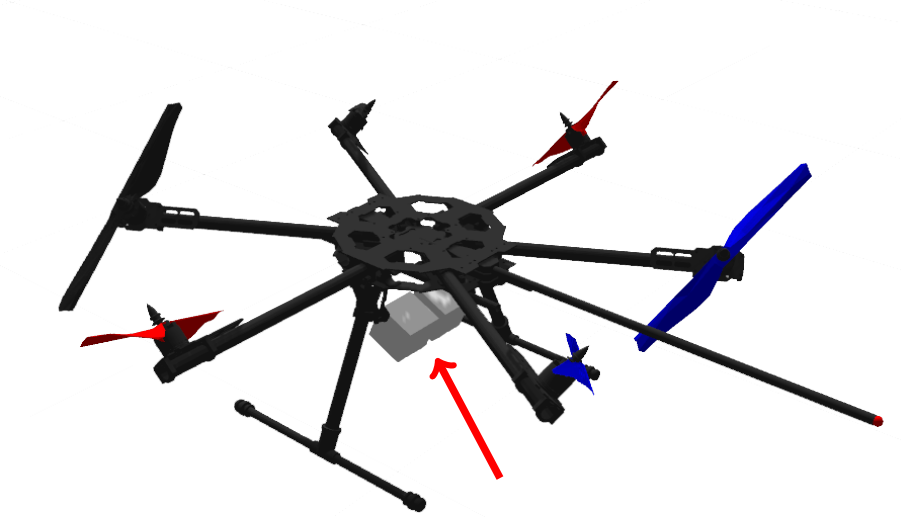


Figure 1.2: UAV "betaX" with the simplified representation of the stereo camera as two cubes below the central base frame (the red arrow is pointing at them)

1.3 Related Work

As, according to the best knowledge, multiple coverage of object surfaces represents a research gap, in this section coverage path planning (CPP) for single coverage are presented. Moreover, previous works on visual SLAM-algorithms and their evaluation are also covered in this section.

1.3.1 Visual SLAM-Algorithms

Currently, there are three common open source SLAM-algorithms providing support for stereo cameras: ORB-SLAM2 by Mur-Artal and Tardos (2017), RTAB-Map by Labbé and Michaud (2018) and S-PTAM by Pire et al. (2017). All three are graph based SLAM-algorithms. They have been also independently evaluated against each other.
In the comparative study of Gaspar et al. (2018), they stated that S-PTAM might lack on low times between image input. Further that RTAB-Map is in general very dependent on the environment. Regarding ORB-SLAM2 they showed that in the majority of their experiments it provides a good motion estimation as well as lower error. Although it shows higher CPU utilization compared to the others, it was the one being able to reproduce a trajectory close to "ground truth" in most cases. "Ground truth" denotes the exact position/pose or path that the algorithms are trying to estimate. Their tests were based on several publicly available datasets to benchmark, namely the KITTI dataset of Geiger et al. (2012), the MIT Stata Center dataset of Fallon et al. (2013) and the New College vision and laser data set of Smith et al. (2009). Those datasets provide stereo image streams and the according "ground truth".
In difference to the previous, the work of Giubilato et al. (2018) was not based on existing datasets, as they created their own instead.
Giubilato et al. (2018) noticed that although ORB-SLAM2 might have the highest drift away from the "ground truth" path over distance, its robust loop-closure makes it the only one capable of estimating the full trajectory. They considered RTAB-Map and S-PTAM to fail due to their low number of tracked features.
As a result of these insights, ORB-SLAM2 is evaluated in chapter 3 on its suitability for this research.

1.3.2 Coverage Path Planning

Galceran and Carreras (2013a) showed that CPP by unmanned vehicles in current literature seems to focus mostly on methods where the environment can be modeled as a planar surface and solving the problem in two dimensions.
But the field is wide, reaching from covering building fronts as in the work of Teixeira and Chli (2016) to in-detail inspection of the ocean floor researched by Galceran and Carreras (2013b).
A common coverage path to cover a single object with a UAV seems to be flying concentric circles around it and increasing height.
One representative work with such object-centric coverage paths is Cheng et al. (2008). This was reasoned as they decided to simplify single buildings also as cylindrical coverage models, as shown in figure 1.3. Another research investigating a single detached object has been done

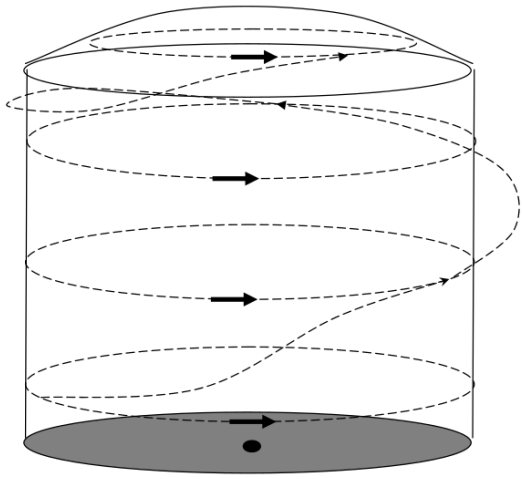


Figure 1.3: Cylindrical coverage model with coverage path as dashed line as used by Cheng et al. (2008)

by Mansouri et al. (2018). They also follow the same approach of rotating around the object and increasing height to fully cover it.
To the best knowledge, a work involving multiple coverage of areas of the object of interest has not been published yet.

1.4 Report Organization

The other parts of this thesis are presented in the following chapters and appendices.
Chapter 2 provides the necessary background knowledge to the reader. Topics covered are path/trajectory planning, in particular also CPP, as well as description of the SLAM problem. Furthermore, the components and their functionalities of the, in this research, used SLAM algorithm ORB-SLAM2 by Mur-Artal and Tardos (2017) are described.
In chapter 3 the first experiments are presented. Those determine the performance of ORB-SLAM2 and its suitability within the simulation.
The following chapter 4 consists of sections introducing and evaluating different CPP methods for the 3D reconstruction as a point cloud of the object of interest in the simulation.
Chapter 5 is recapitulating the relevant insights of the previous two chapters and relating it to the research questions of this thesis.
General suggestions for future work are presented in chapter 6.
Details of the implementation and used components can be found in appendix A.
Appendix B shows all textures applied to the object of interest.

2 Background

In this chapter, relevant background information is presented. First in section 2.1 the concepts of path planning and trajectory planning. It is also specifically elaborated towards coverage path planning (CPP).
The problem regarding a robot following a path and thereby localizing itself as well as mapping the environment is covered in section 2.2. The algorithm, namely ORB-SLAM2, which is used to deal with this problem in this thesis, is described in section 2.3.

2.1 Path Planning

According to Chung et al. (2016), "a path is a geometric representation of a plan to move from a start to a target pose". Moreover, they define finding a path without colliding with obstacles as path planning.
They define trajectories and trajectory planning as a refinement of a path and path planning. In comparison, a trajectory additionally includes parameters like velocities, accelerations and/or jerks along a path. In this research, all paths/trajectories are always defined with respect to velocities, therefore these terms can be seen as interchangeable.
Planning is differentiated in offline and online planning. In the case of offline planning, a static path is calculated before the flight and thereby cannot be influenced anymore during it. Online planning methods are depending on conditions during the flight. They can react to changes in the environment, and/or other dynamic events.
A special case is CPP, which adds the requirement of coverage of an area or volume by a path. In an early work of Cao et al. (1988), where CPP is referred to as "region filling", from the original development of a robot lawn mower, generic conclusions are made. Doing so they have derived six criteria for CPP:

1. The robot must move through all the points in the target area covering it completely.
2. The robot must inherently have the ability to fill the region without overlapping paths.
3. Continuous and sequential operation without any repetition of path is required.
4. The robot must avoid all obstacles.
5. Simple motion trajectories (e.g., straight lines or circles) should be used for simplicity and control.
6. An "optimal" path is desired under available conditions.

Although those are inspired by a robot moving in two dimensions in a planar environment, Galceran and Carreras (2013a) stated that these are applicable in other coverage scenarios too. They remark that not all of them might be satisfied in complex scenarios.
A complex scenario, for example, is coverage of objects in a 3D-environment by a UAV. As Cheng et al. (2008) showed, coverage, in this case, refers to coverage of an attached sensor. A sensor might be, as in this research, a camera. For this reason, coverage of a surface is determined by being visible within its FOV.
Mentioned in section 1.2, in this research it will be investigated if multiple coverage of a surface can improve a from this camera input created 3D-model.
To do so the FOV has to cover the same areas, which have been investigated before, as a whole or partially again. This is clearly violating the second criterion of Cao et al. (1988) by following overlapping paths.

2.2 SLAM - Simultaneous Localization and Mapping

SLAM describes the problem of simultaneous localization and mapping. This is described as creating a map of the surrounding environment from sensor input and estimating the position of the exploring robot in it, e.g. an autonomous UAV.
The relations in this scenario are presented demonstrative by Stachniss et al. (2016) with a graphical model shown in figure 2.1. Following this graphical model in this part of the report

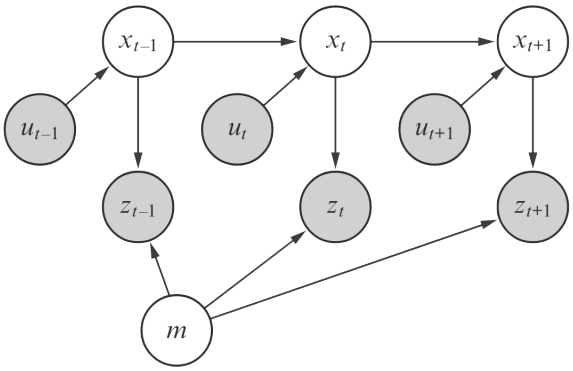


Figure 2.1: Model by Stachniss et al. (2016) demonstrating the SLAM-problem. Shaded nodes represent direct observations used to recover the others. The arrows show causal relationships.

their notation is used.
With m the map of the environment of a robot is denoted. Describing the locations of other objects in the surrounding. Like m, a robot's pose x_t is not directly observable. They are representing points in time by t.
A sequence of poses representing a path is given as:

$$X_T = \{x_0, x_1, x_2, ..., x_T\} \tag{2.1}$$

A terminal time of a sequence is given by T. According to Stachniss et al. (2016) estimation algorithms often use the initial pose x_0 as a reference.
What in general terms can be observed by a robot are sensor measurements z_t of features in m. Given as a sequence:

$$Z_T = \{z_0, z_1, z_2, ..., z_T\} \tag{2.2}$$

From z_t it is possible to relate between m and x_t.
The relation of a previous point in time x_{t-1} and the current point x_t is usually called odometry. Odometry u_t is thereby describing the motion of a robot. Given as a sequence:

$$U_T = \{u_0, u_1, u_2, ..., u_T\} \tag{2.3}$$

With these formalizations, the problem can be mathematically described in probabilistic terminology and further specified. Stachniss et al. (2016) categorized it into two main forms that can be found in the literature:

2.2.1 Full SLAM

Full SLAM is solving the problem for the full path X_T of the exploration. The path consists of all poses x_t during the exploration. Mathematically described as the joint posterior probability from discoverable Z_T and U_T over X_T and m.

$$p(X_T, m \mid Z_T, U_T) \tag{2.4}$$

Stachniss et al. (2016) mentioned that algorithms addressing this type are often processing all the collected discoverable data offline at once.

2.2.2 Online SLAM

According to Stachniss et al. (2016) in contrast to the previous full SLAM, online SLAM is estimating the current position x_t, not the whole path X_T.
Thereby, it can be denoted as follows:

$$p(x_t, m \mid Z_T, U_T) \tag{2.5}$$

Algorithms for this type can operate iteratively. Based on the previous, they are able to estimate the current pose x_t of the robot. For example, the in this research used ORB-SLAM2 represents such an online SLAM-algorithm and is described in section 2.3.

2.3 ORB-SLAM2

ORB-SLAM2 by Mur-Artal and Tardos (2017) is a derivative of their earlier work on ORB-SLAM (Mur-Artal et al. (2015)). It was extended to use beside monocular cameras also (synchronized) stereo and RGB-D cameras as input sensors. The estimation of the environment results in a 3D model represented as a point cloud. It is capable of estimating the current pose of the camera with respect to this map of points. Thereby it is also providing visual odometry.
Map points represent visual features in the environment. The map is estimated by following a keyframe-based approach, where not all input frames from the camera are used for solving the SLAM-problem. By only using specific keyframes for feature detection the computational efficiency is improved.
ORB-SALM2 is divided into three main components, tracking, local mapping and loop closing. Those are realized as threads. An overview of them can be found in figure 2.2 and their functionality is described in the following sections 2.3.1 to 2.3.3.

2.3.1 Tracking

In the tracking thread, ORB-SLAM2 is processing each input frame and searching for distinguishable features, which are used as map points. Therefore, it is using the ORB feature detection algorithm of Rublee et al. (2011).
By using a constant velocity model, the camera pose gets predicted from the previous frame, and a guided search is performed to search matching features between the frames. In case tracking was lost, the current frame gets compared to keyframes to match features and thereby relocalize. If in both cases enough features are found the current pose of the camera gets predicted.
With these initial features and camera pose the map is projected to search for map point correspondence of features. To reduce the computation load only points of related keyframes are taken into account and called “local map”. Related keyframes are keyframes that should have map points in common with the current frame and their neighbors. Neighboring keyframes are determined on their shared map points with each other. Based on the corresponding points, which are also in the frame visible, the camera pose gets optimized and updated.

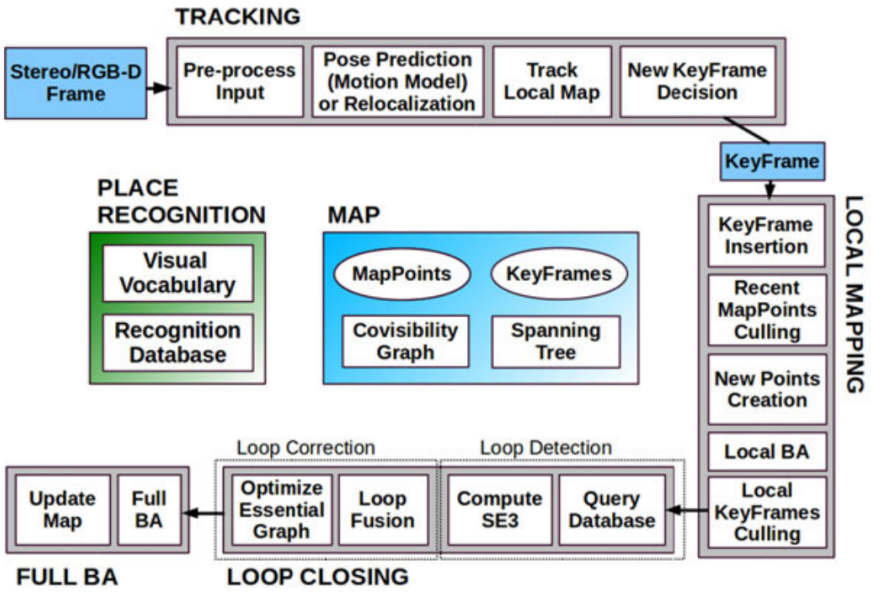


Figure 2.2: Structure of ORB-SLAM2 by Mur-Artal and Tardos (2017), showing the three main threads tracking, local mapping and loop closing. The last one can call the additional full BA thread.

Another core function of the tracking is to determine new keyframes. According to Mur-Artal et al. (2015) the following criteria have to be met for the current frame:

- Since the last relocalization, more than 20 frames must have passed.
- Since the last keyframe insertion at least 20 frames have passed, or the local mapping thread is idle.
- At least 50 points are tracked.
- At least 90% of the points of the keyframe with the highest map point correspondence are tracked.

One exception for all these exits. Using the stereoscopic version of the algorithm in this research, the initial frame is used as initial keyframe, as Mur-Artal and Tardos (2017) point out. At this point, the camera is also set to the origin and an initial map is created from detected features.

2.3.2 Local Mapping

For every new keyframe, local mapping is invoked. At insertion, it is getting related to the other keyframes depending on common map points.
Unmatched features are searched within the related keyframes. To become a map point it has to be initially observed in two keyframes.
On the long-term, map point culling is determining map points to be kept in the map. Map points get culled dependent on two conditions:

- After another keyframe was determined since the creation of a map point, the point has to be detected within three keyframes or more.
- The map point must be tracked in more than 25% of frames it should be visible

A local bundle adjustment (BA) is applied to optimize the current keyframes, all to it related key frames as well as the map points in them.

Not only map points can get culled by the local mapping, but also redundant keyframes. This can lead to a map point culling for those points that do not fulfill the previously mentioned conditions anymore.
Keyframes are removed if 90% or more of its map points are found in at least three other key frames with the same or higher accuracy.

2.3.3 Loop Closing

ORB-SLAM2 provides a loop closing feature, which targets to reduce the accumulated error during the observation. When reaching the same pose, this drift in the map should be possible to estimate. Hence, the last keyframe processed by the local mapping is compared with a database to determine if the scene was previously observed.
Between the detected loop keyframe and the current key frame a map point matching is applied to determine the error accumulated in the loop to construct a “similarity transformation”. In a first step, this transformation is applied to the current frame and its neighbors to fuse matched map points.
Finally, all the keyframes within the loop are getting transformed to remove the drift error. This corrects also the map points covered by those key frames.
In ORB-SLAM2 a full BA is invoked after closing a loop. As it is computationally very costly and to not avoid detecting new loops, it is running in a separate thread. Therefore, it represents a detached block in figure 2.2. Full BA distinguishes from the previously mentioned local BA that it is optimizing all key frames and all map points. When it is finished the map has to be updated with the output of the full BA. Meanwhile, newly introduced keyframes and their map points are transformed by propagating the correction, which was applied to the optimized ones.

3 Evaluation of ORB-SLAM2's performance within the simulation

In this chapter ORB-SLAM2's performance is evaluated to determine its suitability to be used in the simulation to scan an object and create a point cloud from it. Therefore spatial relations of the components within the simulation are described first in section 3.1. This should provide the necessary overview to relate the flight trajectory presented in section 3.2. The trajectory is followed in both experiments of this chapter.
The first experiment, which can be found in section 3.3, is evaluating the best texture out of a set. The evaluation is dependent on the number of map points ORB-SLAM2 is mapping and if it is not losing track. The experiment in section 3.4 is measuring the error between the "ground truth" and estimated flight trajectory.

3.1 Spatial Configuration in Simulation

In figure 3.1 one can see the object of interest, represented by a cylinder with a radius of 1 m and 3 m height in front of the UAV as well as the world frame Ψ_W. Ψ_W axes are represented by colored lines (x red, y green, z blue). The grid on the $x\,y$-plane consists of squares with 1 m side length.

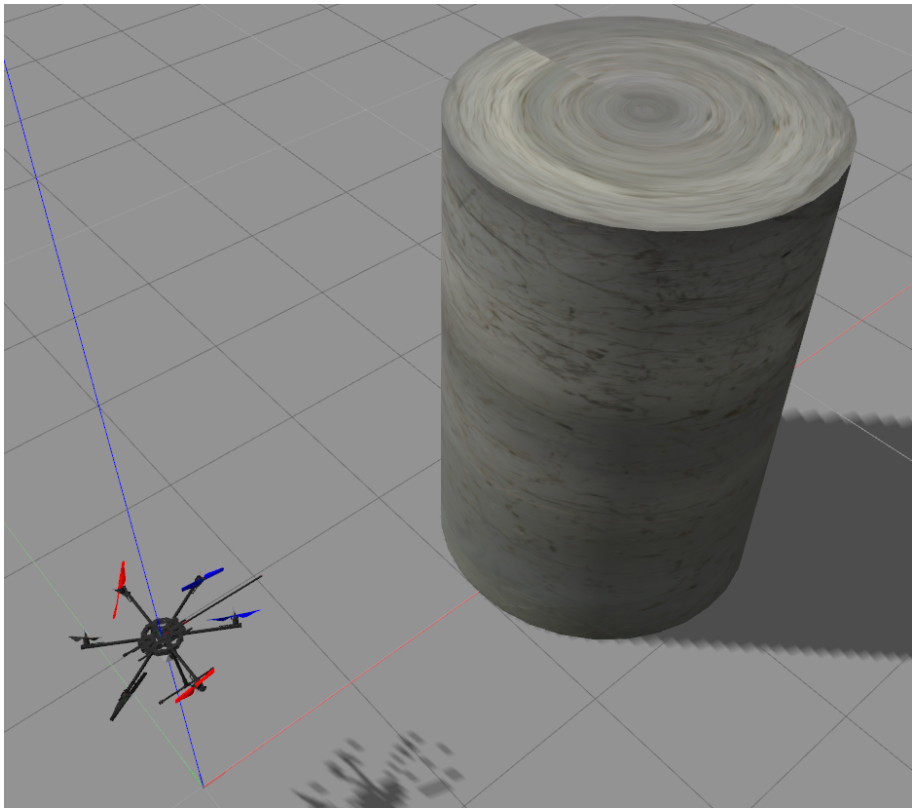


Figure 3.1: UAV at initial position $\boldsymbol{P}^W_{U_{init}}$ in front of the textured object.

The position and orientation of the cylinder in Ψ_W are:

$$\boldsymbol{P}^W_O = \begin{bmatrix} 3 \\ 0 \\ 1.5 \end{bmatrix} \qquad \boldsymbol{R}^W_O = \boldsymbol{I} \tag{3.1}$$

The initial position and orientation of the UAV in the world frame Ψ_W are:

$$\boldsymbol{P}^W_{U_{init}} = \begin{bmatrix} 0 \\ 0 \\ 1.5 \end{bmatrix} \qquad \boldsymbol{R}^W_{U_{init}} = \boldsymbol{I} \tag{3.2}$$

The distance to the ground is necessary as otherwise ORB-SLAM2 initializes only or mainly with the drone shadow within its stereo camera's FOV. As described in section 2.3.1, the first frame is used to create an initial map when ORB-SLAM2 is started. For this initial map the detected features, which would become map points, would be the edges of the drone shadow on the ground. This would result in detected map points' positions in Ψ_W dependent on P^W_U. When P^W_U changes those are not present anymore and it loses track. This results in not accumulating new features/map points as it only can localize itself in the starting position.
Shadows, in general, are due to the directed light source. Instead of uniformly lightning the whole simulation environment a single global directed light source is used to provide a visually more challenging environment.
The two image sensor's simplified visual representations of the simulated stereo camera are the white cubes below the UAV in figure 1.2.
The position and orientation of the camera, as a single object consisting of both sensors, in the UAV base frame Ψ_U are:

$$\boldsymbol{P}^U_{phC} = \begin{bmatrix} 0 \\ 0 \\ -0.1 \end{bmatrix} \qquad \boldsymbol{R}^U_{phC} = \begin{bmatrix} 0.9082 & 0 & 0.4186 \\ 0 & 1 & 0 \\ -0.4186 & 0 & 0.9082 \end{bmatrix} \tag{3.3}$$

The position below the center of the UAV in combination with a pitch $\theta = 24.75°$ has multiple reasons.
It avoids the tilted propellers of being in the FOV, which would interfere the image processing heavily. Therefore, $\theta = \frac{\text{VFOV}}{2}$ is ensuring the upper boarder of the vertical field of view (VFOV) being parallel to the x-axis of Ψ_U. Placing the camera below the UAV allows parts of the object near to ground being visible while flying close to the object. The distance to the base frame provides reasonable space for a mounted stereo camera for future works which do not base on simulation. Figure 3.2 provides an overview of this spatial configuration.

3.2 Flight Trajectory

As mentioned in section 1.3.2, in previous researches of Cheng et al. (2008) and Mansouri et al. (2018), to completely cover similar shapes, rotating around them and increasing step-wise height was chosen as CPP method. To provide a more steady and continuous motion, but following the same principle, for this research, a helical trajectory is chosen. This should lead to better results from the rigidly connected camera's output as input to ORB-SLAM2. From its initial position $\boldsymbol{P}^W_U = \boldsymbol{P}^W_{U_{init}}$, the UAV follows the clockwise helix flight trajectory facing towards the cylinder as in equation (3.4):

$$\begin{aligned} x(t) &= -r \cdot cos(\omega t) + x_0 \\ y(t) &= r \cdot sin(\omega t) + y_0 \\ z(t) &= v_z \cdot t + z_0 \\ \psi(t) &= -\omega t + \psi_0 \end{aligned} \tag{3.4}$$

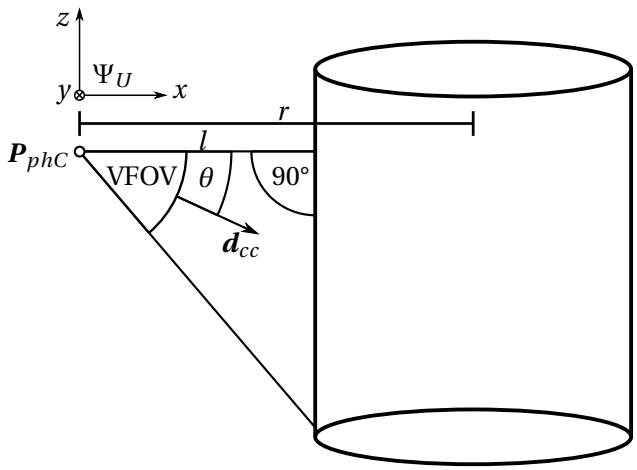


Figure 3.2: The position of the camera $\boldsymbol{P}^{U}_{phC}$ with respect to the UAV base frame Ψ_U, showing the resulting VFOV in relation to the object of interest. How the pitch with an angle of θ is changing the direction of the camera's center $\boldsymbol{d}_{cc}$ is depicted. The radius r related to the object's center and the distance l of the camera/UAV to the object's surface can also be found.

with $r = 3\,\text{m}$ $\omega = 0.25\,\text{rad/s}$ and $v_z = 0.025\,\text{m/s}$.
This trajectory leads to an overlap of undiscovered and discovered areas of the object, to provide the possibility to localize only with the single object in the simulation. Furthermore, it enables to detect a loop when flying the path and perform loop closing.

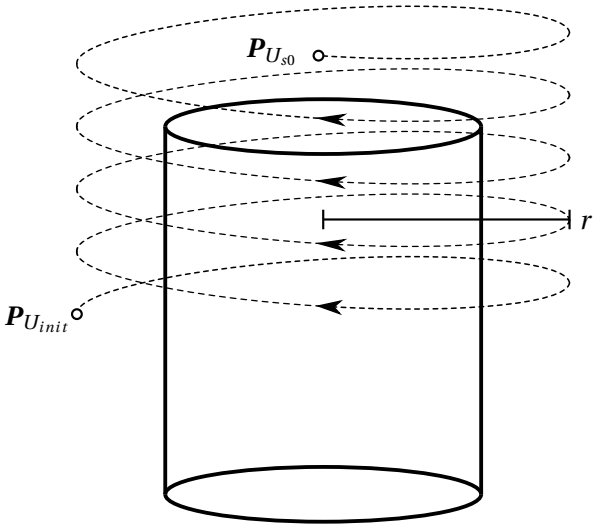


Figure 3.3: Helical flight path of the UAV as a dashed line around the object starting at $\boldsymbol{P}_{U_{init}}$ to reach $\boldsymbol{P}_{U_{s0}}$

The velocities are chosen with respect to the limits of the simulation hardware.
Starting at $\boldsymbol{P}^{W}_{U_{init}}$ the UAV flies around the cylinder following the trajectory, which is illustrated as a dashed line in figure 3.3. Reaching $\boldsymbol{P}^{W}_{U_{s0}}$ where $z(t) = 4.5\,\text{m}$ the UAV as well as the mapping

of ORB-SLAM2 gets stopped. The limit in z-direction is set to 4.5 m as at higher points, the distance between camera and object gets so large that no new features can be detected.

3.3 Different Texture Evaluation on Object of Interest

As ORB-SLAM2 is based on feature matching different object surface characteristics should make a noticeable difference in the resulting point cloud output.
Therefore the object of interest, which is placed in the simulation environment to be investigated by the UAV, is scanned with different textures mapped on it. The textures can be found in figure B.1 in appendix B. The trajectory followed to scan it can be found in section 3.2.
Comparing the resulting arithmetic mean of detected points against each other, it should be possible to determine its suitability for further usage. The suitability is defined based on the higher number of detected map points by ORB-SLAM2. A description of how it is mapping points can be found in section 2.3.2. Furthermore, if ORB-SLAM2 is losing track while the UAV is scanning the object, is a reason for unsuitability. When ORB-SLAM2 is losing track it is not adding new features as map points until it has relocalized the UAV's, more precisely the camera's pose.

3.3.1 Results and Discussion

In table 3.1 the results when flying the trajectory from equation (3.4) with $r = 3\,\text{m}$, $\omega = 0.25\,\text{rad/s}$ $v_z = 0.025\,\text{m/s}$ can be found. "Red grey bricks" provides the highest number of determined

texture	single flights				mean
worn aluminium	6533	6240	6987	6373	6533.25
old scratched metal	6556	7233	7774	7437	7250
concrete	5350	5181	5006	5114	5162.75
clean concrete	6025	6388	5673	5999	6021.25
red grey bricks	a	9608	9566	9645	9606.33
factory rock wall	b	b	a	b	b

Table 3.1: Number of points captured by ORB-SLAM2 at multiple flights of the UAV as well as the resulting mean, for each evaluated texture. "a" and "b" are denoting flights during which tracking was lost.

features/map points but the "a" in its first run denotes that in some cases ORB-SLAM2 is losing track due to wrong loop closing and afterward cannot relocalize. This is caused as it seems to be too similar in different areas due to its repetitive brick pattern.
In the case of "factory rock wall" there are wrong loop detections at every test. Once like above but "b" flights show that it is relocalizing at a wrong position and continuing mapping. This remaps the exiting points to wrong positions and therefore leads to a wrong pose of the existing map within Ψ_W. Also, new points' positions are determined wrong and thereby reconstructing a wrong and/or multiple representations of the cylinder's shape.
From the remaining textures[1], which do not lead to the above-described errors, "old scratched metal", which can be found in figure B.1d, shows the highest number of points detected by ORB-SLAM2. This very likely relates to its inhomogeneous placed high number of scratches leading to more distinguishable features.
Due to this, it was used as a texture for the object in the whole project.
Between each flight, ORB-SLAM2 has to be restarted. When using its provided reset function after a full test trajectory it always loses the tracking at the first rotation around the object when facing the shady side of the object. In shady areas, the algorithm, in general, might

[1] "clean concrete" was only partially taken as otherwise it provides a 4 times higher resolution compared to the others

have difficulties to not lose track, as also Gaspar et al. (2018) detected it losing track during a transition to a darker area.
Such replicable behavior with textures that otherwise does not lead to this fault implies that ORB-SLAM2 at some point is not completely freeing resources.
When resetting it in the starting position, without flying a test trajectory before, this cannot be discovered.

3.4 Error Between "Ground Truth" and Estimated Flight Trajectory

In this section, the error between the by ORB-SLAM2 estimated camera position compared to the as "ground truth" specified position provided by the simulation is investigated. It should provide insight into the position error of its tracking as well as the influence of its loop closing feature. Similar evaluations have been done by Giubilato et al. (2018) and Gaspar et al. (2018). Details of its tracking method can be found in section 2.3.1. Loop closing is described in section 2.3.3.
Previously in section 3.3 the texture "old scratched metal", which can be found in figure B.1d, provided the highest arithmetic mean of mapped points without losing track. Due to this, it was also used in this experiment.
Flying the trajectory described in section 3.2 with this texture on the object of interest should reveal the error between the from ORB-SLAM2 estimated position of the camera center in Ψ_W compared to the "ground truth". With respect to the $\boldsymbol{P}^W_{estC}(t)$ being the position of the camera estimated by ORB-SLAM2's tracking and $\boldsymbol{P}^W_{phC}(t)$ the "physical"/ground truth position within the simulation, $|\Delta x|+|\Delta y|$ is the accumulated absolute error in x and y-direction between them, $|\Delta z|$ describes the absolute difference on the z-axis.
Implementation details can be found in appendix A.1.

3.4.1 Results and Discussion

The error between $\boldsymbol{P}^W_{estC}(t)$ and $\boldsymbol{P}^W_{phC}(t)$ evolving over time is shown in figure 3.4.
One can see after approximately 10s that the error is increasing as the UAV starts flying the trajectory.
Especially on the xy-plane the error increases with the distance of the UAV to the starting position resulting in the relatively high peaks in the graph. Such behavior has also been described by Giubilato et al. (2018).
Close to the initial position the error gets lower. This should be caused due to ORB-SLAM2 estimating the position of new map points in reference to already detected ones.
Also, the later peaks, representing each a rotation, show a more uniform shape. This might be caused to the loop closing feature of the algorithm which is activated at the end of the first rotation. It corrects the accumulated error over the detected loop in the flight path. The accumulated error gets distributed over the in the loop detected points. These rearranged points should act as a more accurate reference at the later flights for new points. The observed error of the later rotations against this expectation is not decreasing just the previously mentioned more steady change can be seen in figure 3.4. Similar results have been observed for the z direction although the significantly lower absolute error.
As the magnitude of the $|\Delta x| + |\Delta y|$ error is similar to that of the real-world experiment of Giubilato et al. (2018) for translational forward motion, ORB-SLAM2 seems to be suitable for the simulation.

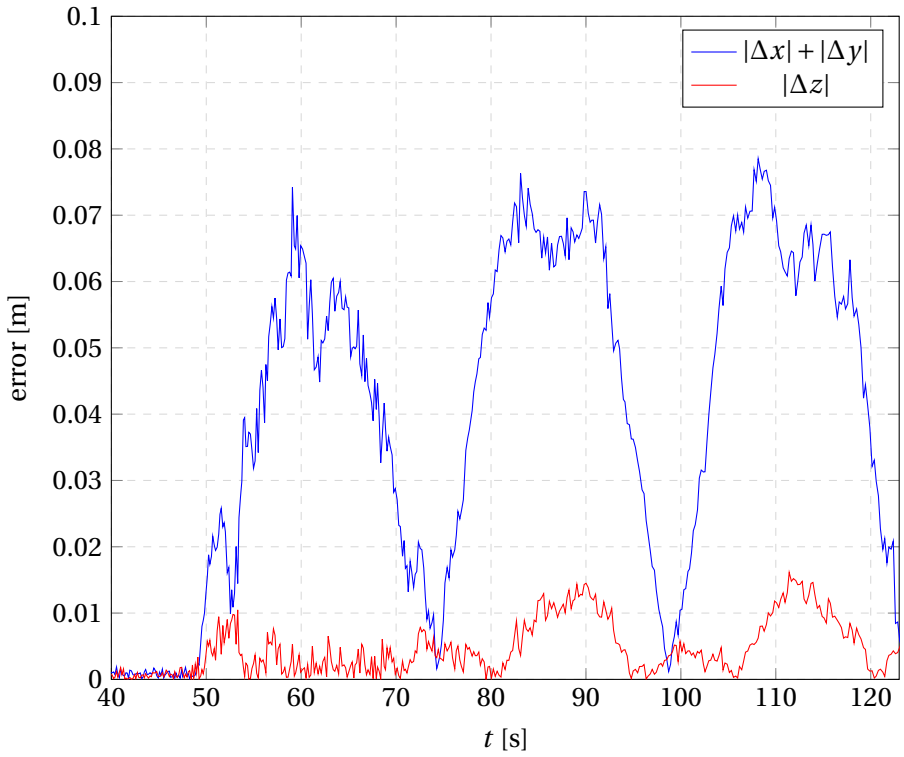


Figure 3.4: Plots of the error $|\Delta x| + |\Delta y|$ on the $x\,y$-plane and the error $|\Delta z|$ in z-direction between the by ORB-SLAM2 estimated camera position $\boldsymbol{P}^{W}_{estC}$ and the "ground truth" camera position $\boldsymbol{P}^{W}_{phC}$

4 Trajectory Planning Methods for 3D Reconstruction

In chapter 3 ORB-SLAM2 itself was investigated on its performance and resulting suitability to be used within the simulation. Building on top of it, in this chapter different methods, to calculate flight trajectories for scanning the object of interest and 3D reconstructing it as a point cloud, are tested. The parameter to evaluate the reconstruction is denoted as the entropy n. This entropy is defined as the number of map points detected by ORB-SLAM2.
In section 4.1 the common design of the methods is shown. It describes how the different methods are based on a common coordinate system as well as how to change the position of the UAV, following circular motions, in it. Furthermore, how the CPP is divided into steps.
In sections 4.2 to 4.4 the different CPP methods are described by elaborating what trajectories are followed in each step and why. There are also the results of each method provided and discussed for flights with different velocities and distances to the object. Parts of the results were also presented by Sirmaçek et al. (2019).

4.1 Common Design of Methods

In this section, the common design of the methods described and investigated in sections 4.2 to 4.4 is provided to avoid redundancy.

4.1.1 Path Steps

All methods are divided into steps describing the trajectories of the method's whole path. These steps are limited by their execution time. Therefore, the duration of a step is denoted by Δt_s.
Moreover, some steps are determining how to proceed with the trajectory planning. This form of online trajectory planning is dependent on the, by ORB-SLAM2, determined map points at parts of the object. These four parts are called segments and in relation to an introduced cylindrical coordinate system described in section 4.1.3.

4.1.2 Spatial Configuration

The spatial configuration of the object, as well as the camera position relative to the UAV, remains the same as described in section 3.1. The position of the UAV $\boldsymbol{P}_U$ is described by using a frame with a cylindrical coordinate system, which is presented in the following section.

4.1.3 Cylindrical Coordinate System

As on the $x\,y$-plane, all methods in sections 4.2 to 4.4 follow circular motions around the object on their trajectories. Therefore, the frame Ψ_P using a polar coordinate system on the $x\,y$-plane in the world frame Ψ_W was defined.
It is describing ϕ as the angle and by the radius r the distance of the origin at which a reachable flight position $\boldsymbol{P}_U^P$ of the UAV can be found.
The coordinate system's origin is defined in the object's center $x_{OC} = 3\,\text{m}$, $y_{OC} = 0\,\text{m}$ leading to the following translations to the in Cartesian coordinates defined $x\,y$-plane of Ψ_W for the position $\boldsymbol{P}_U^W$ of the UAV setpoint:

$$\begin{aligned} x(t) &= r \cdot cos(\phi(t)) + x_{OC} \\ y(t) &= r \cdot sin(\phi(t)) + y_{OC} \end{aligned} \tag{4.1}$$

The z-axis to evolve it to a 3D cylindrical coordinate system is equivalent in both frames. Velocities are described by the angular velocity ω. Radius r is for the whole path of a method and any position constant.

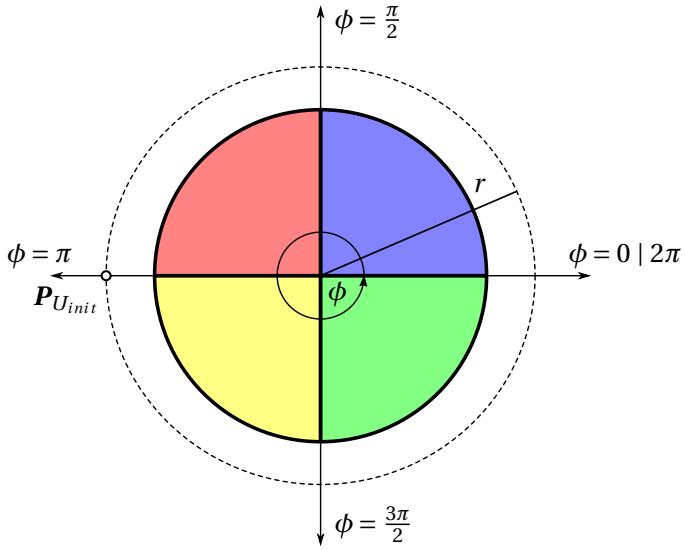


Figure 4.1: Top view of the cylinder with its colored segments in the polar coordinate system of Ψ_P. The arrow at the angle ϕ shows the positive rotation direction ($d = 1$)

The cylindrical coordinate system's underlying polar coordinate system including a top view of the cylinder with the previously described segments can be found in figure 4.1. Segments are the colored quarters within the inner circle which is representing the cylinder.
Segment 0 (S_0) in red, segment 1 (S_1) in blue, segment 2 (S_2) in green and segment 3 (S_3) in yellow. Simulated sunlight is directed towards the center of S_0, S_1 and S_3 are in the transition and S_2 is in the shadow. The entropy n is also investigated for those segments. Hence, n_{total} denotes the number of points for the whole object and n_{sega} for $a = \{0,1,2,3\}$ the number of points for each segment.

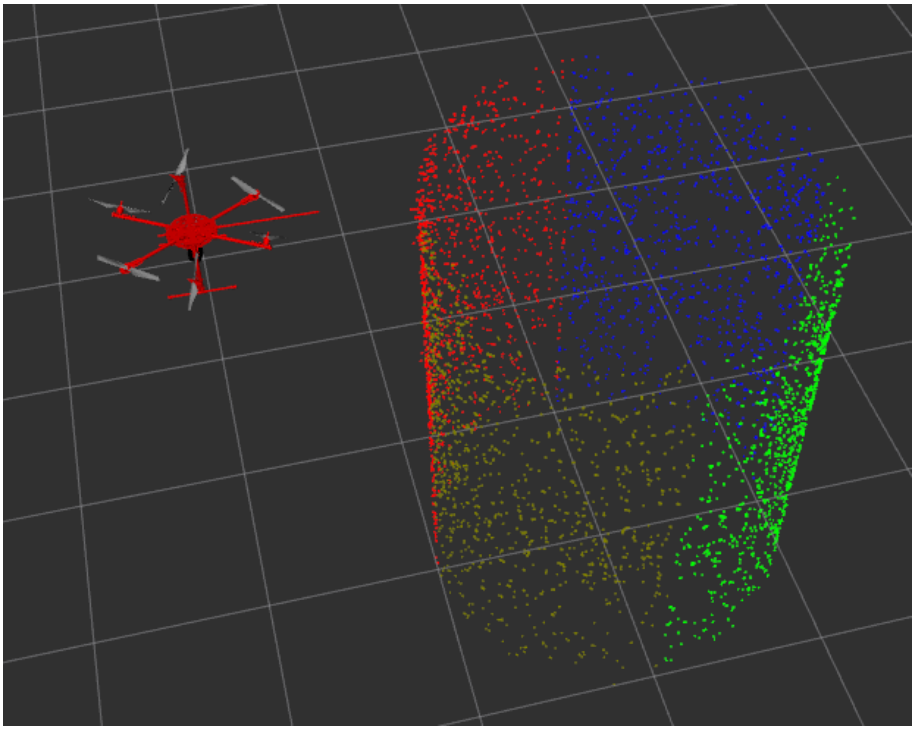


Figure 4.2: UAV next to the point cloud gained from a flight using the helix CPP method, described in section 4.2

A dashed circle with distance r to the origin represents possible flight positions. $\boldsymbol{P}_{U_{init}}$ can be found on it. It is the initial position of every method's path in these two dimensions. Relative to frame Ψ_P it is:

$$\boldsymbol{P}^P_{U_{init}} = \begin{bmatrix} r \\ \phi_{U_{init}} = \pi \\ z_{U_{init}} \end{bmatrix} \tag{4.2}$$

$z_{U_{init}}$ is defining the height and $\phi_{U_{init}}$ the angular position at the start of the first step's trajectory. The latter is for every method π rad.
The pose of the UAV, as well as the points per segment, have been visualized during the experiments. An example screenshot of the UAV next to the point cloud after a flight using the helix method, described in section 4.2, can be found in figure 4.2. The colors of the points are identical to the colors of the segments, in which they are located, and have been described above for figure 4.1.

4.1.4 Circular Motion

All methods have in common that they only or also consist of steps with a circular motion. In these cases, the trajectory duration Δt_s is defined as in equation (4.3). $\Delta\phi$ describes the angular distance which has to be flight and ω the desired angular velocity.

$$\Delta t_s = \frac{\Delta\phi}{\omega} \tag{4.3}$$

Furthermore, the flight direction has to be taken into account. It is denoted by d. In the case of $d = 1$ it is called a positive direction, which is shown by the arrow at the angle ϕ in figure 4.1. For the opposite negative direction $d = 0$.
With these parameters, the current angular position $\phi(t)$ for a step including a circular flight is calculated as in the pseudocode of algorithm 4.1. With ϕ_0 the angular position at the start of

Algorithm 4.1: Calculation of current angle $\phi(t)$ of the trajectory for a step of Δt_s duration

```
while t < Δt_s do                                   1
    if d=1 then                                     2
        φ(t) ← (φ_0 + ωt) mod 2π                    3
    else                                            4
        φ(t) ← φ_0 − (ωt mod 2π)                    5
        if φ(t) < 0 then                            6
            φ(t) ← φ(t) + 2π                        7
        end                                         8
    end                                             9
end                                                 10
```

the trajectory $\phi(t = 0)$ is described.
The binary operation providing the remainder of a Euclidean division is denoted as " mod ". It is used with 2π as a second operand to scale $\phi(t)$ to a corresponding value in the interval $[0, 2\pi]$. Also adding 2π in line 7 is for this purpose.

4.2 Method 1: Helix

This method's path is similar to the in section 3.2 presented helix flight. In contrast, just the lateral surface is investigated to simplify the comparison with the other methods.

As described in section 3.2, in previous researches of Cheng et al. (2008) and Mansouri et al. (2018), coverage paths have been considered efficient, which are rotating around the object and stepwise increase height. To follow this principle but provide a constant motion, a helical trajectory is chosen as this first method.
It is used to determine a maximum of detectable features, more specifically map points, by increasing the number of rotations during one flight. Increasing the number of rotations, while not changing the total height to fly, leads to covering the same area more often. Therefore, this method should show if multiple coverage of the same area can improve the 3D-model of the object of interest.

4.2.1 Path Steps

The Helix method only consists of one step, which is flying the helix starting at a previous reached initial position $\boldsymbol{P}^P_{U_{init}}$ to $\boldsymbol{P}^P_{U_{s0}}$. During this flight the UAV has to climb a height difference $h_{diff} = 1.5\,\text{m}$.
This path can be seen as a dashed line around the cylinder in figure 4.3

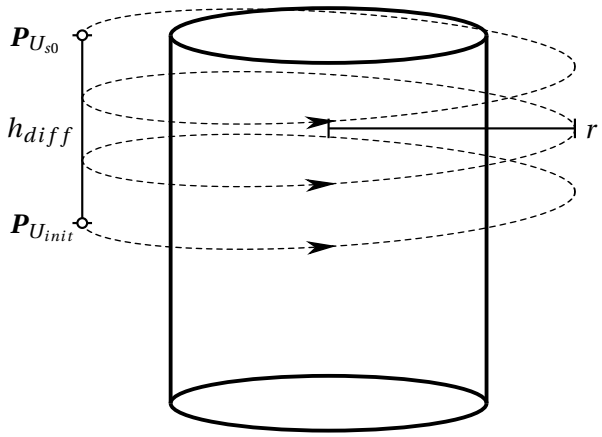


Figure 4.3: Flight path of the UAV as dashed line around object starting at $\boldsymbol{P}_{U_{init}}$ to $\boldsymbol{P}_{U_{s0}}$

The circular motion of the trajectory follows the principle described in section 4.1.4. To gain $\Delta\phi$ for equation (4.3) it is computed as in equation (4.4). The number of desired rotations n_{rot} is the limiting factor of the angular distance.

$$\Delta\phi = 2\pi \cdot n_{rot} \tag{4.4}$$

Besides the circular motion, there is also the velocity v_z along the z-axis. To let the UAV not exceed the height distance h_{diff} between $\boldsymbol{P}^P_{U_{init}}$ and $\boldsymbol{P}^P_{U_{s0}}$ it is also related to Δt_s from equation (4.3):

$$v_z = \frac{h_{diff}}{\Delta t_s} \tag{4.5}$$

Leading to $z(t)$ of the setpoint of the UAV $\boldsymbol{P}^P_U$:

$$z(t) = z_{U_{init}} + v_z t \tag{4.6}$$

4.2.2 Results

In this section, the results of multiple flights with a number of rotations n_{rot} from 2 to 100 for a different radius r can be found. All of them with an angular velocity $\omega = 0.25\,\text{rad/s}$ starting in positive direction $d = 1$ at $\boldsymbol{P}^P_{U_{init}}$:

$$\boldsymbol{P}^P_{U_{init}} = \begin{bmatrix} r \\ \pi \\ z_{U_{init}} = 1.5 \end{bmatrix} \tag{4.7}$$

Covering a flight distance of $h_{diff} = 1.5\,\text{m}$. Therefore, $\boldsymbol{P}^P_{U_{S0}}$ is:

$$\boldsymbol{P}^P_{U_{S0}} = \begin{bmatrix} r \\ \pi \\ z_{U_{init}} + h_{diff} \end{bmatrix} = \begin{bmatrix} r \\ \pi \\ 3 \end{bmatrix} \tag{4.8}$$

For $r = 2\,\text{m}$ in table 4.1, $r = 2.5\,\text{m}$ in table 4.2 and for $r = 3\,\text{m}$ table 4.3. A comparing plot of the results can be found in figure 4.4.

$\Delta t_s\,[s]$	n_{rot}	n_{total}	n_{seg0}	n_{seg1}	n_{seg2}	n_{seg3}
50.27	2	3823	1462	691	716	954
125.66	5	5047	1737	967	1016	1327
251.33	10	6982	2142	1540	1622	1678
376.99	15	7573	2291	1623	1708	1951
502.65	20	7944	2252	1973	1786	1933
628.32	25	8802	2354	2250	2058	2140
1256.64	50	8637	2394	2117	2022	2104
2513.27	100	8846	2472	2202	2053	2119

Table 4.1: Number of points captured when flying a helix with a radius of 2 m (1 m distance to the object surface) with $\omega = 0.25\,\text{rad/s}$

$\Delta t_s\,[s]$	n_{rot}	n_{total}	n_{seg0}	n_{seg1}	n_{seg2}	n_{seg3}
50.27	2	3569	1261	724	710	874
125.66	5	3670	1258	781	716	915
251.33	10	4575	1335	1140	1027	1073
376.99	15	5200	1523	1290	1128	1259
502.65	20	5145	1438	1424	1086	1197
628.32	25	5237	1537	1283	1218	1199
1256.64	50	5375	1238	1526	1297	1314
2513.27	100	5182	1234	1117	1394	1437

Table 4.2: Number of points captured when flying a helix with a radius of 2.5 m (1.5 m distance to the object surface) with $\omega = 0.25$ rad/s

$\Delta t_s\,[s]$	n_{rot}	n_{total}	n_{seg0}	n_{seg1}	n_{seg2}	n_{seg3}
50.27	2	7713	1924	2200	2144	1445
125.66	5	5200	1340	1198	1324	1338
251.33	10	4565	1385	1071	1001	1108
376.99	15	4477	1285	1071	961	1160
502.65	20	4255	1201	955	987	1112
628.32	25	3993	1105	990	894	1004
1256.64	50	3856	1067	960	895	934
2513.27	100	3931	1134	898	902	997

Table 4.3: Number of points captured when flying a helix with a radius of 3 m (2 m distance to the object surface) with $\omega = 0.25$ rad/s

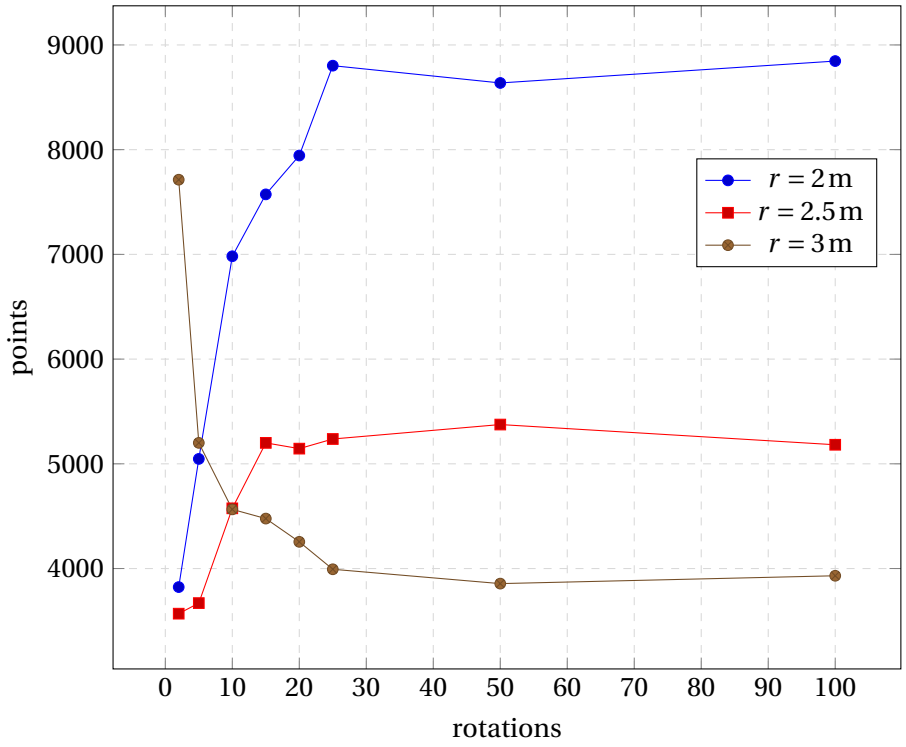


Figure 4.4: Plots showing the number of points captured when flying a helix with different radii r and number of rotations at $\omega = 0.25\,\text{rad/s}$

4.2.3 Discussion

In figure 4.4 and tables 4.1 to 4.3 a saturation of accumulated points n_{total} for the flights with 25 rotations and more can be discovered for all radii. This represents flight durations Δt_s at $\omega = 0.25\,\text{rad/s}$ which are equal or longer than $502.65s$ ($8min\ 22.65s$).
Furthermore, for those flights, it appears that increasing the distance to the object leads to less detected points. This might reason in the lower resolution of detectable features when the distance to the object gets increased. An interesting outcome is that, in contrast to a radius $r = 2.5\,\text{m}$ or $r = 2\,\text{m}$, in the case of $r = 3\,\text{m}$ the entropy n_{total} is decreasing while increasing rotations. This might be caused due to the higher distance to the object, which could lead to more wrong detected map points. With more rotations, and thereby more often covering the same area, those possibly wrong detected points might get removed from the map. As a consequence increasing the quantity of rotations seems to lead to better error correction by ORB-SLAM2.
In other cases, $r = 2.5\,\text{m}$ or $r = 2\,\text{m}$, the increase of rotations leads also to a higher number of in total detected features. This could indicate that because of a higher number of rotations ORB-SLAM2 is able to detect more features as the surface gets more often covered.
The entropies of the segments n_{sega} behave the same as the total number of map points n_{total}. In the case of $r = 2\,\text{m}$ the, saturation seems to be reached already for $n_{rot} \geq 15$, in difference to the other two radii. Also for those flights, the segments' entropies are varying more than in the case of $r = 2\,\text{m}$ and $r = 3\,\text{m}$.

4.3 Method 2: Circular Scan with Static Height Autonomous Segment Rescan

Five trajectories with only circular motion are forming this method. An initial circle followed by four segment rescans. This is to investigate if rescanning from the same position leads to a noticeable improvement in the number of map points for the segments and/or the whole object.
Segments to rescan are determined dependent on the map point-entropy n.

4.3.1 Path Steps

Step 0: Initial Circle

In section 2.3.3 the loop closing feature of ORB-SLAM2 was described as being intended to reduce the error in the estimated map, according to Mur-Artal and Tardos (2017). Gaspar et al. (2018) referred to loop closing as a "key feature". Mentioned previously in section 1.3.1, Giubilato et al. (2018) pointed out that they identified ORB-SLAM2's loop closure as the reason for being able to reconstruct the whole path in their test.
Therefore, to ensure a possible loop closure at an early stage as well as being able to evaluate on the map point-entropy, in the next step a circle around the object is flight as an initial trajectory. The starting position $\boldsymbol{P}^P_{U_{init}}$ is:

$$\boldsymbol{P}^P_{U_{init}} = \begin{bmatrix} r \\ \pi \\ z_{U_{init}} = 3 \end{bmatrix} \tag{4.9}$$

From $\boldsymbol{P}^P_{U_{init}}$ flying a full circle at constant height $z_{U_{init}} = 3\,\text{m}$ where $\Delta\phi = 2\pi$ for equation (4.3), results in the position at the end of this step $\boldsymbol{P}^P_{U_{s0}} = \boldsymbol{P}^P_{U_{init}}$. A visualization of the trajectory can be found in figure 4.5.

Step 1: Lowest Global Entropy

After flying the initial circle a first map was built by ORB-SLAM2. From this map, the number of map points within a segment's boundaries is known for each segment.

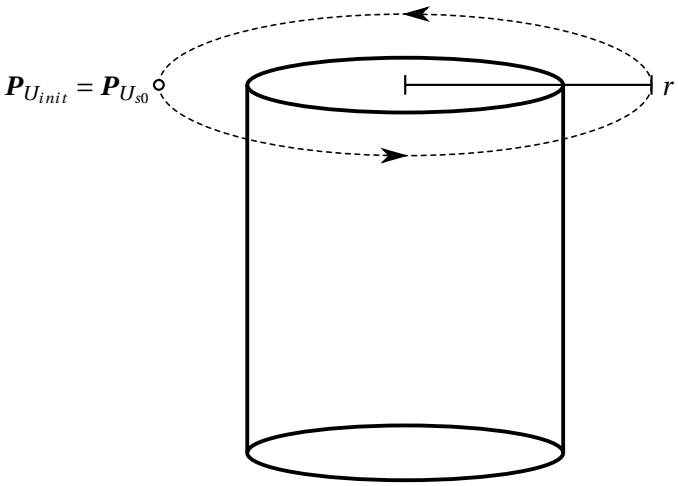


Figure 4.5: Method 1's step 0 circle trajectory as a dashed line from the position $\boldsymbol{P}^P_{U_{init}}$ to $\boldsymbol{P}^P_{U_{s0}} = \boldsymbol{P}^P_{U_{init}}$

The goal of this step is to reach a position in front of the segment with the lowest number of map points. The possible destination positions $\boldsymbol{P}^P_{dst}$ in front of the segments are $\boldsymbol{P}^P_{S_0}$, $\boldsymbol{P}^P_{S_1}$, $\boldsymbol{P}^P_{S_2}$ and $\boldsymbol{P}^P_{S_3}$ shown in figure 4.6. With $\phi_{S_0} = \frac{3\pi}{4}$ rad, $\phi_{S_1} = \frac{\pi}{4}$ rad, $\phi_{S_2} = \frac{7\pi}{4}$ rad and $\phi_{S_3} = \frac{5\pi}{4}$ rad.
To reach these positions from the starting position $\boldsymbol{P}^P_{st} = \boldsymbol{P}_{Uinit}{}^P$ the direction d as well as the angular distance $\Delta\phi$ have to be provided for a circular flight as described in section 4.1.4.
For this step the directions are defined as follows:

$$d = \begin{cases} 1 & \text{for } \boldsymbol{P}^P_{S2} \vee \boldsymbol{P}^P_{S3} \\ 0 & \text{for } \boldsymbol{P}^P_{S0} \vee \boldsymbol{P}^P_{S1} \end{cases} \tag{4.10}$$

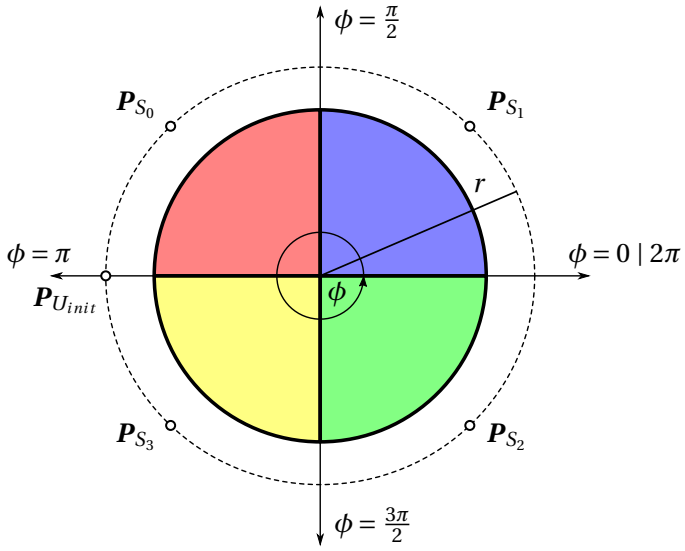


Figure 4.6: Top view of the cylinder with its colored segments and possible rescan positions $\boldsymbol{P}_{S_0}$, $\boldsymbol{P}_{S_1}$, $\boldsymbol{P}_{S_2}$ and $\boldsymbol{P}_{S_3}$ in front of their centers

In equation (4.11) the different cases for the calculation of the angular distance $\Delta\phi$ can be found. Those are dependent on the relation between the angle ϕ_{st} of $\boldsymbol{P}^P_{st}$ and ϕ_{dst} of $\boldsymbol{P}^P_{dst}$. Although only two of the cases are possible in this step, in the case of later steps, all are taken into account and for completeness also added here.

$$\Delta\phi = \begin{cases} \phi_{dst} - \phi_{st} & \text{for } \phi_{st} < \phi_{dst},\ d = 1 \\ 2\pi + \phi_{dst} - \phi_{st} & \text{for } \phi_{st} > \phi_{dst},\ d = 1 \\ \phi_{st} - \phi_{dst} & \text{for } \phi_{st} > \phi_{dst},\ d = 0 \\ 2\pi + \phi_{st} - \phi_{dst} & \text{for } \phi_{st} < \phi_{dst},\ d = 0 \end{cases} \tag{4.11}$$

Following the example being shown in figure 4.7, the segment after the initial circle step 0 with the lowest entropy is segment 3. Because of this, the destination position to fly for this step $\boldsymbol{P}^P_{dst} = \boldsymbol{P}^P_{U_{s1}} = \boldsymbol{P}^P_{S3}$.

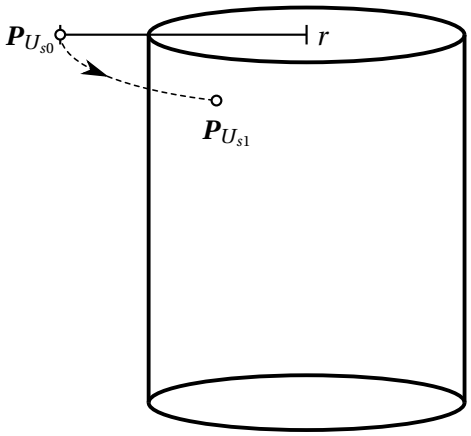


Figure 4.7: Method 1's step 1 example trajectory from the position $\boldsymbol{P}^P_{U_{s0}}$, which is reached after step 0, to segment 3 with the lowest entropy at $\boldsymbol{P}^P_{U_{s1}} = \boldsymbol{P}^P_{S3}$

Step 2 to 4: Lowest Neighbor Entropy and Keep Direction

After reaching the segment with the lowest entropy in the previous step, in step 2 the direction towards the next and following segments to scan is determined.
Independent of the direction d the previous step 1 was reached, in both directions the two neighbor segments' numbers of points are compared. The direction to the segment with the lower entropy is defined as the direction for the remaining steps. In the case of an equal entropy it is flying in negative direction $d = 0$.
By only inspecting the neighbor segments in step 2 and using the same direction for the steps 3 and 4 to visit the remaining two segments after another, it is possible to keep the distance to fly at a minimum.
Due to the constant angular velocity ω and fixed radius r during the whole path, a lower distance results in a proportional lower time to visit all remaining segments.
Continuing from the previous example and the start position being the position after step 1, $\boldsymbol{P}^P_{st} = \boldsymbol{P}_{s1}$, the neighbor segment with the lowest entropy to segment 3 is segment 0. This leads

to $\boldsymbol{P}^P_{dst} = \boldsymbol{P}^P_{U_{s2}} = \boldsymbol{P}^P_{S0}$ and the UAV flying to segment 0 in the negative direction during step 2. This can be seen as in figure 4.8 as well as following the same direction for step 3 and 4 such that $\boldsymbol{P}^P_{U_{s3}} = \boldsymbol{P}^P_{S1}$ and $\boldsymbol{P}^P_{U_{s4}} = \boldsymbol{P}^P_{S2}$.

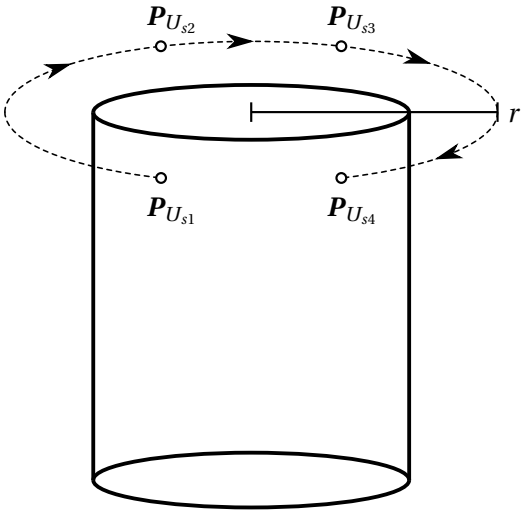


Figure 4.8: Method 2's example trajectories of step 2, 3 and 4

4.3.2 Results

Results of multiple flights showing the different number of points for each step can be found in this section's tables 4.4 to 4.7 and figures 4.9 to 4.12.
Those provide all possible combinations for a radius $r = 2\,\text{m}$ and $r = 3\,\text{m}$ with an angular velocity $\omega = 0.25\,\text{rad/s}$ and $\omega = 0.55\,\text{rad/s}$ starting step 0 in positive direction $d = 1$ at $\boldsymbol{P}^P_{U_{init}}$:

$$\boldsymbol{P}^P_{U_{init}} = \begin{bmatrix} r \\ \pi \\ z_{U_{init}} = 3.0 \end{bmatrix} \tag{4.12}$$

$t\,[s]$	step	segment id	n_{total}	n_{seg0}	n_{seg1}	n_{seg2}	n_{seg3}
25.13	0	−1	2460	1068	355	316	721
35.57	1	2	2404	427	350	493	1134
42.86	2	1	2538	427	737	615	759
50.16	3	0	2945	955	800	431	759
57.45	4	3	3120	1092	671	431	926

Table 4.4: Number of points n after each step when flying the "circular scan with static height autonomous segment rescan" with a radius of 2 m (1 m distance to the object surface) with $\omega = 0.25\,\text{rad/s}$

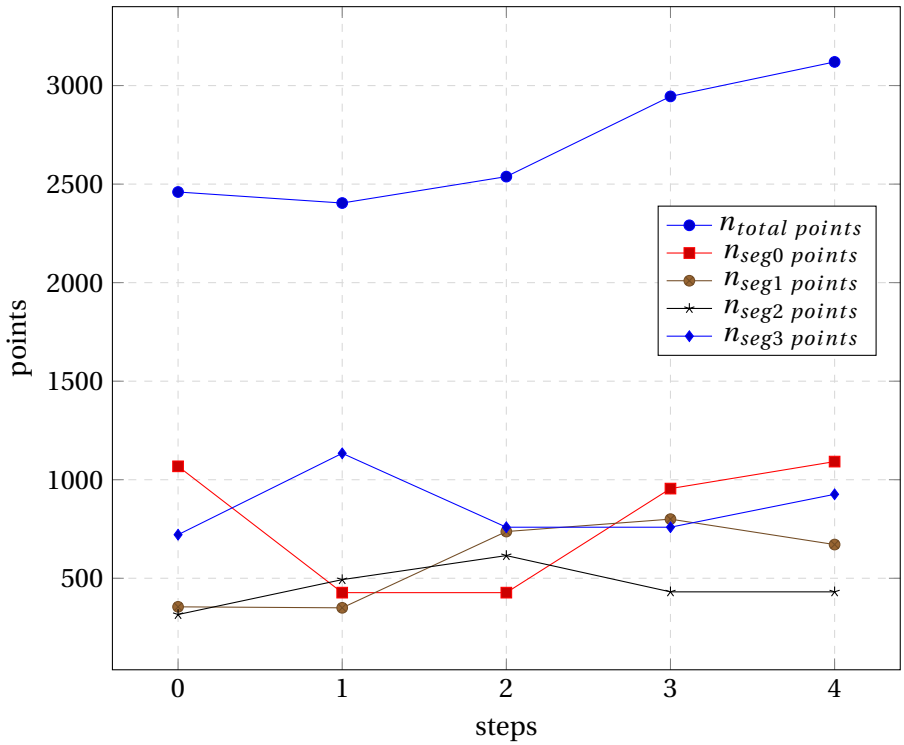


Figure 4.9: Plots showing the number of points after each step when flying the "circular scan with static height autonomous segment rescan" with a radius of 2 m (1 m distance to the object surface) with ω = 0.25 rad/s

$t\,[s]$	step	segment id	n_{total}	n_{seg0}	n_{seg1}	n_{seg2}	n_{seg3}
11.42	0	−1	2496	1129	463	343	561
16.72	1	1	2704	1238	338	343	785
20.6	2	2	2825	549	1096	610	570
24.46	3	3	2901	549	703	903	746
28.32	4	0	3274	892	703	522	1157

Table 4.5: Number of points n after each step when flying the "circular scan with static height autonomous segment rescan" with a radius of 2 m (1 m distance to the object surface) with ω = 0.55 rad/s

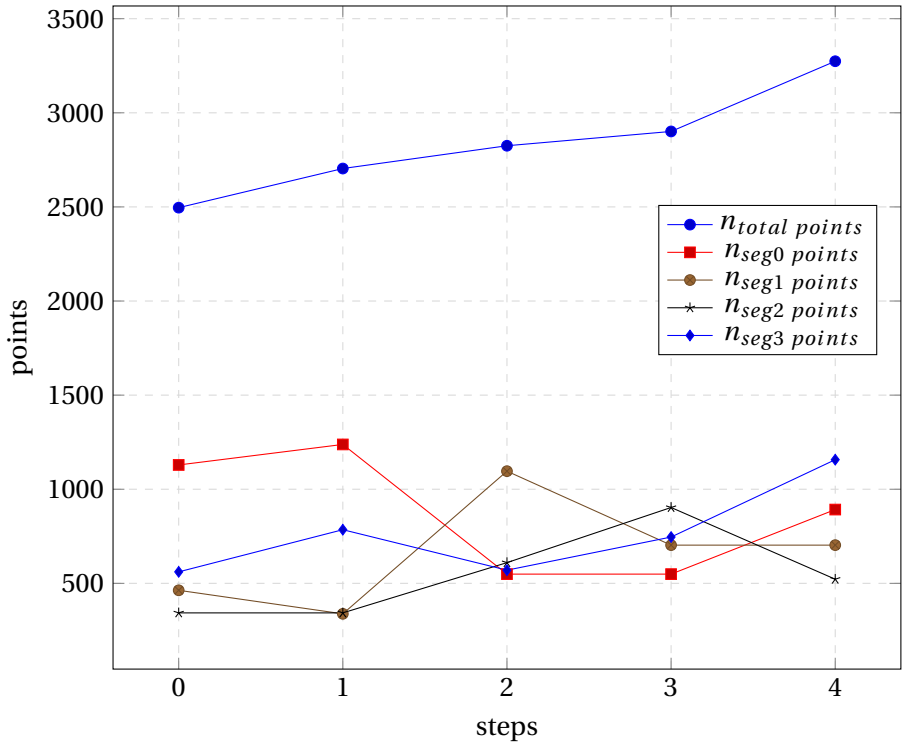


Figure 4.10: Plots showing the number of points after each step when flying the "circular scan with static height autonomous segment rescan" with a radius of 2 m (1 m distance to the object surface) with $\omega = 0.55\,\text{rad/s}$

$t\,[s]$	step	segment id	n_{total}	n_{seg0}	n_{seg1}	n_{seg2}	n_{seg3}
25.13	0	−1	3554	1037	800	810	907
35.58	1	1	3560	885	994	812	869
42.87	2	2	3500	830	864	935	871
50.17	3	3	3340	830	798	763	949
57.46	4	0	3258	906	798	704	850

Table 4.6: Number of points n after each step when flying the "circular scan with static height autonomous segment rescan" with a radius of 3 m (2 m distance to the object surface) with $\omega = 0.25\,\text{rad/s}$

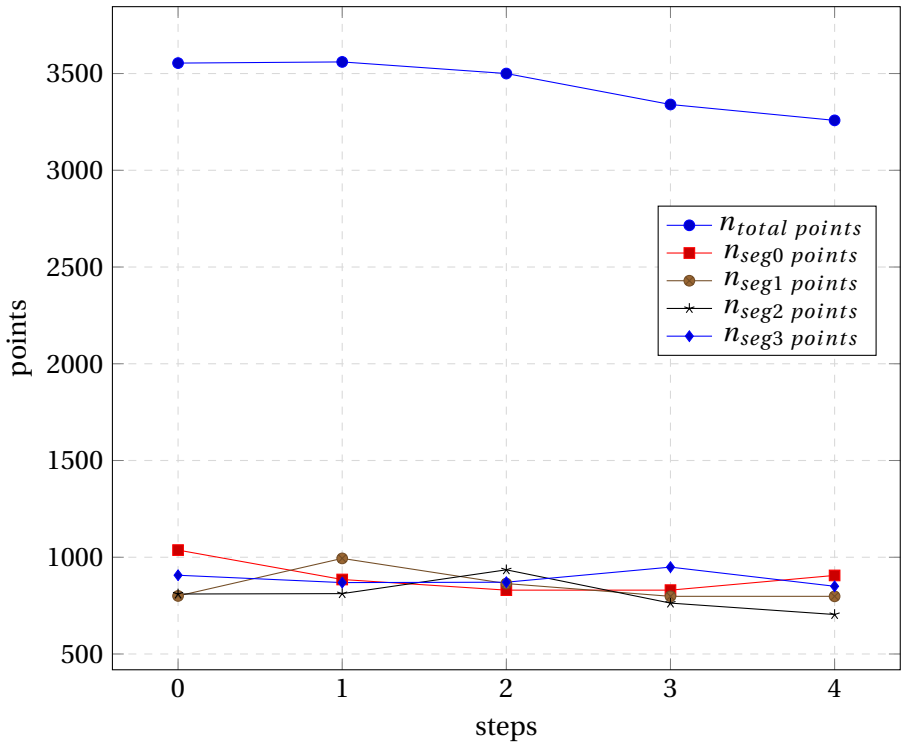


Figure 4.11: Plots showing the number of points after each step when flying the "circular scan with static height autonomous segment rescan" with a radius of 3 m (2 m distance to the object surface) with $\omega = 0.25$ rad/s

$t\,[s]$	step	segment id	n_{total}	n_{seg0}	n_{seg1}	n_{seg2}	n_{seg3}
11.42	0	−1	3529	944	875	1037	673
13.87	1	3	3573	835	854	1042	842
17.73	2	0	3657	771	854	1060	972
21.6	3	1	3712	960	873	1043	836
25.46	4	2	3671	800	992	1065	814

Table 4.7: Number of points n after each step when flying the "circular scan with static height autonomous segment rescan" with a radius of 3 m (2 m distance to the object surface) with $\omega = 0.55$ rad/s

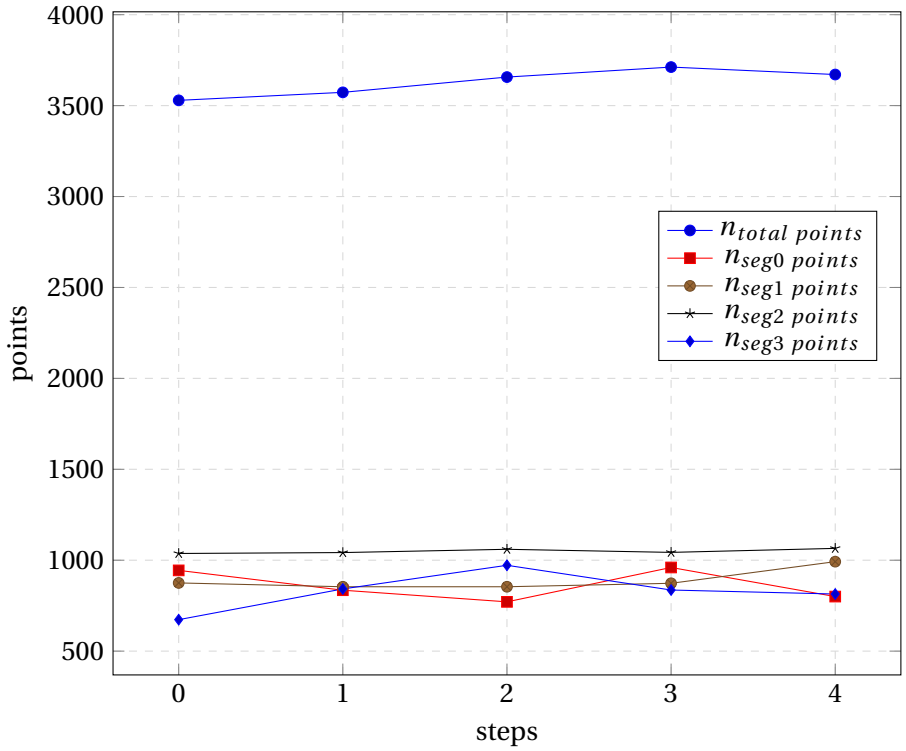


Figure 4.12: Plots showing the number of points after each step when flying the "circular scan with static height autonomous segment rescan" with a radius of 3 m (2 m distance to the object surface) with $\omega = 0.55\,\text{rad/s}$

4.3.3 Discussion

Comparing the results for $r = 2\,\text{m}$ in tables 4.4 and 4.5 and figures 4.9 and 4.10 with those for $r = 3\,\text{m}$ of tables 4.6 and 4.7 and figures 4.11 and 4.12, one can identify a significant difference in the total number of map points n_{total}. On the one hand, the reason might be a larger area of the object being visible due to the higher distance, but on the other hand, more wrong detected points could also be a reason. The latter cannot be excluded, especially due to indications found in the method described in section 4.2. There an increase in the number of rotations, for $r = 3\,\text{m}$, was leading to a decrease in the quantity of map points, which was indicating a high number of wrong detected map points for fewer rescans.
Furthermore, in the case of $r = 2\,\text{m}$, the total number of points for the whole object is noticeably increasing. However, for $r = 3\,\text{m}$ it is not changing significantly over the steps. Also by investigating the number of points of the segments, the first show a higher variation indicating a high number of points getting not only detected but also discarded. When passing segments obviously n_{sega} is increasing due to new detected points. After it is out of the FOV, while scanning other segments, only the error correction is affecting n_{sega} by culling map points.
Within step 2 to 4, the UAV is passing the remaining half of the previously reached segment and the first half of the destination segment of the current step. Due to this in the following step points for this segment can be accumulated, after a segment was reached in the previous step. In the flight presented in table 4.4 and figure 4.9, one can identify this for example for segment 2. Its $\boldsymbol{P}^{P}_{S2}$ is reached in step 1, as it had the lowest entropy after step 0 with 316 map points, increasing n_{seg2} to 493. As in the next step 2 to reach the neighbor segment 1 its remaining half gets passed, afterward $n_{seg2} = 615$. In step 3, where segment 2 is not visible only ORB-SLAM2's error correction seems affecting it as $n_{seg2} = 431$.
Slightly different for step 1 either a half or a full and a half segment can be passed. The second

also holds for the currently investigated flight. To reach $\boldsymbol{P}_{S2}^{P}$ the UAV has to completely pass segment 3 as $\phi_{U_{init}} < \phi_{S3} < \phi_{S2}$ and according to equation (4.10) in this case $d = 1$.
This leads to an increase of points in segment 3 after step 1 but in contrast to segment 2 already to a decrease in the following step.
A flight in the different direction, but also passing a full segment in step 1, can be found in table 4.5 and figure 4.10. Due to $\phi_{U_{init}} > \phi_{S0} > \phi_{S1}$ and according to equation (4.10) $d = 0$, the UAV is after the initial circle with $d = 1$ changing direction and thereby immediately passing segment 0 and the first half of segment 1. In the case of segment 0, it is increasing the quantity of map points but n_{seg1} is decreasing. This assumes that the amount of culled points is exceeding the newly detected ones or the aggregation of the latter is delayed, which could also cause the relatively high increase after the following step.
The other instance, where a half segment is passed in step 1, occurs for the flight of table 4.7 and figure 4.12 when flying from $\boldsymbol{P}_{U_{init}}^{P}$ to $\boldsymbol{P}_{S3}^{P}$. In contrast to the flights with $r = 2\,\text{m}$, a more steady quantity of map points per segment can be observed.
A reason for this, in general when $r = 3\,\text{m}$, could be more keyframes are created due to a higher amount of detected features resulting from the larger area of the object within the FOV. Furthermore, due to the higher radius, a longer distance is flight, which causes longer flight time. This leads to more captured frames and therefore the possibility of more created keyframes. More keyframes would offer the opportunity for more culled map points within the duration of one step. Another reason could be as previously mentioned that already in section 4.2 for $r = 3\,\text{m}$ few rescans might avoid proper map point culling.
In contrast to changing the radius, changing the angular velocity ω does not lead to noticeable different characteristics in the results. This supposes that ORB-SLAM2 at this order of magnitude behaves stably.

4.4 Method 3: Circular Scan with Vertical Autonomous Segment Rescan

This method is very similar to the previously described one in section 4.3. It also follows the idea of segment rescanning and evolves it with additional steps for flying a vertical trajectory at each segment.
By also reaching positions providing a closer view of areas with probably a low number of map points or no points, it should increase the entropy significantly.

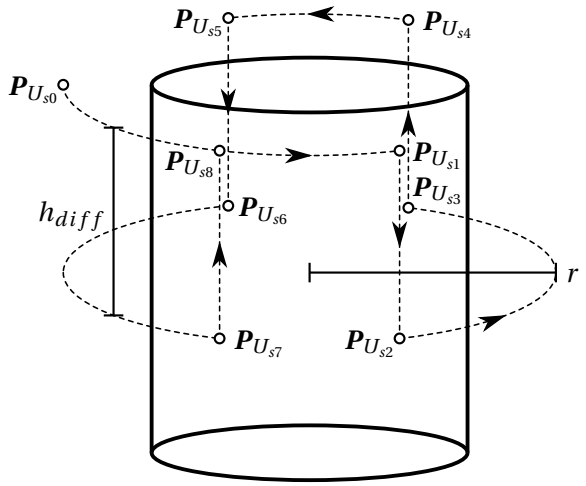


Figure 4.13: Method 3's trajectories for each step after the initial circle step 0

4.4.1 Path Steps

Step 0 and 1: Initial Circle and Lowest Global Entropy

These first two steps are identical to those described in section 4.3.1.

Step 3, 5, 7: Lowest Neighbor Entropy and Keep Direction

The steps to reach the next segment are nearly identical to the previously described method's steps 2 to 4 in section 4.3.1. Only their $\boldsymbol{P}^P_{st}$ and $\boldsymbol{P}^P_{dst}$ might change in the z-coordinate due to the additional vertical steps described in the following description of steps 2, 4, 6 and 8.

Step 2, 4, 6 and 8: Vertical Segment Scan

Dependent on the radius, or rather the distance, to the object, a varying large area is covered by the stereo camera's FOV. By also flying the initial circle in this method or the whole method, as in section 4.3, around the top of the object, the lower regions of the cylinder might suffer from a low density of points or are not covered.
To overcome this in these steps a vertical motion along the previously reached segment is flight. The desired velocity v_z and the height difference h_{diff} have to be provided to the algorithm to calculate the limiting flight time Δt_s for the step as in equation (4.13).

$$\Delta t_s = \frac{h_{diff}}{v_z} \tag{4.13}$$

The direction of the motion has to alternate in negative or positive z-direction starting from the step's z-position at the begin of the trajectory z_0. As step 0 and step 1 do not change the height of the UAV, for the first vertical segment scan, step 2 follows its trajectory $z(t)$ represented by line 3 in algorithm 4.2.
Visualizing this in figure 4.13, after the example flight of step 1 from $\boldsymbol{P}^P_{U_{s0}}$ to $\boldsymbol{P}^P_{U_{s1}} = \boldsymbol{P}^P_{S2}$, for step 2 it is going down the distance h_{diff} to $\boldsymbol{P}^P_{U_{s2}}$.
After a flight of step 3 to the next segment $\boldsymbol{P}^P_{U_{s3}} = \boldsymbol{P}^P_{S1}$, the UAV is flying to the original height. To reach $\boldsymbol{P}_{U_{s4}}$ in step 4 the trajectory of line 5 in algorithm 4.2 is calculated.
Step 6 follows the same principle as step 2 and step 8 the same as step 4.

Algorithm 4.2: Calculation of $z(t)$ of the trajectory for a vertical step of Δt_s duration

```
while t < Δt_s do                                   1
    if step 2 ∨ step 6 then                         2
        z(t) ← z_0 − v_z t                          3
    else                                            4
        z(t) ← z_0 − h_diff + v_z t                 5
    end                                             6
end                                                 7
```

4.4.2 Results

In this section, the results of multiple object coverage flights are presented with different angular velocities ω, vertical velocities v_z and at a different radius r around the cylinder in tables 4.8 to 4.15 and figures 4.14 to 4.21.
It was already elaborated on ORB-SLAM2 losing track if not being restarted after a previous coverage flight in section 3.3. For the methods presented in sections 4.2 and 4.3, this was necessary as well, but the behavior and consequences here are, compared to the others, more demonstrative and, due to this, provided in table 4.16 and figure 4.22.

$t\,[s]$	step	segment id	n_{total}	n_{seg0}	n_{seg1}	n_{seg2}	n_{seg3}
25.13	0	−1	2660	1061	370	400	829
35.58	1	1	2660	1061	370	400	829
46.58	2	1	2812	477	1255	415	665
53.88	3	2	3160	452	1103	940	665
64.88	4	2	3558	452	913	1513	680
72.17	5	3	3463	447	910	1135	971
83.18	6	3	3912	464	910	912	1626
90.47	7	0	4304	1043	910	894	1457
101.47	8	0	4538	1510	926	894	1208

Table 4.8: Number of points n after each step when flying the "circular scan with vertical autonomous segment rescan" at each segment with a radius of 2 m (1 m distance to the object surface) with $\omega = 0.25\,\text{rad/s}$, $v_z = 0.15\,\text{m/s}$

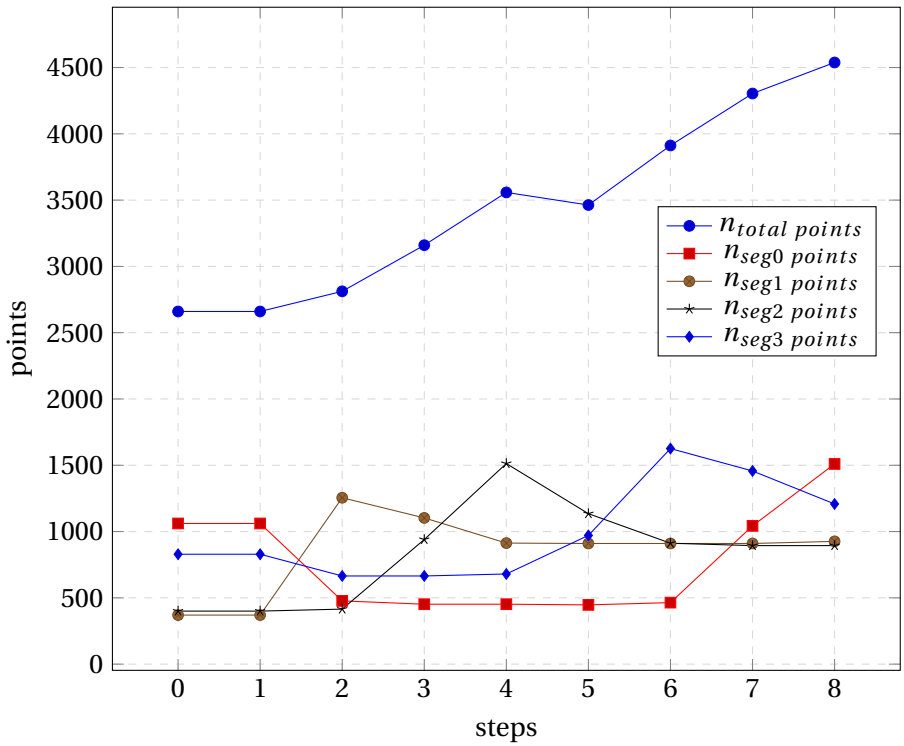


Figure 4.14: Plots showing the number of points after each step when flying the "circular scan with vertical autonomous segment rescan" at each segment with a radius of 2 m (1 m distance to the object surface) with $\omega = 0.25\,\text{rad/s}$, $v_z = 0.15\,\text{m/s}$

$t\,[s]$	step	segment id	n_{total}	n_{seg0}	n_{seg1}	n_{seg2}	n_{seg3}
11.42	0	−1	2908	1461	503	360	584
16.72	1	2	2407	467	375	367	1198
27.72	2	2	2863	457	376	1345	685
31.58	3	1	3341	457	696	1508	680
42.6	4	1	3381	474	1328	899	680
46.46	5	0	3460	599	1287	894	680
57.46	6	0	4190	1716	890	893	691
61.32	7	3	4408	1731	872	893	912
72.32	8	3	4502	1122	871	900	1609

Table 4.9: Number of points n after each step when flying the "circular scan with vertical autonomous segment rescan" at each segment with a radius of 2 m (1 m distance to the object surface) with $\omega = 0.55\,\text{rad/s}$, $v_z = 0.15\,\text{m/s}$

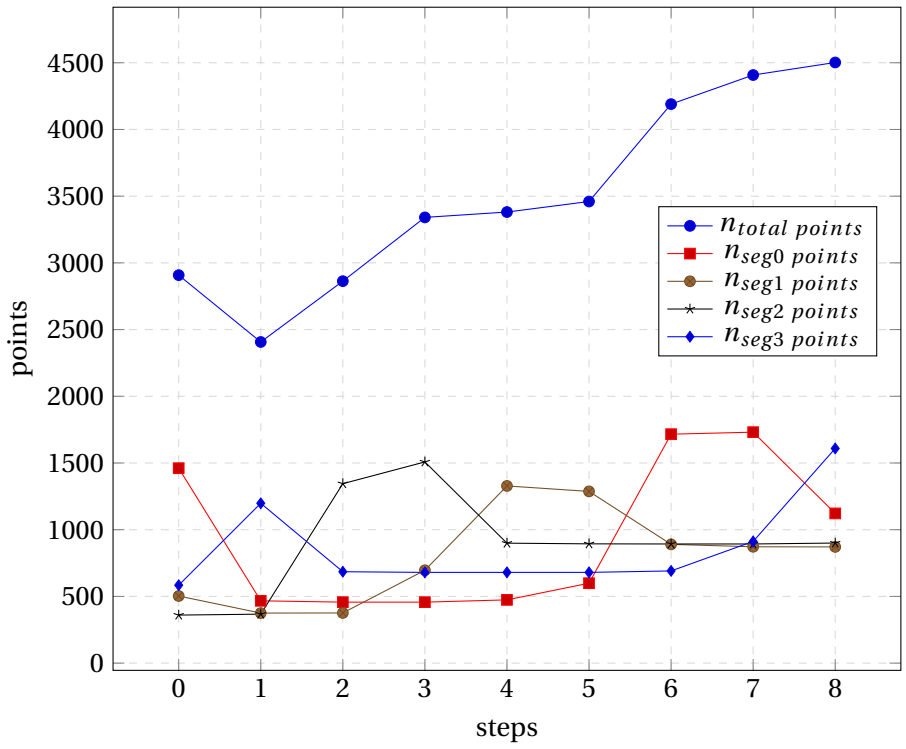


Figure 4.15: Plots showing the number of points after each step when flying the "circular scan with vertical autonomous segment rescan" at each segment with a radius of 2 m (1 m distance to the object surface) with $\omega = 0.55\,\text{rad/s}$, $v_z = 0.15\,\text{m/s}$

$t\,[s]$	step	segment id	n_{total}	n_{seg0}	n_{seg1}	n_{seg2}	n_{seg3}
25.13	0	−1	2495	1079	354	330	732
35.57	1	2	2380	385	344	824	827
41.57	2	2	2897	385	352	1439	721
48.86	3	1	3208	385	788	1328	707
54.86	4	1	3544	403	1340	1094	707
62.15	5	0	3494	396	1305	1086	707
68.15	6	0	4142	1593	750	1074	725
75.44	7	3	4278	1203	733	1074	1268
81.44	8	3	4430	927	733	1085	1685

Table 4.10: Number of points n after each step when flying the "circular scan with vertical autonomous segment rescan" at each segment with a radius of 2 m (1 m distance to the object surface) with ω = 0.25 rad/s, $v_z = 0.3$ m/s

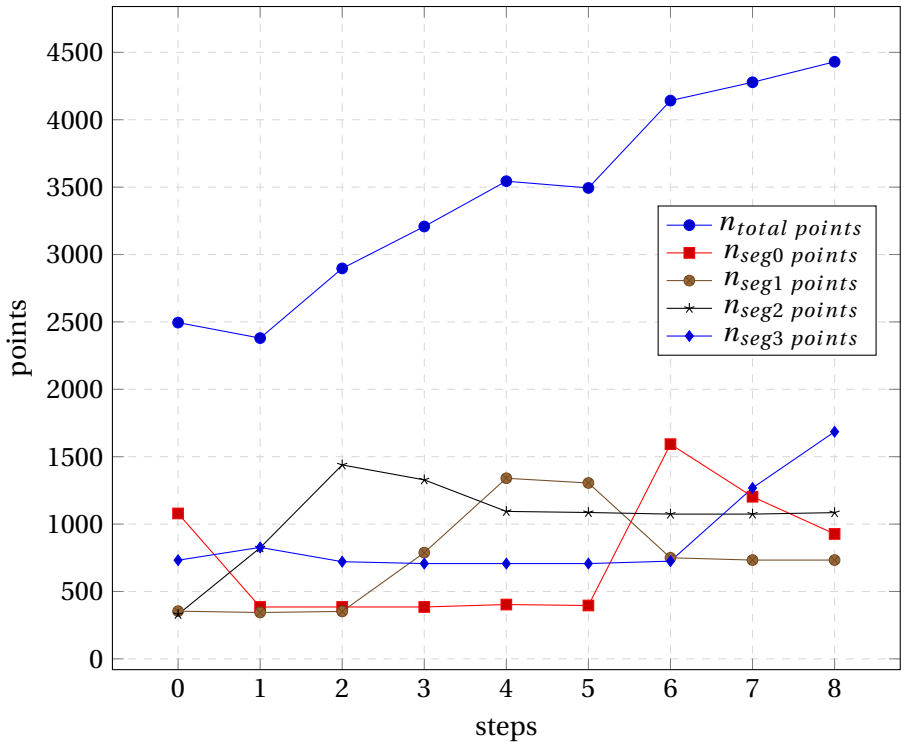


Figure 4.16: Plots showing the number of points after each step when flying the "circular scan with vertical autonomous segment rescan" at each segment with a radius of 2 m (1 m distance to the object surface) with $\omega = 0.25$ rad/s, $v_z = 0.3$ m/s

$t\,[s]$	step	segment id	n_{total}	n_{seg0}	n_{seg1}	n_{seg2}	n_{seg3}
11.42	0	−1	2502	1041	451	348	662
16.73	1	2	2432	453	376	416	1187
22.73	2	2	2891	453	380	1283	775
26.59	3	1	3091	453	577	1301	760
32.61	4	1	3403	468	1316	859	760
36.47	5	0	3588	695	1290	843	760
42.47	6	0	3991	1483	889	843	776
46.33	7	3	4264	1537	871	843	1013
52.33	8	3	4448	974	871	852	1751

Table 4.11: Number of points n after each step when flying the "circular scan with vertical autonomous segment rescan" at each segment with a radius of 2 m (1 m distance to the object surface) with $\omega = 0.55\,\text{rad/s}$, $v_z = 0.3\,\text{m/s}$

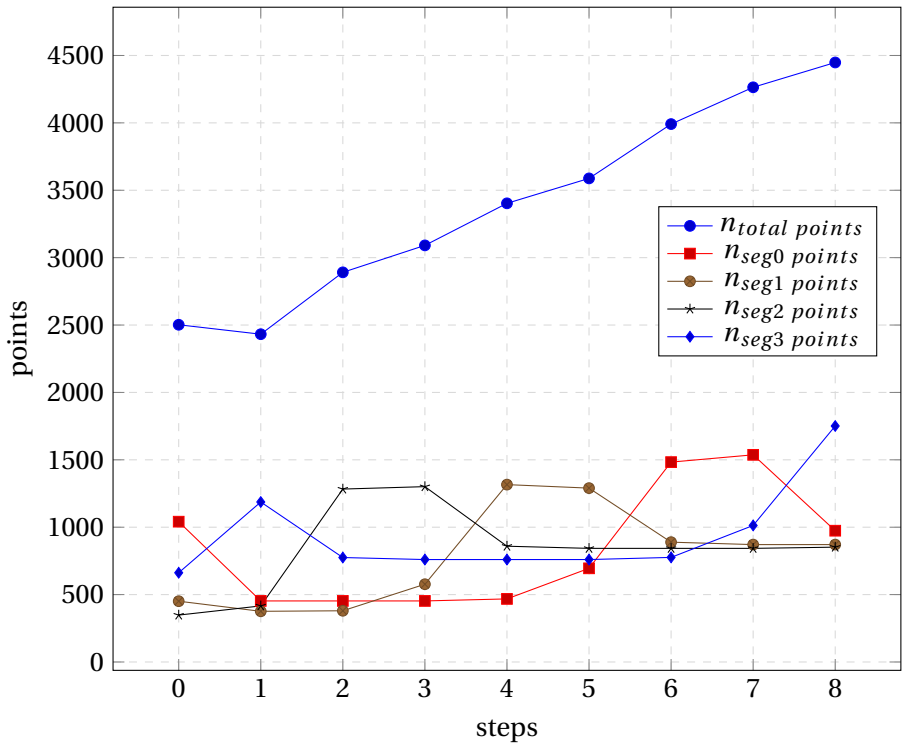


Figure 4.17: Plots showing the number of points after each step when flying the "circular scan with vertical autonomous segment rescan" at each segment with a radius of 2 m (1 m distance to the object surface) with $\omega = 0.55\,\text{rad/s}$, $v_z = 0.3\,\text{m/s}$

$t\,[s]$	step	segment id	n_{total}	n_{seg0}	n_{seg1}	n_{seg2}	n_{seg3}
25.13	0	−1	3617	1065	862	802	888
35.57	1	2	3464	858	854	915	837
46.57	2	2	4035	858	879	1469	829
53.86	3	3	4385	858	875	1518	1134
64.87	4	3	4831	919	875	1528	1509
72.16	5	0	4705	997	877	1515	1316
83.17	6	0	5158	1414	906	1516	1322
90.46	7	1	5612	1444	1339	1516	1313
101.47	8	1	5962	1463	1616	1570	1313

Table 4.12: Number of points n after each step when flying the "circular scan with vertical autonomous segment rescan" at each segment with a radius of 3 m (2 m distance to the object surface) with $\omega = 0.25\,\text{rad/s}$, $v_z = 0.15\,\text{m/s}$

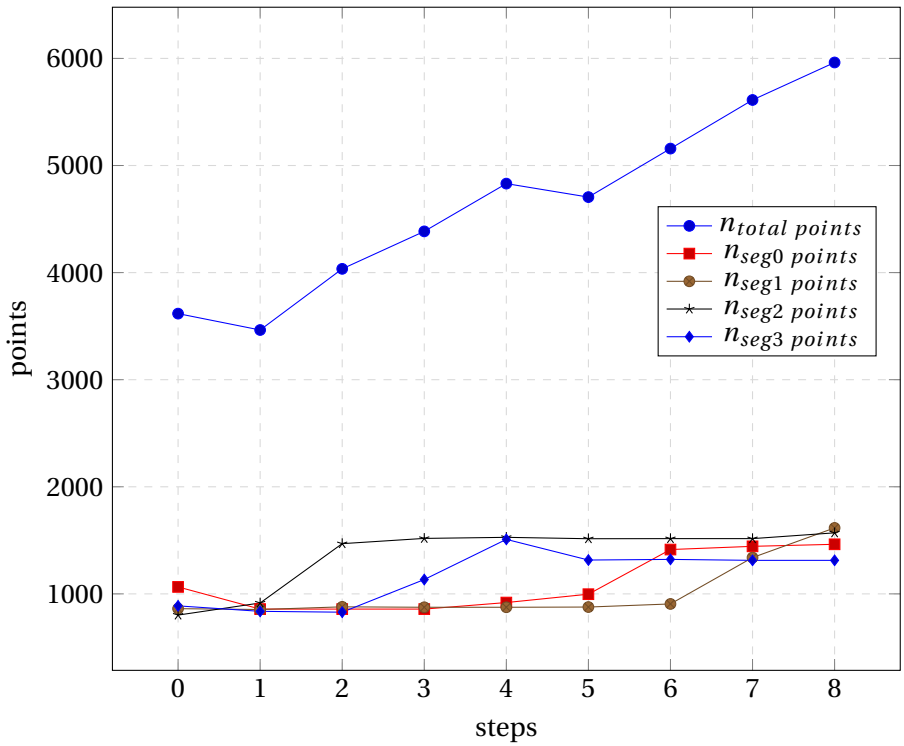


Figure 4.18: Plots showing the number of points after each step when flying the "circular scan with vertical autonomous segment rescan" at each segment with a radius of 3 m (2 m distance to the object surface) with $\omega = 0.25\,\text{rad/s}$, $v_z = 0.15\,\text{m/s}$

$t\,[s]$	step	segment id	n_{total}	n_{seg0}	n_{seg1}	n_{seg2}	n_{seg3}
11.42	0	−1	4249	1251	1330	1145	523
13.86	1	3	4297	1118	1298	1135	746
24.87	2	3	4984	1125	1298	1189	1372
28.73	3	0	5424	1334	1298	1186	1606
39.73	4	0	5621	1652	1330	1184	1455
43.6	5	1	5621	1651	1331	1184	1455
54.6	6	1	5975	1556	1744	1235	1440
58.47	7	2	6599	1555	1896	1708	1440
69.47	8	2	6715	1555	1799	1891	1470

Table 4.13: Number of points n after each step when flying the "circular scan with vertical autonomous segment rescan" at each segment with a radius of 3 m (2 m distance to the object surface) with $\omega = 0.55\,\text{rad/s}$, $v_z = 0.15\,\text{m/s}$

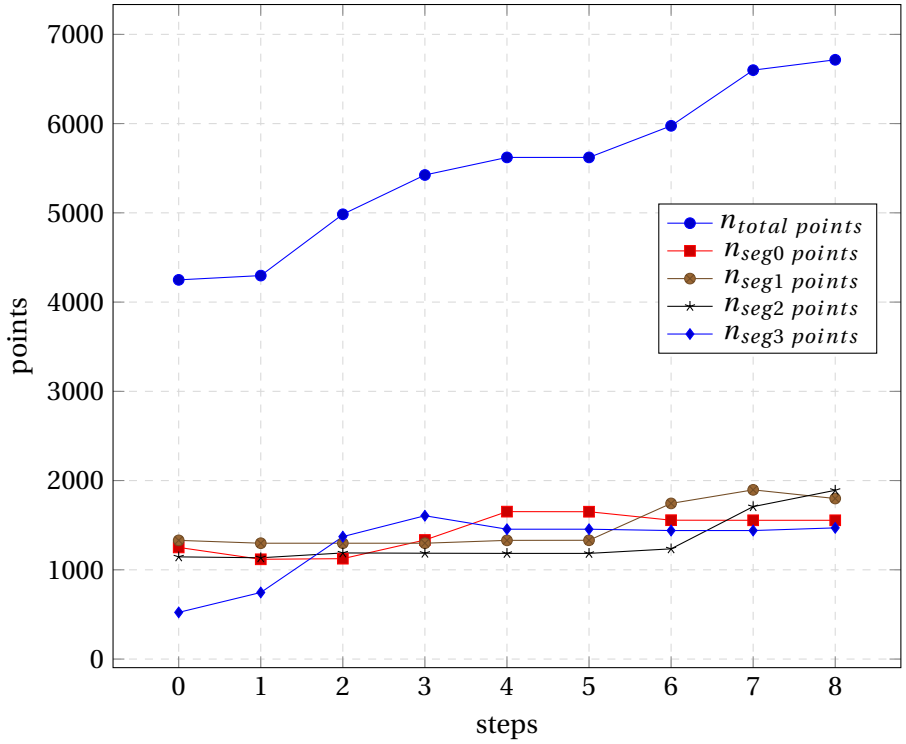


Figure 4.19: Plots showing the number of points after each step when flying the "circular scan with vertical autonomous segment rescan" at each segment with a radius of 3 m (2 m distance to the object surface) with $\omega = 0.55\,\text{rad/s}$, $v_z = 0.15\,\text{m/s}$

$t\,[s]$	step	segment id	n_{total}	n_{seg0}	n_{seg1}	n_{seg2}	n_{seg3}
25.13	0	−1	3560	1073	870	721	896
35.57	1	2	3447	846	844	919	838
41.57	2	2	3748	846	868	1237	797
48.86	3	3	4219	848	870	1404	1097
54.87	4	3	4662	923	870	1410	1459
62.16	5	0	4547	913	870	1409	1355
68.18	6	0	4966	1351	900	1398	1317
75.47	7	1	5489	1485	1294	1398	1312
81.47	8	1	5880	1493	1627	1448	1312

Table 4.14: Number of points n after each step when flying the "circular scan with vertical autonomous segment rescan" at each segment with a radius of 3 m (2 m distance to the object surface) with $\omega = 0.25\,\text{rad/s}$, $v_z = 0.3\,\text{m/s}$

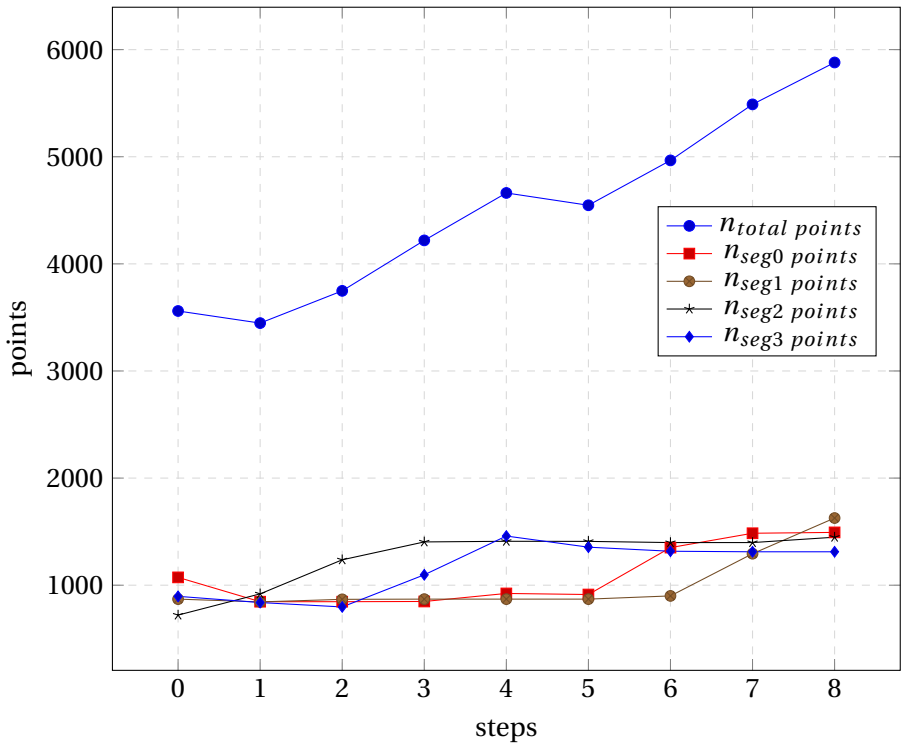


Figure 4.20: Plots showing the number of points after each step when flying the "circular scan with vertical autonomous segment rescan" at each segment with a radius of 3 m (2 m distance to the object surface) with $\omega = 0.25\,\text{rad/s}$, $v_z = 0.3\,\text{m/s}$

$t\,[s]$	step	segment id	n_{total}	n_{seg0}	n_{seg1}	n_{seg2}	n_{seg3}
11.42	0	−1	3116	940	783	759	634
16.73	1	2	3348	707	764	878	999
22.73	2	2	3987	691	800	1652	844
26.59	3	1	4399	691	999	1879	830
32.59	4	1	5106	755	1701	1820	830
36.45	5	0	5114	852	1635	1797	830
42.45	6	0	5387	1204	1528	1789	866
46.31	7	3	6003	1552	1530	1789	1132
52.32	8	3	6283	1422	1529	1817	1515

Table 4.15: Number of points n after each step when flying the "circular scan with vertical autonomous segment rescan" at each segment with a radius of 3 m (2 m distance to the object surface) with $\omega = 0.55\,\text{rad/s}$, $v_z = 0.3\,\text{m/s}$

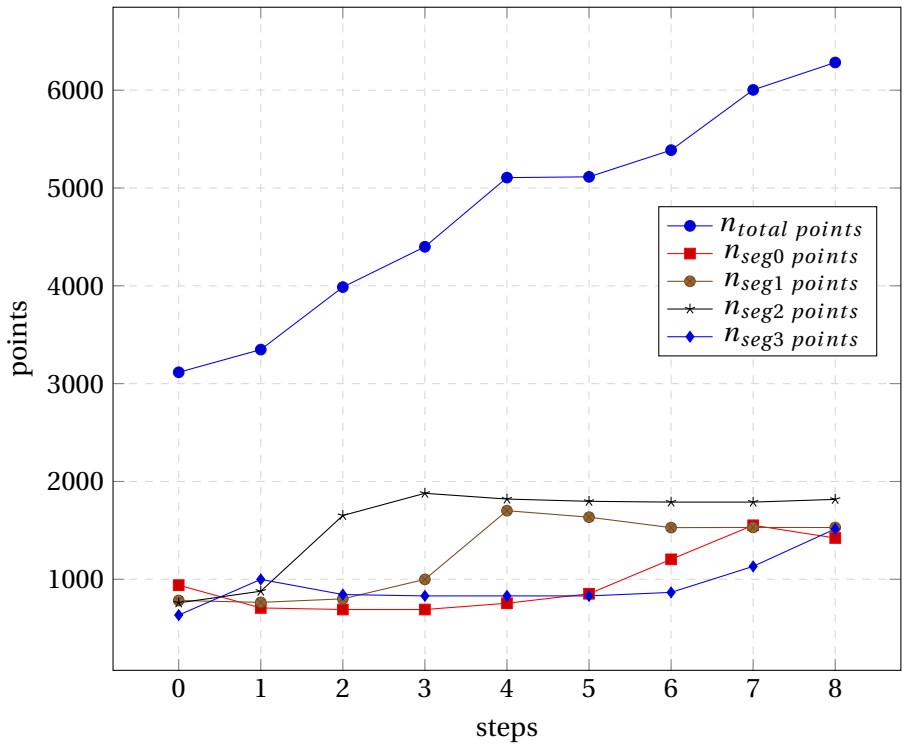


Figure 4.21: Plots showing the number of points after each step when flying the "circular scan with vertical autonomous segment rescan" at each segment with a radius of 3 m (2 m distance to the object surface) with $\omega = 0.55\,\text{rad/s}$, $v_z = 0.3\,\text{m/s}$

$t\,[s]$	step	segment id	n_{total}	n_{seg0}	n_{seg1}	n_{seg2}	n_{seg3}
25.13	0	−1	2634	1125	387	392	730
35.57	1	1	2970	790	1111	398	671
41.58	2	1	2973	749	1157	396	671
48.87	3	2	2973	749	1157	396	671
54.88	4	2	2973	749	1157	396	671
62.17	5	3	2973	749	1157	396	671
68.17	6	3	3498	728	739	414	1617
75.46	7	0	3897	1379	739	394	1385
81.46	8	0	4155	1815	753	394	1193

Table 4.16: Number of points n after each step when flying the "circular scan with vertical autonomous segment rescan" at each segment with a radius of 2 m (1 m distance to the object surface) with ω = 0.25 rad/s, v_z = 0.3 m/s. ORB-SLAM2 losing track during step 2.

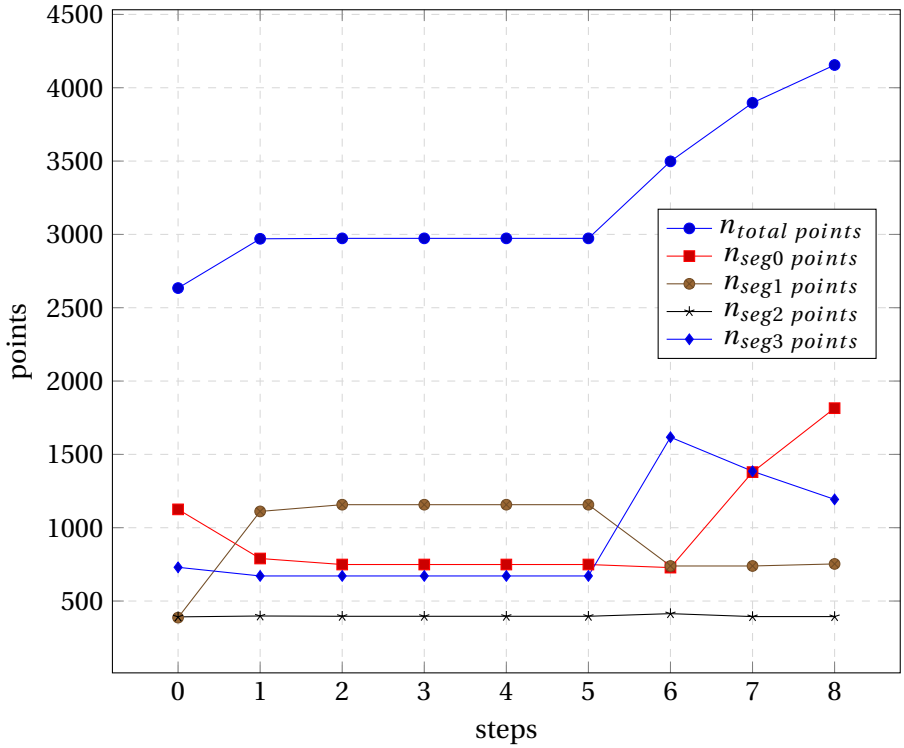


Figure 4.22: Plots showing the number of points after each step when flying the "circular scan with vertical autonomous segment rescan" at each segment with a radius of 2 m (1 m distance to the object surface) with ω = 0.25 rad/s, v_z = 0.3 m/s. ORB-SLAM2 losing track during step 2.

4.4.3 Discussion

The impact of the newly introduced steps for the vertical scan can be identified by a noticeable increase of the n_{sega} of the different segments at the corresponding steps 2, 4, 6 and 8 in the cases where tracking was not lost as in tables 4.8 to 4.15 and figures 4.14 to 4.21. A large part of the in these steps gained increase in additional map points seems to be not considered as wrong detected features by ORB-SLAM2. Compared to the results without them in section 4.3 even the total number of map points for r = 3 m is increasing significantly over steps, as one can see in tables 4.12 to 4.15 and figures 4.18 to 4.21.

This seems to be also the cause that after all steps n_{total} increased to a far higher quantity than in the previous method with static height for $r = 2\,\text{m}$ and $r = 3\,\text{m}$.
Besides the changes of n_{sega} for transitions to the next segment showing the same characteristics as described in section 4.3, for the cases where $r = 3\,\text{m}$ an influence of the vertical scan can be found. For segments reached in step 1 and flight down in step 2 or reached in step 5 and flight down in step 6, in the step after, which is performing the transition to the next segment, n_{sega} is increasing. In unlike for segments reached in step 3 and flight up in step 4, during the transition step afterwards the number of map points is decreasing. Because for the first case, the transition is at z=1.5m, but for the second case at $z = 3\,\text{m}$. Therefore at $z = 3\,\text{m}$, the UAV is rescanning along the same path as the initial circle of step 0.
The rescan at $z = 3\,\text{m}$ might lead to a lower quantity of new detected points and might be outnumbered by the map points culled due to error correction. Another similarity with the previous method, where flights with $r = 2\,\text{m}$ showed also a less steady quantity of map points per segment than those with $r = 3\,\text{m}$, strengthens the raised assumptions of more correction steps within a step or few rescans leading to less wrong detected map points getting culled.
Furthermore, changing the angular velocity ω again does not lead to noticeable different characteristics in the results. Same holds for v_z of the vertical scan steps. Leading to the assumption that ORB-SLAM2 in z-direction for this order of magnitude behaves stably.
Therefore, one can compare the result of this method for a radius of 2 m, with $\omega = 0.55\,\text{rad/s}$, $v_z = 0.3\,\text{m/s}$, presented in table 4.11 and figure 4.17, with the results of flying the helix method with two rotations at a radius of 2 m, with $\omega = 0.25\,\text{rad/s}$, in table 4.1 and figure 4.4. In the first case, a total duration of $52.33s$ leads to 4448 map points detected by ORB-SLAM2. This is noticeably more than the 3823 points in the case of the helix flight, but for a similar duration of $50.27s$.
This shows the potential in this method as it is outperforming the common helix method for similar flight time and could reduce flight time at higher velocities.
In table 4.16 and figure 4.22 the results of a flight with ORB-SLAM2 losing track during step 2 is shown. One can notice it by the constant number of map points after step 2 to 5.
Although ORB-SLAM2 lost tracking, due to not being restarted before the test flight, is an unpleasant outcome, it provides the possibility to show a basic degree of robustness of the trajectory planning method. As long as during step 0 the tracking is not lost but at a later step, it is possible to relocalize the UAV and continue building the map. Even if the tracking is lost as in this case at segment 1, it is found again at the opposite side of the cylinder at segment 3.

5 Conclusions

The initial question of this research was the following:

> "Which SLAM algorithm would be useful to create 3D visual models on-the-fly?"

Due time constraints searching the literature as described in section 1.3.1 was necessary to determine a suitable candidate. Following previous studies ORB-SLAM2, by Mur-Artal and Tardos (2017), was considered as the most appropriate to be further invested in chapter 3. There it turned out that it shows weaknesses when textures on the object of interest are used, which show a repetitive pattern. This was related to its loop closing functionality. Loop closing is intended to reduce the position error, which was also shown by Gaspar et al. (2018) and Giubilato et al. (2018). The same was expected for this research but could not be confirmed. However, with textures that contained some variety, a reliable tracking could be discovered in the simulation. Furthermore, the position estimated by ORB-SLAM2 showed an absolute error of similar magnitude as discovered in a real-world experiment by Giubilato et al. (2018).
Also, the following question could be answered:

> "Does multiple coverage of an object's surface lead to an improved 3D visual model of it?"

In the case of the helix method in section 4.2, it was shown that increasing rotations leads at approximately 25 rotations, for each distance to the object, to a saturation in the total number of points. On the one hand, this indicated that by further increasing them the point cloud 3D model of the object cannot be improved anymore, but on the other hand it did until this point by multiple coverage.
Accordingly, the two test series for a radius of 2 m and 2.5 m showed an increase of map points until reaching saturation. An interesting outcome was that for a radius of 3 m the number of points was decreasing while increasing rotations, in contrast to the other two test series. This suggests that the higher number of points from flights with lower rotations originates in wrongly detected features. Removing wrong detected map points is thereby the other type of improvement of a point cloud, that could be discovered.
This insight also showed that there might be uncertainty if the increase in the number of map points is representing always an improvement of the model. This uncertainty required investigations in the behavior after each step of the other two methods in sections 4.3 and 4.4, to examine the following research question:

> "In order to autonomously improve the 3D visual model of an object, is it possible to determine suitable flight directions on-the-fly, which lead also to a reduction in flight time?"

Generally, the variability in the number of map points could be related to the point culling of ORB-SLAM2.
In the case of the method "circular scan with vertical autonomous segment rescan", in section 4.4, also a significant increase in the number of points could be detected during all flights. Furthermore, for different vertical and angular velocities, it is showing similar results. Also, a to a helix test flight comparable flight of this method could be determined and shown to be capable of outperforming it for similar flight time and showing potential to reduce flight time at higher velocities.

6 Future work

Especially due to time limitations some extensions of this work could not be considered, but are presented in this chapter for possible future work(s):

- Fine-grained 3D segmentation. In this work, the whole object was split in four quarters on a two-dimensional plane. Using for example voxels to segment the point cloud in 3D could be used to develop more complex multiple coverage methods.
- Add accuracy as an additional constraint for reasoning on the model improvement. Considering the impact of map point culling of ORB-SLAM2 on the number of points, it might be beneficial to reason if it increases the accuracy of a resulting model.
- Test the error of the by ORB-SLAM2 estimated camera orientation compared to the real one. Similar as in the case of its position in section 3.4, this could provide insight regarding the accuracy of ORB-SLAM2 in the simulation.
- Usage of different shaped objects of interest. Although it might have been beneficial to compare the different methods with an object that provides a constant distance to the camera, others could be providing additional insight.
- Real-world experiments. By repeating the tests with the same setup in real it should be possible to determine if the simulation is an appropriate testbed.

A Implementation

In this appendix details about the implementation of this research are presented.
The whole work is based on the Robot Operating System (ROS) middleware (Open Robotics (2019b)) in combination with the Gazebo robotics simulator (Open Robotics (2019a)).
Using the RotorS Gazebo simulator framework, of Furrer et al. (2016), for it provides several advantages due to its modular design. One is the possibility to use own designed UAV-models within it, in this case, "betaX" shown in figure 1.2. The other to provide an own controller for it. The controller for "betaX", by Rashad et al. (2019), used in this research will be published soon. In total the structure of its modular design is also aimed towards matching a real UAV as close as possible. According to Furrer et al. (2016), this should provide easy portability of used components to be also used on a real UAV without any changes in the ideal case.
In the following appendices A.1 and A.2, the data flow between the different components for scenarios in this research is shown.
One of the components is ORB-SLAM2 by Mur-Artal and Tardos (2017). It has to be stated that an improved version has been used. This version was made public by the appliedAI Initiative (2019). It is more comprehensive and up-to-date with respect to its integration into ROS. Its point cloud output and the pose of the UAV relative to it has been in every scenario observed with rviz (Open Robotics (2019c)). For simplicity, this is neither mentioned in appendices A.1 and A.2 nor added to figures A.1 and A.2.

A.1 Error Between "Ground Truth" and Estimated Flight Trajectory

In section 3.4 the absolute error, on the xy-plane $|\Delta x| + |\Delta y|$ and in z-direction $|\Delta z|$, between the camera position $\boldsymbol{P}^{W}_{estC}(t)$, estimated by ORB-SLAM2, and the "physical"/ground truth position within the simulation $\boldsymbol{P}^{W}_{phC}(t)$, was investigated. The interaction of components needed for this is described here.
The simulation, represented by the RotorS block in figure A.1, is aware of the physical camera pose but not of the estimated camera pose by itself. It is retrieved from ORB-SLAM2, which has

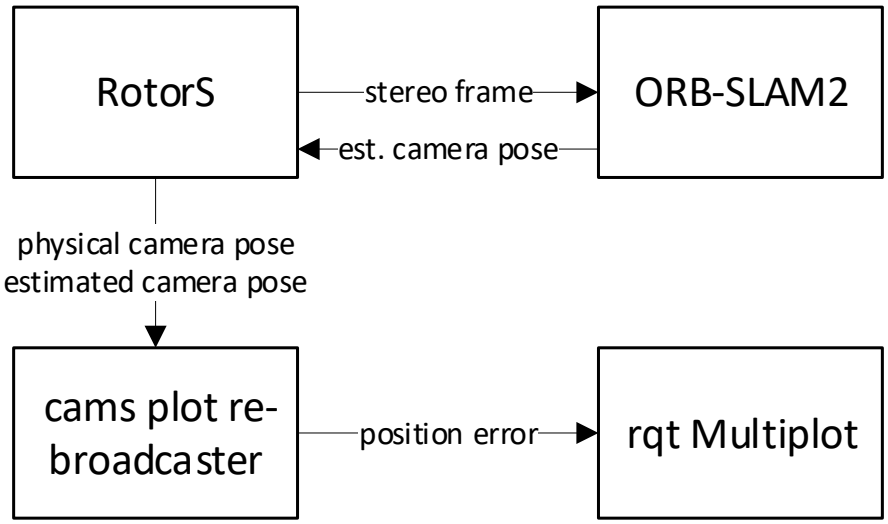


Figure A.1: Data flow between the involved components when measuring the error between "ground truth" and estimated flight trajectory

to get stereo frames from the simulation's camera to estimate it as described in section 2.3.1.
The "physical", as well as the estimated camera pose, are taken from the simulation by the "cams plot re-broadcaster". This component was developed for this research to calculate the position error and provide it to rqt Multiplot. The latter has been developed by ANYbotics (2019). It was used to show live the error during the simulated flights of the UAV and also for capturing it related to the simulation time to be presented as a plot in figure 3.4.

A.2 Trajectory Planning Methods for 3D Reconstruction

Different trajectory planning methods have been described in chapter 4. Two of them, which are expressed in sections 4.3 and 4.4, are doing online trajectory planning based on the point cloud provided by ORB-SLAM2.
To create and continuously improve this map of points, as described in sections 2.3.2 and 2.3.3, it receives in this work stereo frames. These stereo frames are gained from a simulated stereo camera in the simulation, which is represented as the RotorS block in the dataflow diagram in figure A.2.
Originating at ORB-SLAM2 the point cloud is received by the "Point Cloud Processor". It was developed for this research to determine the number of points per segments and do the previously mentioned online trajectory planning. Moreover, to remove the majority of the map point outliers, it is filtering out points outside of a cuboid, which is enclosing the object. The results of the methods, which are described in chapter 4, are also written into log files by the "Point Cloud Processor".
The outcome of its trajectory planning represents a constantly published setpoint, consisting of the position and yaw angle, sent to the graphical user interface (GUI). The position is calculated as described in chapter 4, but the yaw angle is set such that the camera on the UAV points always towards the cylinder. The GUI, which is generally capable of calculating offline trajectories and sending the regarding setpoints to RotorS' UAV controller, in this case, is overriding its calculated setpoint and providing the one from the "Point Cloud Processor" to the simulation.

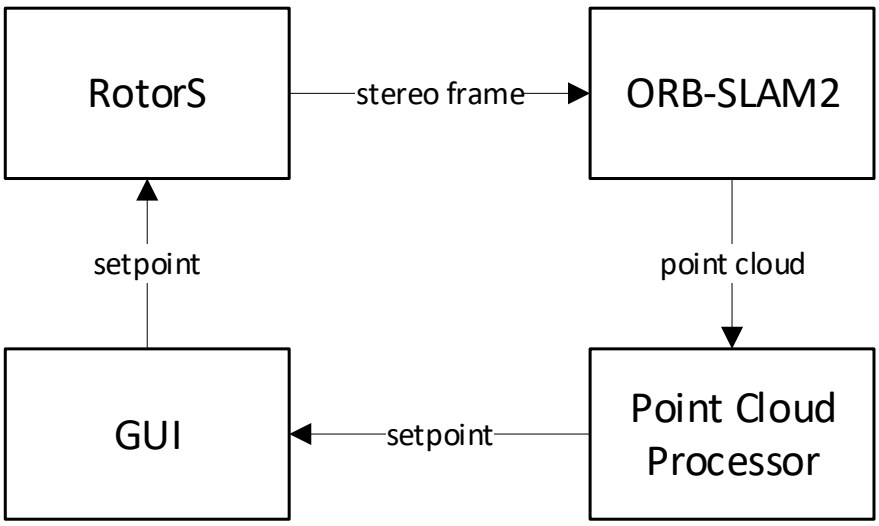


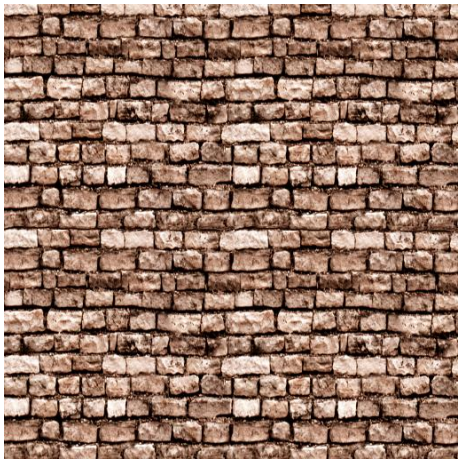
Figure A.2: Data flow between the involved components when executing the trajectory planning methods for 3D reconstruction

B Evaluated Textures

In figure B.1 the textures, which were used in this research, and their names, can be found. Those were taken either from opengameart.org (Open Game Art (2019)) or Dmitriy Chugai (2019) and were released for personal and commercial use.

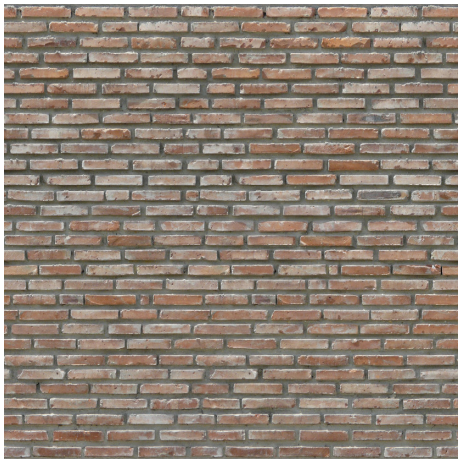
(a) clean concrete

(b) concrete



(c) factory rock wall

(d) old scratched metal



(e) red grey bricks

(f) worn aluminum

Figure B.1: Different textures, which were used in this research, with their names

Bibliography

ANYbotics (2019), rqt Multiplot.
https://github.com/anybotics/rqt_multiplot_plugin

appliedAI Initiative (2019), A ROS implementation of ORB-SLAM2.
https://github.com/appliedAI-Initiative/orb_slam_2_ros

Avola, D., G. L. Foresti, N. Martinel, C. Micheloni, D. Pannone and C. Piciarelli (2017), Aerial video surveillance system for small-scale UAV environment monitoring, in *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, IEEE, pp. 1–6, ISBN 978-1-5386-2939-0, doi:10.1109/AVSS.2017.8078523.
http://ieeexplore.ieee.org/document/8078523/

Cao, Z. L., Y. Huang and E. L. Hall (1988), Region filling operations with random obstacle avoidance for mobile robots, **vol. 5**, no.2, pp. 87–102, ISSN 10974563, doi:10.1002/rob.4620050202.
http://doi.wiley.com/10.1002/rob.4620050202

Cheng, P., J. Keller and V. Kumar (2008), Time-optimal UAV trajectory planning for 3D urban structure coverage, in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, IEEE, pp. 2750–2757, ISBN 9781424420582, ISSN 2153-0858, doi:10.1109/IROS.2008.4650988.
http://ieeexplore.ieee.org/document/4650988/

Chung, W. K., L.-C. Fu and T. Kröger (2016), Motion Control, in *Springer Handbook of Robotics*, Springer International Publishing, pp. 163–194, doi:10.1007/978-3-319-32552-1_8.
http://link.springer.com/10.1007/978-3-319-32552-1_8

Dmitriy Chugai (2019), Texturelib.
http://texturelib.com/

Fallon, M., H. Johannsson, M. Kaess and J. J. Leonard (2013), The MIT Stata Center dataset, **vol. 32**, no.14, pp. 1695–1699, ISSN 0278-3649, doi:10.1177/0278364913509035.
http://journals.sagepub.com/doi/10.1177/0278364913509035

Furrer, F., M. Burri, M. Achtelik and R. Siegwart (2016), *Robot Operating System (ROS): The Complete Reference (Volume 1)*, Springer International Publishing, Cham, chapter RotorS—A Modular Gazebo MAV Simulator Framework, pp. 595–625, ISBN 978-3-319-26054-9, doi:10.1007/978-3-319-26054-9_23.
http://dx.doi.org/10.1007/978-3-319-26054-9_23

Galceran, E. and M. Carreras (2013a), A survey on coverage path planning for robotics, **vol. 61**, no.12, pp. 1258–1276, ISSN 0921-8890, doi:10.1016/J.ROBOT.2013.09.004.
https://www.sciencedirect.com/science/article/pii/S092188901300167X?via%3Dihub

Galceran, E. and M. Carreras (2013b), Planning coverage paths on bathymetric maps for in-detail inspection of the ocean floor, in *Proceedings - IEEE International Conference on Robotics and Automation*, IEEE, pp. 4159–4164, ISBN 9781467356411, ISSN 10504729, doi:10.1109/ICRA.2013.6631164.
http://ieeexplore.ieee.org/document/6631164/

Gaspar, A. R., A. Nunes, A. Pinto and A. Matos (2018), Comparative Study of Visual Odometry and SLAM Techniques, in *Advances in Intelligent Systems and Computing*, volume 694, Springer, Cham, pp. 463–474, ISBN 9783319708355, ISSN 21945357, doi:10.1007/978-3-319-70836-2_38.
http://link.springer.com/10.1007/978-3-319-70836-2_38

Geiger, A., P. Lenz and R. Urtasun (2012), Are we ready for autonomous driving? The KITTI vision benchmark suite, in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, pp. 3354–3361, ISBN 978-1-4673-1228-8, doi:10.1109/CVPR.2012.6248074.
http://ieeexplore.ieee.org/document/6248074/

Giubilato, R., S. Chiodini, M. Pertile and S. Debei (2018), An experimental comparison of ROS-compatible stereo visual SLAM methods for planetary rovers, in *5th IEEE International Workshop on Metrology for AeroSpace, MetroAeroSpace 2018 - Proceedings*, IEEE, pp. 386–391, ISBN 9781538624746, doi:10.1109/MetroAeroSpace.2018.8453534.
https://ieeexplore.ieee.org/document/8453534/

González-Jorge, H., J. Martínez-Sánchez, M. Bueno, Arias, Pedor, H. González-Jorge, J. Martínez-Sánchez, M. Bueno, Arias and Pedor (2017), Unmanned Aerial Systems for Civil Applications: A Review, **vol. 1**, no.1, p. 2, ISSN 2504-446X, doi:10.3390/drones1010002.
http://www.mdpi.com/2504-446X/1/1/2

Ham, Y., K. K. Han, J. J. Lin and M. Golparvar-Fard (2016), Visual monitoring of civil infrastructure systems via camera-equipped Unmanned Aerial Vehicles (UAVs): a review of related works, **vol. 4**, no.1, p. 1, ISSN 2213-7459, doi:10.1186/s40327-015-0029-z.
http://www.viejournal.com/content/4/1/1

Hoffmann, H., R. Jensen, A. Thomsen, H. Nieto, J. Rasmussen and T. Friborg (2016), Crop water stress maps for an entire growing season from visible and thermal UAV imagery, **vol. 13**, no.24, pp. 6545–6563, ISSN 1726-4189, doi:10.5194/bg-13-6545-2016.
https://www.biogeosciences.net/13/6545/2016/

Labbé, M. and F. Michaud (2018), RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation, *Journal of Field Robotics*, ISSN 15564959, doi:10.1002/rob.21831.
http://doi.wiley.com/10.1002/rob.21831

Mansouri, S. S., C. Kanellakis, E. Fresk, D. Kominiak and G. Nikolakopoulos (2018), Cooperative coverage path planning for visual inspection, *Control Engineering Practice*, **vol. 74**, pp. 118–131, ISSN 09670661, doi:10.1016/j.conengprac.2018.03.002.
https://www.sciencedirect.com/science/article/pii/S0967066118300315

Mattar, R. A., R. Kalai, R. A. Mattar and R. Kalai (2018), Development of a Wall-Sticking Drone for Non-Destructive Ultrasonic and Corrosion Testing, **vol. 2**, no.1, p. 8, ISSN 2504-446X, doi:10.3390/drones2010008.
http://www.mdpi.com/2504-446X/2/1/8

Mur-Artal, R., J. M. Montiel and J. D. Tardos (2015), ORB-SLAM: A Versatile and Accurate Monocular SLAM System, **vol. 31**, no.5, pp. 1147–1163, ISSN 15523098, doi:10.1109/TRO.2015.2463671.
https://ieeexplore.ieee.org/document/7219438/

Mur-Artal, R. and J. D. Tardos (2017), ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras, **vol. 33**, no.5, pp. 1255–1262, ISSN 1552-3098, doi:10.1109/TRO.2017.2705103.
http://ieeexplore.ieee.org/document/7946260/

Na, W. S. and J. Baek (2016), Impedance-Based Non-Destructive Testing Method Combined with Unmanned Aerial Vehicle for Structural Health Monitoring of Civil Infrastructures, **vol. 7**, no.1, p. 15, ISSN 2076-3417, doi:10.3390/app7010015.
http://www.mdpi.com/2076-3417/7/1/15

Open Game Art (2019), opengameart.org.
https://opengameart.org/

Open Robotics (2019a), Gazebo simulator.
http://gazebosim.org/

Open Robotics (2019b), Robot Operating System.
http://www.ros.org/

Open Robotics (2019c), rviz - 3D visualizer for the Robot Operating System (ROS) framework.
https://github.com/ros-visualization/rviz

Pire, T., T. Fischer, G. Castro, P. De Cristóforis, J. Civera and J. Jacobo Berlles (2017), S-PTAM: Stereo Parallel Tracking and Mapping, *Robotics and Autonomous Systems*, **vol. 93**, pp. 27–42, ISSN 09218890, doi:10.1016/j.robot.2017.03.019.
https://linkinghub.elsevier.com/retrieve/pii/S0921889015302955

Rashad, R., J. B. Engelen and S. Stramigioli (2019), Energy tank-based wrench/impedance control of a fully-actuated hexarotor: A geometric port-hamiltonian approach, in *IEEE International Conference on Robotics and Automation (ICRA)*.

Rublee, E., V. Rabaud, K. Konolige and G. Bradski (2011), ORB: An efficient alternative to SIFT or SURF, in *Proceedings of the IEEE International Conference on Computer Vision*, IEEE, pp. 2564–2571, ISBN 9781457711015, doi:10.1109/ICCV.2011.6126544.
http://ieeexplore.ieee.org/document/6126544/

Schroeder, D. (2012), U.S. Department of Energy - Photo of the Week: July 20, 2012.
https://www.flickr.com/photos/departmentofenergy/7652216320/

Sirmaçek, B., R. Rashad and P. Radl (2019), Autonomous UAV-Based 3D-Reconstruction of Structures for Aerial Physical Interaction, *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, **vol. XLII-2/W13**, pp. 601–605, doi:10.5194/isprs-archives-XLII-2-W13-601-2019.
https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLII-2-W13/601/2019/

Smith, M., I. Baldwin, W. Churchill, R. Paul and P. Newman (2009), The New College Vision and Laser Data Set, **vol. 28**, no.5, pp. 595–599, ISSN 0278-3649, doi:10.1177/0278364909103911.
http://journals.sagepub.com/doi/10.1177/0278364909103911

Stachniss, C., J. J. Leonard and S. Thrun (2016), Simultaneous Localization and Mapping, in *Springer Handbook of Robotics*, Springer International Publishing, pp. 1153–1176, doi:10.1007/978-3-319-32552-1_46.
http://link.springer.com/10.1007/978-3-319-32552-1_46

Teixeira, L. and M. Chli (2016), Real-time mesh-based scene estimation for aerial inspection, in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, pp. 4863–4869, ISBN 978-1-5090-3762-9, doi:10.1109/IROS.2016.7759714.
http://ieeexplore.ieee.org/document/7759714/

Zaheer, Z., A. Usmani, E. Khan and M. A. Qadeer (2016), Aerial surveillance system using UAV, in *2016 Thirteenth International Conference on Wireless and Optical Communications Networks (WOCN)*, IEEE, pp. 1–7, ISBN 978-1-4673-8975-4, doi:10.1109/WOCN.2016.7759885.
http://ieeexplore.ieee.org/document/7759885/