
# Autonomous Coastline Exploration with a Focus on Reef Structures

by

**Naomi Pentrel & Denis Rochau**

Advanced Robotics

Prof. Dr. Andreas Birk

Date of Submission: 18th May 2015

---

Jacobs University Bremen - School of Engineering and Science

## Abstract

Mappings of underwater reef structures are needed in different fields. Regardless of why we want to explore underwater reefs, when exploring them we need a strategy to map them. The strategy we use for mapping the reef should accomplish two things: it should collect the data for the whole area we are interested in and it should keep our robot safe while it explores the area.

The strategy we will explore in this paper goes along the reef in an iterative manner as it creates the map. As such it starts at a position on the top right of the map and works its way down. When it senses the ground or has gone far enough according to our specifications, it will move horizontally to an unexplored piece of the reef and go back up. After reaching the initial height it will again move horizontally to an unexplored piece of the reef and continue our algorithm.

To get a good mapping of the reef and keep the robot safe at the same time we have to optimize the above algorithm. The detailed algorithm and the suggested optimizations will be explored in this paper. The result will be an algorithm that allows the sequential mapping of 3D structures underground.

# Contents

**1 Introduction** **1**

**2 Related Work** **2**

**3 Preliminaries** **3**
3.1 ROS & Octomap . . . . . 3
3.2 Robot Overview . . . . . 3

**4 Towards an Optimal Scanning Algorithm** **4**
4.1 Distance Between Robot and Reef . . . . . 4
4.2 Improving the Value of scanDistance . . . . . 6
4.3 Rotations . . . . . 7

**5 Limitations and Possible Extensions** **9**
5.1 Obstacle avoidance . . . . . 9
5.2 Map Creation . . . . . 9
5.3 Movement . . . . . 10

**6 Conclusion** **10**

*At this point, we would like to thank Prof. Dr. Birk and Ravi Rathnam for helping and supporting us during this research project.*

# 1. Introduction

When mapping underwater reef structures we care about the completeness of the map, as well as the optimization of the path the robot takes as it scans the reef. To achieve this we will start with a very basic algorithm. As we test this algorithm on a test reef, we will eliminate errors and improve the algorithm as needed.

We will be exploring the reef shown in figure 4 as a test. The idea of the basic algorithm that we will be using is shown in figure 2. The robot starts in its starting position as shown in figure 2. From there it will start scanning and move downwards until it either reaches the ground or a predefined *maxDepth*. Once we reach that point the robot will move to the right for a predefined distance *d*. At that point the robot goes back up until it reaches the predefined startDepth where it will go to the right for a predefined distance *d*. This sequence is repeated until we reach the maxWidth.

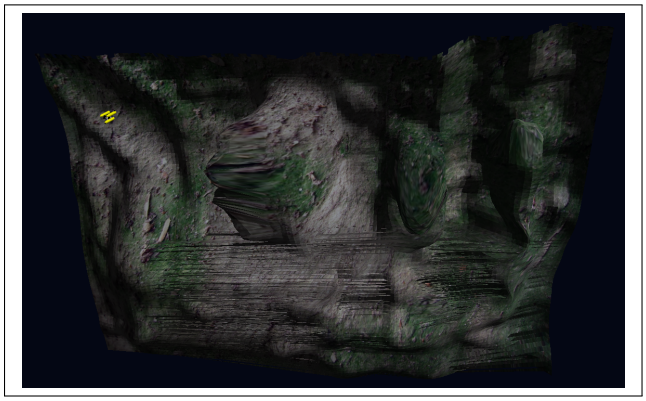

**Figure 1:** *Underwater Reef*

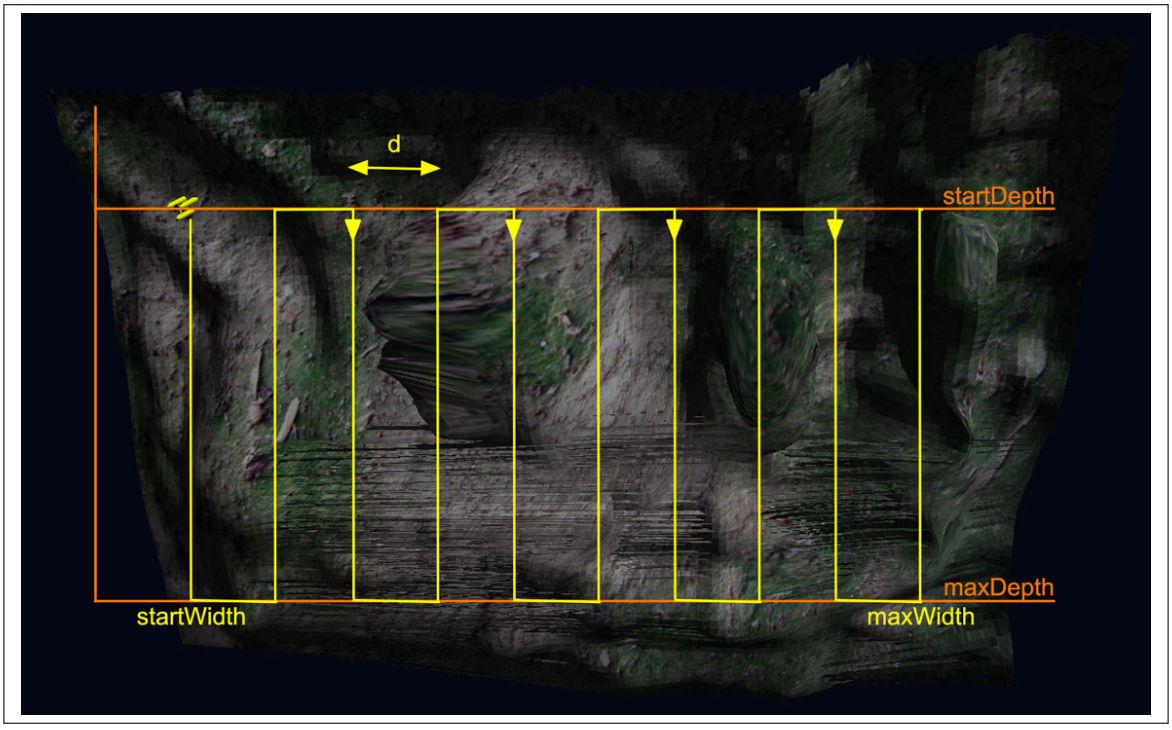


**Figure 2:** *Basic Algorithm*

Obviously the above algorithm only illustrates the basic idea. In reality, we have to also make sure the robot can actually move the way the algorithm wants it to move. Additionally, the robot should actually not be too far away from the reef so that it can scan the reef. In this paper we will therefore start from the basic algorithm discussed above and incrementally optimize until it successfully scans the reef.

## 2. Related Work

The mapping of underwater structures is a common task in underwater robotics. Yet, only a few papers deal with the actual algorithm of mapping structures which is used to guide robots during their exploration. Therefore we could not find much previous work, especially none which deals with guidance algorithms for the mapping of a horizontal structure via an autonomous underwater robot.

One paper that deals with an actual guiding algorithm is *A behavior-based architecture for autonomous underwater exploration* [RWDW02]. There the authors use a behavior-based control architecture to safely guide a robot through a reef. Their robot follows a rope on the sea floor by utilizing a camera to get optical input. The rope acts as a tool for collision avoidance and path planning as it can be safely laid out by humans beforehand.

In another work, *3-D terrain covering and map building algorithm for an AUV* [LCLL09], the authors deal with the guidance and the mapping of an ocean floor with multiple islands or underwater mountains around. The approach they use is to explore the whole space by iteratively exploring a plane for each depth level. The situation and the approach used are drastically different from what we try to do as we try to scan horizontal structures and we do this using an improved interative approach.

There are many other papers that deal with cooperative exploration of multiple robots and the path planning for those robots. This does not apply well to our problem though, as we are only dealing with one robot and do not have to account for problems and challenges like the optimal division of work [WZ12]. Most other papers which deal with actual guidance algorithms do so in combination with related mapping techniques like SLAM (e.g. [MWBDW02]).

# 3. Preliminaries

## 3.1. ROS & Octomap

For this research project we will make use of the Robot Operating System (ROS). ROS is a collection of frameworks for the development of robot control software. It provides us with many packages that implement functionalities like point cloud publishing and the simulation of different sensors. Within ROS, this project will use the octomap_server [HWB+13].

The octomap_server works by listening to a 3D point cloud produced by one or more sensors and then combining those point cloud into an OctoMap. An OctoMap is a 3D map based on an underlying octree data structure. This OctoMap is then distributed to all nodes which are subscribed to the octomap server topics, in a compact binary format. Thus we can build and save 3D OctoMaps.

## 3.2. Robot Overview

The robot we will be using in this project has the following basic capabilities:

- movement along the x, y, and z axes,
- rotation around the x, y, and z axes,
- panning and tilting of sensor,
- mapping functionality through range sensor,

Using these basic capabilities we are able to map sample reefs.

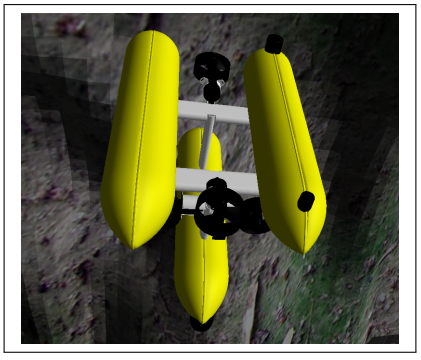

**Figure 3:** *Robot*

# 4. Towards an Optimal Scanning Algorithm

Looking back to the initial idea of the algorithm which we illustrated in section 1, we will now explore how to improve on this algorithm [PR]. First, we will ensure that the robot always keeps a minimum distance to the reef while also not being too far away from the reef.

## 4.1. Distance Between Robot and Reef

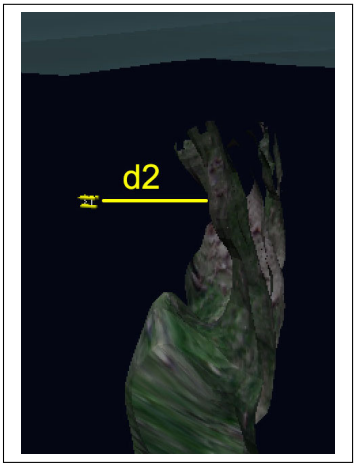


**Figure 4:** *Reef Sideways*

If the robot starts in a random position, it is in our interest to get close enough to the reef to be able to scan the reef. Therefore we will move in the direction of the reef. To do this we use the following logic to check whether the robot is far enough away from the cliff to not crash but close enough to scan the cliff (see figure 5). Ideally the robot should always stay at exactly the *minimumDistanceValue* $\pm$ *offset*. The *scanDistance* is the scanned point of the reef that is closest to the robot.

```
void ExploreAlgorithm::controlDistanceToCliff() {
    if(minimumDistanceValue > scanDistance) {
        moveTowardsCliff();
    } else if(minimumDistanceValue < (scanDistance - offset)) {
        moveAwayFromCliff();
    }
}
```

**Figure 5:** *Pseudocode Snippet to Show how to Control the Distance to the Cliff*

The robot does not start to descend until it is within the specified distance away from the reef. Once it starts to descend it will continuously check if it is still within the specified distance. If this is not the case it will move horizontally either towards or away from the cliff to regain the specified distance using the logic illustrated in figure 5. When this is achieved it will continue its descend.

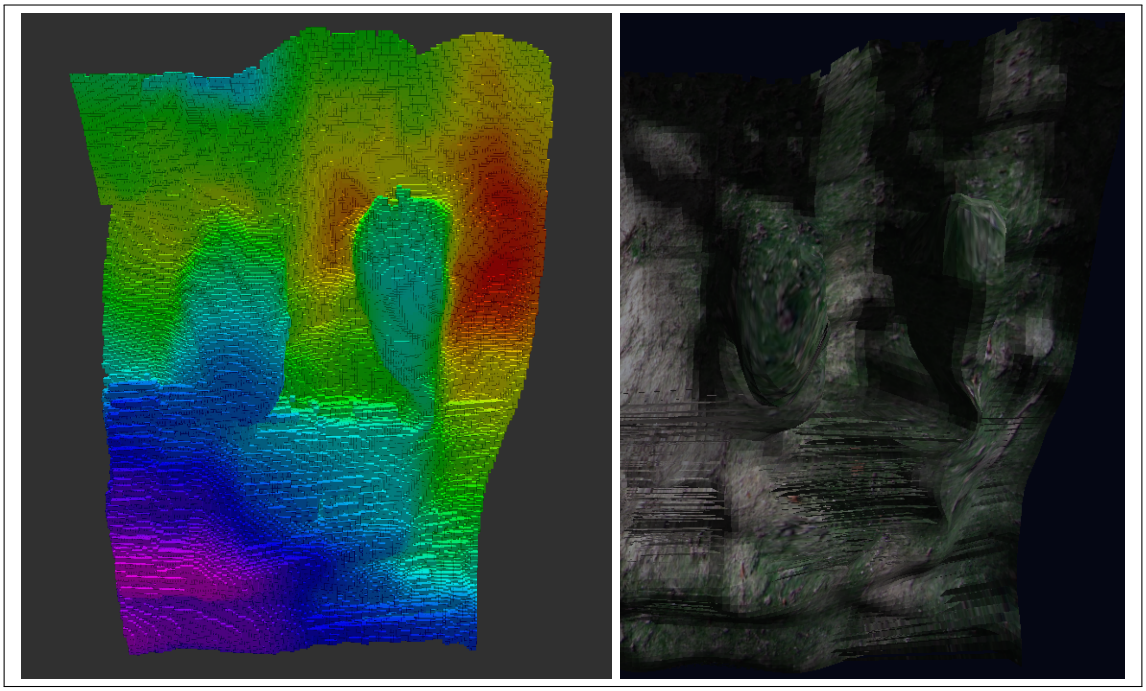

**Figure 6:** *Mapping using the Distance Algorithm*

Overall this approach already provides us with good results as visible in figure 6. However, using this logic, the robot may run into issues because it does not have sensors above, below, or behind. This will not be covered in this paper but we will discuss this in section 5 and elaborate how it could be done in **??**. Apart from running into objects, the robot can also fail to scan all areas due to the fact that we use the closest point in our field of vision to determine where we should be (see figure 5). In the case of spikes some areas may thus remain unscanned as can be seen in figure 7. The latter problem will be discussed in the next subsection.

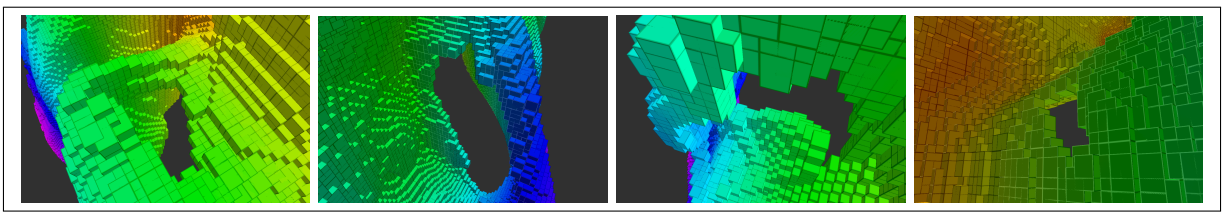

**Figure 7:** *Mapping Problems using the Distance Algorithm*

## 4.2. Improving the Value of scanDistance

In the previous section we have identified that one problem with our algorithm is that the value of *scanDistance* might make us not explore areas because of spikes. While we need to make sure that the robot does not crash into anything, we still want to maximize what we can scan. Therefore, we will calculate the value *scanDistance* not of the full field of vision of the robot but rather of a part of it that is as big as the robot.

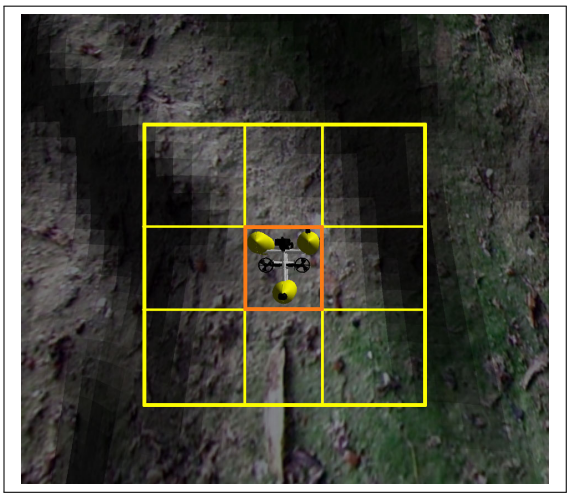

**Figure 8:** *Robot's Field of Vision*

Using the subpart of the field of vision of the robot that is as big as the robot ensures that we can move as close as possible to the reef while still not crashing into anything. This is illustrated in figure 8. In subsection 4.1, the robot would take the whole field of vision illustrated by the outer yellow rectangle. In the new approach we would like to only use the smaller orange rectangle.

```
double DistanceFinder::imageCb(const sensor_msgs::ImageConstPtr& msg) {
  cv_bridge::CvImageConstPtr cv_ptr = getImage(msg, msg->encoding);
  double scanDistance = std::numeric_limits<double>::max();
  int startRobotFieldWidth = (fieldOfVisionWidth - robotWidth)/2;
  int endRobotFieldWidth = startRobotField + robotWidth;
  int startRobotFieldHeight = (fieldOfVisionHeight - robotHeight)/2;
  int endRobotFieldHeight = startRobotFieldHeight + robotWidth;

  for(int i = startRobotFieldWidth; i < endRobotFieldWidth; i++)
    for(int j = startRobotFieldHeight; j < endRobotFieldHeight; j++)
      if(cv_ptr->image.at<float>(i,j) < scanDistance)
        scanDistance = cv_ptr->image.at<float>(i,j);
  return scanDistance;
}
```

**Figure 9:** *Pseudocode Snippet to Show how to Control the Distance to the Cliff*

The code in figure 9 will return the correct *scanDistance* for the area of the robot robotField. This value will now be used in the code which we use to keep the correct distance to the reef (see figure 5). Using this improved version of the algorithm, the newly scanned map is better. The overall scanned map can be see in figure 10.

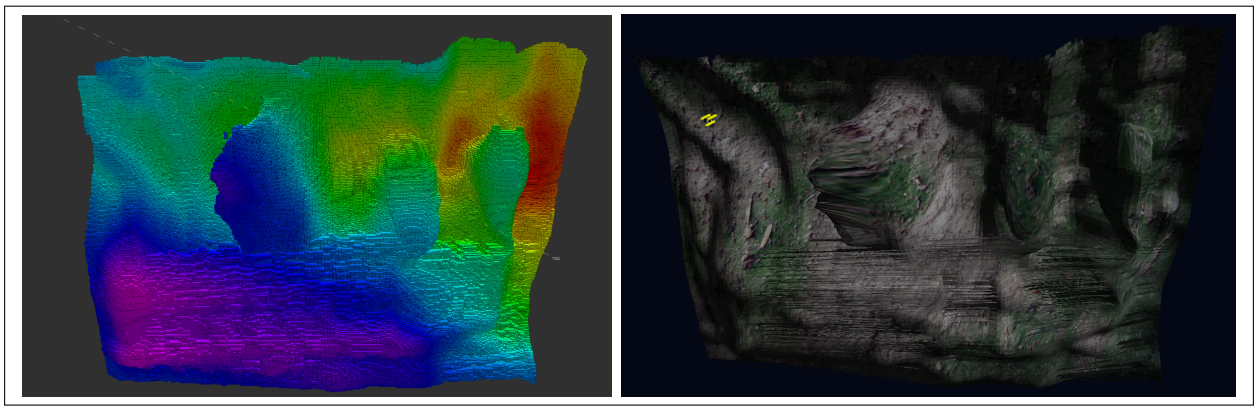

**Figure 10:** *Mapping using the Distance Algorithm*

When examining the map closer, we again look for holes in the map. Specifically, we will again look at the places where we found holes in the map before (see figure 7). As can be seen in figure 14, one hole is completely gone. The three other holes are slightly reduced but not yet fully mapped. We will tackle them in the next section.

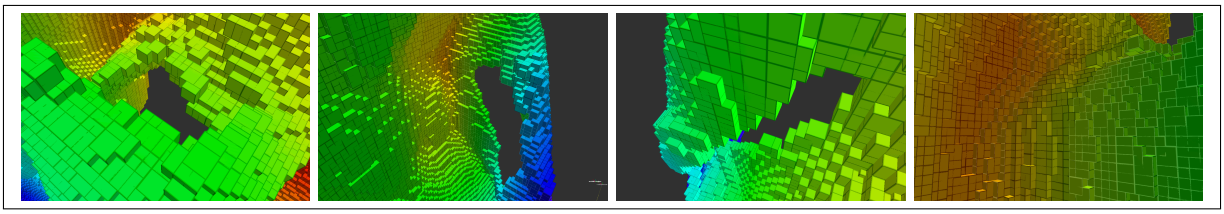

**Figure 11:** *Mapping Problems using the Distance Algorithm*

## 4.3. Rotations

The remaining holes are mostly on the sides of reef structures. Therefore we want to rotate our robot every few seconds around the z axis. Thus we scan the sides of the reef structures as well. The approach we use could also be mimicked by moving the sensor but we chose to rotate the whole robot as can be seen in the code in figure 12.

```
void ExploreAlgorithm::rotateAround() {
    tf::Quaternion q = this->transform.getRotation();
    tf::Matrix3x3 m(q);
    this->odom.twist.twist.angular.z = 1;
    double yaw = getCurrentYaw(m)
    yaw = yaw + PI;
    if (fullyRotated(yaw)) {
    stopRotating();
    }
    this->position_pub.publish(this->odom);
}
```

**Figure 12:** *Pseudocode Snippet to Show how to Control the Distance to the Cliff*

The code turns the robot 360 degrees, hence once around its z-axis. This ensures that the sides of the reef structures are scanned as well. As can be seen in figure 13, this creates a full mapping of the area. When looking at the places where we previously had holes (see figure 14), we notice that only one hole is still present. We have therefore successfully covered the other two. The remaining hole could be removed if we were to change the pitch of our robot.[1]

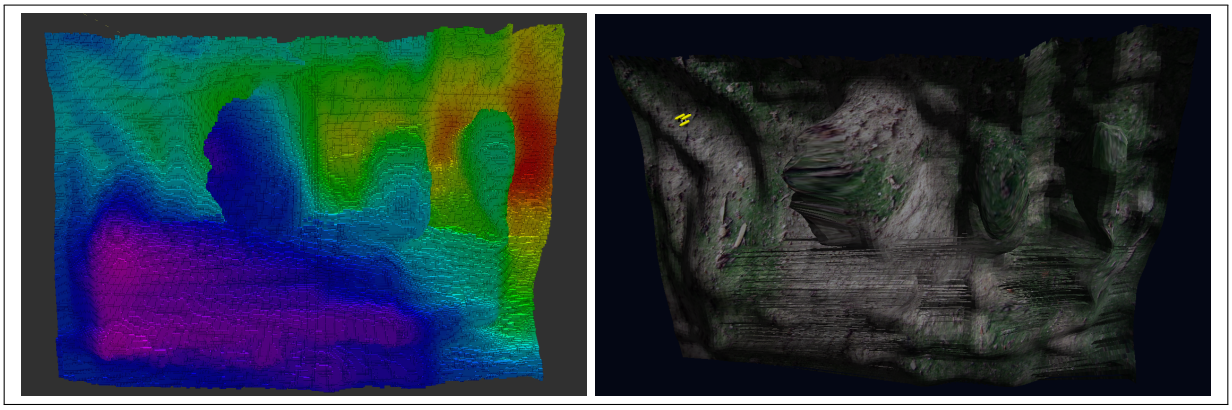

**Figure 13:** *Mapping using the Distance Algorithm with Rotation*

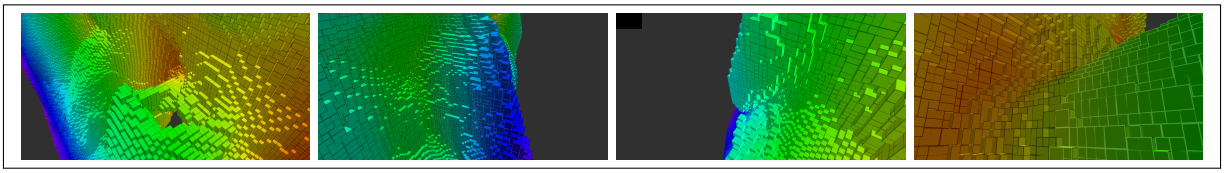

**Figure 14:** *Mapping Problems using the Distance Algorithm with Rotation*

[1] To see a condensed video of the mapping see http://youtu.be/LIdYnKuljzE

# 5. Limitations and Possible Extensions

Due to time constraints, this paper is not extensive enough to cover all optimization possibilities. This section will therefore point out limitations of our algorithm and ideas on how to implement these.

## 5.1. Obstacle avoidance

Using the current algorithm, the robot might still run into some obstacles. If the robot for example navigates into a hole, it will not realize that it will crash into the ground. If there is something behind the robot and the robot tries to go backwards it will also not realize this. There exist different possible solutions for this problem. We could, for instance, add more sensors to the robot and use them to implement a behavior-based collision avoidance algorithm similar to our approach of keeping the distance to the reef. The advantage of this approach would be that it is easy to implement and we would additionally be able to react to dynamic objects in the environment like fish.

Since adding sensors would be cost intensive, we ideally have to move the robot's sensor or the robot itself to create more information about its surroundings. We could then query the OctoMap around it by casting rays in our movement direction and then check if we have information about the area in which we want to move. If we already have mapped the area in that direction, we can use a behavior-based approach to keep a distance to objects. If we have no information available, we can rotate our sensor or robot towards the direction of movement and obtain the necessary information. Afterwards we continue our algorithm and use a behavior-based approach to keep a distance to any newly scanned objects. A clear disadvantage of this is that dynamic objects like fish might confuse our robot.

## 5.2. Map Creation

Under the current algorithm our map creation procedure is not optimal. The robot, for example, rotates every few seconds to obtain more data, which costs us time and would cost a real world robot a considerable amount of energy. It would therefore be advisable to actually rotate only the sensor when possible and only rotate the whole robot we actually need to do so in order to obtain more information. The idea here is to create a list of unscanned and incomplete parts by querying the OctoMap. Afterwards we could take this set and calculate a good movement and rotation path for our robot so that it can scan

those parts. Another way we can improve our map creation is to add more sensors to the robot. As before this approach is easy but relatively expensive.

### 5.3. Movement

The robot moves in a stair-like format. It, for example, moves backwards, down, backwards, down, etc. Ideally the robot should move in a seamless motion and use interpolation to combine the knowledge of having to move backwards and down. Moreover we also could include our obtained OctoMap to create a reference path along which we want to move. Another point to mention is that we disregard how an underwater robot would behave in real life. For a real life robot, the algorithm would need to be changed according to the limitations of the real world movement capabilities of the robot.

One prominent example for is apparent in our approach to moving mostly vertically instead of horizontally along the reef. Most underwater robots can move more effective horizontally and it is therefore a good idea to move sideways if possible. Nevertheless this could be implemented by simply changing some variables in our code.

## 6. Conclusion

In this research project, we were trying to create a mapping of a given underwater reef. Using a robot, we began with a very simple algorithm that explores a rectangle in a sequential fashion. In the next step we ensured that the robot would always keep a certain distance to the reef but also not be too far away from the reef.

At this point in time the created mapping was already very good but it contained a few holes which we tried to fix in the next step. Therefore we improved the calculation of the distance the robot has to keep to the reef by not using the robots whole field of vision to compute how far away it needs to be but rather by only using a smaller fraction of its vision, namely the part where the robot would move if it were to move forwards. Thus we ensured that the robot can get closer to the reef even if there are spikes or other structures around that would otherwise deter the robot from going closer.

Using this improved algorithm, the robot was indeed able to create a better mapping and close up one of the holes from the previous mapping while reducing the size of the holes

in the other cases. These other cases were at spots on the side of elevated reef structures. To further improve our algorithm, we therefore added horizontal rotational movements to the algorithm in order to see the sides of the structures around us. This filled most of the remaining holes. The only hole that remained in the map is on the top of an elevated reef structure. To remove it, we would need to add vertical rotations as well.

Unfortunately, this research could not be tested in a real life scenario, and we can therefore not make any claims about its real-life performance. We strongly recommend creating a better algorithm for obstacle avoidance before testing this algorithm in a real life scenario. We also would suggest that we develop an algorithm to ensure that we only rotate the robot when absolutely necessary which then predominantly only rotates the sensors.

In conclusion, the created algorithm successfully creates a mapping of a reef structure. As pointed out in section 5, there are some improvements that could be made to make the mapping even better. As is the mapping is already very good and almost complete. Depending on the requirements for the level of completeness of the mapping, this research might already be enough to give satisfactory results in a real-life scenario.

## References

[HWB+13] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013. Software available at `http://octomap.github.com`.

[LCLL09] Tae-Seok Lee, Jeong-Sik Choi, Jeong-Hee Lee, and Beom-Hee Lee. 3-d terrain covering and map building algorithm for an auv. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 4420–4425. IEEE, 2009.

[MWBDW02] Alexei A Makarenko, Stefan B Williams, Frederic Bourgault, and Hugh F Durrant-Whyte. An experiment in integrated exploration. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 1, pages 534–539. IEEE, 2002.

[PR] Naomi Pentrel and Denis Rochau. Coastline explorer. https://github.com/npentrel/CoastLineExplorer.

[RWDW02] Julio Rosenblatt, Stefan Williams, and Hugh Durrant-Whyte. A behavior-based architecture for autonomous underwater exploration. *Information Sciences*, 145(1):69–87, 2002.

[WZ12] Wencen Wu and Fumin Zhang. Robust cooperative exploration with a switching strategy. *Robotics, IEEE Transactions on*, 28(4):828–839, 2012.