Article

Reactive Control for Collision Evasion with Extended Obstacles

Jonghoek Kim

Electronic and Electrical Department, Sungkyunkwan University, Suwon 03063, Korea; jonghoek@gmail.com

Abstract: Evading collisions in three-dimensional underwater environments is critical in exploration of an Autonomous Underwater Vehicle (AUV). In underwater environments, AUV measures an obstacle surface by utilizing a three-dimensional active sonar. This article addresses reactive collision evasion control by considering extended obstacles. Here, an extended obstacle is an arbitrary obstacle that can generate any number of measurements and not a point target generating at most one measurement. Considering 3D environments, our manuscript considers collision evasion with both moving obstacles and static obstacles. The proposed reactive collision evasion controllers are developed by considering hardware limits, such as the maximum speed or acceleration limit of an AUV. We further address how to make an AUV move towards a goal, while avoiding collision with extended obstacles. As far as we know, the proposed collision evasion controllers are novel in handling collision avoidance with an extended obstacle, in the case where an AUV measures 3D-obstacle boundaries by utilizing sonar sensors. The effectiveness of the proposed controllers is demonstrated by MATLAB simulations.

Keywords: reactive collision evasion; underwater environment; maximum acceleration; extended obstacle; active sonar sensor

1. Introduction

Recently, autonomous underwater vehicles (AUVs) have been utilized in many applications, such as underwater exploration. Collision evasion is crucial for safe maneuvers of AUVs. In underwater environments, electromagnetic signals dissipate quickly. Thus, an AUV uses sonar sensors with a limited range in order to detect nearby underwater objects. This article addresses reactive controls that do not require path planning ahead of time. The reactive collision evasion control is developed by considering limited sensors and the hardware limits of AUV.

We consider the case where an AUV uses active sonar sensors to sense its surroundings. We assume that the AUV has sonar sensors for calculating the point cloud of underwater objects [1]. The authors of [1] proposed a method to reconstruct an object’s three-dimensional geometry based on sonar images captured while an AUV moves over the object. By reconstructing the 3D shape of the object, [1] further addressed a method for classifying the object using neural networks. In our paper, point clouds of an object are utilized for collision evasion control.

In our manuscript, the AUV can calculate three-dimensional points, termed objectPoints, on object surfaces. The proposed reactive controllers make AUV evade collision with every objectPoint.

The AUV in our article uses sonar sensors to calculate the point cloud of underwater objects [1]. AUV measures an obstacle surface using sonar sensors, followed by estimating the obstacle’s position and velocity using the information of objectPoints. By measuring the difference between two clouds of objectPoints measured at distinct sampling indexes, the extended obstacle’s velocity can be estimated. In order to estimate the velocity (speed and orientation) of objectPoints on an extended obstacle, AUV utilizes iterative closest point (ICP) algorithms [2–7]. In order to limit the influence of outliers, we use the robust
ICP algorithm in [6]. Moreover, for efficient search, this paper uses k-D tree for matching processes [8].

There are many papers on developing collision evasion controls in two-dimensional environments [9–12]. In [13], the obstacle avoidance research of wave glider in 2D marine environment was conducted. Considering 2D environments, artificial potential field (APF) has been widely used for avoid collision between a robot and obstacles [13,14]. APF is generated by the addition of the attraction potential field (generated by the goal) and the repulsive potential field (generated by obstacles).

As far as we know, APF is not provably complete in achieving collision avoidance. For instance, consider the case where the robot needs to move into a tunnel for reaching its goal. Due to the repulsive field generated by obstacles at the tunnel entrance, the robot has difficulty in maneuvering through the tunnel. Moreover, APF was not applied for collision avoidance in 3D environments. Moreover, APF is not suitable for avoiding moving obstacles, since the velocity of an obstacle was not considered in APF. Considering 3D environments, our article tackles collision evasion with both maneuvering obstacles and static obstacles.

Velocity obstacle (VO) methods were utilized for collision evasion with obstacles, which may move [9,11,15–17]. VO methods are computationally efficient and, thus, appropriate for mobile robots [18]. VO methods are provably complete in achieving collision avoidance, as long as an obstacle maintains its velocity for all future times [15]. VO methods assume that an obstacle moves with a constant velocity for all future times. However, this is not a valid assumption in practice.

In our manuscript, the proposed reactive collision avoidance controls are developed and inspired by VO methods. We address 3D reactive collision avoidance controls, assuming that an obstacle moves with a constant velocity within \( k_p \) sampling-indexes in the future. At each sampling-index, an obstacle’s pose (position and velocity) is estimated using sonar sensor measurements, and we run collision avoidance controls at every sampling-index. Our study proves that as long as the obstacle velocity (speed and orientation) is estimated correctly using ICP algorithms, collision avoidance is assured within \( k_p \) sampling indexes in the future.

Considering 3D environments, our manuscript handles collision evasion with both moving obstacles and static obstacles. Several papers [19–21] considered collision evasion in three-dimensional environments. The authors of [20] presented a collision avoidance algorithm based on potential fields for fixed-wing unmanned aerial vehicles (UAVs) with constrained field-of-view (FOV) sensors such as cameras. The authors of [22] explored 3D path planning for UAVs in 3D point-cloud environments. The approach in [22] searched for obstacle-free and smooth paths by analyzing a point cloud of the target environment, using a modified rapidly exploring random tree (RRT)-based path planning algorithm. However, collision avoidance controllers in [20–22] are not suitable for evading a fast moving obstacle, since they did not consider the velocity of a maneuvering obstacle.

References [19,23–26] developed collision avoidance controllers while modeling obstacles as spheres or bounding boxes. The authors of [23] addressed a reactive algorithm for avoiding obstacles in three-dimensional space by modeling obstacles as spheres.

In our article, we do not model obstacles as spheres or bounding boxes. Our study handles collision evasion with extended obstacles (moving obstacles as well as static obstacles). Here, an extended obstacle is an arbitrary obstacle generating any number of measurements and not a point target generating at most one measurement. To the best of our knowledge, the proposed collision evasion controllers are novel, since our manuscript handles the case where an AUV measures three-dimensional obstacle surfaces by utilizing sonar sensors. In our paper, the AUV measures an obstacle surface and estimates its position and velocity using the information of objectPoints on the surface.

Considering 3D environments, our collision avoidance controls avoid moving obstacles as well as static obstacles. The proposed reactive controllers utilize the velocity information of an obstacle; hence, it is suitable for evading a fast-moving obstacle. In
practice, AUVs may need to approach its goal while avoiding obstacles. We thus address a method to make AUVs approach its goal, while avoiding collision with extended obstacles.

The proposed reactive collision-evasion controllers are developed by considering an AUV’s hardware limits, such as the maximum speed or the acceleration limit of the AUV. At each sampling-index $k$, the AUV calculates its velocity command, which allows it to approach the goal, while avoiding collision with extended obstacles (maneuvering obstacles as well as static obstacles). The velocity command is set considering hardware limits, such as the maximum speed or acceleration limit of the AUV.

Our article handles the case where AUV measures three-dimensional obstacle surfaces using sonar sensors. To the best of our knowledge, the proposed collision evasion controllers are novel, since our article tackles collision avoidance with an extended obstacle. The outperformance of the proposed evasion controllers is demonstrated by utilizing MATLAB simulations.

Section 2 discusses the preliminaries of this article. Section 3 handles assumptions and definitions of this article. Section 4 handles how to estimate the velocity (speed and orientation) of objectPoints associated with an extended obstacle. Section 5 addresses the proposed reactive collision-evasion control. Section 6 discusses how to make AUVs move towards the goal, while avoiding collision with extended obstacles. Section 7 introduces MATLAB simulations to demonstrate the performance of the proposed controls. Section 8 addresses our conclusions.

2. Preliminaries

This article utilizes two frames: an inertial frame $\{I\}$ and a body frame $\{B\}$ [27]. We address several definitions in rigid-body dynamics [27].

The origin of $\{I\}$ is a point with three axes pointing north, east, and south. $\{B\}$ is fixed to AUV’s body such that $\{B\}$ is originated at AUV’s gravity center.

In rigid-body dynamics [27], $\theta$ and $\psi$ define pitch and yaw, respectively. For notation convenience, let $c(\eta)$ represent $\cos(\eta)$. In addition, let $s(\eta)$ represent $\sin(\eta)$. Let $t(\eta)$ represent $\tan(\eta)$.

The rotation matrix representing the counterclockwise (CC) rotation of an angle $\psi$ centered at the $z$-axis in $\{B\}$ is as follows.

$$
\mathbf{r}(\psi) = \begin{pmatrix}
c(\psi) & -s(\psi) & 0 \\
s(\psi) & c(\psi) & 0 \\
0 & 0 & 1
\end{pmatrix}.
$$

(1)

The rotation matrix representing the CC rotation of an angle $\theta$ centered at the $y$-axis in $\{B\}$ is as follows.

$$
\mathbf{r}(\theta) = \begin{pmatrix}
c(\theta) & 0 & s(\theta) \\
0 & 1 & 0 \\
-s(\theta) & 0 & c(\theta)
\end{pmatrix}.
$$

(2)

The combined rotation matrix is built by multiplying (1) and (2) to obtain the following.

$$
\mathbf{r}(\psi, \theta) = \mathbf{r}(\psi)\mathbf{r}(\theta).
$$

(3)

3. Definitions and Assumptions

This article assumes that the localization of AUV is performed using AUV’s on-board sensors, such as the inertial measurement unit (IMU). The question of how to localize AUV is not within the scope of this article. Various methods [28–30] exist for the localization of an AUV.

We introduce the notations used in this paper. Let $A(v_1, v_2) = \cos(\frac{v_1 \cdot v_2}{\|v_1\|\|v_2\|})$ define the angle between two three-dimensional vectors $v_1$ and $v_2$. Moreover, $L(A \text{ and } B)$ indicates the line segment for which its two end points are $A$ and $B$, respectively.
Let $q \in \mathbb{R}^3$ present the three-dimensional coordinates of the AUV. Let $v = \dot{q} \in \mathbb{R}^3$ present the AUV’s velocity. Let $h = \frac{v}{\|v\|} \in \mathbb{R}^3$ define the orientation vector of the AUV.

We consider discrete-time systems. Considering a variable $A$, $A(k)$ indicates $A$ at sampling-index $k$. For instance, AUV at sampling-index $k$ is located at $q(k) \in \mathbb{R}^3$.

Suppose that the current sampling-index is $k$. The process model of AUV is as follows:

$$q(k + 1) = q(k) + v(k) \ast dt,$$  \hspace{1cm} (4)

where $dt$ indicates the sample duration in discrete-time systems. In (4), $v(k)$ can be considered as the velocity command (control input) of the AUV at sampling-index $k$. The motion model in (4) is commonly used in reactive collision avoidance using VO methods [9,11,15–17].

Inspired by VO methods [9,11,15–17], we search for the velocity command, $v(k)$, which satisfies collision avoidance at each sampling-index $k$. At each sampling-index $k$, the velocity command $v(k)$ is set by considering hardware limits, such as the maximum speed or the acceleration limit of the AUV. The AUV’s maximum speed is $s_{\text{max}}$, i.e., $\|v(k)\| \leq s_{\text{max}}$. The maximum acceleration rate of AUV is $a_{\text{max}}$. In other words, we have the following.

$$-a_{\text{max}} \leq \frac{(\|v(k + 1)\| - \|v(k)\|)}{dt} \leq a_{\text{max}}.$$  \hspace{1cm} (5)

In practice, the AUV turns with a bounded turn rate. Considering the bounded turn rate of the AUV, this article uses the following:

$$A(v(k), v(k + 1)) < a \ast dt$$  \hspace{1cm} (6)

at each sampling-index $k$.

At each sampling-index $k$, we calculate the velocity command $v(k)$ in (4), which allows the AUV to approach a goal, while avoiding collisions with extended obstacles (moving obstacles as well as static obstacles). Velocity command $v(k)$ in (4) can be considered as a high-level control command of the AUV. Once $v(k)$ is set at sampling-index $k$, the desired waypoint at sampling-index $k + 1$ is set as $q(k + 1)$ in (4). Then, the straight line segment from the current AUV’s position $q(k)$ to the next waypoint $q(k + 1)$ is set as the reference trajectory from sampling-index $k$ to $k + 1$. Trajectory tracking controls in [31–34] can be applied to make the AUV track the straight-line segment trajectory, even in environments with model dynamic uncertainties and the presence of external disturbances representing ocean currents and waves. Note that detailed kinematic model in AUV’s local coordinate frame is handled in the trajectory tracking controls in [31–34].

At each sampling-index $k$, we calculate velocity command $v(k)$ in (4), assuming that an obstacle moves with a constant velocity within $k_p$ sampling indexes in the future. At each sampling index, an obstacle’s pose (position and velocity) is estimated. In addition, we run collision avoidance controls at every sampling-index.

The AUV has sonar sensors to derive the point cloud of underwater objects [1]. The AUV measures three-dimensional points on obstacle surfaces utilizing three-dimensional sonar sensors. Recall that every three-dimensional point is termed objectPoint, say $o_p$. Rendering the AUV’s conjecture as an extended obstacle’s velocity (speed and orientation) using objectPoints is addressed in Section 4.

We say that the AUV and an objectPoint $o_p$ is in a near-collision state as the relative distance between them is decreases than compared to a certain constant, say $r > 0$. Here, $r > 0$ is set by considering the size of the AUV. In other words, a sphere with radius $r$ is sufficiently large to contain the entire AUV. Let $S_r(o_p)$ define the sphere with radius $r$, for which its center is at $o_p$. 

4. Conjecture the Velocity of ObjectPoints on an Extended Obstacle

To conjecture the velocity of objectPoints on an extended obstacle at sampling-index $k$, the AUV utilizes a robust ICP algorithm [6]. Let $k$ present the current sampling index. Let $S(k)$ define the set of objectPoints measured at sampling-index $k$.

Suppose that $S(k)$ is given by $S(k) = \{x_i\}, i \in \{1, \ldots, M_k\}$ Moreover, suppose that $S(k-k_d) = \{y_j\}, j \in \{1, \ldots, M_{k-k_d}\}$. Here, $k_d > 0$ is a tuning constant. This indicates that the AUV measures $M_k$ objectPoints at sampling-index $k$. In addition, the AUV measures $M_{k-k_d}$ objectPoints at sampling-index $k-k_d$. Observe that $x_i, y_j \in \mathbb{R}^3$ are three-dimensional objectPoint coordinates.

Suppose that both $M_k$ and $M_{k-k_d}$ exceed a certain threshold, say $Th_{ICP}$. Then, the ICP algorithm is applied to both $S(k)$ and $S(k-k_d)$ in order to find a velocity vector of the obstacle at sampling-index $k$.

The ICP algorithm at sampling-index $k$ estimates a rigid motion with rotation matrix $R_k$ and translation $t_k$, which minimizes the following error $E$:

$$E(R_k, t_k) = \sum_{i=1}^{M_k} w_i \| (R_k x_i + t_k - y_j^*) \|^2,$$

(7)

where $w_i$ is the weight for the $i$-th cloud point and is selected considering Huber’s robust function [6,7]. The method for selecting $w_i$ is addressed in [6,7].

Given $R_k$ and $t_k$, point $y_j^* \in S(k-k_d)$ is denoted as the optimal correspondence of $x_i$, which is the closest point to the transformed $x_i \in S(k)$.

$$j^* = \arg\min_{j, i \in \{1, \ldots, M_{k-k_d}\}} \| (R_k x_i + t_k - y_j) \|^2.$$

(8)

Given an initial transformation $(R_k, t_k)$, the ICP algorithm iteratively solves the problem by alternating between estimating the transformation with (7), and finding the closest-point matches with (8). For an efficient search, the k-D tree is utilized for the matching process [8]. This iterative process guarantees convergence to a local minimum. Once the ICP algorithm is completed at sampling-index $k$, translation $t_k$ indicates the velocity of the maneuvering obstacle at sampling-index $k$.

Note that the ICP algorithm is an iterative algorithm and, thus, requires an initial transformation $(R_k, t_k)$. We next present a method for setting an initial transformation $(R_k, t_k)$. The rotation matrix $R_k$ is initialized as a diagonal matrix. Moreover, the translation vector $t_k$ is initialized as follows:

$$t_k = \frac{\sum_{j=1}^{M_{k-k_d}} y_j}{M_{k-k_d}} - \frac{\sum_{i=1}^{M_k} x_i}{M_k}.$$

(9)

There may be a case where $(R_k, t_k)$ is not estimated correctly due to the mismatch of point pairs. We assume that the maximum speed of an obstacle is known a priori. If $\|t_k\|$ exceeds the maximum speed of an obstacle, then we set the following:

$$t_k = t_{k-1},$$

(10)

which implies that we use the translation, which was estimated at the previous sampling index.

5. Reactive Controllers for Collision Evasion
5.1. Collision Prediction

At each sampling-index $k$, we calculate the velocity command $v(k)$ in (4). Let $v$ denote the velocity command $v(k)$ for convenience.

At each sampling-index $k$, the AUV calculates $v$, assuming that the objectPoint $o_p$ maneuvers with a velocity $v_O$ for $k_p$ sampling indexes in the future. Recall that Section 4 discussed how to conjecture $v_O$ based on the sonar measurements of AUV. At each sampling-
Sensors 2022, 22, 5478

index, the position and velocity of \( o_p \) are estimated. Furthermore, one runs collision avoidance controls at every sampling index.

We discuss how to evade the case where AUV and an objectPoint, say \( o_{p} \), collide between sampling-index \( k \) and \( k + k_p \). No collision happens between sampling-index \( k \) and \( k_p \) if

\[
\| \mathbf{q}(k) - \mathbf{o}_p(k) + (\mathbf{v} - \mathbf{v}_O)u \ast dt \| > r. \tag{11}
\]

for all \( 0 \leq u \leq k_p \). As one increases \( u \) in (11) from 0 to \( k_p \), \( \mathbf{q}(k) + (\mathbf{v} - \mathbf{v}_O)u \ast dt \) generates discrete points along line segment \( L(\mathbf{q}(k), \mathbf{q}(k) + (\mathbf{v} - \mathbf{v}_O)k_p \ast dt) \).

Utilizing (11), Lemma 1 is derived. This lemma proves that as long as the obstacle velocity \( \mathbf{v}_O \) is estimated correctly using ICP algorithms in Section 4, collision avoidance between \( o_p \) and AUV is assured within \( k_p \) sampling indexes in the future.

**Lemma 1.** An objectPoint \( o_p \) maneuvers with a velocity \( \mathbf{v}_O \) for \( k_p \) sampling indexes in the future. In addition, the AUV maneuvers with a velocity \( \mathbf{v} \). Assume that \( \mathbf{q}(k) \) is outside \( S_r(\mathbf{o}_p) \) at sampling-index \( k \). The AUV does not collide with \( o_p \) for \( k_p \) sampling indexes in the future, as long as \( L(\mathbf{q}(k), \mathbf{q}(k) + (\mathbf{v} - \mathbf{v}_O)k_p \ast dt) \) does not meet \( S_r(\mathbf{o}_p) \) at sampling-index \( k \).

The velocities of an object and that of AUV are considered in collision avoidance, as presented in Lemma 1. Using Lemma 1, the AUV can predict its collision within \( k_p \) sampling indexes in the future. At each sampling-index \( k \), the AUV examines whether its velocity command \( \mathbf{v} \) meets the following non-collision requirement: For every objectPoint \( o_p \), \( L(\mathbf{q}(k), \mathbf{q}(k) + (\mathbf{v} - \mathbf{v}_O)k_p \ast dt) \) does not meet \( S_r(\mathbf{o}_p) \) at sampling-index \( k \).

If \( L(\mathbf{q}(k), \mathbf{q}(k) + (\mathbf{v} - \mathbf{v}_O)k_p \ast dt) \) does not meet \( S_r(\mathbf{o}_p) \), then the AUV utilizes \( \mathbf{v} \) as its velocity command for a single sample duration \( dt \). If \( L(\mathbf{q}(k), \mathbf{q}(k) + (\mathbf{v} - \mathbf{v}_O)k_p \ast dt) \) meets \( S_r(\mathbf{o}_p) \) for any objectPoint \( o_p \), then the AUV needs to find a non-collision velocity vector, as addressed in Section 5.2.

### 5.2. Find a Non-Collision Velocity Vector at Sampling-Index \( k + 1 \)

As the non-collision requirement is not fulfilled for at least one objectPoint, the AUV needs to find a velocity command, \( \mathbf{v} \), satisfying the non-collision requirement. Once a velocity command, which satisfies the non-collision requirement, is found, then the AUV utilizes the found velocity as its velocity command for a single sample duration \( dt \). Note that, at every sampling-index, we run collision avoidance controls.

Algorithm 1 shows how to find a non-collision velocity at sampling-index \( k + 1 \) utilizing (17). Let \( \mathbf{u} = (1, 0, 0)^T \) present the orientation of the vehicle in the body frame. Under (3), one derives the following:

\[
\mathbf{v}(k) = \mathbf{r}(\psi(k), \theta(k)) \ast \| \mathbf{v}(k) \| \ast \mathbf{u}. \tag{12}
\]

We control only the pitch and the yaw of the body in order to achieve the desired orientation. We, thus, find the body orientation, say \( \psi(k), \theta(k) \), associated with \( \mathbf{v}(k) \). Under (12), one calculates the following:

\[
\mathbf{h}(k) = (\mathbf{h}(k, 1) , \mathbf{h}(k, 2), \mathbf{h}(k, 3))^T, \tag{13}
\]

where \( \mathbf{h}(k) = \frac{\mathbf{v}(k)}{\| \mathbf{v}(k) \|} \) indicates AUV’s orientation vector at sampling-index \( k \). Furthermore, \( \mathbf{h}(k, 1) = c(\psi(k)) \ast c(\theta(k)) \), \( \mathbf{h}(k, 2) = s(\psi(k)) \ast c(\theta(k)) \), and \( \mathbf{h}(k, 3) = -s(\theta(k)) \). Here, \( \mathbf{h}(k, j) \) indicates the \( j \)-th element of \( \mathbf{h}(k) \).

Under (13), one calculates the following:

\[
\theta(k) = \text{atan2}(-\mathbf{h}(k, 3), \sqrt{\mathbf{h}(k, 1)^2 + \mathbf{h}(k, 2)^2}). \tag{14}
\]
Under (13), one calculates $\psi(k)$. If $c(\theta(k)) > 0$, then one utilizes the following.

$$\psi(k) = \text{atan2}(h(k, 2), h(k, 1)).$$  \hfill (15)

If $c(\theta(k)) < 0$, then one utilizes the following.

$$\psi(k) = \text{atan2}(-h(k, 2), -h(k, 1)).$$  \hfill (16)

Recall that the AUV’s speed is limited by $s_{\text{max}}$ and that $\|v\| \leq a_{\text{max}}$. Considering these limits, $\|v(k + 1)\|$ is limited as follows.

$$A \leq \|v(k + 1)\| \leq A + B.$$  \hfill (17)

Here, $A = \text{max}(\|v(k)\| - a_{\text{max}} * dt, s_{\text{min}})$ and $B = \text{min}(\|v(k)\| + a_{\text{max}} * dt, s_{\text{max}}) - (\|v(k)\| - a_{\text{max}} * dt)$. In addition, $s_{\text{min}}$ is the minimum speed of the AUV, which is used to avoid the case where the AUV stops moving.

Algorithm 1 finds a non-collision speed utilizing the following FOR loop: for $v_0 = A$, $v_0 = v_0 + \frac{B}{N}$, while $v_0 \leq A + B$. This FOR statement indicates that one examines a velocity vector with low speeds before examining a velocity vector with high speeds. In this manner, the AUV does not speed up abruptly during its maneuver. Note that a low speed is desirable, considering the safety of both the AUV and a moving object, such as other AUVs.

**Algorithm 1** Find a non-collision velocity command at sampling-index $k + 1$ (change yaw only)

1. The current sampling-index is $k$;
2. for $v_0 = A$, $v_0 = v_0 + \frac{B}{N}$, while $v_0 \leq A + B$ do
3. $V^1_k = R(\psi(k), \theta(k)) * v_0 * \mathbf{u}$;
4. if col($V^1_k$, objectSet) = 0; then
5. Return $V^1_k$ as the velocity command, and this algorithm is finished;
6. end if
7. for $\psi_0 = -a * dt$, $\psi_0 = \psi_0 + \frac{\pi}{N}$, while $\psi_0 < a * dt$ do
8. $V^1_k = R(\psi(k), \theta(k)) * R(\psi_0) * v_0 * \mathbf{u}$;
9. if col($V^1_k$, objectSet) = 0; then
10. Return $V^1_k$ as the velocity command, and this algorithm is finished;
11. end if
12. $V^1_k = R(\psi(k), \theta(k)) * R(-\psi_0) * v_0 * \mathbf{u}$;
13. if col($V^1_k$, objectSet) = 0; then
14. Return $V^1_k$ as the velocity command, and this algorithm is finished;
15. end if
16. end for
17. end for
18. Run Algorithm 2 (change both yaw and pitch);

Algorithm 1 finds a non-collision velocity command while the AUV does not change its pitch. In this manner, the AUV’s unnecessary depth change can be avoided. The AUV only finds a non-collision velocity while changing its yaw angle only.

Algorithm 1 finds a non-collision speed utilizing the following FOR loop: for $\psi_0 = -a * dt$, $\psi_0 = \psi_0 + \frac{\pi}{N}$, while $\psi_0 < a * dt$. The right turn is examined before the left turn in Algorithm 1. This order is set by considering the collision regulations (COLREGS) [35]. Suppose that there is a maneuvering object, such as other AUV, which also obeys COLREGs. As both the moving object and AUV obey COLREGs, they can avoid collisions with each other when they are close to each other.

In Algorithm 1, objectSet indicates the set of objectPoint sensed at sampling-index $k$. If $V^1_k$ and every objectPoint in objectSet meet the non-collision requirement, then this implies
that $V^I_k$ is a non-collision velocity. Thus, we set $V^I_k$ as the velocity vector at sampling-index $k + 1$, and Algorithm 1 is completed.

**Algorithm 2** Find a non-collision velocity command at sampling-index $k + 1$ (change both yaw and pitch)

- The current sampling-index is $k$.
- $\text{minColNum} = \infty$.
- for $v_0 = A, v_0 = v_0 + \frac{B}{N}$, while $v_0 \leq A + B$ do
  - for $\theta_0 = -\alpha \ast dt, \theta_0 = \theta_0 + \frac{2\pi \ast \beta}{N}$, while $\theta_0 < \alpha \ast dt$ do
    - $V^I_k = R(\psi(k), \theta(k)) \ast R(\theta_0) \ast v_0 \ast \mathbf{u}$;
    - $\text{colNum} = \text{col}(V^I_k, \text{objectSet})$;
    - if $\text{colNum} == 0$ then
      - Return $V^I_k$ as the velocity command, and this algorithm is finished;
    - end if
    - $\text{psi} = -\alpha \ast dt, \psi_0 = \psi_0 + \frac{2\pi \ast \beta}{N}$, while $\psi_0 < \alpha \ast dt$ do
      - $V^I_k = R(\psi(k), \theta(k)) \ast R(\theta_0) \ast R(\psi_0) \ast v_0 \ast \mathbf{u}$;
      - $\text{colNum} = \text{col}(V^I_k, \text{objectSet})$;
      - if $\text{colNum} == 0$ then
        - Return $V^I_k$ as the velocity command, and this algorithm is finished;
      - end if
      - $\text{updateMin} (\text{colNum}, \text{minColNum}, 0, \theta_0, v_0)$ in Algorithm 3;
    - end if
    - Return $V^I_k$ as the velocity command, and this algorithm is finished;
  - end for
  - $\text{updateMin} (\text{colNum}, \text{minColNum}, 0, \theta_0, v_0)$ in Algorithm 3;
- end if
- $\text{psi} = -\alpha \ast dt, \psi_0 = \psi_0 + \frac{2\pi \ast \beta}{N}$, while $\psi_0 < \alpha \ast dt$ do
  - $V^I_k = R(\psi(k), \theta(k)) \ast R(\theta_0) \ast R(\psi_0) \ast v_0 \ast \mathbf{u}$;
  - $\text{colNum} = \text{col}(V^I_k, \text{objectSet})$;
  - if $\text{colNum} == 0$ then
    - Return $V^I_k$ as the velocity command, and this algorithm is finished;
  - end if
  - $\text{updateMin} (\text{colNum}, \text{minColNum}, \psi_0, \theta_0, v_0)$ in Algorithm 3;
- end for
- Return $V^I_k = R(\psi(k), \theta(k)) \ast R(\text{minTheta}) \ast R(\text{minPsi}) \ast \text{minV} \ast \mathbf{u}$ as the velocity command;

In Algorithm 4, $\text{col}(V^I_k, \text{objectSet})$ returns the number of objectPoints which collide with the velocity $V^I_k$. If $\text{colNum} == 0$, then $V^I_k$ avoids colliding with every objectPoint in objectSet. Thus, we set $V^I_k$ as the velocity command at sampling-index $k + 1$, and this algorithm is completed. On the other hand, if $\text{colNum} \neq 0$, then $V^I_k$ collides with at least one objectPoint in objectSet.

In Algorithm 4, $\text{maxColNum}$ is set by considering the computational load of the algorithm. By setting $\text{maxColNum}$ as a small number, one does not have to examine the non-collision requirement for every objectPoint in objectSet. In simulations, one uses $\text{maxColNum} = 100$.

**Algorithm 3 updateMin(colNum, minColNum, psi, theta, v0)**

- if $\text{colNum} < \text{minColNum}$ then
  - $\text{minPsi} = \psi_0$;
  - $\text{minTheta} = \theta_0$;
  - $\text{minV} = v_0$;
  - $\text{minColNum} = \text{colNum}$;
- end if
Find a Non-Collision Velocity Vector at Sampling-Index $k + 1$ (Change Both Yaw and Pitch)

In the case where a non-collision velocity is not found under Algorithm 1, then Algorithm 2 runs to find a non-collision velocity command at sampling-index $k + 1$, while AUV changes both its pitch and yaw. Since AUV’s pitch varies using Algorithm 1, the depth of AUV changes using this algorithm. In Algorithm 2, $(\theta_0 = -\alpha \ast dt$, $\theta_0 = \theta_0 + \frac{2\alpha_\max dt}{N}$, while $\theta_0 < \alpha \ast dt)$ is utilized to change the pitch of AUV.

Suppose that under Algorithm 2, one cannot find a non-collision velocity command, which avoids colliding with every objectPoint in objectSet. In this case, one finds a velocity command with the minimum collision probability by finding a velocity that collides with the minimum number of objectPoints. The velocity with the minimum collision probability is $\mathbf{V}_k^I = R(\psi(k), \theta(k)) \ast R(minTheta) \ast R(minPsi) \ast minV \ast \mathbf{u}$. See the last line of Algorithm 2.

5.3. AUV with a Designated Depth

Suppose that AUV has a designated depth. This designated depth can be useful, since we may not want the depth of the AUV to be too deep or too shallow. Moreover, suppose that there are many AUVs in the workspace. Then, we can set a distinct designated depth for every AUV. In this way, each AUV can move on distinct sea layer, while not colliding with other AUVs.

In Algorithm 2, $(\theta_0 = -\alpha \ast dt$, $\theta_0 = \theta_0 + \frac{2\alpha_\max dt}{N}$, while $\theta_0 < \alpha \ast dt)$ is utilized to change the pitch of AUV. Suppose that the designated depth is deeper than AUV’s current depth. In this case, $(\theta_0 = -\alpha \ast dt$, $\theta_0 = \theta_0 + \frac{2\alpha_\max dt}{N}$, while $\theta_0 < \alpha \ast dt)$ is utilized so that one examines a negative pitch before examining a positive pitch. Suppose that the designated depth is shallower than AUV’s current depth. In this case, $(\theta_0 = \alpha \ast dt$, $\theta_0 = \theta_0 - \frac{2\alpha_\max dt}{N}$, while $\theta_0 > -\alpha \ast dt)$ is utilized so that one examines a positive pitch before examining a negative pitch.

6. Controllers for Moving towards the Goal

We address how to set AUV’s velocity command so that it approaches the goal, while avoiding collision with obstacles. Let $\mathbf{G}$ denote the goal of AUV such that AUV needs to arrive at the goal safely. Using (4), let $\mathbf{v}(k)$ denote the $L^2$ norm of $\mathbf{v}(k)$, i.e., $\mathbf{v}(k) = \|\mathbf{v}(k)\|$. Moreover, let $\mathbf{G}_d = \mathbf{G} - \mathbf{q}(k)$ for convenience.

AUV increases its speed so that it can reach the goal as fast as possible. This implies that one increases the AUV’s speed to $v_d$, which is given as follows.

$$v_d = \min(\mathbf{v}(k) + dt \ast a_{\max}, s_{\max}).$$  \hspace{1cm} (18)

Here, the min operator is utilized, since $(\mathbf{v}(k) + dt \ast a_{\max})$ cannot be larger than the AUV’s maximum speed $s_{\max}$.

Suppose the following.

$$A(\mathbf{v}(k), \mathbf{G}_d) < \alpha \ast dt.$$  \hspace{1cm} (19)
In this case, AUV’s velocity ($v(k+1)$ in (4)) is set as follows.

$$v_g = \frac{v_d \times G_q}{\|G_q\|}. \tag{20}$$

Using $v_g$ in (20), the AUV heads towards the goal.

In the case where $v_g$ in (20) satisfies the non-collision requirement in Section 5.1, $v_g$ is utilized as the velocity of the AUV for a single sample duration. Using the definition of $G_q$, $v_g$ is set so that the AUV moves towards the goal.

See Figure 1 for an illustration. The sphere is centered at $q(k)$, and its radius is $\|v(k)\|$. The AUV is at $q(k)$, which is the center of the sphere. $v(k)$ is depicted as the bold arrow. $G_q$ is depicted as the dash dotted arrow. As the AUV at $q(k)$ moves in the direction of $G_q$, it can head towards the goal of $G$.

In the case where $v_g$ does not satisfy the non-collision requirement, we use Algorithm 1 to find a velocity vector for avoiding collisions with obstacles. As long as $v_g$ satisfies the non-collision requirement, $v_g$ allows the AUV to reach the goal.

Suppose that

$$A(v(k), G_q) \geq \alpha \ast dt \hspace{1cm} \tag{21}$$

holds. In this case, the AUV at $q(k)$ cannot set $v_g$ in (20) as its velocity command due to the hardware limitations in (6). Therefore, the AUV turns with its maximum acceleration so that its velocity is as close to $v_g$ as possible.

Suppose that (21) holds. The AUV sets its velocity ($v(k+1)$ in (4)) as follows:

$$v_d = v_{d,t} + v_{d,n}. \tag{22}$$

Here, $v_{d,t}$ is parallel to $v(k)$, and $v_{d,n}$ is normal to $v(k)$. $v_d$ is set so that AUV turns with its maximum acceleration such that its velocity is as close to $v_g$ as possible. We acknowledge that $v_d$ in (22) cannot assure that the AUV reaches the goal, since the AUV’s acceleration is bounded above.
In Figure 1, $v_d$ is depicted as the red arrow. Figure 1 illustrates the following:

$$A(v(k), v_d) = \alpha * dt. \quad (23)$$

Figure 1 further illustrates that as the AUV at $q(k)$ moves with velocity $v_d$, the AUV approaches the goal $G$.

In (22), one uses the following:

$$v_{d,t} = v_d * c(\alpha * dt \frac{v(k)}{v(k)}) \quad (24)$$

which is parallel to $v(k)$. Moreover, we have the following:

$$v_{d,n} = v_d * s(\alpha * dt) \frac{G_q - G_q^{pro}}{\|G_q - G_q^{pro}\|} \quad (25)$$

where $G_q^{pro}$ is the projection of $G_q$ onto $v(k)$ and is given by the following.

$$G_q^{pro} = \frac{(G_q \cdot v(k))v(k)}{(v(k))^2}. \quad (26)$$

In Figure 1, $G_q^{pro}$ is depicted as the dotted arrow. Moreover, $G_q - G_q^{pro}$ is depicted as the dashed arrow. As depicted in Figure 1, $G_q^{pro}$ is normal to $v(k)$. This further implies that $v_{d,n}$ in (25) is normal to $v(k)$. The subscript $n$ in $v_{d,n}$ in (22) implies that $v_{d,n}$ is normal to $v(k)$, and the subscript $t$ in $v_{d,t}$ in (22) implies that $v_{d,t}$ is tangential to $v(k)$.

Using (22), (24), and (25), one satisfies the following.

$$v_d = \|v_{d,t} + v_{d,n}\| = \|v_d\| \quad (27)$$

Figure 1 illustrates that $v_d > v(k)$ using (18).

In the case where $v_d$ in (22) satisfies the non-collision requirement in Section 5.1, $v_d$ is utilized as the velocity of AUV for a single sample duration. However, in the case where $v_d$ does not satisfy the non-collision requirement, we use Algorithm 1 to find a velocity vector for avoiding collisions with obstacles.

7. MATLAB Simulations

The performance of the proposed controllers in Algorithm 1 is demonstrated by utilizing MATLAB computer simulations. We use MATLAB 2010 on the computer with the following specification: Intel(R)Core(TM)i5−7600KCPU@3.80GHz.

The sample duration is $dt = 1$ second, and $k_p = 80$ is utilized in the reactive collision evasion control. The parameters associated with the ICP algorithm are $Th_{ICP} = 10$ and $k_d = 2$.

The AUV starts from $(0, 50, 10)$ in meters. The process model of the AUV appeared in (4). In practice, AUV’s motion is perturbed by process noise, such as ocean currents. Considering this process noise, we use the following:

$$q(k + 1) = q(k) + v(k) * dt + \begin{pmatrix} \text{rand} \\ \text{rand} \\ \text{rand} \end{pmatrix} * n_p - n_p/2, \quad (28)$$

instead of (4). In (28), rand generates a random number between 0 and 1. $n_p$ indicates the process noise strength in (28). In MATLAB simulations, we use the $n_p = 1$ meter by considering the drift due to ocean currents.

The goal of AUV is $(0, 500, 50)$ in meters. AUV approaches the goal while evading collision with obstacles. To approach the goal, AUV uses the control law in Section 6.
AUV’s initial speed, maximum speed, maximum speed rate, and maximum turn rate are 6.3 m/s, $s_{\text{max}} = 12.75$ m/s, $a_{\text{max}} = 2$ m/s$^2$, and $\alpha = 1.2$ rad/s, respectively. Moreover, the minimum speed of the AUV is set as $s_{\text{min}} = 1.2$ m/s. Note that the AUV needs to move along a narrow tunnel in simulations. The tunnel width is only 50 m.

We simulate the case where the AUV uses forward-looking sonar sensors for collision evasion. The maximum sensing range of the sonar sensors is 120 m. The sensor has 180 degrees in horizontal scan, such that 10 rays are evenly emanated in the horizontal direction. Moreover, it has 180 degrees in vertical scan, such that 10 rays are evenly emanated in the vertical direction. This implies that $10 \times 10$ sonar rays are emanated in total. As the number of sonar rays increases, we can have denser objectPoints on the obstacle boundary, which improves the performance of our collision avoidance controls. Our simulations use only $10 \times 10$ sonar rays, considering the hardware limit of AUV.

As a method to provide a realistic simulation environment, sonar sensor rays generate noisy position measurements. Recall that $S(k)$ defines the set of objectPoints measured at sampling-index $k$ and that $S(k)$ is given by $S(k) = \{x_i\} \in \{1, \ldots, M_k\}$. In MATLAB simulations, we use the following:

$$x_i = x_i^p + \begin{pmatrix} \text{randn} \\ \text{randn} \\ \text{randn} \end{pmatrix} \ast n_r. \quad (29)$$

Here, $x_i^p$ indicates an objectPoint derived using sonar rays without measurement noise. $x_i^p$ is derived by calculating the intersection between a sonar ray and an obstacle surface. In (29), randn generates a Gaussian noise with mean zero and variance 1. Moreover, $n_r$ indicates the standard deviation for the measurement noise in sonar rays. In MATLAB simulations, we use $n_r = 0.05$ m.

As another method to provide a realistic simulation environment, sonar sensor rays are emanated with a low detection rate of 0.95. In other words, one randomly chooses 95 percent of all $10 \times 10$ sonar rays and utilizes the chosen rays to sense an obstacle boundary. As a chosen sonar ray intersects a boundary, an objectPoint is generated at the intersection point.

Recall we say that the AUV is in the near-collision state, in the case where the distance between AUV and any objectPoint is shorter than $r$. In simulations, $r = 10$ is set by considering the size of AUV. In the case where the relative distance between AUV and an obstacle boundary is less than $r_c = 5 < r = 10$ m, we assume that a collision has occurred. We use $r_c < r$ for AUV collisions, since sonar measurement error and the process noise can lead to the worst case where AUV enters the near-collision state. In other words, $r - r_c$ is the safety margin for the AUV.

In MATLAB simulations, the objectPoints measured by the AUV’s sonar sensors are illustrated with black points. In addition, the AUV’s path is illustrated with red circles. The goal is illustrated with the green asterisk. The simulations use one cylindrical obstacle and multiple box-shaped obstacles. The blue points in the figures indicate the vertices of each obstacle.

7.1. Scenario 1

We consider Scenario 1 with static obstacles. Figure 2 depicts the top view of the workspace. In addition, Figure 3 depicts the three-dimensional view of the workspace. These figures depict that AUV arrives at the goal (green asterisk) safely. See that objectPoints measured by sonar sensors are generated on obstacle surfaces.
Figure 2. Scenario 1 with static obstacles. AUV starts from \((0, 50, 10)\) (top view).

Figure 3. Scenario 1 with static obstacles. AUV starts from \((0, 50, 10)\) (three-dimensional view).

Figure 4 depicts the state (speed, yaw, and pitch) of AUVs as time elapses, in the scenario of Figure 2. See that the AUV’s speed is upper bounded by \(s_{\text{max}}\). The last subplot in Figure 4 shows the obstacle distance (distance from AUV to the closest obstacle) measured by AUV’s sonar. Since the maximum sensing range of the sonar sensors is 120 m, the obstacle’s distance is always less than 120 m. Initially, the AUV measures a cylindrical obstacle (circle located at \((0,200)\) in Figure 2) and initiates changing its yaw for collision evasion. See that the AUV varies its state (speed, yaw, and pitch), while it moves.
Figure 4. Scenario 1 with static obstacles. The state (speed, yaw, and pitch) of the AUV as time elapses, in the scenario of Figure 2. The last subplot in Figure 4 shows the obstacle’s distance (distance from the AUV to the closest obstacle) measured by AUV’s sonar. Since the maximum sensing range of the sonar sensors is 120 m, the obstacle’s distance is always less than 120 m.

Using MATLAB, it takes 44 s to run the entire simulation in Scenario 1. This computational time includes the time consumed to generate virtual obstacle environments, while we run the entire MATLAB simulation.

7.2. Scenario 2

We consider Scenario 2, in which the cylindrical obstacle in Figure 2 maneuvers with velocity \((-3, -3, 0)\) in m/s. At every 10 s, the moving obstacle (top and bottom circles of the cylindrical obstacle) is plotted. Figure 5 plots the top view of the workspace. Moreover, Figure 6 plots the three-dimensional view of the workspace. AUV arrives at the goal (green asterisk) while satisfying collision evasion.

Figure 5. Scenario 2. The cylindrical obstacle maneuvers with velocity \((-3, -3, 0)\) in m/s (top view). At every 10 s, the maneuvering obstacle (top and bottom circles of the cylindrical obstacle) is depicted. The AUV reaches the goal (green asterisk) while assuring collision evasion.
Figure 6. Scenario 2. The cylindrical obstacle maneuvers with velocity \((-3,-3,0)\) in m/s (3D view). At every 10 s, the moving obstacle (top and bottom circles of the cylindrical obstacle) is plotted. The AUV arrives at the goal (green asterisk) while satisfying collision evasion.

Figure 7 depicts the state (speed, yaw, and pitch) of AUV as time elapses, considering the scenario in Figure 6. The last subplot in Figure 7 represents the obstacle distance (distance to the closest obstacle) measured by AUV’s sonar. Initially, the AUV measures a maneuvering cylindrical obstacle surface and changes its yaw for evading collision. See that the AUV varies its state (speed, yaw, and pitch) during the maneuver.

Figure 7. Scenario 2. The state (speed, yaw, and pitch) of the AUV as time elapses, considering the scenario in Figure 6. The last subplot shows the obstacle distance (distance to the closest obstacle) measured by AUV’s sonar.

Using MATLAB, it takes 55 s to run the entire simulation in Scenario 2. This computational time includes the time spent to generate virtual obstacle environments, as one runs the entire MATLAB simulation.
7.3. Scenario 3

We consider Scenario 3, in which the cylindrical obstacle in Figure 2 maneuvers with velocity \((0, -3, 0)\) in m/s. At every 10 s, the moving obstacle (top and bottom circles of the cylindrical obstacle) is depicted. Figure 8 depicts the top view of the workspace, and Figure 9 plots the three-dimensional view of the workspace. These figures demonstrate that the AUV reaches the goal (green asterisk) while achieving collision evasion. The movie file of Figure 9 is uploaded on the following website: https://www.youtube.com/watch?v=WklOj81QUsA (accessed on 22 July 2022).

Figure 8. Scenario 3. The cylindrical obstacle maneuvers with velocity \((0, -3, 0)\) in m/s (top view). At every 10 s, the maneuvering obstacle (top and bottom circles of the cylindrical obstacle) is depicted.

Figure 9. Scenario 3. The cylindrical obstacle maneuvers with velocity \((0, -3, 0)\) in m/s (three-dimensional view). At every 10 s, the moving obstacle (top and bottom circles of the cylindrical obstacle) is depicted.

Figure 10 depicts the state (speed, yaw, and pitch) of the AUV as time passes, considering the scenario in Figure 9. The last subplot in Figure 10 presents the obstacle distance (distance from AUV to the closest obstacle) measured by the AUV’s sonar. Initially, the
AUV measures a maneuvering cylindrical obstacle surface and varies its yaw for evading collision. See that the AUV changes its state (speed, yaw, and pitch) during the maneuver.

Figure 10. Scenario 3. The state (speed, yaw, and pitch) of the AUV as time elapses, considering the scenario in Figure 9. The last subplot shows the obstacle’s distance (distance to the closest obstacle) measured by AUV’s sonar.

Using MATLAB, it takes 41 s to run the entire simulation in Scenario 3. This computational time includes the time spent to construct virtual obstacle environments, while we run the entire MATLAB simulation.

8. Conclusions

This article introduced the reactive collision evasion controls, considering three-dimensional scenarios where AUV measures extended obstacles (moving obstacles as well as static obstacles) utilizing on-board sonar sensors. The velocity of each objectPoint of an extended obstacle is estimated using ICP algorithms. The estimated obstacle velocity is then utilized as the input for reactive collision evasion of the obstacle.

We further discussed how to make the AUV move towards the goal while avoiding collision with extended obstacles. The performance of the proposed reactive controllers was demonstrated utilizing computer simulations. In the future, we will demonstrate the performance of the proposed controls utilizing experiments with real underwater robots.

The proposed collision evasion controls can be integrated with trajectory tracking controls in the literature [32–34]. We derive the position and orientation of the AUV \( \eta = [x, y, z, \phi, \theta, \psi] \) at each sampling-index \( k \) [34]. Here, \( [x, y, z] \) presents the position coordinates, and \( [\phi, \theta, \psi] \) presents the orientation of AUV. In \( \eta \) at sampling-index \( k \), \( [x, y, z] \) presents the velocity command \( v(k) \) in (4). At each sampling-index \( k \), the AUV examines whether the calculated command \( v(k) \) meets the non-collision requirement. If the non-collision requirement is met, then the AUV uses the command \( v(k) \) for tracking the trajectory. Otherwise, the AUV uses the proposed collision evasion controls for avoiding collision with obstacles.

This article considers reactive control for a single AUV. In the future, we will consider path planning of multiple AUVs while evading collision simultaneously. References [36–39] can be utilized for localization of multiple AUVs. Moreover, Refs. [40–43] can be utilized for the reliable formation control of multiple AUVs.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.
Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Sung, M.; Kim, J.; Cho, H.; Lee, M.; Yu, S.C. Underwater-Sonar-Image-Based 3D Point Cloud Reconstruction for High Data Utilization and Object Classification Using a Neural Network. *Electronics* 2020, 9, 1763. [CrossRef]

2. Besl, P.J.; McKay, N.D. A method for registration of 3-D shapes. *IEEE Trans. Pattern Anal. Mach. Intell.* 1992, 14, 239–256. [CrossRef]

3. Pomerleau, F.; Colas, F.; Siegwart, R.; Magnenat, S. Comparing ICP variants on real-world data sets. *Auton. Robot.* 2013, 34, 133–148. [CrossRef]

4. He, Y.; Liang, B.; Yang, J.; Li, S.; He, J. An Iterative Closest Points Algorithm for Registration of 3D Laser Scanner Point Clouds with Geometric Features. *Sensors* 2017, 17, 1862. [CrossRef] [PubMed]

5. Chetverikov, D.; Svirko, D.; Stepanov, D.; Krsek, P. The Trimmed Iterative Closest Point algorithm. In Proceedings of the 2002 International Conference on Pattern Recognition, Quebec City, QC, Canada, 11–15 August 2002; Volume 3, pp. 545–548.

6. Bergstrom, P.; Edlund, O. Robust registration of point sets using iteratively reweighted least squares. *Comput. Optim. Appl.* 2014, 58, 543–561. [CrossRef]

7. Naus, K.; Marchel, L. Use of a Weighted ICP Algorithm to Precisely Determine USV Movement Parameters. *Appl. Sci.* 2019, 9, 3530. [CrossRef]

8. Greenspan, M.; Yurick, M. Approximate k-d tree search for efficient ICP. In Proceedings of the Fourth International Conference on 3-D Digital Imaging and Modeling, Banff, AB, Canada, 6–10 October 2003; pp. 442–448.

9. van den Berg, J.; Guy, S.J.; Ming Lin and, D.M. Reciprocal n-Body Collision Avoidance. *Robot. Res. Springer Tracts Adv. Robot.* 2011, 70, 3–19.

10. Kosecka, J.; Tomlin, C.; Pappas, G.; Sastry, S. Generation of conflict resolution maneuvers for air traffic management. In Proceedings of the International Conference of Intelligent Robotic Systems, Grenoble, France, 8–13 September 1997; pp. 1598–1603.

11. Chakravarthy, A.; Ghose, D. Obstacle avoidance in a dynamic environment: A collision cone approach. *IEEE Trans. Syst. Man Cybern.* 1998, 28, 562–574. [CrossRef]

12. Lalish, E.; Morgansen, K. Distributed reactive collision avoidance. *Auton. Robot.* 2012, 32, 207–226. [CrossRef]

13. Wang, D.; Wang, P.; Zhang, X.; Guo, X.; Shu, Y.; Tian, X. An obstacle avoidance strategy for the wave glider based on the improved artificial potential field and collision prediction model. *Ocean. Eng.* 2020, 206, 107356. [CrossRef]

14. Rostami, S.M.H.; Sangahia, A.K.; Wang, J.; Liu, X. Obstacle avoidance of mobile robots using modified artificial potential field algorithm. *EURASIP J. Wirel. Commun. Netw.* 2019, 70, 1–19. [CrossRef]

15. Lalish, E. *Distributed Reactive Collision Avoidance*; University of Washington: Seattle, DC, USA, 2009.

16. Sunkara, V.; Chakravarthy, A. Collision avoidance laws for objects with arbitrary shapes. In Proceedings of the 2016 IEEE 55th Conference on Decision and Control (CDC), Las Vegas, NV, USA, 12–14 December 2016; pp. 5158–5164. [CrossRef]

17. Leonard, J.; Savvaris, A.; Tsourdos, A. Distributed reactive collision avoidance for a swarm of quadrotors. *Proc. Inst. Mech. Eng. Part J, Aerosp. Eng.* 2017, 231, 1035–1055. [CrossRef]

18. Kuwata, Y.; Wolf, M.T.; Zarezhitsky, D.; Huntsberger, T.L. Safe Maritime Autonomous Navigation with COLREGS, Using Velocity Obstacles. *IEEE J. Oceanic Eng.* 2014, 39, 110–119. [CrossRef]

19. Yang, X.; Alvarez, L.M.; Bruggemann, T. A 3D Collision Avoidance Strategy for UAVs in a Non-Cooperative Environment. *J. Intell. Robot. Syst.* 2013, 70, 315–327. [CrossRef]

20. Roelofsen, S.; Martinoli, A.; Gillet, D. 3D collision avoidance algorithm for Unmanned Aerial Vehicles with limited field of view constraints. In Proceedings of the 2016 IEEE 55th Conference on Decision and Control (CDC), Las Vegas, NV, USA, 12–14 December 2016; pp. 2555–2560.

21. Xiong, W.; Cong, G.; Yi, B. Variable probability based bidirectional RRT algorithm for UAV path planning. In Proceedings of the Control and Decision Conference (2014 CCDC), the 26th Chinese, Changsha, China, 31 May–2 June 2014; pp. 2217–2222.

22. Zheng, Z.; Bewley, T.R.; Kuester, F. Point Cloud-Based Target-Oriented 3D Path Planning for UAVs. In Proceedings of the 2020 International Conference on Unmanned Aircraft Systems (ICUAS), Athens, Greece, 9–12 June 2020; pp. 790–798.

23. Wiig, M.S.; Pettersen, K.Y.; Kroghst, T.R. A 3D reactive collision avoidance algorithm for underactuated underwater vehicles. *J. Field Robot.* 2020, 37, 1094–1122. [CrossRef]

24. Choi, Y.K.; Wang, W.; Liu, Y.; Kim, M. Continuous Collision Detection for Two Moving Elliptic Disks. *IEEE Trans. Robot.* 2006, 22, 213–224. [CrossRef]

25. Lü, L.; Zong, C.; Lei, X.; Chen, B.; Zhao, P. Fixed-Wing UAV Path Planning in a Dynamic Environment via Dynamic RRT Algorithm. *Mech. Mach. Sci. Lect. Notes Electr. Eng.* 2017, 408, 271–282.

26. Ju, M.Y.; Liu, J.S.; Shiang, S.P.; Chien, Y.R.; Hwang, K.S.; Lee, W.C. A novel collision detection method based on enclosed ellipsoid. In Proceedings of the 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164), Seoul, Korea, 21–26 May 2001; Volume 3, pp. 2897–2902.

27. Fossen, T.I. *Guidance and Control of Ocean Vehicles*; John Wiley and Sons: Hoboken, NJ, USA, 1994.
28. Allotta, B.; Costanzi, R.; Fanelli, F.; Monni, N.; Paolucci, L.; Ridolfi, A. Sea currents estimation during AUV navigation using Unscented Kalman Filter. IFAC-PapersOnLine 2017, 50, 13668–13673. [CrossRef]
29. Allotta, B.; Caiti, A.; Costanzi, R.; Corato, F.D.; Fenucci, D.; Monni1, N.; Pugi, L.; Ridolfi, A. Cooperative navigation of AUVs via acoustic communication networking: Field experience with the Typhoon vehicles. Auton. Robot. 2016, 40, 1229–1244. [CrossRef]
30. Allotta, B.; Costanzi, R.; Fanelli, F.; Monni, N.; Ridolfi, A. Single axis FOG aided attitude estimation algorithm for mobile robots. Mechatronics 2015, 30, 158–173. [CrossRef]
31. Heshmati-Alamdari, S.; Nikou, A.; Dimarogonas, D.V. Robust Trajectory Tracking Control for Underactuated Autonomous Underwater Vehicles in Uncertain Environments. IEEE Trans. Autom. Sci. Eng. 2021, 18, 1288–1301. [CrossRef]
32. Elhaki, O.; Shojaei, K.; Mehrmohammadi, P. Reinforcement learning-based saturated adaptive robust neural-network control of underactuated autonomous underwater vehicles. Expert Syst. Appl. 2022, 197, 116714. [CrossRef]
33. Shojaei, K. Three-dimensional tracking control of autonomous underwater vehicles with limited torque and without velocity sensors. Robotics 2018, 36, 374–394. [CrossRef]
34. Zheng, J.; Song, L.; Liu, L.; Yu, W.; Wang, Y.; Chen, C. Fixed-time sliding mode tracking control for autonomous underwater vehicles. Appl. Ocean. Res. 2021, 117, 102928. [CrossRef]
35. Agrawal, P.; Donl, J.M. Colregs-compliant target following for an unmanned surface vehicle in dynamic environments. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Hamburg, Germany, 28 September–3 October 2015; pp. 1065–1070.
36. Kim, J. Cooperative Localization and Unknown Currents Estimation Using Multiple Autonomous Underwater Vehicles. IEEE Robot. Autom. Lett. 2020, 5, 2365–2371. [CrossRef]
37. Bahr, A.; Leonard, J.J.; Fallon, M.F. Cooperative Localization for Autonomous Underwater Vehicles. Int. J. Robot. Res. 2009, 28, 714–728. [CrossRef]
38. Chen, Q.; You, K.; Song, S. Cooperative localization for autonomous underwater vehicles using parallel projection. In Proceedings of the 2017 13th IEEE International Conference on Control Automation (ICCA), Ohrid, Macedonia, 3–6 July 2017; pp. 788–793.
39. Xu, B.; Li, S.; Razaqi, A.A.; Zhang, J. Cooperative Localization in Harsh Underwater Environment Based on the MC-ANFIS. IEEE Access 2019, 7, 55407–55421. [CrossRef]
40. Ando, H.; Oasa, Y.; Suzuki, I.; Yamashita, M. Distributed memoryless point convergence algorithm for mobile robots with limited visibility. IEEE Trans. Robot. Autom. 1999, 15, 818–828. [CrossRef]
41. Jadbabaie, A.; Lin, J.; Morse, A.S. Coordination of groups of mobile autonomous agents using nearest neighbor rules. IEEE Trans. Autom. Control. 2003, 48, 988–1001. [CrossRef]
42. Gulzar, M.; Rizvi, S.; Javed, M.; Munir, U.; Asif, H. Multi-Agent Cooperative Control Consensus: A Comparative Review. Electronics 2018, 7, 22. [CrossRef]
43. Kim, J. Three dimensional distributed rendezvous in spherical underwater robots considering power consumption. Ocean. Eng. 2020, 199, 107050. [CrossRef]