Letter

Mapless Navigation Based on 2D LIDAR in Complex Unknown Environments

Kai Yan and Baoli Ma *

The Seventh Research Division, Beihang University, Beijing 100191, China; yankai2016@buaa.edu.cn

* Correspondence: mabaoli@buaa.edu.cn

Received: 14 September 2020; Accepted: 9 October 2020; Published: 14 October 2020

Abstract: This paper presents a novel approach for navigation in complex and unknown environments. At present, existing local path planners whose control output is the mapping of current sensor data have been widely studied. However, these methods cannot really solve the problem of being trapped by obstacles. We analyzed the reasons and made improvements, and finally our approach can avoid being trapped in complex environments. The proposed method is based on 2D LIDAR. A central part of the approach is finding out gaps in the environment by analyzing sensor data. Then, we choose one of the gaps we find as the sub-goal. Linear and angular velocities are provided by the approach considering nonholonomic mobile robots. The method does not rely on global planners and environment maps. Therefore, it has the advantages of low computational complexity and fast response, which is of great significance to robots with low computing power; it will also help to reduce the manufacturing cost of robots. In addition, simulations and real tests were performed using the Turtlebot2 robotic platform. Successful results are achieved in both simulations and experimental tests.

Keywords: local path planners; complex environments; 2D LIDAR; Turtlebot2

1. Introduction

The ability of autonomous navigation is important for the application of mobile robot systems. Robots should be able to reach the goal position safely among unknown environments. Excellent navigation performance is the basic premise on particular tasks. After decades of research, significant results have been achieved. Until now, robot navigation approaches can roughly be divided into two categories: global and local. The two types of methods are completely different from the research perspective, and they have their advantages and disadvantages.

1.1. Global Approaches

The first type is also called global path planning or deliberative navigation. The geometric path from the current position of the robot to the target is obtained using an environment map. There are lots of approaches to address path planning in the literature, such as artificial potential field (APF) [1], A-star(A^*), D-star(D^*) [2], rapidly exploring random trees (RRT) [3], and probabilistic roadmaps (PRMs) [4].

The advantages of global approaches are that the computed path can be optimal in terms of Euclidean distance traveled, and the robot will not be trapped in any environments (if paths exit). However, for most applications, the global world model is changeable or not available. Computing a path off-line is problematic. Under such circumstances, to reach the goal, updating the map and replanning the path repeatedly is required combined with the simultaneous location and mapping (SLAM) technology, which requires large amounts of computational resources. With the increase of map size, computational complexity increases exponentially. Therefore, global approaches

are computationally expensive and hardly suit real-time implementation for robots with lower computing power.

1.2. Local Approaches

The local approach is also known as obstacle avoidance or reactive navigation. In every control cycle, the current control command of the robot is the mapping of the on-line sensory data. The key advantages of local techniques over global ones lie in their low computational complexity and fast response. Local approaches can effectively avoid collisions with obstacles. However, pure local approaches are still challenged by complex and cluttered environments in which the robot may be trapped and cannot achieve its goal. The primary requirement of a navigation approach is to reach the target without collision, while other existing local approaches are not guaranteed to reach the goal every time. Therefore, existing local approaches are difficult to navigate independently in a complex environment. Normally, local approaches are combined with global methods to form a hybrid structure [5–8].

The paper is organized as follows: Section 2 describes related work and the our research. Section 3 describes in detail the proposed navigation approach. In Section 4, simulations and real tests are discussed using the Turtlebot2 robotic platform, and successful results are achieved. Sections 5 and 6 provides discussions and conclusions.

2. Related Work

Decades of efforts on local approaches have lead to some satisfactory algorithms. In the work of [1], Khatib puts forward the concept of artificial potential field (APF). The main idea is that the movement of the robot is the result of forces of two virtual potential fields. The surrounding environment exerts a repulsive force on the robot and the target exerts an attractive force on the robot. The robot heading is the same as the direction of the resultant force. The linear velocity of the robot is related to the magnitude of the resultant force.

In [9], the vector field histogram (VFH) is proposed, which uses a two-dimensional occupancy grid as an environment model. The model is updated continuously using the range data sampled by on-board ultrasonic proximity sensors. After a data-reduction process, a polar histogram of the environment around the robot is obtained using the occupancy information. Within the histogram, all openings that allow the robot to pass through are found. Then a cost function is applied to choose the optimal opening with the lowest cost. The travel direction is then aligned with the direction of the chosen opening. In the improved version VFH+ [10], the robot's trajectories based on its kinematic model is taken into account and thus the enhanced method can reduce the risk of collision.

The dynamic window approach (DWA) [11] can generate a smooth trajectory by selecting the motion commands in velocity space, which consists of translational and rotational velocity. Each velocity uniquely corresponds to a trajectory. Obstacles on the occupancy grid map can impose restrictions on the velocities. The objective function is applied to trade-off velocity, clearance, and target heading, then the velocity that maximizes the objective function is selected.

However, none of the three local methods mentioned above can cope with the complex environment. They may be trapped by obstacles and cannot reach the target. As far as I know, only a family of bug algorithms [12] is proved to be convergent [13]. The bug algorithms navigate the robot by following the obstacle boundary if an obstacle is encountered. However, the path generated by this simple behavior can be much longer than the optimal path.

Is it possible to navigate in complex environments using simple local approaches? In [14], the intelligent navigation behaviors of animals are believed to be caused by simple local approaches. Think carefully about how we walk ourselves: While walking, we first find out the direction of movement or set a temporary target in front of us. We do not plan a complete geometric path in our minds during the journey to the target. Our behavior is just a response to the surrounding environment. It seems like that the way humans navigate is a local approach in most cases. We were

never trapped in simple obstacles, such as U-shaped obstacles, but this is a big problem for robots using the local algorithm.

Inspired by human navigation behaviors, this paper extends earlier work [15] and overcomes the shortcomings of existing local approaches, then proposes a new local navigation strategy. Our approach differs from previous approaches in that: (a) it achieves the goal without a global planner, but will not be trapped in complex environments. It means that the proposed approach can accomplish the navigation task independently and has the advantages of low computational complexity and fast response; (b) there is no need to maintain an environment map, which are required for all global approaches and most local approaches; (c) the generated path is close to the optimal one. Besides, the algorithm considers the size and kinematics of the robot and has no restriction on the shape of obstacles. Therefore, our approach can make the robot with low computing power complete complex navigation tasks.

3. Method Description

A major part of the proposed approach is identifying gaps of the environment based on 2D LIDAR [16]. The method of identifying the gap is described in detail below. Finding a gap means that there is a large enough free space near the gap to allow the robot to pass. Then, choose one of the gaps we found as the new sub-goal, considering positions of the final target and previous sub-goals. According to the sub-goal and real-time sensor information, the robot's motion control is obtained. The specific method is described below. Compared with existing local algorithms that directly map the sensor data to the on-line control command, the proposed algorithm takes into account not only current sensor data but also other relevant information—the previous sub-goals. This is the key factor that keeps our algorithms from getting trapped. In every time interval, the navigation architecture is illustrated in Figure 1. The process iterates until the robot reaches the final goal.

3.1. Find Gaps of the Environment

The sensor employed in the proposed approach is 2D LIDAR. The LIDAR is located at the center of the robot with a field of view (FOV) of θ_{FOV} (Figure 2). The LiDAR sends beams evenly within its entire FOV. The angle between two adjacent beams is referred as LIDAR resolution, denoted in this paper as θ_0 . R_0 is the farthest detection distance of the LIDAR. For each beam with the beam number i , θ_i and d_i are the measured angle and the distance respectively. d_m, d_{m+1}, d_{m+2} in Figure 2 is the measured distance of beam number $m, m+1, m+2$.

Define the heading of the robot as 0 degrees, then θ_i is calculated as:

$$\theta_i = i\theta_0 - \theta_{FOV}/2 \quad (1)$$

where $i \in [1, N]$, with N being the total number of the LiDAR beams calculated as:

$$N = \theta_{FOV}/\theta_0 + 1 \quad (2)$$

Similar methods that navigate a robot by identifying gaps using sensor data have been proposed in [17,18]. However, these methods will be invalid when faced with dense obstacles. In this paper, a more general method for identifying gaps is designed, which can find the appropriate gaps between densely interlaced obstacles.

Figure 2 shows that the value of d_m and d_{m+1} is very close, but there is a big difference between d_{m+1} and d_{m+2} . Because the adjacent beams m and $m+1$ are directed at the same object, the smaller the LIDAR resolution is, the closer the values of d_m and d_{m+1} will be. Therefore, set a constant value d_T . When the difference between d_m and d_{m+1} is greater than d_T , we think that the adjacent beams m and $m+1$ are directed at two different objects and call the points where the two beams hit the objects as discontinuous points. At this time, the gap between these two objects may allow the robot to pass. Therefore, the first step is to find all the discontinuous points.

Figure 1. Flow diagram of the approach in one time interval.

Figure 2. Representation of the LIDAR beams. The actual number of beams is much greater than illustrated here.

(1) Finding out the discontinuous points

As mentioned above, there are discontinuous points when:

$$|d_m - d_{m+1}| > d_T \quad (3)$$

where d_T is constant. Obviously, d_T should be larger than the diameter of the robot if the robot wants to pass between two discontinuous points. In theory, the value of d_T should be as close to the robot

diameter as possible because a larger d_T may cause the loss of some discontinuities that allow the robot to pass. However, the LIDAR data d_m, d_{m+1} has deviations. If d_T is equal to or close to the robot diameter, even though the discontinuous points satisfies $|d_m - d_{m+1}| > d_T$, the real distance between the two discontinuous points may be less than the robot diameter. On the other hand, the LIDAR beams are not continuous. The farther the obstacle is from the robot, the larger the difference between d_m, d_{m+1} is. When the robot is near a wall, the phenomenon will be more clear. This can be explained by Figure 3.

Figure 3. Representation of the LIDAR beams when the robot is near to a wall.

To show more clearly, the number of LIDAR beams in Figure 3 is much smaller than the actual number of beams. We can see that the difference between d_m and d_{m+1} is greater than the robot diameter. However, the adjacent beams m and $m + 1$ are directed at the same object. So there are no openings to allow the robot to pass. At this time, setting d_T to a larger value can avoid mistakenly identifying the discontinuous points. Therefore, when we set the value of d_T , there should be a trade-off between avoiding misidentification of discontinuities and avoiding loss of discontinuities. Also, More dense LIDAR beams can avoid misidentification of discontinuities. In the following experiments, the value of d_T is obtained considering the error, resolution and farthest detection distance of the LIDAR.

(2) Making sure the discontinuous position is passable

Taking into account the size of the robot, the robot can pass only if the free space at the discontinuity is large enough. As shown in Figure 4, points A and B are discontinuous points that we found in the first step. There is a semicircle centered on point A on the right side of the line OA. If there are points detected by the LIDAR beams within the semicircle (shown in Figure 4b), we think that the robot cannot pass through the gap between point A and point B. As you can see in the picture, there is another semicircle centered at point B. The function of this semicircle is to eliminate the effects of LIDAR error signal data and extremely small obstacles. In Figure 4c, if there is a very small obstacle or error LIDAR signal at point A, checking whether there are points detected by the LIDAR beams within the semicircle at B can eliminate the interference. One LIDAR point is found in the semicircle centered on B, so there is no valid gap there.

Figure 4. Examples of gaps. The black objects are obstacles that can be detected by LIDAR, and the gray objects are obstacles that cannot be detected.

There are no points detected by LIDAR beams within the two semicircles in Figure 4a. Therefore, the robot can pass through the gap between point A and point B. Point C is used to represent this gap, whose coordinates can be obtained by the fact that OC is the perpendicular bisector of the semicircle's radius. Besides, the value of d_t is set to be slightly larger than the diameter of the robot for safety considerations. If point A or point B exceeds the maximum range of LIDAR detection, then the semicircle corresponding to this point does not need to be considered, as shown in Figure 5.

(3) Making sure the passageway to the gap is not blocked

Figure 5. Examples of gaps. Rectangle CDEO and BCOA have a common side CD, and the lengths of sides CD and CB are equal to d_T .

In the previous step, the gap that allows the robot to pass is found. However, moving directly toward the gap may be impracticable. As shown in Figure 5a, we find the gap by step (2). Nevertheless, after adding two point-like obstacles (Figure 5b), point C cannot be reached directly. Therefore, we cannot regard gap C in Figure 5b as a valid gap. To rule out this situation, we have defined two sets of points M and N. Point set M contains all the points detected by the LIDAR in the rectangle CDEO, and point set N contains all the points detected by the LIDAR in the rectangle BCOA. By checking gaps that found in the step (2) with the following conditions, we get the valid gaps that are not blocked:

$$d_{p_1 p_2} > d_T \quad \forall p_1 \in M, \forall p_2 \in N \quad (4)$$

where $d_{p_1 p_2}$ is the distance between p_1 and p_2 . The method to obtain the elements of set M and N is to find the points that meet the following two conditions among the points detected by the LIDAR: (1) The distance from this point to the straight line OC is less than d_T . (2) The distance from this point to the robot center O is less than the length of the OC. The set determined by this method is slightly different from the set M and N defined before, but this does not affect the research of this article.

By the above three steps, we can find all the valid gaps. The next part will show how to choose one of these gaps as a sub-goal.

3.2. Selecting Sub-Goal from Gaps

Local navigation approaches are challenged by complex environments. For example, most of the local approaches cannot detour a big U-shaped obstacle. Some solutions [19,20] develop navigation strategies based on an empirical evaluation of trap situations. But they may still get stuck in some special environments. In [17], the author also adopted the concept of the gap, but the sub-goal was

selected from gaps by the cost function method, which sometimes produced strange trajectories. Other local approaches, such as [21–23], did not solve the problem of navigation in complex environment. These local approaches have the common feature that they have no memory of the sensor data detected in the past. Our findings show that if one local planner does not take the past sensor data into account and there is no global planner available, then any improvement to the local planner cannot avoid being trapped by obstacles while considering the optimal path. Therefore, many existing local approaches cannot completely solve the trapped problem, even if they claim that they can avoid local minimum problems. This can be illustrated in Figure 6.

Figure 6. The identified gaps and manual selected direction of motion in two different positions.

In Figure 6, the red point and green point are the robot position and the target position, respectively. Circles drawn with dotted lines represent the identified gaps. Firstly, we think that the two images are independent. Assuming that the robot can select the gap at the level of human intelligence, the optimal gap and motion direction is shown in Figure 6. However, the problem arises when Figure 6a is followed by Figure 6b. In Figure 6b, the robot arrives at the gap that selected in Figure 6a. Nevertheless, the manual selected optimal gap is close to the start point in Figure 6a, causing the robot to repeat the movement. Even if we derive the new sub-goal combining the current motion direction, the problem still exists.

To eliminate this defect, we have improved the traditional local approaches. When selecting a new sub-goal, the proposed approach combines the current sensor data with the previous sub-goals, which can effectively avoid the robot being trapped. The method we adopt is very simple, that is, the new sub-goal avoids the gaps that passed previously. In the remainder of this section, we will describe the method of selecting the sub-goal from these gaps.

In the previous part, we already know how to find out valid gaps. As shown in Figure 4a, there is a gap (indicated by point C). We find this gap since point A, point B, and its nearby LIDAR points satisfy those three conditions. The point that is closer to the robot between A and B is referred to as the origin of gap C. In order to avoid the robot being trapped because of continuous circular motion, the method we adopt is creating a blacklist to store the global coordinates of the origin of previously visited sub-goals (gaps). When selecting a new sub-goal, gaps whose origin is close to the point in the blacklist will not be taken into account. Furthermore, the origin of the sub-goal will be stored in the list only when the robot reaches this sub-goal and then leaves it a certain distance of d_{list} . In this way, the robot will not return to the gaps that have been to before.

In Figure 7a, the optimal sub-goal for manual selection is gap A. However, our robot does not have such a high level of intelligence. If a formula is used to calculate the optimal sub-target, the new problem comes out. For example, we can select the gap with the smallest value of d_{sum} as the sub-goal:

$$d_{sum} = d_r + d_g \quad (5)$$

where d_r is the distance from the gap to the robot, d_g is the distance from the gap to the goal. Obviously, the optimal sub-goal is gap B, see Figure 7a. Since the distance from the robot to the

gap B or D is less than d_{list} , the origin of gaps B, D is not stored in the blacklist. Therefore the gap D cannot be ignored. However, gap D is selected as a new sub-goal since it has the smallest value of d_{sum} , see Figure 7b. The new sub-goal D is close to the starting point, causing the robot to repeat the movement.

Figure 7. The direction of motion selected by a formula in two different positions.

To solve this problem, we add a new restriction to the sub-goal: the distance between the new sub-goal and the last sub-goal should be greater than the distance between the current position and the last sub-goal. For example, in Figure 7a, the starting point is treated as the first sub-goal, and the new sub-goal is gap B. In Figure 7b, the robot reaches gap B. The distance between gap D and the first sub-goal (the last sub-goal) is smaller than the distance between the current position B and the first sub-goal. Therefore, gap D can not be selected as the second sub-goal. In the remaining gaps, gap A is selected as the second sub-goal because of the minimum value of d_{sum} .

In short, the method of selecting sub-goal from gaps is divided into three steps.

- (1) Exclude gaps whose origins are close to the point in the blacklist.
- (2) Exclude gaps that are closer to the last sub-goal than the current position.
- (3) Select the gap with the smallest value d_{sum} as the new sub-goal.

3.3. Motion Control

Looking back at the flow chart in Figure 1, we can see that the sub-goal is not recalculated in each time interval. Only if the current sub-goal has been reached or is blocked, the sub-goal is updated. Methods of finding gaps and selecting sub-goal from gaps have been described in detail earlier. The location of the robot can be obtained by the equipped encoder. The cumulative error has little effect on the proposed method because the previous sub-targets that are farther away from the current position usually do not need to be considered and the previous sub-targets closer to the current position have small coordinates cumulative error. Therefore the global coordinates of the sub-goal can be derived, and it is easy to judge whether the sub-goal has been reached. The method of determining whether a sub-goal is blocked is described in Equation (4). Therefore, the last problem in Figure 1 is how to derive robot control commands based on the current sub-goal and LIDAR data.

In this paper, it is considered the wheeled mobile robot of unicycle type, in which the state variables are x, y (the world coordinates of the robot's middle point), and θ (angle of the vehicle with the world X-axis). The kinematics of the robot can be modeled by:

$$\begin{cases} \dot{x} = v \cos \theta \\ \dot{y} = v \sin \theta \\ \dot{\theta} = \omega \end{cases} \quad (6)$$

First, we design a piecewise control input. Once the current sub-goal is updated, a turning angle is then produced to drive the robot to face the direction of the sub-goal. At this time, the linear velocity v is set to zero to avoid collisions:

$$\begin{cases} v = 0 \\ \omega = k_2 \text{atan}(k_3 \theta_g) \end{cases} \quad (7)$$

When θ_g is close to zero, we can set the linear velocity v to a fixed value and keep the robot moving towards the sub-goal:

$$\begin{cases} v = k_1 \\ \omega = k_2 \text{atan}(k_3 \theta_g) \end{cases} \quad (8)$$

where k_1, k_2, k_3 are three positive gains, θ_g is the angle of the sub-goal in the robot coordinate system.

In Equation (7), the robot rotates and does not collide. However, Equation (8) may cause collisions because of small obstacles and previously undetected obstacles. Therefore Equation (8) should be adjusted according to real-time LIDAR data.

First of all, we define a region D (Figure 8) in the robot coordinate system. When obstacles are detected in area D , the control input (Equation (8)) should be changed accordingly.

Figure 8. Representation of region D .

Where r is the actual radius of the robot, R is the safe radius, which is larger than r to maintain a safe distance from the obstacle. The yellow area is D . The method of determining whether point $P(\theta, d)$ belongs to area D is given by:

$$P(\theta, d) \in D \quad \text{if } R < d < R_D \text{ and } |d \sin(\theta)| < R \text{ and } \theta \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right) \quad (9)$$

Then, consider a point-like obstacle, as shown in Figure 9.

Where point A is the detected obstacle and has polar coordinates (θ, d) . If the robot passes from the left side of the point-like obstacle A and the trajectory is approximated as an arc, we can derive the radius r_{BO} (Equation (10)) of the arc where the robot center is located according to the right triangle ABC. Since the robot's linear velocity is fixed value v , the value of angular velocity ω (Equation (11)) is derived in situation Figure 9.

$$(r_{BO} + R)^2 = (r_{BO} - d \sin \theta)^2 + (d \cos \theta)^2$$

$$\Rightarrow r_{BO} = (d^2 - R^2) / (2(R + d \sin \theta)) \quad (10)$$

$$\Rightarrow \omega = v / r_{BO} = 2v(R + d \sin \theta) / (d^2 - R^2) \quad (11)$$

Figure 9. The robot passes by the left side of point-like obstacle A.

Similarly, when the robot passes from the right side of the obstacle A, the angular velocity ω is calculated by:

$$\omega = v / r_{BO} = 2v(-R + d \sin \theta) / (d^2 - R^2) \quad (12)$$

Generally speaking, real obstacles cannot be considered as a point. As shown in Figure 10, there is a black object in area D and multiple points (d, θ) are detected by LIDAR beams.

Figure 10. There is an obstacle in area D.

According to these detected points in area D, the distance from the obstacle to the left and right borders of area D can be calculated by:

$$d_l = \min\{R - d \cos(\theta) \mid (d, \theta) \in D\} \quad (13)$$

$$d_r = \min\{R + d \cos(\theta) \mid (d, \theta) \in D\} \quad (14)$$

For every point detected by LIDAR in area D, two angular velocities can be calculated using Equations (11) and (12). Therefore, the maximum value ω_{max} and the minimum value ω_{min} among these angular velocities can be derived. According to Equations (11) and (12), we can find that the value of ω can be very large when the distance d to the obstacle is close to the value of R . However, the angular velocities ω of the robot is bounded, and its maximum and minimum values are represented by ω_1, ω_2 ,

respectively. At this moment, the linear speed v of the robot should be reduced to avoid the angular velocity ω exceeding the boundary. In short, in order to detour obstacles as shown in Figure 10, the following changes are made in contrast to Equation (8):

$$\begin{cases} \omega = \omega_{max}, v = k_1 & \text{if } d_l \geq d_r \text{ and } \omega_{max} < \omega_1 \\ \omega = \omega_1, v = k_1 \omega_1 / \omega_{max} & \text{if } d_l \geq d_r \text{ and } \omega_{max} \geq \omega_1 \\ \omega = \omega_{min}, v = k_1 & \text{if } d_l < d_r \text{ and } \omega_{min} \geq \omega_2 \\ \omega = \omega_2, v = k_1 \omega_2 / \omega_{min} & \text{if } d_l < d_r \text{ and } \omega_{min} < \omega_2 \end{cases} \quad (15)$$

$\omega_{max}, \omega_{min}$ correspond to angular velocities of the robot just passing through the left side and right side of the obstacle. Is it possible that area D in the sub-target direction is entirely blocked by the obstacle causing $\omega_{max}, \omega_{min}$ close to 0? No, because in each time interval, we will investigate whether the present sub-goal is blocked. If the current sub-goal is blocked, a new sub-goal will be selected.

To sum up, the steps of motion control in each time interval are as follows:

- (1) If there are no obstacles in area D, control input is obtained by Equation (7) and (8). Otherwise, go to step (2) and (3)
- (2) Calculate values of $d_l, d_r, \omega_{max}, \omega_{min}$ using the points detected in area D.
- (3) Replace Equation (8) with Equation (15).

4. Simulations and Experiments

4.1. Robotic Platform

The simulation and experimental platform for this novel approach is a wheeled differential drive robot, called Turtlebot2. We added a Hokuyo UST-20LX laser range finder to the robot. The sensor has a field of view of 270° and a maximum scanning range of 10 m. The angular resolution is 0.25° and 1080 measurements can be obtained per cycle. A laptop with an AMD A4-6210 APU processor running Ubuntu 16.04 is an onboard computer. The robot operating system (ROS) [24] running on the laptop is the middleware. Before the real tests of the proposed approach, the simulation was performed using the Gazebo/Rviz.

Gazebo is an open-source 3D robotic simulator, which integrated high-performance physics engines. It can provide simulation models for various scenarios in the real world. It can also model sensors, such as laser range finders and cameras. The simulation structure is shown in Figure 11. Gazebo models the robot, sensors, and environments. Rviz provides several plugins that display information such as images, models, paths, and more. The simulation software system is based on ROS, and each module communicates via ROS topic.

Figure 11. The simulation structure.

4.2. Further Implementation Details

The method proposed in this paper has been introduced in the second section. However, there are still many details in the experiment that have not yet been explained. The first question is the parameter settings. The parameters used during the experiment are given in Table 1.

Table 1. The parameters in the experiment.

Parameter	Significance	Value
r	The radius of Turtlebot2	0.17 m
d_T	The parameter in Equation (3)	0.6 m
k_1	The parameter in Equation (8)	0.2
k_2	The parameter in Equation (7)	1.5
k_3	The parameter in Equation (8)	1
R	The safe radius R of the robot in Figure 8	0.25 m
R_D	The radius of region D in Figure 8	0.5 m

Since it is not necessary to accurately reach the position of the sub-goal, when the distance between the present position and the sub-goal is less than 0.2, it is regarded as reaching the sub-goal. As shown in Figure 4a, this gap is represented by point C. When calculating the coordinates of point C, extend OC by 0.2 to compensate for the error. When selecting sub-goal from gaps, we established a blacklist. The gaps whose origin is close to the point in the blacklist will be excluded. In our experiments, that the distance to points in the blacklist is less than 0.6 is considered 'close'. In step (3) of selecting sub-goal from gaps, the gap with the smallest value d_{sum} is selected as the new sub-goal. In fact, we found that choosing the gap with the smallest value of d_g as a sub-goal can achieve better results. Also, as shown in Figure 1, in every control cycle, first considers if the final goal is blocked, if it is blocked, continue to move to the current sub-goal, otherwise go to the final target. The methods mentioned in this section are not unique. Maybe another solution is better, but the scheme we have adopted has already achieved our purpose well.

In this experiment, the equipped encoder is the only way to positioning. Even if there are cumulative errors, it has little effect on the approach. Because we only care about the coordinates of several recent gaps to prevent going back to the position we just passed. The position of the recent gaps has little cumulative error relative to the position of the robot.

4.3. Simulation Results

The avoidance of local minima is the main weakness of reactive based navigation approaches [25]. This leads to most existing navigation systems being dominated by global methods, while the reaction layer is simply a performer. The approach proposed in this paper stress on improving the local navigation to avoid being trapped by obstacles, so that the local algorithm can complete complex navigation tasks independently. Therefore, whether the robot can successfully reach the goal is the most concerning issue in this article.

Because the actual results of every obstacle avoidance algorithms are related to the parameter setting and other local approaches are difficult to reach the target in complex scenarios, hence in the simulation part, we did not compare the proposed approach with other methods. As complex environments can be more easily constructed in simulation environments, extensive simulation studies were carried to prove the effectiveness of the new method. Figure 12 shows six typical simulation scenarios, which are the screenshots of the simulation after the robot reaches the target. Since the gazebo software itself cannot display the motion path of the robot, we manually mark where the robot passed with grey circles. The position of the robot in the picture is the final target.

Figure 12. Examples of simulation.

First, look at scenario 1, we build a scene similar to the one in Figure 6, and the robot arrives at the target successfully. Because our method takes into account the size of the robot, small openings between the obstacles are not identified as valid gaps. In scenarios 2 and 3, the robot can pass concave obstacles. In scenario 2, after finding the sub-goal point, the robot could adjust the motion control in time because of the cylindrical obstacle. Scenarios 4, 5, and 6 are more complicated. The approach proposed in this paper successfully reached the target in the six scenarios, and the resulting path is close to the optimal path. But these are challenging tasks for other local approaches.

4.4. Real Robot Experiments

To further verify the applicability of the new navigation algorithm proposed in this paper, we have implemented it on a real Turtlebot2 robot. Other hardware and parameters in the real experiment and the simulation test are the same. The advantage is that the simulation results are basically the same as the real experimental results. Because both simulation and experiment are based on ROS, the simulation code is easy to transplant to the real robot. Figure 13 shows the robot we use.

The goal was set 6 m in front of the robot. Figure 14 illustrates the environment where the experiments were performed. Figure 15 shows linear velocity and the angular velocity of the robot during the experiment. The horizontal axis represents time. In addition we marked 9 positions (A–I) in Figure 15.

At time A, B, E, G in Figure 15, the robot's reached the current sub-goal and found a new sub-goal. At this time, the robot's linear velocity was zero and turned to the new sub-goal, corresponding to Figure 14a–d. At time H, the final target was found, and we did not set linear velocity to zero, corresponding to Figure 14e. At the time I, the robot reached the final target, corresponding to Figure 14f. At time C, D, F, angular velocity fluctuation is the result of Equation (15). LIDAR points in area D are detected because of LIDAR noise or obstacles in front. Thus angular velocity is adjusted.

Figure 13. Turtlebot2.

Figure 14. The scenario of real experiment.

Figure 15. Linear velocity (red) and the angular velocity (blue) of the robot during the experiment.

5. Discussions

In order to verify our method, many experiments were carried out. In most of the simulation tests, satisfactory results have been derived. However, there are some cases of navigation failure. For example, when the robot and the wall are almost collinear, the LIDAR detected points on the wall are sparse, resulting in wrongly identified gaps. Sensor noise may also cause gaps to be identified incorrectly. On the other hand, there are also cases where some gaps are not recognized. In addition, when it comes to moving obstacles, if the optimal sub-goal changed due to the movement of obstacles, the proposed method has the problem of not being able to update the optimal sub-goal in time. These problems may cause the path to be longer or even the target to be unreachable. Though we cannot guarantee that if a solution exists our method will find it, the experiment achieved satisfactory results. The problems in this article will be further studied in the future.

6. Conclusions

This paper presented a sub-goal seeking approach for local navigation in complex, unknown environments, which can identify gaps and select one of the gaps as a sub-goal to guide the robot to move toward the target. This paper has two main contributions. One is to design a clever gap

recognition method, and the other is finding the common defect of the local algorithms and making improvements. It can avoid the robot from being trapped in complex environments so that the local algorithm can complete complex navigation tasks independently and no longer depends on the global algorithms. Therefore, the proposed approach has low computational complexity, which is of great significance to robots with low computing power. And it will also help to reduce the cost of robots. Finally, simulations and real tests are performed. The results suggest that our approach is robust in controlling the robot and can reach the goal in various environments.

Author Contributions: Conceptualization, K.Y. and B.M.; methodology, K.Y.; software, K.Y.; validation, K.Y. and B.M.; formal analysis, K.Y.; investigation, K.Y.; resources, B.M.; data curation, K.Y.; writing—original draft preparation, K.Y.; writing—review and editing, K.Y.; visualization, K.Y.; supervision, K.Y.; project administration, B.M.; funding acquisition, B.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: This work was supported by National Natural Science Foundation of China (No.61573034,61327807).

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

References

1. Khatib, O. Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous Robot Vehicles*; Springer: Berlin, Germany, 1986; pp. 396–404.
2. Stentz, A. Optimal and efficient path planning for partially known environments. In *Intelligent Unmanned Ground Vehicles*; Springer: Berlin, Germany, 1997; pp. 203–220.
3. LaValle, S.M. Rapidly-eXploring Random Trees: A New Tool for Path Planning. 1998. Available online: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.35.1853&rep=rep1&type=pdf> (accessed on 1 June 2020).
4. Kavraki, L.E.; Svestka, P.; Latombe, J.C.; Overmars, M.H. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Autom.* **1996**, *12*, 566–580. [\[CrossRef\]](#)
5. Adouane, L. Reactive versus cognitive vehicle navigation based on optimal local and global PELC. *Robot. Auton. Syst.* **2017**, *88*, 51–70. [\[CrossRef\]](#)
6. Choi, Y.; Jimenez, H.; Mavris, D.N. Two-layer obstacle collision avoidance with machine learning for more energy-efficient unmanned aircraft trajectories. *Robot. Auton. Syst.* **2017**, *98*, 158–173. [\[CrossRef\]](#)
7. Minguez, J.; Lamiraux, F.; Laumond, J.P. Motion planning and obstacle avoidance. In *Springer Handbook of Robotics*; Springer: Berlin, Germany, 2016; pp. 1177–1202.
8. Sgorbissa, A.; Zaccaria, R. Planning and obstacle avoidance in mobile robotics. *Robot. Auton. Syst.* **2012**, *60*, 628–638. [\[CrossRef\]](#)
9. Borenstein, J.; Koren, Y. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Trans. Robot. Autom.* **1991**, *7*, 278–288. [\[CrossRef\]](#)
10. Ulrich, I.; Borenstein, J. VFH+: Reliable obstacle avoidance for fast mobile robots. In Proceedings of the 1998 IEEE International Conference on Robotics and Automation (Cat. No. 98CH36146), Leuven, Belgium, 20 May 1998; Volume 2, pp. 1572–1577.
11. Fox, D.; Burgard, W.; Thrun, S. The dynamic window approach to collision avoidance. *IEEE Robot. Autom. Mag.* **1997**, *4*, 23–33. [\[CrossRef\]](#)
12. Ng, J.; Bräunl, T. Performance comparison of bug navigation algorithms. *J. Intell. Robot. Syst.* **2007**, *50*, 73–84. [\[CrossRef\]](#)
13. Zhu, Y.; Zhang, T.; Song, J.; Li, X. A new hybrid navigation algorithm for mobile robots in environments with incomplete knowledge. *Knowl.-Based Syst.* **2012**, *27*, 302–313. [\[CrossRef\]](#)
14. Camhi, J.; Johnson, E. High-frequency steering maneuvers mediated by tactile cues: Antennal wall-following in the cockroach. *J. Exp. Biol.* **1999**, *202*, 631–643. [\[PubMed\]](#)
15. Yan, K.; Ma, B. Obstacle Avoidance Based on 2D-Lidar in Unknown Environment. In *Chinese Intelligent Systems Conference*; Springer: Berlin, Germany, 2019; pp. 609–618.
16. MacFarlane, D.; Kim, W.; Nilvi, J.; Nuth, T.; Tan, V. Novel two-dimensional laser scanning technique. *Opt. Lasers Eng.* **1992**, *17*, 1–8. [\[CrossRef\]](#)

17. Ye, C.; Webb, P. A sub goal seeking approach for reactive navigation in complex unknown environments. *Robot. Auton. Syst.* **2009**, *57*, 877–888. [[CrossRef](#)]
18. Sezer, V.; Gokasan, M. A novel obstacle avoidance algorithm: “Follow the Gap Method”. *Robot. Auton. Syst.* **2012**, *60*, 1123–1134. [[CrossRef](#)]
19. Chengqing, L.; Ang, M.H.; Krishnan, H.; Yong, L.S. Virtual obstacle concept for local-minimum-recovery in potential-field based navigation. In Proceedings of the 2000 ICRA, Millennium Conference, IEEE International Conference on Robotics and Automation, Symposia Proceedings (Cat. No. 00CH37065), San Francisco, CA, USA, 24–28 April 2000; Volume 2, pp. 983–988.
20. Xu, W.; Tso, S.; Lu, Z. A virtual target approach for resolving the limit cycle problem in navigation of a fuzzy behaviour-based mobile robot. In Proceedings of the 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems, Innovations in Theory, Practice and Applications (Cat. No. 98CH36190), Victoria, BC, Canada, 17 October 1998; Volume 1, pp. 44–49.
21. Harik, E.H.C.; Audun, K. The Heading Weight Function: A Novel LiDAR-Based Local Planner for Nonholonomic Mobile Robots. *Sensors* **2019**, *19*, 3606. [[CrossRef](#)] [[PubMed](#)]
22. Shim, Y.; Kim, G.W. Range Sensor-Based Efficient Obstacle Avoidance through Selective Decision-Making. *Sensors* **2018**, *18*, 1030. [[CrossRef](#)] [[PubMed](#)]
23. Llamazares, A.; Ivan, V.; Molinos, E.; Ocaña, M.; Vijayakumar, S. Dynamic Obstacle Avoidance Using Bayesian Occupancy Filter and Approximate Inference. *Sensors* **2013**, *13*, 2929–2944. [[CrossRef](#)] [[PubMed](#)]
24. Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; Ng, A.Y. ROS: An open-source Robot Operating System. Available online: <http://www.willowgarage.com/sites/default/files/icraoss09-ROS.pdf> (accessed on 2 June 2020).
25. Minguez, J.; Montano, L. Nearness diagram (ND) navigation: Collision avoidance in troublesome scenarios. *IEEE Trans. Robot. Autom.* **2004**, *20*, 45–59. [[CrossRef](#)]

Publisher’s Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).